

Table of Contents

Introduction	1.1
ConfigExamples	1.2
Git	1.3
Github	1.3.1
git-branch	1.3.2
git	1.3.3
githttps	1.3.4
Guides	1.4
Self-Cultivation-of-Programmers	1.4.1
Learn	1.5
GPU2CPU	1.5.1
LearnAIMA	1.5.2
Ch12	1.5.2.1
Ch13	1.5.2.2
BayesianNetwork	1.5.2.2.1
Ch18	1.5.2.3
index	1.5.2.4
LearnCUDA	1.5.3
APT-CUDA	1.5.3.1
CUDA	1.5.3.2
CUDA_multi	1.5.3.3
NCCL	1.5.3.4
TensorCores	1.5.3.5
LearnCV	1.5.4
ComputerVisoin	1.5.4.1
OpenCV	1.5.4.2
LearnConda	1.5.5
ChangeBaseEnv	1.5.5.1
Conda	1.5.5.2
LearnDocker	1.5.6
docker	1.5.6.1
LearnDrones	1.5.7
Ceres-Solver	1.5.7.1
Eigen	1.5.7.2
FastDrone-250	1.5.7.3
Note	1.5.7.4

Planner	1.5.7.5
Distributed-Swarm-Trajectory-Opt-for-Formation-Flight-in-Dense-Envs-2022	1.5.7.5.1
EGO-Swarm-2021	1.5.7.5.2
FastPlanner-2019	1.5.7.5.3
Gradient-Based-Motion-Planning	1.5.7.5.4
MicroFlyingRobots-2021	1.5.7.5.5
Robust-Efficient-Trajectory-Planning-for-Formation-Flight-in-Dense-Environments-2023	
TaskAssignment	1.5.7.6
2022.06-	
Task_assignment_algorithms_for_unmanned_aerial_vehicle_networks_A_comprehensive_survey	1.5.7.6.1
2024-	
Distributed_task_allocation_algorithm_for_heterogeneous_unmanned_aerial_vehic_swarm_based_on_coalitionFormation_game	1.5.7.6.2
2024.12-Application_of_Task_Allocation_Algorithms_in_Multi-UAV_Intelligent_Transportation_Systems_A_Critical_Review	1.5.7.6.3
index	1.5.7.6.4
LearnGPU2CPU-by-GPT	1.5.8
Note	1.5.8.1
LearnGameDev	1.5.9
Note	1.5.9.1
LearnGameTheory	1.5.10
CoalitionalGames	1.5.10.1
GameTheory	1.5.10.2
LearnJinJia2	1.5.11
Jinjia2	1.5.11.1
LearnLalrpop	1.5.12
Lalrpop	1.5.12.1
LearnMLIR	1.5.13
CodegenDialectOverview	1.5.13.1
Dialects	1.5.13.2
Linalg	1.5.13.3
MLIR-LanRef	1.5.13.4
PIR	1.5.13.5
LearnMocCUDA	1.5.14
CUDA-Wrap	1.5.14.1
LibBlocksRuntime	1.5.14.2
LibDispatch	1.5.14.3
LibUnwind	1.5.14.4

MakefileNote	1.5.14.5
MocCUDA	1.5.14.6
OpenBLAS	1.5.14.7
OpenMPI	1.5.14.8
Spack	1.5.14.9
pytorch.needed.libs.Note	1.5.14.10
LearnMordenCpp	1.5.15
Note	1.5.15.1
LearnPest	1.5.16
ParsingExpressionGrammar	1.5.16.1
pest	1.5.16.2
LearnPolygeist	1.5.17
2021_Polygeist-Raising C to Polyhedral MLIR-translated_only	1.5.17.1
2021_Polygeist-Raising C to Polyhedral MLIRtranslated_and_original	1.5.17.2
Note	1.5.17.3
LearnPolyhedralModel	1.5.18
AffineTransformation	1.5.18.1
Note	1.5.18.2
LearnROCM	1.5.19
ROCM	1.5.19.1
LearnROS	1.5.20
ROS-Packages	1.5.20.1
ROS	1.5.20.2
ROS2	1.5.20.3
ROS2_debian12	1.5.20.4
SourceInstall	1.5.20.5
LearnRust	1.5.21
LifetimeExample	1.5.21.1
Problems_Solutions	1.5.21.2
Resources	1.5.21.3
RustBook	1.5.21.4
RustLanRef	1.5.21.5
RustProgrammingLanguage	1.5.21.6
PersonalWebsite	1.5.22
Jekyll	1.5.22.1
Note	1.5.22.2
Ruby	1.5.22.3
al-folio	1.5.22.4

Linux	1.6
Display	1.6.1
GNOME-Default	1.6.1.1
Hyprland	1.6.1.2
Wayland	1.6.1.3
Flashback	1.6.2
InputMethod	1.6.3
LSB	1.6.4
LinuxFamily	1.6.5
ArchLinux	1.6.5.1
DebianFamily	1.6.5.2
GentooLinux	1.6.5.3
RedHatFamily	1.6.5.4
Nautilus	1.6.6
PrintingDrivers	1.6.7
UserManagement	1.6.8
apt	1.6.9
curl	1.6.10
Misc	1.7
00_Links	1.7.1
01_GitHubReps	1.7.2
Bochs	1.7.3
Compression	1.7.4
ComputationForcePlatforms	1.7.5
Copy	1.7.6
Drawing	1.7.7
Flatpack	1.7.8
Fonts	1.7.9
FuzzTesting	1.7.10
AFL.RS	1.7.10.1
Issues_Solutions	1.7.10.2
GIS	1.7.11
OpenShp	1.7.11.1
GRUB	1.7.12
GitBook	1.7.13
Hyper-V	1.7.14
LLVM	1.7.15
Logics	1.7.16

Markdown	1.7.17
MemoryOrder	1.7.18
NPM	1.7.19
PowerShell	1.7.20
RegularExp	1.7.21
StorageTechs	1.7.22
TWRP	1.7.23
TimeSeriesPrediction	1.7.24
ACF	1.7.24.1
ACF_vs_PACF	1.7.24.2
ARIMA	1.7.24.3
PACF	1.7.24.4
Typst	1.7.25
VScode	1.7.26
Vitural-Card	1.7.27
WSL	1.7.28
Network	1.8
ChangeMac	1.8.1
DNS	1.8.2
RandomMac	1.8.3
Tailscale	1.8.4
Wifi	1.8.5
Papers	1.9
AttentionIsTuringComplete	1.9.1
Spatially varying nanophotonic neural networks	1.9.2
Programming	1.10
CRTP	1.10.1
PythonLibs	1.11
ArgParse	1.11.1
AsyncSSH	1.11.2
Dask	1.11.3
Dlib	1.11.4
DlibPythonBindings	1.11.4.1
LearnPyQt5	1.11.5
PyQt5	1.11.5.1
QtStyleSheets	1.11.5.2
ReadingNotes	1.12
A_Brief_History_of_Humankind	1.12.1

Feynman	1.12.2
math_philosophy	1.12.3
philosophy	1.12.4
研究与学风-张申府-1948	1.12.5
计算之魂-吴军	1.12.6
cp0	1.12.6.1
cp1	1.12.6.2
cp2	1.12.6.3
index	1.12.6.4
论青年修养-张申府-1946	1.12.7
论青年的修养-张闻天-1938	1.12.8
语录-毛泽东	1.12.9
青年在选择职业时的考虑-马克思-1835	1.12.10

My Notes for nearly Everything, by hhw

- `Learn` : Record learning resources, environment configuration, common questions and precautions when learning something or a certain skill
- `Linux` : Linux related notes, including Linux distribution, display, user management, input method, etc.
- `PythonLibs` -
- `Programming` : Programming related notes, about design patterns and so on.
- `ReadingNotes` : Personal reading notes, including books and articles.
- `Network` : Network related notes, including network protocols, network architecture, etc.
- `Papers` : Papers I read and want to remember.
- `ConfigExamples` : Configuration examples for different tools and editors.
- `Git` : Git related notes, including basic usage, branch management, etc.
- `Misc` : Miscellaneous notes, uncategorized notes.

★ If you think this repository is helpful to you, please give me a star! 😊

Configuration Examples

- Configuration Examples

- [.clang-format](#)
- [.gitignore](#)

.clang-format

```
---
```

```
Language:          Cpp
AccessModifierOffset: -2
AlignAfterOpenBracket: Align
AlignArrayOfStructures: None
AlignConsecutiveMacros: None
AlignConsecutiveAssignments: None
AlignConsecutiveBitFields: None
AlignConsecutiveDeclarations: None
AlignEscapedNewlines: Left
AlignOperands: Align
AlignTrailingComments: true
...
```

.gitignore

```
**/.**
**/__pycache__
**/data
**/__pycache__
**/*.*.model
*.zip
*.pyc
*.pyo
```

Github

- [Displaying Contributors Image on README files with no Pain!](#)
- [contributors-img: contrib.rocks](#)
- [contributors-img](#)
- [all-contributors](#)
- [Best-README-Template](#)
- [GitHub Actions for GitHub Pages](#)

To set a default upstream branch for `git push` and `git pull`, you need to configure the branch to track a specific remote branch. Here's how to do it:

Set Default Upstream Branch

- 1. Using `git push` Command** The easiest way to set an upstream branch is to use the `--set-upstream` option with `git push`:

```
git push --set-upstream origin <branch-name>
```

This command does two things:

- Pushes your branch to the remote repository.
- Configures the branch to track the corresponding branch on the remote.

- 2. Using `git branch` Command** You can manually set the upstream branch for the current branch:

```
git branch --set-upstream-to=origin/<remote-branch> <local-branch>
```

Example:

```
git branch --set-upstream-to=origin/main main
```

- 3. While Creating a New Branch** If you create a new branch and want it to track a remote branch from the start:

```
git checkout -b <branch-name> --track origin/<remote-branch>
```

Example:

```
git checkout -b feature/new-feature --track origin/feature/new-feature
```

Verify Upstream Branch

To confirm the upstream branch configuration:

```
git branch -vv
```

- The output will display the upstream branch associated with each local branch.

Remove or Change Upstream Branch

- 1. To Remove an Upstream Branch** If you no longer want a branch to track a remote branch:

```
git branch --unset-upstream
```

2. **To Change the Upstream Branch** Use the `--set-upstream-to` option to point to a different remote branch:

```
git branch --set-upstream-to=origin/new-branch
```

How it Works for `git push` and `git pull`

- Once the upstream branch is set, `git push` and `git pull` will automatically use the configured upstream branch.
- Example workflow:

1. Set upstream:

```
git push --set-upstream origin main
```

2. Push and pull without specifying branches:

```
git push  
git pull
```

Let me know if you need further assistance!

Git

```
git remote set-url --add origin <git_url>

# You have divergent branches and need to specify how to reconcile them.
# You can do so by running one of the following commands sometime before
# your next pull:

git config pull.rebase false  # merge
git config pull.rebase true   # rebase
git config pull.ff only      # fast-forward only
```

in ubuntu, i use https git repository, each time i pull or push need to type in account and password, how to avoid this? (cant use ssh)

To avoid typing your username and password each time you push or pull from a Git repository over HTTPS on Ubuntu, you can use Git's credential helper to cache your credentials. Here's how you can set it up:

1. Use the Credential Cache Helper

The credential cache helper keeps your credentials in memory for a short period (default is 15 minutes).

Run the following command to enable it:

```
git config --global credential.helper cache
```

If you want to change the cache timeout, you can set it in seconds. For example, to cache credentials for 1 hour (3600 seconds):

```
git config --global credential.helper 'cache --timeout=3600'
```

2. Use the Credential Store Helper

The credential store helper saves your credentials in a plain text file on disk, which is more persistent but less secure than the cache method.

To enable the credential store helper:

```
git config --global credential.helper store
```

When you use `git pull` or `git push` for the first time after configuring this, Git will prompt you for your username and password, and then store them in a file at `~/.git-credentials`.

3. Use the libsecret Credential Helper

The `libsecret` credential helper integrates with GNOME Keyring to securely store your credentials.

First, install the required package:

```
sudo apt-get install libsecret-1-0 libsecret-1-dev
```

Then, you need to compile the `libsecret` credential helper. This is a one-time setup:

```
cd /usr/share/doc/git/contrib/credential/libsecret
sudo make
```

Finally, configure Git to use the `libsecret` helper:

```
git config --global credential.helper /usr/share/doc/git/contrib/credential/libsecret/git-credential
```

4. Store Credentials for a Single Repository

If you want to store credentials for just one repository and not globally, navigate to your repository and run:

```
git config credential.helper store
```

Steps to Verify

After configuring the credential helper, try pulling or pushing to your repository:

```
git pull
```

You should be prompted for your credentials the first time, and then they will be cached or stored based on your chosen method.

Security Note

Storing credentials in plain text (using the store helper) can be insecure, as anyone with access to your file system can read them. Use the cache helper or `libsecret` for better security.

Self Cultivation of Programmers

- reference: [程序员的自我修养](#)

写给年轻程序员的建议

1. 比“一般人”更努力
2. 适时建立个人权威
3. 保持好奇心
4. 抛开代码与人沟通，结识更优秀的人 or 通过代码/参与项目结识其他开发者
5. 为优秀的人工作或努力
6. 协调生活与工作，enjoy your life
7. 相信自己的天赋和创造力
8. 尽可能持续地、长久地做一件有意义的事，把它做好做精
9. 保持严谨、维持声誉
10. 勤于思考，不轻易说简单/不可能
11. 找到自己所热爱的，对人对事保持真诚和热情
12. 锻炼表达能力，敢说多说，创造机会、把握机会多讲多分享
13. 深入地思考，不要浅尝辄止、皮毛肤浅
14. 善于总结归纳，从纷繁的表面现象中总结抽象发现问题的本质、发现关键的问题
15. 纵横捭阖、合纵连横 将不同领域不同角度的东西相贯通、发现相似之处、共通质处，并从迁移比较中获得启发
16. 对新鲜技术保持敏锐，学习新技术、新观念并将其应用到自己的工作中
17. 平衡取舍，在有限的时间有限的资源有限的能力内解决问题
18. 快速迭代更新
19. 从失败中学习，积累经验，总结方法
20. 对自己、对项目、对代码、对事情有着高的自我要求
21. 不要图省事而做一些愚蠢的事情

如何找到靠谱的程序员

- 简历：重点项目经验；与编程有关的社会化活动
- 自我介绍，介绍自己最擅长的、最有激情的
 - 语言表达能力、逻辑思维能力、专业能力
 - 是否有清晰的自我认知和自我肯定
 - 是否有自己所热爱的，是否葆有激情与热忱
- 基础打牢了吗？
 - 计算机原理？计算机网络基本原理...
- 对自己专精的技术是否足够深入？

Learn Notes

GPU to CPU

MocCUDA

MocCUDA

High-Performance GPU-to-CPU Transpilation and Optimization via High-Level Parallel Constructs 通过高级并行结构实现高性能 GPU 到 CPU 的转译和优化

William S. Moses, Ivan R. Ivanov, Jens Domke, Toshio Endo, Johannes Doerfert, Oleksandr Zinenko

PPoPP '23: Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming PPoPP '23: 第 28 届 ACM SIGPLAN 并行编程原理与实践年度研讨会论文集

While parallelism remains the main source of performance, architectural implementations and programming models change with each new hardware generation, often leading to costly application re-engineering. Most tools for performance portability require manual and costly application porting to yet another programming model.

虽然并行性仍然是性能的主要来源，但架构实现和编程模型会随着每一代新硬件的变化而变化，这通常会导致昂贵的应用程序重新设计。大多数性能可移植性工具都需要手动且昂贵的应用程序移植到另一个编程模型。

We propose an alternative approach that automatically translates programs written in one programming model (CUDA), into another (CPU threads) based on Polygeist/MLIR. Our approach includes a representation of parallel constructs that allows conventional compiler transformations to apply transparently and without modification and enables parallelism-specific optimizations. We evaluate our framework by transpiling and optimizing the CUDA Rodinia benchmark suite for a multi-core CPU and achieve a 76% geomean speedup over handwritten OpenMP code. Further, we show how CUDA kernels from PyTorch can efficiently run and scale on the CPU-only Supercomputer Fugaku without user intervention. Our PyTorch compatibility layer making use of transpiled CUDA PyTorch kernels outperforms the PyTorch CPU native backend by $2.7\times$.

我们提出了一种替代方法，将用一种编程模型（CUDA）编写的程序自动转换为基于 Polygeist/MLIR 的另一个编程模型（CPU 线程）。我们的方法包括并行结构的表示，它允许传统的编译器转换透明且无需修改地应用，并支持特定于并行性的优化。我们通过为多核 CPU 转译和优化 CUDA Rodinia 基准测试套件来评估我们的框架，并与手写的 OpenMP 代码相比实现了 76% 的几何平均速度提升。此外，我们还展示了 PyTorch 的 CUDA 内核如何在纯 CPU 超级计算机 Fugaku 上高效运行和扩展，而无需用户干预。我们的 PyTorch 兼容层使用转译的 CUDA PyTorch 内核，其性能比 PyTorch CPU 本机后端高出 $2.7\times$ 倍。

We propose a compiler model for most common GPU constructs: multi-level parallelism, level-wide synchronization, and level-local memory. This differs from CPU parallelism, which provides a single level of parallelism, a unified memory and peer synchronization. In contrast to source and AST-level approaches, which operate before the optimization pipeline, and existing compiler approaches, which model synchronization as a “black-box” optimization barrier, we model synchronization entirely from memory semantics. This both allows synchronization-based code to inter-operate with existing optimizations and enables novel parallel-specific optimizations. 我们为最常见的 GPU 结构提出了一个编译器模型：多级并行、级别范围同步和级别本地内存。这与 CPU 并行不同，后者提供单级并行、统一内存和对等同步。与在优化管道之前运行的源和 AST 级方法以及将同步建模为“黑盒”优化屏障的现有编译器方法相比，我们完全根据内存语义对同步进行建模。这既允许基于同步的代码与现有优化进行互操作，又可以实现新颖的并行特定优化。

Our model is implemented in the MLIR layer [20] of the LLVM compiler infrastructure [21] and it leverages MLIR's nested-module approach for GPU codes [22]. We extended the Polygeist [23] C/C++ frontend to support CUDA and to produce MLIR which preserves high-level parallelism and program structure. Our prototype compiler is capable of compiling PyTorch CUDA kernels, as well as other compute-intensive benchmarks, to any CPU architecture supported by LLVM. In addition to transformations accounting for the differences in the execution model, we also exploit parallelism on the CPU via OpenMP. Finally, our MocCUDA PyTorch integration uses our approach to compile and execute CUDA kernels in absence of a GPU while substituting unsupported calls. 我们的模型在 LLVM 编译器基础设施[21]的 MLIR 层[20]中实现，它利用 MLIR 的 GPU 代码嵌套模块方法[22]我们扩展了 Polygeist [23] C/C++ 前端以支持 CUDA 并生成MLIR 保留了高级并行性和程序结构。我们的原型编译器能够将 PyTorch CUDA 内核以及其他计算密集型基准编译到 LLVM 支持的任何 CPU 架构。除了考虑执行模型差异的转换之外，我们还通过 OpenMP 利用 CPU 上的并行性。最后，我们的 MocCUDA PyTorch 集成使用我们的方法在没有 GPU 的情况下编译和执行 CUDA 内核，同时替换不支持的调用。

Overall, our paper makes the following contributions: 总的来说，我们的论文做出了以下贡献:

- A common high-level and platform-agnostic representation of SIMT-style parallelism backed by a semantic definition of barrier synchronization that ensures correctness through memory semantics, which ensures transparent application of existing optimizations.
- SIMT 式并行的通用高级且与平台无关的表示形式，由屏障同步的语义定义支持，通过内存语义确保正确性，从而确保现有优化的透明应用。
- Novel parallel-specific optimizations which can exploit our high-level parallel semantics to optimize programs.
- 新颖的并行特定优化可以利用我们的高级并行语义来优化程序。
- An extension to the Polygeist C/C++ frontend for MLIR which is capable of directly mapping GPU and CPU parallel constructs into our high-level parallelism primitives.
- MLIR 的 Polygeist C/C++ 前端的扩展，能够直接将 GPU 和 CPU 并行结构映射到我们的高级并行原语中。
- An end-to-end transpilation of CUDA to CPU for a subset of the Rodinia [24] benchmark suite and the internal CUDA kernels within PyTorch [2] necessary to run a Resnet-50 on the CPU-only Fugaku supercomputer.
- 将 CUDA 端到端转译为 CPU，用于 Rodinia [24]基准测试套件的子集以及 PyTorch [2]中的内部 CUDA 内核，这是在纯 CPU Fugaku 超级计算机上运行 Resnet-50 所必需的。

II-CPolygeist II-C多灵

Polygeist is a C and C++ frontend for MLIR based on Clang [23]. It is capable of translating a broad range of C++ programs into a mix of MLIR dialects that preserve elements of the high-level structure of the program. In particular, Polygeist preserves structured control flow (loops and conditionals) as MLIR SCF dialect. It also simplifies analyses by preserving multi-dimensional array constructs whenever possible by relying on the MLIR's multi-dimensional memory reference (memref) type. Finally, Polygeist is able to identify parts of the program suitable for polyhedral optimization [28] and represent them using the Affine dialect. Polygeist 是基于 Clang [23] 的MLIR 的 C 和 C++ 前端。它能够将各种 C++ 程序转换为 MLIR 方言的混合，从而保留程序高级结构的元素。特别是，Polygeist 将结构化控制流（循环和条件）保留为 MLIR SCF 方言。它还通过依赖 MLIR 的多维内存引用 (memref) 类型尽可能保留多维数组结构来简化分析。最后，Polygeist 能够识别程序中适合多面体优化的部分[28]并使用仿射方言表示它们。

Representation of GPU Kernel Launch

We define the representation of a GPU kernel launch as follows (illustrated in Fig. 3):

- A 3D parallel for-loop over all blocks in the grid.
- A stack allocation for any shared memory, scoped to be unique per block.
- A 3D parallel for-loop over all threads in a block.
- A custom Polygeist barrier operation that provides equivalent semantics to a CUDA/ROCm synchronization.

我们定义GPU内核启动的表示如下(如图3所示):

- 在网格中所有块上的3D并行for循环。
- 对任何共享内存的堆栈分配,每个块的范围是唯一的。
- 在一个块中所有线程上的3D并行for循环。
- 提供与 CUDA / ROCm同步等效语义的自定义 Polygeist 屏障操作。

[PolygeistGPU-Docker](#)

From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming

<https://www.sciencedirect.com/science/article/abs/pii/S0167819111001335>

In this work, we evaluate OpenCL as a programming tool for developing performance-portable applications for GPGPU. While the Khronos group developed OpenCL with programming portability in mind, performance is not necessarily portable. OpenCL has required performance-impacting initializations that do not exist in other languages such as CUDA. Understanding these implications allows us to provide a single library with decent performance on a variety of platforms. We choose triangular solver (TRSM) and matrix multiplication (GEMM) as representative level 3 BLAS routines to implement in OpenCL. We profile TRSM to get the time distribution of the OpenCL runtime system. We then provide tuned GEMM kernels for both the NVIDIA Tesla C2050 and ATI Radeon 5870, the latest GPUs offered by both companies. We explore the benefits of using the texture cache, the performance ramifications of copying data into images, discrepancies in the OpenCL and CUDA compilers' optimizations, and other issues that affect the performance. Experimental results show that nearly 50% of peak performance can be obtained in GEMM on both GPUs in OpenCL. We also show that the performance of these kernels is not highly portable. Finally, we propose the use of auto-tuning to better explore these kernels' parameter space using search harness.

在这项工作中，我们将 OpenCL 评估为用于为 GPGPU 开发性能可移植应用程序的编程工具。虽然 Khronos 小组在开发 OpenCL 时考虑到了编程的可移植性，但性能不一定是可移植的。OpenCL 需要影响性能的初始化，而其他语言（如 CUDA）中不存在这些初始化。了解这些含义使我们能够在各种平台上提供具有良好性能的单个库。我们选择三角求解器（TRSM）和矩阵乘法（GEMM）作为在 OpenCL 中实现的代表性 3 级 BLAS 例程。我们分析 TRSM 以获得 OpenCL 运行时系统的时间分布。然后，我们为 NVIDIA Tesla C2050 和 ATI Radeon 5870（两家公司提供的最新 GPU）提供经过调整的 GEMM 内核。我们探讨了使用纹理缓存的好处、将数据复制到图像中的性能影响、OpenCL 和 CUDA 编译器优化中的差异以及影响性能的其他问题。实验结果表明，在 OpenCL 的两个 GPU 上，GEMM 都可以获得近 50% 的峰值性能。我们还表明，这些内核的性能并不是高度可移植的。最后，我们建议使用自动调整，以使用户搜索工具更好地探索这些内核的参数空间。

Kokkos: Enabling manycore performance portability through polymorphic memory access patterns

Kokkos: 通过多态内存访问模式实现实众核性能可移植性

Achieving Portability and Performance through OpenACC

通过 OpenACC 实现可移植性和性能

<https://ieeexplore.ieee.org/document/7081674>

OpenACC is a directive-based programming model designed to allow easy access to emerging advanced architecture systems for existing production codes based on Fortran, C and C++. It also provides an approach to coding contemporary technologies without the need to learn complex vendor-specific languages, or understand the hardware at the deepest level. Portability and performance are the key features of this programming model, which are essential to productivity in real scientific applications. OpenACC support is provided by a number of vendors and is defined by an open standard. However the standard is relatively new, and the implementations are relatively immature. This paper experimentally evaluates the currently available compilers by assessing two approaches to the OpenACC programming model: the "parallel" and "kernels" constructs. The implementation of both of these construct is compared, for each vendor, showing performance differences of up to 84%.

Additionally, we observe performance differences of up to 13% between the best vendor implementations.

OpenACC features which appear to cause performance issues in certain compilers are identified and linked to differing default vector length clauses between vendors. These studies are carried out over a range of hardware including GPU, APU, Xeon and Xeon Phi based architectures. Finally, OpenACC performance, and productivity, are compared against the alternative native programming approaches on each targeted platform, including CUDA, OpenCL, OpenMP 4.0 and Intel Offload, in addition to MPI and OpenMP.

OpenACC 是一种基于指令的编程模型，旨在允许轻松访问基于 Fortran、C 和 C++ 的现有产品代码的新高级架构系统。它还提供了一种对现代技术进行编码的方法，而无需学习复杂的供应商特定语言，也无需更深入地了解硬件。可移植性和性能是该编程模型的关键特性，这对于实际科学应用中的生产力至关重要。OpenACC 支持由许多供应商提供，并由开放标准定义。但是，该标准相对较新，实现也相对不成熟。本文通过评估 OpenACC 编程模型的两种方法（“并行”和“内核”结构）来实验评估当前可用的编译器。对每个供应商的这两种结构的实施进行了比较，显示性能差异高达 84%。此外，我们观察到最佳供应商实施之间的性能差异高达 13%。识别出在某些编译器中似乎会导致性能问题的 OpenACC 功能，并将其链接到供应商之间不同的默认向量长度子句。这些研究是在一系列硬件上进行的，包括基于 GPU、APU、Xeon 和 Xeon Phi 的架构。最后，将 OpenACC 的性能和生产力与每个目标平台上的替代原生编程方法进行了比较，包括 CUDA、OpenCL、OpenMP 4.0 和 Intel Offload，以及 MPI 和 OpenMP。

GPU(CUDA) to CPU

GPUOcelot

[GPUOcelot](#)

GPUOcelot: A dynamic compilation framework for PTX

GPUOcelot: PTX 的动态编译框架

Ocelot is a just-in-time compiler, which retargets PTX assembler (used internally by CUDA) for non-NVIDIA hardware.

Ocelot是一个即时编译器，它针对非 NVIDIA 硬件重新定位 PTX 汇编器（由 CUDA 内部使用）。

HIPIFY

[HIPIFY: Convert CUDA to Portable C++ Code](#)

hipify-clang and hipify-perl are tools that automatically translate NVIDIA CUDA source code into portable HIP C++.

hipify-clang和hipify-perl是自动将 NVIDIA CUDA 源代码转换为可移植 HIP C++ 的工具。

HIP

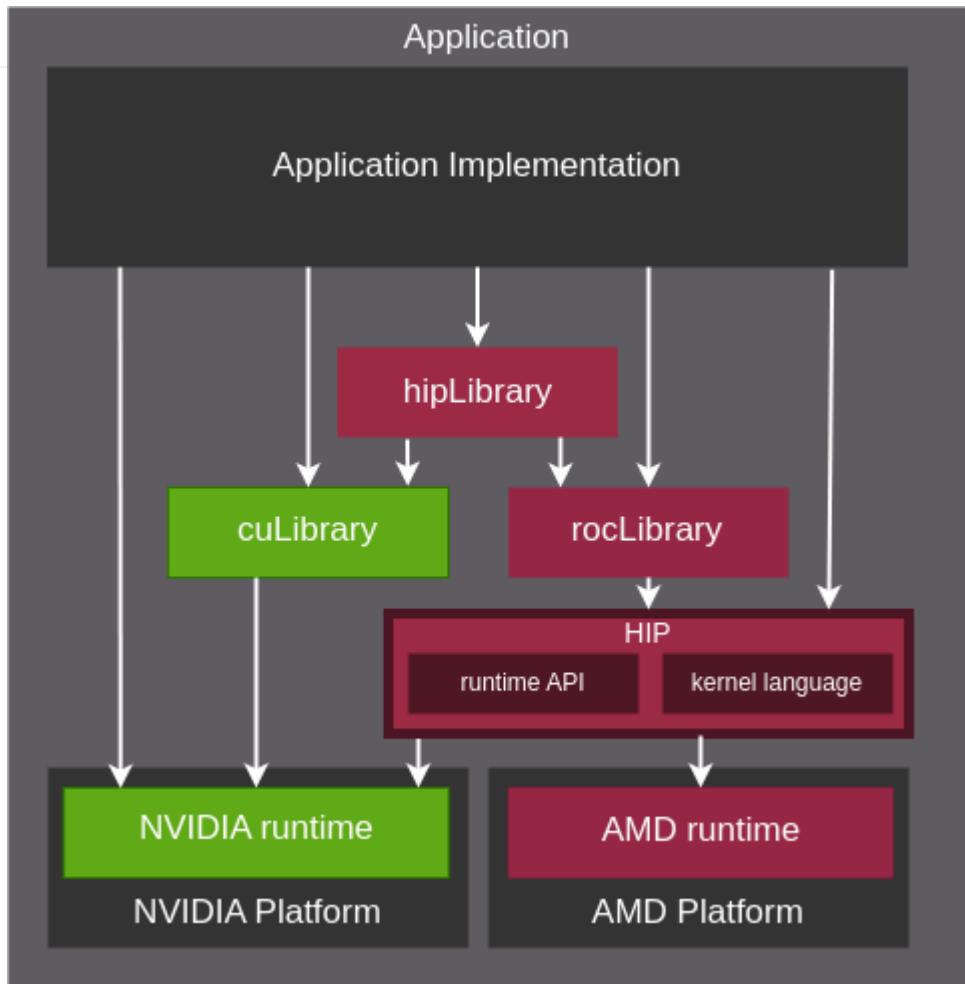
[HIP: C++ Heterogeneous-Compute Interface for Portability](#)

HIP is a C++ Runtime API and Kernel Language that allows developers to create portable applications for AMD and NVIDIA GPUs from single source code.

HIP 是一种 C++ 运行时 API 和内核语言，允许开发人员从单一源代码创建适用于 AMD 和 NVIDIA GPU 的可移植应用程序。

Key features include: 主要特点包括:

- HIP is very thin and has little or no performance impact over coding directly in CUDA mode.
HIP 非常薄，与直接在 CUDA 模式下编码相比，几乎没有性能影响。
- HIP allows coding in a single-source C++ programming language including features such as templates, C++11 lambdas, classes, namespaces, and more.
HIP 允许使用单源 C++ 编程语言进行编码，包括模板、C++11 lambda、类、命名空间等功能。
- HIP allows developers to use the "best" development environment and tools on each target platform.
HIP 允许开发人员在每个目标平台上使用“最好的”开发环境和工具。
- The [HIPIFY](#) tools automatically convert source from CUDA to HIP.
[HIPIFY](#) 工具自动将源从 CUDA 转换为 HIP。
- Developers can specialize for the platform (CUDA or AMD) to tune for performance or handle tricky cases.
开发人员可以专门针对平台（CUDA 或 AMD）来调整性能或处理棘手的情况。



Domain-Specific Multi-Level IR Rewriting for GPU: The Open Earth Compiler for GPU-accelerated Climate Simulation

领域特定多级 IR 重写用于 GPU: GPU 加速气候模拟的开放地球编译器

<https://dl.acm.org/doi/10.1145/3469030>

2021.9.03

Most compilers have a single core intermediate representation (IR) (e.g., LLVM) sometimes complemented with vaguely defined IR-like data structures. This IR is commonly low-level and close to machine instructions. As a result, optimizations relying on domain-specific information are either not possible or require complex analysis to recover the missing information. In contrast, multi-level rewriting instantiates a hierarchy of dialects (IRs), lowers programs level-by-level, and performs code transformations at the most suitable level. We demonstrate the effectiveness of this approach for the weather and climate domain. In particular, we develop a prototype compiler and design stencil- and GPU-specific dialects based on a set of newly introduced design principles. We find that two domain-specific optimizations (500 lines of code) realized on top of LLVM's extensible MLIR compiler infrastructure suffice to outperform state-of-the-art solutions. In essence, multi-level rewriting promises to herald the age of specialized compilers composed from domain- and target-specific dialects implemented on top of a shared infrastructure.

大多数编译器都有一个单一核心的中间表示（IR）（例如，LLVM），有时辅以模糊定义的类似 IR 的数据结构。这个 IR 通常是低级且接近机器指令的。因此，依赖于特定领域信息的优化要么不可能实现，要么需要复杂的分析来恢复缺失的信息。相比之下，多级重写实例化了一组方言（IR）的层次结构，逐级

降低程序级别，并在最合适的级别执行代码转换。我们展示了这种方法在天气和气候领域中的有效性。特别是，我们开发了一个原型编译器，并基于一组新引入的设计原则设计了模板和 GPU 特定的方言。我们发现，在 LLVM 的可扩展 MLIR 编译器基础设施之上实现的两个特定领域优化（500 行代码）足以超越最先进的解决方案。本质上，多级重写有望预示着由领域和目标特定方言组成的专用编译器的时代，这些方言是在共享基础设施之上实现的。

第 12 章 不确定性的量化

本章表明概率论是不确定推理的合适基础，并简要介绍了它的应用。

- 不确定性的产生是由于惰性和无知。在复杂的、非确定性的或部分可观测的环境中，不确定性是不可避免的。
- 概率表达了智能体无法对一个语句的真值做出明确的判断。概率概括了智能体对于证据的信念。
- **决策论**结合了智能体的**信念和欲望**，将**最大期望效用**的动作定义为最佳动作。
 - 决策论 = 概率论 + 效用理论，MEU
- 基本的概率陈述包括简单命题和复杂命题上的先验概率（或无条件概率）和后验概率（或条件概率）。
- **概率公理**约束逻辑相关命题的概率。违背公理的智能体在某些情况下的行为必定是不理性的。
- **完全联合概率分布**为随机变量的每种完整赋值指定了概率。通常，完全联合概率分布过于庞大，以至于无法显式地创建和使用，但如果其可用时，它可以用于回答查询，只需要简单地将其中与查询命题对应的可能世界的条目相加即可。
- 随机变量子集间的**绝对独立性**允许将完全联合分布分解成小的联合分布，极大地降低它的复杂度。
- **贝叶斯法则**允许通过已知的条件概率去计算未知概率，条件概率通常在因果方向上。将贝叶斯法则应用于多条证据时会遇到与完全联合分布相同的规模扩展问题。
- 域中的直接因果关系带来的条件独立性允许完全联合分布被分解成小的条件分布。**朴素贝叶斯模型**假设给定单原因变量时，所有结果变量具有条件独立性。模型大小随结果个数线性增长。
- wumpus 世界的智能体可以计算世界中未观测的方面的概率，从而改进纯逻辑智能体的决策。条件独立性简化了这些计算。

- ([33 封私信 / 81 条消息](#)) 相互独立和条件独立的关系？ - 知乎 ([zhihu.com](#))

概率定义

概率的公理化定义

概率是一个事件集合上的度量，满足以下 3 条公理。

1. 每个事件的度量在 0 和 1 之间，可写为 $0 \leq P(\{X=x_i\}) \leq 1$ ，其中 X 是表示事件的随机变量， x_i 是 X 的可能的值。一般来说，随机变量用大写字母表示，它们的值用小写字母表示。
2. 整个集合的度量为 1，即

$$\sum_{i=1}^n P(X = x_i) = 1$$

1. 不相交事件的并集的概率等于单个事件概率的和，即

$$P(X = x_1 \vee X = x_2) = P(X = x_1) + P(X = x_2)$$

这里 x_1 和 x_2 是不相交的。

容斥原理：

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$

什么叫“概率公理的论据”？

为什么不违背“概率公理”的信念集合呢？

因为违背概率公理的信念集合是不合理的，设想一个赌局，参加者按照他持有的信念状态下注，下注数量与自己持有的信念状态成正比。可以证明，对于按照违反概率公理的信念状态进行下注的智能体，一定存在某种赌局的组合，使得该智能体每次都会输钱。？？？

完全联合概率分布：所有随机变量的联合分布，即给定所有变量的所有取值组合的概率密度/概率

一个概率模型可以由完全联合概率分布完全确定

而每个命题的概率是该命题为真的可能世界的概率的求和，那么，如果已知完全联合概率分布，我们原则上可以计算任何命题的概率（因为我们已经指定/给出了每一个可能的样本点的概率，要做的只是找出满足要求的样本点，然后求和）。

实际上呢？

1. 完全联合概率分布实际上不可知/不知道
2. 完全概率分布的求和计算复杂度难以承受

使用完全联合概率分布进行推断

概率推断：使用完全联合概率分布作为知识库，给定观测证据，为每个查询命题计算后验概率。

边缘概率：抽取变量子集 or 单个变量的分布

边缘化/求和消元：对其他/不关注的变量在其值域上求和

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}, \mathbf{Z} = \mathbf{z})$$

条件化：

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y} | \mathbf{z}) P(\mathbf{z})$$

通用推断过程：

- X 待查询单变量
- E 证据变量列表，e 为观测值列表
- Y 剩余为观测变量

$$\mathbf{P}(X | \mathbf{e}) = \alpha \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y})$$

其中求和是针对所有可能的 \boldsymbol{y} （也就是未观测变量 \boldsymbol{Y} 的值的所有可能组合）。注意，变量 X 、 E 和 Y 构成了域变量的完整集合，所以 $\mathbf{P}(X, \mathbf{e}, \mathbf{y})$ 仅仅是完全联合分布的一个概率子集。

α 为归一化常数，要会合理利用以简化计算

独立性 Independence

[Independence \(probability theory\) - Wikipedia](#)

独立性是“事理”上对两个随机事件的关系进行评判，即不相关，互不影响，对方发生与否不影响我发生的概率，我发生与否也不影响对方发生的概率。

独立性定义为:

两个命题 a 和 b 是相互独立的, 即 a 和 b 的概率满足如下关系:

$$P(a | b) = P(a) \quad or \quad P(b | a) = P(b) \quad or \quad P(a \wedge b) = P(a)P(b)$$

两个随机变量 X 和 Y 是独立的, 当且仅当:

$$\mathbf{P}(X | Y) = \mathbf{P}(X) \quad or \quad \mathbf{P}(Y | X) = \mathbf{P}(Y) \quad or \quad \mathbf{P}(X, Y) = \mathbf{P}(X)\mathbf{P}(Y)$$

- “独立性”是一种定义, 是一个断言, 也就是说:
 - 如果两个事件独立, 那么一定满足下述关系式;
 - 如果两个事件满足上述关系式, 那么就称这两个事件相互独立。

多变量:

两两独立 pairwise independent

有限事件的集合 $\{A_i\}_{i=1}^n$ 中每对事件都是相互独立的, 则称这些事件是两两独立的:

$$\Pr(A_i \cap A_j) = \Pr(A_i)\Pr(A_j) \quad (\forall i, j \in \{1, \dots, n\}, i \neq j)$$

相互独立 mutually independent

有限事件的集合 $\{A_i\}_{i=1}^n$ 中, 每个事件都与其他任何事件构成的交集独立, 则这些事件是**相互独立**的

对样本空间的**任意有限子集**序列 A_1, \dots, A_n , 有:

$$\Pr(A_1 \cap \dots \cap A_n) = \Pr(A_1) \dots \Pr(A_n)$$

或:

$$\Pr\left(\bigcap_{i=1}^n A_i\right) = \prod_{i=1}^n \Pr(A_i)$$

贝叶斯法则

$$\mathbf{P}(Y | X) = \frac{\mathbf{P}(X | Y)\mathbf{P}(Y)}{\mathbf{P}(X)}$$

有证据变量的形式:

$$\mathbf{P}(Y | X, \mathbf{e}) = \frac{\mathbf{P}(X | Y, \mathbf{e})\mathbf{P}(Y | \mathbf{e})}{\mathbf{P}(X | \mathbf{e})}$$

条件独立性

一般定义:

$$\mathbf{P}(X, Y | Z) = \mathbf{P}(X | Z)\mathbf{P}(Y | Z)$$

- 绝对独立性 vs 条件独立性
 - 一般情况下, 两者不能互推, 即: **条件独立得不出独立, 独立得不出条件独立**
 - 绝对独立的两个变量, 在给定另一个变量后, 可能会相关。
 - 相关的两个变量, 在给定另一个变量后, 可能会条件独立。

朴素贝叶斯模型

使用朴素贝叶斯进行文本分类

给定原因时，所有结果都是条件独立的，那么完全联合分布可写作：

$$\mathbf{P}(\text{Cause}, \text{Effect}_1, \dots, \text{Effect}_n) = \mathbf{P}(\text{Cause}) \prod_i \mathbf{P}(\text{Effect}_i | \text{Cause})$$

经常作为在给定原因时，结果变量并不是严格独立的情况的近似。

给定一些观测结果：

$$\mathbf{P}(\text{Cause} | \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(\text{Cause}, \mathbf{e}, \mathbf{y})$$

进一步：

$$\begin{aligned} \mathbf{P}(\text{Cause} | \mathbf{e}) &= \alpha \sum_{\mathbf{y}} \mathbf{P}(\text{Cause}) \mathbf{P}(\mathbf{y} | \text{Cause}) \left(\prod_j \mathbf{P}(e_j | \text{Cause}) \right) \\ &= \alpha \mathbf{P}(\text{Cause}) \left(\prod_j \mathbf{P}(e_j | \text{Cause}) \right) \sum_{\mathbf{y}} \mathbf{P}(\mathbf{y} | \text{Cause}) \\ &= \alpha \mathbf{P}(\text{Cause}) \prod_j \mathbf{P}(e_j | \text{Cause}) \end{aligned}$$

对于每一个可能的原因，将原因的先验概率乘以在给定原因时所观测到的结果的条件概率，然后将结果归一化。

计算复杂度于观测到的结果数量呈线性关系？不依赖于为观测到的结果数量。

第 13 章 概率推理

- 13.1 不确定域的知识表示
- 13.2 贝叶斯网络的语义
 - 13.2.1 贝叶斯网络中的条件独立性关系
 - 13.2.2 条件分布的高效表示
 - 13.2.3 连续变量的贝叶斯网络
 - 13.2.4 案例研究：汽车保险
- 13.3 贝叶斯网络中的精确推断
 - 13.3.1 通过枚举进行推断
 - 13.3.2 变量消元算法
 - 13.3.3 精确推断的复杂性
 - 13.3.4 聚类算法
- 13.4 贝叶斯网络中的近似推理
 - 13.4.1 直接采样方法
 - 13.4.2 通过马尔可夫链模拟进行推断
 - 13.4.3 编译近似推断
- 13.5 因果网络
 - 13.5.1 表示动作：`do` 操作
 - 13.5.2 后门准则

本章介绍了**贝叶斯网络**，它是一种发展成熟的不确定知识表示方法。贝叶斯网络的作用大致类似于确定知识的命题逻辑。

- 贝叶斯网络是**有向无环图**，其中节点对应着随机变量；给定父节点，每个节点都能求出该节点的条件分布。
- 贝叶斯网络提供了一种简洁的方法来表示域内的**条件独立性关系**。
- 贝叶斯网络指定了其变量的联合概率分布。对所有变量的任一赋值的概率定义为局部条件分布中相应项的乘积。贝叶斯网络通常比显式枚举的联合分布小得多。
- 许多条件分布可以用正规分布族紧致地表示。**混合贝叶斯网络**，包括离散变量和连续变量，使用各种正则分布。
- 贝叶斯网络中的推断是指在给定一组证据变量的情况下，计算一组查询变量的概率分布。精确推断算法，如**变量消元**，尽可能高效地评估条件概率乘积之和。
- 在**多重树**（单连通网络）中，精确推断的时间与网络的规模呈线性关系。在一般情况下，这个问题是棘手的。
- 像**似然加权**和**马尔可夫链蒙特卡罗**这样的随机采样技术可以给出网络中的真实后验分布合理的估计，与精确算法相比，它们可以处理规模更大的网络。
- 贝叶斯网络捕捉概率影响，而**因果网络**捕捉因果关系，并允许对干预结果和观测值进行预测。

第 18 章 多智能体决策

- 18.1 多智能体环境的特性
 - 18.1.1 单个决策者
 - 18.1.2 多决策者
 - 18.1.3 多智能体规划
 - 18.1.4 多智能体规划：合作与协调
- 18.2 非合作博弈论
 - 18.2.1 单步博弈：正则形式博弈
 - 18.2.2 社会福利
 - 18.2.3 重复博弈
 - 18.2.4 序贯博弈：扩展形式
 - 18.2.5 不确定收益与辅助博弈
- 18.3 合作博弈论
 - 18.3.1 联盟结构与结果
 - 18.3.2 合作博弈中的策略
 - 18.3.3 合作博弈中的计算
- 18.4 做集体决策
 - 18.4.1 在合同网中分配任务
 - 18.4.2 通过拍卖分配稀缺资源
 - 18.4.3 投票
 - 18.4.4 议价

小结

- 当环境中与其他智能体需要合作或竞争时，多智能体规划是必要的。我们可以制定联合规划，但如果两个智能体要就执行哪个联合规划达成一致，就必须以某种协调形式加以加强。
- 博弈论描述的是在多个智能体相互作用的情况下，智能体的理性行为。博弈论之于多智能体决策，正如决策论之于单智能体决策。
- 博弈论中的解概念旨在描述博弈的理性结果——如果每个智能体都采取理性行为的情况下可能出现的结果。
- 非合作博弈论假设智能体必须独立做出决策。纳什均衡是非合作博弈论中最重要的解概念。纳什均衡是一种智能体没有动机偏离其指定策略的策略组合。我们有处理重复博弈和序贯博弈的技巧。
- 合作博弈论考虑的是智能体为了合作而达成具有约束力的协约以形成联盟的设置。合作博弈中的解概念试图阐明哪些联盟是稳定的（核），以及如何公平分配联盟获得的值（沙普利值）。
- 对于某些重要得多智能体决策类有专门的技术：任务共享的合同网，被用来有效地分配稀缺资源的拍卖，就共同利益进行议价以达成协约，以及聚集偏好的投票程序。

第一部分 人工智能基础

第1章 绪论

- 1.1 什么是人工智能
 - 1.1.1 类人行为：图灵测试方法
 - 1.1.2 类人思考：认知建模方法
 - 1.1.3 理性思考：“思维法则”方法
 - 1.1.4 理性行为：理性智能体方法
 - 1.1.5 益机
- 1.2 人工智能的基础
 - 1.2.1 哲学
 - 1.2.2 数学
 - 1.2.3 经济学
 - 1.2.4 神经科学
 - 1.2.5 心理学
 - 1.2.6 计算机工程
 - 1.2.7 控制理论与控制论
 - 1.2.8 语言学
- 1.3 人工智能的历史
 - 1.3.1 人工智能的诞生（1943—1956）
 - 1.3.2 早期热情高涨，期望无限（1952—1969）
 - 1.3.3 一些现实（1966—1973）
 - 1.3.4 专家系统（1969—1986）
 - 1.3.5 神经网络的回归（1986—现在）
 - 1.3.6 概率推理和机器学习（1987—现在）
 - 1.3.7 大数据（2001—现在）
 - 1.3.8 深度学习（2011—现在）
- 1.4 目前的先进技术
- 1.5 人工智能的风险和收益

第2章 智能体

- 2.1 智能体和环境
 - 2.2 良好行为：理性的概念
 - 2.2.1 性能度量
 - 2.2.2 理性
 - 2.2.3 全知、学习和自主
 - 2.3 环境的本质
 - 2.3.1 指定任务环境
 - 2.3.2 任务环境的属性
 - 2.4 智能体的结构
 - 2.4.1 智能体程序
 - 2.4.2 简单反射型智能体
 - 2.4.3 基于模型的反射型智能体
 - 2.4.4 基于目标的智能体
 - 2.4.5 基于效用的智能体
 - 2.4.6 学习型智能体
-

- 2.4.7 智能体程序的组件如何工作
-

第二部分 问题求解

第3章 通过搜索进行问题求解

- 3.1 问题求解智能体
 - 3.1.1 搜索问题和解
 - 3.1.2 问题形式化
- 3.2 问题示例
 - 3.2.1 标准化问题
 - 3.2.2 真实世界问题
 - 3.3 搜索算法
 - 3.3.1 最佳优先搜索
 - 3.3.2 搜索数据结构
 - 3.3.3 冗余路径
 - 3.3.4 问题求解性能评估
- 3.4 无信息搜索策略
 - 3.4.1 广度优先搜索
 - 3.4.2 Dijkstra 算法或一致代价搜索
 - 3.4.3 深度优先搜索与内存问题
 - 3.4.4 深度受限和迭代加深搜索
 - 3.4.5 双向搜索
 - 3.4.6 无信息搜索算法对比
- 3.5 有信息（启发式）搜索策略
 - 3.5.1 贪心最佳优先搜索
 - 3.5.2 A*搜索
 - 3.5.3 搜索等值线
 - 3.5.4 满意搜索：不可容许的启发式函数与加权 A*搜索
 - 3.5.5 内存受限搜索
 - 3.5.6 双向启发式搜索
- 3.6 启发式函数
 - 3.6.1 启发式函数的准确性对性能的影响
 - 3.6.2 从松弛问题出生成启发式函数
 - 3.6.3 从子问题出生成启发式函数：模式数据库
 - 3.6.4 使用地标生成启发式函数
 - 3.6.5 学习以更好地搜索
 - 3.6.6 从经验中学习启发式函数

第4章 复杂环境中的搜索

- 4.1 局部搜索和最优化问题
 - 4.1.1 爬山搜索
 - 4.1.2 模拟退火
 - 4.1.3 局部束搜索
-

- 4.1.4 进化算法
- 4.2 连续空间中的局部搜索
- 4.3 使用非确定性动作的搜索
 - 4.3.1 不稳定的真空吸尘器世界
 - 4.3.2 与或搜索树
 - 4.3.3 反复尝试
- 4.4 部分可观测环境中的搜索
 - 4.4.1 无观测信息的搜索
 - 4.4.2 部分可观测环境中的搜索
 - 4.4.3 求解部分可观测问题
 - 4.4.4 部分可观测环境中的智能体
- 4.5 在线搜索智能体和未知环境
 - 4.5.1 在线搜索问题
 - 4.5.2 在线搜索智能体
 - 4.5.3 在线局部搜索
 - 4.5.4 在线搜索中的学习

第 5 章 对抗搜索和博弈

- 5.1 博弈论
- 5.2 博弈中的优化决策
 - 5.2.1 极小化极大搜索算法 5.2.2 多人博弈中的最优决策
 - 5.2.3 剪枝
 - 5.2.4 移动顺序
- 5.3 启发式树搜索
 - 5.3.1 评价函数 5.3.2 截断搜索 5.3.3 前向剪枝 5.3.4 搜索和查表 5.4 蒙特卡罗树搜索
- 5.5 随机博弈
- 机会博弈的评价函数
- 5.6 部分可观测博弈
 - 5.6.1 四国军棋：部分可观测的国际象棋 5.6.2 纸牌游戏
- 5.7 博弈搜索算法的局限性

第 6 章 约束满足问题

- 6.1 定义约束满足问题
 - 6.1.1 问题示例：地图着色 6.1.2 问题示例：车间作业调度 6.1.3 CSP 形式体系的变体
- 6.2 约束传播：CSP 中的推断
 - 6.2.1 节点一致性 6.2.2 弧一致性 6.2.3 路径一致性 6.2.4 k 一致性 6.2.5 全局约束 6.2.6 数独
- 6.3 CSP 的回溯搜索
 - 6.3.1 变量排序和值排序 6.3.2 交替进行搜索和推理 6.3.3 智能回溯：向后看 6.3.4 约束学习

6.4 CSP 的局部搜索

6.5 问题的结构

6.5.1 割集调整 6.5.2 树分解 6.5.3 值对称

第三部分 知识、推理和规划

第 7 章 逻辑智能体

7.1 基于知识的智能体

7.2 wumpus 世界

7.3 逻辑

7.4 命题逻辑：一种非常简单的逻辑

7.4.1 语法 7.4.2 语义 7.4.3 一个简单的知识库 7.4.4 一个简单的推断过程

7.5 命题定理证明

7.5.1 推断与证明 7.5.2 通过归结证明 7.5.3 霍恩子句与确定子句 7.5.4 前向链接与反向链接

7.6 高效命题模型检验

7.6.1 完备的回溯算法 7.6.2 局部搜索算法 7.6.3 随机 SAT 问题概览

7.7 基于命题逻辑的智能体

7.7.1 世界的当前状态 7.7.2 混合智能体 7.7.3 逻辑状态估计 7.7.4 用命题推断进行规划

第 8 章 一阶逻辑

- 8.1 回顾表示
 - 8.1.1 思想的语言
 - 8.1.2 结合形式语言和自然语言的优点
 - 8.2 一阶逻辑的语法和语义
 - 8.2.1 一阶逻辑模型
 - 8.2.2 符号与解释
 - 8.2.3 项
 - 8.2.4 原子语句
 - 8.2.5 复合语句
 - 8.2.6 量词
 - 8.2.7 等词
 - 8.2.8 数据库语义
 - 8.3 使用一阶逻辑
 - 8.3.1 一阶逻辑的断言与查询
 - 8.3.2 亲属关系论域
 - 8.3.3 数、集合与列表
 - 8.3.4 wumpus 世界
 - 8.4 一阶逻辑中的知识工程
 - 8.4.1 知识工程的过程
 - 8.4.2 电子电路论域
-

第 9 章 一阶逻辑中的推断

- 9.1 命题推断与一阶推断 约简为命题推断
- 9.2 合一与一阶推断
- 9.2.1 合一
- 9.2.2 存储与检索
- 9.3 前向链接
- 9.3.1 一阶确定子句
- 9.3.2 简单的前向链接算法
- 9.3.3 高效前向链接
- 9.4 反向链接
- 9.4.1 反向链接算法
- 9.4.2 逻辑编程
- 9.4.3 冗余推断和无限循环
- 9.4.4 Prolog 的数据库语义
- 9.4.5 约束逻辑编程
- 9.5 归结
- 9.5.1 一阶逻辑的合取范式
- 9.5.2 归结推断规则
- 9.5.3 证明范例
- 9.5.4 归结的完备性
- 9.5.5 等词
- 9.5.6 归结策略

第 10 章 知识表示

- 10.1 本体论工程
- 10.2 类别与对象
- 10.2.1 物理组成
- 10.2.2 量度
- 10.2.3 对象：事物和物质
- 10.3 事件
- 10.3.1 时间
- 10.3.2 流和对象
- 10.4 精神对象和模态逻辑 其他模态逻辑
- 10.5 类别的推理系统
- 10.5.1 语义网络
- 10.5.2 描述逻辑
- 10.6 用缺省信息推理
- 10.6.1 限定与缺省逻辑
- 10.6.2 真值维护系统

第 11 章 自动规划

- 11.1 经典规划的定义
- 11.1.1 范例领域：航空货物运输
- 11.1.2 范例领域：备用轮胎问题
- 11.1.3 范例领域：积木世界
- 11.2 经典规划的算法

- 11.2.1 规划的前向状态空间搜索
- 11.2.2 规划的反向状态空间搜索
- 11.2.3 使用布尔可满足性规划
- 11.2.4 其他经典规划方法
- 11.3 规划的启发式方法
- 11.3.1 领域无关剪枝
- 11.3.2 规划中的状态抽象
- 11.4 分层规划
- 11.4.1 高层动作
- 11.4.2 搜索基元解
- 11.4.3 搜索抽象解
- 11.5 非确定性域的规划和行动
- 11.5.1 无传感器规划
- 11.5.2 应变规划
- 11.5.3 在线规划
- 11.6 时间、调度和资源
- 11.6.1 时间约束和资源约束的表示
- 11.6.2 解决调度问题
- 11.7 规划方法分析

第四部分 不确定知识和不确定推理

第 12 章 不确定性的量化

- 12.1 不确定性下的动作
 - 12.1.1 不确定性概述
 - 12.1.2 不确定性与理性决策
- 12.2 基本概率记号
 - 12.2.1 概率是关于什么的
 - 12.2.2 概率断言中的命题语言
 - 12.2.3 概率公理及其合理性
- 12.3 使用完全联合分布进行推断
- 12.4 独立性
- 12.5 贝叶斯法则及其应用
 - 12.5.1 应用贝叶斯法则：简单实例
 - 12.5.2 应用贝叶斯法则：合并证据
- 12.6 朴素贝叶斯模型
 - 使用朴素贝叶斯进行文本分类
- 12.7 重游 wumpus 世界

第 13 章 概率推理

- 13.1 不确定域的知识表示
- 13.2 贝叶斯网络的语义
 - 13.2.1 贝叶斯网络中的条件独立性关系
 - 13.2.2 条件分布的高效表示
 - 13.2.3 连续变量的贝叶斯网络
 - 13.2.4 案例研究：汽车保险

- 13.3 贝叶斯网络中的精确推断
 - 13.3.1 通过枚举进行推断
 - 13.3.2 变量消元算法
 - 13.3.3 精确推断的复杂性
 - 13.3.4 聚类算法
- 13.4 贝叶斯网络中的近似推理
 - 13.4.1 直接采样方法
 - 13.4.2 通过马尔可夫链模拟进行推断
 - 13.4.3 编译近似推断
- 13.5 因果网络
 - 13.5.1 表示动作: do 操作
 - 13.5.2 后门准则

第 14 章 时间上的概率推理

- 14.1 时间与不确定性
 - 14.1.1 状态与观测
 - 14.1.2 转移模型与传感器模型
- 14.2 时序模型中的推断
 - 14.2.1 滤波与预测
 - 14.2.2 平滑
 - 14.2.3 寻找最可能序列
- 14.3 隐马尔可夫模型
 - 14.3.1 简化矩阵算法
 - 14.3.2 隐马尔可夫模型示例: 定位
- 14.4 卡尔曼滤波器
 - 14.4.1 更新高斯分布
 - 14.4.2 简单的一维示例
 - 14.4.3 一般情况
 - 14.4.4 卡尔曼滤波的适用范围
- 14.5 动态贝叶斯网络
 - 14.5.1 构建动态贝叶斯网络
 - 14.5.2 动态贝叶斯网络中的精确推断
 - 14.5.3 动态贝叶斯网络中的近似推断

第 15 章 概率编程

- 15.1 关系概率模型
 - 15.1.1 语法与语义
 - 15.1.2 实例: 评定玩家的技能等级
 - 15.1.3 关系概率模型中的推断
- 15.2 开宇宙概率模型
 - 15.2.1 语义与语法
 - 15.2.2 开宇宙概率模型的推断
 - 15.2.3 示例
- 15.3 追踪复杂世界
 - 15.3.1 示例: 多目标跟踪
 - 15.3.2 示例: 交通监控

- 15.4 作为概率模型的程序
 - 15.4.1 示例：文本阅读
 - 15.4.2 语法与语义
 - 15.4.3 推断结果
 - 15.4.4 结合马尔可夫模型改进生成程序
 - 15.4.5 生成程序的推断

第 16 章 做简单决策

- 16.1 在不确定性下结合信念与愿望
 - 16.2 效用理论基础
 - 16.2.1 理性偏好的约束
 - 16.2.2 理性偏好导致效用
 - 16.3 效用函数
 - 16.3.1 效用评估和效用尺度
 - 16.3.2 金钱的效用
 - 16.3.3 期望效用与决策后失望
 - 16.3.4 人类判断与非理性
 - 16.4 多属性效用函数
 - 16.4.1 占优
 - 16.4.2 偏好结构与多属性效用
 - 16.5 决策网络
 - 16.5.1 使用决策网络表示决策问题
 - 16.5.2 评估决策网络
 - 16.6 信息价值
 - 16.6.1 简单示例
 - 16.6.2 完美信息的一般公式
 - 16.6.3 价值信息的性质
 - 16.6.4 信息收集智能体的实现
 - 16.6.5 非短视信息收集
 - 16.6.6 敏感性分析与健壮决策
 - 16.7 未知偏好
 - 16.7.1 个人偏好的不确定性
 - 16.7.2 顺从人类

第 17 章 做复杂决策

- 17.1 序贯决策问题
 - 17.1.1 时间上的效用
 - 17.1.2 最优策略与状态效用
 - 17.1.3 奖励规模
 - 17.1.4 表示 MDP
- 17.2 MDP 的算法
 - 17.2.1 价值迭代
 - 17.2.2 策略迭代
 - 17.2.3 线性规划
 - 17.2.4 MDP 的在线算法
- 17.3 老虎机问题

-
- 17.3.1 计算基廷斯指数
 - 17.3.2 伯努利老虎机
 - 17.3.3 近似最优老虎机策略
 - 17.3.4 不可索引变体
 - 17.4 部分可观测 MDP
 - POMDP 的定义
 - 17.5 求解 POMDP 的算法
 - 17.5.1 POMDP 的价值迭代
 - 17.5.2 POMDP 的在线算法

第 18 章 多智能体决策

- 18.1 多智能体环境的特性
 - 18.1.1 单个决策者
 - 18.1.2 多决策者
 - 18.1.3 多智能体规划
 - 18.1.4 多智能体规划：合作与协调
- 18.2 非合作博弈论
 - 18.2.1 单步博弈：正则形式博弈
 - 18.2.2 社会福利
 - 18.2.3 重复博弈
 - 18.2.4 序贯博弈：扩展形式
 - 18.2.5 不确定收益与辅助博弈
- 18.3 合作博弈论
 - 18.3.1 联盟结构与结果
 - 18.3.2 合作博弈中的策略
 - 18.3.3 合作博弈中的计算
- 18.4 做集体决策
 - 18.4.1 在合同网中分配任务
 - 18.4.2 通过拍卖分配稀缺资源
 - 18.4.3 投票
 - 18.4.4 议价

第五部分 机器学习

第 19 章 样例学习

- 19.1 学习的形式
 - 19.2 监督学习 问题示例：餐厅等待问题
 - 19.3 决策树学习
 - 19.3.1 决策树的表达能力
 - 19.3.2 从样例中学习决策树
 - 19.3.3 选择测试属性
 - 19.3.4 泛化与过拟合
 - 19.3.5 拓展决策树的适用范围
 - 19.4 模型选择与模型优化
 - 19.4.1 模型选择
 - 19.4.2 从错误率到损失函数
-

-
- 19.4.3 正则化
 - 19.4.4 超参数调整
 - 19.5 学习理论 PAC 学习示例：学习决策列表
 - 19.6 线性回归与分类
 - 19.6.1 单变量线性回归
 - 19.6.2 梯度下降
 - 19.6.3 多变量线性回归
 - 19.6.4 带有硬阈值的线性分类器
 - 19.6.5 基于逻辑斯谛回归的线性分类器
 - 19.7 非参数模型
 - 19.7.1 最近邻模型
 - 19.7.2 使用 k-d 树寻找最近邻
 - 19.7.3 局部敏感哈希
 - 19.7.4 非参数回归
 - 19.7.5 支持向量机
 - 19.7.6 核技巧
 - 19.8 集成学习
 - 19.8.1 自助聚合法
 - 19.8.2 随机森林法
 - 19.8.3 堆叠法
 - 19.8.4 自适应提升法
 - 19.8.5 梯度提升法
 - 19.8.6 在线学习
 - 19.9 开发机器学习系统
 - 19.9.1 问题形式化
 - 19.9.2 数据收集、评估和管理
 - 19.9.3 模型选择与训练
 - 19.9.4 信任、可解释性、可说明性
 - 19.9.5 操作、监控和维护

第 20 章 概率模型学习

- 20.1 统计学习
- 20.2 完全数据学习
- 20.2.1 最大似然参数学习：离散模型
- 20.2.2 朴素贝叶斯模型
- 20.2.3 生成模型和判别模型
- 20.2.4 最大似然参数学习：连续模型
- 20.2.5 贝叶斯参数学习
- 20.2.6 贝叶斯线性回归
- 20.2.7 贝叶斯网络结构学习
- 20.2.8 非参数模型密度估计
- 20.3 隐变量学习：EM 算法
 - 20.3.1 无监督聚类：学习混合高斯
 - 20.3.2 学习带隐变量的贝叶斯网络参数值
 - 20.3.3 学习隐马尔可夫模型
 - 20.3.4 EM 算法的一般形式
 - 20.3.5 学习带隐变量的贝叶斯网络结构

第 21 章 深度学习

-
- 21.1 简单前馈网络
 - 21.1.1 网络作为复杂函数
 - 21.1.2 梯度与学习
 - 21.2 深度学习的计算图
 - 21.2.1 输入编码
 - 21.2.2 输出层与损失函数
 - 21.2.3 隐藏层
 - 21.3 卷积网络
 - 21.3.1 池化与下采样
 - 21.3.2 卷积神经网络的张量运算
 - 21.3.3 残差网络
 - 21.4 学习算法
 - 21.4.1 计算图中的梯度计算
 - 21.4.2 批量归一化
 - 21.5 泛化
 - 21.5.1 选择正确的网络架构
 - 21.5.2 神经架构搜索
 - 21.5.3 权重衰减
 - 21.5.4 暂退法
 - 21.6 循环神经网络
 - 21.6.1 训练基本的循环神经网络
 - 21.6.2 长短期记忆 RNN
 - 21.7 无监督学习与迁移学习
 - 21.7.1 无监督学习
 - 21.7.2 迁移学习和多任务学习
 - 21.8 应用
 - 21.8.1 视觉
 - 21.8.2 自然语言处理
 - 21.8.3 强化学习

第 22 章 强化学习

22.1 从奖励中学习 22.2 被动强化学习 22.2.1 直接效用估计 22.2.2 自适应动态规划 22.2.3 时序差分学习
 22.3 主动强化学习 22.3.1 探索 22.3.2 安全探索 22.3.3 时序差分 Q 学习 22.4 强化学习中的泛化 22.4.1 近似直接效用估计 22.4.2 近似时序差分学习 22.4.3 深度强化学习 22.4.4 奖励函数设计 22.4.5 分层强化学习
 22.5 策略搜索 22.6 学徒学习与逆强化学习 22.7 强化学习的应用 22.7.1 在电子游戏中的应用 22.7.2 在机器人控制中的应用

第六部分 沟通、感知和行动

第 23 章 自然语言处理

23.1 语言模型

23.1.1 词袋模型 23.1.2 n 元单词模型 23.1.3 其他 n 元模型 23.1.4 n 元模型的平滑 23.1.5 单词表示 23.1.6 词性标注 23.1.7 语言模型的比较 23.2 文法 E0 的词典 23.3 句法分析 23.3.1 依存分析 23.3.2 从样例中学习句法分析器 23.4 扩展文法 23.4.1 语义解释 23.4.2 学习语义文法 23.5 真实自然语言的复杂性 23.6 自然语言任务

第 24 章 自然语言处理中的深度学习

24.1 词嵌入 24.2 自然语言处理中的循环神经网络 24.2.1 使用循环神经网络的语言模型 24.2.2 用循环神经网络进行分类 24.2.3 自然语言处理任务中的 LSTM 模型 24.3 序列到序列模型 24.3.1 注意力 24.3.2 解码 24.4 Transformer 架构 24.4.1 自注意力 24.4.2 从自注意力到 Transformer 24.5 预训练和迁移学习 24.5.1 预训练词嵌入 24.5.2 预训练上下文表示 24.5.3 掩码语言模型 24.6 最高水平 (SOTA)

第 25 章 计算机视觉

25.1 引言 25.2 图像形成 25.2.1 无透镜成像：针孔照相机 25.2.2 透镜系统 25.2.3 缩放正交投影 25.2.4 光线与明暗 25.2.5 颜色 25.3 简单图像特征 25.3.1 边缘 25.3.2 纹理 25.3.3 光流 25.3.4 自然图像分割 25.4 图像分类 25.4.1 基于卷积神经网络的图像分类 25.4.2 卷积神经网络对图像分类问题有效的原因 25.5 物体检测 25.6 三维世界 25.6.1 多个视图下的三维线索 25.6.2 双目立体视觉 25.6.3 移动摄像机给出的三维线索 25.6.4 单个视图的三维线索 25.7 计算机视觉的应用 25.7.1 理解人类行为 25.7.2 匹配图片与文字 25.7.3 多视图重建 25.7.4 单视图中的几何 25.7.5 生成图片 25.7.6 利用视觉控制运动

第 26 章 机器人学

26.1 机器人 26.2 机器人硬件 26.2.1 机器人的硬件层面分类 26.2.2 感知世界 26.2.3 产生运动 26.3 机器人学解决哪些问题 26.4 机器人感知 26.4.1 定位与地图构建 26.4.2 其他感知类型 26.4.3 机器人感知中的监督学习与无监督学习 26.5 规划与控制 26.5.1 构形空间 26.5.2 运动规划 26.5.3 轨迹跟踪控制 26.5.4 最优控制 26.6 规划不确定的运动 26.7 机器人学中的强化学习 26.7.1 利用模型 26.7.2 利用其他信息 26.8 人类与机器人 26.8.1 协调 26.8.2 学习做人类期望的事情 26.9 其他机器人框架 26.9.1 反应式控制器 26.9.2 包容架构 26.10 应用领域

第七部分 总结

第 27 章 人工智能的哲学、伦理和安全性

27.1 人工智能的极限 27.1.1 由非形式化得出的论据 27.1.2 由能力缺陷得出的论据 27.1.3 数学异议 27.1.4 衡量人工智能 27.2 机器能真正地思考吗 27.2.1 中文房间 27.2.2 意识与感质 27.3 人工智能的伦理 27.3.1 致命性自主武器 27.3.2 监控、安全与隐私 27.3.3 公平与偏见 27.3.4 信任与透明度 27.3.5 工作前景 27.3.6 机器人权利 27.3.7 人工智能安全性

第 28 章 人工智能的未来

28.1 人工智能组件 28.2 人工智能架构

附录 A 数学背景知识

A.1 复杂性分析和 $\mathcal{O}()$ 记号 A.1.1 渐近分析 A.1.2 NP 困难和固有的难题 A.2 向量、矩阵和线性代数 A.3 概率分布

读者服务

```
(base) → sys-ycompiler git:(master) ✘ sudo apt install nvidia-cuda-dev
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
dkms firmware-nvidia-gsp glx-alternative-mesa glx-alternative-nvidia glx-diversions libaccinj64-1:
libcublaslt11 libcuda1 libcuda1 libcudart11.0 libcufft10 libcufftw10 libcuinj64-11.8 libcupti-dev libcupti-
libcusolver11 libcusolvermg11 libcusparse11 libgl-dev libglx-dev libnppc11 libnppial11 libnppicc1:
libnppim11 libnppist11 libnppisu11 libnppitc11 libnpps11 libnvblas11 libnvivid1 libnvidia-cfg1 l:
libnvidia-pkcs11-openssl3 libnvidia-ptxjitcompiler1 libnvjpeg11 libnvrtc-builtins11.8 libnvrtc11.:
libvdpau-dev linux-compiler-gcc-12-x86 linux-headers-6.1.0-27-amd64 linux-headers-6.1.0-27-common
node-html5shiv nvidia-alternative nvidia-installer-cleanup nvidia-kernel-common nvidia-kernel-dkms
nvidia-legacy-check nvidia-modprobe nvidia-persistenced nvidia-smi nvidia-support update-glx

Suggested packages:
menu nvidia-driver | nvidia-driver-any nvidia-cuda-mps nvidia-cuda-toolkit libvdpau-doc nodejs

Recommended packages:
libcudalib:i386

The following NEW packages will be installed:
dkms firmware-nvidia-gsp glx-alternative-mesa glx-alternative-nvidia glx-diversions libaccinj64-1:
libcublaslt11 libcuda1 libcuda1 libcudart11.0 libcufft10 libcufftw10 libcuinj64-11.8 libcupti-dev libcupti-
libcusolver11 libcusolvermg11 libcusparse11 libgl-dev libglx-dev libnppc11 libnppial11 libnppicc1:
libnppim11 libnppist11 libnppisu11 libnppitc11 libnpps11 libnvblas11 libnvivid1 libnvidia-cfg1 l:
libnvidia-pkcs11-openssl3 libnvidia-ptxjitcompiler1 libnvjpeg11 libnvrtc-builtins11.8 libnvrtc11.:
libvdpau-dev linux-compiler-gcc-12-x86 linux-headers-6.1.0-27-amd64 linux-headers-6.1.0-27-common
node-html5shiv nvidia-alternative nvidia-cuda-dev nvidia-installer-cleanup nvidia-kernel-common n:
nvidia-legacy-check nvidia-modprobe nvidia-persistenced nvidia-smi nvidia-support update-glx

0 upgraded, 67 newly installed, 0 to remove and 2 not upgraded.
Need to get 1,592 MB of archives.

After this operation, 5,429 MB of additional disk space will be used.

Do you want to continue? [Y/n] Y
Get:1 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 dkms all 3.0.10-8+deb12u1 [48.7 kB]
Get:2 http://mirrors.ustc.edu.cn/debian bookworm/contrib amd64 update-glx amd64 1.2.2 [5,432 B]
Get:3 http://mirrors.ustc.edu.cn/debian bookworm/contrib amd64 glx-alternative-mesa amd64 1.2.2 [4,:
Get:4 http://mirrors.ustc.edu.cn/debian bookworm/contrib amd64 nvidia-installer-cleanup amd64 202202:
Get:5 http://mirrors.ustc.edu.cn/debian bookworm/contrib amd64 glx-diversions amd64 1.2.2 [7,408 B]
Get:6 http://mirrors.ustc.edu.cn/debian bookworm/contrib amd64 glx-alternative-nvidia amd64 1.2.2 [!:
Get:7 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-legacy-check amd64 535.183.0:
Get:8 http://mirrors.ustc.edu.cn/debian bookworm/contrib amd64 nvidia-support amd64 20220217+3-deb12:
Get:9 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-alternative amd64 535.183.01:
Get:10 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvidia-ptxjitcompiler1 amd64 53:
Get:11 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvidia-pkcs11-openssl3 amd64 53:
Get:12 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcuda1 amd64 535.183.01-1-deb12u:
Get:13 http://mirrors.ustc.edu.cn/debian bookworm/non-free-firmware amd64 firmware-nvidia-gsp amd64
Get:14 http://mirrors.ustc.edu.cn/debian bookworm/contrib amd64 nvidia-kernel-common amd64 20220217-:
Get:15 http://mirrors.ustc.edu.cn/debian bookworm/contrib amd64 nvidia-modprobe amd64 535.161.07-1-:
Get:16 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-kernel-support amd64 535.18:
Get:17 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-kernel-dkms amd64 535.183.0:
```



```
Get:63 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 linux-kbuild-6.1 amd64 6.1.115-1 [1,17]
Get:64 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 linux-headers-6.1.0-27-amd64 amd64 6.1
Get:65 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 linux-headers-amd64 amd64 6.1.115-1 [1,
Get:66 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-cuda-dev amd64 11.8.89~11.8
Ign:66 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-cuda-dev amd64 11.8.89~11.8
Get:67 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-smi amd64 535.183.01-1~deb12u1
Get:36 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcusparse11 amd64 11.7.5.86~11.8
Get:66 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-cuda-dev amd64 11.8.89~11.8
Fetched 944 MB in 21min 31s (731 kB/s)
Extracting templates from packages: 100%
Preconfiguring packages ...
Selecting previously unselected package dkms.
(Reading database ... 276371 files and directories currently installed.)
Preparing to unpack .../dkms_3.0.10-8+deb12u1_all.deb ...
Unpacking dkms (3.0.10-8+deb12u1) ...
Selecting previously unselected package update-glx.
Preparing to unpack .../update-glx_1.2.2_amd64.deb ...
Unpacking update-glx (1.2.2) ...
Selecting previously unselected package glx-alternative-mesa.
Preparing to unpack .../glx-alternative-mesa_1.2.2_amd64.deb ...
Unpacking glx-alternative-mesa (1.2.2) ...
Selecting previously unselected package nvidia-installer-cleanup.
Preparing to unpack .../nvidia-installer-cleanup_20220217+3~deb12u1_amd64.deb ...
Unpacking nvidia-installer-cleanup (20220217+3~deb12u1) ...
Setting up nvidia-installer-cleanup (20220217+3~deb12u1) ...
Selecting previously unselected package glx-diversions.
(Reading database ... 276535 files and directories currently installed.)
Preparing to unpack .../glx-diversions_1.2.2_amd64.deb ...
Unpacking glx-diversions (1.2.2) ...
Selecting previously unselected package glx-alternative-nvidia.
Preparing to unpack .../glx-alternative-nvidia_1.2.2_amd64.deb ...
Unpacking glx-alternative-nvidia (1.2.2) ...
Selecting previously unselected package nvidia-legacy-check.
Preparing to unpack .../nvidia-legacy-check_535.183.01-1~deb12u1_amd64.deb ...
Unpacking nvidia-legacy-check (535.183.01-1~deb12u1) ...
Selecting previously unselected package nvidia-support.
Preparing to unpack .../nvidia-support_20220217+3~deb12u1_amd64.deb ...
Unpacking nvidia-support (20220217+3~deb12u1) ...
Setting up nvidia-legacy-check (535.183.01-1~deb12u1) ...
Selecting previously unselected package nvidia-alternative.
(Reading database ... 276573 files and directories currently installed.)
Preparing to unpack .../00-nvidia-alternative_535.183.01-1~deb12u1_amd64.deb ...
Unpacking nvidia-alternative (535.183.01-1~deb12u1) ...
Selecting previously unselected package libnvidia-ptxjitcompiler1:amd64.
Preparing to unpack .../01-libnvidia-ptxjitcompiler1_535.183.01-1~deb12u1_amd64.deb ...
Unpacking libnvidia-ptxjitcompiler1:amd64 (535.183.01-1~deb12u1) ...
Selecting previously unselected package libnvidia-pkcs11-openssl3:amd64.
Preparing to unpack .../02-libnvidia-pkcs11-openssl3_535.183.01-1~deb12u1_amd64.deb ...
```

```
Unpacking libnvidia-pkcs11-openssl3:amd64 (535.183.01-1~deb12u1) ...
Selecting previously unselected package libcudai:amd64.
Preparing to unpack .../03-libcuda1_535.183.01-1~deb12u1_amd64.deb ...
Unpacking libcudai:amd64 (535.183.01-1~deb12u1) ...
Selecting previously unselected package firmware-nvidia-gsp.
Preparing to unpack .../04-firmware-nvidia-gsp_535.183.01-1~deb12u1_amd64.deb ...
Unpacking firmware-nvidia-gsp (535.183.01-1~deb12u1) ...
Selecting previously unselected package nvidia-kernel-common.
Preparing to unpack .../05-nvidia-kernel-common_20220217+3~deb12u1_amd64.deb ...
Unpacking nvidia-kernel-common (20220217+3~deb12u1) ...
Selecting previously unselected package nvidia-modprobe.
Preparing to unpack .../06-nvidia-modprobe_535.161.07-1~deb12u1_amd64.deb ...
Unpacking nvidia-modprobe (535.161.07-1~deb12u1) ...
Selecting previously unselected package nvidia-kernel-support.
Preparing to unpack .../07-nvidia-kernel-support_535.183.01-1~deb12u1_amd64.deb ...
Unpacking nvidia-kernel-support (535.183.01-1~deb12u1) ...
Selecting previously unselected package nvidia-kernel-dkms.
Preparing to unpack .../08-nvidia-kernel-dkms_535.183.01-1~deb12u1_amd64.deb ...
Unpacking nvidia-kernel-dkms (535.183.01-1~deb12u1) ...
Selecting previously unselected package libnvidia-cfg1:amd64.
Preparing to unpack .../09-libnvidia-cfg1_535.183.01-1~deb12u1_amd64.deb ...
Unpacking libnvidia-cfg1:amd64 (535.183.01-1~deb12u1) ...
Selecting previously unselected package nvidia-persistenced.
Preparing to unpack .../10-nvidia-persistenced_535.171.04-1~deb12u1_amd64.deb ...
Unpacking nvidia-persistenced (535.171.04-1~deb12u1) ...
Selecting previously unselected package libcupti11.8:amd64.
Preparing to unpack .../11-libcupti11.8_11.8.87~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libcupti11.8:amd64 (11.8.87~11.8.0-5~deb12u1) ...
Selecting previously unselected package libaccinj64-11.8:amd64.
Preparing to unpack .../12-libaccinj64-11.8_11.8.87~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libaccinj64-11.8:amd64 (11.8.87~11.8.0-5~deb12u1) ...
Selecting previously unselected package libcu++-dev.
Preparing to unpack .../13-libcu++-dev_1.8.1-2_all.deb ...
Unpacking libcu++-dev (1.8.1-2) ...
Selecting previously unselected package libcub-dev.
Preparing to unpack .../14-libcub-dev_1.17.2-2_all.deb ...
Unpacking libcub-dev (1.17.2-2) ...
Selecting previously unselected package libcublaslt11:amd64.
Preparing to unpack .../15-libcublaslt11_11.11.3.6-11.8.0-5~deb12u1_amd64.deb ...
Unpacking libcublaslt11:amd64 (11.11.3.6~11.8.0-5~deb12u1) ...
Selecting previously unselected package libcublas11:amd64.
Preparing to unpack .../16-libcublas11_11.11.3.6-11.8.0-5~deb12u1_amd64.deb ...
Unpacking libcublas11:amd64 (11.11.3.6~11.8.0-5~deb12u1) ...
Selecting previously unselected package libcudart11.0:amd64.
Preparing to unpack .../17-libcudart11.0_11.8.89~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libcudart11.0:amd64 (11.8.89~11.8.0-5~deb12u1) ...
Selecting previously unselected package libcufft10:amd64.
Preparing to unpack .../18-libcufft10_11.1.1+~10.9.0.58~11.8.0-5~deb12u1_amd64.deb ...
```

```
Unpacking libcufft10:amd64 (11.1.1+~10.9.0.58-11.8.0-5-deb12u1) ...
Selecting previously unselected package libcufftw10:amd64.
Preparing to unpack .../19-libcufftw10_11.1.1+~10.9.0.58-11.8.0-5-deb12u1_amd64.deb ...
Unpacking libcufftw10:amd64 (11.1.1+~10.9.0.58-11.8.0-5-deb12u1) ...
Selecting previously unselected package libcuinj64-11.8:amd64.
Preparing to unpack .../20-libcuinj64-11.8_11.8.87~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libcuinj64-11.8:amd64 (11.8.87~11.8.0-5~deb12u1) ...
Selecting previously unselected package libcupti-dev:amd64.
Preparing to unpack .../21-libcupti-dev_11.8.87~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libcupti-dev:amd64 (11.8.87~11.8.0-5~deb12u1) ...
Selecting previously unselected package node-html5shiv.
Preparing to unpack .../22-node-html5shiv_3.7.3+dfsg-5_all.deb ...
Unpacking node-html5shiv (3.7.3+dfsg-5) ...
Selecting previously unselected package libcupti-doc.
Preparing to unpack .../23-libcupti-doc_11.8.87~11.8.0-5~deb12u1_all.deb ...
Unpacking libcupti-doc (11.8.87~11.8.0-5~deb12u1) ...
Selecting previously unselected package libcurand10:amd64.
Preparing to unpack .../24-libcurand10_11.1.1+~10.3.0.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libcurand10:amd64 (11.1.1+~10.3.0.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libcusolver11:amd64.
Preparing to unpack .../25-libcusolver11_11.4.1.48~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libcusolver11:amd64 (11.4.1.48~11.8.0-5~deb12u1) ...
Selecting previously unselected package libcusolvermg11:amd64.
Preparing to unpack .../26-libcusolvermg11_11.4.1.48~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libcusolvermg11:amd64 (11.4.1.48~11.8.0-5~deb12u1) ...
Selecting previously unselected package libcusparse11:amd64.
Preparing to unpack .../27-libcusparse11_11.7.5.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libcusparse11:amd64 (11.7.5.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libglx-dev:amd64.
Preparing to unpack .../28-libglx-dev_1.6.0-1_amd64.deb ...
Unpacking libglx-dev:amd64 (1.6.0-1) ...
Selecting previously unselected package libgl-dev:amd64.
Preparing to unpack .../29-libgl-dev_1.6.0-1_amd64.deb ...
Unpacking libgl-dev:amd64 (1.6.0-1) ...
Selecting previously unselected package libnppc11:amd64.
Preparing to unpack .../30-libnppc11_11.8.0.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnppc11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnppial11:amd64.
Preparing to unpack .../31-libnppial11_11.8.0.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnppial11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnppicc11:amd64.
Preparing to unpack .../32-libnppicc11_11.8.0.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnppicc11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnppidei11:amd64.
Preparing to unpack .../33-libnppidei11_11.8.0.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnppidei11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnppif11:amd64.
Preparing to unpack .../34-libnppif11_11.8.0.86~11.8.0-5~deb12u1_amd64.deb ...
```

```
Unpacking libnppif11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnppig11:amd64.
Preparing to unpack .../35-libnppig11_11.8.0.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnppig11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnppim11:amd64.
Preparing to unpack .../36-libnppim11_11.8.0.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnppim11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnppist11:amd64.
Preparing to unpack .../37-libnppist11_11.8.0.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnppist11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnppisu11:amd64.
Preparing to unpack .../38-libnppisu11_11.8.0.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnppisu11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnppitc11:amd64.
Preparing to unpack .../39-libnppitc11_11.8.0.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnppitc11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnpps11:amd64.
Preparing to unpack .../40-libnpps11_11.8.0.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnpps11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnvblas11:amd64.
Preparing to unpack .../41-libnvblas11_11.11.3.6~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnvblas11:amd64 (11.11.3.6~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnvcuvid1:amd64.
Preparing to unpack .../42-libnvcuvid1_535.183.01-1~deb12u1_amd64.deb ...
Unpacking libnvcuvid1:amd64 (535.183.01-1~deb12u1) ...
Selecting previously unselected package libnvidia-ml1:amd64.
Preparing to unpack .../43-libnvidia-ml1_535.183.01-1~deb12u1_amd64.deb ...
Unpacking libnvidia-ml1:amd64 (535.183.01-1~deb12u1) ...
Selecting previously unselected package libnvidia-ml-dev:amd64.
Preparing to unpack .../44-libnvidia-ml-dev_11.8.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnvidia-ml-dev:amd64 (11.8.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnvjpeg11:amd64.
Preparing to unpack .../45-libnvjpeg11_11.9.0.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnvjpeg11:amd64 (11.9.0.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnvrtc-builtins11.8:amd64.
Preparing to unpack .../46-libnvrtc-builtins11.8_11.8.89~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnvrtc-builtins11.8:amd64 (11.8.89~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnvrtc11.2:amd64.
Preparing to unpack .../47-libnvrtc11.2_11.8.89~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnvrtc11.2:amd64 (11.8.89~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnvtoolsext1:amd64.
Preparing to unpack .../48-libnvtoolsext1_11.8.86~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnvtoolsext1:amd64 (11.8.86~11.8.0-5~deb12u1) ...
Selecting previously unselected package libnvvm4:amd64.
Preparing to unpack .../49-libnvvm4_11.8.89~11.8.0-5~deb12u1_amd64.deb ...
Unpacking libnvvm4:amd64 (11.8.89~11.8.0-5~deb12u1) ...
Selecting previously unselected package libthrust-dev.
Preparing to unpack .../50-libthrust-dev_1.17.2-2_all.deb ...
```

```
Unpacking libthrust-dev (1.17.2-2) ...
Selecting previously unselected package libvdpau-dev:amd64.
Preparing to unpack .../51-libvdpau-dev_1.5-2_amd64.deb ...
Unpacking libvdpau-dev:amd64 (1.5-2) ...
Selecting previously unselected package linux-compiler-gcc-12-x86.
Preparing to unpack .../52-linux-compiler-gcc-12-x86_6.1.115-1_amd64.deb ...
Unpacking linux-compiler-gcc-12-x86 (6.1.115-1) ...
Selecting previously unselected package linux-headers-6.1.0-27-common.
Preparing to unpack .../53-linux-headers-6.1.0-27-common_6.1.115-1_all.deb ...
Unpacking linux-headers-6.1.0-27-common (6.1.115-1) ...
Selecting previously unselected package linux-kbuild-6.1.
Preparing to unpack .../54-linux-kbuild-6.1_6.1.115-1_amd64.deb ...
Unpacking linux-kbuild-6.1 (6.1.115-1) ...
Selecting previously unselected package linux-headers-6.1.0-27-amd64.
Preparing to unpack .../55-linux-headers-6.1.0-27-amd64_6.1.115-1_amd64.deb ...
Unpacking linux-headers-6.1.0-27-amd64 (6.1.115-1) ...
Selecting previously unselected package linux-headers-amd64.
Preparing to unpack .../56-linux-headers-amd64_6.1.115-1_amd64.deb ...
Unpacking linux-headers-amd64 (6.1.115-1) ...
Selecting previously unselected package nvidia-cuda-dev:amd64.
Preparing to unpack .../57-nvidia-cuda-dev_11.8.89~11.8.0-5~deb12u1_amd64.deb ...
Unpacking nvidia-cuda-dev:amd64 (11.8.89~11.8.0-5~deb12u1) ...
Selecting previously unselected package nvidia-smi.
Preparing to unpack .../58-nvidia-smi_535.183.01-1~deb12u1_amd64.deb ...
Unpacking nvidia-smi (535.183.01-1~deb12u1) ...
Setting up nvidia-support (20220217+3~deb12u1) ...
Setting up linux-headers-6.1.0-27-common (6.1.115-1) ...
Setting up libcusparse11:amd64 (11.7.5.86~11.8.0-5~deb12u1) ...
Setting up nvidia-kernel-common (20220217+3~deb12u1) ...
Setting up libnppc11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Setting up libcu++-dev (1.8.1-2) ...
Setting up node-html5shiv (3.7.3+dfsg-5) ...
Setting up libcupti-doc (11.8.87~11.8.0-5~deb12u1) ...
Setting up update-glx (1.2.2) ...
Setting up libcuda11.0:amd64 (11.8.89~11.8.0-5~deb12u1) ...
Setting up libnppisu11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Setting up dkms (3.0.10-8+deb12u1) ...
Setting up linux-compiler-gcc-12-x86 (6.1.115-1) ...
Setting up nvidia-modprobe (535.161.07-1~deb12u1) ...
Setting up libnppicc11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Setting up libcupti11.8:amd64 (11.8.87~11.8.0-5~deb12u1) ...
Setting up libnvjpeg11:amd64 (11.9.0.86~11.8.0-5~deb12u1) ...
Setting up libcublaslt11:amd64 (11.11.3.6~11.8.0-5~deb12u1) ...
Setting up libnvrtc-builtins11.8:amd64 (11.8.89~11.8.0-5~deb12u1) ...
Setting up libnpps11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Setting up libnppim11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Setting up libcuftt10:amd64 (11.1.1+-10.9.0.58~11.8.0-5~deb12u1) ...
Setting up libnvidia-ptxjitcompiler1:amd64 (535.183.01-1~deb12u1) ...
```

```

Setting up libnppitc11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Setting up libaccinj64-11.8:amd64 (11.8.87~11.8.0-5~deb12u1) ...
Setting up libnppist11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Setting up firmware-nvidia-gsp (535.183.01-1~deb12u1) ...
Setting up libglx-dev:amd64 (1.6.0-1) ...
Setting up libnvvm4:amd64 (11.8.89~11.8.0-5~deb12u1) ...
Setting up libvdpau-dev:amd64 (1.5-2) ...
Setting up libnvtoolsext1:amd64 (11.8.86~11.8.0-5~deb12u1) ...
Setting up libcub-dev (1.17.2-2) ...
Setting up linux-kbuild-6.1 (6.1.115-1) ...
Setting up libnppig11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Setting up libgl-dev:amd64 (1.6.0-1) ...
Setting up libcurand10:amd64 (11.1.1+~10.3.0.86~11.8.0-5~deb12u1) ...
Setting up libnppidei11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Setting up libthrust-dev (1.17.2-2) ...
Setting up libnppial11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Setting up libnppif11:amd64 (11.8.0.86~11.8.0-5~deb12u1) ...
Setting up libcufftw10:amd64 (11.1.1+~10.9.0.58~11.8.0-5~deb12u1) ...
Setting up libcublas11:amd64 (11.11.3.6~11.8.0-5~deb12u1) ...
Setting up libcupti-dev:amd64 (11.8.87~11.8.0-5~deb12u1) ...
Setting up glx-alternative-mesa (1.2.2) ...
Setting up libnvblas11:amd64 (11.11.3.6~11.8.0-5~deb12u1) ...
Setting up glx-diversions (1.2.2) ...
Removing diverted 'libGL.so' symlink with unexpected target 'libGL.so.1'.
Restoring diverted 'libGL.so' symlink.
update-alternatives: using /usr/lib/mesa-diverted to provide /usr/lib/glx (glx) in auto mode
Setting up libcusolver11:amd64 (11.4.1.48~11.8.0-5~deb12u1) ...
Setting up libnvrtc11.2:amd64 (11.8.89~11.8.0-5~deb12u1) ...
Setting up linux-headers-6.1.0-27-amd64 (6.1.115-1) ...
/etc/kernel/header_postinst.d/dkms:
dkms: running auto installation service for kernel 6.1.0-27-amd64.
dkms: autoinstall for kernel: 6.1.0-27-amd64.
Setting up libcusolvermg11:amd64 (11.4.1.48~11.8.0-5~deb12u1) ...
Setting up linux-headers-amd64 (6.1.115-1) ...
Processing triggers for libc-bin (2.36-9+deb12u9) ...
Processing triggers for man-db (2.11.2-2) ...
Processing triggers for mailcap (3.70+nmu1) ...
Processing triggers for desktop-file-utils (0.26-1) ...
Processing triggers for initramfs-tools (0.142+deb12u1) ...
update-initramfs: Generating /boot/initrd.img-6.1.0-27-amd64
Processing triggers for gnome-menus (3.36.0-1.1) ...
Processing triggers for glx-alternative-mesa (1.2.2) ...
update-alternatives: updating alternative /usr/lib/mesa-diverted because link group glx has changed
update-alternatives: using /usr/lib/mesa-diverted to provide /usr/lib/mesa-diverted/libGL.so-master
Setting up glx-alternative-nvidia (1.2.2) ...
Processing triggers for glx-alternative-nvidia (1.2.2) ...
Setting up nvidia-alternative (535.183.01-1~deb12u1) ...
Processing triggers for nvidia-alternative (535.183.01-1~deb12u1) ...

```

```

update-alternatives: using /usr/lib/nvidia/current to provide /usr/lib/nvidia/nvidia (nvidia) in auto mode
Setting up libnvidia-cfg1:amd64 (535.183.01-1~deb12u1) ...
Setting up nvidia-kernel-support (535.183.01-1~deb12u1) ...
Setting up libnvidia-pkcs11-openssl3:amd64 (535.183.01-1~deb12u1) ...
Setting up libnvidia-ml1:amd64 (535.183.01-1~deb12u1) ...
Setting up libnvidia-ml-dev:amd64 (11.8.86~11.8.0-5~deb12u1) ...
Setting up nvidia-persistenced (535.171.04-1~deb12u1) ...
adduser: Warning: The home dir /var/run/nvpd/ you specified can't be accessed: No such file or directory
Adding system user `nvpd' (UID 117) ...
Adding new group `nvpd' (GID 128) ...
Adding new user `nvpd' (UID 117) with group `nvpd' ...
Not creating home directory `/var/run/nvpd/'.
Created symlink /etc/systemd/system/multi-user.target.wants/nvidia-persistenced.service → /lib/systemd/system/nvidia-persistenced.service.
Could not execute systemctl: at /usr/bin/deb-systemd-invoke line 145.
Setting up libcuda1:amd64 (535.183.01-1~deb12u1) ...
Setting up libcuinj64-11.8:amd64 (11.8.87~11.8.0-5~deb12u1) ...
Setting up nvidia-smi (535.183.01-1~deb12u1) ...
Setting up libnvcuvid1:amd64 (535.183.01-1~deb12u1) ...
Setting up nvidia-cuda-dev:amd64 (11.8.89~11.8.0-5~deb12u1) ...
Processing triggers for nvidia-alternative (535.183.01-1~deb12u1) ...
update-alternatives: updating alternative /usr/lib/nvidia/current because link group nvidia has changed
Setting up nvidia-kernel-dkms (535.183.01-1~deb12u1) ...
Loading new nvidia-current-535.183.01 DKMS files...
Building for 6.1.0-27-amd64
Building initial module for 6.1.0-27-amd64
Done.

nvidia-current.ko:
Running module version sanity check.
- Original module
  - No original module exists within this kernel
- Installation
  - Installing to /lib/modules/6.1.0-27-amd64/updates/dkms/

nvidia-current-modeset.ko:
Running module version sanity check.
- Original module
  - No original module exists within this kernel
- Installation
  - Installing to /lib/modules/6.1.0-27-amd64/updates/dkms/

nvidia-current-drm.ko:
Running module version sanity check.
- Original module
  - No original module exists within this kernel
- Installation
  - Installing to /lib/modules/6.1.0-27-amd64/updates/dkms/

```

```
nvidia-current-uvm.ko:  
Running module version sanity check.  
- Original module  
- No original module exists within this kernel  
- Installation  
- Installing to /lib/modules/6.1.0-27-amd64/updates/dkms/  
  
nvidia-current-peermem.ko:  
Running module version sanity check.  
- Original module  
- No original module exists within this kernel  
- Installation  
- Installing to /lib/modules/6.1.0-27-amd64/updates/dkms/  
depmod...  
Processing triggers for libc-bin (2.36-9+deb12u9) ...  
Processing triggers for initramfs-tools (0.142+deb12u1) ...  
update-initramfs: Generating /boot/initrd.img-6.1.0-27-amd64  
Processing triggers for update-glx (1.2.2) ...  
Processing triggers for glx-alternative-nvidia (1.2.2) ...  
update-alternatives: using /usr/lib/nvidia to provide /usr/lib/glx (glx) in auto mode  
Processing triggers for glx-alternative-mesa (1.2.2) ...  
Processing triggers for libc-bin (2.36-9+deb12u9) ...  
Processing triggers for initramfs-tools (0.142+deb12u1) ...  
update-initramfs: Generating /boot/initrd.img-6.1.0-27-amd64
```

CUDA Learn

- [NVIDIA CUDA documentation](#)
- [NVIDIA cuda-education](#)
- [cuda-samples](#)
- [CUDA Programming Course – High-Performance Computing with GPUs](#)
- [d_what_are_some_good_resources_to_learn_cuda](#)
- [What-are-some-of-the-best-resources-to-learn-CUDA-C](#)
- [cuda-training-series](#)
- [even-easier-introduction-cuda](#)
- [demystifying-gpu-architectures-for-deep-learning](#)
- [numba](#)
- [GPU-Puzzles](#)
- [CUDA 编程入门极简教程](#)
- [GPU 编程](#)
- [nvvm-ir-spec](#)

NVIDIA CUDA (Compute Unified Device Architecture)

The NVIDIA® CUDA® Toolkit provides a comprehensive development environment for C and C++ developers building GPU-accelerated applications. With the CUDA Toolkit, you can develop, optimize, and deploy your applications on GPU-accelerated embedded systems, desktop workstations, enterprise data centers, cloud-based platforms and HPC supercomputers. The toolkit includes GPU-accelerated libraries, debugging and optimization tools, a C/C++ compiler, and a runtime library to deploy your application.

NVIDIA® CUDA® 工具包为构建 GPU 加速应用程序的 C 和 C++ 开发人员提供了一个全面的开发环境。借助 CUDA 工具包，您可以在 GPU 加速的嵌入式系统、桌面工作站、企业数据中心、基于云的平台和 HPC 超级计算机上开发、优化和部署您的应用程序。该工具包包括 GPU 加速库、调试和优化工具、C/C++ 编译器以及用于部署应用程序的运行时库。

Using built-in capabilities for distributing computations across multi-GPU configurations, scientists and researchers can develop applications that scale from single GPU workstations to cloud installations with thousands of GPUs.

使用内置功能在多 GPU 配置之间分配计算，科学家和研究人员可以开发从单个 GPU 工作站扩展到具有数千个 GPU 的云安装的应用程序。

CUDA C++ Programming Guide v12.6

[CUDA C++ Programming Guide](#)

CUDA 编程模型的关键组成部分 (GPT)

CUDA 编程模型使开发者能够编写代码，充分利用 NVIDIA GPU 的强大并行计算能力。它基于单指令多线程（SIMT）架构，其中多个线程同时执行相同的指令，但处理不同的数据。CUDA 通过分层的线程结构和内存管理系统，高效组织计算任务。

1. 线程层次结构：

- 线程（Thread）：执行特定任务的最小执行单元。

- 线程块（Thread Block）：线程的集合，线程块中的线程共同执行任务。一个线程块最多包含 1024 个线程（具体取决于 GPU 架构）。
- 网格（Grid）：线程块的集合。网格可以是 1D、2D 或 3D，以便更方便地将线程映射到数据上。

通过唯一的索引（如 `threadIdx`、`blockIdx`、`blockDim` 和 `gridDim`），每个线程可以访问特定的数据部分。

2. 内存层次结构：

- 全局内存（Global Memory）：所有线程都可以访问，但访问延迟较高。
- 共享内存（Shared Memory）：线程块内的线程共享的一种快速、低延迟的内存。
- 局部内存（Local Memory）：每个线程的私有内存，但由于位于全局内存中，访问速度较慢。
- 寄存器（Registers）：速度极快，但数量有限，用于存储线程的临时变量。

3. 内核（Kernel）：

- CUDA 内核是运行在 GPU 上的函数，使用 C/C++ 语言编写并带有特殊的语法标记。内核从 CPU 发起，并由 GPU 的线程并行执行。

Introduction

The advent of multicore CPUs and manycore GPUs means that mainstream processor chips are now parallel systems.

The challenge is to develop application software that transparently scales its parallelism to leverage the increasing number of processor cores.

The CUDA parallel programming model is designed to overcome this challenge while maintaining a low learning curve for programmers familiar with C.

Its core is three key abstractions:

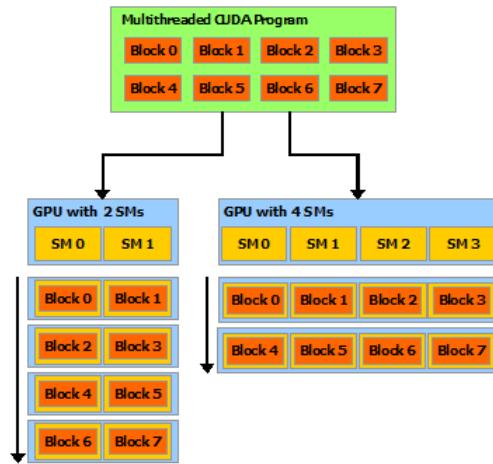
- a hierarchy of thread groups: 层级线程组
- shared memories: 共享内存
- barrier synchronization: 障碍同步

These abstractions provide fine-grained data parallelism and thread parallelism, nested within coarse-grained data parallelism and task parallelism. They guide the programmer to partition the problem into coarse sub-problems that can be solved independently in parallel by blocks of threads, and each sub-problem into finer pieces that can be solved cooperatively in parallel by all threads within the block.

这些抽象提供了细粒度数据并行性和线程并行性，嵌套在粗粒度数据并行性和任务并行性中。它们引导程序员将问题划分为可以由线程块独立并行解决的粗略子问题，并将每个子问题划分为可以由块内的所有线程并行协作解决的更精细的部分。

This decomposition preserves language expressivity by allowing threads to cooperate when solving each sub-problem, and at the same time enables automatic scalability. Indeed, each block of threads can be scheduled on any of the available multiprocessors within a GPU, in any order, concurrently or sequentially, so that a compiled CUDA program can execute on any number of multiprocessors as illustrated by Figure 3, and only the runtime system needs to know the physical multiprocessor count.

这种分解通过允许线程在解决每个子问题时进行合作来保留语言表达能力，同时实现自动可扩展性。事实上，每个线程块都可以以任何顺序（同时或顺序）调度到 GPU 内的任何可用多处理器上，以便编译后的 CUDA 程序可以在任意数量的多处理器上执行，如图 3 所示，并且仅运行时系统需要知道物理多处理器数量。



A GPU is built around an array of Streaming Multiprocessors (SMs)

GPU 由流式多处理器 (SM) 阵列构建

Programming Model 编程模型

- Kernels: 内核函数
- Thread Hierachy: 线程层次结构
- Memory Hierachy: 内存层次结构
- Heterogeneous Programming: 异构编程
- Asynchronous SIMT Programming Model: 异步 SIMT 编程模型
- Compute Capability: 计算能力

Kernels: 内核函数

CUDA C++ extends C++ by allowing the programmer to define C++ functions, called kernels, that, when called, are executed N times in parallel by N different CUDA threads, as opposed to only once like regular C++ functions.

CUDA C++ 通过允许程序员定义称为内核的 C++ 函数来扩展 C++, 这些函数在调用时由 N 个不同的 CUDA 线程并行执行 N 次，而不是像常规 C++ 函数那样只能执行一次。

A kernel is defined using the `__global__` declaration specifier and the number of CUDA threads that execute that kernel for a given kernel call is specified using a new `<<<...>>>` execution configuration syntax (see C++ Language Extensions). Each thread that executes the kernel is given a unique `thread ID` that is accessible within the kernel through built-in variables.

使用 `__global__` 声明说明符定义内核，并使用新的 `<<<...>>>` 执行配置语法指定为给定内核调用执行该内核的 CUDA 线程数（请参阅 C++ 语言扩展）。每个执行内核的线程都会被赋予一个唯一的线程 ID，该 ID 可以在内核中通过内置变量进行访问。

Thread Hierachy: 线程层次结构

- grids - blocks - threads
- 块内线程驻留在同一个 core 上，共享内存
- blocks, threads 由三维下标索引
 - `threadIdx.x, .y, .z`
 - `blockIdx.x, .y, .z`
- block 尺寸:

- blockDim.x, .y, .z

For convenience, threadIdx is a 3-component vector, so that threads can be identified using a one-dimensional, two-dimensional, or three-dimensional thread index, forming a one-dimensional, two-dimensional, or three-dimensional block of threads, called a thread block. This provides a natural way to invoke computation across the elements in a domain such as a vector, matrix, or volume.

The index of a thread and its thread ID relate to each other in a straightforward way:

- For a one-dimensional block, they are the same;
- for a two-dimensional block of size (Dx, Dy) , the thread ID of a thread of index (x, y) is $(x + y Dx)$;
- for a three-dimensional block of size (Dx, Dy, Dz) , the thread ID of a thread of index (x, y, z) is $(x + y Dx + z Dy)$.

There is a limit to the number of threads per block, since all threads of a block are expected to reside on the same streaming multiprocessor core and must share the limited memory resources of that core. On current GPUs, a thread block may contain up to 1024 threads.

每个块的线程数量是有限的，因为块的所有线程都应该驻留在同一个流式多处理器核心上，并且必须共享该核心的有限内存资源。在当前的 GPU 上，一个线程块最多可以包含 1024 个线程。

However, a kernel can be executed by multiple equally-shaped thread blocks, so that the total number of threads is equal to the number of threads per block times the number of blocks.

然而，一个内核可以由多个形状相同的线程块来执行，因此线程总数等于每个块的线程数乘以块数。

Blocks are organized into a one-dimensional, two-dimensional, or three-dimensional grid of thread blocks as illustrated by Figure 4. The number of thread blocks in a grid is usually dictated by the size of the data being processed, which typically exceeds the number of processors in the system.

块被组织成一维、二维或三维线程块网格，如图 4 所示。网格中线程块的数量通常由正在处理的数据大小决定，该数据通常超过系统中处理器的数量。

Extending the previous MatAdd() example to handle multiple blocks, the code becomes as follows.

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
float C[N][N]) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        C[i][j] = A[i][j] + B[i][j];
}

int main() {
    ...
    // Kernel invocation
    dim3 threadsPerBlock(16, 16);
    dim3 numBlocks(N / threadsPerBlock.x, N / threadsPerBlock.y);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

Threads within a block can cooperate by sharing data through some `shared memory` and by `synchronizing` their execution to coordinate memory accesses. More precisely, one can specify synchronization points in the kernel by calling the `__syncthreads()` intrinsic function;

`__syncthreads()` acts as a `barrier` at which all threads in the block must wait before any is allowed to proceed. In addition to `__syncthreads()`, the Cooperative Groups API provides a rich set of thread-synchronization primitives.

块内的线程可以通过某些共享内存共享数据并同步其执行来协调内存访问来进行协作。更准确地说，可以通过调用 `__syncthreads()` 内部函数来指定内核中的同步点；

`__syncthreads()` 充当屏障，块中的所有线程都必须等待，然后才允许任何线程继续进行。除了 `__syncthreads()` 之外，协作组 API 还提供了一组丰富的线程同步原语。

For efficient cooperation, the shared memory is expected to be a low-latency memory near each processor core (much like an L1 cache) and `__syncthreads()` is expected to be lightweight.

为了高效合作，共享内存应该是每个处理器核心附近的低延迟内存（很像 L1 缓存），并且 `__syncthreads()` 应该是轻量级的。

Memory Hierarchy: 内存层次结构

CUDA threads may access data from multiple memory spaces during their execution as illustrated by Figure 6. Each thread has private local memory. Each thread block has shared memory visible to all threads of the block and with the same lifetime as the block. Thread blocks in a thread block cluster can perform read, write, and atomics operations on each other's shared memory. All threads have access to the same global memory.

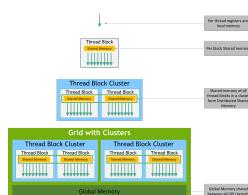
CUDA 线程在执行期间可以访问多个内存空间中的数据，如图 6 所示。每个线程都有私有本地内存。每个线程块都有对该块的所有线程可见的共享内存，并且与该块具有相同的生命周期。线程块簇中的线程块可以对彼此的共享内存执行读、写和原子操作。所有线程都可以访问相同的全局内存。

There are also two additional read-only memory spaces accessible by all threads: the constant and texture memory spaces. The global, constant, and texture memory spaces are optimized for different memory usages (see Device Memory Accesses). Texture memory also offers different addressing modes, as well as data filtering, for some specific data formats (see Texture and Surface Memory).

还有两个可供所有线程访问的附加只读内存空间：常量内存空间和纹理内存空间。全局、常量和纹理内存空间针对不同的内存使用进行了优化（请参阅设备内存访问）。纹理内存还为某些特定的数据格式提供不同的寻址模式以及数据过滤（请参阅纹理和表面内存）。

The global, constant, and texture memory spaces are persistent across kernel launches by the same application.

全局、常量和纹理内存空间在同一应用程序的内核启动过程中是持久的。



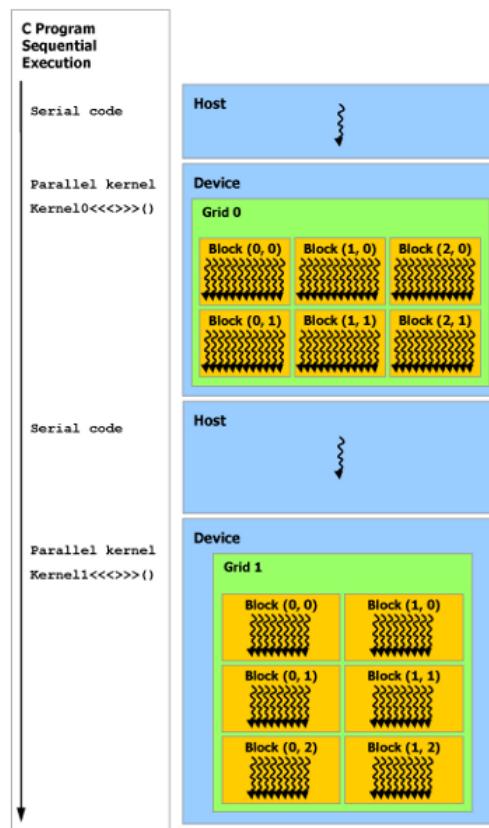
Heterogeneous Programming: 异构编程

As illustrated by Figure 7, the CUDA programming model assumes that the CUDA threads execute on a physically separate device that operates as a coprocessor to the host running the C++ program. This is the case, for example, when the kernels execute on a GPU and the rest of the C++ program executes on a CPU. 如图 7 所

示, CUDA 编程模型假设 CUDA 线程在物理上独立的设备上执行, 该设备作为运行 C++ 程序的主机的协处理器运行。例如, 当内核在 GPU 上执行而 C++ 程序的其余部分在 CPU 上执行时, 就会出现这种情况。

The CUDA programming model also assumes that both the host and the device maintain their own separate memory spaces in DRAM, referred to as host memory and device memory, respectively. Therefore, a program manages the global, constant, and texture memory spaces visible to kernels through calls to the CUDA runtime (described in Programming Interface). This includes device memory allocation and deallocation as well as data transfer between host and device memory. CUDA 编程模型还假设主机和设备都在 DRAM 中维护自己独立的内存空间, 分别称为主机内存和设备内存。因此, 程序通过调用 CUDA 运行时 (在编程接口中描述) 来管理内核可见的全局、常量和纹理内存空间。这包括设备内存分配和释放以及主机和设备内存之间的数据传输。

Unified Memory provides managed memory to bridge the host and device memory spaces. Managed memory is accessible from all CPUs and GPUs in the system as a single, coherent memory image with a common address space. This capability enables oversubscription of device memory and can greatly simplify the task of porting applications by eliminating the need to explicitly mirror data on host and device. See Unified Memory Programming for an introduction to Unified Memory. 统一内存提供托管内存来桥接主机和设备内存空间。托管内存可作为具有公共地址空间的单个一致内存映像从系统中的所有 CPU 和 GPU 进行访问。此功能可实现设备内存的超额订阅, 并且无需在主机和设备上显式镜像数据, 从而大大简化移植应用程序的任务。有关统一内存的介绍, 请参阅统一内存编程。



Serial code executes on the host while parallel code executes on the device. 串行代码在主机上执行, 而并行代码在设备上执行。

Asynchronous SIMT Programming Model: 异步 SIMT 编程模型

In the CUDA programming model a thread is the lowest level of abstraction for doing a computation or a memory operation. Starting with devices based on the NVIDIA Ampere GPU architecture, the CUDA programming model provides acceleration to memory operations via the asynchronous programming model. The asynchronous programming model defines the behavior of asynchronous operations with respect to CUDA threads.

在 CUDA 编程模型中，线程是执行计算或内存操作的最低抽象级别。从基于 NVIDIA Ampere GPU 架构的设备开始，CUDA 编程模型通过异步编程模型提供内存操作加速。异步编程模型定义了与 CUDA 线程相关的异步操作的行为。

The asynchronous programming model defines the behavior of `Asynchronous Barrier` for synchronization between CUDA threads. The model also explains and defines how `cuda::memcpy_async` can be used to move data asynchronously from global memory while computing in the GPU.

异步编程模型定义了用于 CUDA 线程之间同步的异步屏障的行为。该模型还解释并定义了如何使用 `cuda::memcpy_async` 在 GPU 中计算时从全局内存异步移动数据。

2.5.1. Asynchronous Operations

2.5.1.异步操作

An asynchronous operation is defined as an operation that is initiated by a CUDA thread and is executed asynchronously as-if by another thread. In a well formed program one or more CUDA threads synchronize with the asynchronous operation. The CUDA thread that initiated the asynchronous operation is not required to be among the synchronizing threads. 异步操作被定义为由 CUDA 线程发起并像由另一个线程一样异步执行的操作。在格式良好的程序中，一个或多个 CUDA 线程与异步操作同步。启动异步操作的 CUDA 线程不需要位于同步线程中。

Such an asynchronous thread (an as-if thread) is always associated with the CUDA thread that initiated the asynchronous operation. An asynchronous operation uses a synchronization object to synchronize the completion of the operation. Such a synchronization object can be explicitly managed by a user (e.g., `cuda::memcpy_async`) or implicitly managed within a library (e.g., `cooperative_groups::memcpy_async`)。这样的异步线程（as-if 线程）始终与启动异步操作的 CUDA 线程相关联。异步操作使用同步对象来同步操作的完成。这样的同步对象可以由用户显式管理（例如，`cuda::memcpy_async`）或在库中隐式管理（例如，`cooperative_groups::memcpy_async`）。

A synchronization object could be a `cuda::barrier` or a `cuda::pipeline`. These objects are explained in detail in Asynchronous Barrier and Asynchronous Data Copies using `cuda::pipeline`. These synchronization objects can be used at different thread scopes. A scope defines the set of threads that may use the synchronization object to synchronize with the asynchronous operation. The following table defines the thread scopes available in CUDA C++ and the threads that can be synchronized with each. 同步对象可以是 `cuda::barrier` 或 `cuda::pipeline`。这些对象在使用 `cuda::pipeline` 的异步屏障和异步数据副本中详细解释。这些同步对象可以在不同的线程范围内使用。范围定义了可以使用同步对象来与异步操作同步的线程集。下表定义了 CUDA C++ 中可用的线程范围以及可以与每个线程同步的线程。

Compute Capability: 计算能力

Programming Interface 编程接口

Hardware Implementation 硬件实现

Performance Guidelines 性能指南

PTX Parallel Thread Execution

PTX: a low-level parallel thread execution virtual machine and instruction set architecture.

PTX 是一种低级并行线程执行虚拟机和指令集体系结构。

PTX exposes the GPU as data-parallel computing device.

Numba

Overview □ 概述 □

Numba supports CUDA GPU programming by directly compiling a restricted subset of Python code into CUDA kernels and device functions following the CUDA execution model. Kernels written in Numba appear to have direct access to NumPy arrays. NumPy arrays are transferred between the CPU and the GPU automatically.

Numba 通过将 Python 代码的受限子集直接编译为遵循 CUDA 执行模型的 CUDA 内核和设备函数来支持 CUDA GPU 编程。用 Numba 编写的内核似乎可以直接访问 NumPy 数组。NumPy 数组在 CPU 和 GPU 之间自动传输。

Install CUDA

[CUDA Toolkit Archive](#)

[NVIDIA CUDA Installation Guide for Linux](#)

Other Resources

Programming Massively Parallel Processors: A Hands-on Approach

Programming Massively Parallel Processors: A Hands-on Approach, Second Edition, teaches students how to program massively parallel processors. It offers a detailed discussion of various techniques for constructing parallel programs. Case studies are used to demonstrate the development process, which begins with computational thinking and ends with effective and efficient parallel programs. This guide shows both student and professional alike the basic concepts of parallel programming and GPU architecture. Topics of performance, floating-point format, parallel patterns, and dynamic parallelism are covered in depth. This revised edition contains more parallel programming examples, commonly-used libraries such as Thrust, and explanations of the latest tools. It also provides new coverage of CUDA 5.0, improved performance, enhanced development tools,

increased hardware support, and more; increased coverage of related technology, OpenCL and new material on algorithm patterns, GPU clusters, host programming, and data parallelism; and two new case studies (on MRI reconstruction and molecular visualization) that explore the latest applications of CUDA and GPUs for scientific research and high-performance computing. This book should be a valuable resource for advanced students, software engineers, programmers, and hardware engineers.

Programming Massively Parallel Processors: A Hands-on Approach, 第二版, 教授学生如何对大规模并行处理器进行编程。它详细讨论了用于构建并行程序的各种技术。案例研究用于演示开发过程, 该过程从计算思维开始, 以有效和高效的并行程序结束。本指南向学生和专业人士展示了并行编程和 GPU 架构的基本概念。深入介绍了性能、浮点格式、并行模式和动态并行性等主题。此修订版包含更多并行编程示例、常用库（如 Thrust）以及最新工具的解释。它还提供了 CUDA 5.0 的新覆盖范围、改进的性能、增强的开发工具、增强的硬件支持等;增加了相关技术、OpenCL 和有关算法模式、GPU 集群、主机编程和数据并行性的新材料的覆盖范围;以及两个新的案例研究（关于 MRI 重建和分子可视化），探索 CUDA 和 GPU 在科学的研究和高性能计算中的最新应用。这本书应该是高级学生、软件工程师、程序员和硬件工程师的宝贵资源。

CUDA Examples

```

#include <cuda_runtime.h>
#include <iostream>

// CUDA kernel function for vector addition
__global__ void vectorAdd(const float* A, const float* B, float* C, int N) {
    int i = blockIdx.x * blockDim.x + threadIdx.x; // Calculate global thread index
    if (i < N) {
        C[i] = A[i] + B[i];
    }
}

int main() {
    int N = 1024;
    size_t size = N * sizeof(float);

    // Allocate host memory
    float *h_A = (float*)malloc(size);
    float *h_B = (float*)malloc(size);
    float *h_C = (float*)malloc(size);

    // Initialize vectors
    for (int i = 0; i < N; i++) {
        h_A[i] = static_cast<float>(i);
        h_B[i] = static_cast<float>(i * 2);
    }

    // Allocate device memory
    float *d_A, *d_B, *d_C;
    cudaMalloc((void**)&d_A, size);
    cudaMalloc((void**)&d_B, size);
    cudaMalloc((void**)&d_C, size);

    // Copy data from host to device
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    // Launch kernel
    int threadsPerBlock = 256;
    int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;
    vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);

    // Copy result back to host
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

    // Print some results
    for (int i = 0; i < 10; i++) {

```

```
    std::cout << h_C[i] << std::endl;
}

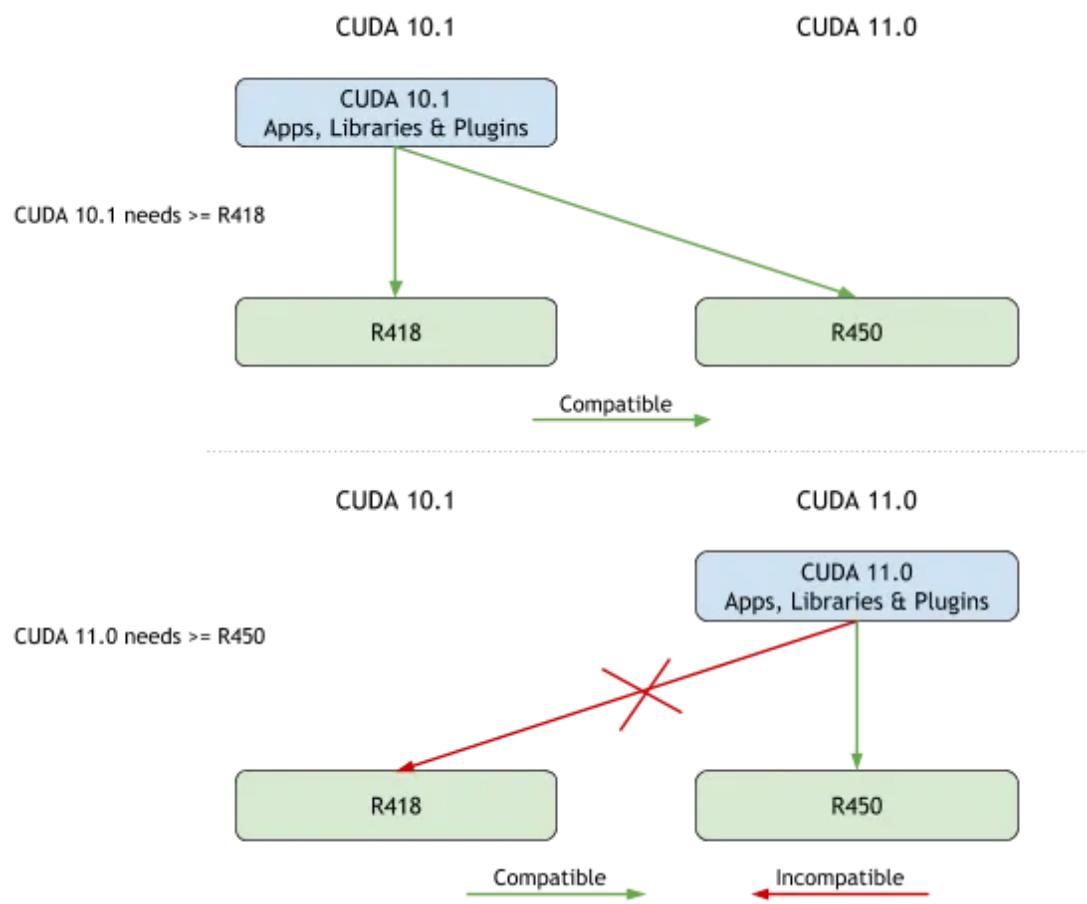
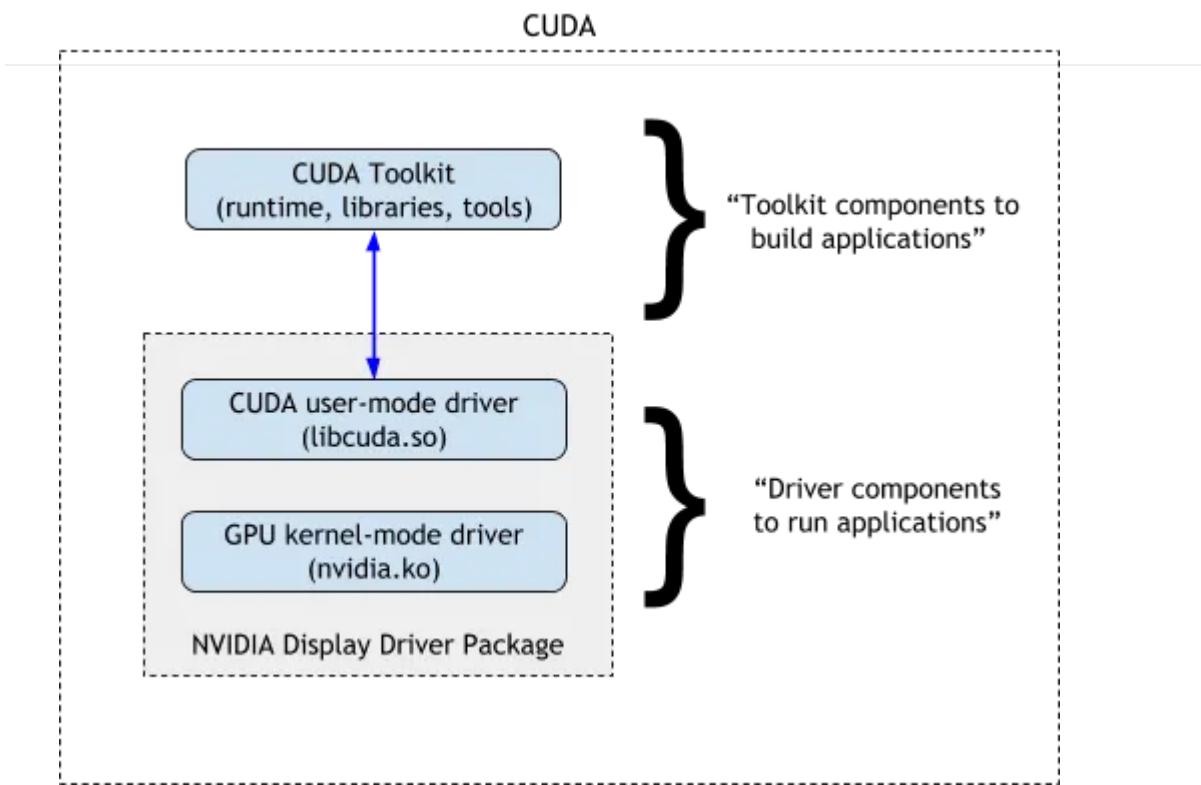
// Free memory
free(h_A); free(h_B); free(h_C);
cudaFree(d_A); cudaFree(d_B); cudaFree(d_C);

return 0;
}
```

CUDA Multi Version

Installing multiple CUDA + cuDNN versions in the same machine for Tensorflow and Pytorch

[cuda-compatibility](#)



```
# add below to your env bash file.

function _switch_cuda {
    v=$1
    export PATH=$PATH:/usr/local/cuda-$v/bin
    export CUDADIR=/usr/local/cuda-$v
    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-$v/lib64
    nvcc --version
}
```

And call this function to switch to a corresponding cuda version on your bash session

```
_switch_cuda 11.0 # change the version of your like to load bash.
```

Multiple Version of CUDA Libraries On The Same Machine

```
sudo sh cuda-9.1.run --silent --toolkit --toolkitpath=/usr/local/cuda-9.1
```

Managing Multiple CUDA Versions on a Single Machine: A Comprehensive Guide

```
export PATH=/usr/local/cuda-11.8/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-11.8/lib64:$LD_LIBRARY_PATH

# Activate the virtual environment
echo "export PATH=/usr/local/cuda-11.8/bin:$PATH" >> venv/my_env/bin/activate
echo "LD_LIBRARY_PATH=/usr/local/cuda-11.8/lib64:$LD_LIBRARY_PATH" >> venv/my_env/bin/activate
```

<https://developer.nvidia.com/nccl>

NVIDIA Collective Communications Library (NCCL)

NVIDIA NCCL 英伟达 NCCL

The NVIDIA Collective Communication Library (NCCL) implements multi-GPU and multi-node communication primitives optimized for NVIDIA GPUs and Networking. NCCL provides routines such as all-gather, all-reduce, broadcast, reduce, reduce-scatter as well as point-to-point send and receive that are optimized to achieve high bandwidth and low latency over PCIe and NVLink high-speed interconnects within a node and over NVIDIA Mellanox Network across nodes.

NVIDIA 集体通信库 (NCCL) 实现了针对 NVIDIA GPU 和网络进行优化的多 GPU 和多节点通信原语。NCCL 提供全收集、全归约、广播、归约、归约分散以及点对点发送和接收等例程，这些例程经过优化，可在 PCIe 和 NVLink 高速互连上实现高带宽和低延迟。一个节点以及跨节点的 NVIDIA Mellanox 网络。

Tensor Cores

CUDA Toolkit Documentation 12.2 (nvidia.com)

NVIDIA 英伟达 Tensor Core 基本原理(上)【AI 芯片】GPU 架构 04_哔哩哔哩_bilibili

Volta 架构第一代 Tensor Core

CUDA Core:

- GPU 并行模式实现深度学习功能过于通用，最常见 Conv/GEMM 操作，依旧要被编码成 FMA，硬件层面还是需要把数据按：寄存器-ALU-寄存器-ALU-寄存器，方式来回搬运。

Tensor Core:

- V100 Tensor Core 提供可编程矩阵乘法和累加单元 (matrix-multiply-and-accumulate units)，可为 AI 训练和推理提供 125 Tensor TFLOPS 算力。V100 包含 64 个 Tensor 内核：每个 SM 8 个。



Tensor Core:

专门做矩阵乘，矩阵加，可以在一步中执行混合精度的矩阵乘法和累积。这意味着它们可以乘以两个矩阵，然后将第三个矩阵添加到结果中，这是深度学习中的基本操作，速度非常快。

GEMM 通用矩阵乘 General Matrix Multiplication

FMA 乘积累加指令 Fused Multiply–accumulate operation

Volta 架构第一代 Tensor Core

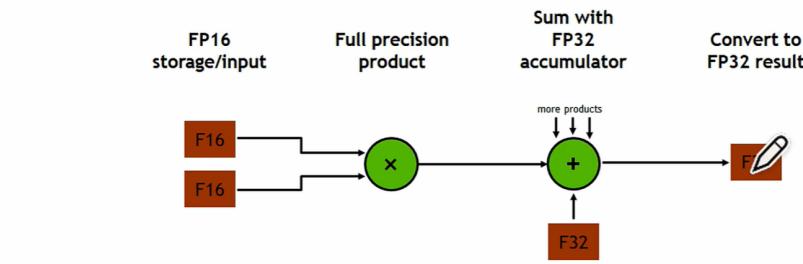
- 每个 Tensor Core 每周期能执行 $4 \times 4 \times 4$ GEMM，64 个 FMA。执行运算 $D = A * B + C$ ，其中 A、B、C 和 D 是 4×4 矩阵。矩阵乘法输入 A 和 B 是 FP16 矩阵，而累加矩阵 C 和 D 可以是 FP16 或 FP32 矩阵。
- Tensor Core 执行融合乘法加法，其中两个 4×4 FP16 矩阵相乘，然后将结果添加到 4×4 FP16 或 FP32 矩阵中，最终输出新的 4×4 FP16 或 FP32 矩阵。

$$D = \left(\begin{array}{cccc} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{array} \right) \left(\begin{array}{cccc} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{array} \right) + \left(\begin{array}{cccc} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{array} \right)$$

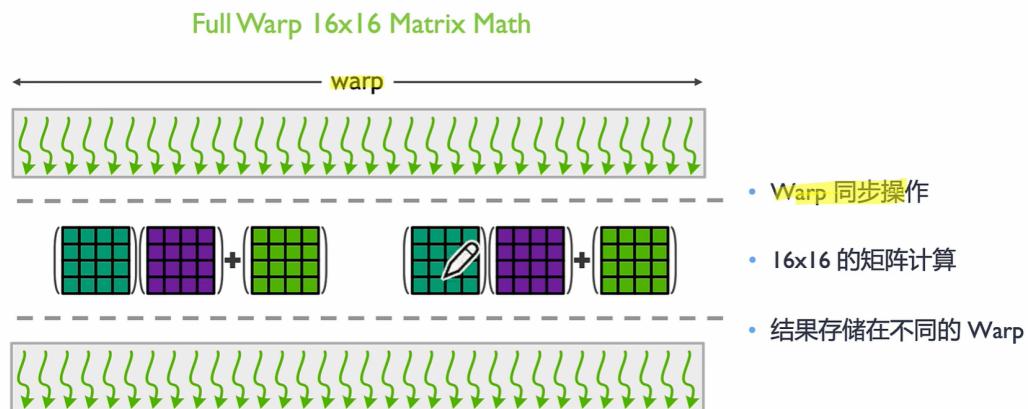
FP16 or FP32 FP16 FP16 FP16 or FP32

Volta 架构第一代 Tensor Core

- 每个 Tensor Core 每个时钟执行 64 个 FP32 FMA 混合精度运算，SM 中 8 个 Tensor Core，每个时钟周期内总共执行 512 个浮点运算。
- 因此在 AI 应用中，Volta V100 GPU 的吞吐量与 Pascal P100 GPU 相比，每个 SM 的 AI 吞吐量增加了 8 倍，总共增加了 12 倍。



张量同步的方式



CUDA Tensor Core编程

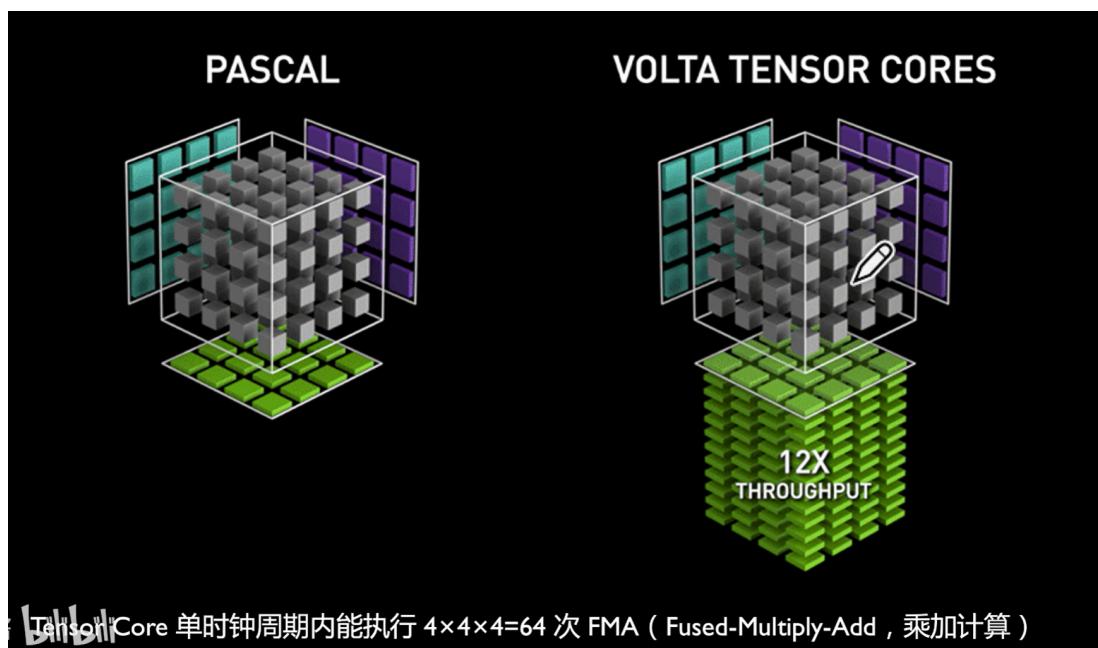
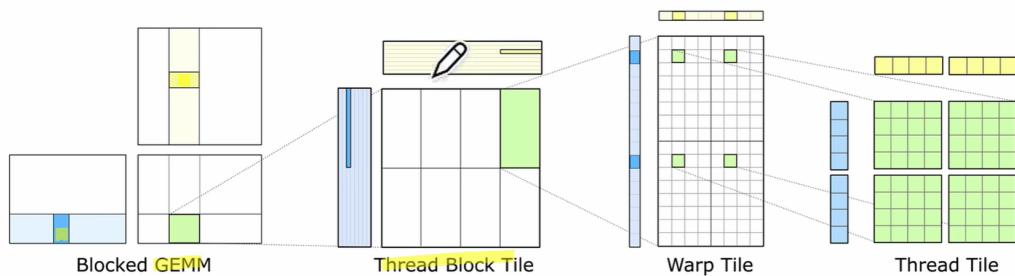
16x16x16 Warp Matrix Multiply and Accumulate(WMMA)

```
wmma::mma_sync(Dmat, Amat, Bmat, Cmat);
```

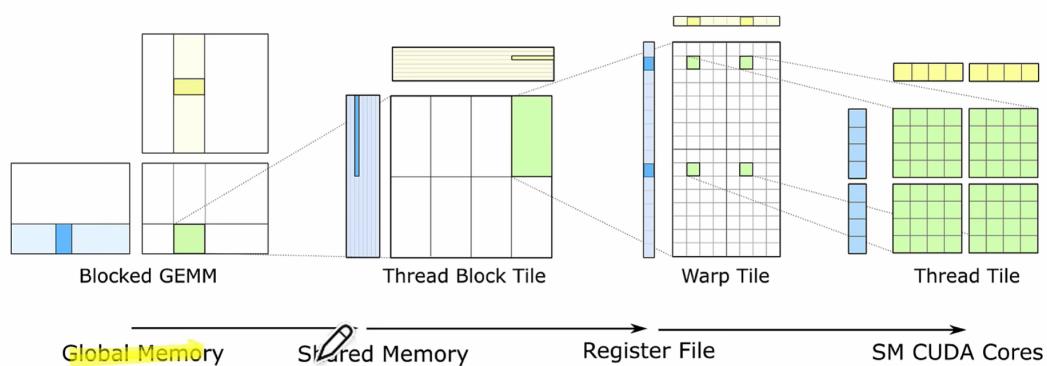
$$D = \left(\begin{array}{c|c} \text{FP16 or FP32} & \text{FP16} \end{array} \right) \left(\begin{array}{c|c} \text{FP16} & \text{FP16} \end{array} \right) + \left(\begin{array}{c|c} \text{FP16 or FP32} & \text{FP16} \end{array} \right)$$

CNN vs GEMM

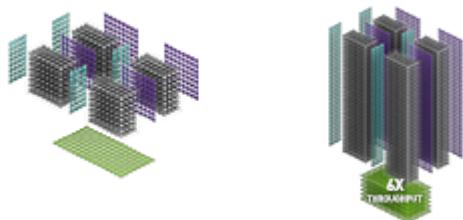
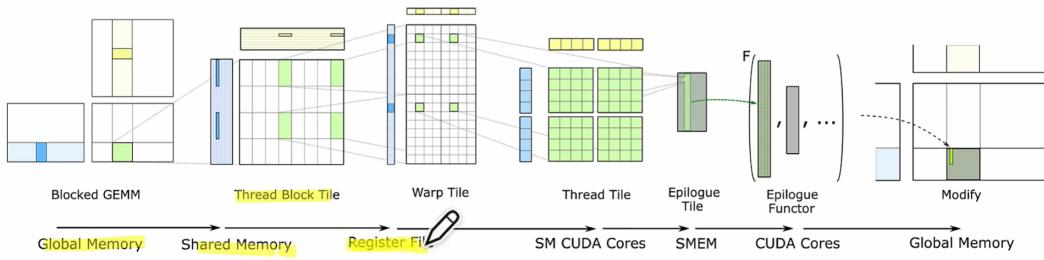
- 卷积的计算可以转换为两个矩阵相乘的求解，得到最终的卷积计算结果。
- GEMM 计算可以被成批地放在一起，作为单个大型矩阵乘法运算运行。
- 在 GPU CUDA Core 中，通过线程 Block 提供具体的计算。



GEMM软硬件分层：每一层都进行数据复用



完整 GEMM 计算/存储数据流



CUDA 核心和 Tensor 核心

CUDA 核心和 Tensor 核心都是 NVIDIA GPU 中的架构组件，但它们是为不同的目的设计的并以不同的方式操作。以下是它们之间的差异：

1. 目的：

- CUDA 核心：CUDA 核心有时被称为“流式多处理器”或“着色器核心”，它们是通用的并且处理各种任务，包括顶点着色、像素着色、几何着色和计算任务。它们是负责传统 GPU 任务的主要组件。
- Tensor 核心：Tensor 核心是 NVIDIA 架构的较新添加，与 Volta 代一起引入。它们是专门为加速深度学习工作负载而设计的硬件单元，特别是在神经网络训练和推断中常见的矩阵操作。

2. 操作：

- CUDA 核心：CUDA 核心可以处理浮点算术、整数算术和其他传统的 GPU 任务。
- Tensor 核心：Tensor 核心可以在一步中执行混合精度的矩阵乘法和累积。这意味着它们可以乘以两个矩阵，然后将第三个矩阵添加到结果中，这是深度学习中的基本操作，速度非常快。

3. 精度：

- CUDA 核心：支持从高精度（如 FP64）到较低精度（如 FP32）的广泛精度。
- Tensor 核心：主要针对混合精度计算进行了优化，通常使用 FP16（半精度）进行乘法和 FP32（单精度）进行累积。这种混合精度方法在深度学习中是有益的，因为许多模型可以容忍降低的精度，而不会对准确性产生重大影响。

4. 在 GPU 架构中的存在：

-
- CUDA 核心：自 2007 年引入 CUDA 平台以来，CUDA 核心一直是 NVIDIA GPU 的主要组成部分。
 - Tensor 核心：它们从 Volta 架构开始引入（例如在 Tesla V100 中），并且在随后的架构中（如 Turing、Ampere 等）都存在。

5. 性能：

- CUDA 核心：通用，可以有效地处理各种任务。
- Tensor 核心：在与深度学习有关的特定矩阵操作方面，与 CUDA 核心相比，Tensor 核心提供了明显更快的性能。

总之，虽然 CUDA 核心和 Tensor 核心都是 NVIDIA 的 GPU 架构的重要部分，但 CUDA 核心是处理各种任务的通用核心，而 Tensor 核心是针对深度学习矩阵操作优化的专用单元。当购买 NVIDIA 硬件时，应根据预期的主要用途（例如，游戏、图形、通用计算、深度学习）考虑 CUDA 核心和 Tensor 核心之间的平衡。

Computer Vision

- Image Classification
- Object Detection: bounding box
- Image Segmentation: pixel-wise classification, cutting edges

opencv-python

`haarcascade_frontalface_default.xml` : Trained XML classifiers describes some features of some object we want to detect a cascade function is trained from a lot of positive(faces) and negative(non-faces) images.

Image

- Read an image from file (using `cv::imread`)
- Display an image in an OpenCV window (using `cv::imshow`)
- Write an image to a file (using `cv::imwrite`)
- `IMREAD_COLOR` loads the image in the BGR 8-bit format. This is the default that is used here.
- `IMREAD_UNCHANGED` loads the image as is (including the alpha channel if present)
- `IMREAD_GRAYSCALE` loads the image as an intensity one
- `core` section, as here are defined the basic building blocks of the library
- `imgcodecs` module, which provides functions for reading and writing
- `highgui` module, as this contains the functions to show an image in a window

Video

`VideoCapture()`

`VideoWriter()`

Drawing

- Learn to draw different geometric shapes with OpenCV
- You will learn these functions :
 - `cv.line()`, `cv.circle()` , `cv.rectangle()`, `cv.ellipse()`, `cv.putText()` etc.

`img, color, thickness, lineType`

Mouse

Learn to handle mouse events in OpenCV You will learn these functions : `cv.setMouseCallback()`

```
'EVENT_FLAG_ALTKEY', 'EVENT_FLAG_CTRLKEY', 'EVENT_FLAG_LBUTTON',
'EVENT_FLAG_MBUTTON', 'EVENT_FLAG_RBUTTON', 'EVENT_FLAG_SHIFTKEY',
'EVENT_LBUTTONDOWNDBLCLK', 'EVENT_LBUTTONDOWN', 'EVENT_LBUTTONUP',
'EVENT_MBUTTONDOWNDBLCLK', 'EVENT_MBUTTONDOWN', 'EVENT_MBUTTONUP',
'EVENT_MOUSEHWHEEL', 'EVENT_MOUSEMOVE', 'EVENT_MOUSEWHEEL',
'EVENT_RBUTTONDOWNDBLCLK', 'EVENT_RBUTTONDOWN', 'EVENT_RBUTTONUP']
```

Trackbar

For `cv.createTrackbar()` function, first argument is the trackbar name, second one is the window name to which it is attached, third argument is the default value, fourth one is the maximum value and fifth one is the callback function which is executed every time trackbar value changes. The callback function always has a default argument which is the trackbar position. In our case, function does nothing, so we simply pass.

Another important application of trackbar is to use it as a button or switch. OpenCV, by default, doesn't have button functionality. So you can use trackbar to get such functionality. In our application, we have created one switch in which application works only if switch is ON, otherwise screen is always black.

Operations on Images

- Basic Operations on Images
 - Learn to read and edit pixel values, working with image ROI and other basic operations.
- Arithmetic Operations on Images
 - Perform arithmetic operations on images
- Performance Measurement and Improvement Techniques
 - Getting a solution is important. But getting it in the fastest way is more important. Learn to check the speed of your code, optimize the code etc.

Image Processing

Feature Detection and Description

Video Analysis

Change the Default Base Environment in Conda

Problem:

on linux (debian12), how to change the default 'base' env in conda? current 'base' env use python12, i wnat use python11 as my default base env fo conda, how to do that?

Solution:

To change the default `base` environment in Conda to use Python 3.11 instead of Python 3.12 on Debian 12, follow these steps:

Step 1: Check Your Current `base` Environment

1. Activate the `base` environment:

```
conda activate base
```

2. Check the Python version:

```
python --version
```

If it's showing Python 3.12, you'll need to update it.

Step 2: Install Python 3.11 in the `base` Environment

1. Update Conda to ensure you have the latest packages:

```
conda update conda  
conda update --all
```

2. Replace the Python version in the `base` environment:

```
conda install python=3.11
```

This will resolve dependencies and replace Python 3.12 with Python 3.11 in the `base` environment.

Step 3: Verify the Change

1. Confirm the Python version after installation:

```
python --version
```

It should now show Python 3.11.

2. Deactivate and reactivate `base` to ensure changes persist:

```
conda deactivate  
conda activate base
```

Step 4: Set Conda to Always Use the `base` Environment on Start (Optional)

If you always want to activate the `base` environment when opening a terminal:

1. Edit your shell initialization file (e.g., `.bashrc` or `.zshrc`):

```
nano ~/.bashrc
```

2. Add this line at the end:

```
conda activate base
```

3. Save and reload your shell:

```
source ~/.bashrc
```

Now, your Conda `base` environment will use Python 3.11 by default!

pypi: <https://pypi.tuna.tsinghua.edu.cn/simple>

<https://mirrors.tuna.tsinghua.edu.cn/help/pypi/>

anaconda:

<https://mirrors.tuna.tsinghua.edu.cn/help/anaconda/>

```
conda config --set custom_channels.auto https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/
```

```

# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/debian \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# Install Docker Engine:
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

```

<https://docs.docker.com/desktop/install/debian/>

```

docker stop $(docker ps -a -q)
docker image pull name:tag

docker search image_name
docker pull image_name:tag
docker images # list all images
docker rmi image_name:tag # remove image
docker run -it image_name:tag /bin/bash # run container

# start docker desktop
systemctl --user start docker-desktop
# start on sign in
systemctl --user enable docker-desktop
systemctl --user stop docker-desktop

```

<https://medium.com/@SrvZ/docker-proxy-and-my-struggles-a4fd6de21861>

Docker Compose Docker 组合

Docker Compose is a tool for defining and running multi-container applications. It is the key to unlocking a streamlined and efficient development and deployment experience.

Docker Compose 是一个用于定义和运行多容器应用程序的工具。它是解锁精简高效的开发和部署体验的关键。

Compose simplifies the control of your entire application stack, making it easy to manage services, networks, and volumes in a single, comprehensible YAML configuration file. Then, with a single command, you create and start all the services from your configuration file.

Compose 简化了对整个应用程序堆栈的控制，使您可以在单个易于理解的 YAML 配置文件中轻松管理服务、网络和卷。然后，使用单个命令，您可以从配置文件创建并启动所有服务。

Compose works in all environments; production, staging, development, testing, as well as CI workflows. It also has commands for managing the whole lifecycle of your application:

Compose 适用于所有环境；生产、登台、开发、测试以及 CI 工作流程。它还具有用于管理应用程序整个生命周期的命令：

- Start, stop, and rebuild services
 - 启动、停止和重建服务
- View the status of running services
 - 查看正在运行的服务的状态
- Stream the log output of running services
 - 流式传输正在运行的服务的日志输出
- Run a one-off command on a service
 - 在服务上运行一次性命令

Ceres-Solver

[doc](#)

Install (on debian12)

```
# CMake
sudo apt-get install cmake

# google-glog + gflags
# libgoogle-glog-dev: library that implements application-level logging
# libgflags-dev: commandline flags module for C++
sudo apt-get install libgoogle-glog-dev libgflags-dev

# Use ATLAS for BLAS & LAPACK
# libatlas-base-dev: Automatically Tuned Linear Algebra Software, generic static
sudo apt-get install libatlas-base-dev

# Eigen3
# libeigen3-dev: lightweight C++ template library for linear algebra
sudo apt-get install libeigen3-dev

# SuiteSparse (optional)
# libsuitesparse-dev: libraries for sparse matrices computations (development files)
sudo apt-get install libsuitesparse-dev # to slow, skip it
```

```
# FindTBB.cmake
file(STRINGS
    "${TBB_INCLUDE_DIR}/tbb/version.h" # tbb_stddef.h -> version.h
    # https://github.com/ceres-solver/ceres-solver/issues/1036
    TBB_VERSION_CONTENTS
    REGEX "VERSION")
```

```
tar zxf ceres-solver-2.2.0.tar.gz
mkdir ceres-bin
cd ceres-bin
cmake ../ceres-solver-2.2.0
make -j3
make test
# Optionally install Ceres, it can also be exported using CMake which
# allows Ceres to be used without requiring installation, see the documentation
# for the EXPORT_BUILD_DIR option for more information.
make install
```

Eigen

- [eigen](#)
- [gitlab-eigen](#)

Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.

Eigen 是一个用于线性代数的 C++ 模板库：矩阵、向量、数值求解器和相关算法。

Fast-Drone 250

eigen error: no match for ‘operator=’

Important:

- lower eigen from 3.4.0 to 3.3.7
- lower liblog4cxx from 1.0.0 to 0.10.0

dependent ros package (noetic):

- control_toolbox - melodic-devel
- ddynamic_reconfigure - kinetic-devel
- geographic_info - master
- geometry2 - noetic-devel
- mavlink-gbp-release - release/noetic/mavlink
- mavros - master
- pcl_msgs - noetic-devel
- perception_pcl - melodic-devel
- realtime_tools - melodic-devel
- unique_identifier - master

in your catkin workspace:

```
catkin_make_isolated --install \
-DCMAKE_BUILD_TYPE=Release \
-DPYTHON_EXECUTABLE=/usr/bin/python3
```

Learn Drones

- [HKUST-Aerial-Robotics](#)
- [ZJU-FAST-Lab](#)
- [Fast-Drone-250](#)
- [ego-planner-swarm](#)
- [ego-planner](#)
- [multi_uav_simulator](#)

Tools

- RealSense Driver
 - [github](#)
 - [source installation](#)
 - The shared object will be installed in `/usr/local/lib`, header files in `/usr/local/include`.
 - `sudo apt install libpcl-dev`
- Mavros: Micro Air Vehicle Robot Operating System
 - **MAVLink** extendable communication node for ROS with proxy for **Ground Control Station**.
 - This package provides communication driver for various autopilots with MAVLink communication protocol. Additional it provides UDP MAVLink bridge for ground control stations (e.g. QGroundControl).
 - mavros用于无人机通信，可以将飞控与主控的信息进行交换
- [ceres-solver](#):
 - [googlesource](#)
 - An open source lib for modeling and solving large, complicated optimization problems.
 - It can be used to solve Non-linear Least Squares problems with bounds constraints and general unconstrained optimization problems. 它可用于求解具有边界约束的非线性最小二乘法问题和一般的无约束优化问题。
- [glog](#)
 - C++ implementation of the Google logging module.

```
pip uninstall em
pip install empy
```

```
# install ros noetic dependencies
# [ddynamic_reconfigure](https://github.com/pal-robotics/ddynamic_reconfigure)
catkin_make_isolated --install -DPYTHON_EXECUTABLE=/usr/bin/python3 --force-cmake

# must be: dont be higher than 0.10.0
Package: liblog4cxx-dev
Version: 0.10.0-15ubuntu2
```

在稠密环境中的分布式集群路径规划优化-2022

Distributed swarm trajectory optimization for formation flight in dense environments

针对空中编队，预定形式导航在各种场景中被广泛应用。然而，相关的规划策略通常缺乏在繁杂环境中避免障碍物的能力。为了解决这个问题，我们提出了一种基于优化的方法，确保编队飞行的轨迹生成不会发生碰撞。本文中，我们提出了一种新颖的 可微度量 来衡量编队之间的 整体相似距离 。然后，我们将这个度量转化为一个优化框架，通过多项式轨迹来实现 空间-时间规划 。在该框架中还加入了避免碰撞的最小化惩罚，以便同时处理编队保持和障碍物避让。为了验证我们方法的效率，我们与其他前沿作品进行了基准比较。结合自主分布式空中编队系统，我们提出的方法在富有障碍物的实际环境实验中展示了其高效性和稳健性。我们将发布源代码供社区参考。

EGO-Swarm-21

本文提出了一种去中心化的、异步的系统的解决方案，用于仅利用机载资源在障碍物丰富的未知场景中实现多机器人自主导航。规划系统是在基于梯度的局部规划框架下制定的，通过将碰撞风险表述为非线性优化问题的惩罚来避免碰撞。为了提高鲁棒性并摆脱局部最小值，我们采用了一种轻量级拓扑轨迹生成方法。然后，智能体（Agent）利用不可靠的轨迹共享网络，在几毫秒内生成安全、平滑和动态可行的轨迹。通过使用深度图像中的智能体检测来纠正智能体之间的相对定位漂移。我们的方法在模拟和实际实验中都得到了验证。源代码已发布，供社区参考。

- nontriviality of obstacle parameterization 障碍物参数化的复杂性
- limited sensing range 有限的感知范围
- unreliable and bandwidth limited communication 不可靠和带宽有限的通信
- positioning drift caused by inconsistent localization 不一致定位引起的位置漂移

拓扑规划和互动避障

1 Introduction

四旋翼机的灵活性使得该机器能够在未知环境中进行单一机器自主导航，并在开放或已知领域中进行多机精确编队控制。然而，很少有工作同时结合这两者来展示能够在共享相同未知空间的四旋翼机群体中导航的真实系统，尤其是仅通过机载处理。部署多个四旋翼机在未知环境中存在的困难包括但不限于障碍物参数化的复杂性、有限的感知范围、不可靠和带宽有限的通信以及由不一致定位引起的位置漂移。一些相关工作，例如，推进了真实的四旋翼机群体能够避障的能力。然而，上述困难在大多数之前的工作中经常被忽略，在运动捕捉系统或纯粹模拟中，限制了它们算法在实践中的应用。

本文提出了一种系统性的解决方案，能够在存在上述困难的拥挤环境中对四旋翼机群体进行高性能行进。此外，它不需要外部定位和计算或预先构建的地图。所提出的系统名为\$\\textbf{EGO-Swarm}\$\$，是我们先前工作 *ESDF-free Gradient-based Local planner (EGO-Planner)* 的扩展，为未知环境中单一四旋翼机的机载局部规划奠定了坚实的基础。该扩展包括两个部分，拓扑规划 和 互动避障。

如图 3 所示，非凸配置空间可能会导致不希望的行为，例如动力学不可行性或拥挤导航。因此，像拓扑规划这样的策略可以有助于摆脱局部最小值。基于 EGO-Planner 中的碰撞成本公式，前端拓扑路径搜索是隐含完成的，因此几乎不需要计算。

通过将群体碰撞的加权惩罚添加到目标函数中，实现了分散的互动避障。这个惩罚是通过将未来一段时间内的智能体分布与正在优化的轨迹进行比较来评估的。为了最小化数据传输并允许不可靠的通信，使用广播网络共享轨迹。而校正相对定位漂移（可以增加到半米），则是通过比较目击智能体的观测和轨迹评估的预测来进行。

- 拓扑规划：topological planning
- 互动避障：reciprocal collision avoidance

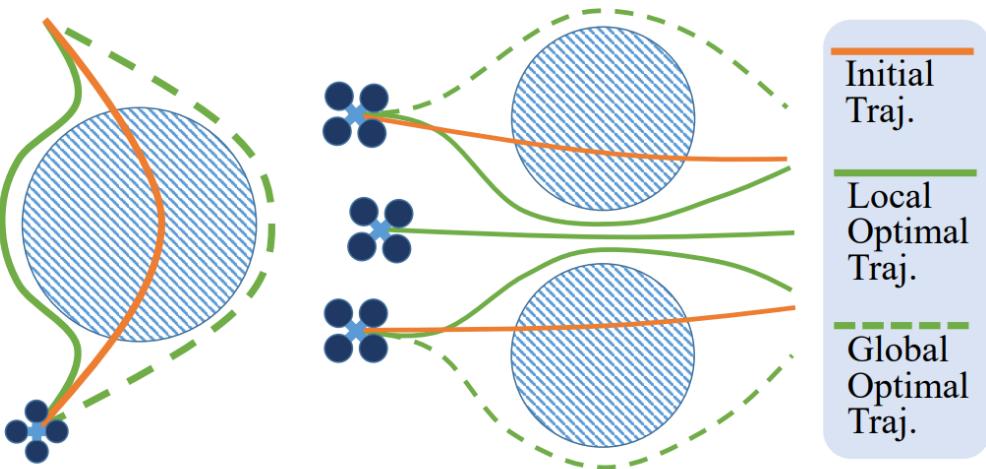


Fig. 3: 左侧:局部最优轨迹的动力学不可行性。右侧:三个机体挤满飞行通道。

2 Related Works

A. Single Quadrotor Local Planning 单四旋翼飞行器局部规划

基于梯度的运动规划 是四旋翼机器人局部规划的主流方法。基于开创性的研究 [4, 5]，这些研究将 局部规划问题 形式化为 无约束的非线性优化问题，提出了一系列的工作 [6]–[10]。这些工作考虑了轨迹的平滑性、可行性和安全性，采用了多种参数化方法，包括多项式和 B 样条曲线。最近我们提出了一种名为 EGO-Planner 的单四旋翼导航系统 [3]，通过采用更紧凑的环境表示方法，进一步减少了计算时间。本文的研究即基于此工作。

[基于梯度的运动规划](#)

[EGO-Planner-20](#)

B. Topological Planning 拓扑规划

拓扑规划用于避免局部最小值。

基于复分析中的二维曲面同胚等价关系 [11]，Rosmann 等 [12] 提出了一种在不同拓扑结构下使用 Voronoi 和基于采样的前端以及 TEB (Timed-Elastic-Bands) 局部规划器 [13] 作为后端的轨迹规划方法。

然而，三维中的同胚等价关系要简单得多。为了捕捉到区别明显的有用路径，Jaillet 等 [14] 构建了视觉变形路网，它包含了比同伦等价类的典型路径更丰富且更相关的信息。基于 [14]，Zhou 等 [15] 通过提出一种有效的拓扑等价性检测实现了实时拓扑规划。我们进一步扩展了 EGO-Planner 以加速拓扑规划的前端部分。

C. Decentralized Drone Swarm 去中心化无人机集群

去中心化方法已经在文献中提出，例如 [16]–[20]。速度障碍被用来保证点机器人 [16]、全向性代理 [17] 和非全向性代理 [18] 的无碰撞航迹。Liu 等人 [21] 提出了一种去中心化异步规划策略，用于避免静态/动态障碍物和车辆间碰撞。虽然这些算法通过仿真进行了验证，但未集成感知、建图和规划能力。实验结

果在文献中已经展示 [19, 20]。[19]实现了多车辆点对点转移而无需外部障碍物，而 [20]则依赖于专门的规划优先级。然而，这些算法均未在实地环境中实现全自主化。

3 IMPLICIT TOPOLOGICAL TRAJECTORY GENERATION OF GRADIENT-BASED LOCAL PLANNING

基于梯度的局部规划隐式拓扑轨迹生成

本节首先介绍了我们之前关于 EGO-Planner 的工作^{\cite{zhou2020ego}}，这是我们提出的群体系统的基础。然后解释了所提出的拓扑规划策略。

A. An ESDF-Free Gradient-based local planner

作为基于梯度的局部规划器，EGO-Planner 将轨迹生成问题形式化为一个非线性优化问题，它在平滑度 \$J_s\$、碰撞 \$J_c\$、动态可行性 \$J_d\$ 和终端进度 \$J_t\$ 之间进行权衡。优化问题的决策变量来自于控制点 \$\mathbf{Q}\$，这些控制点用于参数化轨迹的均匀 B-样条 \$\Phi\$。问题定义如下：

$$\min_{\mathbf{Q}} J_{\text{EGO}} = \sum \lambda_r J_r,$$

其中 \$r=\{s,c,d,t\}\$，下标为 \$\lambda\$ 表示相应的权重。\$J\$ 的项可以分为两类：最小误差和软障碍约束。最小误差项 \$J_s\$ 和 \$J_t\$，通过最小化决策变量线性变换 \$L(\mathbf{Q})\$ 与期望值 \$\mathcal{D}\$ 之间的总误差，被表示为

$$J_r = \sum_{\mathbf{Q} \in \Phi} \|L(\mathbf{Q}) - \mathcal{D}\|_n^n$$

软边界约束项 \$J_c\$ 和 \$J_d\$ 是惩罚决策变量超过特定阈值 \$\mathcal{T}\$ 的常见表达式。

$$J_r = \sum_{\mathbf{Q} \in \Phi} \begin{cases} \left\| \frac{L(\mathbf{Q}) - (\mathcal{T} - \epsilon)}{s} \right\|_n^n & L(\mathbf{Q}) > (\mathcal{T} - \epsilon) \\ 0 & L(\mathbf{Q}) \leq (\mathcal{T} - \epsilon) \end{cases}$$

其中，参数 \$s\$、\$n\$ 和 \$\epsilon\$ 影响着单边约束近似的准确度，如^{\cite{rosmann2012trajectory}}所述。变换 \$L(\cdot)\$ 和参数根据惩罚类型选择。由于篇幅有限，在此处省略了 \$L(\cdot)\$ 的具体形式，可在^{\cite{zhou2020ego}}中找到。

在 EGO-Planner 中，我们提出了一种新颖的障碍物距离估计方法，该方法根据每个 \$\mathbf{Q}\$ 独立拥有的环境信息进行参数化。由若干 \$\{\mathbf{p}, \mathbf{v}\}\$ 对参数化的信息高度抽象自周围障碍物，其中 \$\mathbf{p}\$ 表示障碍物表面上的锚点，\$\mathbf{v}\$ 表示从内部指向外部的安全方向，如图^{\ref{pic:ap_v_pair}}所示。然后，第 \$i\$ 个控制点 \$\mathbf{Q}_i\$ 到第 \$j\$ 个障碍物的障碍物距离 \$d_{ij}\$ 被定义为

\$\{\mathbf{p}, \mathbf{v}\}\$ 对的生成和轨迹优化过程如图^{\ref{pic:ap_v_pair}}和^{???}所示。首先，给出一个简单的初始轨迹 \$\Phi\$，不考虑碰撞。然后，搜索连接碰撞段两端的安全路径 \$\Gamma\$。然后，从 \$\Phi\$ 到 \$\Gamma\$ 生成向量 \$\mathbf{v}\$，并在障碍物表面定义 \$\mathbf{p}\$。通过生成的 \$\{\mathbf{p}, \mathbf{v}\}\$ 对，规划器最大化 \$d_{ij}\$ 并返回优化的轨迹。由于文章长度有限，我们只提供了 EGO-Planner 基本思想的简化描述。详细解释可参见^{\cite{zhou2020ego}}。

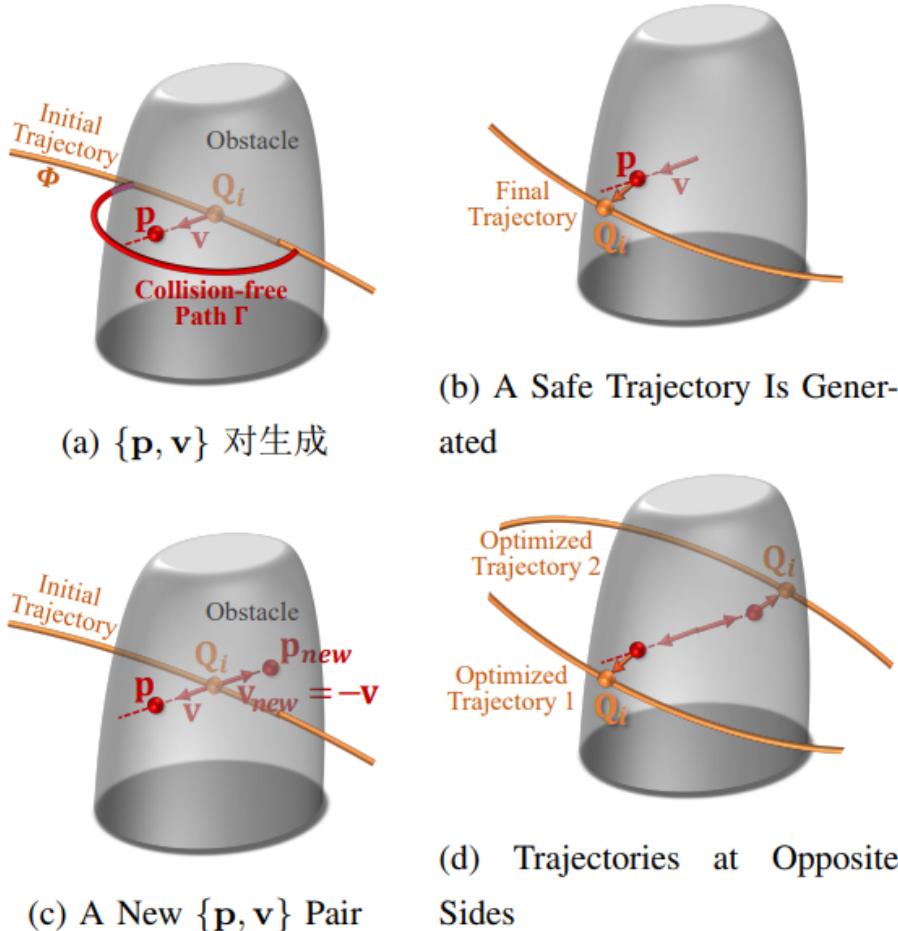


Fig. 4: EGO-Planner 和 拓扑轨迹生成的示意图如下所示: (a) 搜索绕过碰撞轨迹的安全路径, 然后生成 $\{p, v\}$ 对; (b) 优化器找到满足 $d = (\mathbf{Q}_i - \mathbf{p}) \cdot \mathbf{v} >$ 一个固定值的安全轨迹; (c) 生成一个新的 $\{p, v\}$ 对, 其中 $\mathbf{v}_{new} := -\mathbf{v}$, 且 \mathbf{p}_{new} 位于障碍物的对面; (d) 生成满足 $\{\mathbf{p}_{new}, \mathbf{v}_{new}\}$ 对约束条件的不同轨迹。

B. Implicit Topological Trajectory Generation

分析^{cite{jaillet2008path, zhou2020robust}}发现, 广泛使用的同伦概念无法很好地捕捉三维情况下的候选轨迹, 如图^{???}所示。因此, Jaillet 等人^{cite{jaillet2008path}}提出了在三维空间中更有用的关系, 称为可见性变形 (VD-visibility deformation), 而 Zhou 等人^{cite{zhou2020robust}}进一步提取了 VD 的一个子集, 称为均匀可见性变形 (UVD-uniform visibility deformation), 这使得实时操作成为可能。然而, 本文仍然沿用之前的研究中使用的术语“拓扑规划”, 以避免歧义。满足 UVD 的轨迹被认为是同胚的。

^{cite{zhou2020robust}}中定义的 UVD 为:

定义 1:

两条轨迹 $\tau_1(s)$ 、 $\tau_2(s)$, 其参数为 $s \in [0, 1]$, 且满足 $\tau_1(0) = \tau_2(0)$ 、
 $\tau_1(1) = \tau_2(1)$ 的条件, 如果对于所有的 s , 线段 $\tau_1(s)\tau_2(s)$ 没有碰撞, 则属于相同的 UVD 类别。

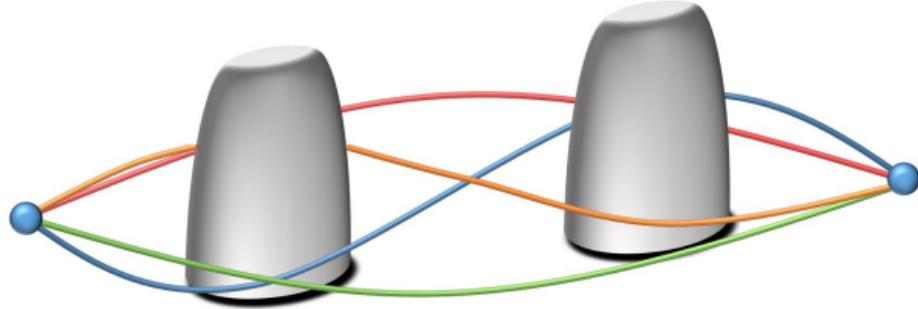


Fig. 5: Four trajectories belonging to the same 3-D homotopy stay in different local minima, which we aim to seek.

传统的拓扑规划方法^[jaillet2008path, rosmann2012trajectory, rosmann2017integrated, zhou2020robust]主要由拓扑不同的路径搜索和后端优化组成，主要关注在拓扑上不同的同伦中寻找多个初始路径。与这些方法不同，所提出的方法通过将 \mathbf{v} 反转为 $\mathbf{v}_{new} := -\mathbf{v}$ ，在不同方向上构建距离场。然后，一个搜索过程在障碍物表面上沿着 \mathbf{v}_{new} 确定一个新的锚点 \mathbf{p}_{new} ，如图ref{pic:multip_v}所示。它们构成了新的一对 $\mathbf{p}_{new}, \mathbf{v}_{new}$ ，导致了一个不同的局部极小值。需要注意的是，没有采用明确的路径搜索，但是任何经过 \mathbf{p} 和 \mathbf{p}_{new} 的路径对应着Def.??在这两个点上的违反。随后，不同的轨迹在不同的线程中并行进行优化，如图4.d所示。执行成本最低的轨迹。

4 无人机集群导航

A. Reciprocal Collision Avoidance 互相避免碰撞

设 t 时刻第 k 个代理在 K 个代理中的位置状态为 $\mathbf{x}_k(t) \in \mathcal{X} \subset \mathbb{R}^3$ 。
 $\mathcal{X}_k^{free}(t) \subset \mathcal{X}$ 是考虑其他代理存在时代理 k 状态空间中的自由区域。因此， $\mathcal{X}_k^{free}(t) := \mathcal{X} \setminus \{ i \in \mathbb{Z} \setminus k, i \leq K \mid \mathbf{x}_i(t) \}$ ，并且对于 Φ_k 的定义域中的任何 t ， $\mathbf{\Phi}_k(t) \in \mathcal{X}_k^{free}(t)$ ，如图6所示。与^[liu2017search]不同，这里忽略了已在 Sec.III-A 中处理的障碍物和动态约束。

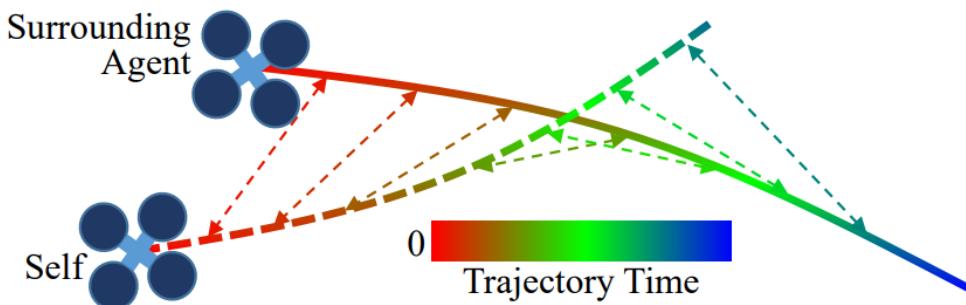


Fig. 6: Drone *Self* generates a trajectory by comparing the distance with a received trajectory from a surrounding drone of the same trajectory time.

与障碍物碰撞和动态不可行性的惩罚类似，我们为代理 k 的群体避碰制定了惩罚函数 $J_{w,k}$ ，作为软约束。

$$d_{k,i}(t) = \|\mathbf{E}^{1/2} [\Phi_k(t) - \Phi_i(t)]\| - (\mathcal{C} + \epsilon)$$

其中， $i \in \mathbb{Z}, i \leq K$ ， t_s 和 t_e 是轨迹 $\Phi_k(t)$ 时间跨度内的全局起始时间和结束时间。 \mathcal{C} 是用户定义的代理间隙。 $\mathbf{E} := \text{diag}(1, 1, 1/c)$ 将欧式距离转换为具有较短主轴的椭圆距离，以减轻下冲风险。将加权项 J_w 添加到方程(1)中，得到每个代理的总优化问题。

$$\min_{\mathbf{Q}} J = J_{\text{EGO}} + \lambda_w J_w.$$

任何包含从决策变量到轨迹上点的映射的轨迹参数化方法都可以应用于公式 (???)。本文使用 p_b 阶均匀 B 样条对轨迹进行参数化，这使得位置评估具有矩阵表示 [\[zhou2019robust\]](#)。

$$\begin{aligned}\Phi(t) &= \mathbf{s}(t)^\top \mathbf{M}_{p_b+1} \mathbf{q}_m \\ \mathbf{s}(t) &= [1 \quad s(t) \quad s^2(t) \quad \dots \quad s^{p_b}(t)]^\top \\ \mathbf{q}_m &= [\mathbf{Q}_{m-p_b} \quad \mathbf{Q}_{m-p_b+1} \quad \mathbf{Q}_{m-p_b+2} \quad \dots \quad \mathbf{Q}_m]^\top\end{aligned}$$

其中 \mathbf{M} 是由 p_b 确定的常数矩阵，当 t 属于节点跨度 $(t_m, t_{m+1}]$ 时， $\mathbf{s}(t) = (t - t_m) \triangle t$ 。

B. Localization Drift Compensation 定位偏移补偿

作为个体在未知环境中的定位（没有可靠的高频环路闭合），在飞行过程中漂移不断积累。Xu 等人 [\[xu2020decentralized\]](#) 提出了一种基于额外 UWB 距离测量的航空群体状态估计方法，并实现了准确的协同定位。然而，我们更关注穿越充满障碍的环境，并且必须为其他应用程序保留计算和通信资源。

Therefore, inspired by [\[xu2020decentralized\]](#), a simplified and lightweight relative drift estimation method is proposed by comparing the predicted position evaluated from received agents' trajectories and the measured positions from depth images of witnessed agents.

因此，在 [\[xu2020decentralized\]](#) 的启发下，通过比较从接收到的代理轨迹评估的预测位置与目击代理深度图像中测量位置，提出了一种简化和轻量级的相对漂移估计方法。当轨迹跟踪误差可以忽略不计，并且可能发生碰撞的任何两个代理中至少有一台能看到另一台时，该策略生效。因此，我们使用 [\[MeIKum1105\]](#) 中的控制器进行精确跟踪，并使用广角相机来降低失去代理的可能性。

消除漂移的过程如下。在评估代理 i 的当前位置 $\Phi_i(t_{\text{now}})$ 后，确定一个以 $\Phi_i(t_{\text{now}})$ 为中心，半径为 R 的球形信任区域 $S \subset \mathbb{R}^3$ ，其中 R 是一个经验参数，表示从实验中估计的典型漂移的上界。然后，将 S 映射到当前捕获的深度图像，即满足的区域 $S' \subset \mathbb{R}^2$ 。

$$z [\mathbf{s}'^\top \quad 1]^\top = \mathbf{K} \mathbf{T}_w^c [\mathbf{s}^\top \quad 1]^\top,$$

其中 $s' \in S$ ， $s \in S'$ ， K 和 T 是相机的内参和外参矩阵， z 是 s 沿主光轴偏离光学中心的偏差。 S' 是一个需要进行复杂计算才能得到的椭圆锥曲线。因此，我们采用了一个近似的轴对齐椭圆 S' 来代替精确的 S 。精确定义信赖区域并不是必要的，因为它只是一个经验区域。

然后我们将 S' 中的每个点投影到世界坐标系，并收集属于 S 的点，这样得到一个点簇 $P \subset S$ 。然后，代理观测的位置 P 被视为 P 的中心（第一原始矩-first raw moment）。

$$\mathbf{P} = \mu'_1(\mathcal{P}).$$

如果 \mathcal{P} 仅包含对应代理的观测而没有任何无关的物体，则等式（??）成立，但这不能保证。然而，由于每个代理都对邻近物体有间隔的轨迹规划，等式（??）在大多数时间内成立。为了改进代理检测的鲁棒性，还添加了其他条件，如像素数目、 \mathcal{P} 的二阶中心矩，当前测量与先前测量之间的差异等。更严格的标准会增加误识率，但由于定位漂移变化缓慢，这是无害的。最后， $\mathbf{\Phi}_i(t\{now\})$ 和 \mathbf{P} 之间的误差被送入一个滤波器，然后获得估计的漂移。

C. Agent Removal from Depth Images 从深度图中删除代理

我们使用占据 栅格地图 来存储静态障碍物，并使用深度图像进行地图融合。移动代理在第??节中得到处理。因此，在地图构建中无需记录移动代理并将其视为障碍物，甚至会产生不必要的干扰。为了消除移动物体的影响，我们在第??节中通过遮罩和移除检测到的代理的像素从深度图像中剔除，如图 7 所示。除此之外，遮挡了大部分视野的移动物体也会对视觉惯性里程计（VIO）产生干扰。因此，灰度图像上的代理也会通过相应深度图像的相同遮罩进行移除。这里使用的代理检测标准较为宽松，因为误判的正样本比误判的负样本更为有害。

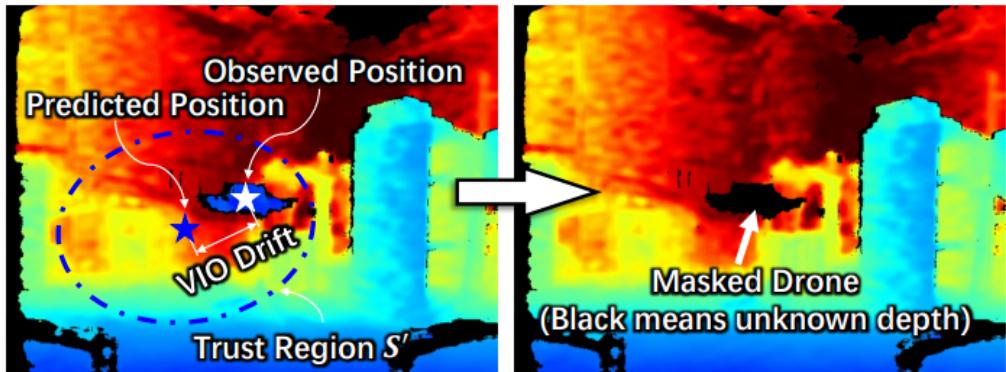


Fig. 7: 左侧:在我们使用的定位方法VIO中，通过在可信区域内比较从接收到的轨迹预测的位置与深度图中观测到的位置，估计浮动情况。右侧:为避免对地图建图产生影响，观测到的代理被屏蔽处理。

5 System Architecture

系统架构如图 8 所示，其中包含了单个代理和多 Agent 通信系统的详细架构。

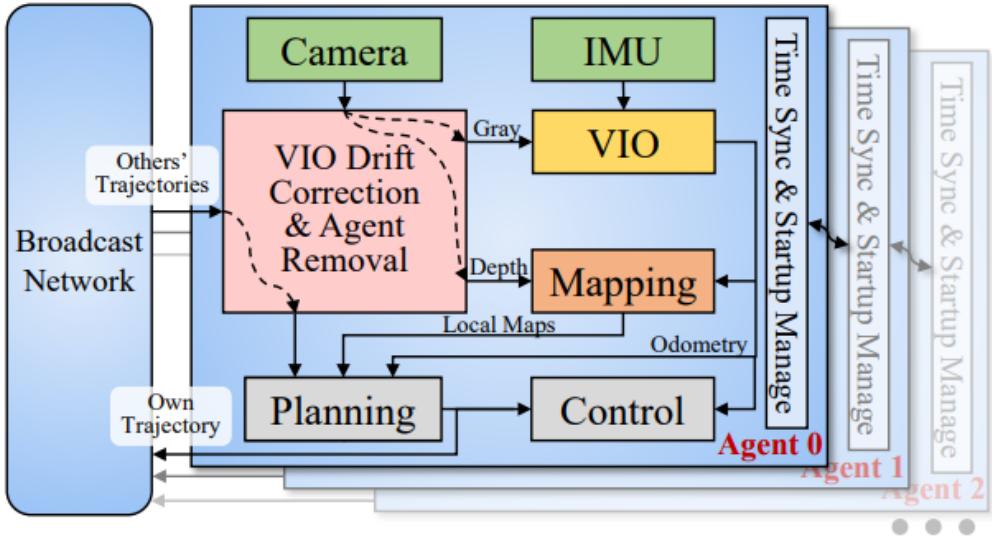


Fig. 8: System Architecture

A. Navigation System of A Single Agent

单一智能体系统，包括硬件和软件设置，基于我们先前的工作 EGO-Planner \cite{zhou2020ego}，增加了一个模块来补偿 VIO 漂移并在图像中去除被观察到的智能体。对于未知环境中的轨迹生成，使用本地规划器。当当前轨迹与新发现的障碍物发生碰撞或智能体接近当前轨迹的结束时，激活规划。

B. Communication Framework

系统由两个网络连接，即共享轨迹的广播网络和同步时间戳以及管理顺序启动的链网络。

1. Broadcast Network 广播网络:

一旦一个 agent 生成了一个新的无碰撞轨迹，它会立即广播给所有的 agent。其他 agent 接收并存储这个轨迹，以便在必要时为自己生成安全轨迹。这种闭环策略在网络连接稳定且延迟可忽略的理想情况下能正常工作。然而，在实践中不能保证这一点。因此，我们提出了两种方法来减少碰撞的可能性。

首先，在网络容量下以给定频率广播一个轨迹。这不会引起计算负担，因为典型的轨迹包含 3D 航点和其他参数，大小小于 0.5KB。相比之下，现代无线网络如蓝牙\footnote{\url{https://www.bluetooth.com/}}可以实现超过 1Mbps 的速度。其次，每个 agent 在从广播网络接收到轨迹后立即检查碰撞，如果发现潜在碰撞，就会生成一个新的无碰撞轨迹。这一策略可以解决多个 agent 在很短的时间内生成轨迹，而没有接收到其他 agent 的轨迹的问题，这是由于延迟或数据包丢失造成的。

此外，考虑到 agent 数量的增加会导致计算复杂度的增加。在规划之前，每个 agent 将其当前位置与接收到的周围 agent 的轨迹进行比较，任何超出规划范围的轨迹将被忽略。

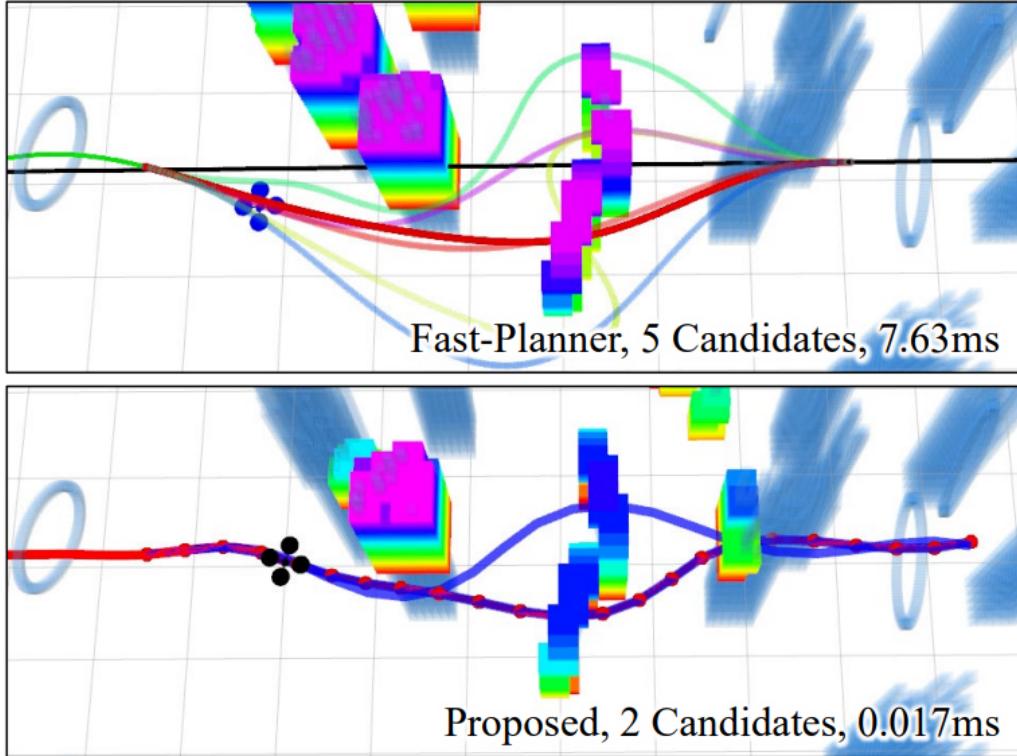


Fig. 9: 前端拓扑路径搜索的比较。所提出的规划器找到的候选拓扑不同轨迹较少，但相比快速规划器，计算量较少。需要注意的是，该时间仅用于前端局部最小值发现，而显示的轨迹是为了更好的可视化而优化的。

2) Chain Network 链式网络:

一个基于连接的稳定链网络被用于时间戳同步和系统启动管理。在系统启动时，代理生成预定义顺序的轨迹。每个代理通过链网络在收到优先级更高的代理的轨迹后生成初始轨迹。这种策略避免了在系统启动过程中由于同时生成轨迹而引起的混乱，因为代理在那时没有其他轨迹的信息。

6 基准测试

在模拟中，使用 i7-9700KF CPU 进行基准比较。对于参数设置，规划视距设置为 7.5m。

$\lambda_s=1.0, \lambda_c=\lambda_w=\lambda_t=0.5, \lambda_d=0.1$ 。地图分辨率为 0.1m。重新规划每秒触发一次或者预测到碰撞时触发。这个设置在模拟和真实世界实验中都被采用。

A. Topological Planning

我们将所建议的 EGO-Swarm 与 Fast-Planner^[zhou2020robust]在候选轨迹数和前端拓扑路径搜索的计算时间方面的拓扑规划性能进行比较。如图 9 所示，EGO-Swarm 发现的候选轨迹较少，这意味着找到全局最优解的概率较低，但比^[zhou2020robust]快 100 倍。由于 Fast-Planner 通过 PRM^[kavraki1996probabilistic]图搜索、路径缩短和路径修剪来找到拓扑上不同的路径，这些方法耗时较长但自由度较高，与所提出的隐式拓扑路径搜索方法相比。

B. Swarm Planning

1. In Empty Space 空地

我们将提出的方法与 DMPC^[luis2019trajectory], ORCA^[van2011reciprocalnbody]和 RBP^[park2020efficient]进行比较, 比较的指标包括飞行距离(d_{fly}), 飞行时间 (t_{fly}), 每个机器人的碰撞次数和计算时间(t_{cal})。每个比较的方法都使用默认参数, 除了最大速度和加速度。如图^[pic:multicomp]所示, 八个机器人在一个圆圈上进行交换转换。*Tab. ???*中的结果是所有机器人的平均值。 t_{cal} 用'*'表示, 因为我们记录的离线方法 DMPC 和 RBP 的计算时间是计划所有机器人整个轨迹的总时间, 而对于 ORCA 和 EGO-Swarm 来说, 它是每个机器人的本地重新规划时间。

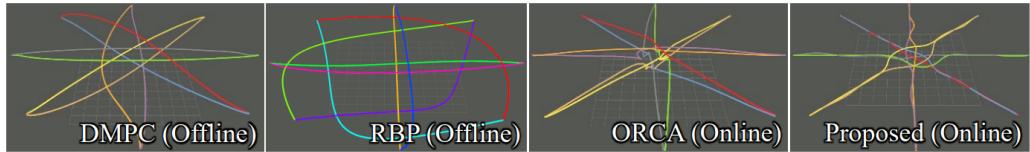


Fig. 10: 各种群组规划器在无障碍情况下的计划轨迹。

*Tab. ???*和图 10 表明, RBP 倾向于生成安全但保守的轨迹, 因为构建凸相对安全飞行走廊^[park2020efficient]显著压缩了解空间。

DMPC 设计用于分布式部署。然而, 在真实世界的应用中无法保证准确和高频的姿态通信。

高效的规则使 ORCA 更新速度快。然而, 使用速度作为控制命令使其不适用于四旋翼等三阶系统。碰撞风险也限制了它的应用。

相比之下, 我们提出的方法生成了最短的无碰撞、非保守性的轨迹, 并具有快速计算的特点。因此, 它可以实现四旋翼的实时应用。

1. In Obstacle-rich Environments 在障碍物密集的环境

我们模拟了十架无人机以每秒 2 米的速度限制从地图的一边飞到另一边, 四旋翼半径为 0.2 米。图 2 是 0.42 个障碍物/ m^2 的模拟快照。每个代理单独感知环境, 并且构建的局部地图以不同的颜色显示。

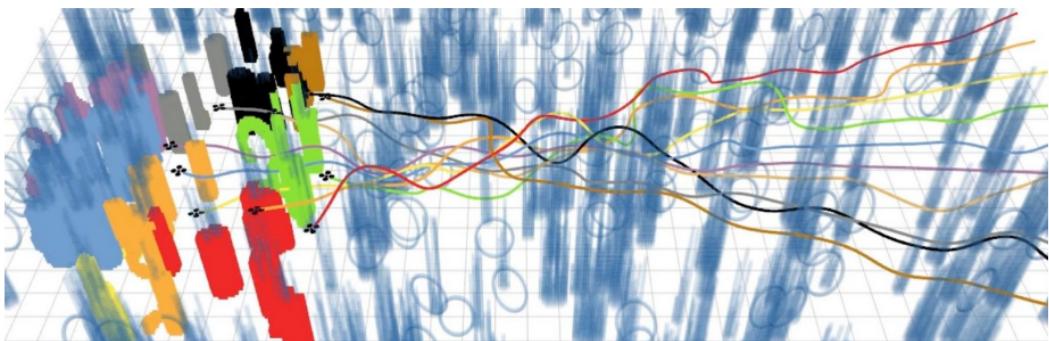


Fig. 2: 十个四旋翼无人机在模拟中飞行。有色方块代表四旋翼的局部感知。每种颜色与一个四旋翼关联。

结果总结在表 2 中, 其中 d_{fly} 是平均飞行距离, d_{safe} 是飞行测试期间与障碍物的最近距离。设计了一种逆向点对点转换, 使得在地图中心周围不可避免地进行相互避碰。在这种情景下, 属于群体的每个代理都成功地规划了平稳且安全的轨迹。

— TABLE II: EGO-Swarm在模拟不同障碍密度下的性能 —

Density(obs/m ²)	d_{fly} (m)	vel.(m/s)	d_{safe} (m)	t_{cal} (ms)
0.42	49.4	1.61	0.208	5.00
0.28	45.4	1.59	0.22	2.87
0.14	43.7	1.57	0.266	1.21
0	42.2	1.55	/	0.49

1. Scalability Analysis 可扩展性分析

我们在一个情景中评估计算性能，其中排列成一条直线的代理飞向随机目标点，距离 50 米。正如图 11 所示，在第 5.B.1 节的按需碰撞检查策略的影响下，随着代理数量的增加，时间复杂度逐渐趋于平稳。

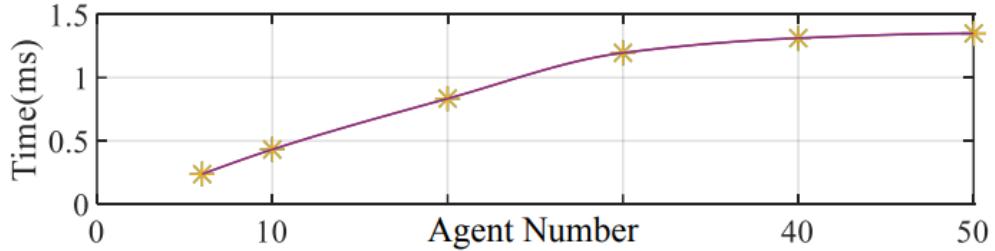


Fig. 11: The relationship between the computation time and the number of Agents in empty space.

7 Real-World exp

1) Indoor

在室内实验中，速度限制为 1.5 米/秒，如图 12 所示。最上方的图显示了三架四旋翼无人机进行循环交换，并进行互相避碰。在中间的图中，四旋翼无人机成功依次通过一个狭窄的门。在底部的图中，环境更加拥挤。三架四旋翼无人机成功穿越了这个环境。

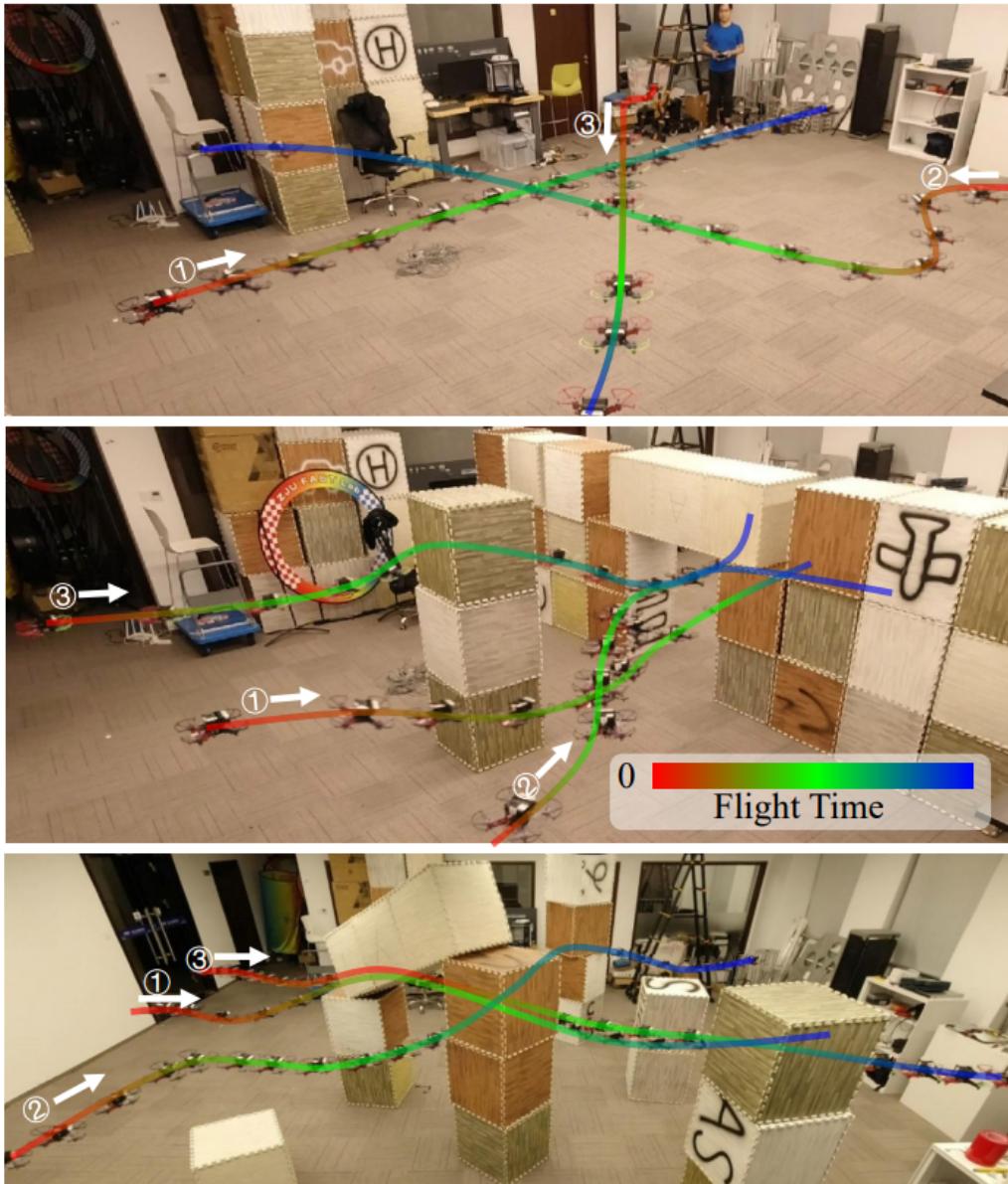


Fig. 12: Indoor Experiments

在一个树木间距大约为 2 米的森林中，如图 1 所示，三个四旋翼无人机一起从森林中起飞，并成功到达森林外的目标位置。速度限制设定为 1.5 米/秒。为进一步强调相互避让，我们将目标位置相对于起始位置的顺序颠倒，使得相互避让成为不可避免的，就像我们在第???节中所做的那样。有关实验的更多信息，请参考我们在 github 上的代码，并观看我们附上的视频^{footnote}{<https://github.com/ZJU-FAST-Lab/ego-planner-swarm>}。



Fig. 1: 三架四旋翼飞行器在森林中的三个快照。

8 总结

本文提出了一种在未知杂乱环境中利用仅限于机载资源的多机器人导航的系统性解决方案。基准比较表明其计算时间短且轨迹质量高。实际世界实验证明了其鲁棒性和效率。

Fast Planner 2019

B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, Robust and efficient quadrotor trajectory generation for fast autonomous flight, IEEE Robotics and Automation Letters, vol. 4, no. 4, pp. 3529–3536, 2019

基于梯度的运动规划

CHOMP

CHOMP: Gradient Optimization Techniques for Efficient Motion Planning

摘要:

现有的高维运动规划算法在某种程度上既过度又不足。在障碍物稀疏分布的领域中，基于采样的规划器用于导航“狭窄通道”的启发式方法可能过于复杂；此外，还需要进一步后处理，以去除这些规划器生成路径中的颠簸或多余的动作。在本文中，我们介绍了 CHOMP，一种使用 协变梯度技术 covariant gradient techniques 持续优化路径的新方法，用于提高采样轨迹的质量。我们的优化技术既可以优化高阶动态性，又可以相对于以前的路径优化策略在更广泛的输入路径上收敛。特别地，我们放宽了这些策略要求的输入路径上的无碰撞可行性先决条件。因此，CHOMP 可以用作许多实际规划查询中的独立运动规划器。我们演示了我们提议的方法在一个 6-DOF 机器臂的操纵规划以及在一个行走的四足机器人的轨迹生成中的有效性。

Stomp

Stomp: Stochastic trajectory optimization for motion planning

摘要 — 我们提出了一种使用 随机轨迹优化框架 进行运动规划的新方法。此方法依赖于生成噪声轨迹来探索初始（可能不可行）轨迹周围的空间，然后结合它们产生一个成本更低的更新轨迹。在每次迭代中，基于障碍和平滑性成本的组合优化一个成本函数。我们使用的特定优化算法不需要梯度信息，因此可以在成本函数中包括可能无法获得导数的一般成本（例如，对应于约束和电机扭矩的成本）。我们在模拟和移动操作系统上展示了这种方法，用于无约束和受约束的任务。我们实验性地显示，STOMP 的随机性使其能够克服 CHOMP 这样的基于梯度的方法可能陷入的局部最小值。

Continuous-time trajectory optimization for online uav replanning

Realtime trajectory replanning for mavs using uniform b-splines and a 3d circular buffer

An efficient b-spline-based kinodynamic replanning framework for quadrotors

Robust and efficient quadrotor trajectory generation for fast autonomous flight

Raptor: Robust and perception aware trajectory replanning for quadrotor fast flight

Swarm of micro flying robots in the wild

Science Robotics

Xin Zhou, Xiangyong Wen, Zhepei Wang, Yuman Gao, Haojia Li, Qianhao Wang, Tianskai Yang, Haojian Lu, Yanjun Cao, Chao Xu, Fei Gao

整体结论

传统的优化算法对集群规划耗时长，虽然通过将其分组能简化到某个程度，但仍有限。而去中心化、将问题适当设定，耗时显著下降，且跟无人机数量成正比。

采用设定

1. 采用静态的栅格地图，并在其上叠加由深度图识别的动态物体。
2. 定位根据 视觉惯性的里程计（基于静态环境的灰度图）实现，没有 GPS。
3. 无人机的识别不依靠深度学习，而是深度图的 pixel gathering（采用 boundingbox 并据此计算无人机之间的距离）。
4. 分成不同场景和任务，采用不同的通讯方式、参数配置，而不是一套方案通用。

关键观点

1. 通信方式：
 - 中心化在密集阵型表现得更好，随着数量增加其复杂度线性提高；
 - 点对点通信在稀疏时数量增加几乎不增加复杂度（因为超出一定距离就不通讯），但在密集时复杂度将指数提高。
2. 轨迹参数化（实现难度依序提高）：
 - i. 分段多项式：需要大量矩阵运算而对计算机负担较重；分段的关键点位置的选择对效果和算力影响很大；要加入连续性约束。
 - ii. Bezier 和 B-Spline 曲线：convex hull 特性让增加约束很方便，而且天然具备连续性；缺点是偏向保守，无法发挥极限机动能力。
 - iii. MINCO（一种多项式轨迹类）：针对集成链路系统设计的新方法，能接受大量约束条件设置的同时，保持时空最优性能；但必须对其结果进行充分的再检查，而且实现起来很复杂。
3. 通信内容：根据执行的任务而不同。例如“飞越森林”任务中，目的地已经作为参数提前写入每个无人机之中，因此无人机之间只需要交流“实时的轨迹”。
4. 通讯距离必须大于两倍的规划距离：小于这个距离的无人机将互相影响，大于这个距离的在物理上就留有足够的反应空间。
5. 采用触发而非周期式的通信，在 65m 的竹林穿越任务中，10 架无人机平均发送了 100 条轨迹，接受 1000 条轨迹。（注：应该是轨迹发生重规划时才对外发送，以减少通讯量。可以看到部分的无人机发送量比其他的高 50% 以上。）

摘要：空中机器人被广泛部署，但高度杂乱的环境，如茂密的森林，无人机仍然无法进入，对成群的无人机来说更是如此。在这些情况下，以前未知的环境和狭窄的走廊以及群体协调的要求可能会带来挑战。为了在野外实现群体导航，我们开发了 具有轨迹规划器的微型但完全自主的无人机，该无人机可以根据来自机载传感器的有限信息及时准确地运行。规划问题满足飞行效率、避障、机器人间避碰、动态可行性、群体协调等多种任务要求，实现了可扩展的规划器。此外，所提出的规划器基于时空联合优化同步变形轨迹形状并调整时间分配。因此，即使在最受限制的环境中，也可以在几毫秒内彻底利用解空间后获得高质量的轨迹。规划器最终集成到开发的手掌大小的群平台中，具有板载感知、定位和控制功能。基准比较验证了规划器在轨迹质量和计算时间方面的卓越性能。各种真实世界的现场实验证明了我们系统的可扩展性。我们的方法在三个方面发展了空中机器人技术：杂乱环境导航的能力、对不同任务要求的可扩展性以及在没有外部设施的情况下作为群的协调。

密集环境下编队飞行的稳健高效的轨迹规划-2023

Robust and Efficient Trajectory Planning for Formation Flight in Dense Environments

IEEE Transactions on Robotics

<https://arxiv.org/abs/2210.04048>

编队飞行在空中机器人群的各种应用中具有巨大潜力。然而，现有方法缺乏在密集环境中实现完全自主的大规模编队飞行的能力。为了弥补这一差距，我们提出了一个完整的编队飞行系统，该系统能有效地将现实世界的约束条件整合到空中编队导航中。本文提出了一种 可微分的基于图的度量方法 (differentiable graph-based metric)，用于量化编队之间的 整体相似度误差。该指标不受旋转、平移和缩放的影响，为编队协调提供了更大的自由度。

我们设计了一个 分布式轨迹优化框架，该框架考虑了编队相似性、避障和动态可行性。通过将优化过程解耦，使大规模编队飞行在计算上可行。

为了提高编队在高度受限场景中的导航弹性，我们提出了一种 蜂群重组方法，通过生成局部导航目标，自适应地调整编队参数和任务分配。本研究提出了一种名为 " 全局重映射-局部重规划 " (global-remap-local-replan) 的 新型蜂群协议策略，并提出了一个 编队级路径规划器，以协调全局规划和局部轨迹优化。

为了验证所提出的方法，我们从适应性、可预测性、弹性、复原力和效率等方面设计了综合基准，并与其他前沿工作进行了模拟。最后，通过与带有机载计算机和传感器的手掌大小的蜂群平台集成，我们提出的方法在密集的室外环境中实现了最大规模的编队飞行，证明了它的高效性和鲁棒性。

空中群聚、编队飞行、避障、运动规划、分布式轨迹优化。

尽管大量的研究工作集中在编队飞行导航上，但在充满障碍物的区域中实现稳健的编队飞行的研究还很少。实际应用中存在三个核心挑战：

- (a) 编队维持与障碍物避免之间的固有冲突是不可避免且难以缓解的。
- (b) 预定义编队在受限环境中缺乏弹性适应能力。
- (c) 由于未知障碍物或期望编队形状的突然变化而导致的无序状态，群体系统难以迅速恢复。

基于上述挑战，我们得出结论，理想的编队飞行系统应具备在避免障碍物的同时维持编队的能力，根据受限环境调整群体编队分布，并在紧急情况下快速重组编队的能力。这些特征被概括为 PAPER 准则：

- **Portability 移动性**：空中机器人集群应由轻型平台、可扩展系统和分布式架构组成。可扩展系统意味着主要组件（如估计、决策、规划和控制模块）在每个机器人上都是相同的。分布式架构能够天然地抵御个体硬件故障。这是实现大规模编队飞行的基础。
- **Adaptability 适应性**：当面临障碍时，机器人应当在局部适应其轨迹，以避免碰撞，并尽量减少对整体编队性能的影响。这种能力可以缓解编队维持和障碍物避让之间的冲突。
- **Predictability 可预测性**：反应式的局部反馈方法的视野短浅，无法提前考虑到约束条件。机器人应该在预测的时间范围内优化运动，使得形成能够对其周围未来环境的变化做出平稳响应，这对于密集区域是必要的。
- **Elasticity 弹性**：在受限环境中，如狭窄的走廊或洞穴中，可能不存在适用于固定编队形状的可行且安全的路径。因此，群体机器人需要通过调整编队分布（如形状的规模或任务分配）来具备弹性和灵活的变形能力，同时保持编队的完全操纵性。
- **Resilience 可恢复性**：编队飞行可能会遇到由未知障碍物或者期望编队形状的突然改变引发的不利情况。导航系统应具备弹性重新组织和引导整个编队的能力，以便飞行能够及时从无序状态中恢复。

一个完整的编队飞行系统应满足上述 PAPER 标准，并确保每个标准的条件与其他条件相容。

我们之前的工作¹只部分满足了 PAPER 标准的前三个条款。我们将编队飞行问题作为一种 耦合的协同轨迹优化问题 来解决，主要适用于小规模编队场景。然而，使用基于图的相似度量解决编队的协作约束计算量较大，导致每次优化迭代过程中的开销增加。此外，将动态的机器人间关系集成到耦合轨迹优化问题中会显著影响优化过程的效率，使其在更大规模的编队或更复杂的场景中不太适用。

在本文中，我们提出了一个完整的编队飞行系统，满足所有 PAPER 标准。为解决[4]中的挑战，我们引入了一种解耦的编队优化方法，以显著提高计算效率。该方法包括两个组成部分。

首先，预先计算 最优编队位置序列，避免了优化过程中重复的度量计算。

其次，采用固定时间间隔采样方法将动态的机器人间关系转化为 静态约束，极大地降低了优化问题的复杂性。这些改进使得我们的方法适用于大规模的群体。

此外，之前的方法缺乏重新组织群体编队的能力，在恶劣条件下，特别是当初始位置或任务分配不当时，可能导致编队飞行无序。为解决这个问题，我们提出了一种 群体重新组织方法，可以通过优化编队参数和任务分配来弹性地调整编队分布，以响应外部约束。

随后，我们提出了一种 群体一致性策略，称为 全局重映射-局部重新规划，该策略可使群体重新组织的结果得到快速实施，从而在群体智能体之间实现共识。

此外，还设计了一种以编队为整体的 全局路径规划方法，用于引导群体走出障碍物的死锁。

最后，我们将估计、建图、决策、规划和控制模块整合到了带有机载计算机和传感器的手掌大小编队平台中 Micro Flying Robots-21，实现了在密集环境中的大规模编队飞行。具体贡献如下。

我们引入了一种 最优 形成位置序列（formation position sequence），该序列是使用可微的基于图的度量方法预先计算得出的²。这个序列代表了具有最低相似度误差的最优位置，减少了优化过程中重复计算的需求。

我们设计了一个 解耦的空间-时间轨迹优化框架，有效处理动态的机器人之间的关系、避障和动态可行性。与我们之前的研究成果³相比，我们在大规模群体中实现了更高的计算效率。

我们提出了一种群集重新组织方法，实现群集分布的弹性变形，同时解决最佳形态对齐和任务分配问题（简称 ALAS）。该方法提高了群集形成对受限环境的弹性。它减轻了对适当的形态对齐和任务分配的依赖。

Task assignment algorithms for unmanned aerial vehicle networks: A comprehensive survey 2022.06

Department of Computer Engineering, Chosun University, Gwangju 61452, South Korea 计算机工程学院, 朝鮮大学, 韩国光州 61452

- Vehicular Communications
- [sciencedirect](#)
- [doi](#)

Abstract 抽象的

Unmanned aerial vehicles (UAVs) have significant prospects in a plethora of public and civic spheres. Recently, UAVs have focused primarily on applications where human presence is either impossible or hazardous. A swarm of small UAVs can cooperatively complete operations more proficiently and economically than a single large UAV. However, many issues must be resolved before stable and reliable multi-UAV networks can be realized. Task assignment in fleets of UAVs is concerned with cooperative decision-making and control. UAVs possess various functional abilities and kinematic constraints while carrying limited resources onboard. UAVs are nominated to execute multiple sequential tasks supportively on numerous ground targets. The prime objective of task assignment is to minimize the task accomplishment time and UAV energy consumption. To date, several task assignment algorithms have been designed for UAV networks, and they are comprehensively surveyed in this paper in terms of their main ideas, operational features, advantages, and limitations. These task assignment algorithms are then compared in terms of their significant characteristics and performance factors. To the best of the authors' knowledge, no survey on task assignment techniques for different UAV missions currently exists in the literature. We also discuss open issues and challenges and then suggest projections for task assignment algorithms concerning possible future directions.

无人机（UAV）在众多公共和公民领域具有广阔的前景。最近，无人机主要关注人类不可能存在或存在危险的应用。一群小型无人机可以比单个大型无人机更高效、更经济地协同完成行动。然而，要实现稳定可靠的多无人机网络，还必须解决许多问题。无人机机队的任务分配涉及协作决策和控制。无人机具有多种功能和运动学限制，但机载资源有限。无人机被指定在众多地面目标上执行多项连续任务。任务分配的首要目标是最小化任务完成时间和无人机能耗。迄今为止，已经设计了几种针对无人机网络的任务分配算法，本文对它们的主要思想、操作特点、优点和局限性进行了全面的综述。然后对这些任务分配算法的显着特征和性能因素进行比较。据作者所知，目前文献中还没有针对不同无人机任务的任务分配技术的调查。我们还讨论了悬而未决的问题和挑战，然后就有关未来可能方向的任务分配算法提出了预测。

Keywords 关键词

- Unmanned aerial vehicle; Multi-UAV networks; UAV autonomy; Task assignment; Task coordination; Heterogeneity;
- 无人机 多无人机网络 无人机自主性 任务分配 任务协调 异质性
- Introduction
 1. Challenges
 2. Motivation
 3. Contribution of the study
 4. Organization of the paper

- Overview of existing surveys on UAVs and our contributions
- Considerations for UAV task assignment
 - 1. Single- and multi-UAV systems
 - 2. Classification of UAVs
 - 3. Background study
- Design issues for task assignment algorithm
 - 1. Multi-UAV system formation
 - 2. Computational complexity
 - 3. Collision-free UAV operation
 - 4. Backhaul network
 - 5. Uniform load balancing
 - 6. Antenna designs
- Task Assignment Algorithms
 - 1. Centralized task assignment algorithms
 - 2. Distributed task assignment algorithms
 - 3. Bio-inspired task assignment algorithms
- Comparison of task assignment algorithms for UAVs
- Open issues, research challenges, and future directions
- Conclusion

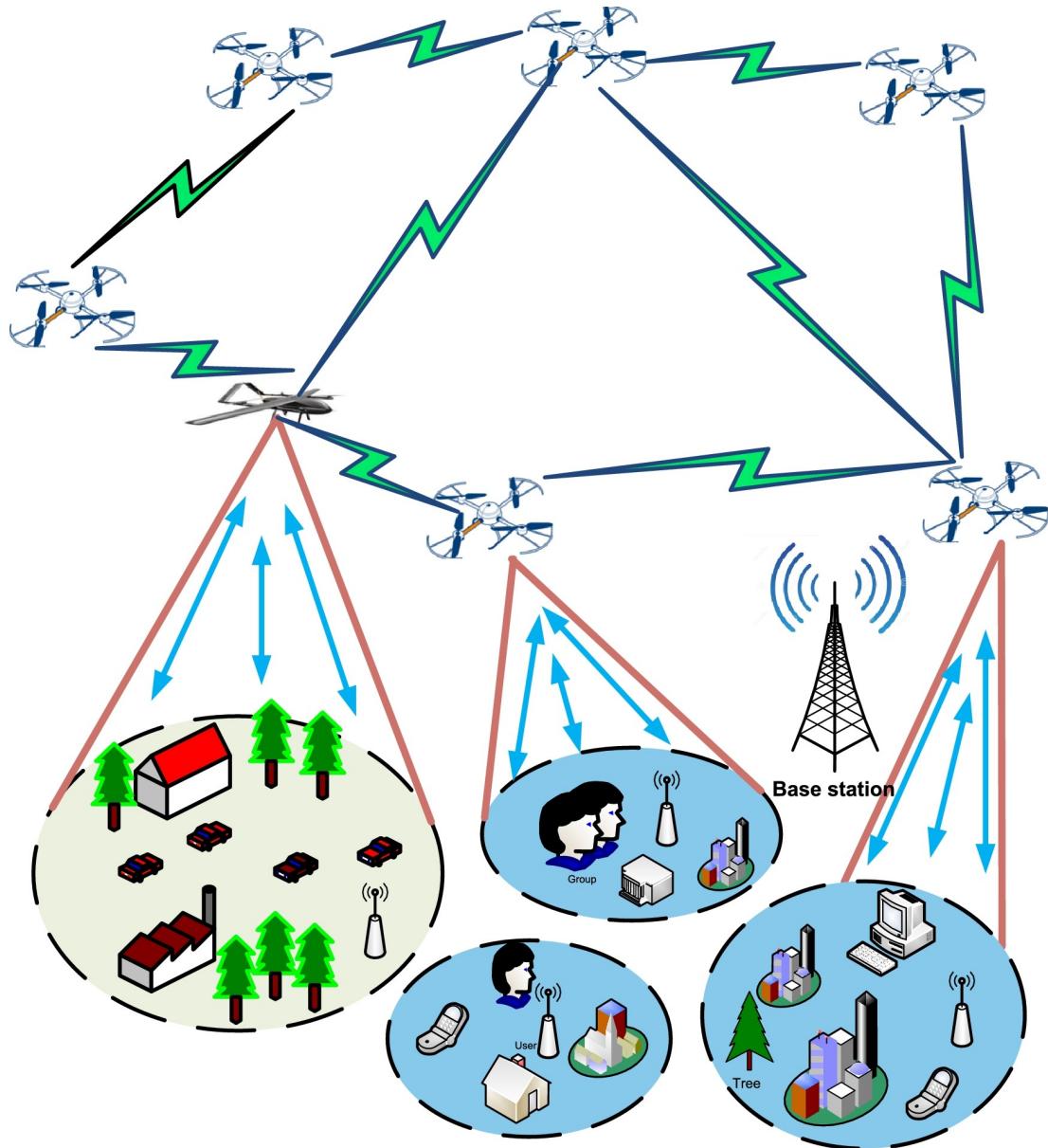
1. Introduction 1. 简介

The remarkable and prompt growth of unmanned aerial vehicles (UAVs) in many different applications has recently fascinated many researchers and operators. The incorporation of UAVs is noticeable in such applications, which are either dangerous or impossible for a human. UAVs can be deployed quickly and easily, possess high maneuverability, are cost-efficient, have the self-organizing ability, and are flexible and scalable. Hence, UAVs have received momentous consideration for civil and military applications. The applications of UAVs include but are not limited to emergency and crisis management [1], [2], [3], search and rescue operations in disastrous environments [4], [5], monitoring agricultural fields [6], post-disaster operations [7], remote sensing [8], [9], wildfire monitoring [10], traffic monitoring [11], freight transportation [12], and relay networks [13], [14], [15]. In the present day, UAVs have also shown wide use in pandemic environments to enforce social distancing [16], spread disinfectants in infected areas [17], carry testing kits and other medical supplies [18], and more. UAVs communication can be distinguished from other ad-hoc communication in the following ways: a) dynamic channel for air-to-air (A2A) and air-to-ground (A2G) propagations; b) spatial as well as temporal inconsistencies induced by high UAV mobility; c) shadowing effect due to design structure and rotation of UAV. UAVs of different sizes and specifications are available. Generally, single-UAV-based applications prefer large UAVs with higher abilities, while small UAVs are chosen for UAV formations and swarms. A multi-UAV operation scenario is illustrated in Fig. 1 and is expected to increase significantly in the future. As shown in the figure, several UAVs are assigned different tasks and are supposed to execute tasks with proper coordination with other UAVs.

无人机（UAV）在众多不同应用中的显著且迅速增长最近吸引了众多研究人员和操作者的兴趣。无人机在这些应用中的融入尤为明显，这些应用要么危险，要么对人类来说不可能。无人机可以快速、轻松地部署，具有高机动性，成本低廉，具有自组织能力，且灵活可扩展。因此，无人机在民用和军事应用中受到了极大的关注。无人机应用包括但不限于紧急和危机管理[1]、[2]、[3]、灾难环境中的搜救行动[4]、[5]、监测农田[6]、灾后行动[7]、遥感[8]、[9]、野火监测[10]、交通监控[11]、货物运输[12]和接力网络[13]、[14]、[15]。在当今时代，无人机在疫情环境中也显示出广泛的应用，以执行社交距离[16]、在感染区域喷洒消毒剂[17]、携带检测套件和其他医疗用品[18]等。

无人机通信可以通过以下方式与其他临时通信区分开来： a) 动态信道用于空对空（A2A）和空对地（A2G）传播； b) 由于高无人机移动性引起的时间和空间不一致性； c) 由于无人机设计结构和旋转产生的阴影效应。

不同尺寸和规格的无人机可供选择。通常，基于单架无人机的应用更倾向于选择能力更强的大型无人机，而小型无人机则用于无人机编队和集群。图 1 展示了多架无人机操作场景，预计未来将显著增加。如图所示，几架无人机被分配不同的任务，并应与其他无人机进行适当的协调以执行任务。



1.1. Challenges of UAV networks

1.1. 无人机网络的挑战 The progress made in microchip technology has helped widen the application range of UAVs [19]. Though the UAV network has been promising, some challenges need to be resolved for effective UAVs based on reliable networks. Significant research has been done in the literature that analyzes the challenges and prospects for successful UAVs operations. In [20], [21], a study focusing on the challenges faced by UAVs during deployment, interference, UAV-based relaying, regulations, energy consumption, collision avoidance, and security is made. Various challenges regarding spectral efficiency in multi-UAVs operations are reviewed in [22].

Different issues faced in UAV communication, i.e., routing, handover, and energy efficiency, are deliberated in [23]. Furthermore, some studies have focused on UAVs and UAV-based communications, and state-of-the-art works consider UAVs' particular domain or efficacy [24], [25]. UAV flights can be understood as several degrees of autonomy. Obtaining autonomy for UAVs is a complicated issue, and the complication level depends on different phases in the decision-making process and the necessary level of cooperation between UAVs [26], [27].

微芯片技术的进步有助于扩大无人机（UAVs）的应用范围[19]。尽管无人机网络前景广阔，但为了实现基于可靠网络的无人机有效运行，仍需解决一些挑战。文献中已对成功无人机操作的挑战和前景进行了大量研究。在[20]、[21]中，对无人机部署、干扰、基于无人机的中继、法规、能耗、避障和安全等方面面临的挑战进行了研究。在[22]中，回顾了多无人机操作中关于频谱效率的各种挑战。在[23]中，讨论了无人机通信中面临的不同问题，即路由、切换和能效。此外，一些研究专注于无人机及其基于无人机的通信，而最先进的工作则考虑了无人机特定的领域或功效[24]、[25]。无人机飞行可以理解为几个级别的自主性。为无人机获得自主性是一个复杂的问题，其复杂程度取决于决策过程中的不同阶段以及无人机之间必要的合作水平[26]、[27]。

The maximum cooperation level is essential for task assignment algorithms [28], [29], [30], where UAVs are expected to share information, allocate tasks amongst each other, and consign responsibilities with suitable timing and organization. Task assignment can be described as a graph theory problem and explained using deterministic search algorithms. Due to their computational complexity, traditional deterministic algorithms can handle only small and simple issues. Tasks of a heuristic nature can be solved, but suitable efficiency may not be achieved [31]. Proper communication and exchange of information among UAV groups can improve UAV competence to encounter performance necessities associated with the speedy and trustworthy implementation of tasks. Appropriate collaboration among UAVs is complex and very challenging to implement. An optimization algorithm that considers task superiority and synchronization, flightworthy trajectories, and other constraints of real-world scenarios are highly desired. Furthermore, UAVs differ in their operating abilities, and the cooperation of multiple heterogeneous UAVs can complement mission environments with different types of tasks [32].

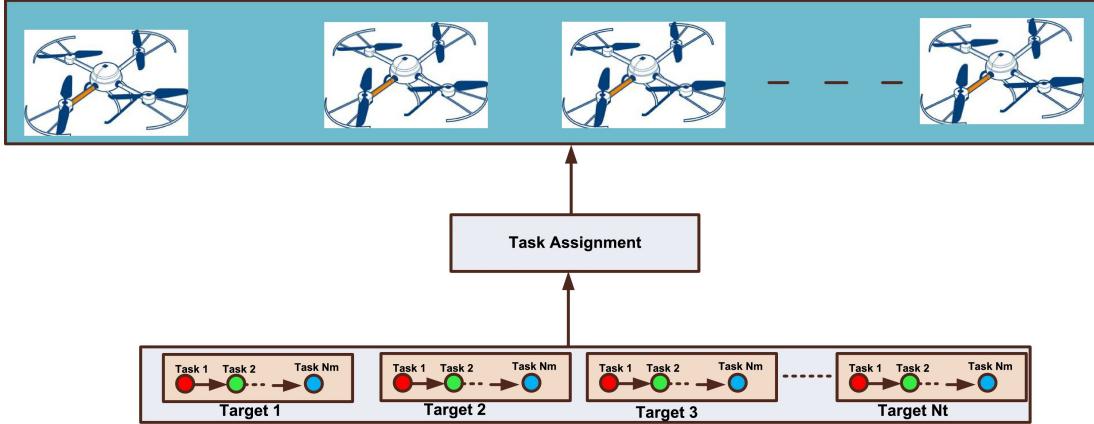
最大合作级别对于任务分配算法[28]、[29]、[30]至关重要，在这些算法中，无人机预计将共享信息，相互分配任务，并在适当的时机和组织下委托责任。任务分配可以描述为一个图论问题，并使用确定性搜索算法进行解释。由于它们的计算复杂性，传统的确定性算法只能处理小而简单的问题。可以解决启发式任务，但可能无法达到合适的效率[31]。在无人机群体之间进行适当的沟通和信息交流可以提高无人机应对与快速和可靠任务执行相关的性能需求的能力。无人机之间的适当协作复杂且极具挑战性。一种考虑任务优先级和同步、飞行轨迹以及现实场景其他约束的优化算法非常受欢迎。此外，无人机在操作能力上存在差异，多个异构无人机的合作可以补充具有不同类型任务的使命环境[32]。

1.2. Motivation 1.2. 动机

UAV communication imposes many challenges owing to the outstanding issues present in different constituents of UAV networks, as discussed in the previous section. On the other hand, these challenges motivate many researchers to focus their studies on unsolved issues. Real-world UAV-based operations rely on collaboration between UAVs to manage the threat of attrition. Cooperation is fundamental for planning efficient missions in profoundly vulnerable environments. Efficient task allocation algorithms act as they are designed and account for uncertainties in the dynamic environment [33]. Task assignment is a combinatorial optimization process by which a UAV or swarm of UAVs is allocated to accomplish many tasks. Fig. 2 shows how a task assignment algorithm assigns a sequence of tasks within a target region to available UAVs such that the overall cost of the task is minimized.

无人机通信由于无人机网络不同组成部分中存在突出的问题而面临许多挑战，如前节所述。另一方面，这些挑战激励许多研究人员将他们的研究集中在未解决的问题上。现实中的基于无人机的操作依赖于无人机之间的协作来管理损耗威胁。在极其脆弱的环境中，合作对于规划高效任务至关重要。有效的任务

分配算法按设计运行，并考虑动态环境中的不确定性[33]。任务分配是一个组合优化过程，通过该过程，无人机或无人机群被分配以完成许多任务。图 2 显示了任务分配算法如何将一系列任务分配给可用的无人机，以便任务的总成本最小化。



The task allocation process for UAVs has become an emergent issue in the past few years with the widespread use of multi-UAV-based mission planning systems [34], [35]. A task manager is accountable for identifying all tasks and decides the availability and capability of UAVs to assign tasks to other UAVs proficiently. Several task assignment algorithms have been studied in the literature over several decades. Task assignments can be broadly categorized into coordinated, distributed, stochastic, deterministic, evolutionary, and multi-fusion-based algorithms.

无人机任务分配过程在过去几年随着基于多无人机任务规划系统的广泛应用而成为一个新兴问题[34], [35]。任务管理器负责识别所有任务，并决定无人机分配任务给其他无人机的能力和可用性。几十年来，文献中已经研究了多种任务分配算法。任务分配可以广泛分为协调、分布式、随机、确定性、进化以及基于多融合的算法。

In this survey, we extensively investigated task assignment optimization algorithms designed for UAV networks to address their key features and characteristics. In addition, we provide detailed explanations of the basic operational principles. In the near future, UAVs may require an ever-increasing number of higher-level planning capabilities to accomplish their missions effectively. Their missions are expected to be complex, necessitating the use of multiple heterogeneous UAVs to cooperate efficaciously while achieving the overall mission objective. The main contributions of our study are discussed in the following subsection.

在本调查中，我们广泛研究了针对无人机网络的作业分配优化算法，以解决其关键特性和特征。此外，我们还提供了基本操作原理的详细解释。在不久的将来，无人机可能需要越来越多的更高级别的规划能力，以有效地完成其任务。预计他们的任务将是复杂的，需要使用多个异构无人机有效地合作，同时实现整体任务目标。我们研究的主要贡献将在下一个节中讨论。

1.3. Contribution of the study

1.3. 研究贡献

UAVs essentially depend on sensors' information to categorize targets, discard false tasks, and determine comprehensible sequences of decisions and actions to achieve their objectives. The significant contributions of our survey are as follows.

无人机本质上依赖于传感器的信息来分类目标、丢弃错误任务，并确定可理解的决策和行动序列以实现其目标。本调查的显著贡献如下。

- Based on their main ideas, existing surveys on UAVs and their applications are reviewed.

- 基于它们的主要思想，对现有关于无人机及其应用的调查进行了回顾。
- The motivation behind the study of task assignment algorithms is elaborated on along with design considerations.
 - 研究任务分配算法背后的动机以及设计考虑被详细阐述。
- Existing task assignment optimizations for UAV networks are classified and compared based on their main ideas, advantages, limitations, and possible enhancements. No other earlier works have surveyed the task assignment algorithms proposed for the single and multi-UAVs applications.
 - 现有无人机网络的任务分配优化被根据其主要思想、优点、局限性和可能的改进进行分类和比较。没有其他早期工作对为单无人机和多无人机应用提出的任务分配算法进行了调查。
- State-of-the-art task assignment optimizations are extensively compared and contrasted in terms of their principles, features, advantages, and limitations.
 - 最先进的任务分配优化在原理、特性、优点和局限性方面进行了广泛的比较和对比。
- Task assignment optimizations are qualitatively compared with various features and characteristics with a comprehensive and comparative discussion.
 - 任务分配优化与各种特征和特性进行了定性比较，并进行了全面和比较性的讨论。
- Finally, critical open issues and research challenges faced during task assignments in multi-UAV systems are summarized and deliberated for further enhancement and improvement.
 - 最终，多无人机系统任务分配过程中遇到的关键开放问题和研究挑战被总结并讨论，以进一步改进和提高。

1.4. Organization of the paper

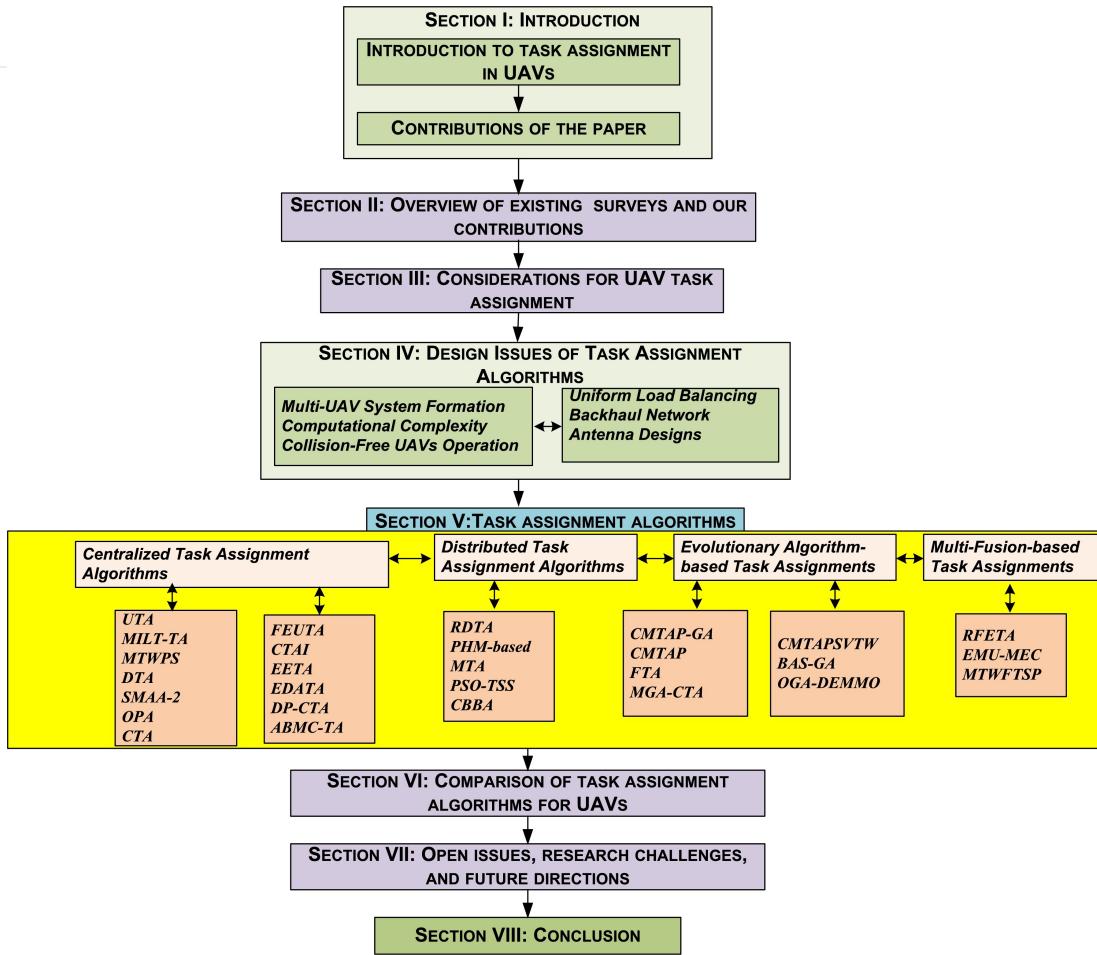
1.4. 论文组织结构

The remainder of this paper is organized as follows. Section 2 provides an overview of the state-of-the-art surveys conducted in different fields of UAVs. Section 3 introduces and discusses additional considerations for UAV task assignment algorithms. Task assignments suggested for UAV networks are classified in to different categories and extensively reviewed in Section 4. The task assignment algorithms studied in Section 4 are qualitatively compared and discussed in Section 5. In Section 6, open issues, research challenges, and future projections are summarized and discussed. Finally, concluding remarks are presented in Section 7. The organizational structure of the paper is depicted in Fig. 3.

本文剩余部分组织如下。

- 第 2 节概述了在不同无人机领域进行的最新调查。
- 第 3 节介绍了无人机任务分配算法的附加考虑因素并进行讨论。
- 第 4 节对无人机网络中建议的任务分配进行了分类和广泛审查。
- 第 4 节中研究过的任务分配算法在第 5 节中进行了定性比较和讨论。
- 在第 6 节中，总结了开放性问题、研究挑战和未来展望，并进行了讨论。
- 最后，在第 7 节中提出了结论性评述。

本文的组织结构如图 3 所示。



2. Overview of existing surveys on UAVs and our contributions 关于无人机现有调查的概述及我们的贡献

Some surveys and tutorials for UAVs and UAV-based communications have existed in the literature of the past several decades. Most of them have focused on the different issues faced in UAV communication, such as energy and other resource efficiency, security aspects, charging techniques, and channel modeling. Existing surveys on various aspects of UAV and UAV-based applications are summarized in Table 1 to validate the distinctiveness of our study.

一些关于无人机及其通信的调研和教程在过去几十年文献中已有存在。其中大部分聚焦于无人机通信中面临的不同问题，如能源和其他资源效率、安全方面、充电技术和信道建模。表 1 总结了关于无人机及其基于无人机应用各方面现有调研，以验证我们研究的独特性。

Table 1. Summary of existing surveys.

Reference	Year	Description	Classification of UAVs	Challenges	Topic discussed	Intel dec
[36]	2019	Presents an ample tutorial on using the UAVs in wireless networks	✗	✓	Smart cities	✓
[37]	2021	A survey of security for critical UAV applications	✗	✓	UAV security	✓
[38]	2016	Surveys UAV-based civil applications	✗	✓	UAV for civil application	✗
[39]	2020	A survey of routing protocols for UAVs	✓	✓	Routing in UAVs	✓
[40]	2019	A survey of UAVs from a cyber-physical system perspective	✗	✓	UAV networks	✓
[41]	2018	Surveys the methods of UAV channel modeling	✗	✓	Channel modeling in UAVs	✗
[42]	2021	Survey of MAC protocols for FANETs	✓	✓	MAC for UAVs	✓
[43]	2017	Overview of the communication architecture as well as routing protocols for UAVs is presented	✗	✗	UAV communication architectures	✗

Reference	Year	Description	Classification of UAVs	Challenges	Topic discussed	Intel dec
[44]	2016	UAVs and UAVs related issues are discussed	✓	✓	UAV based IoTs	✗
Our work	–	Extensively surveys task assignment algorithms for UAVs along with task assignment considerations	✓	✓	Task Assignment in UAVs	✓

The authors in [36] reviewed the challenges encountered during the cooperation of UAVs with the Internet of Things (IoT) devices. A detailed study of security for critical UAV applications, such as denial of service attacks, man-in-the-middle attacks, and de-authentication attacks, was presented in [37]. Besides this, blockchain, software-defined networks, machine learning techniques, and edge computing have been studied as emerging technologies. Different features and provisions anticipated for networks of UAVs envisioned from the viewpoint of networking and communications were reported in [38]. UAV applications are classified based on their communication necessities, such as search and rescue, construction, coverage, and delivering goods. Existing UAV applications are categorized as short-long-range, high-low capacity, and real-time vs. delay tolerant. The exclusive features of UAV networks (such as high maneuverability dispersed UAV nodes and frequently changing network topology) present challenges during network design and routing. As a result, the classification of UAVs, the design of communications and applications, and an exhaustive review of the prevailing routing protocols proposed for UAVs were presented [39].

作者在[36]中回顾了无人机与物联网（IoT）设备合作过程中遇到的问题。在[37]中，对关键无人机应用的安全进行了详细研究，包括拒绝服务攻击、中间人攻击和去认证攻击。此外，区块链、软件定义网络、机器学习技术和边缘计算被研究为新兴技术。在[38]中，从网络和通信的角度报告了针对无人机网络预期的一些功能和规定。无人机应用根据其通信需求进行分类，例如搜索和救援、建筑、覆盖和送货。现有的无人机应用被分为短距离-长距离、高容量-低容量以及实时-延迟容忍。无人机网络（如高机动性分散无人机节点和频繁变化的网络拓扑）的独特特性在设计和路由过程中提出了挑战。因此，无人机分类、通信设计及应用，以及对为无人机提出的现有路由协议的全面审查都得到了介绍[39]。

A comprehensive study of UAV systems from a cyber-physical system (CPS) perspective was presented, considering three different CPS components [40]. Different channel classification for UAVs, measurement operations, and statistical channel models were studied and comprehensively surveyed [41]. It includes the following: (1) The UAV channel measuring operations in low-altitude platforms (LAPs) created on narrow-band or wide-band channels, low cost or low power channels, and broadly organized ground infrastructures. (2) The experimental methods for A2G and A2A propagation channels. (3) Finally, the channel modeling approaches for UAVs are classified as deterministic, stochastic, and geometric-stochastic models and reviewed extensively. A survey of state-of-the-art medium access control (MAC) protocols proposed for UAV networks was presented from the viewpoint of main features, working ideologies, benefits, and limitations [42]. An extensive analysis of the latest use of UAVs in different IoT devices is made, along with challenges faced and design requirements [43]. Decentralized architectures of multi-layer UAV communication with prospects are discussed in [44].

对无人机系统从网络物理系统（CPS）角度的全面研究被提出，考虑了三个不同的 CPS 组件[40]。研究了无人机、测量操作和统计信道模型的不同信道分类，并进行了全面调查[41]。包括以下内容：（1）在窄带或宽带信道、低成本或低功耗信道以及广泛组织的地面基础设施上创建的低空平台（LAPs）中的无人机信道测量操作。（2）A2G 和 A2A 传播信道的实验方法。（3）最后，将无人机信道建模方法分为确定性、随机性和几何-随机模型，并进行了广泛审查。从主要特征、工作理念、优势和局限性等方面对为无人机网络提出的最新中继接入控制（MAC）协议进行了调查[42]。对无人机在不同物联网设备中的最新应用进行了广泛分析，包括面临的挑战和设计要求[43]。在[44]中讨论了多层无人机通信的去中心化架构及其前景。

3. Considerations for UAV task assignment 无人机任务分配的考虑因素

Recently, UAVs have been widely deployed in many sectors due to the unprecedented progress in drone and wireless technologies. The proper deployment and operation of UAVs offers reliable and effective solutions to a variety of real-world scenarios. UAVs can be used as base stations (BSs) to collect and deliver on-demand wireless communications in anticipated regions. In particular, UAVs as flying BSs support and maintain connectivity between wireless networks. In comparison to traditional ground BSs, by using UAVs as aerial BSs, network performance can be improved by adjusting their heights, avoiding static and dynamic obstacles, providing line-of-sight (LoS) links to ground nodes, etc. [45], [46]. Owing to the attributes inherent in UAVs, such as maneuverability, flexibility, and adaptive altitude, they can excellently complement existing wireless and ad hoc networks. It can be observed that there has been a tremendous increase in the number of UAV-based applications. According to a TechSci study [47], it is expected that the tentative revenue from UAV-based applications will rise from 69 billion dollars in 2018 to 141 billion dollars in 2023. Different aspects affect the task assignment process in UAVs. Hence, the factors influencing the UAV task assignment are briefly studied in the subsections below.

近期，由于无人机和无线技术的空前进步，无人机在许多领域得到了广泛应用。无人机恰当的部署和运行为各种现实场景提供了可靠有效的解决方案。无人机可以作为基站（BS）在预期区域收集和提供按需无线通信。特别是，作为空中基站的无人机支持并维护无线网络之间的连接。与传统的地面基站相比，通过使用无人机作为空中基站，可以通过调整其高度、避免静态和动态障碍物、为地面节点提供视距（LoS）链路等方式来提高网络性能。[45], [46]。由于无人机固有的属性，如机动性、灵活性和自适应高度，它们可以很好地补充现有的无线和自组织网络。可以观察到，基于无人机应用的数量急剧增加。根据 TechSci 的研究[47]，预计基于无人机应用的收入将从 2018 年的 690 亿美元增长到 2023 年的 1410 亿美元。不同方面影响无人机任务分配过程。因此，以下小节简要研究了影响无人机任务分配的因素。

3.1. Single- and multi-UAV systems

3.1. 单机和多无人机系统

Previously, single UAVs were commonly used to achieve missions [48], [49]. Single and large UAVs are used in single UAV-based systems, which directly communicate with ground infrastructures. However, any problems with that one UAV can terminate the entire mission. Utilizing technological advancements and immense research, the integration of multiple small UAVs has been widely considered recently. Multiple UAV-based systems add reliability, multi-tasking ability, and survivability to wireless communications [50], [51]. A comparison of single- and multi-UAV operations is given in Table 2.

之前，单个无人机通常用于完成任务[48], [49]。单个和大型无人机用于基于单个无人机的系统中，它们直接与地面基础设施通信。然而，任何单个无人机的问题都可能终止整个任务。利用技术进步和大量研究，最近广泛考虑了多个小型无人机的集成。基于多个无人机的系统增加了无线通信的可靠性、多任务

能力和生存能力[50], [51]。表 2 给出了单无人机和多无人机操作的比较。

Table 2. Comparison of single- and multi-UAV operation.

Parameters	Single-UAV operation	Multi-UAV operation
Failure probability	High	Low
Survivability	Very low	High
Antenna used	Omni-directional	Directional
System reconfiguration	No	Yes
Complexity	Low	High
Cost	High	Low
Coverage	Limited	High
Coordination issues	Very low	High
Execution time	Slow	Fast

Furthermore, multi-UAV-based systems can achieve tasks in much less time and efficiently. In multi-UAV systems, UAVs are assigned either homogeneous or heterogeneous tasks. In both cases, each UAV requires a continually increasing number of high-level directing and planning competencies to accomplish missions. Most missions are very complex and need effective collaboration to achieve **mission objectives**. UAVs mainly rely on information from onboard sensors to effectively identify, classify, and select proper targets and determine an understandable sequence of choices.

此外，基于多无人机（UAV）的系统可以在更短的时间内高效地完成任务。在多 UAV 系统中，无人机被分配执行同质或异质任务。在两种情况下，每架无人机都需要不断增加的高级指导和规划能力来完成使命。大多数任务都非常复杂，需要有效的协作来实现任务目标。无人机主要依靠机载传感器的信息来有效地识别、分类和选择合适的靶标，并确定可理解的决策序列。

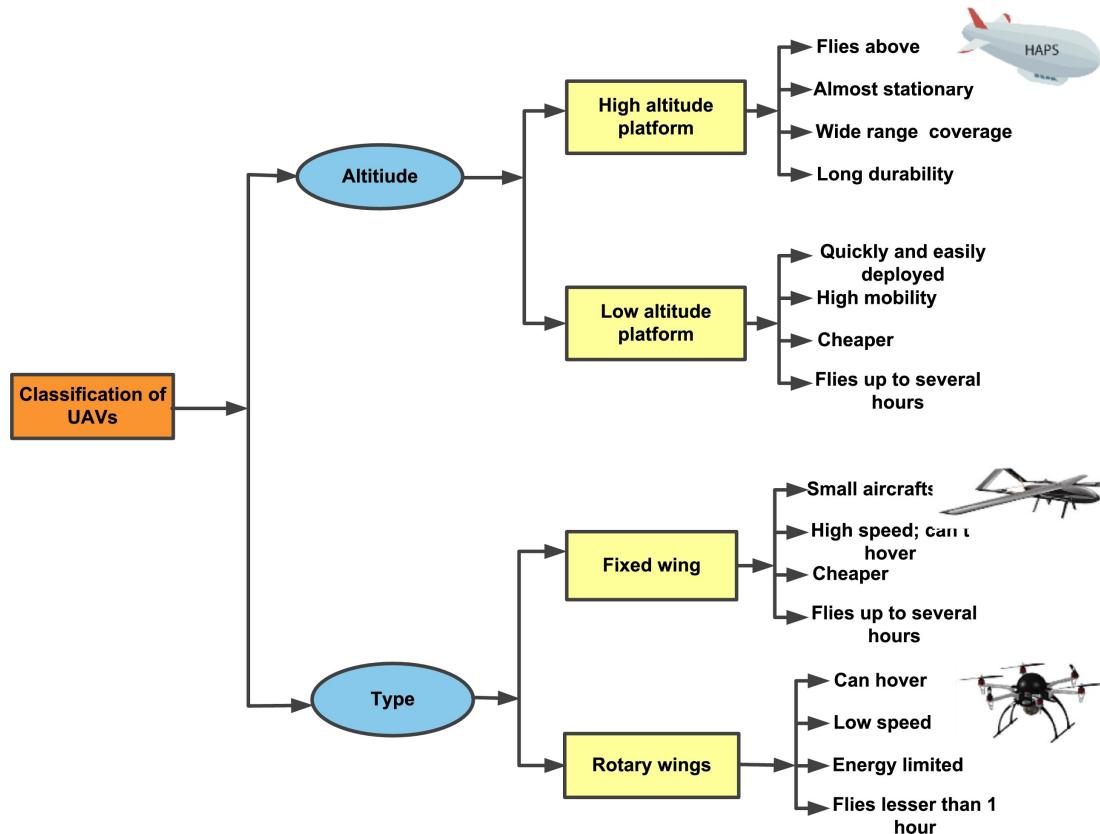
3.2. Classification of UAVs

3.2. 无人机分类

Depending on the application and mission objectives, an appropriate UAV type must be designated to meet several requirements and achieve the anticipated **quality of service** (QoS). UAV selection for a particular application must consider several factors, such as the capability of the UAV, maximum height it can fly up to, and available onboard devices. UAVs can thus be classified based on altitudes they can fly as high-altitude platforms (HAPs) and LAPs. HAPs can fly above 17 km, are almost stationary, have high coverage, and are durable.

Alternatively, LAPs have high mobility, can fly in the range of 10 m to a few kilometers, can fly for a few hours, and are cheaper. UAVs are also categorized based on type, such as fixed-wing and rotary-wing. In comparison to rotary-wing UAVs, fixed-wing UAVs, such as small aircrafts have more weight, higher speed, and must move forward to remain airborne. In contrast, rotary-wing UAVs can hover and stay static over a given area [52], [53]. An overview of UAV categorization based on types and altitudes with their functions and capabilities is shown in Fig. 4.

根据应用和任务目标，必须指定一种适当的无人机类型以满足多个要求并实现预期的服务质量 (QoS)。针对特定应用的无人机选择必须考虑几个因素，如无人机的性能、它能飞到的最大高度以及可用的机载设备。因此，无人机可以根据其飞行高度分为高空平台 (HAP) 和低空平台 (LAP)。HAP 可以飞行在 17 公里以上，几乎静止，覆盖范围广，耐用。相比之下，LAP 具有高机动性，可以在 10 米到几公里的范围内飞行，可以飞行几小时，且成本较低。无人机还可以根据类型分类，如固定翼和旋翼。与旋翼无人机相比，固定翼无人机，如小型飞机，重量更大，速度更快，必须向前移动才能保持空中飞行。相比之下，旋翼无人机可以在指定区域内悬停并保持静止[52], [53]。图 4 展示了基于类型和高度对无人机进行分类的概述，包括其功能和能力。



3.3. Background study 3.3. 背景研究

A task assignment is a combinatorial and open-loop optimization process that minimizes a predefined objective function by assigning single or [multiple agents](#) to multiple tasks. Task assignment is efficient if all the tasks are completed appropriately, satisfying the constraints, and is fundamental in many multi-agent-based applications. Only a few restrictions are optimized in most existing [research works](#), ignoring other factors that deteriorate network performance. In multi-agent systems, agents with distinct capabilities and types are selected to deal with heterogeneous tasks. While assigning tasks to the agents, the capabilities of agents can be an essential constraint. Another critical factor that determines the efficiency of task assignments is time constraints. Most of the practical applications anticipate real-time task accomplishment. During task execution, risks and uncertainties present in the environment provide yet other conditions for efficient task assignment algorithms.

一项任务分配是一个组合和开环优化过程，通过将单个或多个代理分配到多个任务来最小化预定义的目标函数。如果所有任务都得到适当完成，满足约束条件，则任务分配是高效的，这在许多基于多代理的应用中是基本的。大多数现有研究工作中只优化了少数限制，忽略了其他降低网络性能的因素。在多代

理系统中，选择具有不同能力和类型的代理来处理异构任务。在分配任务给代理时，代理的能力可能是一个基本约束。决定任务分配效率的另一个关键因素是时间约束。大多数实际应用都预计实时完成任务。在任务执行过程中，环境中存在的风险和不确定性为有效的任务分配算法提供了其他条件。

Task assignment has been studied for many other wireless networks in the literature. Satisfying decision theory [54] for balancing the necessities of the team with requests of individuals, [traveling salesman problem \(TSP\)](#) [55], [dynamic programming](#) [56], mixed-integer linear programming (MILP) [57], weapon target assignment (WTA) [58] are some [optimization algorithms](#) that have been applied to solve task assignment. Complex task assignment can be solved by using multi-objective genetic algorithms (GA) [59], [ant colony optimization](#) (ACO) [60], [reinforcement learning](#) (RL) [61] and [particle swarm optimization](#) (PSO) [62]. However, due to the inadequacies of constrained resources in agents, decisions are sometimes impacted, resulting in target misinterpretation. In such circumstances, the capability of agents to configure a prompt solution is highly desired. The ability to integrate ambiguities while executing robust missions to indeterminate actions is an important consideration in the design of task assigning algorithms. Meanwhile, evidence on the environment must be updated and react to the substantial alterations in the mission environment while properly rejecting confrontational incorrect information. Nevertheless, cooperation and collaboration among agents are most likely to achieve better network performance, which is challenging [63], [64], [65].

任务分配在文献中已被研究了许多其他无线网络。为了平衡团队的需求与个人的请求，满足决策理论 [54]，旅行商问题（TSP）[55]，动态规划[56]，混合整数线性规划（MILP）[57]，武器目标分配（WTA）[58]是一些应用于解决任务分配的优化算法。可以通过使用多目标遗传算法（GA）[59]，蚁群优化（ACO）[60]，强化学习（RL）[61]和粒子群优化（PSO）[62]来解决复杂任务分配。然而，由于代理中约束资源的不足，决策有时受到影响，导致目标误解。在这种情况下，代理配置快速解决方案的能力非常受欢迎。在执行鲁棒任务的同时整合模糊性，对不确定动作的能力是任务分配算法设计中一个重要的考虑因素。与此同时，必须更新关于环境的证据，并针对任务环境的实质性变化做出反应，同时适当拒绝对抗性的错误信息。然而，代理之间的合作和协作最有可能实现更好的网络性能，这具有挑战性 [63], [64], [65]。

To boost escalating UAV applications and continue their working reliability, task assignment algorithms are compulsory in deliberating the concerns of UAV communications. The maximum collaboration level is required during task assignment, where UAVs can promptly [share information](#), tasks and allot tasks to other UAVs with apposite [task scheduling](#) and arrangement. This level of cooperation and information sharing complicates the autonomous task assignment process. Task assignment algorithms for a swarm of UAVs have become the subject of much research, and many algorithms have been suggested for UAV task assignment. Hence, a brief review of all state-of-the-art task assignment algorithms proposed for multiple UAVs is presented in this paper, which will help researchers and engineers to explore this topic further.

为了提高不断增长的无人机应用并保持其工作可靠性，任务分配算法在考虑无人机通信问题时是必不可少的。在任务分配过程中需要达到最大协作水平，此时无人机可以迅速共享信息、任务，并通过适当的任务调度和安排将任务分配给其他无人机。这种合作和信息共享程度使得自主任务分配过程变得复杂。针对无人机群的任务分配算法已成为研究热点，许多算法已被提出用于无人机任务分配。因此，本文简要回顾了为多无人机提出的所有最先进的任务分配算法，这将有助于研究人员和工程师进一步探讨这一主题。

The symbols and acronyms used in this paper are listed alongside their descriptions in [Table 3](#).

该论文中使用的符号和缩写及其描述列于表 3 中。

[Table 3. List of notations and abbreviations.](#)

Symbol	Description
N_v	Number of UAVs
N_t	Number of targets
$X^{k_{ij}}$	Decision variable in the range of 0–1
k	Order of UAV
L^k_{ij}	Shortest path length
f^k_{ij}	Waiting time
$p_i(t)$	Survival probability
$S(u_t)$	Score of targets
r_t	Remaining targets at time t
m_t	Remaining weapons at time t
u_t	Number of targets to be hit at time t
λ	Time discount coefficient
$c_{ij}(x_i, p_i)$	Score function
t_u	Time required to complete allocated task
C_k^-	Minimal value at time k
\oplus	Exclusive OR
x_k	Decision variable at time k
$N_{\{Lim\}^k}$	Maximum flight path
$L_{\{Lim\}^k}$	Maximum limit of number of tasks
CPS	Cyber-physical system
LAPs	Low-altitude platforms
A2G	Air-to-ground
A2A	Air-to-air
MAC	Medium access control
LoS	Line-of-sight
BS	Base station
DE	Differential evolution

Symbol	Description
QoS	Quality of service
HAPs	High-altitude platforms
MEC	Mobile edge computing
SO	System orchestrator
VAIoTs	Value added IoTs
EAUS	Energy-aware UAV selection
DAUS	Delay-aware UAV selection
FTUS	Fair tradeoff UAV selection
IVHM	Integrated vehicle health management
RUL	Residual useful life
BCD	Block coordinate descent
EAT	Earliest available time
G2G	Ground-to-ground
TOT	Time over target
IoTs	Internet of things
FTA	Fault tree analysis
BSUM	Block successive upper-bound minimization
MILP	Mixed integer linear programming
TSP	Traveling salesman problem
WTA	Weapon target assignment
RL	Reinforcement learning
GA	Genetic algorithm
PSO	Particle swarm optimization
ACO	Ant colony optimization
DE	Differential evolution
ANN	Artificial neural network

4. Design issues of task assignment algorithms 任务分配算法的设计问题

Task assignment algorithms for UAVs face significant difficulties, such as unacceptable computation time for real-time implementation and degraded performance due to uncertain environmental aspects. This section addresses the primary design issues of task assignment algorithms for different mission planning systems for fleets of UAVs.

无人机的任务分配算法面临着巨大的困难，例如实时执行的计算时间不可接受，以及由于不确定的环境因素而导致性能下降。本节讨论无人机机队不同任务规划系统的任务分配算法的主要设计问题。

4.1. Multi-UAV system formation

4.1. 多无人机系统编队

Multiple UAVs should be systematically formed for the task assignment in numerous UAVs. Developing a multi-UAV system is required to accomplish a task cooperatively. Recently, the issues faced during the formation of multi-UAV systems are getting substantial research attention in many studies. The collaborative use of a multi-UAV system offers many advantages over a single UAV in large-scale applications, such as accomplishing complex and large-scaled tasks, improving the probability of successful task accomplishment even during the failure of some UAVs. Moreover, multi-UAV systems provide diverse competencies in various application circumstances. However, designing multi-UAV systems and using them cooperatively to achieve the mission objectives brings particular challenges and complexities [66], [67].

多架无人机应系统性地组成以完成众多无人机中的任务分配。开发多无人机系统以协同完成任务是必要的。最近，在许多研究中，多无人机系统形成过程中遇到的问题受到了实质性的研究关注。多无人机系统的协作使用在大型应用中比单无人机具有许多优势，例如完成复杂和大规模的任务，即使在某些无人机失败的情况下也能提高任务成功的概率。此外，多无人机系统在各种应用环境中提供多样化的能力。然而，设计和使用多无人机系统以协同实现任务目标带来了特定的挑战和复杂性[66]，[67]。

应系统地组建多架无人机，在众多无人机中完成任务分配。开发多无人机系统需要协同完成任务。近年来，多无人机系统组建过程中面临的问题受到许多研究的广泛关注。在大规模应用中，多无人机系统的协同使用比单架无人机具有许多优势，例如完成复杂和大规模的任务，即使在某些无人机发生故障的情况下也能提高成功完成任务的概率。此外，多无人机系统在不同的应用环境中提供不同的能力。然而，设计多无人机系统并协同使用它们来实现任务目标带来了特殊的挑战和复杂性[66]，[67]。

4.2. Computational complexity

4.2. 计算复杂度

Having autonomous capacity allows UAVs to execute assignments with negligible or no human interference. Cooperative UAVs are expected to outperform the sum of individual UAVs. The interdependency of multiple tasks and trajectory optimization determines the simultaneous operations of UAVs. To confirm the proper dependence among UAVs, the coordination of path plans and task assignments is required in real-time. Complexity is the focal aspect in designing cooperative UAV missions [68], [69].

具有自主能力使无人机能够执行任务，几乎无需或无需人类干预。预计协作无人机将优于单个无人机的总和。多个任务和轨迹优化的相互依赖性决定了无人机的协同作业。为了确认无人机之间的适当依赖关系，需要实时协调路径计划和任务分配。复杂性是设计协作无人机任务的关键方面[68]，[69]。

Along with the kinematic constraints of UAVs, factors such as problem size and level of cooperation expected in different types of mission operations induce complexities. Moreover, UAVs are expected to operate in harsh and critical environments, where real-time service is desired. In such scenarios, UAVs must handle the uncertainties and constraints of dynamic environments. UAVs must compute and make decisions at the local level to handle these adversities, which add to the complexity of UAV networks. Recently, [cloud computing](#) and [edge computing](#) techniques have been extensively used to address the computational complexities of UAV communication [70], [71].

随着无人机运动学约束，问题规模和不同类型任务操作中期望的合作水平等因素导致复杂性。此外，无人机预计将在恶劣和关键环境中运行，需要实时服务。在这种情况下，无人机必须处理动态环境的不可确定性和约束。无人机必须在本地级别进行计算和决策以应对这些挑战，这增加了无人机网络的复杂性。最近，云计算和边缘计算技术已被广泛用于解决无人机通信的计算复杂性[70]，[71]。

4.3. Collision-free UAV operation

4.3. 无碰撞无人机操作

The existence of multiple UAVs in a network surges the chance of collision. Collision among UAVs can occur when UAVs come across each other's way during their operation when they lack autonomy to handle obstacles and are time-constrained. Hence, one of the furthermost significant necessities in multi-UAV networks is that UAVs should not clash each other on their flight route and also their signal should not interfere with each other's. A number of [collision avoidance](#) mechanisms are studied and used in the literature [72]. Mixed-integer based model is studied to obtain the collision-free plan for multiple UAVs in [73]. Operating UAVs in different altitudes is considered to avoid collision among UAVs in [74]. Clustering can help to minimize collision to some extent. Another significant aspect in determining the collision-free UAV operation is the [trajectory planning](#).

网络中多个无人机（UAV）的存在增加了碰撞的机会。当无人机在操作过程中遇到彼此的路径，缺乏处理障碍物的自主性且时间受限时，无人机之间可能会发生碰撞。因此，在多无人机网络中，最迫切的需求之一是无人机在飞行路线上不应相互碰撞，并且它们的信号也不应相互干扰。文献[72]中研究了多种碰撞避免机制。文献[73]研究了基于混合整数模型，以获得多个无人机的无碰撞计划。文献[74]中考虑了在不同高度操作无人机以避免无人机之间的碰撞。聚类可以在一定程度上帮助减少碰撞。确定无碰撞无人机操作的另一重要方面是轨迹规划。

The existence of multiple UAVs in a network surges the chance of collision. Collision among UAVs can occur when UAVs come across each other's way during their operation when they lack the autonomy to handle obstacles and are time-constrained. Hence, one of the furthermost significant necessities in multi-UAV networks is that UAVs should not clash on their flight route and their signal should not interfere with each other's. Some [collision avoidance](#) mechanisms are studied and used in the literature [72]. Mixed-integer based model is studied to obtain the collision-free plan for multiple UAVs in [73]. Operating UAVs in different altitudes is considered to avoid collision among UAVs in [74]. Clustering can help to minimize collision to some extent. Another significant aspect in determining the collision-free UAV operation is the [trajectory planning](#).

网络中多个无人机（UAV）的存在增加了碰撞的机会。当无人机在操作过程中遇到彼此的路径，缺乏处理障碍物的自主性且时间受限时，无人机之间可能会发生碰撞。因此，在多无人机网络中，最迫切的需求之一是无人机在飞行路线上不应相撞，它们的信号也不应相互干扰。文献[72]中研究了并使用了一些碰撞避免机制。文献[73]中研究了基于混合整数模型，以获得多个无人机的无碰撞计划。文献[74]中考虑了在不同高度操作无人机以避免无人机之间的碰撞。聚类可以在一定程度上帮助减少碰撞。确定无碰撞无人机操作的另一重要方面是轨迹规划。

4.4. Backhaul network 4.4. 回程网络

Robust backhaul is crucial when UAVs are not fully autonomous. A backhaul network collects and disseminates information for effective UAV cooperation and decision-making processes. Information includes the status of flights, sensed data, and flight control. The limitations and requirements of backhaul links for the design and deployment of UAVs were studied in [75]. Similarly, in [76], backhaul and latency-aware UAV positioning with time complexity were discussed to calculate the optimal height. An aerial backhaul scheme was formulated to form a network with multi-hop backhaul in the sky [77]. All these studies support UAVs to create backhaul in a distributed manner. Mm-wave is a recent and widespread technique for enabling broadband backhaul in UAVs [78], [79].

稳健的回程对于无人机不完全自主运行至关重要。回程网络收集和传播信息，以实现有效的无人机合作和决策过程。信息包括飞行状态、感知数据和飞行控制。在[75]中研究了回程链路的限制和要求，以设计无人机部署。在[76]中，讨论了具有时间复杂度的回程和延迟感知的无人机定位，以计算最佳高度。在[77]中制定了一种空中回程方案，以在空中形成多跳回程网络。所有这些研究都支持无人机以分布式方式创建回程。毫米波是使无人机实现宽带回程的最近和广泛应用的技术[78]，[79]。

当无人机不完全自主时，强大的回程至关重要。回程网络收集和传播信息，以实现有效的无人机合作和决策过程。信息包括航班状态、传感数据和飞行控制。[75]研究了无人机设计和部署的回程链路的限制和要求。类似地，在[76]中，讨论了具有时间复杂度的回程和延迟感知无人机定位，以计算最佳高度。制定空中回传方案，形成空中多跳回传网络[77]。所有这些研究都支持无人机以分布式方式创建回程。毫米波是一种最新且广泛应用的技术，用于在无人机中实现宽带回程[78]、[79]。

4.5. Uniform load balancing

4.5. 均匀负载均衡

In multi-UAV operations, the uniform distribution of tasks among UAVs is necessary. Differential evolution (DE)-based load balancing mechanism has been discussed in [80], where accessing problem is modeled as a generalized assignment problem. Grounded on the indication of task load within a specific region of task execution, task assignment can be done approving even dissemination of tasks as far as possible. Another technique for load balancing is to assign a group of arbitrarily created non-intersecting coordinates. The load balancing technique using multi-criterion decision-making is studied [81]. The optimal user association to **balance load** using hybrid cognitive radio relay [82]. Task assignment methods can be helpful for load balancing in UAV-assisted IoT communication [83], [84].

在多 UAV 操作中，在 UAV 之间均匀分配任务是必要的。基于差分进化（DE）的负载均衡机制已在[80]中讨论，其中访问问题被建模为广义分配问题。基于任务执行特定区域内的任务负载指示，可以进行任务分配，甚至尽可能广泛地传播任务。另一种负载均衡技术是为任意创建的一组非相交坐标分配任务。使用多标准决策制定的负载均衡技术已在[81]中研究。使用混合认知无线电中继平衡负载的最佳用户关联已在[82]中研究。任务分配方法对于 UAV 辅助的物联网通信中的负载均衡可能是有帮助的[83]、[84]。

4.6. Antenna designs 4.6. 天线设计

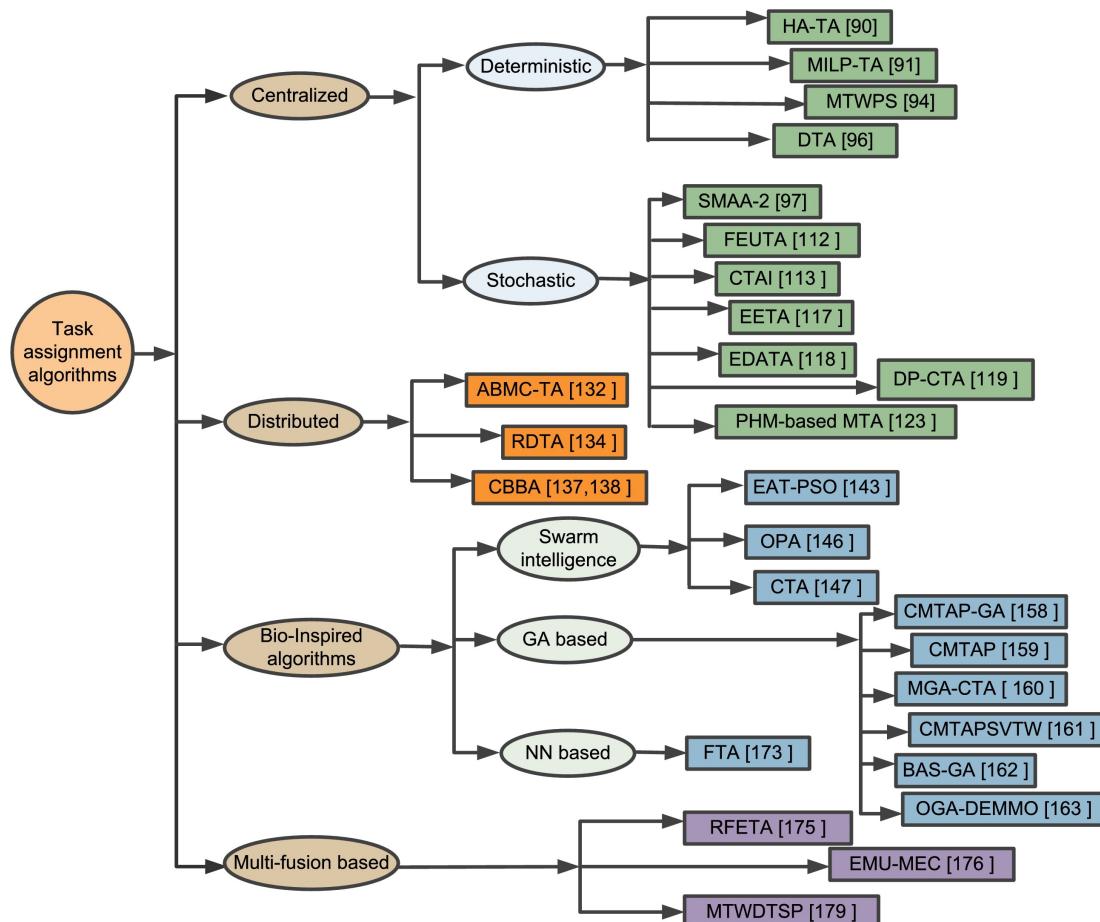
The design and number of antennas also determine the performance of UAVs, and hence they must be decided cautiously. UAVs essentially incorporate directional and **omnidirectional antennas**. In short, omnidirectional antennas perform better in environments with high mobility but face energy drain and security issues. In contrast, **directional antennas** have an improved transmission range but require complex protocols and exact antenna orientation algorithms [85], [86]. Also, different issues brought about by directional communication, such as head-of-line, deafness, and **hidden terminals** must be addressed. UAVs also use smart and adaptive antennas.

设计天线数量也决定了无人机性能，因此必须谨慎决定。无人机本质上包含定向和全向天线。简而言之，全向天线在高度移动的环境中表现更好，但面临能量消耗和安全问题。相比之下，定向天线具有更远的传输范围，但需要复杂的协议和精确的天线方向算法[85], [86]。此外，定向通信带来的不同问题，如头节点、聋哑和隐藏终端等问题也必须解决。无人机还使用智能和自适应天线。

5. Task assignment algorithms 任务分配算法

The task assignment of UAVs is performed to minimize the entire cost by assigning UAVs to accomplish a number of tasks. A UAV can be assigned a single task or more. The issues faced by task assignment algorithms are computational complexities, task coupling, problem size, time constraints, and heterogeneity. Keeping this in mind, different task assignment algorithms have been designed for different application-specific UAV operations. We classify task assignment algorithms into four categories: centralized, distributed, bio-inspired, and multi-fusion, as illustrated in Fig. 5. The basic operating principles of each algorithm type are briefly discussed along with their advantages, disadvantages, and possible improvements in the subsections below.

无人机的任务分配是通过分配无人机完成多项任务来最小化整体成本。无人机可以被分配一个或多个任务。任务分配算法面临的问题是计算复杂性、任务耦合、问题规模、时间限制和异构性。考虑到这一点，针对不同的特定应用无人机操作设计了不同的任务分配算法。我们将任务分配算法分为四类：集中式、分布式、仿生和多重融合，如图5所示。下面的小节将简要讨论每种算法类型的基本工作原理及其优点、缺点和可能的改进。



5.1. Centralized task assignment algorithms 集中式任务分配算法

Centralized task assignment algorithms require a central planner that [gathers information](#) from all UAVs, calculates the optimum strategy, and passes that information among the UAVs. The central planner can be a ground station receiving information from all the UAVs, calculating optimal plan, and informing all UAVs about the plan. In some cases, one of the UAVs can also act as a planner. Information sharing is straightforward in centralized task assignment schemes, where each UAV communicates with the central planning agent. Some cooperative algorithms proposed for task assignment are [87], [88], [89].

集中式任务分配算法需要一个中央规划器，该规划器从所有无人机收集信息，计算最佳策略，并在无人机之间传递该信息。中央规划器可以是地面站，接收来自所有无人机的信息，计算最优计划，并将该计划通知所有无人机。在某些情况下，其中一架无人机还可以充当规划者。在集中式任务分配方案中，信息共享非常简单，每个无人机都与中央规划代理进行通信。提出的一些用于任务分配的协作算法是[87]、[88]、[89]。

5.1.1. Health-Aware Task Assignment (HA-TA)

5.1.1.1. 健康意识任务分配 (HA-TA)

- A robust decision-making process that improves the group's functioning reliability and competencies of distributed and self-directed UAVs using improved **self-awareness systems** and adaptive mission planning was studied in [90].
- The task planner manages the list of tasks in the task assignment component, which decides and selects existing UAVs, which can perform tasks depending on the information about the tasks and proficiencies of the UAVs.
- After assigning tasks, they are referred to the trajectory manager, which plans possible trajectories for each UAV.
- The results from the trajectory manager are given as a series of waypoints to each UAV.
- [UAV controllers](#) compute the definite controls desired to follow the waypoints using these path coordinates.
- The **health management** issue is focused on using a response mechanism of the performance model used by the task assignment algorithm.
- [90]研究了一种稳健的决策过程，该过程使用改进的自我意识系统和自适应任务规划来提高分布式和自主无人机的团队功能可靠性和能力。
- **任务规划器**管理任务分配组件中的任务列表，该组件决定并选择现有的无人机，这些无人机可以根据有关无人机的任务和熟练程度的信息来执行任务。
- 分配任务后，它们被转交给**轨迹管理器**，轨迹管理器为每架无人机规划可能的轨迹。
- 轨迹管理器的结果作为每架无人机的一系列**航路点**给出。无人机控制器使用这些路径坐标计算遵循航路点所需的明确控制。
- **健康管理**问题的重点是使用任务分配算法所使用的性能模型的响应机制。

The developed **health-aware task assignment algorithm** was validated to be operative through simulation and real aeronautical experimentations. The preliminary outcomes were very favorable; nevertheless, many can be achieved in the health management sector, accounting for sensor performance and control [actuator](#) failure modes. Additionally, a robust performance in the face of uncertainty was achieved.

通过模拟和真实航空实验证了所开发的健康感知任务分配算法的有效性。初步结果非常有利；然而，在健康管理领域可以实现许多目标，考虑传感器性能和控制[执行器](#)故障模式。此外，在面对不确定性时也取得了稳健的表现。

- Advantages: Improves operational reliabilities and capabilities through **adaptive task assignment systems** and **system awareness**; demonstrated effective performance through simulation and experiments.
- 优点：通过自适应任务分配系统和系统意识提高操作可靠性和能力；通过模拟和实验证明了有效的性能。
- Disadvantages: Randomly occurring failures provide a level of uncertainty; uncertainty present at all planning stages due to **incomplete knowledge** at the flight plan level, health of sensors during task assignment, and enduring maintenance at the mission planning level.
- 缺点：随机发生的故障提供了一定程度的不确定性；由于飞行计划层面的**知识不完整**、任务分配期间传感器的健康状况以及任务规划层面的持久维护，所有规划阶段都存在不确定性。
- Possible enhancements: The amount and quality of feedback information can be improved, and a sophisticated **stochastic model** of health states like energy usage and sensors' performance can be embedded to deal with uncertainties.
- 可能的增强功能：可以提高反馈信息的数量和质量，并且可以嵌入能源使用和传感器性能等健康状态的复杂**随机模型**来处理不确定性。

5.1.2. MILP-TA

The issues of task assignment and optimal formulation to solve combined multi-assignment for a widespread search **munition** area were addressed in [91]. **MILP** can assign infeasible tasks by adding time to UAV paths due to timing constraints. It makes use of the discrete **approximation** of real-world scenarios. Ammunitions are essential to explore, categorize, attack, and confirm the demolition of achievable goals. Information on the target area is assumed to be communicated between all elements of the UAV swarm. A formulation based on **MILP** consists of optimization function, upper bound and lower bound on variables and constraints using the variables. A UAV is allowed to visit any target only twice to prevent looping. UAVs can visit a sink once to find new targets before reassignment occurs. This helps to avoid inconsistencies in UAVs entering and leaving the sink. Existing **path planning** and lengthening algorithms were detailed in [92], [93]. Through this formulation, the UAV flight path varies to ensure that the timing constraints are fulfilled and varying **task completion times** are integrated.

[91]解决了任务分配和优化制定问题，以解决广泛的搜索**弹药**区域的组合多重分配问题。由于时间限制，**MILP**可以通过增加无人机路径的时间来分配不可行的任务。它利用现实世界场景的离散近似。**弹药**对于探索、分类、攻击和确认摧毁可实现的目标至关重要。假设有关目标区域的信息在无人机群的所有单元之间进行通信。基于**MILP**的公式由优化函数、变量的上限和下限以及使用变量的约束组成。无人机只能访问任何目标两次，以防止循环。无人机可以在重新分配之前访问接收器一次以寻找新目标。这有助于避免无人机进入和离开接收器的不一致。现有的路径规划和延长算法在[92]、[93]中有详细介绍。通过这个公式，无人机的飞行路径会发生变化，以确保满足时间约束并整合不同的**任务完成时间**。

- Advantages: Useful in offline task assignment calculations; provides optimal solution for UAV groups with combined tasks with timing and task order limitations.
- 优点：在离线任务分配计算中很有用；为具有时间和任务顺序限制的组合任务的无人机群提供最佳解决方案。
- Disadvantages: Makes a discrete representation of real-world problems, which makes solutions impractical; solution requires high **computation time** and makes unreliable for real-time use.
- 缺点：对现实世界问题进行离散表示，导致解决方案不切实际；该解决方案需要大量**计算时间**并且对于实时使用来说不可靠。
- Possible enhancements: Kinematic constraints of UAVs should be focused; aspects of complex and dynamic environment such as obstacles, uncertainties, and **wind speed** must be considered.

- 可能的改进：应关注无人机的运动约束；必须考虑复杂动态环境的各个方面，例如障碍物、不确定性和风速。

5.1.3. Modified Two-Part Wolf Pack Search (MTWPS)

5.1.3. 改进的两部分狼群搜索 (MTWPS)

MTWPS, a **combinatorial optimization** model-based task assignment using **graph and optimization methods**, was studied in [94]. An easy computing function is used for large UAVs and target sizes, and to solve the time-sensitive uncertainty; a practical online hierarchical **planning algorithm** is used. MTWPS includes traditional offline centralized situations and online capability with time-sensitive uncertainty. A number of UAVs are assigned to classify, reply, and verify tasks on targets sequentially. UAVs have different flight heights to avoid a collision. The **combinatorial optimization problem** is modeled as follows:

MTWPS 是一种使用图和优化方法的基于组合优化模型的任务分配，在[94]中进行了研究。针对大型无人机和目标尺寸使用简单的计算函数，解决时间敏感的不确定性；使用实用的在线分层规划算法。

MTWPS 包括传统的离线集中情况和具有时间敏感不确定性的在线能力。多架无人机被分配顺序对目标进行分类、回复和验证任务。无人机具有不同的飞行高度以避免碰撞。组合优化问题建模如下：

- Advantages: Deals with online uncertainties such as communication issues, UAV malfunction, and time-sensitive target problem; easy computing used to lessen simulation time for large-scale UAV networks.
- 优点：处理通信问题、无人机故障、时间敏感目标问题等线上不确定性问题；简单的计算用于减少大型无人机网络的仿真时间。
- Limitations: A UAV's waiting time depends on the performance of other UAVs, which makes waiting time complicated for large-scale networks; the **deadlock** problem is ignored.
- 局限性：一架无人机的等待时间取决于其他无人机的性能，这使得大规模网络的等待时间变得复杂；**死锁**问题被忽略。
- Possible enhancements: Different types of deadlocks require better handling, which can be focused on in the future; threats and other risks present in the stochastic environment should also be focused on.
- 可能的改进：不同类型的死锁需要更好的处理，可以在以后重点关注；还应关注随机环境中存在的威胁和其他风险。

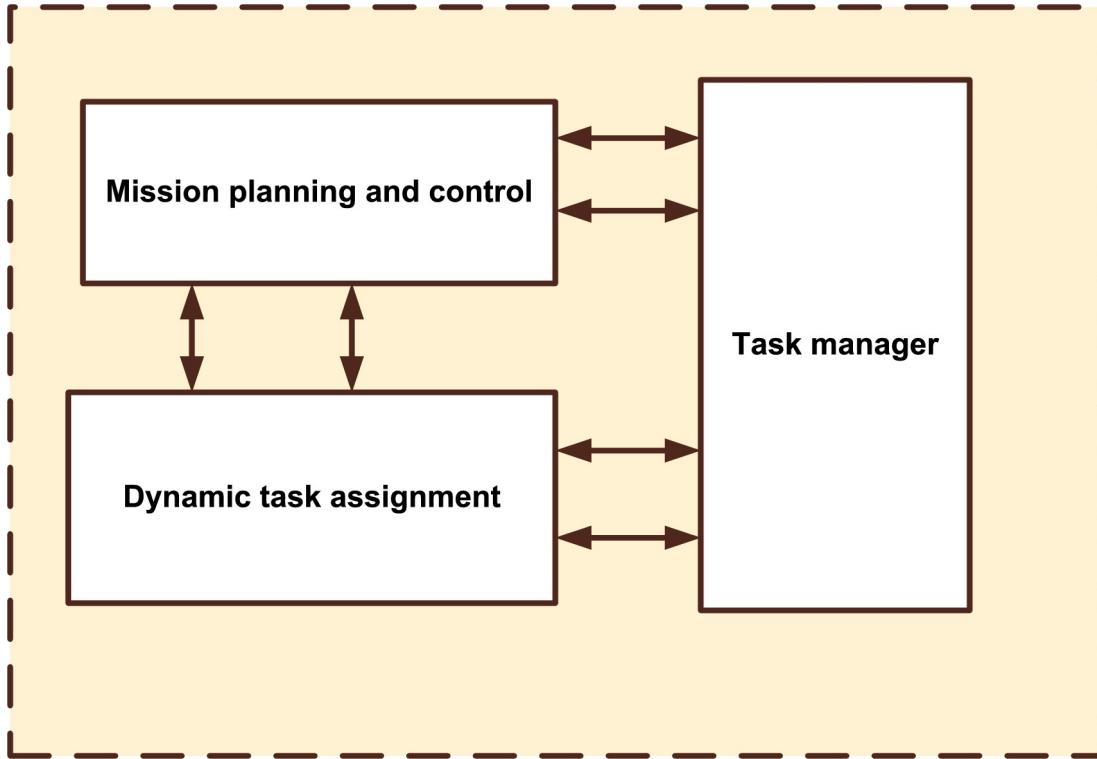
5.1.4. Dynamic Task Assignment (DTA)

5.1.4. 动态任务分配 (DTA)

A mission planning and task assignment framework for the coordination of self-controlled UAVs to improve the robustness and scalability of multi-UAV operations was studied in [96]. An outline of the proposed framework is shown in Fig. 6. The main goal of the battlefield scenario is to eliminate enemy agents; hence, the maximum mission time includes search and attacks. For probable actions to attack enemies, n agents are used, and each agent is given a specific number for identification, i.e., `agent_id`. Agent information includes position coordinates, energy level, and payload status. However, enemy agents do not have prior knowledge of the size and location of UAVs. Because of the energy constraints in UAVs, a cost-effective approach is required to allocate tasks to an optimal UAV set. The cost of every attack action is determined using the position of the identified target, the position of available agents, available **battery**, and available payload.

文献[96]研究了一种用于协调自控无人机的任务规划和任务分配框架，以提高多无人机操作的鲁棒性和可扩展性。所提出的框架的概要如图 6 所示。战场场景的主要目标是消灭敌方特工；因此，最大任务时间包括搜索和攻击。对于攻击敌人的可能动作，使用 n 个代理，并且每个代理被赋予一个特定的编号用于

识别，即 `agent_id`。代理信息包括位置坐标、能量水平和负载状态。然而，敌方特工事先并不了解无人机的大小和位置。由于无人机的能量限制，需要一种经济高效的方法来将任务分配给最佳无人机组。每次攻击行动的成本是根据已识别目标的位置、可用代理的位置、可用电池和可用有效负载来确定的。



- Advantages: Time efficiency, scalability, and robustness are achieved; achieves the mission goal with minimum cost.
• 优点：实现了时间效率、可扩展性和鲁棒性；以最小的成本实现任务目标。
- Limitations: Does not consider the uncertainties and aspects of dynamic environments; preliminary information regarding the size and location of enemy UAVs is unknown.
• 局限性：没有考虑动态环境的不确定性和方面；有关敌方无人机的大小和位置的初步信息尚不清楚。
- Possible enhancements: Minimizing the combination of all costs is desired, requiring a solver algorithm; threats and uncertainties of the complex dynamic environments need consideration.
• 可能的增强功能：需要最小化所有成本的组合，需要求解器算法；需要考虑复杂动态环境的威胁和不确定性。

5.1.5. Stochastic Multi-Criteria Acceptability Analysis (SMAA-2)

5.1.5. 随机多标准可接受性分析 (SMAA-2)

In [97], a group of UAVs was used to complete attacking tasks on ground targets in an uncertain environment. UAVs with different capabilities were allotted tasks to attack before starting the mission. Uncertain information led the criteria values of the task assignment process to be random, and the criteria weights were not known accurately. A novel task assignment process that relies on SMAA was presented in some previous works [98], [99], [100], [101], [102], [103], [104], [105], [106], [107], [108], [109], [110], [111] to deal with the abovementioned problem. Target attack tasks were assigned to multiple UAVs.

文献[97]利用一组无人机在不确定环境下完成对地面目标的攻击任务。在开始任务之前，具有不同能力的无人机被分配了攻击任务。信息的不确定性导致任务分配过程的准则取值具有随机性，且准则权重无法准确得知。之前的一些工作[98]、[99]、[100]、[101]、[102]、[103]、[104]、[105]、[106]中提出了一种依赖 SMAA 的新颖任务分配过程。[107]、[108]、[109]、[110]、[111]处理上述内容问题。目标攻击任务分配给多架无人机。

- Advantages: Uncertainties are analyzed; finds satisfactory solution even in severe situations.
优点：分析不确定性；即使在严峻的情况下也能找到令人满意的解决方案。
- Disadvantages: Sensor error or target movement leads to unclear localization; probability of UAVs being destroyed while handling attacks remains.
缺点：传感器误差或目标移动导致定位不清晰；无人机在应对攻击时被摧毁的可能性仍然存在。
- Possible enhancements: Threats present in the defense system of targets can be made more apparent; a confidence factor is introduced to judge the accuracy of criteria values.
可能的增强：目标防御系统中存在的威胁可以更加明显；引入置信因子来判断标准值的准确性。

5.1.6. Filter-Embedded UAV Task Assignment (FEUTA)

5.1.6. 嵌入式滤波器无人机任务分配 (FEUTA)

The classical task assignment problem was modified to form coordinated teams of UAVs to reduce the effect of noise in SA on the solution [112]. The degree of change during the reassignment process was limited appropriately. The measured approach performs reassessments at the degree of updated knowledge to immediately react to substantial fluctuations observed in the environment. Additionally, FEUTA embeds a sophisticated filtering operation during the task assignment process relying on modified weight coefficients that depend on environmental variations.

经典的任务分配问题被修改为形成协调的无人机团队，以减少 SA 中的噪声对解决方案的影响[112]。调动过程中的变动程度受到适当限制。测量方法以更新知识的程度执行重新分配，以立即对环境中观察到的大幅波动做出反应。此外，FEUTA 在任务分配过程中嵌入了复杂的过滤操作，依赖于取决于环境变化的修改权重系数。

- Advantages: Condenses the consequence of noise caused due to high-frequency on the planner; mitigates the effects of noise.
优点：浓缩了高频带来的噪声对规划者的影响；减轻噪音的影响。
- Limitations: Ignores the aspects such as threats and uncertainties, which makes it unreliable for the real environment; accepts noise entirely.
局限性：忽略了威胁和不确定性等方面，导致其对于真实环境不可靠；完全接受噪音。
- Possible enhancements: The importance of robust methods proposed for FEUTA and their connection to noise rejection algorithms must be investigated.
可能的增强功能：必须研究为 FEUTA 提出的鲁棒方法的重要性及其与噪声抑制算法的联系。

5.1.7. Coordinated Target Assignment and Intercept (CTAI)

5.1.7. 协调目标分配和拦截（CTAI）

Cooperative control solutions assign UAVs to pass through target locations known to have many threats [113].

The complete problem is divided into a target assignment, coordinated UAV, path planning, practicable trajectory design, and asymptotic trajectory based on hierarchical coordinated control.

Different heights are considered for UAVs to avoid collisions.

Satisfying decision theory [54], control settings, [114], [115], and multi-agent interactions [116] are used.

Two levels are considered for UAV cooperation. There should be negotiation with UAVs and target assignment vector at a higher level to allocate UAVs to targets. After that, UAVs of each team coordinate to identify feasible team time of target (TOT). Coordination of TOT is resolved through the encapsulation of important myopic information for optimization.

协作控制解决方案分配无人机穿过已知存在许多威胁的目标位置[113]。

完整的问题分为目标分配、无人机协调、路径规划、可行轨迹设计以及基于分层协调控制的渐近轨迹。

无人机考虑不同的高度以避免碰撞。

使用满足决策理论[54]、控制设置、[114]、[115]和多主体交互[116]。

无人机合作考虑两个层面。应与无人机和目标分配向量在更高级别进行协商，以将无人机分配给目标。之后，每个团队的无人机进行协调以确定可行的团队目标时间（TOT）。TOT的协调是通过封装重要的近视信息进行优化来解决的。

- Advantages: Decomposition of motion planning issues helps obtain a near-real-time solution; risk is mitigated by increasing distance between UAV and threats.
- 优点：运动规划问题的分解有助于获得近实时的解决方案；通过增加无人机与威胁之间的距离来降低风险。
- Limitations: Tracking error increases due to disturbances; only static threats are considered, and thus the results obtained are suboptimal.
- 局限性：由于干扰，跟踪误差增大；仅考虑静态威胁，因此获得的结果不是最优的。
- Possible enhancements: Timing of target intersection can be considered; aspects of the dynamic, complex 3D environment such as wind speed, risks need consideration.
- 可能的增强功能：可以考虑目标交叉的时间；动态、复杂的3D环境如风速、风险等方面需要考虑。

5.1.8. Energy-Efficient Task Assignment (EETA)

5.1.8. 节能任务分配 (EETA)

To minimize the power consumption of ground devices and UAVs, assigning tasks and allocating resources were studied for UAV-assisted mobile edge computing (MEC) systems [117]. Energy minimization for IoT devices and UAVs through combined optimization of trajectory, appropriate task assignment, and resource assignment. It is quite challenging to solve the non-convex structure imposed by the proposed system using conventional convex optimization. Hence, block successive upper-bound minimization (BSUM) is used.

为了最大限度地减少地面设备和无人机的功耗，研究了无人机辅助移动边缘计算（MEC）系统的任务分配和资源分配[117]。通过组合优化轨迹、适当的任务分配和资源分配，实现物联网设备和无人机的能源最小化。使用传统的凸优化来解决所提出的系统所施加的非凸结构是相当具有挑战性的。因此，使用块连续上限最小化（BSUM）。

- Advantages: Lessens energy depletion, and optimizes task offloading, resource sharing, and trajectory without increasing delay; minimizes UAV's propulsion and computation power.

- 优点：减少能量消耗，在不增加延迟的情况下优化任务卸载、资源共享和轨迹；最大限度地减少无人机的推进力和计算能力。
- Limitations: Single UAV is used, which increases the risk of mission failure.
- 局限性：使用单架无人机，增加了任务失败的风险。
- Possible enhancements: UAVs' kinematic constraints such as external forces, motions, and physical limitations of the dynamic environment such as threats and uncertainties must be considered during the assignment process.
- 可能的增强功能：在任务过程中必须考虑无人机的运动学约束（例如外力、运动）以及动态环境的物理限制（例如威胁和不确定性）。

5.1.9. Energy and Delay Aware Task Assignment (EDATA)

5.1.9. 能量和延迟感知任务分配 (EDATA)

A robust system orchestrator (SO) to achieve value-added IoTs (VAIoTS) attained by incorporating UAVs with IoT payloads was presented in [118]. The flight of UAVs is organized using a central, SO that holds the complete information of UAVs, current positions, energy levels, flight operations, and their onboard IoT devices. SO employs energy-aware UAV selection (EAUS), delay-aware UAV selection (DAUS), and fair tradeoff UAV selection (FTUS). EAUS aims to reduce the total energy depletion by the UAVs, DAUS seeks to decrease the UAV time of operation, and FTUS aims to find a fair solution between the two conflicting objectives of energy and time. The power and time required for traveling, sensing, processing, and data transmission are considered and optimized.

[118]中提出了一种强大的系统协调器（SO），通过将无人机与物联网有效负载相结合来实现增值物联网（VAIoTS）。无人机的飞行是使用中央 SO 来组织的，该 SO 保存着无人机的完整信息、当前位置、能量水平、飞行操作及其机载物联网设备。SO 采用能量感知无人机选择 (EAUS)、延迟感知无人机选择 (DAUS) 和公平权衡无人机选择 (FTUS)。EAUS 旨在减少无人机的总能量消耗，DAUS 旨在减少无人机的运行时间，FTUS 旨在在能源和时间这两个相互冲突的目标之间找到公平的解决方案。考虑并优化了行进、传感、处理和数据传输所需的功率和时间。

- Advantages: Energy consumption, operational time, and fairness of the system are optimized jointly; environmental effects on flights and performance are considered.
- 优点：系统的能耗、运行时间、公平性共同优化；考虑了环境对飞行和性能的影响。
- Limitations: GPS localization errors and signal losses exist; increase in execution time for FTUS, which impacts the performance of the model.
- 局限性：存在GPS定位误差和信号丢失；FTUS 的执行时间增加，这会影响模型的性能。
- Possible enhancements: Cases of accidents and UAV failures must be considered; the modeling part could be improved concerning the energy consumed by GPS in errors and signal losses.
- 可能的增强功能：必须考虑事故和无人机故障的情况；建模部分可以针对GPS误差和信号损失所消耗的能量进行改进。

5.1.10. Dynamic Programming-based Cooperative Task Assignment (DP-CTA)

5.1.10. 基于动态规划的协作任务分配 (DP-CTA)

DP-CTA, which accounts for cooperation among UAVs rather than just coordination in adversarial environments, was studied [119]. As discussed in [120], [121], [122], DP-CTA relies on weapon target assignment, where weapons are assigned to targets for optimizing the mission objectives.

Two approximation methods of DP have been developed for large-scale networks.

- A one-step method is responsible for generating a cooperative solution quickly,
- and a two-step method generates an optimal solution with minimum possible computation time.
- Using a one-step look-ahead at stage and state, maximizes the expression as

[119] 研究了 DP-CTA，它考虑了无人机之间的合作，而不仅仅是对抗环境中的协调。正如[120]、[121]、[122]中所讨论的，DP-CTA 依赖于武器目标分配，其中武器被分配到目标以优化任务目标。已经针对大规模网络开发了两种 DP 近似方法。

- 一步法负责快速生成协作解，而两步法则以尽可能最小的计算时间生成最优解。
- 使用阶段和状态的一步前瞻，最大化表达式：

$$\max_{u_t, |u_t| \leq m_t} \{S(u_t) + \lambda \bar{J}_{t+1}(r_t - u_t, m_t - |u_t|)\} \quad t \in \{0, \dots, N-1\}$$

- Advantages: Generates cooperative and optimal solution quickly; recovers planning of real-world air operations providing improvements.
- 优点：快速生成协同最优解；恢复现实世界空中作战的规划并提供改进。
- Limitations: Computation time grows with the number of targets, and the uncertainties of real complex environment are ignored.
- 局限性：计算时间随着目标数量的增加而增长，并且忽略了真实复杂环境的不确定性。
- Possible enhancements: Constraints of the real dynamic environment such as sudden threats, effects of wind, and temperature on UAV flights must be focused on.
- 可能的改进：必须关注真实动态环境的约束，例如突发威胁、风的影响、温度对无人机飞行的影响。

5.1.11. Prognostics and Health Monitoring (PHM)-based Multi-UAV Task Assignment (PHM-based MTA)

5.1.11. 基于预测和健康监测 (PHM) 的多无人机任务分配（基于 PHM 的 MTA）

Applying [integrated vehicle health management](#) (IVHM), based on PHM for multiple UAVs, was considered in [123] to achieve objectives with negligible spontaneous disruptions. IVHM is the ability of the system to evaluate current as well as future system health state and incorporate that with the structure of existing resources and demands. PHM is the capability to measure different states of health, forecasting forthcoming failures and predicting the probable RUL of the system based on measurements. Information from PHM consists of residual useful life (RUL) approximations. System-level RUL is designed for decision support to assign a group of UAVs to weighted tasks. RUL is determined through [fault tree](#) analysis (FTA), which is then served by a distribution function from the [probability distribution function](#) concerning time and probability of failure in critical scenarios. FTA is a widely used failure analysis technique where graphs explore potential causes of undesired states.

[123] 考虑对多架无人机应用基于 PHM 的集成车辆健康管理(IVHM)，以实现自发干扰可忽略不计的目标。IVHM 是系统评估当前和未来系统健康状态并将其与现有资源和需求结构相结合的能力。PHM 能够测量不同的健康状态、预测即将发生的故障并根据测量结果预测系统可能的 RUL。来自 PHM 的信息

由剩余使用寿命 (RUL) 近似值组成。系统级 RUL 旨在支持决策支持，将一组无人机分配给加权任务。RUL 通过故障树分析 (FTA) 确定，然后由关于关键场景中故障时间和概率的概率分布函数的分布函数提供服务。FTA 是一种广泛使用的故障分析技术，通过图表探索不良状态的潜在原因。

- Advantages: Applies system-level RUL methodology to accomplish task assignment including failure probability.
- 优点：应用系统级 RUL 方法来完成包括故障概率在内的任务分配。
- Limitations: It may not be feasible to know the probability of failure in advance; threats and uncertainties of real stochastic environments are ignored.
- 局限性：提前知道失败的概率可能不可行；真实随机环境的威胁和不确定性被忽略。
- Possible enhancements: PHM-based TA for extensive haul task reaction of algorithm during failure requires further investigation; consideration of aspects of real environment such as wind, temperature, threats is a must.
- 可能的增强功能：基于 PHM 的 TA 在故障期间算法的广泛拖运任务反应需要进一步研究；必须考虑真实环境的各个方面，例如风、温度、威胁。

5.2. Distributed task assignment algorithms 分布式任务分配算法

Centralized task assignments are sometimes not feasible for UAV swarms due to underlying communication restrictions, robustness concerns during inconsistencies, and scalability. Computation requirement gets impractical once the number of agents grows, which is also a limiting factor for centralized algorithms. Distributed methods can be helpful in such situations to minimize these complications. Distributed task assignment algorithms primarily focus on market-based procedures, i.e., contract network algorithms and auction algorithms [124], [125], [126], [127]. However, decentralized algorithms require proper coordination to improve overall performance [128], [129], [130], [131]. The fundamental problem is that accomplishing harmonization requires UAVs to interchange large amounts of environmental information, existing states, and forthcoming purposes. Communication on this scale is not always possible and increases UAVs' visibility to threats. Some recent decentralized approaches to overcome the issues of centralized task assignment on each UAV are discussed in the following subsections. Information sharing is a vital part of decentralized task assignment algorithms.

由于潜在的通信限制、不一致期间的鲁棒性问题以及可扩展性，集中式任务分配有时对于无人机群来说是不可行的。一旦代理数量增加，计算需求就变得不切实际，这也是中心化算法的限制因素。在这种情况下，分布式方法有助于最大限度地减少这些并发症。分布式任务分配算法主要关注基于市场的程序，即合约网络算法和拍卖算法[124]，[125]，[126]，[127]。然而，去中心化算法需要适当的协调来提高整体性能[128]，[129]，[130]，[131]。根本问题是，实现协调需要无人机交换大量环境信息、现有状态和即将实现的目的。这种规模的通信并不总是可行，并且增加了无人机对威胁的可见性。以下小节将讨论最近的一些分散方法来克服每个无人机上集中任务分配的问题。信息共享是去中心化任务分配算法的重要组成部分。

5.2.1. Auction-based Multiple Constraints Task Assignment (ABMC-TA)

5.2.1. 基于拍卖的多重约束任务分配 (ABMC-TA)

An auction-based task assignment algorithm for assigning dynamic tasks to UAVs using multi-layer cost calculation was studied in [132]. The cost computation method divides the computation cost into four layers concerning the four constraint types. UAVs cooperate in a surveillance task considering four features: sensor type (C), *field of view* (F), lowest distinct pixels (P), and resolution (R). UAVs that can meet these features are

selected when a new task appears. In addition, the task has a specific time window that specifies the start time for the earliest, latest, and total time required to accomplish that particular task. An auction-based task assignment algorithm is studied in [133], where UAVs are deployed as traps.

[132]研究了一种基于拍卖的任务分配算法，该算法使用多层成本计算将动态任务分配给无人机。成本计算方法将计算成本分为涉及四种约束类型的四层。无人机在执行监视任务时要考虑四个特征：传感器类型 (C)、视场 (F)、最低明显像素 (P) 和分辨率 (R)。当新的任务出现时，就会选择能够满足这些特征的无人机。此外，任务还有一个特定的时间窗口，指定完成该特定任务所需的最早、最晚开始时间和总时间。[133]研究了基于拍卖的任务分配算法，其中无人机被部署为陷阱。

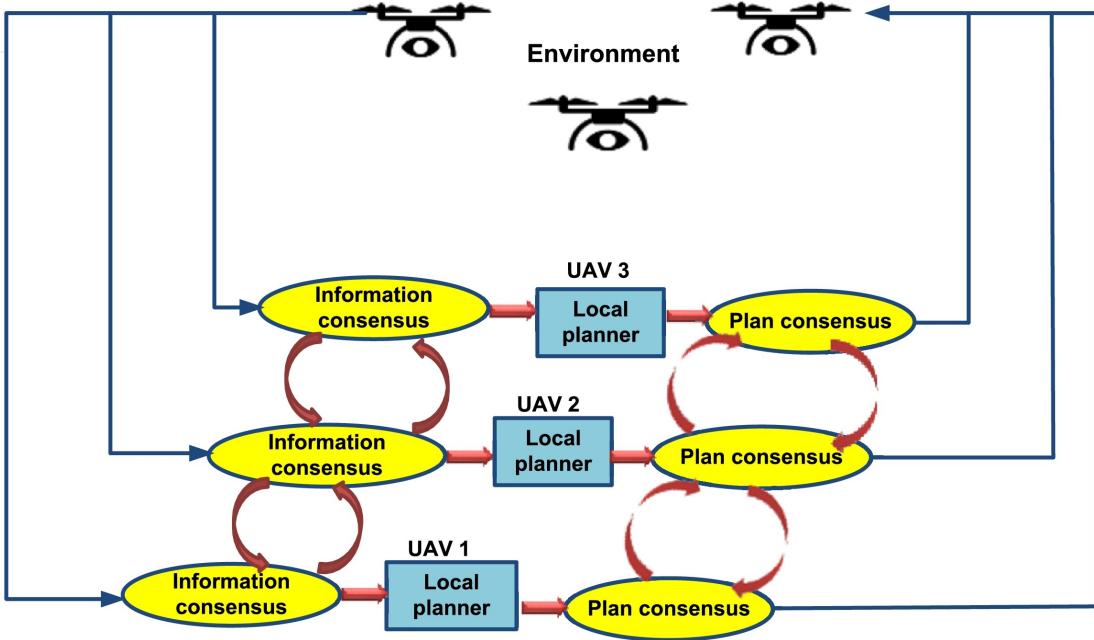
- Advantages: Solves multi-constraint task assignment issues efficiently considering the constraints of sensors, time, danger, and fuel.
优点：考虑传感器、时间、危险和燃料的约束，有效解决多约束任务分配问题。
- Limitations: 2D environment with static obstacles considered for implementation which cannot justify the constraints of real dynamic environment; kinematics of UAV such as motion and impacts of external forces on its flight path needs consideration.
局限性：考虑实施时具有静态障碍物的二维环境，无法证明真实动态环境的约束；需要考虑无人机的运动学，例如运动和外力对其飞行路径的影响。
- Possible enhancements: Complexities imposed by real 3D environment need to be considered to confirm its practical validity; more experimental study and investigation are required to verify the efficiency of the proposed method in real environment.
可能的增强功能：需要考虑真实 3D 环境带来的复杂性，以确认其实际有效性；需要更多的实验研究和调查来验证所提出的方法在实际环境中的有效性。

5.2.2. Robust Decentralized Task Assignment (RDTA)

5.2.2. 鲁棒的去中心化任务分配 (RDTA)

Decentralized task assignment for a swarm of cooperative UAVs, which performs well for sparse networks, was studied [134]. A set of waypoints is identified for UAVs, and a matrix gives their locations. A vector provides the waypoint values. A matrix characterizes the capabilities of UAVs; signifies a UAV that can handle tasks with waypoint w. Two significant phases, information consensus, and planning, are used, as shown in Fig. 7. In the first phase, UAV communication improves the consistency of the data. However, due to limitations on communication resulting from noise and slow convergence, consensus cannot be reached. In this case, the second phase, i.e., planning, must be used. RDTA uses a modified planning phase to eliminate potential conflicts [135]. In stage 1, UAVs use updated information to generate a set of candidate plans. Then, the petal algorithm is used, as discussed in [136]. In stage 2, UAVs develop the ultimate non-conflicting goal based on the candidate plan.

- 研究了一群协作无人机的分散任务分配，它在稀疏网络中表现良好[134]。
- 为无人机识别一组航路点，并用一个矩阵给出它们的位置。
- 矢量提供航路点值。矩阵描述了无人机的能力；w 表示可以处理航路点的任务的无人机。
- 如图 7 所示，使用了两个重要的阶段：信息共识和规划。
- 第一阶段，无人机通信提高了数据的一致性。然而，由于噪声和收敛速度慢造成的通信限制，无法达成共识。在这种情况下，必须使用第二阶段，即规划。
- RDTA 使用修改后的规划阶段来消除潜在的冲突[135]。
- 在第一阶段，无人机使用更新的信息来生成一组候选计划。
 - 然后，使用花瓣算法，如[136]中讨论的。
- 在第二阶段，无人机根据候选计划制定最终的不冲突目标。



- Advantages: “No Fly Zones” are considered; a modified candidate plan selection algorithm is used to improve performance and increase the robustness to the inconsistencies.
- 优点：考虑“禁飞区”；修改后的候选计划选择算法用于提高性能并增加对不一致的鲁棒性。
- Disadvantages: Does not consider the heterogeneity of UAVs and targets and the constraints of the dynamic environment such as threats and effects of external forces (wind, temperature, etc.) on UAVs performance.
- 缺点：没有考虑无人机和目标的异质性以及外力（风、温度等）对无人机性能的威胁和影响等动态环境的约束。
- Possible enhancements: Conflicts arising due to low convergence need to be eliminated; heterogeneity of UAVs and other environmental aspects such as threats and uncertainties must be considered.
- 可能的改进：需要消除由于低收敛性而产生的冲突；必须考虑无人机的异质性以及威胁和不确定性等其他环境因素。

5.2.3. Consensus-based Bundle Algorithm (CBBA)

5.2.3. 基于共识的捆绑算法 (CBBA)

A conflict-free task assignment to resolve two real UAV operating complications, i.e., collision-free paths and churning behavior of UAV flight path, was considered in [137], [138]. CBBA was extended to account for obstacles and mitigate churning to reduce the algorithm's sensitivity to noise with the minimum computational burden. For a given task and agent, CBBA task assignment can be derived as

$$\max \sum_{i=1}^{N_u} \left(\sum_{j=1}^{N_t} c_{ij} (x_i, p_i) x_{ij} \right)$$

[137]、[138]考虑了无冲突任务分配来解决两个实际的无人机操作复杂性，即无碰撞路径和无人机飞行路径的搅动行为。CBBA 被扩展以考虑障碍并减轻搅动，从而以最小的计算负担降低算法对噪声的敏感性。对于给定的任务和代理，CBBA 任务分配可以导出为

$$\max \sum_{i=1}^{N_u} \left(\sum_{j=1}^{N_t} c_{ij} (x_i, p_i) x_{ij} \right)$$

在哪里 $\mathbf{x}_{ij}=1$ 如果 A_i 被分配给任务 j 而 $\mathbf{x}_{ij} \in \{0,1\}^{N_t}$ 是第 j 个元素为的向量 \mathbf{x}_{ij} , 和 $c_{ij}(x_i, p_i)$ 是作业 x_i 和路径 p_i 的得分函数。[139]还研究了基于 CBBA 的无人机群两层任务分配。

- Advantages: Handles operational complexities of multi-UAVs; provides reliable, conflict-free UAV task assignments.
- 优点: 处理多无人机操作复杂性; 提供可靠、无冲突的无人机任务分配。
- Limitations: All targets may not be discovered; dynamic threats and uncertainties are ignored.
- 局限性: 可能无法发现所有目标; 动态威胁和不确定性被忽略。
- Possible enhancements: Kinematic and dynamic UAV constraints on air and ground can be incorporated; UAVs need to be penalized for massive heading changes.
- 可能的增强功能: 可以纳入空中和地面无人机的运动和动态约束; 无人机需要因大幅航向变化而受到惩罚。

5.3. Bio-inspired task assignment algorithms 仿生任务分配算法

Bio-inspired algorithms originate from mimicking biological behavior based on their analyzing ability to deal with problems. These algorithms leave out the process of building difficult environmental models, and recommend a strong [searching algorithm](#) to meet the objective firmly. In our study, we have further classified the bio-inspired algorithms into swarm intelligence-based algorithms, GA-based algorithms, and [artificial neural network](#) (ANN)-based algorithms. On the other hand, [evolutionary algorithms](#) are stochastic methods used to find optimal solutions, and they have been applied in a variety of [optimization problems](#) [140], [141], [142]. Motivated by the nature of biological organisms, individuals learn and adapt to environmental situations for survival. The fitness function evaluates and decides which individuals are suited for the next generation in each iteration.

仿生算法源于基于其处理问题的分析能力来模仿生物行为。这些算法省略了构建困难的环境模型的过程，并推荐强大的搜索算法来坚定地实现目标。在我们的研究中，我们将仿生算法进一步分类为基于群体智能的算法、基于遗传算法的算法和基于人工神经网络（ANN）的算法。另一方面，进化算法是用于寻找最优解的随机方法，它们已应用于各种优化问题[140]、[141]、[142]。在生物有机体本质的激励下，个体学习并适应环境状况以求生存。适应度函数在每次迭代中评估并决定哪些个体适合下一代。

Swarm intelligence-based task assignment

基于群体智能的任务分配

Swarm intelligence is the communal behavior of self-organized and distributed particles generally witnessed in nature. These algorithms deploy a search-focused approach, where every particle operates independently. Later on, they collaborate with surrounding neighbors to explore the environment. Two different phases are involved: exploration and search. While exploring, particles sense data and then broadcast the sensed data to their neighbors through different communication channels. The data are then received by other agents of the swarm. In the search phase, every agent uses its specific data and the data obtained from its neighbors to discover a favorable direction to travel. [ACO](#) [60] and [PSO](#) [62] are some commonly used swarm intelligence-based algorithms for task assignment in the literature.

群体智能是自然界中普遍存在的自组织和分布式粒子的集体行为。这些算法采用以搜索为中心的方法，其中每个粒子独立运行。随后，他们与周围的邻居合作探索环境。涉及两个不同的阶段：探索和搜索。在探索时，粒子感知数据，然后通过不同的通信渠道将感知到的数据广播给邻居。然后数据被群体的其

他代理接收。在搜索阶段，每个代理使用其特定数据和从邻居获得的数据来发现有利的行进方向。ACO [60]和PSO [62]是文献中一些常用的基于群体智能的任务分配算法。

5.3.1. Earliest available time with PSO (EAT-PSO)

5.3.1.1. PSO 的最早可用时间 (EAT-PSO)

A task assignment algorithm for UAVs to operate in an [indoor environment](#) has been studied [143]. A [mathematical problem](#) model and a heuristic approach to solve the problems faced by indoor UAV operation are suggested. For real-time actions, rapid response to indeterminate actions requires a practicable schedule. EAT is incorporated with [PSO](#) to find the optimum plan with a concise calculation time. Indistinguishable UAVs with cameras and substance handling tools capable of handling the desired tasks in an indoor environment were studied. UAVs can take more than one task during flight. The framework in [144] was upgraded by fitting its solvability. The scheduler allocates tasks to UAVs, and the time to start a consigned task for every UAV, flight path, hover time, waiting time, and recharge schedule are all planned. PSO-based cooperative task assignment and path planning is presented in [145].

人们已经研究了无人机在[室内环境](#)中运行的任务分配算法[143]。提出了解决室内无人机操作所面临问题的数学问题模型和启发式方法。对于实时行动，对不确定行动的快速响应需要一个可行的时间表。EAT与PSO相结合，以简洁的计算时间找到最佳计划。研究了具有摄像头和物质处理工具的、无法区分的无人机，这些无人机能够在室内环境中处理所需的任务。无人机在飞行过程中可以执行多项任务。[\[144\]](#)中的框架通过拟合其可解性进行了升级。调度器给无人机分配任务，每架无人机启动委托任务的时间、飞行路径、悬停时间、等待时间、充电时间表等都已规划好。[\[145\]](#)中提出了基于PSO的协作任务分配和路径规划。

- Advantages: Achieves a minimum total make span, forms a seamless UAV operation schedule, and short computation time in finding a high-quality [feasible solution](#).
- 优点：实现最小的总制造周期，形成无缝的无人机操作计划，并缩短找到高质量[可行解决方案](#)的计算时间。
- Limitations: Allows a single UAV to occupy a position at a particular time, ignoring uncertain events and obstacles.
- 局限性：允许单架无人机在特定时间占据某个位置，忽略不确定的事件和障碍物。
- Possible enhancements: An anti-collision refinement phase can be incorporated, a detailed study on the distributed control of UAVs can be conducted, and the robustness of the system can be assessed.
- 可能的增强功能：可以纳入防碰撞细化阶段，可以对无人机的分布式控制进行详细研究，并可以评估系统的鲁棒性。

5.3.2. Orchard Picking Algorithms (OPA)

5.3.2.1. 果园采摘算法 (OPA)

Influenced by the garnering proficiency of farmers, a task assignment for multiple UAVs was presented in [146]. The nearest neighbor, which is deliberated with minimum possible neighboring distance as a pointer, rapidly resolves the optimum arrangement of numerous tasks for collective accomplishment. Every target that must be destroyed necessitates various ammunition, and a collaborative task assignment model for heterogeneous UAVs is established accordingly. Three different types of task sets, i.e., [reconnaissance](#) (D), attack (A), and evaluation (V), are considered. Specifies the load task set of UAVs, where subscripts T and U indicate belonging to the task and UAV, respectively. Depending on the [geometric relation](#) between two triples, i.e., the ammunition required to

destroy the target entirely and the capacity of UAVs to load this ammunition, it is decided whether tasks are fully executed. If T and U denote the sets of targets and UAVs, respectively, and, where V is the number of types of missiles, the total number of type missiles for targets to be destroyed is given as

受农民收割能力的影响, [146]提出了多无人机的任务分配。最近邻以最小可能的邻近距离为指针, 快速解决众多任务的最佳安排, 以实现集体完成。每一个需要摧毁的目标都需要不同的弹药, 并据此建立了异构无人机的协同任务分配模型。考虑三种不同类型的任务集, 即侦察 (D)、攻击 (A) 和评估 (V)。指定无人机的负载任务集, 下标 T 和 U 分别表示属于任务和无人机。根据两个三元组之间的几何关系, 即完全摧毁目标所需的弹药和无人机装载该弹药的能力, 来决定任务是否完全执行。如果 D 和 U 分别表示目标和无人机的集合, 其中 V 是导弹类型的数量, 则要摧毁的目标类型导弹的总数为

- Advantages: Works with short execution time, highly flexible, strongly robust, and scalable; quickly finds optimal solution; avoids difficult **cost function** based on the tasks, and has high calculation efficiency.
- 优点: 执行时间短、灵活性高、鲁棒性强、可扩展; 快速找到最优解; 避免了基于任务的复杂**成本函数**, 计算效率高。
- Limitations: Does not consider the constraints of the actual battlefield; threats and uncertainties that may arise in the dynamic environment are ignored.
- 局限性: 没有考虑实际战场的限制; 动态环境中可能出现的威胁和不确定性被忽略。
- Possible enhancements: Adaptive approach of cooperative task assignment can be introduced to balance the load of UAVs and adapt to real-time battlefield situations.
- 可能的增强功能: 可以引入协作任务分配的自适应方法来平衡无人机的负载并适应实时战场情况。

5.3.3. Cooperative Task Assignment (CTA)

5.3.3. 合作任务分配 (CTA)

Cooperative task assignment for UAVs in obstacle-based environments was presented in [147]. The upgraded ACO-based optimization and **Hungarian algorithm** are collectively used for multi-UAV formation. A battlefield environment is considered, where UAVs fly from different positions, search targets in that area, and destroy them. If $x_{i,j} \in [0,1]$ is the decision variable, $x_{i,j}=1$ when the UAV is assigned a task, and $x_{i,j}=0$ otherwise. Cooperative task assignment and path planning for multi-UAVs using K-means algorithm and hybrid optimization algorithms is also studied in [148].

[147]中提出了基于障碍的环境中无人机的协作任务分配。升级后的基于ACO的优化和**匈牙利算法**共同用于多无人机编队。考虑战场环境, 无人机从不同位置飞行, 搜索该区域的目标并摧毁它们。如果 $x_{i,j} \in [0,1]$ 是决策变量, $x_{i,j}=1$ 当无人机被分配任务时, 以及 $x_{i,j}=0$ 否则。文献[148]还研究了使用K-means算法和混合优化算法的多无人机协作任务分配和路径规划。

- Advantages: Efficiently solves considerate task assignment problems of UAVs under multiple circumstances and constraints; fast convergence and a highly stable iterative curve are assured.
- 优点: 高效解决多环境、多约束下的无人机任务分配问题; 确保快速收敛和高度稳定的迭代曲线。
- Limitations: Focuses on flight length only and ignores other constraints of task assignment; threats and uncertainties of the real environment are ignored, making the solution impractical.
- 局限性: 仅关注飞行长度, 忽略任务分配的其他约束; 现实环境的威胁和不确定性被忽视, 使得解决方案不切实际。
- Possible enhancements: Estimation of cost function could be adjusted considering the flight mission; in an actual battlefield environment, target values vary, and distribution of UAV firepower must be considered.

- 可能的改进：可以考虑飞行任务来调整成本函数的估计；在实际战场环境中，目标值各不相同，必须考虑无人机火力分布。

GA-based task assignment 基于 GA 的任务分配

Recently, [GA](#) has a wide range of applications and it is demonstrated to be appropriate for task assignment problems [149], [150], [151], [152]. The central idea of GA is the survival of the fittest using good genetic factor (i.e., efficient assignment) to the succeeding generation. It also produces new [search space](#) combining different genes. A GA-based approach for assigning [processing tasks](#) to engines spontaneously is studied in [153]. Some other task assignment algorithms using GA are studied as well [154], [155], [156], [157].

最近，[遗传算法](#)具有广泛的应用，并且它被证明适用于任务分配问题[149]，[150]，[151]，[152]。遗传算法的中心思想是利用良好的遗传因素（即有效分配）给下一代，适者生存。它还产生结合不同基因的新[搜索空间](#)。[153]研究了一种基于 GA 的方法，用于自发地将[处理任务](#)分配给引擎。还研究了一些使用 GA 的其他任务分配算法[154]、[155]、[156]、[157]。

5.3.4. Cooperative Multiple Task Assignment Problem with GA (CMTAP-GA)

5.3.4. 遗传算法的协作多任务分配问题 (CMTAP-GA)

The combined issue of task assignment and path planning in the form of a graph was presented in [158]. The CMTAP is suggested for circumstances concerning a group of different types of UAVs. A Dubins car model-based motion planning is used to consider each vehicle's specific constraint's minimum turn radius. Consuming a determinate set for defining the visit angle of a UAV over a target, the combined problem of task assignment and path optimization was posed as a graph. This novel method results in suboptimal route assignments. Refining the visitation angle [discretization](#) results in an upgraded solution. Owing to the computational complications of the subsequent combinatorial optimization problem, GA for stochastic exploration of the space of solutions was proposed. Two different cases of UAVs [group configuration](#) were illustrated: homogeneous, where all UAVs are indistinguishable and heterogeneous, with varying competencies of working and kinematic limitations.

[158]中以图表的形式提出了任务分配和路径规划的组合问题。CMTAP 建议用于涉及一组不同类型无人机的情况。基于杜宾斯汽车模型的运动规划用于考虑每辆车的特定约束的最小转弯半径。使用确定集来定义无人机在目标上的访问角度，将任务分配和路径优化的组合问题呈现为图表。这种新颖的方法会导致路线分配不理想。改进访问角度离散化会产生升级的解决方案。由于后续组合优化问题的计算复杂性，提出了对解空间进行随机探索的遗传算法。说明了无人机组配置的两种不同情况：同类，其中所有无人机都无法区分和异构，具有不同的工作能力和运动学限制。

- Advantages: Promptly offers feasible task assignment solutions and real-time implementation for high dimensions; considers task priority and coordination, time constraints, and trajectory.
- 优点：针对高维度，及时提供可行的任务分配方案并实时执行；考虑任务优先级和协调、时间限制和轨迹。
- Disadvantages: Performance degrades if the search space is harsh; it ignores the aspects of real dynamic environments, which makes the proposed method impractical in real-time implementation.
- 缺点：如果搜索空间严酷，性能会下降；它忽略了真实动态环境的各个方面，这使得所提出的方法在实时实施中不切实际。
- Possible enhancements: Proper [synchronization](#) must be considered to maintain coordination among UAVs; constraints of the dynamic environment such as threats, uncertainties, and external forces and their impacts on UAV's motion need to be investigated.

- 可能的增强功能：必须考虑适当的同步以保持无人机之间的协调；需要研究威胁、不确定性和外力等动态环境的约束及其对无人机运动的影响。

5.3.5. Cooperative Multiple Task Assignment Problem (CMTAP)

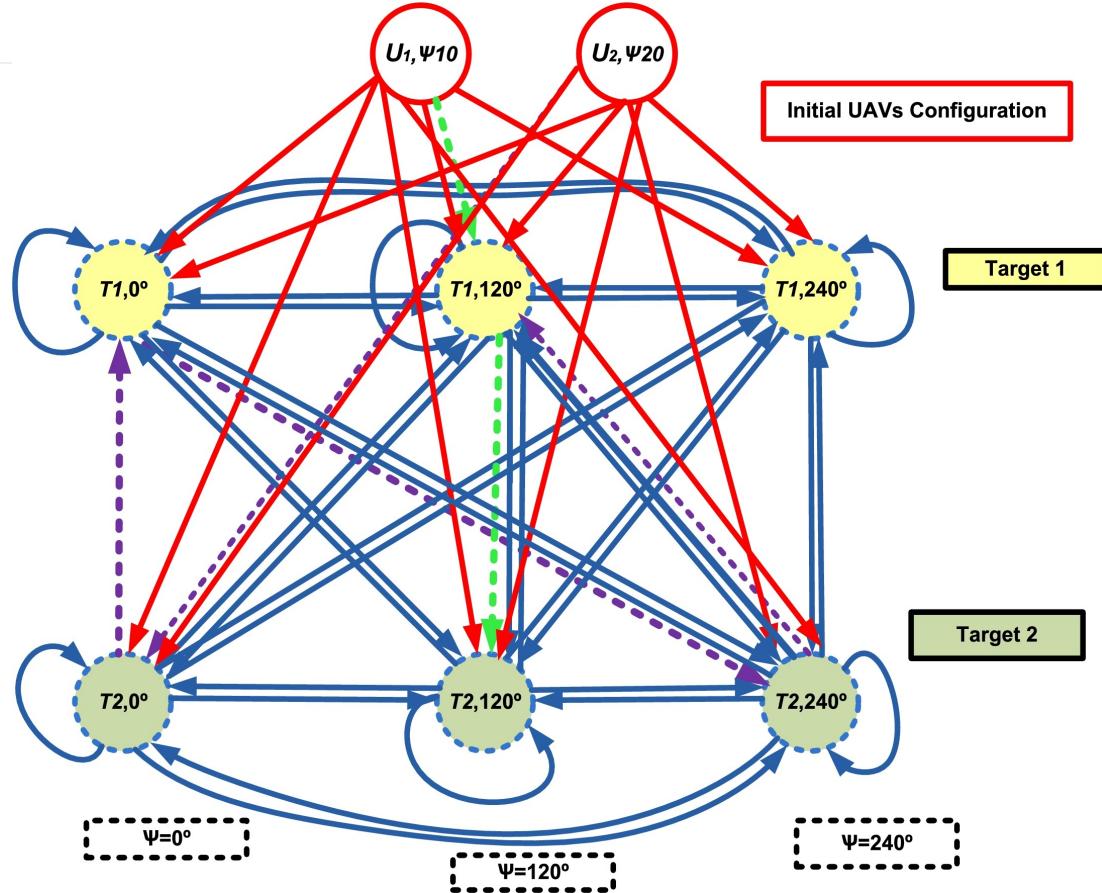
5.3.5。协作多任务分配问题 (CMTAP)

Assigning cooperative UAVs for multiple tasks on multiple targets was proposed as a combinatorial optimization problem [159]. The authors used GA to account for distinctive necessities, such as task preference and harmonization, timing constraints, and flight restrictions. A matrix exemplification of GA chromosomes simplifies the encoding procedure and the application of genetic operatives. The set of tasks is performed by the teams of the UAVs on each target, i.e., classify, attack, verify. Each task requires execution after a successful target classification from the given observation ability. It is assumed that the probability of achieving the assigned task if the physical requirements are met sums to one. If not, the assignment algorithm is conducted again.

为多个目标上的多个任务分配协作无人机被提出作为组合优化问题[159]。作者使用遗传算法来考虑独特的必要性，例如任务偏好和协调、时间限制和飞行限制。GA 染色体的矩阵示例简化了编码过程和遗传操作的应用。这组任务由无人机团队对每个目标执行，即分类、攻击、验证。每个任务都需要在根据给定的观察能力成功进行目标分类后执行。假设如果满足物理要求，则完成指定任务的概率总计为 1。如果没有，则再次进行分配算法。

Fig. 8 presents a graph representation of the CMTAP solution with two UAVs, $U=\{U_1, U_2\}$ and two targets $T=\{T_1, T_2\}$ to perform three tasks $N_m=3$. The green line indicates the path of the first UAV, U_1 , starting from the initial point, visiting the target, T_1 , moving $\psi=120^\circ$, and completing its assignment visiting T_2 . However, the purple path shows the routes of U_2 visiting T_2 with $\psi=0^\circ$ and proceeding towards T_1 , returning to T_2 with $\psi=240^\circ$, and flying to T_1 with $\psi=240^\circ$.

图 8 展示了使用两架无人机的 CMTAP 解决方案的图形表示， $U=\{U_1, U_2\}$ 和两个目标 $T=\{T_1, T_2\}$ 执行三项任务 $N_m=3$ 。绿线表示第一架无人机的路径 U_1 ，从初始点出发，访问目标点 T_1 ，移动 $\psi=120^\circ$ ，并完成其任务访问 T_2 。然而，紫色路径显示了 U_2 来访 T_2 和 $\psi=0^\circ$ 并朝着 T_1 ，返回到 T_2 和 $\psi=240^\circ$ ，并飞往 T_1 和 $\psi=240^\circ$ 。



- Advantages: Solves computational complexity for classical combinatorial optimization methods with higher efficiency and considers task priority and timing constraints.
- 优点: 以更高的效率解决经典组合优化方法的计算复杂性，并考虑任务优先级和时序约束。
- Disadvantages: Does not consider the dynamic aspects of the environment such as impacts of threats, uncertainties, and external forces on UAVs flight; heterogeneity of tasks, targets, and UAVs are ignored.
- 缺点: 没有考虑环境的动态方面，如威胁、不确定性、外力对无人机飞行的影响；任务、目标和无人机的异质性被忽略。
- Possible enhancements: Moving, sudden targets, and actual environmental scenarios must be considered; computational efficiency needs to be realized to achieve a real-time solution.
- 可能的增强功能: 必须考虑移动、突发目标和实际环境场景；需要提高计算效率才能实现实时解决方案。

5.3.6. Modified GA-based Cooperative Task Assignment (MGA-CTA)

5.3.6. 改进的基于 GA 的协作任务分配 (MGA-CTA)

Task assignment for heterogeneous multiple UAVs, concerned with cooperative decision-making and control, was studied in [160]. In comparison to previous GA-based algorithms, genetic factors of chromosomes were distinct with regard to the tasks to be accomplished on targets in this work. In addition, a mirror demonstration of UAVs for expressing inadequate resources of UAVs was used. Information on UAV resources is maintained and updated in a timely manner to process distinct types of genes. A task assignment that assigns multiple UAVs to several tasks on various targets is a possible solution.

[160]研究了异构多无人机的任务分配，涉及协作决策和控制。与之前基于遗传算法的算法相比，染色体的遗传因素在这项工作中要在目标上完成的任务方面是不同的。此外，还通过无人机镜像演示来表达无人机资源不足的情况。及时维护和更新无人机资源信息，以处理不同类型的基因。将多架无人机分配给不同目标上的多项任务是一种可能的解决方案。

- Advantages: Considers and addresses the heterogeneity of UAVs, targets, and environments; mirror representation to deal with the limited resources.
优点：考虑并解决无人机、目标和环境的异构性；镜像表示来处理有限的资源。
- Limitations: Only one kind of resource, i.e., weapon, is considered; stationary targets considered in simulation, weapons are used in one way only.
局限性：只考虑一种资源，即武器；在模拟中考虑固定目标时，武器仅以一种方式使用。
- Possible enhancements: Different operational and kinematic constraints of UAV must be considered; aspects of real dynamic environments such as threats, uncertainties, and other external forces and their impacts need to be analyzed.
可能的增强功能：必须考虑无人机的不同操作和运动学约束；需要分析真实动态环境的各个方面，例如威胁、不确定性和其他外部力量及其影响。

5.3.7. Cooperative Multiple Task Assignment Problem with Stochastic Velocities and Time Windows (CMTAPSVTW)

5.3.7. 具有随机速度和时间窗的协作多任务分配问题 (CMTAPSVTW)

A unique improved GA is anticipated to acquire a competent solution for realistic mission environments [161]. UAVs with superior abilities and some limitations are considered for task consignment. The model of Dubins car is accepted to create UAV flight paths along with two extra flights to coordinate the paths and create actual paths. 独特的改进型遗传算法预计将为现实任务环境提供有效的解决方案[161]。任务委托时会考虑使用能力优越但有一定局限性的无人机。Dubins 汽车模型被用来创建无人机飞行路径以及两个额外的航班来协调路径并创建实际路径。

- Advantages: Considers kinematic constraints (motion, external forces, propulsion), resource constraints (fuel and onboard weapons), and time constraints (time windows and sequence of tasks).
优点：考虑运动学约束（运动、外力、推进力）、资源约束（燃料和机载武器）和时间约束（时间窗口和任务顺序）。
- Limitations: Does not cover practical mission scenarios such as weather constraints, radar, terrain barriers during simulation.
局限性：不涵盖模拟过程中的天气限制、雷达、地形障碍等实际任务场景。
- Possible enhancements: Distributed model with more stochastic features can be considered to cover practical mission scenarios; a comparison of a centralized and distributed algorithm for stochastic task assignment needs to be made.
可能的改进：可以考虑使用具有更多随机特征的分布式模型来覆盖实际的任务场景；需要对随机任务分配的集中式和分布式算法进行比较。

5.3.8. Battle Antennae Search with GA (BAS-GA)

5.3.8. GA 战斗天线搜索 (BAS-GA)

The BAS algorithm-based multi-UAV task assignment using an improved GA was discussed in [162]. First, the target sequence is searched using the BAS algorithm, and then double-crossing operatives are used to increase the multiplicity of the target arrangement to find the optimal solution. Finally, dynamic adjustment of **mutation probability** is considered to improve the local search ability to avoid local optima. Homogeneous UAVs are assigned to accomplish tasks within a target area. Three main types of tasks are considered, i.e., Task= {Reconnaissance,Attack,Verify}.

[162]讨论了使用改进的 GA 的基于 BAS 算法的多无人机任务分配。首先使用 BAS 算法搜索目标序列，然后使用双交叉算子增加目标排列的重数来寻找最优解。最后，考虑动态调整**变异概率**，提高局部搜索能力，避免局部最优。同类无人机被分配来完成目标区域内的任务。考虑三种主要类型的任务，即 Task= {Reconnaissance,Attack,Verify}。

- Advantages: Diversity of searching capacity improves convergence time; local search ability is increased; has good convergence from small to large missions.
优点：搜索能力的多样性提高了收敛时间；本地搜索能力增强；从小到大任务具有良好的收敛性。
- Limitations: Does not consider the kinematics of UAV which impacts on performance and constraints of the realistic dynamic environment such as threats and uncertainties.
局限性：没有考虑无人机的运动学对性能的影响以及现实动态环境的约束，例如威胁和不确定性。
- Possible enhancements: Rapid task adjustment using centralized and distributed methods can be experimented on; integration of path plan into task assignment for broad task planning for multi-UAVs can be done.
可能的增强功能：可以尝试使用集中式和分布式方法快速调整任务；可以将路径规划集成到任务分配中，以实现多无人机的广泛任务规划。

5.3.9. Opposition-based GA Using Double-Chromosomes Encoding and Multiple Mutation Operators (OGA-DEM MO)

5.3.9. 使用双染色体编码和多突变算子的基于对立的遗传算法 (OGA-DEM MO)

Multi-UAV-based **reconnaissance** for heterogeneous targets using GA to optimize the task sequence of UAVs was considered in [163]. Targets are classified as line targets, point targets, and area targets, considering the features of the target geometry and view of the sensor's field. To solve the computational issues, OGA-DEM MO was developed to enhance the variety of the population to improve the global exploration ability. OGA-DEM MO aims to find the best task assignment to UAVs, maximizing overall performance effectiveness.

[163]考虑了基于多无人机的异构目标**侦察**，使用遗传算法来优化无人机的任务序列。考虑到目标几何形状和传感器视场的特征，将目标分为线目标、点目标和区域目标。为了解决计算问题，开发了OGA-DEM MO来增强种群的多样性，以提高全局探索能力。OGA-DEM MO 旨在找到无人机的最佳任务分配，最大限度地提高整体性能效率。

- Advantages: Minimizes the execution time and total UAV consumption and improves the **optimality** of the algorithm and convergence energy.
优点：最小化执行时间和无人机总消耗，提高算法的**最优性**和收敛能量。
- Limitations: No constraints are considered for **reconnaissance**; if the sensor's field of view covers the ground target, the task on that target is ended; constant UAV velocity is considered.
局限性：**侦察**时不考虑任何约束；如果传感器的视野覆盖了地面目标，则对该目标的任务结束；考虑恒定的无人机速度。

- Possible enhancements: Constraints of natural and dynamic environments such as threats, uncertainties, and other external forces can be considered.
- 可能的增强功能：可以考虑自然和动态环境的约束，例如威胁、不确定性和其他外力。

Artificial Neural Network (ANN)-based Task Assignment

基于人工神经网络 (ANN) 的任务分配

Coordination and cooperation always remain as a challenge in multi-robot and multi-agent systems. The ultimate goal of the agents is to find an **optimal policy** to maximize the cumulative reward instead of the **local optimal solution** in real time. In recent years, ANN has been developed as a promising approach to solve the issues of dynamic environment by continuously interacting with the environment. DRL [164], [165], DNN [166], SOM [167], and DQN [168] are some commonly used ANN algorithms. Existing ANN-based task assignment algorithms were studied in the literature [169], [170], [171], [172].

协调与合作始终是多机器人和多智能体系统中的一个挑战。智能体的最终目标是实时找到最大化累积奖励的**最优策略**，而不是局部最优解。近年来，人工神经网络已成为一种通过与环境持续交互来解决动态环境问题的有前途的方法。DRL [164]、[165]、DNN [166]、SOM [167]和DQN [168]是一些常用的ANN 算法。现有的基于 ANN 的任务分配算法在文献[169]、[170]、[171]、[172]中进行了研究。

5.3.10. Fast Task Assignment (FTA)

5.3.10. 快速任务分配 (FTA)

A Q-learning-based task assignment for heterogeneous UAVs in uncertain environments through neural network (NN) estimation and experience replay with priority to solve R L problems was studied in [173].

$\$T=\{T_1, T_2, \dots, T_N\}$ represents tasks in order of priority, and $\{a_t(t_k), \operatorname{operatorname}{def}(T_k), e_l e(T_k)\}$ is attack capability, defense capability, and electronic jamming capability, respectively, to complete task T_k . Environmental uncertainty is denoted as $\$env_k = \{w_k, r_k\}$, where w_k represents wind speed, and r_k represents rainfall. In addition, UAVs are denoted as $\$U=\left\{U_j\right\}_{j=1}^{12}$, $\{U_m\}_{m=1}^m$, and $\{U_e\}_{e=1}^e$ is the attack capability, defense capability, and electronic jamming capability of a j -type UAV. Task assignment can be formulated by repetitively choosing UAV type in correspondence with the maximum Q-value using the greedy strategy as depicted in Fig. 9.

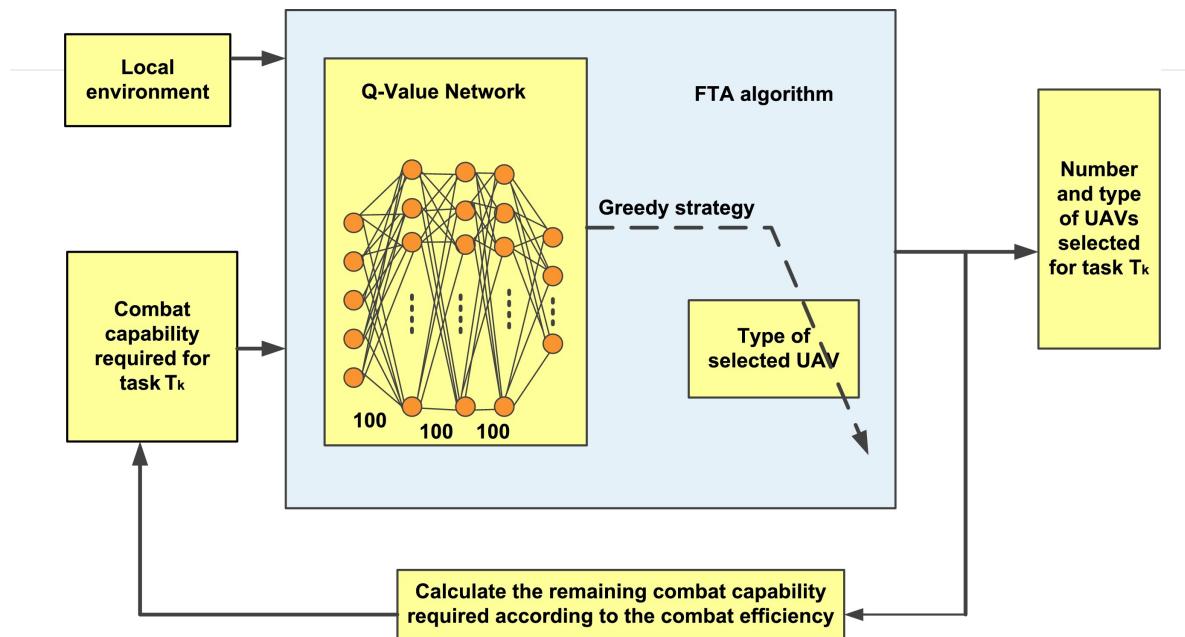
[173]研究了一种基于 Q 学习的异构无人机在不确定环境中的任务分配，通过神经网络 (NN) 估计和经验回放优先解决 R L 问题。

$\$T=\{T_1, T_2, \dots, T_N\}$ 按优先级顺序表示任务，并且 $\{\operatorname{operatorname}{att}(T_k), \operatorname{operatorname}{def}(T_k), \operatorname{operatorname}{ele}(T_k)\}$ 分别为完成任务 T_k 的攻击能力、防御能力和电子干扰能力。

环境不确定性表示为 $\$env_k = \{w_k, r_k\}$ ，其中 w_k 代表风速， r_k 代表降雨量。

此外，无人机被表示为 $\$U=\left\{U_j\right\}_{j=1}^{12}$ ， $\{U_m\}_{m=1}^m$ ，和 $\{U_e\}_{e=1}^e$ 是 j 型无人机的攻击能力、防御能力和电子干扰能力。

可以通过使用贪婪策略重复选择与最大 Q 值相对应的无人机类型来制定任务分配，如图 9 所示。



- Advantages: Computationally fast and efficient, highly adaptive, and handles uncertainty; deals with different types of tasks.
- 优点: 计算快速高效、适应性强、处理不确定性; 处理不同类型的任务。
- Disadvantages: Considers only stationary targets; complexities and aspects of the real battlefield environment are ignored.
- 缺点: 仅考虑静止目标; 真实战场环境的复杂性和各个方面都被忽略了。
- Possible enhancements: Moving and sudden targets and complexities of real battlefield environmental scenarios must be considered.
- 可能的增强功能: 必须考虑移动和突然的目标以及真实战场环境场景的复杂性。

5.4. Multi-fusion-based task assignments 基于多重融合的任务分配

To meet the requirements of UAV applications, the fusion of two or more task assignment algorithms is practiced in the literature. GA, ACO, and PSO-based multi-UAV task assignments for team orienteering problems are studied [174]. In this manner, the limitations imposed by specific algorithms can be complemented by the advantages of other algorithms, and hence, the best solution can be achieved. A few studies that utilized a combination of two or more algorithms are presented below.

为了满足无人机应用的要求, 文献中实践了两种或多种任务分配算法的融合。研究了针对团队定向问题的基于 GA、ACO 和 PSO 的多无人机任务分配 [174]。这样, 特定算法的限制可以通过其他算法的优点来补充, 从而获得最佳解决方案。下面介绍了一些结合使用两种或多种算法的研究。

5.4.1. Robust Filter Embedded Task Assignment (RFETA)

5.4.1. 鲁棒过滤器嵌入式任务分配 (RFETA)

A combination of robust and FETA algorithms for uncertain and dynamic environments was presented in [175]. One of the approaches is used to design task plans robust to uncertainties, reduce sensitivity to errors in *situational awareness*, and work for long durations. After updating these results, the following approach plans

again, resulting in the best plan with the current information. However, this may lead to churning if updates are frequent. RFETA uses proactive and reactive methods for handling uncertainties and enhances the worst-case scenario expressed as

[175]中提出了针对不确定和动态环境的鲁棒算法和 FETA 算法的组合。

- 其中一种方法用于设计对不确定性具有鲁棒性的任务计划，降低对态势感知错误的敏感性，并长时间工作。
- 更新这些结果后，以下方法再次计划，从而根据当前信息得出最佳计划。
- 但是，如果更新频繁，这可能会导致流失。

RFETA 使用主动和被动方法来处理不确定性，并增强最坏情况的情况，表示为

$$\max_{x_{k1} \in X_k} (\overline{C}_k - \underline{\lambda}_k \cdot \sigma_k)^T x_{k1} - \beta_k^T (x_{k1} \oplus x_{k-1})$$

在哪里 \overline{C}_k 是 k 时刻的最小值， \oplus 是异或， x_{k1} 是时间 k 时的决策变量，并且 $\underline{\lambda}_k \circ \sigma_k$ 表示逐个元素的乘法。

- Advantages: Presents robust results in uncertain environments; sensitive to errors; minimizes the effects of noise; yields fast convergence, and reduces churning.
- 优点：在不确定的环境中呈现稳健的结果；对错误敏感；最大限度地减少噪音的影响；产生快速收敛，并减少搅动。
- Limitations: It does not perform better if lots of information must be considered while planning.
- 局限性：如果在规划时必须考虑大量信息，它的性能并不会更好。
- Possible enhancements: Effects of intermittent measurements on the performance of RFETA can be studied; adaptive formulations of tuning parameter λ for optimization can be studied.
- 可能的增强功能：可以研究间歇测量对 RFETA 性能的影响；可以研究用于优化的调整参数 λ 的自适应公式。

5.4.2. Energy-Efficient Multi-UAV Assisted Multi- Access Edge Computing (EMU-MEC)

5.4.2. 节能多无人机辅助多路访问边缘计算 (EMU-MEC)

Energy efficient multi-UAV task assignment using MEC that offers computing services to resource-constrained UAVs has been studied [176]. Task assignment, joint device association, and computing resource assignment problems were formulated for the minimization of energy depletion considering the power budget, existing resources for computing at UAVs, and deadline restraints. The problem is first decomposed into three sub-problems and solved using the iterative block coordinate descent (BCD) algorithm. A single UAV-aided MEC was studied in [177], [178] where UAV roams around the target area to aid as a server. Performance and stability of the single UAV based system is constrained. Thus, constraints of energy, computing power, task completion are improved by controlling association of devices and resource assignment variables in multi-UAV system in the proposed method.

人们已经研究了使用 MEC 为资源受限的无人机提供计算服务的节能多无人机任务分配[176]。

考虑到功率预算、无人机计算的现有资源和期限限制，制定了任务分配、**联合**设备关联和计算资源分配问题，以最大限度地减少能量消耗。

该问题首先分解为三个子问题，并使用迭代块坐标下降（BCD）算法进行求解。

[177]、[178]研究了单个无人机辅助的 MEC，其中无人机在目标区域周围漫游，作为服务器提供帮助。单一无人机系统的性能和稳定性受到限制。因此，该方法通过控制多无人机系统中设备的关联和资源分配变量来改善能量、计算能力、任务完成度的约束。

- Advantages: Total energy consumption of sensor devices and UAVs is minimized.
- 优点：传感器设备和无人机的总能耗最小化。
- Limitations: Inter-cell inferences from mobile devices of other UAVs are experienced.
- 局限性：来自其他无人机移动设备的小区间干扰是存在的。
- Possible enhancements: UAV trajectory, communication, and computation resource assignment can be investigated.
- 可能的增强功能：可以研究无人机轨迹、通信和计算资源分配。

5.4.3. Multiple Time-Window-based Dubins Traveling Salesmen Problem (MTWDTSP)

5.4.3. 基于多时间窗的杜宾旅行商问题 (MTWDTSP)

Task assignment for multiple UAVs with altered capacities using a multi-objective symbiotic organism search algorithm to enhance UAV task arrangement was studied in [179]. In addition, a task model based on the time window and task assignment based on MTWDTSP was developed for various targets. Double-chain and some criteria are recognized for task assignment in logical and environmental constraints. A group of targets requires an investigation using heterogeneous UAVs subjected to different restraints. The Dubins model [180], [181] is introduced for UAVs with the following suppositions: (i) constant UAV velocity, (ii) reconnaissance job at certain heights, (iii) UAV flies at different heights without collision, and (iv) limited flight time. The dimensions of the UAVs are abridged from 3D to 2D.

[179]研究了使用多目标共生有机体搜索算法来增强无人机任务安排的具有改变能力的多个无人机的任务分配。此外，针对各种目标，开发了基于时间窗口和基于MTWDTSP的任务分配的任务模型。在逻辑和环境约束下，双链和一些标准被认可用于任务分配。一组目标需要使用受到不同约束的异构无人机进行调查。Dubins 模型[180]、[181]是针对无人机引入的，假设如下：(i) 无人机速度恒定，(ii) 在一定高度进行侦察工作，(iii) 无人机在不同高度飞行而不发生碰撞，(iv) 飞行时间有限。无人机的尺寸从3D 缩减为 2D。

- Advantages: Improved convergence speed and efficiency; provides optimized task sequence.
- 优点：提高收敛速度和效率；提供优化的任务序列。
- Limitations: Complexities increases with the number of UAVs and tasks; not all tasks are executed.
- 局限性：复杂性随着无人机和任务数量的增加而增加；并非所有任务都被执行。
- Possible enhancements: Constraints of complex dynamic environments such as the effect of speed of the wind, flight height, and temperature can be considered for better and realistic results.
- 可能的增强功能：可以考虑复杂动态环境的约束，例如风速、飞行高度和温度的影响，以获得更好、更真实的结果。

The advantages, limitations, and possible enhancements of each task assignment algorithms discussed in this section are listed in Table 4. This will help readers to understand better through comparison.

表 4列出了本节中讨论的每种任务分配算法的优点、局限性和可能的增强功能。这样可以帮助读者通过比较更好地理解。

Table 4. Comparison of task assignment algorithms in terms of their advantages, limitations, and possible enhancements.

Protocol	Advantages	Limitations	Possible enhancements
HA-TA [90]	Improves operational reliabilities and capabilities through adaptive task assignment systems and system awareness; demonstrated effective performance through simulation and experiments.	Randomly occurring failures provide a level of uncertainty; uncertainty present at all planning stages due to incomplete knowledge at the flight plan level, health of sensors during task assignment, and enduring maintenance at the mission planning level.	The amount and quality of feedback information can be improved, and a sophisticated stochastic model of health states like energy usage and sensors' performance can be embedded to deal with uncertainties.
MILP-TA [91]	Useful in offline task assignment calculations; provides optimal solution for UAV groups with combined tasks with timing and task order limitations.	Makes a discrete representation of real-world problems, which makes solutions impractical; solution requires high computation time and makes unreliable for real-time use.	Kinematic constraints of UAVs should be focused; aspects of complex and dynamic environment such as obstacles, uncertainties, and wind speed must be considered.
MTWPS [94]	Deals with online uncertainties such as communication issues, UAV malfunction, and time-sensitive target problem; easy computing used to lessen simulation time for large-scale UAV networks.	A UAV's waiting time depends on the performance of other UAVs, which makes waiting time complicated for large-scale networks; the deadlock problem is ignored.	Different types of deadlocks require better handling, which can be focused on in the future; threats and other risks present in the stochastic environment should also be focused on.
DTA [96]	Time efficiency, scalability, and robustness are achieved; achieves the mission goal with minimum cost.	Does not consider the uncertainties and aspects of dynamic environments; preliminary information regarding the size and location of enemy UAVs is unknown.	Minimizing the combination of all costs is desired, requiring a solver algorithm; threats and uncertainties of the complex dynamic environments need consideration.

Protocol	Advantages	Limitations	Possible enhancements
SMAA-2 [97]	Uncertainties are analyzed; finds satisfactory solution even in severe situations.	Sensor error or target movement leads to unclear localization; probability of UAVs being destroyed while handling attacks remains.	Threats present in the defense system of targets can be made more apparent; a confidence factor is introduced to judge the accuracy of criteria values.
FEUTA [112]	Condenses the consequence of noise caused due to high-frequency on the planner; mitigates the effects of noise.	Ignores the aspects such as threats and uncertainties, which makes it unreliable for the real environment; accepts noise entirely.	The importance of robust methods proposed for FEUTA and their connection to noise rejection algorithms must be investigated.
CTAI [113]	Decomposition of motion planning issues helps obtain a near-real-time solution; risk is mitigated by increasing distance between UAV and threats.	Tracking error increases due to disturbances; only static threats are considered, and thus the results obtained are suboptimal.	Timing of target intersection can be considered; aspects of the dynamic, complex 3D environment such as wind speed, risks need consideration.
EETA [117]	Lessens energy depletion, and optimizes task offloading, resource sharing, and trajectory without increasing delay; minimizes UAV's propulsion and computation power.	Single UAV is used, which increases the risk of mission failure.	UAVs' kinematic constraints such as external forces, motions, and physical limitations of the dynamic environment such as threats and uncertainties must be considered during the assignment process.
EDATA [118]	Energy consumption, operational time, and fairness of the system are optimized jointly; environmental effects on flights and performance are considered.	GPS localization errors and signal losses exist; increase in execution time for FTUS, which impacts the performance of the model.	Cases of accidents and UAV failures must be considered; the modeling part could be improved concerning the energy consumed by GPS in errors and signal losses.

Protocol	Advantages	Limitations	Possible enhancements
DP-CTA [119]	Generates cooperative and optimal solution quickly; recovers planning of real-world air operations providing improvements.	Computation time grows with the number of targets, and the uncertainties of real complex environment are ignored.	Constraints of the real dynamic environment such as sudden threats, effects of wind, and temperature on UAV flights must be focused on.
PHM-based MTA [123]	Applies system-level RUL methodology to accomplish task assignment including failure probability.	It may not be feasible to know the probability of failure in advance; threats and uncertainties of real stochastic environments are ignored.	PHM-based TA for extensive haul task reaction of algorithm during failure requires further investigation; consideration of aspects of real environment such as wind, temperature, threats is a must.
ABMC-TA [132]	Solves multi-constraint task assignment issues efficiently considering the constraints of sensors, time, danger, and fuel.	2D environment with static obstacles considered for implementation which cannot justify the constraints of real dynamic environment; kinematics of UAV such as motion and impacts of external forces on its flight path needs consideration.	Complexities imposed by real 3D environment need to be considered to confirm its practical validity; more experimental study and investigation are required to verify the efficiency of the proposed method in real environment.
RDTA [134]	“No Fly Zones” are considered; a modified candidate plan selection algorithm is used to improve performance and increase the robustness to the inconsistencies.	Does not consider the heterogeneity of UAVs and targets and the constraints of the dynamic environment such as threats and effects of external forces (wind, temperature, etc.) on UAVs performance.	Conflicts arising due to low convergence need to be eliminated; heterogeneity of UAVs and other environmental aspects such as threats and uncertainties must be considered.

Protocol	Advantages	Limitations	Possible enhancements
CBBA [137], [138]	Handles operational complexities of multi-UAVs; provides reliable, conflict-free UAV task assignments.	All targets may not be discovered; dynamic threats and uncertainties are ignored.	Kinematic and dynamic UAV constraints on air and ground can be incorporated; UAVs need to be penalized for massive heading changes.
EAT-PSO [143]	Achieves a minimum total make span, forms a seamless UAV operation schedule, and short computation time in finding a high-quality feasible solution.	Allows a single UAV to occupy a position at a particular time, ignoring uncertain events and obstacles.	An anti-collision refinement phase can be incorporated, a detailed study on the distributed control of UAVs can be conducted, and the robustness of the system can be assessed.
OPA [146]	Works with short execution time, highly flexible, strongly robust, and scalable; quickly finds optimal solution; avoids difficult cost function based on the tasks, and has high calculation efficiency.	Does not consider the constraints of the actual battlefield; threats and uncertainties that may arise in the dynamic environment are ignored.	Adaptive approach of cooperative task assignment can be introduced to balance the load of UAVs and adapt to real-time battlefield situations.
CTA [147]	Efficiently solves considerate task assignment problems of UAVs under multiple circumstances and constraints; fast convergence and a highly stable iterative curve are assured.	Focuses on flight length only and ignores other constraints of task assignment; threats and uncertainties of the real environment are ignored, making the solution impractical.	Estimation of cost function could be adjusted considering the flight mission; in an actual battlefield environment, target values vary, and distribution of UAV firepower must be considered.

Protocol	Advantages	Limitations	Possible enhancements
CTMAP-GA [158]	Promptly offers feasible task assignment solutions and real-time implementation for high dimensions; considers task priority and coordination, time constraints, and trajectory.	Performance degrades if the search space is harsh; it ignores the aspects of real dynamic environments, which makes the proposed method impractical in real-time implementation.	Proper synchronization must be considered to maintain coordination among UAVs; constraints of the dynamic environment such as threats, uncertainties, and external forces and their impacts on UAV's motion need to be investigated.
CMTAP [159]	Solves computational complexity for classical combinatorial optimization methods with higher efficiency and considers task priority and timing constraints.	Does not consider the dynamic aspects of the environment such as impacts of threats, uncertainties, and external forces on UAVs flight; heterogeneity of tasks, targets, and UAVs are ignored.	Moving, sudden targets, and actual environmental scenarios must be considered; computational efficiency needs to be realized to achieve a real-time solution.
MGA-CTA [160]	Considers and addresses the heterogeneity of UAVs, targets, and environments; mirror representation to deal with the limited resources.	Only one kind of resource, i.e., weapon, is considered; stationary targets considered in simulation, weapons are used in one way only.	Different operational and kinematic constraints of UAV must be considered; aspects of real dynamic environments such as threats, uncertainties, and other external forces and their impacts need to be analyzed.

Protocol	Advantages	Limitations	Possible enhancements
CMTAPSVTW[161]	Considers kinematic constraints (motion, external forces, propulsion), resource constraints (fuel and onboard weapons), and time constraints (time windows and sequence of tasks).	Does not cover practical mission scenarios such as weather constraints, radar, terrain barriers during simulation.	Distributed model with more stochastic features can be considered to cover practical mission scenarios; a comparison of a centralized and distributed algorithm for stochastic task assignment needs to be made.
BAS-GA [162]	Diversity of searching capacity improves convergence time; local search ability is increased; has good convergence from small to large missions.	Does not consider the kinematics of UAV which impacts on performance and constraints of the realistic dynamic environment such as threats and uncertainties.	Rapid task adjustment using centralized and distributed methods can be experimented on; integration of path plan into task assignment for broad task planning for multi-UAVs can be done.
OGA-DEM MO [163]	Minimizes the execution time and total UAV consumption and improves the optimality of the algorithm and convergence energy.	No constraints are considered for reconnaissance; if the sensor's field of view covers the ground target, the task on that target is ended; constant UAV velocity is considered.	Constraints of natural and dynamic environments such as threats, uncertainties, and other external forces can be considered.
FTA [173]	Computationally fast and efficient, highly adaptive, and handles uncertainty; deals with different types of tasks.	Considers only stationary targets; complexities and aspects of the real battlefield environment are ignored.	Moving and sudden targets and complexities of real battlefield environmental scenarios must be considered.

Protocol	Advantages	Limitations	Possible enhancements
RFETA [175]	Presents robust results in uncertain environments; sensitive to errors; minimizes the effects of noise; yields fast convergence, and reduces churning.	It does not perform better if lots of information must be considered while planning.	Effects of intermittent measurements on the performance of RFETA can be studied; adaptive formulations of tuning parameter λ for optimization can be studied.
EMU-MEC [176]	Total energy consumption of sensor devices and UAVs is minimized.	Inter-cell inferences from mobile devices of other UAVs are experienced	UAV trajectory, communication, and computation resource assignment can be investigated.
MTWDTSP [179]	Improved convergence speed and efficiency; provides optimized task sequence.	Complexities increases with the number of UAVs and tasks; not all tasks are executed.	Constraints of complex dynamic environments such as the effect of speed of the wind, flight height, and temperature can be considered for better and realistic results.

5.4.4. Lesson learned 5.4.4. 吸取的教训

A good task assignment algorithm should be adaptive that can acquire the best performance in any circumstances. Thus, creating an adaptive system, which is capable of adapting not only to the objectives of the assignment but also to the limitations of the environment, is highly anticipated. Otherwise, massive degradation in the performance of the [target system](#) can be observed with the increasing amount of limitations imposed by the environment and UAVs. The task assignment algorithms, reviewed in this section, were evaluated through [computer simulations](#) by the original authors and many of them were also validated through theoretical analysis. In the future, one can consider implementing these algorithms in real systems and evaluate the performance accurateness of the algorithms in actual environment, which is necessary when handling the intricate [environmental issues](#). Though computer based simulations gives preliminary results for the validation of the algorithms, many unforeseen issues may arise while implementing the algorithms in real systems.

一个好的任务分配算法应该具有适应性，在任何情况下都能获得最佳性能。因此，人们高度期望创建一个不仅能够适应任务目标而且能够适应环境限制的自适应系统。否则，随着环境和无人机施加的限制越来越多，可以观察到目标系统性能的大幅下降。本节回顾的任务分配算法由原作者通过[计算机模拟](#)进行了评估，其中许多算法还通过理论分析进行了验证。未来可以考虑在实际系统中实现这些算法，并评估算法在实际环境中的性能准确性，这在处理复杂的环境问题时是必要的。尽管基于计算机的模拟给出了算法验证的初步结果，但在实际系统中实现算法时可能会出现许多不可预见的问题。

6. Comparison of task assignment algorithms for UAVs 无人机任务分配算法比较

In this section, existing task assignment algorithms studied for UAVs and UAV networks are qualitatively compared in terms of their main ideas, algorithms used for the task assignment process, performance metrics considered, and qualitative and operational features. [Table 4](#) summarizes the main ideas of all 27 algorithms discussed in the previous section along with the metrics considered for their improvement. It can be observed that the most popular algorithms used for task assignment are [GA](#), RHTA, MILP, and DP. The primary focus of the task assignment algorithms is to minimize the convergence time and complexities while maintaining cooperation between UAVs.

在本节中，对现有的无人机和无人机网络任务分配算法的主要思想、用于任务分配过程的算法、考虑的性能指标以及定性和操作特征进行了定性比较。[表 4](#)总结了上一节讨论的所有 27 种算法的主要思想以及改进这些算法所考虑的指标。可以看出，最流行的任务分配算法是[GA](#)、RHTA、MILP 和 DP。任务分配算法的主要重点是最大限度地减少收敛时间和复杂性，同时保持无人机之间的合作。

In [Table 5](#), task assignment algorithms are compared in terms of qualitative features and key characteristics, such as task interdependence, types of target, task precedence, types of UAVs considered, types of task assignment algorithms, level of operational complexities of the algorithms, performance evaluation tools used, and environmental scenario considered. Only a few of the algorithms consider real environmental scenarios that are full of uncertainties and adversities. An ideal task assignment algorithm must consider the dynamics and complexities of real-world scenarios. In some missions, task priorities are significant, and hence, they must be considered during the task assignment process. In addition, from the tabular comparison, we can observe that heterogeneous UAVs have been used in recent studies. Most task assignment algorithms are centralized and evolutionary. However, a combination of centralized and distributed task assignment algorithms has not been considered in the literature. The fusion of multiple algorithms can be observed in some task assignment processes. Most algorithms are aimed at minimizing complexities. Moreover, popular simulation tools include Monte Carlo, MATLAB, and mathematical formulations.

在[表 5](#)中，任务分配算法在定性特征和关键特征方面进行了比较，例如任务相互依赖性、目标类型、任务优先级、考虑的无人机类型、任务分配算法类型、算法的操作复杂程度、性能使用的评估工具并考虑环境情景。只有少数算法考虑了充满不确定性和逆境的真实环境场景。理想的任务分配算法必须考虑现实场景的动态性和复杂性。在一些特派团中，任务优先事项很重要，因此在任务分配过程中必须考虑这些优先事项。此外，从表格比较中，我们可以观察到最近的研究中已经使用了异构无人机。大多数任务分配算法都是集中式和进化式的。然而，文献中尚未考虑集中式和分布式任务分配算法的结合。在一些任务分配过程中可以观察到多种算法的融合。大多数算法的目的是最小化复杂性。此外，流行的仿真工具包括蒙特卡洛、MATLAB 和数学公式。

Table 5. Comparison of task assignment algorithms in terms of main ideas and design approaches.

Protocol	Main idea	Design approach
HA-TA [90]	Step towards allowing robust decision making for disseminated autonomous UAVs.	RHTA algorithm
MILP-TA [91]	Optimal formulation for solving the coupled multiple-assignment problem.	MILP algorithm
MTWPS [94]	Task assignment with constraint and time-sensitive uncertainties.	Meta-heuristic optimization called MTPS
DTA [96]	A dynamic task assignment using enemy information for combinatorial optimization.	Hungarian algorithm
SMAA-2 [97]	Offers suitable task assignment under severely uncertain circumstances.	Stochastic multi-criteria acceptability analysis method
FEUTA [112]	Performs reassignment at the rate that information is updated, which permits instantaneous response to any noteworthy fluctuations in the environment.	Noise rejection algorithm and linear integer program
CTAI [113]	Provides end-to-end explanation to cooperative control problem in the occurrence of dynamic threats.	Satisfying decisions theory
EETA [117]	Joint task assignment and resource assignment in UAV-aided MEC.	Block successive upper-bound minimization algorithm
EDATA [118]	EAUS, DAUS, and FTUS are proposed.	Linear integer problem
DP-CTA [119]	Two models of DP estimation methods are designed i.e. one-step look-ahead and two-step look-ahead	Dynamic programming based
PHM-based MTA [123]	Information from a PHM method is applied so as to support decision making through IVHM structure.	Receding horizon task assignment algorithm
ABMC-TA [132]	Auction-based technique that deals with assigning tasks of multi-UAV system under the constrictions of UAV capability, time window, cost of fuel and dangers during the travel.	Auction algorithm and multi-layer cost computation
RDTA [134]	Confirms the resultant plans calculated by team are conflict-free and feasible	RDTA
CBBA [137], [138]	Modified CBBA to handle obstacles and mitigate churning behavior	Consensus-based bundle algorithm

Protocol	Main idea	Design approach
EAT-PSO [143]	A mathematical model that solves the problem and heuristic-based methodology.	Earliest available time (EAT) and PSO algorithm
OPA [146]	Resolves the ideal order of numerous tasks for cooperative accomplishment.	Nearest neighbor method
CTA [147]	Cooperative multi-task assignment model.	Hungarian algorithm
CTMAP-GA [158]	Allows unique requirements of the scenario to be considered, such as task precedence and coordination, timing constraints, and trajectory limitations	GA
CMTAP [159]	Integrated task assignment and path optimization refining the visitation angle	Dubins car and GA
MGA-CTA [160]	Concerned with cooperative decision making and control	Modified GA with multi-type genes
CMTAPSVTW[161]	Cooperative multi-task assignment problem with stochastic velocities and time windows for heterogeneity	Meta-heuristic based on a modified GA
BAS-GA [162]	Uses two cross-operator methods to preserve the globality of the algorithm and perform a deep search of the solution space	Improved GA based on the beetle antenna search algorithm
OGA-DEM MO [163]	Presents a multi-UAV reconnaissance task assignment model for heterogeneous targets and an effective genetic algorithm to optimize UAV task sequence	Opposition-based GA
FTA [173]	Neural network approximation and prioritized replay to offload online computation to offline learning procedure	MDP and Q-learning-based
RFETA [175]	Provides substitute approach that associates robust scheduling with procedures established to eradicate churning	Robust filter embedded task assignment
EMU-MEC [176]	A joint device association, task assignment, and computing resource assignment problem considering the energy budget, available computing power, and task completion deadline constraints	Mixed-integer non-linear programming and block coordinate descent

Protocol	Main idea	Design approach
MTWDTSP [179]	Problem of task consignment is articulated as a manifold time window-based Dubins traveling salesman problem	Modified multi-objective symbiotic organism search algorithm

Similarly, Table 6, Table 6 summarizes different performance metrics, such as energy consumption, latency, efficiency of algorithms, adaptability scalability, and load balancing. Most task assignment algorithms focus on increasing the lifetime of the network by reducing energy consumption. Delay is another major consideration in task assignment algorithms. We can observe that most task assignment algorithms have managed to achieve low latency and high efficiency. Balancing the load among UAVs has not been considered much in state-of-the-art algorithms. However, load balancing also plays a significant role in large-scale mission planning systems. Moreover, performance determinants, such as adaptability and scalability, have not been considered much in previous studies.

类似地，表 6，表 6 总结了不同的性能指标，例如能耗、延迟、算法效率、适应性可扩展性和负载平衡。大多数任务分配算法的重点是通过减少能耗来延长网络的寿命。延迟是任务分配算法中的另一个主要考虑因素。我们可以观察到大多数任务分配算法都能够实现低延迟和高效率。最先进的算法尚未充分考虑平衡无人机之间的负载。然而，负载平衡在大规模任务规划系统中也发挥着重要作用。此外，在之前的研究中，适应性和可扩展性等性能决定因素并未得到太多考虑。

Table 6. Comparison of task assignment algorithms in terms of key features and characteristics. 表 6 .任务分配算法在关键特征和特性方面的比较。

Protocol	Task interdependency	Target	Priority	Environment
HA-TA [90]	No	Heterogeneous	Yes	Complex and uncertain
MILP-TA [91]	Yes	Heterogeneous	Yes	-
MTWPS [94]	No	Mobile	No	Complex and dynamic
DTA [96]	No	-	No	Battlefield
SMAA-2 [97]	No	Mobile	No	Severely uncertain
FEUTA [112]	-	Uncertain	No	Dynamic
CTAI [113]	Yes	Homogeneous	Yes	Dynamic and threat-based
EETA [117]	No	Mobile	No	-
EDATA [118]	No	IoT devices	Yes	Unknown and uncertain
DP-CTA [119]	Yes	Heterogeneous	No	Adversarial
PHM-based MTA [123]	No	Stationary	Yes	Critical
ABMC-TA [132]	No	Heterogeneous	No	Dynamic
RDTA [134]	No	-	No	Uncertain
CBBA [137], [138]	No	Stationary and mobile	No	Obstacle- and noise-based
EAT-PSO [143]	No	Heterogeneous	No	Indoor
OPA [146]	No	Stationary	No	Complex
CTA [147]	No	Mobile	No	Obstacle-based
CTMAP-GA [158]	Yes	Heterogeneous	Yes	-
CMTAP [159]	No	Heterogeneous and stationary	No	-
MGA-CTA [160]	Yes	Heterogeneous	No	-
CMTAPSVTW[161]	Yes	Stationary	Yes	Uncertain and dynamic
BAS-GA [162]	Yes	Stationary	No	-

Protocol	Task interdependency	Target	Priority	Environment
OGA-DEM MO [163]	No	Heterogeneous	No	Real-world mission environments
FTA [173]	No	Stationary	Yes	Complex and uncertain
RFETA [175]	No	Heterogeneous	No	Uncertain and dynamic
EMU-MEC [176]	Yes	IoT	No	Infrastructure-less
MTWDTSP [179]	No	Heterogeneous	Yes	Obstacle- and threat-based

Table 6. Comparison of task assignment algorithms in terms of key features and characteristics (continued). 表 6 . 任务分配算法在关键特征和特性方面的比较（续）。

Protocol	Year	UAV type	Task assignment type	Operational complexity	Performance evaluation tool
HA-TA [90]	2008	Draganfly VTi Pro R/C helicopters	Centralized	Low	MIT's Real-time Indoor Autonomous Vehicle Test Environment
MILP-TA [91]	2003	WASM	Centralized	High	Numerical formulations
MTWPS [94]	2017	Heterogeneous	Centralized	Low	MATLAB
DTA [96]	2019	–	Centralized	Low	SITL and Gazebo
SMAA-2 [97]	2015	Homogeneous	Stochastic and centralized	Low	Monte Carlo
FEUTA [112]	2004	Homogeneous	Centralized and stochastic	High	Numerical formulations
CTAI [113]	2002	Homogeneous	Coordinated and stochastic	High	MATLAB and Simulink
EETA [117]	2021	Single UAV	Centralized and stochastic	Low	Numerical formulation
EDATA [118]	2019	Rotary UAVs	Centralized and stochastic	–	Python and Gurobi optimization tools
DP-CTA [119]	2005	Homogeneous	Stochastic and centralized	Lower	Numerical formulations
PHM-based MTA [123]	2014	Homogeneous	Decentralized	–	MATLAB and CPLEX
ABMC-TA [132]	2016	Homogeneous	Centralized and stochastic	–	Experimented
RDTA [134]	2006	Heterogeneous	Decentralized	Low	Monte Carlo

Protocol	Year	UAV type	Task assignment type	Operational complexity	Performance evaluation tool
CBBA [137], [138]	2009	Heterogeneous	Decentralized	Low	Monte Carlo
EAT-PSO [143]	2019	Multi-copters	Bio-inspired	High	Numerical formulations
OPA [146]	2021	Heterogeneous	Bio-inspired	Low	Numerical formulation
CTA [147]	2017	Homogeneous	Bio-inspired	-	MATLAB
CTMAP-GA [158]	2005	-	Bio-inspired	Low	Monte Carlo
CMTAP [159]	2010	Heterogeneous	Bio-inspired	Low	Monte Carlo
MGA-CTA [160]	2012	Heterogeneous	Bio-inspired	Low	Monte Carlo
CMTAPSVTW[161]	2018	Heterogeneous	Bio-inspired	Low	MATLAB
BAS-GA [162]	2020	Homogeneous	Bio-inspired	Low	-
OGA-DEM MO [163]	2018	Heterogeneous	Bio-inspired	High	MATLAB
FTA [173]	2019	Heterogeneous	Bio-inspired	-	Python 3.7.0, Tensorflow
RFETA [175]	2007	Homogeneous	Multi-fusion	-	Monte Carlo
EMU-MEC [176]	2021	Edge server-equipped UAVs	Multi-fusion	Low	Numerical formulations
MTWDTSP [179]	2019	Heterogeneous	Multi-fusion	Low	Visual C++ and MATLAB

By comparison, for practical applications, stochastic evolutionary algorithms can perform better as they can learn from experience and adapt to the aspects of complex dynamic environments in real time. Moreover, decentralized algorithms assuring coordinated UAV operation can provide alternative solutions to enhance the performance of multi-UAV operations. In addition, robust and highly scalable algorithms are desirable for large missions. See Table 7.

相比之下，在实际应用中，随机进化算法可以表现得更好，因为它们可以从经验中学习并实时适应复杂动态环境的各个方面。此外，确保无人机协调运行的去中心化算法可以提供替代解决方案，以提高多无人机运行的性能。此外，大型任务需要稳健且高度可扩展的算法。参见表 7。

Table 7. Qualitative comparison of task assignment algorithms in terms of performance. 表 7. 任务分配算法在性能方面的定性比较。

Protocol	Energy consumption	Latency	Efficiency	Adaptability	Scalability
HA-TA [90]	Low	Low	High	High	–
MILP-TA [91]	–	High	High	No	No
MTWPS [94]	No	Low	High	No	No
DTA [96]	Low	Low	High	High	High
SMAA-2 [97]	No	No	Improved	No	No
FEUTA [112]	–	Low	Improved	No	No
CTAI [113]	–	Low	High	No	No
EETA [117]	Low	Satisfactory	High	No	No
EDATA [118]	Low	Low	High	No	No
DP-CTA [119]	No	Low	Improved	High	–
PHM-based MTA [123]	Low	Low	High	No	No
ABMC-TA [132]	Low	Low	High	–	–
RDTA [134]	–	Low	High	High	Low
CBBA [137], [138]	No	Low	High	–	–
EAT-PSO [143]	Recharging	Low	High	No	Yes
available					
OPA [146]	No	Low	High	High	High
CTA [147]	No	Low	Improved	No	No
CTMAP-GA [158]	Low	Low	High	High	High
CMTAP [159]	Low	Low	Improved	High	High
MGA-CTA [160]	Low	Low	Improved	No	–
CMTAPSVTW[161]	Low	Low	High	High	High
BAS-GA [162]	Low	Low	High	No	Yes
OGA-DEMOMO [163]	Low	Low	High	No	No
FTA [173]	No	Low	High	High	No
RFETA [175]	No	Low	High	High	High

Protocol	Energy consumption	Latency	Efficiency	Adaptability	Scalability
EMU-MEC [176]	Low	Low	High	—	—
MTWDTSP [179]	Low	Low	High	No	No

7. Open issues, research challenges, and future directions 未决问题、研究挑战和未来方向

Researchers in both academic and industrial fields are striving to utilize the full potential of UAVs. Regardless of the promising roles of UAVs, there are also a number of design challenges to be addressed. In fact, every UAV application has its own challenges and opportunities. From the above detailed review of the literature, we present the following research gaps along with future efforts that could be pursued for the practical deployment of UAVs. In this section, the major characteristics and research gaps faced during the design of task assignment schemes for a set of UAVs are briefly addressed. Furthermore, future research directions for these issues are suggested.

学术和工业领域的研究人员正在努力充分利用无人机的潜力。尽管无人机的作用前景广阔，但仍有许多设计挑战需要解决。事实上，每种无人机应用都有其自身的挑战和机遇。通过上述对文献的详细回顾，我们提出了以下研究空白以及未来为无人机的实际部署可以采取的努力。在本节中，简要讨论了一组无人机的任务分配方案设计过程中面临的主要特征和研究空白。此外，还提出了这些问题的未来研究方向。

7.1. UAV regulations 7.1. 无人机法规

Regulatory concerns are a key limiting factor in the deployment of UAV operations. Regardless of the widespread UAVs applications, there are numerous issues concerning confidentiality, security, collision avoidance, safety, and protection of data. In this respect, UAV guidelines are uninterruptedly being established to govern the operation of UAVs while bearing in mind various factors, such as UAV type, range, height, and speed of UAVs [182], [183]. Generally, five different criteria are frequently considered when developing UAV regulations. Despite their usefulness in each area, UAVs are banned in different places; hence, appropriate *airspace* protocols and registration of drones are necessary for appropriate UAV operation. Specific areas of flying should be recognized for UAV practitioners, where they can operate UAVs without any obstacles following the regulations. Apps that indicate restricted zones must be developed to ensure safe UAV operation. Moreover, combative counter-drone measures should be established to circumvent the utilization of UAVs for *malicious intent* or in an unintentional *security breach*.

监管问题是无人机行动部署的一个关键限制因素。尽管无人机应用广泛，但仍存在许多涉及机密性、安全性、防撞、安全和数据保护的问题。在这方面，不断制定无人机指南来管理无人机的操作，同时考虑各种因素，例如无人机类型、航程、高度和无人机速度[182]、[183]。一般来说，制定无人机法规时经常考虑五种不同的标准。尽管无人机在各个领域都很有用，但在不同的地方都被禁止使用；因此，适当的空域协议和无人机注册对于适当的无人机操作是必要的。应为无人机从业者提供特定的飞行区域，让他们能够按照规定无障碍地操作无人机。必须开发指示限制区域的应用程序，以确保无人机安全运行。此外，应建立战斗性反无人机措施，以避免出于恶意或无意的安全漏洞而利用无人机。

7.2. Dynamic network topology 动态网络拓扑

The foundation of a peer-to-peer wireless network is a dynamic [network topology](#) that determines the relationships among neighboring nodes to be maintained within the network. Due to the high maneuverability of UAVs and constraints of the dynamic environment, the topology of UAV networks remains intermittent most of the time. High UAV mobility causes rapid changes in the [network topology](#) and fluctuations in radio communication channels. Communication between nodes is significantly affected when the topology of the network fluctuates due to failures and mobility. Delayed or incorrect information misleads the communication process. As a result, the network may experience delays, which are intolerable in many UAV applications. FANETs require exact and timely network information to determine the best and real-time communication. Moreover, adaptive and online communication protocols must be explored and designed for dynamic network topologies [184], [185].

对等无线网络的基础是动态网络拓扑，它确定网络内要维护的相邻节点之间的关系。由于无人机的高机动性和动态环境的限制，无人机网络的拓扑结构在大多数情况下保持间歇性。无人机的高机动性导致网络拓扑的快速变化和无线电通信信道的波动。当网络拓扑由于故障和移动性而发生波动时，节点之间的通信会受到显著影响。延迟或不正确的信息会误导沟通过程。因此，网络可能会出现延迟，这在许多无人机应用中是无法容忍的。FANET需要准确、及时的网络信息来确定最佳的实时通信。此外，必须针对动态网络拓扑探索和设计自适应在线通信协议[184]，[185]。

7.3. Localization 7.3.本土化

Localization plays a crucial role in the safe operation of UAVs beyond LoS applications. Because of the high and unpredictable mobility of UAVs, accurate localization of UAVs within a short time is problematic. The position of UAVs in the network affects the formation of appropriate communication between UAVs. A widely adopted solution is to use [GPS](#) [186], [187] to obtain the position coordinates of any [wireless node](#). However, the accuracy of UAV location is limited due to its 3D mobility and highly dynamic and obstacle-based environment. Furthermore, due to the probable large deployment of UAVs, the use of [GPS](#) may add unnecessary costs undermining its use prospective. Different [localization methods](#) have been studied for proficient UAV location. Network-based positioning, which is based on exchanging packets and height-based positioning, can be used to enhance [position information](#). Recently, localization techniques based on fuzzy logic that relies on weighted [centroid](#) and vision-based localizations [188] have been greatly favored for precise localization. Moreover, [wireless links](#) through which [location information](#) is shared must be reliable and robust in the network design [189].

除了视距应用之外，定位对于无人机的安全运行也起着至关重要的作用。由于无人机的机动性高且难以预测，短时间内准确定位无人机是存在问题的。无人机在网络中的位置影响无人机之间适当通信的形成。一种广泛采用的解决方案是使用[GPS](#) [186]、[187]来获取任何无线节点的位置坐标。然而，由于无人机的3D移动性以及高度动态和基于障碍物的环境，无人机定位的准确性受到限制。此外，由于无人机可能大规模部署，[GPS](#)的使用可能会增加不必要的成本，从而损害其使用前景。为了熟练的无人机定位，已经研究了不同的定位方法。基于网络的定位以交换数据包和基于高度的定位为基础，可用于增强位置信息。最近，基于模糊逻辑的定位技术依赖于加权质心和基于视觉的定位[188]，在精确定位方面受到了极大的青睐。此外，共享[位置信息](#)的[无线链路](#)在网络设计中必须可靠且稳健[189]。

7.4. UAV speed 7.4.无人机速度

The mobility of UAVs is one of the major reasons for their rapid adoption in many different applications. Owing to the high mobility of UAVs, frequently changing and unpredictable topologies have been observed. The speed of UAVs varies with their size; small-sized UAVs usually move at 15 m/s approximately, and large UAVs can have a remarkable speed of 150 m/s. The types of UAVs must be carefully selected considering the desired speed

for their targeted applications. The trade-off between the speed of the UAV and its turning agility must also be prioritized. Different emergent technologies, such as designing dynamic and optimized trajectories for UAVs in 3D for better accuracy and speed, have sustained the durability of small UAV nodes. Consequently, restrictions on speed must be considered via field experiments. The optimized and adaptive speed of UAVs benefits highly enhanced UAV communication [190].

无人机的移动性是其在许多不同应用中快速采用的主要原因之一。由于无人机的高机动性，人们观察到频繁变化且不可预测的拓扑结构。无人机的速度因其大小而异；小型无人机的移动速度通常约为 15 m/s，大型无人机的移动速度可达 150 m/s。必须仔细选择无人机的类型，考虑其目标应用所需的速度。无人机的速度和转弯敏捷性之间的权衡也必须优先考虑。不同的新兴技术，例如为无人机设计动态和优化的 3D 轨迹以提高精度和速度，维持了小型无人机节点的耐用性。因此，必须通过现场实验来考虑对速度的限制。无人机的优化和自适应速度有利于高度增强的无人机通信[190]。

7.5. UAV types 7.5。无人机类型

Depending on the goals and application of UAV operation, one must select an appropriate UAV type and meet numerous requirements imposed by the anticipated QoS and environmental constraints. In fact, for the proper utilization of UAVs, numerous factors, such as UAV capabilities and onboard sensor devices, must be considered [191], [192]. UAVs are categorized based on their size, weight, speed, coverage range, and other operational capabilities. Inappropriate UAV selection may deteriorate complete network performance.

根据无人机操作的目标和应用，必须选择合适的无人机类型并满足预期服务质量^{服务质量}和环境限制所提出的众多要求。事实上，为了正确利用无人机，必须考虑许多因素，例如无人机能力和机载传感器设备[191]，[192]。无人机根据其尺寸、重量、速度、覆盖范围和其他操作能力进行分类。不适当的无人机选择可能会降低整体网络性能。

7.6. Channel modeling 7.6 信道建模

Wireless media amongst transmitters and receivers affect the signal propagation. The characteristics of A2G channels are considerably different from those of ground-to-ground (G2G) channels. A2G is highly susceptible to blockage, whereas A2A communication has dominant LoS communications. To design and implement optimal UAV applications, an accurate A2G channel model is a perquisite. In addition, A2G channels are highly dependent on altitude, type, angle of elevation, and type of propagation environment of UAV employment. Therefore, finding a standard channel model for UAV-to-UAV and UAV-to-ground communications requires ample simulations and measurements. In [193], [41], a summary of the prevailing A2G channel modeling works were presented. Ref. [193] provides both simulation and measurement-based results for path loss, delay spread, and fading in A2G communications. Thus, accurate channel characterization is crucial for performance optimization and the design of efficient UAV communication. Measurement operation while modeling the UAV-UAV and UAV-GBS channels with various velocities and moving directions of UAVs must be done in the presence of regularly and irregularly shaped infrastructures.

发射器和接收器之间的无线介质影响信号传播。A2G 信道的特性与地对地（G2G）信道有很大不同。A2G 非常容易受到阻塞，而 A2A 通信则以视距通信为主。为了设计和实现最佳的无人机应用，准确的 A2G 信道模型是必要条件。此外，A2G 信道高度依赖于无人机使用的高度、类型、仰角和传播环境类型。因此，找到无人机对无人机和无人机对地面通信的标准信道模型需要大量的模拟和测量。在[193]、[41]中，总结了流行的A2G信道建模工作。参考号[193]提供了 A2G 通信中路径损耗、延迟扩展和衰落的仿真和基于测量的结果。因此，准确的信道表征对于性能优化和高效无人机通信的设计至关重要。在对具有不同速度和无人机移动方向的 UAV-UAV 和 UAV-GBS 通道进行建模时，必须在存在规则和不规则形状的基础设施的情况下进行测量操作。

7.7. Synchronization 7.7. 同步

It is assumed that for every application sphere of UAV [time synchronization](#), proper time-stamping is required not only to be updated but also for coordination among other UAVs and devices. Synchronization plays a crucial role in UAV applications in disaster or emergency environments, where if one of the UAVs misses information at a particular time, the network will be partitioned, which degrades the network performance [194].

假设对于无人机[时间同步](#)的每个应用领域，不仅需要更新适当的时间戳，而且还需要用于其他无人机和设备之间的协调。同步在灾难或紧急环境中的无人机应用中起着至关重要的作用，如果其中一架无人机在特定时间丢失信息，网络将被分割，从而降低网络性能[194]。

7.8. UAV antennas 7.8 无人机天线

Owing to the ability to move in any direction and at different speeds, a new [antenna design](#) for UAV networks is essential to achieve higher data rates. One of the alternatives for improving the data rate during UAV-to-UAV and UAV-to-BS transmission is to install a [UAV tracking](#) antenna. For UAVs cruising at high speeds, it is desirable to have small, aerodynamic antennas that limit drag but can still yield sufficient bandwidth and coverage. Moreover, [directional antennas](#) are widespread due to the restricted energy and space limitations of particularly small sized UAVs. Moreover, tilted-beam circularly-polarized antenna can be used to save space. Performance is expected to be improved in terms of return losses, [axial ratio](#), and radiation pattern, which can be achieved by means of such antennas. Additional propose for scheming and using [smart antennas](#) is to reduce the UAV communication energy by minimizing the transmission power [85], [86]. Backscatter antennas and [wireless power transfer](#) techniques could also be adopted for longer-term network life. The integration of several adaptive antennas also contributes in the communication of highly mobile UAVs. A compact, sabre-like antenna that is capable of switching between two radiation patterns can be considered for better coverage in UAV communications [195], [196].

由于能够以不同的速度向任何方向移动，无人机网络的新天线设计对于实现更高的数据速率至关重要。提高无人机到无人机和无人机到基站传输期间数据速率的替代方案之一是安装无人机跟踪天线。对于高速巡航的无人机，希望拥有小型空气动力学天线，既能限制阻力，又能产生足够的带宽和覆盖范围。此外，由于特别小型无人机的能量和空间限制，定向天线得到了广泛应用。此外，可以使用斜波束圆极化天线来节省空间。预期在回波损耗、轴比和辐射方向图方面的性能将得到改善，这可以通过此类天线来实现。设计和使用智能天线的额外建议是通过最小化传输功率来减少无人机通信能量[85]，[86]。还可以采用反向散射天线和[无线功率传输](#)技术来延长网络寿命。多个自适应天线的集成也有助于高机动性无人机的通信。可以考虑使用能够在两种辐射模式之间切换的紧凑型军刀式天线，以在无人机通信中实现更好的覆盖范围[195]、[196]。

7.9. Network lifetime 7.9 网络寿命

[Power consumption](#) is always a primary concern in UAV communications. Most often, UAVs are battery-powered devices, and UAVs use their energy for thrust, data gathering, processing, and dissemination. The proficiency of UAV networks and network lifetime are greatly determined by the energy levels of the UAVs. Hence, ensuring energy-efficient UAV mission operations is of primary importance. Different [energy harvesting](#) techniques, such as wireless charging and solar energy can help increase the network lifetime of UAVs. An autonomous concept of [battery swapping](#) was introduced and enhanced in [197], [198], [199], [200]. In the swapping process, UAVs are connected to an external power supply for smooth functioning and to prevent data loss due to dead UAV nodes [201], [202]. Additionally, exhausted batteries are sometimes replaced by abundantly charged batteries to continue the current mission. The clustering of nodes also helps to minimize energy consumption by limiting the number of transmissions [203].

功耗始终是无人机通信的首要问题。大多数情况下，无人机是电池供电的设备，无人机利用其能量进行推力、数据收集、处理和传播。无人机网络的熟练程度和网络寿命在很大程度上取决于无人机的能量水平。因此，确保无人机任务运行的节能至关重要。无线充电和太阳能等不同的能量收集技术可以帮助延长无人机的网络寿命。[\[197\]](#)、[\[198\]](#)、[\[199\]](#)、[\[200\]](#)中引入并增强了电池交换的自主概念。在交换过程中，无人机连接到外部电源以实现平稳运行并防止由于无人机节点失效而导致数据丢失[\[201\]](#)，[\[202\]](#)。此外，耗尽的电池有时会被充满电的电池替换以继续当前的任务。节点集群还有助于通过限制传输数量来最小化能耗[\[203\]](#)。

7.10. Coordinated communication 7.10 协调沟通

The number of UAVs employed in any application may vary from one to numerous. Proper coordination and control among UAVs is necessary for better network performance. To this end, several efforts have been made to design error-free mechanisms to guarantee cooperation among UAVs. However, most of them are still immature and are subject to several challenges. The distributed approach divides the workload between all nodes, making peer-to-peer communication suitable. In the [centralized approach](#), a central entity is responsible for designing and controlling all other nodes. Collision avoidance and proper sharing of tasks between the employed UAVs are significant for efficient cooperation and collaboration. A bandwidth-efficient multi-robot coordination algorithm tackling the problem of UAV swarm formation and maintenance was studied in [\[204\]](#). Network [time synchronization](#) and proper time-stamping for UAV networks require intense investigation for proper coordination among UAVs.

任何应用中使用的无人机数量可能从一架到多架不等。无人机之间的适当协调和控制对于更好的网络性能是必要的。为此，人们做出了一些努力来设计无差错机制来保证无人机之间的合作。然而，它们中的大多数仍然不成熟，并且面临一些挑战。分布式方法在所有节点之间划分工作负载，从而适合点对点通信。在集中式方法中，中央实体负责设计和控制所有其他节点。所使用的无人机之间的避免碰撞和正确分担任务对于高效合作和协作具有重要意义。[\[204\]](#)研究了一种解决无人机群形成和维护问题的带宽高效的多机器人协调算法。无人机网络的网络[时间同步](#)和正确的时间戳需要进行深入调查，以确保无人机之间的正确协调。

7.11. Network density 7.11 网络密度

A UAV network consists of flying nodes that rely on battery power and limited onboard resources; therefore, communication protocols that consume the minimum possible energy with new inbuilt intelligent technologies must be focused on. Based on the types of UAV and objectives of the applications, the density of UAVs can vary from small to extremely dense. In dense UAV networks, suitable [clustering methods](#) must be considered to aid communication. Designing adaptive density-based communication protocols can be another credible solution [\[205\]](#), [\[27\]](#). In addition, adaptive and energy-efficient [clustering approaches](#) improve the performance of dense UAV networks. The assessment of wind and weather conditions affects drone robustness against harsh environmental conditions to enhance network lifetime.

无人机网络由依赖电池电源和有限机载资源的飞行节点组成；因此，必须关注采用新的内置智能技术消耗尽可能少的能量的通信协议。根据无人机的类型和应用目标，无人机的密度可以从小型到极其密集不等。在密集的无人机网络中，必须考虑合适的聚类方法来辅助通信。设计基于自适应密度的通信协议可能是另一种可靠的解决方案[\[205\]](#)、[\[27\]](#)。此外，自适应和节能的[集群方法](#)可以提高密集无人机网络的性能。对风和天气条件的评估会影响无人机在恶劣环境条件下的鲁棒性，从而延长网络寿命。

7.12. Security and privacy 7.12. 安全和隐私

UAVs are exposed to many possible security risks attributed to the susceptibility of protocols and resource-constrained UAV nodes. Security communication is necessary for many UAV applications. Security issues are a **foremost concern** in UAV communications because wireless media are used. Highly secure and safe communication is of **utmost importance** for military applications, where data secrecy is the principal focus. Any alterations in information by a fraudulent node may cause severe degradation of UAV applications. Furthermore, invaders may cause network partitioning and decrease network efficiency. Consequently, it is very important to develop robust and secure UAV communications that can meet the diverse needs of application reliability with minimum untrusted node involvement [206]. The security mechanisms of UAV communications are highly affected by the presence of faulty nodes and erroneous communication channels, which must be controlled with effective approaches. Different cryptographic measures, such as **data encryption** and hashing techniques, can be used to prevent network attacks [207]. **Block chain** technology can be utilized to ensure the safety and security of UAVs based on **collected data**.

由于协议的敏感性和资源受限的无人机节点，无人机面临许多可能的安全风险。安全通信对于许多无人机应用来说是必要的。由于使用无线媒体，安全问题是无人机通信中最重要的问题。高度安全的通信对于军事应用至关重要，其中数据保密是主要关注点。欺诈节点对信息的任何更改都可能导致无人机应用的严重退化。此外，入侵者可能会导致网络分区并降低网络效率。因此，开发强大且安全的无人机通信非常重要，该通信可以满足应用程序可靠性的多样化需求，同时最大限度地减少不可信节点的参与[206]。无人机通信的安全机制受到故障节点和错误通信通道的影响很大，必须采取有效的方法进行控制。可以使用不同的加密措施，例如**数据加密**和散列技术，来防止网络攻击[207]。基于收集的数据，可以利用**区块链技术**来确保无人机的安全。

7.13. Fault-tolerant system 7.13 容错系统

In contrast to manned aerial networks, UAVs allow a wider range of flight operating points and are highly vulnerable to failures. To augment safe and reliable UAV communications, fault-tolerant methods must be considered when designing control systems for multiple UAVs. Different strategies of fault-tolerant control for UAV communications are receiving significant attention from researchers because of the growing awareness of hazards resulting from the failure of different components and the necessity of trustworthy and secure systems in emergent scenarios. A network of UAVs must share critical data, and faults in a particular UAV can break down the entire network. Keeping this in mind, robust fault-tolerant mechanisms should be adopted to overcome the different **failures observed** in UAV networks [208], [209]. Inter-UAV support must be restructured in case the **network partitions** due to the failure of one or more UAVs.

与有人驾驶航空网络相比，无人机允许更广泛的飞行操作点，并且很容易出现故障。为了增强安全可靠的无人机通信，在设计多架无人机的控制系统时必须考虑容错方法。由于人们越来越意识到不同组件故障造成的危险以及在紧急情况下需要可靠和安全的系统，无人机通信的不同容错控制策略正受到研究人员的极大关注。无人机网络必须共享关键数据，特定无人机的故障可能会破坏整个网络。记住这一点，应该采用强大的容错机制来克服无人机网络中观察到的不同故障[208]，[209]。必须重组无人机间支持，以防由于一架或多架无人机故障而导致**网络分区**。

7.14. Integration with cloud computing 7.14 与云计算集成

Cloud services have become the next frontier in advancing the workflows of UAVs. Cloud services have facilitated UAVs by allowing the use of resources at their leisure. The limitations of UAV networks can be incorporated and supported using **cloud computing** techniques, as discussed in [210]. To accomplish higher **data**

throughput in UAV-based applications, aerial caching could be a promising solution. A cloud-assisted approach can also be useful for deriving optimal flight and data acquisition. Some research efforts have been made to provide techniques for linking UAVs to the cloud.

云服务已成为推进无人机工作流程的下一个前沿。云服务允许无人机随意使用资源，从而为无人机提供了便利。正如[210]中所讨论的，可以使用云计算技术来合并和支持无人机网络的局限性。为了在基于无人机的应用中实现更高的数据吞吐量，空中缓存可能是一个有前途的解决方案。云辅助方法也可用于获得最佳飞行和数据采集。已经开展了一些研究工作来提供将无人机连接到云的技术。

7.15. Path planning 7.15 路径规划

Path/trajectory planning is an important and challenging issue in the design of UAV-based communication systems. In fact, task assignment and path planning are highly interrelated aspects of any UAV mission. The UAV path must be optimized with respect to key performance metrics, such as spectral efficiency, throughput, energy, and delay. Although many studies have focused on path planning algorithms for UAVs, UAV networks still experience issues of target location and identification due to high UAV mobility. Thus, it is desirable to rapidly develop dynamic trajectory planning approaches for UAVs to increase the likelihood of end-to-end connections while maintaining adequate intact target area coverage. Various path planning techniques proposed for UAV path planning were analyzed in [211]. There are still several issues in the path-planning process, in which path optimization considers the mobility patterns, obstacles, delay, and energy. Moreover, the algorithm can be equipped with deep learning approaches to help UAVs generate an environment map in real time. Genetic and chaotic operators can also be introduced to improve the convergence rate and hence the efficiency. 3D real-time path planning for different obstacle densities and moving obstacle environments in the presence of uncertainty was studied in [212], [213], [214].

路径/轨迹规划是基于无人机的通信系统设计中的一个重要且具有挑战性的问题。事实上，任务分配和路径规划是任何无人机任务中高度相关的方面。无人机路径必须针对关键性能指标进行优化，例如频谱效率、吞吐量、能量和延迟。尽管许多研究都集中在无人机的路径规划算法上，但由于无人机的高机动性，无人机网络仍然面临目标定位和识别的问题。因此，需要快速开发无人机的动态轨迹规划方法，以增加端到端连接的可能性，同时保持足够的完整目标区域覆盖。[211]中分析了针对无人机路径规划提出的各种路径规划技术。路径规划过程中仍然存在几个问题，其中路径优化考虑了移动模式、障碍物、延迟和能量。此外，该算法可以配备深度学习方法，帮助无人机实时生成环境地图。还可以引入遗传算子和混沌算子来提高收敛速度，从而提高效率。[212]、[213]、[214]研究了在存在不确定性的情况下针对不同障碍物密度和移动障碍物环境的3D实时路径规划。

7.16. Integration with IoT 7.16 与物联网集成

The efficient operation of UAVs requires addressing several key problems [215]. For instance, the number of UAVs that are necessary for full coverage of a given geographical area must be considered. UAVs are self-efficient for sensing and decision making, and this has made UAV networks more demanding. The integration of UAVs with IoT adds huge dimensions and remarkable results. However, there are many additional challenges [216]. Designing scalable UAV networks is in high demand as the application areas to be covered, as well as the number of tasks to be accomplished, are exponentially increasing.

无人机的高效运行需要解决几个关键问题[215]。例如，必须考虑完全覆盖给定地理区域所需的无人机数量。无人机在感知和决策方面具有自我高效性，这对无人机网络提出了更高的要求。无人机与物联网的融合增加了巨大的规模和显着的成果。然而，还有许多额外的挑战[216]。随着要覆盖的应用领域以及要完成的任务数量呈指数级增长，设计可扩展的无人机网络的需求量很大。

7.17. Quality of service 7.17 服务质量

QoS support over UAV communications remains as a challenging issue when previously built-up arrangements of the network are not accessible due to the changes in speed, location, architecture, or separation between UAV nodes. Different types of QoS provisions for UAV-aided IoT networks were recommended in [217]. New resource innovation and planning procedures are necessary to dynamically change the requirements of QoS of UAV networks. Therefore, well-organized approaches for the efficient handover mechanism, proficient methods for determining and solving UAV battery related issues, user-facilitation, and network assortment, capability, and spectral effectiveness needs an additional exploration. Furthermore, designing of the protocols supporting multiple classes of traffic and preemption allowance, identification of position and prioritization of the packets are some unexplored areas. Timely update of the control information guarantees précis information but adds consumption of energy. Competent routing, flight self-sufficiency, and policy-making metrics can confirm the essential QoS of the targeted UAV applications.

当由于速度、位置、架构或无人机节点之间的分离的变化而无法访问先前构建的网络安排时，无人机通信的QoS支持仍然是一个具有挑战性的问题。[217]中推荐了针对无人机辅助物联网网络的不同类型的QoS规定。新的资源创新和规划程序对于动态改变无人机网络的服务质量要求是必要的。因此，高效切换机制的组织良好的方法、确定和解决无人机电池相关问题的熟练方法、用户便利性以及网络分类、能力和频谱有效性需要进一步探索。此外，支持多类流量和抢占允许的协议设计、数据包的位置识别和优先级划分也是一些尚未探索的领域。控制信息的及时更新保证了信息的精确性，但增加了能源的消耗。有效的航线、飞行自给自足和决策指标可以确认目标无人机应用的基本服务质量。

7.18. Scalability 7.18 可扩展性

The large-scale ever-growing use of UAVs in a wide range of applications is anticipated in the next few decades. Thus, UAV networks as well as task assignment algorithms designed for a swarm of UAVs must be highly scalable. Small-scaled UAV networks are the best choice for commercial applications because of their easy positioning, low maintenance, low acquisition costs, and high mobility. The number of UAVs employed, urgency of the UAV applications, and range to be covered are highly interrelated. Large-scaled UAVs are required when the mission area is large and delay is strictly undesirable. However, the complexity of task assignment algorithms increases with increasing number of UAVs and other IoT nodes. Designing scalable and adaptive communication protocols is a principal challenge for both light and dense UAV networks [218].

预计未来几十年，无人机将在各种应用中得到大规模、不断增长的使用。因此，无人机网络以及为无人机群设计的任务分配算法必须具有高度可扩展性。小型无人机网络因其易定位、低维护、低采购成本和高机动性而成为商业应用的最佳选择。使用的无人机数量、无人机应用的紧迫性以及覆盖范围是高度相关的。当任务区域很大并且延迟是绝对不可取的时候，就需要大型无人机。然而，任务分配算法的复杂性随着无人机和其他物联网节点数量的增加而增加。设计可扩展和自适应的通信协议是轻型和密集型无人机网络的主要挑战[218]。

7.19. Mobility management 7.19 移动管理

Networks of UAVs face many design problems related to [network mobility](#). Frequent [topology changes](#) affect the communication and cooperation among UAVs. Different [mobility models](#) have been studied to manage the movement of UAVs, but they are insufficient to completely solve the issues faced in UAV communication. Mobility models must be selected based on network requirements. UAV mobility at low speed reduces the coverage significantly, which results in network delays. Regarding the emergency application of UAVs, mobility models with minimal latency are highly desired. However, continuous connectivity with the network is very challenging, which may result in degraded network performance in such scenarios. Thus, new mobility models for a specific environment and applications that can cope with the challenges of mobility must be focused on.

Recently, different mobility models have been designed for UAVs [219], and they can use variable speeds during different communication phases. Dynamic UAV speed control using [machine learning techniques](#) to adapt the UAV speed according to network requirements is another possible solution. Based on recent advances in [deep reinforcement learning](#) algorithms, green mobility management has been proposed for UAV-assisted IoT.

无人机网络面临许多与网络移动性相关的设计问题。频繁的拓扑变化会影响无人机之间的通信和协作。人们已经研究了不同的移动模型来管理无人机的运动，但不足以完全解决无人机通信中面临的问题。必须根据网络要求选择移动模型。无人机低速移动会显著降低覆盖范围，从而导致网络延迟。对于无人机的紧急应用，非常需要具有最小延迟的移动模型。然而，与网络的持续连接非常具有挑战性，在这种情况下可能会导致网络性能下降。因此，必须关注能够应对移动挑战的特定环境和应用的新移动模型。最近，已经为无人机设计了不同的移动模型[219]，它们可以在不同的通信阶段使用可变的速度。使用[机器学习技术](#)动态控制无人机速度，根据网络要求调整无人机速度是另一种可能的解决方案。基于[深度强化学习](#)算法的最新进展，针对无人机辅助物联网提出了绿色移动管理。

7.20. Task offloading 7.20 任务卸载

In many mission planning systems, numerous tasks must be conducted simultaneously. Multi-UAV systems are in high demand for such applications. Tasks must be divided among the employed UAVs such that all tasks receive proper attention in a timely manner. Owing to time-sensitive and computation-intensive UAV applications and high mobility scenarios, cost-efficient task offloading remains a challenging issue. Many intelligent task offloading algorithms have been presented in the literature for the proper utilization of UAV networks [220], [221] that aim to minimize the overall energy consumption for the accomplishment of predefined tasks.

在许多任务规划系统中，必须同时执行许多任务。此类应用对多无人机系统的需求很高。必须在所使用的无人机之间分配任务，以便所有任务都能及时得到适当的关注。由于时间敏感和计算密集型无人机应用和高移动性场景，具有成本效益的任务卸载仍然是一个具有挑战性的问题。文献中已经提出了许多智能任务卸载算法，用于正确利用无人机网络[220]、[221]，其目的是最大限度地减少完成预定义任务的总体能耗。

7.21. Impact of 5G, B5G, and mm-wave communication 7.21 5G、B5G 和毫米波通信的影响

Providing universal connectivity to miscellaneous IoT devices is a prime challenge for 5G and B5G. 5G communications were deployed globally from 2020, and many more competences are in the progression for standardization, such as mass connectivity, guaranteed low latency, and ultra-reliability. Concerning future requirements and demands, researchers are now interested in beyond 5G/6G communication, which is expected to provide worldwide coverage, improved energy/spectral/cost efficiency, better intelligent security, and more benefits. The challenges, issues, and concerns for these networks are immense, and integrating them with highly mobile aerial nodes adds more complications and technical problems. Mm-wave communication with multi-gigahertz/terahertz bandwidth availability enhances the proficiency and [data transmission rate](#) in comparison to microwave communications. Recently, UAVs have been expected to serve ample applications, and the use of mm-wave bands can be very promising for handling the higher demands for data rate to achieve greater throughput. However, mm-wave suffers from propagation-related inadequacies, which can be compensated by the flexibility of UAVs. Many other deficiencies present in mm-wave signals must be resolved while preserving its benefits. 5G, B5G, and mm-wave-based UAV communications were discussed in [222]. Further investigation of mm-wave and 6G communications is extremely desired for reliable, flexible, robust, and spectrally efficient UAV communications [223], [224].

为各种物联网设备提供通用连接是 5G 和 B5G 面临的主要挑战。5G 通信从 2020 年开始在全球范围内部署，更多能力正在标准化过程中，例如大规模连接、保证低延迟和超可靠性。考虑到未来的要求和需求，研究人员现在对超越 5G/6G 的通信感兴趣，这种通信有望提供全球覆盖、更高的能源/频谱/成本效率、更好的智能安全性和更多效益。这些网络面临着巨大的挑战、问题和担忧，将它们与高度移动的空中节点集成会增加更多的复杂性和技术问题。与微波通信相比，具有多千兆赫/太赫兹带宽可用性的毫米波通信提高了熟练程度和数据传输速率。最近，无人机有望提供广泛的应用，而毫米波段的使用非常有希望满足对数据速率的更高要求，从而实现更大的吞吐量。然而，毫米波存在传播相关的不足，这可以通过无人机的灵活性来弥补。毫米波信号中存在的许多其他缺陷必须得到解决，同时保留其优点。[\[222\]](#) 中讨论了 5G、B5G 和基于毫米波的无人机通信。对于可靠、灵活、稳健和频谱高效的无人机通信而言，迫切需要对毫米波和 6G 通信进行进一步研究[\[223\]](#)、[\[224\]](#)。

7.22. Intelligence-based algorithms 7.22 基于智能的算法

Machine learning techniques play a promising role in UAV-based applications as they can deal with uncertain environmental constraints and can be implemented in real time. Machine learning improves UAV network performance by learning from the surrounding environment and **past experience**. Machine learning can possibly be leveraged to design and optimize UAV-based **wireless communication** systems [\[225\]](#), [\[226\]](#), [\[227\]](#), [\[228\]](#). Using RL methods, UAVs can automatically modify their locations, directions, and motion to serve ground nodes. UAVs can adapt to the constraints of dynamic environments by independently optimizing their paths. Furthermore, **neural networks** and data analysis models can be leveraged to envisage the behavior of ground nodes and efficiently arrange and control UAVs. The mobility of users and load sharing can be achieved for the best distribution and route planning of UAVs.

机器学习技术在基于无人机的应用中发挥着有前途的作用，因为它们可以处理不确定的环境约束并且可以实时实施。机器学习通过学习周围环境和过去的经验来提高无人机网络性能。机器学习可以用来设计和优化基于无人机的无线通信系统[\[225\]](#)，[\[226\]](#)，[\[227\]](#)，[\[228\]](#)。使用强化学习方法，无人机可以自动修改其位置、方向和运动以服务地面节点。无人机可以通过独立优化其路径来适应动态环境的约束。此外，可以利用神经网络和数据分析模型来设想地面节点的行为并有效地安排和控制无人机。可以实现用户的移动性和负载分担，以实现无人机的最佳分布和航线规划。

7.23. Lesson learned 7.23 吸取的教训

In this subsection, key lessons learned while surveying on the task assignment algorithms are provided. Multiple UAVs can be used in many applications to enhance the area of coverage, to minimize the cost of **operation and maintenance**, to do real-time operation, and to meet the network requirements. The task assignment is the significant aspect of multi-UAV based applications. Task allocation in UAVs needs several considerations such as UAV capability, time for task execution, cost of fuel, and risks imposed by the dynamic environment. An efficient task assignment algorithm can assure cooperative and coordinated UAV operation to meet the application requirements. The task assignment is mutually participated, discussed, and contended using multiple UAVs, or each UAV selects task independently in accordance to the observation of environmental constraints. Distributed algorithms emphasizes on achieving efficiency whereas they do not take global considerations into account, which may not result in accurate task assignment. Coordinated algorithms, on the other hand, have a central controller with global information and assign tasks that benefit the overall system. Proper connectivity between UAVs is always required while designing the multi-UAV systems.

在本小节中，提供了在调查任务分配算法时学到的重要经验教训。多架无人机可以在多种应用中使用，以增强覆盖范围，最大限度地降低运维成本，做到实时操作，满足网络要求。任务分配是基于多无人机的应用的重要方面。无人机的任务分配需要考虑多种因素，例如无人机能力、任务执行时间、燃料成本以及动态环境带来的风险。高效的任务分配算法可以保证无人机协同作业，满足应用需求。任务分配采

用多架无人机相互参与、讨论、竞争，或者每架无人机根据对环境约束的观察独立选择任务。分布式算法强调实现效率，而没有考虑全局因素，这可能导致任务分配不准确。另一方面，协调算法有一个具有全局信息的中央控制器，并分配有利于整个系统的任务。在设计多无人机系统时，始终需要无人机之间保持适当的连接。

8. Conclusion

In this article, we have presented a detailed review and discussion of the recent research and development of UAVs in industry as well as academia. UAV fleets are the primary choice for many mission planning systems because of their effectiveness. We have reviewed the prevailing studies on UAV networks and different aspects of UAV communications. Task assignment algorithms play a crucial role in multi-UAV operations for proper cooperation and coordination and need further research. We have extensively surveyed the state-of-the-art task assignment and optimization algorithms for multi-UAV based mission planning systems. We have also reviewed existing algorithms with their advantages, disadvantages, and possible enhancements in the near future. Then, all the task assignment algorithms for UAV networks were classified and compared in terms of their main principles, [operational characteristics](#), and performance metrics.

在本文中，我们对工业界和学术界最近的无人机研究和发展进行了详细的回顾和讨论。无人机机队因其有效性而成为许多任务规划系统的首选。我们回顾了有关无人机网络和无人机通信不同方面的流行研究。任务分配算法在多无人机操作中的适当合作和协调中发挥着至关重要的作用，需要进一步研究。我们广泛研究了基于多无人机的任务规划系统的最先进的任务分配和优化算法。我们还回顾了现有算法的优点、缺点以及在不久的将来可能的增强功能。然后，对无人机网络的所有任务分配算法的主要原理、[运行特点](#)和性能指标进行了分类和比较。

This review has exposed the blossoming attention of researchers and engineers towards the harnessing and utilization of the full potential of UAVs. A number of deep RL algorithms have been suggested in the literature for the mitigation of collision, [trajectory optimization](#), etc. Nevertheless, there is significant scope for designing task assignment algorithms for multi-UAV communications. We believe that this review will assist researchers and engineers in conducting further research on task assignment and optimization algorithms.

这次审查暴露了研究人员和工程师对充分利用无人机潜力的日益关注。文献中已经提出了许多用于减轻碰撞、[轨迹优化](#)等的深度强化学习算法。尽管如此，为多无人机通信设计任务分配算法仍有很大的空间。我们相信这篇综述将有助于研究人员和工程师对任务分配和优化算法进行进一步的研究。

基于联盟形成博弈的异构无人机集群分布式任务分配算法 中国科学:信息科学 南京航空航天大学自动化学院 2024

Distributed task allocation algorithm for heterogeneous unmanned aerial vehicle swarm based on coalition formation game.

- 关键词： 任务分配; 异构无人机集群; 异构资源; 聚类算法; 联盟形成博弈;
- Keywords: task allocation, heterogeneous UAV swarms, heterogeneous resources, clustering algorithm, coalition formation game
- 摘要：
 - 针对无人机集群携带异构资源以及任务的异构需求下的复杂任务分配优化求解问题,提出一种基于联盟形成博弈的分布式任务分配算法.
 - 首先针对任务分配问题规模较大的特点以及资源的异构性,提出一种基于异构资源的改进 K-medoids 聚类算法,通过对无人机集群和任务进行聚类的预处理,降低了任务分配的规模和难度.
 - 考虑任务需求、机载资源以及路径成本等条件建立任务分配模型,将原有任务分配问题转化为联盟划分问题,设计了一种基于联盟形成博弈的分布式任务分配算法进行求解.
 - 最后,将 30 个具有异构需求的任务分配给 100 架携带 3 种异构资源的无人机的仿真结果表明,所提算法能够实现较好的任务分配效果,同时极大提高任务分配的实时性,充分发挥集群效能.
- 基金资助： 国家自然科学基金(批准号:62273177,62020106003,62233009); 江苏省自然科学基金(批准号:BK20211566,BK20222012); 高校学科创新引智基地(批准号:B20007); 空间智能控制技术全国重点实验室开放基金(批准号:HTKJ2023KL502006); 中央高校基本科研业务费(批准号:NI2024001)资助项目;
- doi-link
- 引言
- 问题描述
 1. 数学建模
 - i. 无人机建模
 - ii. 任务建模
 - iii. 联盟形成博弈数学模型
 2. 目标函数与约束条件
- 联盟形成博弈算法设计
 1. 基于异构资源的改进 K-medoids 聚类算法
 - i. 无人机集群聚类
 - ii. 任务聚类
 2. 基于联盟形成博弈的任务分配算法
 3. 算法复杂度分析
- 仿真验证
 1. 基于异构资源的改进 K-medoids 聚类算法结果分析
 2. 基于联盟形成博弈的任务分配算法结果分析
- 结论

引言

常见的分布式任务分配算法主要有博弈论方法、基于市场机制的方法 [7] 和分布式马尔可夫决策过程方法。

- **基于市场机制的方法:**
 - 合同网算法和拍卖算法是应用较为广泛的基于市场机制的方法。
 - 文献 [8] 针对异构无人机和未知目标背景提出了一种**基于合作网协议的任务分配算法**, 能够获得较大的系统效能, 但算法缺少对某无人机可能同属多个任务最优联盟解的冲突消除考虑。
 - 文献 [9] 针对不同作战任务, 建立了一种**面向异构且具有时序约束任务的多轮次分布式拍卖算法**, 本文模型中无人机可执行不同类型的任务, 但缺少对执行相同类型任务的无人机能力的区分。基于分布式马尔可夫法通常训练难度大, 收敛较慢。
- **分布式马尔可夫决策过程方法:**
 - 文献 [10] 针对边缘服务器的分布式任务和资源分配问题, 将其建模为**部分可观测的马尔可夫决策过程**, 提出一种依赖于多智能体的深度强化两步策略, 所提出的解决方案在训练和收敛方面优于基准。
- **博弈论方法:**
 - 目前, 得益于博弈论能够为无人机之间的合作协商提供良好性能, 博弈方法已成为热门研究方向, 应用于各种优化问题 [11]。
 - 文献 [5] 设计了智能体之间的组合策略, 将全局效用映射为势函数获得最优分配方案。
 - 文献 [12] 采用了**基于势博弈的分布式多智能体动态任务分配方法**, 该算法在全局搜索能力方面表现良好。
 - 文献 [13] 设计了**基于偏好联盟形成博弈的任务分配算法**, 利用偏好程度定义机器人加入联盟后的效用, 算法保证了多项式时间复杂性和解的纳什稳定性。
 - 其中, 联盟形成博弈 (coalition formation game, CFG) 模型是常用的博弈模型之一, 目的在于利用无人机之间的相互合作最大化系统性能, 为参与者之间的决策提供了一个高效的模型与工具。在整个博弈过程中, 参与者会自行组成若干个联盟, 从全局来看形成若干分组。
 - 文献 [14] 设计了一种**基于联盟博弈的面向组的分布式聚类算法**, 算法经验证在通信链路稳定性、聚类数量和负载平衡等方面优于现有算法。
 - 文献 [15] 针对无人机前置群干扰提出一种**基于分布式联盟形成博弈的动态任务分配算法**, 将任务分配问题转化为多智能体协同问题, 算法可以实现与集中式优化相似的性能。
 - 文献 [16] 考虑资源属性和任务执行顺序的重叠和互补关系, 在基于顺序重叠联盟形成博弈模型基础上开发了一种**偏好重力引导禁忌搜索算法**, 可获得稳定的联盟结构。

综上所述, 分布式任务分配算法已取得较多研究成果, 尤其在博弈方面。然而, 目前工作仍存在以下问题:

- 一是对于机载资源和任务需求资源的**异构性研究较为薄弱**;
- 二是大规模无人机集群在进行任务分配时求解难度剧增, 导致**算法实时性较差**;
- 三是单个无人机运载能力很难满足任务需求, 需要考虑**无人机之间的协作**。

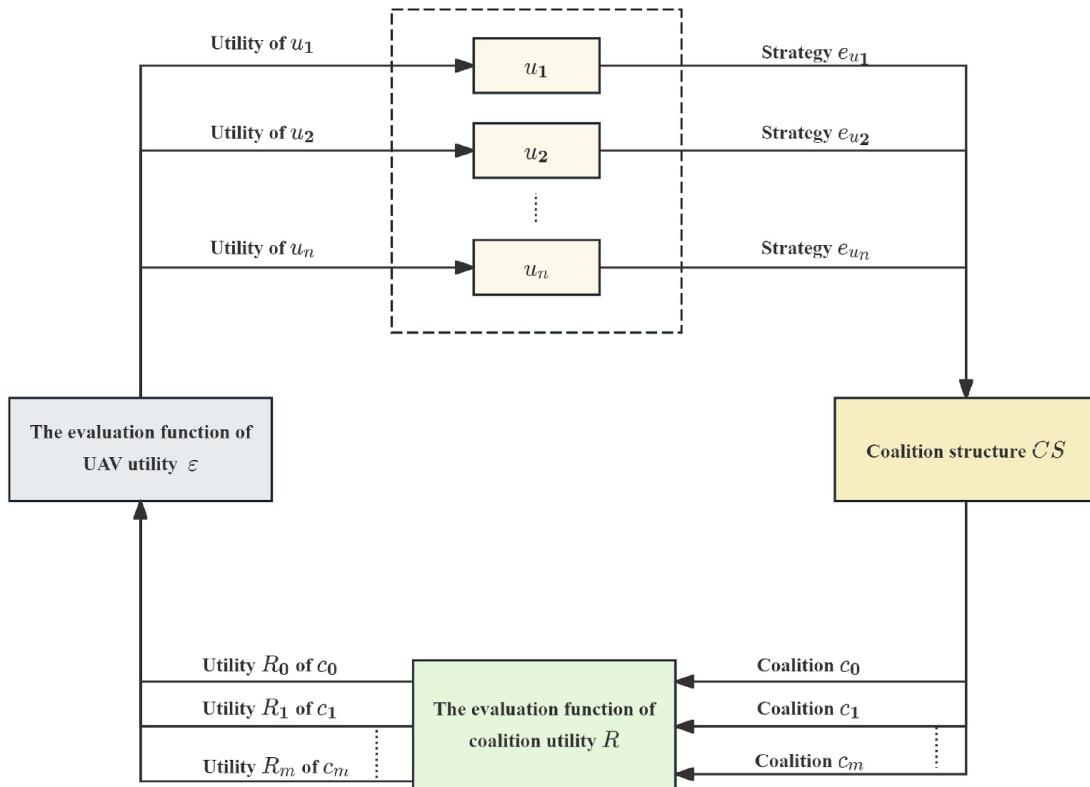
基于联盟形成博弈解决任务分配问题能够:

- 根据无人机在性能、能力和资源方面的差异灵活进行任务分配, 提高系统整体效率;
- 其次具有较高的可拓展性, 能够适应不同规模的无人机集群。同时联盟形成博弈支持分布式决策, 无人机能够自主进行任务分配, 提升系统的鲁棒性和可靠性;
- 通过合理的收益分配机制, 激励无人机积极参与任务执行, 增强协作效果。

因此, 本文提出一种基于联盟形成博弈的分布式任务分配算法, 主要贡献总结如下:

1. 针对原任务分配问题规模大以及资源的异构性, 在 K-medoids 聚类算法的基础上加入关于异构资源数目均衡度的收敛条件以进行改进, 对无人机集群和任务进行合理的预处理, 降低原问题的规模, 提高任

- 务分配的实时性;
2. 建立了联盟形成博弈数学模型, 将任务分配问题转化为联盟划分问题; 通过证明本文联盟形成博弈模型为势博弈, 证明了纳什均衡解的存在性并为算法的设计提供了理论基础;
 3. 设计了基于联盟形成博弈的分布式任务分配算法, 主要包含了最大加权匹配阶段和无人机转移联盟判断阶段, 使得算法相比随机寻优更具有导向性, 能够获得较好的分配效果并且提高任务分配的实时性.



Overall_flowchart_of_the_coalitionFormation_game_model

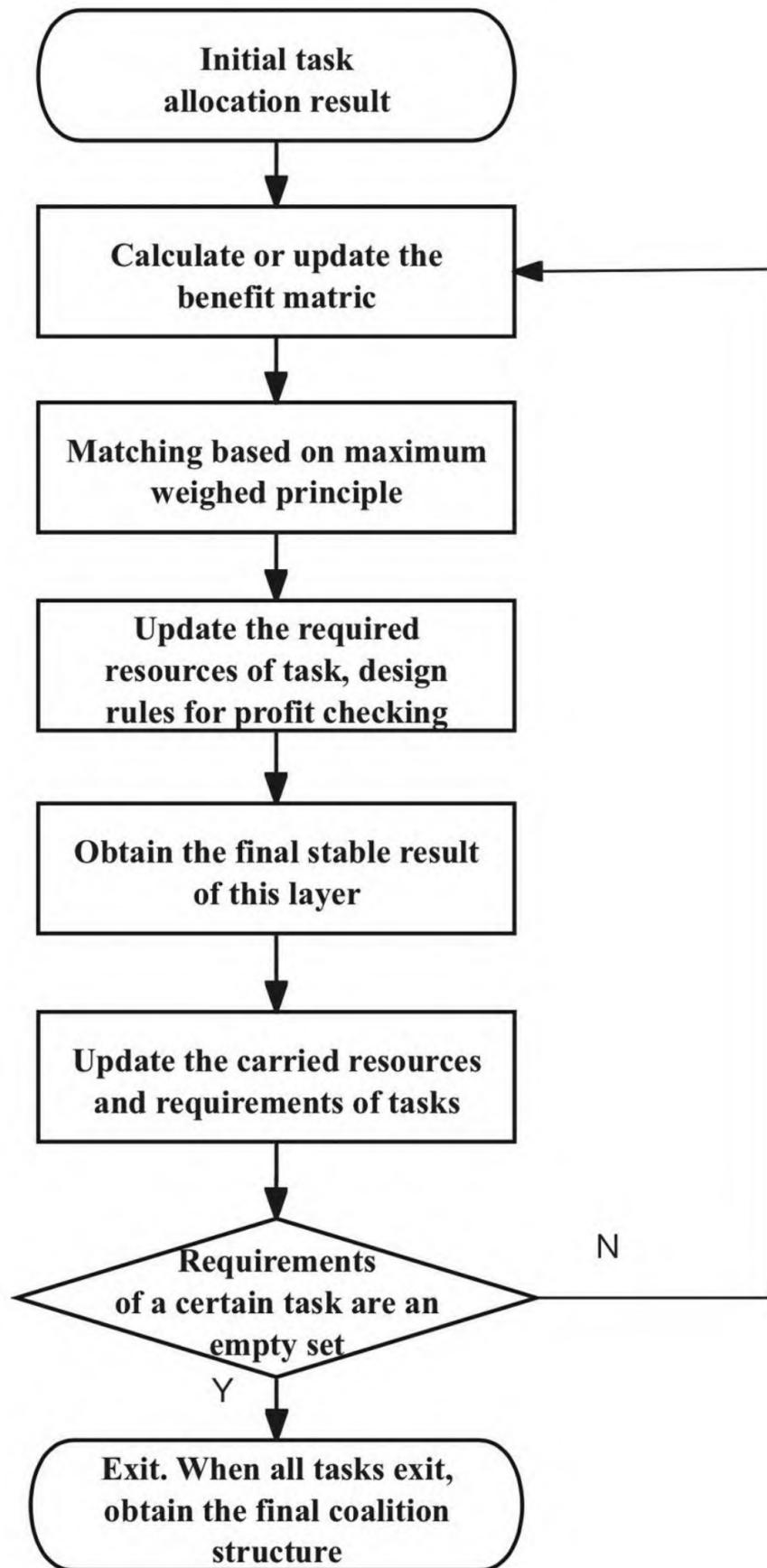


图 2 基于联盟形成博弈的任务分配算法整体流程图

Overall_flowchart_of_the_task_allocation_algorithm_based_on_coalitionFormation_game

List of references

1. Poudel S, Moh S. Task assignment algorithms for unmanned aerial vehicle networks: A comprehensive survey. *IEEE Trans Veh Commun*, 2022, 35: 100469. [doi-link](#)
2. 李鸿一, 陈锦涛, 任鸿儒. Random-sampling-based multi-UAV cooperative search planning for high-rise firefighting. *Sci Sin-Inf*, 2022, 52: 1610-1626. [doi-link](#)
3. Zhou W, Kuang M, Zhu J. An unmanned air combat system based on swarm intelligence. *Sci Sin-Inf*, 2020, 50: 363-374. [doi-link](#)
4. Lei Y Q, Duan H B. Decision-making of multi-UAV combat game via enhanced competitive learning pigeon-inspired optimization. *Sci Sin Tech*, 2024, 54: 136-148. [doi-link](#)
5. Ju K, Mao Z H, Jiang B, et al. Task allocation and reallocation for heterogeneous multiagent systems based on potential game. *Act Autom Sin*, 2022, 48: 2416-2428.
6. Cui W, Li R, Feng Y. Distributed Task Allocation for a Multi-UAV System with Time Window Constraints. *Drones*, 2022, 6: [doi-link](#)
7. Chen X, Wei X M, Xu G Y. Multiple unmanned aerial vehicle decentralized cooperative air combat decision making with fuzzy situation. *J Shanghai Jiaotong Univ*. 2014, 48: 907-913+921.
8. Yan F, Zhu X, Zhou Z. Real-time task allocation for a heterogeneous multi-UAV simultaneous attack. *Sci Sin-Inf*, 2019, 49: 555-569. [doi-link](#)
9. Lyu Y, Zhou R, Li X, et al. Multi-task assignment algorithm based on multi-round distributed auction. *J Beijing Univ Aeronaut Astronaut*, 2023, 1-14.
10. Chen Y, Sun Y, Yu H. Joint Task and Computing Resource Allocation in Distributed Edge Computing Systems via Multi-Agent Deep Reinforcement Learning. *IEEE Trans Netw Sci Eng*, 2024, 11: 3479-3494. [doi-link](#)
11. Xu Y, Jiang B, Yang H. Two-Level Game-Based Distributed Optimal Fault-Tolerant Control for Nonlinear Interconnected Systems. *IEEE Trans Neural Netw Learn Syst*, 2020, 31: 4892-4906. [doi-link](#)
12. Wu H, Shang H. Potential game for dynamic task allocation in multi-agent system. *ISA Trans*, 2020, 102: 208-220. [doi-link](#)
13. Czarnecki E, Dutta A. Scalable hedonic coalition formation for task allocation with heterogeneous robots. *Intel Serv Robotics*, 2021, 14: 501-517. [doi-link](#)
14. Zhang M, Li J, Wu X. Coalition Game Based Distributed Clustering Approach for Group Oriented Unmanned Aerial Vehicle Networks. *Drones*, 2023, 7: [doi-link](#)
15. Zhang T, Wang Y, Ma Z. Task Assignment in UAV-Enabled Front Jammer Swarm: A Coalition Formation Game Approach. *IEEE Trans Aerosp Electron Syst*, 2023, 59: 9562-9575. [doi-link](#)
16. Qi N, Huang Z, Zhou F. A Task-Driven Sequential Overlapping Coalition Formation Game for Resource Allocation in Heterogeneous UAV Networks. *IEEE Trans Mobile Comput*, 2023, 22: 4439-4455. [doi-link](#)
17. Wang J, Jia G, Lin J. Cooperative task allocation for heterogeneous multi-UAV using multi-objective optimization algorithm. *J Cent South Univ*, 2020, 27: 432-448. [doi-link](#)
18. Gao C, Du Y L, Bu Y N, et al. Heterogeneous UAV swarm grouping deployment for complex multiple tasks. *J Syst Eng Electron*, 2024, 46: 972-981.
19. Ma Y, Jiang B, Tao G. Uncertainty decomposition-based fault-tolerant adaptive control of flexible spacecraft. *IEEE Trans Aerosp Electron Syst*, 2015, 51: 1053-1068. [doi-link](#)
20. Mao Z, Jiang B, Shi P. Fault-tolerant control for a class of nonlinear sampled-data systems via a Euler approximate observer. *Automatica*, 2010, 46: 1852-1859. [doi-link](#)

Application of Task Allocation Algorithms in Multi-UAV Intelligent Transportation Systems: A Critical Review

1 Department of Mechanical and Aerospace Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy 都灵理工学院机械与航空航天工程系, Corso Duca degli Abruzzi 24, 10129 都灵, 意大利
 2 School of Language and Literature, Harbin Institute of Technology (Weihai), Weihai, 264200, China 哈尔滨工业大学（威海）语言文学学院, 威海, 264200
 3 Department of Mechanical Engineering, São Carlos School of Engineering, University of São Paulo, Av. Trab. São Carlense, 400, Parque Arnold Schmidt, São Carlos 13566-590, SP, Brazil 圣保罗大学圣卡洛斯工程学院机械工程系, Av.特拉布。 São Carlense, 400, Parque Arnold Schmidt, São Carlos 13566-590, SP, 巴西

Abstract 抽象的

Unmanned aerial vehicles (UAVs), commonly known as drones, are being seen as the most promising type of autonomous vehicles in the context of intelligent transportation system (ITS) technology. A key enabling factor for the current development of ITS technology based on autonomous vehicles is the task allocation architecture. This approach allows tasks to be efficiently assigned to robots of a multi-agent system, taking into account both the robots' capabilities and service requirements. Consequently, this study provides an overview of the application of drones in ITSs, focusing on the applications of task allocation algorithms for UAV networks. Currently, there are different types of algorithms that are employed for task allocation in drone-based intelligent transportation systems, including market-based approaches, game-theory-based algorithms, optimization-based algorithms, machine learning techniques, and other hybrid methodologies. This paper offers a comprehensive literature review of how such approaches are being utilized to optimize the allocation of tasks in UAV-based ITSs. The main characteristics, constraints, and limitations are detailed to highlight their advantages, current achievements, and applicability to different types of UAV-based ITSs. Current research trends in this field as well as gaps in the literature are also thoughtfully discussed.

Keywords:

[UAS](#); [task allocation](#); [aerial robotics](#); [multi-agent system](#); [UAV network](#); [intelligent transportation system](#); [MRTA](#); [optimization](#); [task scheduling](#); [autonomous vehicles](#); [auction](#); [heuristics](#); [dynamic task](#); [multi-UAV](#); [metaheuristics](#); [software architecture](#); [automation](#); [drones](#)

关键词:

[无人机](#); [任务分配](#); [空中机器人技术](#); [多智能体系统](#); [无人机网络](#); [智能交通系统](#); [MRTA](#); [优化](#); [任务调度](#); [自动驾驶汽车](#); [拍卖](#); [启发式](#); [动态任务](#); [多无人机](#); [元启发法](#); [软件架构](#); [自动化](#); [无人机](#)

- Table of Contents 目录
- Abstract 抽象的
- Introduction 介绍
- Game-Theory-Based Algorithms 基于博奕论的算法
- Learning-Based Algorithms 基于学习的算法
- Market-Based Algorithms 基于市场的算法
- Optimization-Based Algorithms 基于优化的算法
- Hybrid Allocation Algorithms 混合分配算法
- Discussion 讨论

-
- Conclusions 结论
 - Funding 资金
 - Acknowledgments 致谢
 - Conflicts of Interest 利益冲突
 - References 参考

Introduction 介绍

The UAV task allocation problem in the context of intelligent transportation systems can be divided into four main categories. First, based on whether UAVs can perform multiple different tasks simultaneously, they are classified as either Single-Task-UAVs (S-T-UAVs) or Multi-Task-UAVs (M-T-UAVs). Second, depending on whether a task requires multi-UAVs to work together, it is categorized as either a Single-UAV-Task (S-UAV-T) or a Multi-UAV-Task (M-UAV-T). Third, depending on whether the drone task allocation is completed in real time, it can be categorized into Online-Task-Allocation (O-T-A) and Offline-Task-Allocation (OF-T-A). Fourth, based on the presence of dependencies between tasks, tasks can be divided into two types: Independent-Tasks (I-T) and Dependent-Tasks (D-T).

智能交通系统背景下的无人机任务分配问题可分为四大类。

- 首先，根据无人机是否可以同时执行多个不同的任务，将其分为单任务无人机（ST-UAV）或多任务无人机（MT-UAV）。
- 其次，根据任务是否需要多无人机协同工作，它被分类为单无人机任务（S-UAV-T）或多无人机任务（M-UAV-T）。
- 第三，根据无人机任务分配是否实时完成，可以分为在线任务分配（OTA）和离线任务分配（OF-TA）。
- 第四，根据任务之间是否存在依赖关系，任务可以分为两种类型：独立任务（IT）和依赖任务（DT）。

The two most used drones in the six task allocation models (S-UAV-T, M-UAV-T, O-T-A, OF-T-A, I-T, and D-T) mentioned above are S-T-UAV and M-T-UAV, and all six task allocation models involve several common objectives, including maximizing the total revenue of the task set, minimizing the flight distance, and minimizing the total cost of the fleet [7]. Some UAV task allocation issues in ITS technology are the same as those previously defined. For example, in distribution systems, due to the large-scale characteristics of some distribution problems, a fleet composed of multiple UAVs needs to cooperate to complete the set of tasks. This M-UAV-T allocation problem is defined as the Vehicle Routing Problem (VRP) [8]. In small-scale delivery systems using a single UAV, the S-UAV-T allocation problem is defined as the Traveling Salesman Problem (TSP). The task allocation problem of a UAV-based ITS is a Non-deterministic Polynomial time (NP-hard) problem. In synthesis, the UAV task allocation problem is the determination of the task sequences for a single UAV or a UAV fleet based on the scope and objectives of the entire task set, thereby ensuring its smooth and efficient completion [9]. At the same time, for the UAV to successfully complete its mission, various constraints of both the task and the UAV need to be considered, including the payload capacity, operational speed, task due date, and the maximum flight distance of the UAV.

上述六种任务分配模型（S-UAV-T、M-UAV-T、OTA、OF-TA、IT 和 DT）中使用最多的两种无人机是 ST-UAV 和 MT-UAV，并且所有六种任务分配模型涉及几个共同目标，包括最大化任务集的总收入、最小化飞行距离以及最小化机队的总成本[7]。ITS 技术中的一些无人机任务分配问题与之前定义的相同。例如，在配送系统中，由于一些配送问题的大规模特征，需要由多架无人机组成的机队协同完成一组任务。这种 M-UAV-T 分配问题被定义为车辆路径问题（VRP）[8]。在使用单个无人机的小规模配送系统中，S-UAV-T 分配问题被定义为旅行商问题（TSP）。基于无人机的智能交通系统的任务分配问题是一

一个非确定性多项式时间（NP-hard）问题。综上所述，无人机任务分配问题是根据整个任务集的范围和目标确定单个无人机或无人机机队的任务序列，从而确保其顺利高效地完成[9]。同时，为了使无人机顺利完成任务，需要考虑任务和无人机的各种约束，包括有效载荷能力、作业速度、任务到期日期和无人机的最大飞行距离。

With the rise of robotic systems technology, the concept of multi-robot task allocation has been established as a dynamic research area in the broad context of operations research applications, and some literature reviews have been recently proposed that also consider UAV-based systems [7,10,11,12]. But, to the best of our knowledge, the literature lacks a critical survey of the application of multi-agent system (MAS)-based task allocation paradigms to a fleet of UAVs conceptualized as an intelligent transportation network. This paper presents a survey of MAS task allocation techniques and their application to drone-based networks for intelligent transportation applications. The main contributions of this work are threefold:

随着机器人系统技术的兴起，多机器人任务分配的概念已被确立为运筹学应用广泛背景下的动态研究领域，并且最近提出了一些文献综述，其中也考虑了基于无人机的系统[7、10、11、12]。但是，据我们所知，文献缺乏对基于多智能体系统（MAS）的任务分配范例在概念化为智能交通网络的无人机机队中的应用的批判性调查。本文对 MAS 任务分配技术及其在基于无人机的智能交通应用网络中的应用进行了调查。这项工作的主要贡献有三个：

- The development of a critical review about MAS task allocation methodologies focusing on multi-UAV networks. This review paper is for engineers, researchers, and scholars who need a critical overview of these emerging topics;
 - 针对多无人机网络的 MAS 任务分配方法进行严格审查。这篇综述论文适用于需要对这些新兴主题进行批判性概述的工程师、研究人员和学者；
- The discussion of state-of-the-art allocation strategies for UAV-based ITSs, focusing on their suitability to the most established applications;
 - 讨论基于无人机的智能交通系统的最先进分配策略，重点关注其对最成熟应用的适用性；
- The discussion of the challenges of task allocation algorithms for UAV-based ITSs as well as the gaps in the literature for informing future trends.
 - 讨论基于无人机的智能交通系统的任务分配算法的挑战以及了解未来趋势的文献中的差距。

This paper is organized as follows. Game-theory-based approaches are presented in [Section 2](#). Learning-based algorithms, auction-based algorithms, and optimization-based allocation algorithms are presented in [Section 3](#), [Section 4](#), and [Section 5](#), respectively. Other hybrid approaches are discussed in [Section 6](#). Finally, a comprehensive discussion of UAV state-of-the-art allocation techniques as well as their pros and cons, their applicability to multi-UAV ITSs, and the current gaps in this field are presented in [Section 7](#). Our conclusions are drawn in [Section 8](#).

本文的结构如下。

- 第2节介绍了基于博弈论的方法。
- 基于学习的算法、基于拍卖的算法和基于优化的分配算法分别在第3节、第4节和第5节中介绍。
- 其他混合方法将在第6节中讨论。
- 最后，第7节全面讨论了无人机最先进的分配技术及其优缺点、它们对多无人机ITS的适用性以及该领域当前的差距。
- 我们的结论在第8节中得出。

Challenges of Task Allocation Algorithms

任务分配算法的挑战

According to the works of [13, 14], UAV task allocation for ITSs can mainly be divided into two categories: OF-T-A (also known as static task allocation) and O-T-A (also known as dynamic task allocation). Unlike static task allocation, dynamic task allocation typically requires the use of fewer computing resources to generate real-time solutions. Centralized algorithms and distributed algorithms are the mainstream algorithms applied to static task allocation and dynamic task allocation, respectively. Currently, algorithms used for static task allocation mainly rely on biologically inspired operators, such as genetic algorithms (GAS) [15], particle swarm optimization (PSO) approaches [16], and differential evolution (DE) algorithms [17], aiming to find approximate optimal solutions in a short period of time. After more than two decades of development, although centralized algorithms have become mature, the aspects of computational time and convergence accuracy still remain significant challenges.

根据[13, 14]的工作，智能交通系统的无人机任务分配主要分为两类：OF-TA（也称为静态任务分配）和OTA（也称为动态任务分配）。与静态任务分配不同，动态任务分配通常需要使用较少的计算资源来生成实时解决方案。集中式算法和分布式算法分别是应用于静态任务分配和动态任务分配的主流算法。目前，用于静态任务分配的算法主要依赖于受生物学启发的算子，例如遗传算法（GAS）[15]、粒子群优化（PSO）方法[16]和差分进化（DE）算法[17]，旨在在短时间内找到近似最优解。经过二十多年的发展，中心化算法虽然已经成熟，但计算时间和收敛精度方面仍然存在重大挑战。

In comparison to OF-T-A algorithms, the development of O-T-A algorithm faces other significant challenges. Firstly, real-time task allocation increases the computational demand of solution algorithms, requiring them to solve NP-hard problems with fewer computational resources. It is well known that real-time algorithms often sacrifice decision quality to ensure their real-time performance; thus, balancing decision quality and algorithmic real-time performance is also a significant challenge. In addition, the generalization capability of the task allocation algorithms poses significant challenges in the task scheduling context of UAV-based ITSs. Determining how algorithms that perform satisfactorily in small-scale networks can adapt to large-scale drone networks has also become an emerging issue for researchers. Finally, the algorithms' robustness presents further challenges. In unexpected situations, such as the loss of control of a drone or inadequate communication network coverage, real-time algorithms should be able to make immediate decisions to ensure that the completion of the task set is not compromised. Therefore, task reallocation is also being addressed in the literature. There are different types of algorithms that are employed in state-of-the-art drone-based intelligent transportation systems, including auction (market)-based approaches, game-theory-based algorithms, optimization-based algorithms, and machine learning (ML) techniques. These approaches and their application to UAV-based ITSs are thoughtfully presented and discussed in the next sections.

与 OF-TA 算法相比，OTA 算法的发展还面临着其他重大挑战。

- 首先，实时任务分配增加了求解算法的计算需求，要求它们用更少的计算资源解决 NP 难题。众所周知，实时算法往往会牺牲决策质量来保证其实时性；因此，平衡决策质量和算法实时性能也是一个重大挑战。
- 此外，任务分配算法的泛化能力对基于无人机的 ITS 的任务调度环境提出了重大挑战。确定在小型网络中表现令人满意的算法如何适应大规模无人机网络也成为研究人员面临的一个新问题。
- 最后，算法的鲁棒性提出了进一步的挑战。在意外情况下，例如无人机失去控制或通信网络覆盖范围不足，实时算法应该能够立即做出决策，以确保任务集的完成不受影响。因此，文献中也讨论了任务重新分配。
- 最先进的基于无人机的智能交通系统采用了不同类型的算法，包括基于拍卖（市场）的方法、基于博弈论的算法、基于优化的算法和机器学习（ML）技术。这些方法及其在基于无人机的 ITS 中的应用将在下一节中进行深入介绍和讨论。

Game-Theory-Based Algorithms 基于博弈论的算法

在[18]的工作中，基于博弈论的自主控制问题被定义并用几个无人机任务规划的例子来说明。基于博弈论的任务分配问题被定义为非结构化环境（随机环境）或具有敌对代理的确定性环境中的决策问题。此后，博弈论算法作为去中心化分布式 M-UAV-T 分配算法得到了广泛的研究[19]。在基于博弈论的智能交通系统无人机任务分配问题中，每架无人机被定义为能够做出决策和执行任务的“玩家”。在实际任务中，由于通信限制，玩家可能完全、部分或不了解网络中的其他无人机。在基于博弈论的 M-UAV-T 分配问题中，一组无人机被定义为 $\$A=\left\{1,2,3, \dots, a\right\}$ 。对于任何无人机 a ，对应的策略集为 $\$D_a=\left\{d_1, d_2, d_3, \dots, d_n\right\}$ 。算法结束时，每架无人机选择的策略对应的一组无人机的收益为 $\$Z=\left\{z_1, z_2, z_3, \dots, z_{n+1}\right\}$ 。参与者、策略集和收益集构成了智能交通系统中多无人机任务分配博弈的三个组成部分。另外，博峦论可以分为两类模型：合作博峦和非合作博峦。

2.1. Non-Cooperative-Game-Based Task Allocation 2.1. 基于非合作博峦的任务分配

2.2. Cooperative Game-Based Task Allocation 2.2. 基于合作博峦的任务分配

Learning-Based Algorithms 基于学习的算法

For real-time task allocation (O-T-A), the learning-based algorithm is another good approach. Compared to traditional artificial neural networks and deep neural networks, reinforcement learning can handle complex tasks and continuously optimize strategies from the optimization process, making it widely used by researchers in real-time task allocation problems for multi-UAVs. In order to solve the M-UAV-T allocation problem, a deep reinforcement learning algorithm is proposed in [27] with the aim of improving the computational efficiency and the convergence accuracy of the task allocation algorithm. Unlike game-theory-based methods, reinforcement-learning-based algorithms typically establish a nonlinear model based on the task allocation problem, as shown in Equation (1).

对于实时任务分配（OTA），基于学习的算法是另一种好方法。与传统的人工神经网络和深度神经网络相比，强化学习可以处理复杂的任务，并从优化过程中不断优化策略，使其被研究人员广泛应用于多无人机的实时任务分配问题。为了解决 M-UAV-T 分配问题，文献[27]提出了一种深度强化学习算法，旨在提高任务分配算法的计算效率和收敛精度。与基于博弈论的方法不同，基于强化学习的算法通常基于任务分配问题建立非线性模型，如方程（1）所示。

The objective function aims to minimize the cost of the problem, which is the same as the reward function in game-based models. $b \leq B_{max}$ denotes a set of constraints considering the boundary conditions.

目标函数旨在最小化问题的成本，这与基于博弈的模型中的奖励函数相同。 $b \leq B_{max}$ 表示考虑边界条件的一组约束。

In addition, the work of [28] also aims at improving the convergence accuracy of the algorithm, thus developing an improved reinforcement learning algorithm. The reinforcement learning algorithm introduces the transfer learning theory. After finding a similar UAV task allocation model in the policy library, the algorithm transfers the training parameter results of the previous source task to the new model through transfer learning. The simulation results show that the algorithm not only effectively improves the performance of UAV task allocation schemes, but also has a strong generalization capability. The authors of [29] developed a multi-agent reinforcement learning method aimed at generating task allocation schemes for heterogeneous UAV fleets. This algorithm can run in locally known environments and has strong robustness.

此外, [28]的工作还旨在提高算法的收敛精度,从而开发出改进的强化学习算法。强化学习算法引入了迁移学习理论。在策略库中找到类似的无人机任务分配模型后,算法通过迁移学习将先前源任务的训练参数结果迁移到新模型中。仿真结果表明,该算法不仅有效提高了无人机任务分配方案的性能,而且具有较强的泛化能力。[29]的作者开发了一种多智能体强化学习方法,旨在为异构无人机机队生成任务分配方案。该算法可以在本地已知的环境中运行,具有很强的鲁棒性。

Ref.	Algorithm	Characteristics	Main Constraints	Limitations
[27]	Deep Q-learning approach	UAVs learn the network state and adapt their locations	Considered all constraints of UAV-based networking tasks.	-
[28]	Deep migration reinforcement learning algorithm based on QMIX	Compared with heuristic algorithms, this method can improve solving efficiency without increasing solving time.	UAV range constraint	Does not consider time constraints for practical scenarios
[29]	Multi-agent reinforcement learning	It can be used in dynamic task scenarios and can achieve real-time task allocation.	Considers the uncertainty of dynamic tasks	-
[30]	Gradient descent method based on deep reinforcement learning	UAVs can automatically and dynamically adjust task allocation strategies in real time.	Time delay of UAV data transmission	Verified only for a specific application scenario

Market-Based Algorithms 基于市场的算法

Auction-based algorithms are widely used for task allocation in drone applications. These algorithms are based on economic principles, as they are alternatively called market-based algorithms, with agents using a negotiation protocol to bid in an auction for task allocation, informed by their local perception of the environment. The agents aim to complete the task assigned with the highest utility or lowest cost by bidding based on the cost or utility they calculate. According to the agents' utility functions, a global objective function is optimized. According to [31], auction-based algorithms present several advantages, including a high solution efficiency and moderate computational costs, in addition to having a dynamic protocol, as they can include or remove new tasks from the allocation procedure.

基于拍卖的算法广泛用于无人机应用中的任务分配。这些算法基于经济原理,因为它们也称为基于市场的算法,代理使用协商协议在任务分配拍卖中竞标,并根据其对环境的本地感知进行了解。代理的目标是根据他们计算的成本或效用进行投标,以最高效用或最低成本完成分配的任务。根据智能体的效用函数,优化全局目标函数。根据[31],基于拍卖的算法除了具有动态协议之外,还具有多种优势,包括高解决效率和适中的计算成本,因为它们可以在分配过程中包含或删除新任务。

The literature presents several works related to auction methodology. A time-sensitive sequential auction (TSSA) algorithm considering time window constraints is proposed in [32] for task allocation in a multi-agent system. An auction-based algorithm for multi-agent task allocation is also proposed in [33]. In this way, auction-based task allocation has received increasing attention since there are different factors that may be considered, including UAVs' capability, battery consumption, execution time, and path routes, among others. The work of [34] proposes an auction-based algorithm for multiple UAVs. A multi-layer cost computation strategy is developed to handle multiple constraints and determine the bid's value.

文献中介绍了一些与拍卖方法相关的著作。[32]提出了一种考虑时间窗口约束的时间敏感顺序拍卖(TSSA)算法，用于多智能体系统中的任务分配。[33]中还提出了一种基于拍卖的多智能体任务分配算法。这样，基于拍卖的任务分配受到了越来越多的关注，因为可以考虑不同的因素，包括无人机的能力、电池消耗、执行时间和路径路线等。[34]的工作提出了一种针对多个无人机的基于拍卖的算法。开发了多层次成本计算策略来处理多个约束并确定出价的价值。

Most of the proposed auction algorithms yield a poor performance for multi-dynamic tasks for multiples drones. To address this issue, a hybrid auction algorithm, based on a decision mechanism and an enhanced objective function, is proposed in [35]. The work of [36] exploits a dynamic decentralized auction-based algorithm for multi-agent systems, such as UAVs. A dynamic task allocation protocol is used, since the agent utilities may change throughout their path towards their targets. This strategy aims to assign a maximum of one task to each member of the fleet, while the same task can be allocated to multiple agents. Thus, the task utilities are calculated according to the agent's states; i.e., they depend on both the rewards from the accomplishment of the assigned tasks and the costs associated with their execution. Differently from game-based algorithms that may not always achieve high levels of global utility, the auction-based algorithm is able to greedily achieve global utility, due to its simplicity and fast convergence.

大多数提出的拍卖算法对于多架无人机的多动态任务性能较差。为了解决这个问题，[35]提出了一种基于决策机制和增强目标函数的混合拍卖算法。[36]的工作为多代理系统（例如无人机）开发了一种基于动态去中心化拍卖的算法。使用动态任务分配协议，因为代理实用程序可能会在其通往目标的整个路径中发生变化。该策略旨在为舰队的每个成员最多分配一项任务，而同一任务可以分配给多个代理。因此，任务效用是根据智能体的状态计算的；即，它们既取决于完成指定任务的回报，也取决于与执行任务相关的成本。与基于游戏的算法可能并不总能实现高水平的全局效用不同，基于拍卖的算法由于其简单性和快速收敛而能够贪婪地实现全局效用。

The use of different auction-based algorithms to solve a heterogenous task allocation problem for multiple UAVs in a drone delivery context is investigated in [37]. The strategy is used to minimize the battery consumption of a UAS-based parcel transportation service by allocating delivery tasks with due date constraints to multiple drones that demand a lower consumption of energy. The allocation of charge tasks is also addressed. These auction-based algorithms were implemented by means of both single-item and multiple-item strategies. Scalar constrained optimization problems are solved by each agent to calculate the UAV's bid for each task. For delivery tasks, the protocol's bid is related to the consumption of energy, while the flight time is chosen for charge task bids. Path planning is also included in the framework to compute the risk-aware path for each task-UAV bid by means of a 2D risk map of the operational area.

[37]研究了使用不同的基于拍卖的算法来解决无人机交付环境中多个无人机的异构任务分配问题。该策略用于通过将具有到期日期限制的交付任务分配给需要较低能源消耗的多架无人机来最大限度地减少基于无人机的包裹运输服务的电池消耗。还解决了充电任务的分配问题。这些基于拍卖的算法是通过单项和多项策略来实现的。每个代理解决标量约束优化问题，以计算无人机对每个任务的出价。对于投递任务，协议的出价与能量消耗相关，而对于充电任务的出价则选择飞行时间。框架中还包含路径规划，通过操作区域的二维风险图来计算每个任务无人机投标的风险感知路径。

The work of [38] investigates the use of a second price auction algorithm for drone intelligent transportation. A deep learning methodology is also included to enhance the auction algorithm's robustness by revenue optimality. In addition, an improved implementation of an auction algorithm for drone delivery is addressed in [39]. Lightweight distributed task allocation is proposed for this application, simplifying the management of delivery and charge tasks with minimal energy consumption. Each agent runs a decentralized protocol, running path planning and optimization algorithms. In contrast to conventional auction-based methods for task scheduling, each agent is designed to function as both the network's auctioneer and bidder, depending on the task type. Recent works combine the auction-based algorithm with other methodologies, as seen in [40,41,42].

[38]的工作研究了无人机智能交通的第二价格拍卖算法的使用。还包括深度学习方法，以通过收入最优性增强拍卖算法的稳健性。此外，[39]中还讨论了无人机交付拍卖算法的改进实现。针对该应用提出了轻量级分布式任务分配，以最小的能耗简化配送和收费任务的管理。每个代理运行一个去中心化协议，运行路径规划和优化算法。与传统的基于拍卖的任务调度方法相比，每个代理都被设计为根据任务类型同时充当网络的拍卖人和投标人。最近的工作将基于拍卖的算法与其他方法相结合，如[40 41 42]中所示。

Table 4 provides an overview of the reviewed approaches, including the main characteristics, constraints, and limitations.

表 4概述了所审查的方法，包括主要特征、约束和限制。

Ref.	Algorithm	Characteristics	Main Constraints	Limitations
[32]	Time-Sensitive Sequential Auction	Improved allocation of tasks that have time constraints	Time window deadlines	-
[33]	Auction	Increases robustness and non-exclusive task assignment	Battery consumption, execution time, and path	Poor performance when tasks could saddle agents with leaden tasks
[34]	Auction-based Multiple Constraints	Solves multiple constraints and provides a way of calculating the price of a bid	Sensor, time window, and fuel cost	Most of the parameters are variable, but the area is fixed. The effectiveness is not investigated.
[35]	Hybrid Auction Algorithm	Promotes its performance and robustness in dynamic task assignment and avoids obstacles	Mission cost, coverage factor	Each UAV can only perform limited tasks and must return to the base to replenish resources
[36]	Greedy Coalition Auction	Allows for dynamic task allocation for spatially distributed multi-agent systems with a positive time efficiency	Path and targets	In the presence of large fleet of autonomous systems, scalability issues may arise due to the high computation cost
[37]	Greedy Auction	Able to effectively handle the complexity and heterogeneity of the problem	Energy efficiency, task due dates, safe path planning	Distributed implementation is not addressed
[38]	Learning-Based Second Price Auction	Enables the algorithm to be truthful, distributed, and scalable	Energy consumption	The data performance is limited to investigate the proposed conditions
[39]	Multi-Auctioneer Market-Based	Enables one to tackle tasks with temporal constraints, minimizing the heterogeneous fleet of UAVs' energy consumption	Comprehensive optimization of energy consumption, hard task due dates	Robustness to lossy communication network is not addressed

Ref.	Algorithm	Characteristics	Main Constraints	Limitations
[40]	Neural Myerson Auction	Designed for UAV charging scheduling. It can provide collision avoidance to build secure and privacy-preserving systems	Energy consumption and cluster selection	The external forces, such as wind and other physical factors, are not considered
[41]	Improved Multi-Objective Auction	Improves the setting of the quotation threshold parameters by the distance factor and designs an adaptive operator strategy	Distance and target	-
[42]	Combinatorial Double Auction	Yields a set of feasible solutions for undertaking complex winner determination problem models	Costs and market satisfaction	Unavoidable limitation regarding the data simulation procedures

Optimization-Based Algorithms 基于优化的算法

The optimization methodology is widely used in applied mathematics to find the optimal solution to a specific problem. The goal of the optimization is to reduce costs or maximize profit through an objective function, aiming to find the best solution from a set of possible solutions. Various constraints can be applied to optimize the cost function and achieve an improved solution. A variety of optimization techniques are evaluated, including three main groups: deterministic, metaheuristic (or stochastic), and heuristic. Methods based on deterministic optimization do not consider randomness; i.e., the output is equal when the same initial condition is adopted. Graphical methods, sequential and linear programming, and mixed integer linear programming (MILP) are some examples of deterministic techniques. Stochastic methods, on the other hand, include randomness in the algorithm, leading to different outcomes even in the presence of the same initial conditions. Evolutionary algorithms, swarm intelligence, Monte Carlo methods, and simulated annealing are some of the current examples for this group of algorithms. Furthermore, heuristic algorithms are an interesting alternative to deterministic methods (that yield a high computational cost), providing fast solutions in good computational time. Heuristic algorithms use practical approaches and shortcuts to obtain solutions that are not necessarily optimal, but sufficient for finding good local solutions.

优化方法广泛应用于应用数学中，以寻找特定问题的最优解决方案。优化的目标是通过目标函数降低成本或最大化利润，旨在从一组可能的解决方案中找到最佳解决方案。可以应用各种约束来优化成本函数并实现改进的解决方案。评估了各种优化技术，包括三个主要组：确定性、元启发式（或随机）和启发式。基于确定性优化的方法不考虑随机性；即采用相同初始条件时，输出相等。图形方法、顺序和线性规划以及混合整数线性规划（MILP）是确定性技术的一些示例。另一方面，随机方法在算法中包含随机性，即使存在相同的初始条件也会导致不同的结果。进化算法、群体智能、蒙特卡罗方法和模拟退火是这组算法的一些当前示例。此外，启发式算法是确定性方法（计算成本较高）的有趣替代方案，可以在良好的计算时间内提供快速解决方案。启发式算法使用实用的方法和捷径来获得不一定是最优的解决方案，但足以找到良好的局部解决方案。

5.1. Deterministic 5.1.确定性

Hybrid Allocation Algorithms 混合分配算法

Hybrid allocation strategies are also designed to improve the transportation efficiency for UAV-based ITSs by merging different types of allocation algorithms.

混合分配策略还旨在通过合并不同类型的分配算法来提高基于无人机的智能交通系统的运输效率。

The work of [74] combines mixed-integer linear programming (MILP) and a simulated annealing (SA)-based heuristic algorithm to find optimal routes for a drone delivery application. Both battery consumption and payload weight are considered to calculate the drone's energy consumption. MILP is employed to minimize costs and the delivery time up to a budget constraint, while the SA is used to find suboptimal solutions of practical scenarios, i.e., the relation between the delivery time and budget. As a drawback, the SA algorithm fails to utilize geographical information to attenuate the choice of impractical routes. Hybrid algorithms can also be used to investigate the optimization of the service itself, as in [75]. Particle swarm optimization (PSO) and the grey wolf optimizer (GWO) are combined to minimize the number of deployed drones, cost, and flight time. The proposed algorithm incorporates different strategies, such as interval transformation, dynamic weighting rules, and a nonlinear convergence factor, to enhance the performance accuracy and to lower the cost. [74]的工作结合了混合整数线性规划 (MILP) 和基于模拟退火 (SA) 的启发式算法来寻找无人机送货应用的最佳路线。计算无人机的能耗时会考虑电池消耗和有效负载重量。MILP 用于在预算约束下最小化成本和交付时间，而 SA 用于寻找实际场景的次优解决方案，即交付时间与预算之间的关系。SA 算法的缺点是无法利用地理信息来减少不切实际的路线的选择。混合算法也可以用于研究服务本身的优化，如[75]中所示。粒子群优化 (PSO) 和灰狼优化器 (GWO) 相结合，可以最大程度地减少部署的无人机数量、成本和飞行时间。该算法结合了区间变换、动态加权规则和非线性收敛因子等不同策略，以提高性能精度并降低成本。

The work of [76] combines the Hungarian algorithm (HA) and machine learning (ML) to optimize the task assignment problem in a drone delivery application. Different mathematical models, such as linear and polynomial regressions, are used to generate distinct cost functions, based on distance and time metrics. Once the cost function is estimated, the Hungarian algorithm is employed for solving the drone intelligent delivery problem. The Hungarian algorithm is defined by a matrix of costs, which represents the cost of each agent or task.

[76]的工作结合了匈牙利算法 (HA) 和机器学习 (ML) 来优化无人机交付应用中的任务分配问题。不同的数学模型（例如线性和多项式回归）用于根据距离和时间度量生成不同的成本函数。一旦成本函数被估计出来，匈牙利算法就被用来解决无人机智能交付问题。匈牙利算法由成本矩阵定义，该矩阵表示每个代理或任务的成本。

Similarly, the combination of a multi-agent RL algorithm and a conflict-free method is investigated in [77] to optimize task allocation and path planning for multiple drones. The strategy guarantees that the shortest path is chosen for the drones, while multi-agent proximal policy optimization (MAPPO) enables collision avoidance between the drones. The work of [78] investigates the design of a rapid allocation algorithm, based on the combination of a greedy auction algorithm and a reassignment strategy. The combination of both strategies enables swift and effective responses, which result in a rapid and efficient completion of tasks while preventing the occurrence of deadlocks. In addition, neural networks are widely employed for task allocation [79]. An assisted learning invasive encroachment neutralization (ALIEN) technique is designed for a secure drone transportation system. The objective function of the ALIEN algorithm aims to maximize the security of the drone transportation system, and it is represented by $\max I = t_i D_i + t_i D^d \{r \lambda_i\} + t_i D \{N_i\}$, where the decision variables represent drone detection, object recognition, and neutralization, respectively.

类似地，[77]研究了多智能体强化学习算法和无冲突方法的结合，以优化多架无人机的任务分配和路径规划。该策略保证为无人机选择最短路径，而多智能体近端策略优化（MAPPO）可以避免无人机之间的碰撞。[78]的工作研究了基于贪婪拍卖算法和重新分配策略的组合的快速分配算法的设计。两种策略的结合可以实现快速有效的响应，从而快速高效地完成任务，同时防止死锁的发生。此外，神经网络广泛用于任务分配[79]。辅助学习侵入性侵占中和（ALIEN）技术是为安全的无人机运输系统而设计的。ALIEN 算法的目标函数旨在最大化无人机运输系统的安全性，其表示为 $I_{max} = t_{ID_i} + t_{ID} \lambda_i + t_{ID\{N_i\}}$ ，其中决策变量分别代表无人机检测、物体识别和中和。

Moreover, the maximum task allocation algorithm is proposed in [80]. The maximum task allocation algorithm for multiple UAVs under time constraints has been significant in meeting requirements for quality of service. The TRMaxAlloc algorithm is designed based on two phases: assignment and reassignment. The PI algorithm is used, in the first stage, to allocate the tasks to the drones, while, in the next stage, the proposed TRMaxAlloc algorithm enables the creation of feasible time slots for the unassigned tasks. As a result, the assignment enables each task to be completed before its corresponding deadline with a lower time cost. Each UAV is assigned to several tasks, as described by the task list $\{s_1, s_2, \dots, s_N\}$. Then, the cost function, expressed as $J = \max |s_i|$, aims to optimize the task allocation problem.

此外，[80]中提出了最大任务分配算法。时间约束下多无人机最大任务分配算法对于满足服务质量要求具有重要意义。TRMaxAlloc 算法的设计基于两个阶段：分配和重新分配。在第一阶段，使用 PI 算法将任务分配给无人机，而在下一阶段，所提出的 TRMaxAlloc 算法可以为未分配的任务创建可行的时隙。因此，分配使每个任务能够在相应的截止日期之前以较低的时间成本完成。每架无人机被分配多个任务，如任务列表所述 $\{s_1, s_2, \dots, s_N\}$ 。那么，成本函数，表示为 $J = \max |s_i|$ ，旨在优化任务分配问题。

A real-time market-based task allocation mechanism is proposed in [81] for a dynamic coalition formation (DCF) problem. Autonomous agents can collaborate independently, creating an optimal global coalition structure to efficiently execute the emerging tasks. The auction algorithm is used for real-time assignment, and a mutual-selection method is employed for obtaining an improved performance in terms of the agent utilization rate and task completion rate. In addition, the work of [82] investigated the combination of a distributed evolutionary algorithm and a greedy algorithm to simultaneously optimize multiple objective functions. This combined methodology aims to improve the model's local optimizing ability with different constraints, such as spatial constraints, time costs, and energy consumption. The proposed strategy aims to efficiently solve large-scale task allocation problems with enhanced and more diverse non-dominated solutions.

[81]针对动态联盟形成（DCF）问题提出了一种基于市场的实时任务分配机制。自治代理可以独立协作，创建最佳的全球联盟结构，以有效地执行新兴任务。采用拍卖算法进行实时分配，并采用相互选择的方法在代理利用率和任务完成率方面获得改进的性能。此外，[82]的工作研究了分布式进化算法和贪婪算法的组合来同时优化多个目标函数。这种组合方法旨在提高模型在不同约束下的局部优化能力，例如空间约束、时间成本和能量消耗。所提出的策略旨在通过增强且更多样化的非支配解决方案来有效解决大规模任务分配问题。

Discussion

The primary objective of this paper is to serve as a state-of-the-art reference for researchers and engineers about the science of task allocation applied to various UAV-based ITSs. Such technology is foreseen to become popular in the next decades in different contexts: smart cities, urban air mobility, smart logistics, connected vehicles, etc. 本文的主要目的是为研究人员和工程师提供有关应用于各种基于无人机的 ITS 的任务分配科学的最新参考。预计此类技术将在未来几十年在不同的环境中流行：智慧城市、城市空中交通、智能物流、互联车辆等。

The main conclusions drawn from this survey are summarized as follows:

本次调查得出的主要结论总结如下：

- Market-based allocation algorithms are, in general, less computationally demanding than other methods, but the bidding procedure has to be designed carefully to avoid unfair allocations. Market-based allocation architectures should be developed for applications with a high level of autonomy and inherent dynamicity (e.g., parcel delivery, traffic monitoring, search and rescue, and passenger transportation), with the drones being able to adjust their bids based on their current status as well as both the service demand and the environmental conditions;

一般来说，基于市场的分配算法比其他方法的计算要求较低，但必须仔细设计投标程序以避免不公平的分配。应为具有高度自主性和固有动态性的应用（例如包裹递送、交通监控、搜索和救援以及客运）开发基于市场的分配架构，无人机能够根据当前情况调整其出价状况以及服务需求和环境条件；

- Optimization-based approaches produce more efficient allocation but should be used to allocate tasks to UAVs in static scenarios with well-defined constraints (e.g., inspection and data collection). The main drawback is the scalability of these approaches with larger fleets due to their computational complexity. Also, complex application scenarios may be difficult to model, and discrepancies between a real application and a simulation model may severely affect the quality of the obtained solution;

基于优化的方法可以产生更有效的分配，但应该用于在具有明确定义的约束（例如检查和数据收集）的静态场景中将任务分配给无人机。主要缺点是由于计算复杂性，这些方法在较大车队中的可扩展性。此外，复杂的应用场景可能难以建模，实际应用与仿真模型之间的差异可能会严重影响所获得解决方案的质量；

- Learning-based task allocation algorithms are suitable for highly dynamic scenarios in which the UAVs can exploit large datasets of past experiences to adapt to variable environmental conditions. A preferable application can be identified as the UAV traffic monitoring service. On the other hand, a learning-based task allocation architecture is not suitable for every type of scenario involving environmental variability; for instance, considering a critical emergency scenario such as disaster response, the trustworthiness of UAV task allocations plays a crucial role, thereby limiting the deployment of such an allocation architecture. Also, the questionable level of generalizability to unseen conditions may be a limiting factor;

基于学习的任务分配算法适用于高度动态的场景，其中无人机可以利用过去经验的大型数据集来适应可变的环境条件。一个优选的应用是无人机交通监控服务。另一方面，基于学习的任务分配架构并不适合所有涉及环境变化的场景。例如，考虑到灾难响应等关键紧急情况，无人机任务分配的可信度起着至关重要的作用，从而限制了这种分配架构的部署。此外，对未见条件的普遍适用性值得怀疑，这可能是一个限制因素；

- Game-theory-based approaches are well suited for applications in which the UAVs can compete against one another or cooperate in the completion of a task with well-defined utilities. Coverage and traffic monitoring tasks represent a valid example since the UAVs of the ITS can compete for the best coverage/monitoring location. The limitations of a game-theory-based task allocation strategy in UAV-based ITS contexts are both the computational burden with large fleets and the capability of the utility function to adequately represent the real-world reward related to the allocation;

基于博弈论的方法非常适合无人机可以相互竞争或与明确定义的实用程序合作完成任务的应用。覆盖和交通监控任务是一个有效的例子，因为 ITS 的无人机可以竞争最佳的覆盖/监控位置。在基于无人机的 ITS 环境中，基于博弈论的任务分配策略的局限性既在于大型机队的计算负担，又在于效用函数充分代表与分配相关的现实世界奖励的能力；

- The design of a hybrid allocation architecture incorporating multiple approaches is the most promising strategy for leveraging the characteristics of each method, thus enhancing the capability of the allocation algorithm to meet the requirements of (i) the environment, (ii) the service, and (iii) the UAV-based ITS. Also, hybrid allocation algorithms feature a higher generalization capability with respect to both the service and the robot type.

结合多种方法的混合分配架构的设计是利用每种方法的特性的最有前途的策略，从而增强分配算法的能力以满足（i）环境、（ii）服务和（iii）基于无人机的智能交通系统。此外，混合分配算法对于服务和机器人类型都具有更高的泛化能力。

Finally, Table 8 summarizes the characteristics of the allocation methods in terms of computational cost, efficiency in finding optimal solutions, scalability to large fleets, capability of handling dynamic tasks, robots, and environments, and the most suitable application domains in the context of UAV-based ITSs.

最后，表 8 总结了分配方法的特征，包括计算成本、寻找最佳解决方案的效率、大型机队的可扩展性、处理动态任务、机器人和环境的能力，以及无人机背景下最合适的应用领域基于 ITS。

Algorithm	Cost	Efficiency	Scalability	Dynamic Tasks and Robots	Dynamic Environment	Application
Auction	+	++	++++	++++	++++	D, TM, SR
Learning	++ +	++++	+++	+++++	+++++	TM, DR, A
Game Theory	++ +	++++	+++	+++	+++	C, TM, DC
Deterministic	++ ++ +	+++++	+	+	+	I, DC
Heuristic	++	+++	++++	+++	++	D, C
Metaheuristic	++ +	++++	+++	+++	+++	D, C

8. Conclusions 八、结论

For the sake of completeness, before delving into a discussion of the reviewed methodologies along with the applicability of the task allocation methods to the most established applications of UAVs in the context of ITS technology, the main applications of UAVs in the context of ITS technology are listed as follows:

- Search and rescue (SR) [83,84,85,86];
- Delivery (D) [87,88,89];
- Traffic monitoring (TM) [90];
- Inspection (I) [91,92,93];
- Disaster response (DR) [94,95,96];
- Surveillance (S) [97,98];

-
- Coverage (C) [99,100];
 - Data collection (DC) [101];
 - Smart mobility (SM) [102];
 - Agriculture (A) [103,104,105].

The use of unmanned aerial vehicles has gained significant attention in the context of intelligent transportation. The use of different sensors and high-resolution cameras enables the drones to support road transportation vehicles and to be used for a variety of parcel delivery tasks, among other applications. However, a scalable and efficient task allocation architecture must be designed for optimizing the coordination of the fleet of drones of an intelligent transportation system. Generally, task allocation is used to minimize the execution time of the tasks with a reliable and well-defined procedure. A categorization can be defined depending on the number and type of vehicles and tasks employed, including single-task UAVs or multi-task UAVs, single-UAV tasks or multi-UAV tasks, online or offline task allocation, and independent or dependent tasks. In addition, a combination of multiple UAVs and trucks as well as the inclusion of several constraints can significantly improve the overall efficiency. Therefore, the constant development of task allocation enables us to create more efficient methodologies that cover a large variety of scenarios.

在智能交通领域，无人机的使用受到了广泛关注。通过使用不同的传感器和高分辨率摄像头，无人机可以支持道路运输车辆，并可用于各种包裹递送任务以及其他应用。然而，必须设计一个可扩展且高效的任务分配架构，以优化智能交通系统无人机机队的协调。通常，任务分配用于通过可靠且定义明确的过程来最小化任务的执行时间。可以根据所使用的车辆和任务的数量和类型来定义分类，包括单任务无人机或多任务无人机、单无人机任务或多无人机任务、在线或离线任务分配以及独立或相关任务。此外，多个无人机和卡车的组合以及包含多个约束可以显着提高整体效率。因此，任务分配的不断发展使我们能够创建更有效的方法来覆盖各种场景。

In this sense, this paper provides a comprehensive literature review of how such approaches are being utilized to optimize the allocation of tasks in UAV-based ITSs. Market-based algorithms, game-theory-based algorithms, optimization-based algorithms, machine learning techniques, and other hybrid methodologies are reviewed and discussed. Furthermore, the main applications of unmanned aerial vehicles in ITSs are presented as well as the suitability of the task allocation algorithms presented throughout the paper with respect to the different applications. The main characteristics of, limitations of, and differences between the algorithms are highlighted, showing their main uses over the last few years. Understanding the main characteristics and the applicability of each type of allocation enables engineers and researchers to properly choose the most appropriate type of task scheduling logic. Moreover, the emerging trends and gaps in the literature are also discussed.

从这个意义上说，本文对如何利用这些方法来优化基于无人机的智能交通系统中的任务分配进行了全面的文献综述。回顾和讨论了基于市场的算法、基于博弈论的算法、基于优化的算法、机器学习技术和其他混合方法。此外，还介绍了无人机在智能交通系统中的主要应用，以及本文提出的任务分配算法对于不同应用的适用性。突出显示了算法的主要特征、局限性和差异，显示了它们在过去几年中的主要用途。了解每种分配类型的主要特征和适用性使工程师和研究人员能够正确选择最合适的任务调度逻辑类型。此外，还讨论了文献中的新兴趋势和差距。

In conclusion, we stress the importance of considering the requirements of the service as well as the environmental conditions and the operational capability of the UAV-based intelligent transportation system when designing a task allocation strategy.

总之，我们强调在设计任务分配策略时考虑服务要求以及基于无人机的智能交通系统的环境条件和运行能力的重要性。

As a further consideration, it is worth noticing that the design of communication channels and their security are fundamental for both implementing (if the allocation architecture is fully decentralized) and validating (if the allocation architecture is centralized or distributed) the allocation of tasks in a fleet of UAVs. Also, the security of the communication channels is a significant challenge for achieving a safe, regulatory-compliant, and resilient real-world deployment of a UAV-based ITS. The efficiency of the task allocation process can be heavily influenced by aspects such as the security of the communication channels as well as their fallibility.

作为进一步的考虑，值得注意的是，通信通道的设计及其安全性对于实现（如果分配架构是完全去中心化的）和验证（如果分配架构是集中式的还是分布式的）任务分配至关重要。无人机机队。此外，通信通道的安全性是实现基于无人机的 ITS 的安全、合规和弹性现实部署的重大挑战。任务分配过程的效率可能会受到通信通道的安全性及其可靠性等方面严重影响。

Future survey-based research will focus on investigating how the UAV communication protocol can influence the efficiency of the task allocation architecture in terms of both security and robustness to fallible communication networks. Also, conceptual modelling frameworks used to implement task allocation algorithms in UAV-based ITSs may also be discussed.

未来基于调查的研究将重点调查无人机通信协议如何影响任务分配架构的效率，包括安全性和对易出错通信网络的鲁棒性。此外，还可以讨论用于在基于无人机的 ITS 中实现任务分配算法的概念建模框架。

Task Assignment

- Task assignment algorithms for unmanned aerial vehicle networks: A comprehensive survey 2022.06

差分高斯光栅化

[Differential Gaussian Rasterization](#)

= GPT =>

[Differential Gaussian Rasterization: CPU Version with C Kernel](#)

Learning Game Development

links

- [awesome-gamedev](#)
- [youtube: godot tutorials videos](#)
- [Learn GDScript From Zero app](#)
- [godotengine](#)

Coalitional Games: 联盟博弈

A coalitional game, also known as a cooperative game, is a game theory model where players form groups to achieve a collective goal. In a coalitional game, players form coalitions, or binding agreements, to strengthen their positions and act as a single entity. Coalitional games are different from non-cooperative games, where players cannot form alliances or agreements must be self-enforced.

联盟博弈，也称为合作博弈，是一种博弈论模型，其中玩家组成团队以实现集体目标。在联盟游戏中，玩家组成联盟或具有约束力的协议，以加强他们的地位并作为一个实体行动。联盟游戏与非合作游戏不同，在非合作游戏中，玩家不能结成联盟或协议必须自行执行。

Here are some key concepts in coalitional games:

- Coalition: A group of players that form a binding agreement
- Coalition value: The worth of a coalition in a game, which is denoted by the symbol (v)
- The core: The set of payoff allocations that ensures no group of players has an incentive to leave their coalition
- Shapley value: An efficient solution concept that is recommended for games with a low number of players

Game Theory

- Coalitional Games: 联盟博弈

Jinja2

```
from jinja2 import Template

# 定义模板
template = Template(open("test.jinja2").read())

# 渲染模板
output = template.render(target="RISCV")

# 打印生成的 HTML
print(output)
```

Lalrpop

LR(1) parser generator for Rust.

[LALRPOP 拉尔波普](#)

LALRPOP is a parser generator, similar in principle to [YACC](#), [ANTLR](#), [Menhir](#), and other such programs. In general, it has the grand ambition of being the most usable parser generator ever. This ambition is most certainly not fully realized: right now, it's fairly standard, maybe even a bit subpar in some areas. But hey, it's young. For the most part, this README is intended to describe the current behavior of LALRPOP, but in some places it includes notes for planned future changes.

LALRPOP 是一个解析器生成器，原理类似于 YACC、ANTLR、Menhir 和其他此类程序。总的来说，它有一个宏伟的雄心壮志，那就是成为有史以来最有用的解析器生成器。这个雄心壮志肯定没有完全实现：现在，它相当标准，甚至在某些领域可能有点低于标准。但是，嘿，它很年轻。在大多数情况下，本 README 旨在描述 LALRPOP 的当前行为，但在某些地方，它包括计划的未来更改的说明。

1. add lalrpop to rust project
2. define grammar in lalrpop file
 - i. define asts, build ast
 - ii. use macros
 - iii. use errors and error recovery
 - iv. pass param to parser

Tutorial

- `< ... >` : means extract the value of the expression inside the angle brackets, here is `...`

Macros

four built-in macros:

- `? : Expr? get Option<Box<Expr>>`
- `* : Expr* get Vec<Expr>>, minimum 0`
- `+ : Expr+ get Vec<Expr>>, minimum 1`
- `(...) : short for creating an nonterminal,`
 - `(<Expr> ",")?` , mean an "optionally parse an Expr followed by a comma"
 - Note the **angle brackets** (`<>`) around `Expr` : these ensures that the value of the `(<Expr> ",")` is the value of the expression, and not a tuple of the expression and the comma.
 - get type `Option<Box<Expr>>`

Grammar Example

```
// Term: Num | '(' Term ')'
pub Term: i32 = {
    <n: Num> => n,
    "(" <t:Term> ")" => t,
}
// Num: r"[0-9]+"
Num: i32 = <s:r"[0-9]+"> => i32::from_str(&s).unwrap();
```

Precedence and associativity can be specified using attributes:

```
pub Expr: i32 = {
    #[precedence(level="0")] // Highest precedence
    Term,
    #[precedence(level="1")] #[assoc(side="left")]
    <l:Expr> "*" <r:Expr> => l * r,
    <l:Expr> "/" <r:Expr> => l / r,
    #[precedence(level="2")] #[assoc(side="left")]
    <l:Expr> "+" <r:Expr> => l + r,
    <l:Expr> "-" <r:Expr> => l - r,
};
```

Use AST to represent the parsed tree:

```
// ast.rs

use std::fmt::{Debug, Error, Formatter};

pub enum Expr {
    Number(i32),
    Op(Box<Expr>, Opcode, Box<Expr>),
    Error,
}

pub enum ExprSymbol<'input> {
    NumSymbol(&'input str),
    Op(Box<ExprSymbol<'input>>, Opcode, Box<ExprSymbol<'input>>),
    Error,
}

#[derive(Copy, Clone)]
pub enum Opcode {
    Mul,
    Div,
    Add,
    Sub,
}

impl Debug for Expr {
    fn fmt(&self, fmt: &mut Formatter<'_>) -> Result<(), Error> {
        use self::Expr::*;
        match *self {
            Number(n) => write!(fmt, "{}", n),
            Op(l, op, r) => write!(fmt, "{} {} {}", l, op, r),
            Error => write!(fmt, "error"),
        }
    }
}

impl Debug for ExprSymbol<'_> {
    fn fmt(&self, fmt: &mut Formatter<'_>) -> Result<(), Error> {
        use self::ExprSymbol::*;
        match *self {
            NumSymbol(n) => write!(fmt, "{}", n),
            Op(l, op, r) => write!(fmt, "{} {} {}", l, op, r),
            Error => write!(fmt, "error"),
        }
    }
}

impl Debug for Opcode {
    fn fmt(&self, fmt: &mut Formatter<'_>) -> Result<(), Error> {
```

```

use self::Opcode::*;

match *self {
    Mul => write!(fmt, "*"),
    Div => write!(fmt, "/"),
    Add => write!(fmt, "+"),
    Sub => write!(fmt, "-"),
}
}

// grammar.lalrpop

use std::str::FromStr;
use crate::ast::{Expr, Opcode};

grammar;

pub Expr: Box<Expr> = { // (1)
    Expr ExprOp Factor => Box::new(Expr::Op(<>)), // (2)
    Factor,
};

ExprOp: Opcode = { // (3)
    "+" => Opcode::Add,
    "-" => Opcode::Sub,
};

Factor: Box<Expr> = {
    Factor FactorOp Term => Box::new(Expr::Op(<>)),
    Term,
};

FactorOp: Opcode = {
    "*" => Opcode::Mul,
    "/" => Opcode::Div,
};

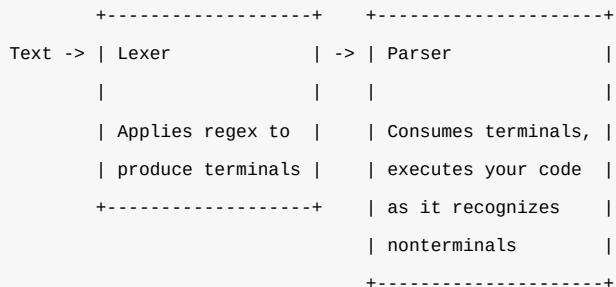
Term: Box<Expr> = {
    Num => Box::new(Expr::Number(<>)),
    "(" <Expr> ")"
};

Num: i32 = {
    r"[0-9]+" => i32::from_str(<>).unwrap()
};

```

Lexer

```
Num: i32 = r"[0-9]+" => i32::from_str(<>).unwrap();  
// ~~~~ ~~~~~ ~~~~~~ ~~~~~~  
// | | | Action code  
// | | Symbol(s) that should match  
// | Return type  
// Name of nonterminal
```



词法分析优先级问题：使用 match 显式指定优先级。

Simple match declarations 简单 match 声明

A match declaration lets you explicitly give the precedence between terminals. In its simplest form, it consists of just ordering regular expressions and string literals into groups, with the higher precedence items coming first. So, for example, we could resolve our conflict above by giving `r"\d{1,3}."` precedence over `r"\w+."`, thus saying that if something can be lexed as a number, we'll do that, and otherwise consider it to be an identifier.

`match` 声明允许您显式指定终端之间的优先级。在最简单的形式中，它只包括将正则表达式和字符串文本排序到组中，优先级较高的项排在最前面。因此，例如，我们可以通过赋予 `r"[0-9]+"` 优先于 `r"w+"` 来解决上面的冲突，从而表示如果某个东西可以被解释为一个数字，我们就会这样做，否则就将其视为标识符。

```
match {
    r"[0-9]+"
} else {
    r"\w+",
    -
}
```

The final `_` indicates that other string literals and regular expressions that appear elsewhere in the grammar (e.g., "(" or "22") should be added into that final level of precedence (without an `_`, it is illegal to use a terminal that does not appear in the match declaration).

结尾的 `_` 表示出现在语法中其他位置的字符串文字和正则表达式（例如，“`(`”或“`22`”）应该添加到最终的优先级中（如果没有 `_`，使用未出现在 `match` 声明中的终结符是非法的）。

```
// fixed literals get precedence over regular expressions
match {
    r"[0-9]+",
    "22"
} else {
    r"\w+",
    -
}
```

使用 `match` 声明为正则表达式命名，这样我们就不必直接在语法中输入它们。

```
match {
    r"[0-9]+",
    "22"
} else {
    r"\w+" => ID, // <-- give a name here
    -
}
```

```
pub Term = {
    Num,
    "(" <Term> ")",
    "22" => "Twenty-two!".to_string(),
    ID => format!("Id({})", <>), // <-- changed this
};
```

Customizing skipping between tokens 自定义在标记之间跳过（用于跳过注释内容）

```
match {
    r"\s*" => { }, // The default whitespace skipping is disabled if an `ignore pattern` is specified
    r"//[^\\n\\r]*[\\n\\r]*" => { }, // Skip `// comments`
    r"/*[^\/*]*\/*+(?:[^\/*][^\/*]*\/*+)*/" => { }, // Skip `/* comments */`
```

Codegen Dialect Overview

[Codegen Dialect Overview](#)

Classification 分类

The multiple code generation-related dialects in MLIR can be roughly organized along two axes: *tensor/buffer* and *payload/structure*.MLIR 中的多个代码生成相关方言可以大致沿着两个轴组织：张量/缓冲区和有效载荷/结构。

A dialect's position on the tensor/buffer axis indicates whether its main data abstraction is a tensor as found in ML frameworks or a memory buffer as expected by conventional lower-level compilers. Tensors are treated as immutable values that are not necessarily associated with memory, that is, operations on tensors usually don't have side effects. Data flow between such operations can be expressed using use-definition chains in traditional static single assignment (SSA) form. This is one of the aspects that makes MLIR a powerful transformation vehicle for ML programs, enabling simple rewriting of tensor operations. On the other hand, buffers are mutable and may be subject to aliasing, i.e. several objects may be pointing to the same underlying memory. Data flow can only be extracted through additional dependency and aliasing analyses. The transition between the tensor abstraction and buffer abstraction is performed by the *bufferization* procedure that progressively associates, and ultimately replaces, tensors with buffers. Several dialects, such as Linalg and Standard, contain operations on both tensors and buffers. Some Linalg operations can even operate on both at the same time.

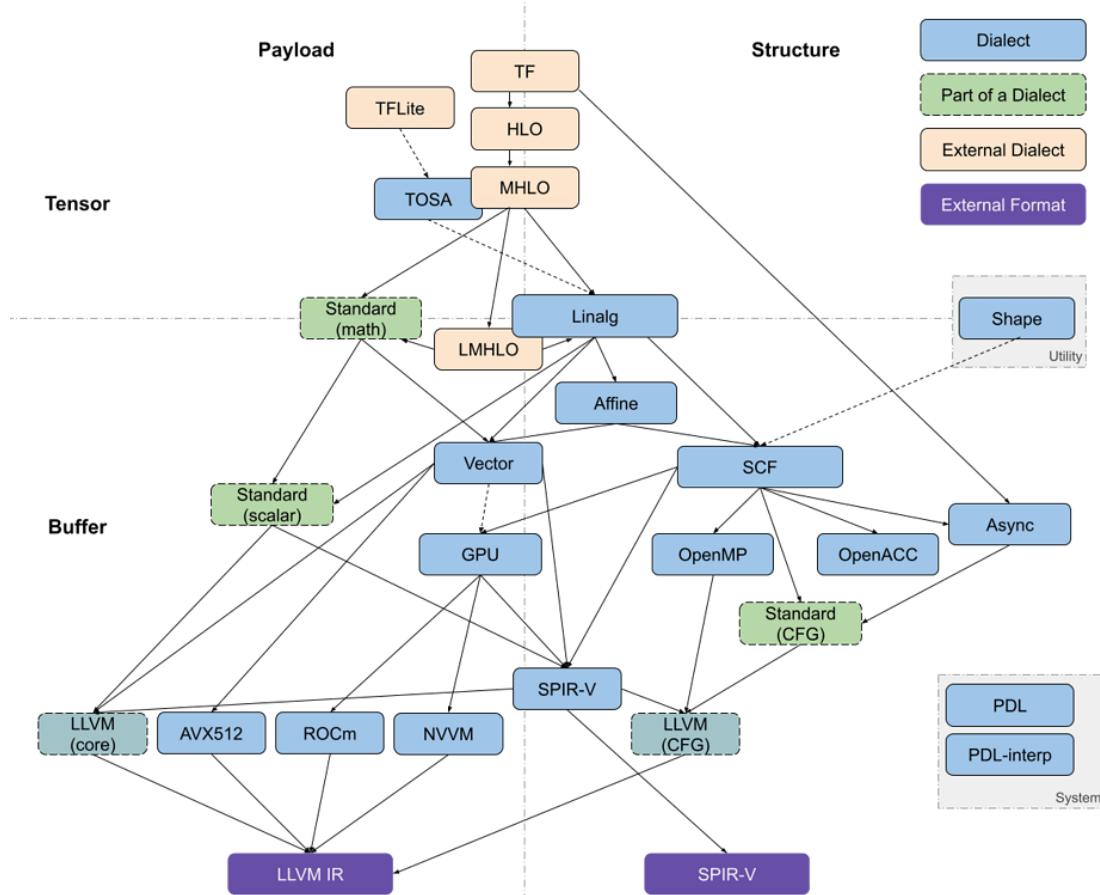
方言在张量/缓冲区轴上的位置表示其主要数据抽象是机器学习框架中找到的张量，还是传统低级编译器期望的内存缓冲区。张量被视为不可变值，不一定与内存相关联，即张量上的操作通常没有副作用。这些操作之间的数据流可以使用传统的静态单赋值（SSA）形式中的使用-定义链来表示。这是使 MLIR 成为机器学习程序强大转换工具的方面之一，它使张量操作的简单重写成为可能。另一方面，缓冲区是可变的，可能受到别名的影响，即多个对象可能指向相同的底层内存。数据流只能通过额外的依赖和别名分析来提取。张量抽象和缓冲区抽象之间的转换是通过缓冲化过程执行的，该过程逐步关联，最终用缓冲区替换张量。一些方言，如 Linalg 和 Standard，包含对张量和缓冲区的操作。一些 Linalg 操作甚至可以同时操作两者。

A dialect's position on the payload/structure axis indicates whether it describes *what* computation should be performed (payload) or *how* it should be performed (structure). For example, most mathematical operations in the Standard dialect specify the computation to be performed, e.g., the arctangent, without further detail. On the other hand, the SCF dialect defines *how* the contained computation is performed, e.g., repeated until some runtime condition is met, without restricting what the condition is and what computations are performed. Similarly, the Async dialect denotes the general execution model applicable at various levels of payload granularity.

方言在负载/结构轴上的位置表示它描述的是应该执行什么计算（负载）还是如何执行（结构）。例如，标准方言中的大多数数学运算指定要执行的计算，例如反正切，而不提供更多细节。另一方面，SCF 方言定义了包含的计算如何执行，例如重复执行直到满足某些运行时条件，而不限制条件是什么以及执行了哪些计算。同样，Async 方言表示适用于负载粒度不同级别的通用执行模型。

This position on this axis is non-binary, especially at higher level of abstraction. Many operations at least partially specify the structure. For example, vector dialect operations imply SIMD execution model. During the compilation process, the instructions of “how” tend to become more detailed and low-level. Simultaneously, lower levels of the abstraction stack tend to separate the structure operations from the payload operations for the sake of transforming only the former while keeping only an abstract understanding of the payload, e.g., the accessed data or the estimated cost.

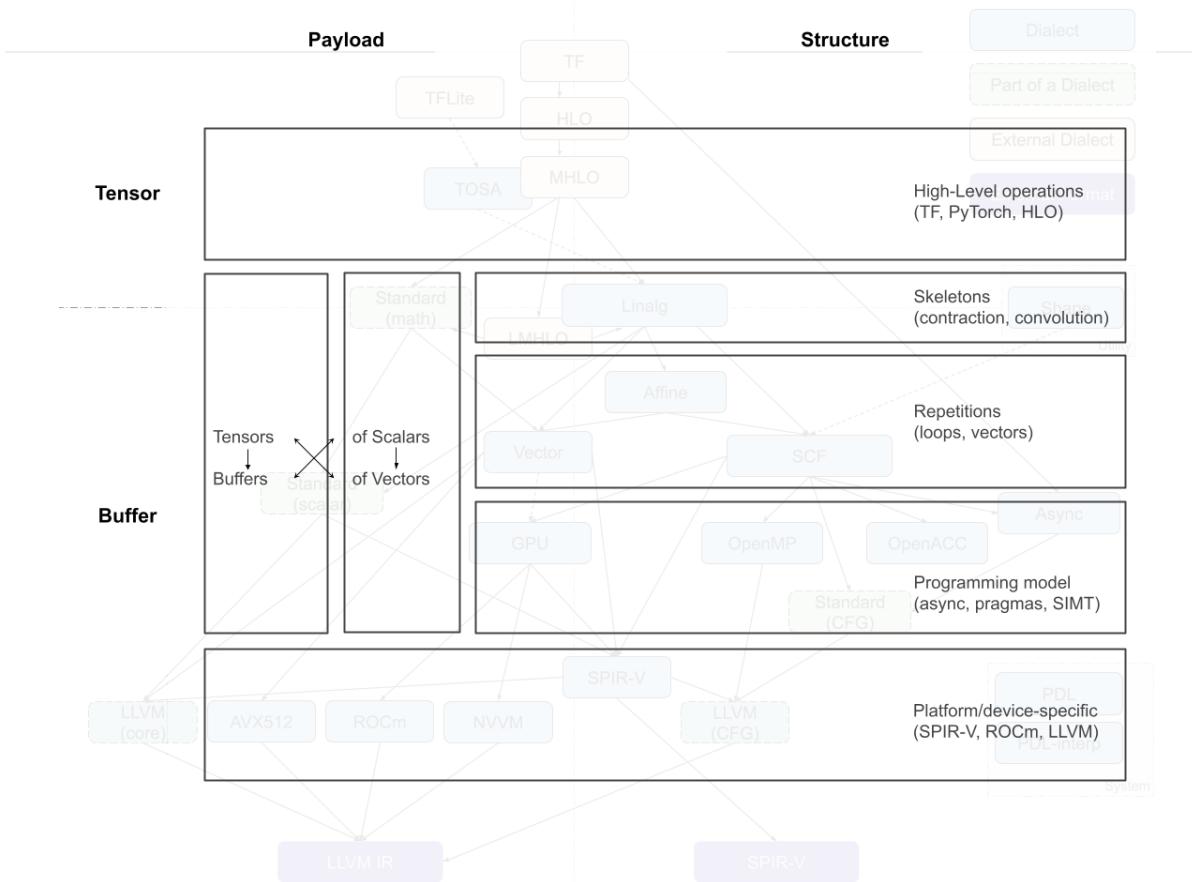
此轴上的位置是非二元的，尤其是在更高层次的抽象中。许多操作至少部分地指定了结构。例如，向量方言操作暗示了 SIMD 执行模型。在编译过程中，关于“如何”的指令往往变得更加详细和底层。同时，抽象堆栈的较低级别倾向于将结构操作与有效负载操作分开，以便仅转换前者，同时只保留对有效负载的抽象理解，例如访问的数据或估计的成本。



Dialects of Interest 感兴趣的语言方言

An MLIR code generation pipeline goes through a sequence of intermediate steps, which are characterized by the most recently introduced dialect. Dialects can be roughly organized into a stack based on the level of abstraction they feature. Converting the representation from a higher-level abstraction to a lower-level abstraction, i.e. lowering, is usually straightforward whereas the inverse process may not be.

MLIR 代码生成管道经过一系列中间步骤，这些步骤由最近引入的方言特征。方言可以根据它们所具有的抽象级别大致组织成堆栈。将表示从高级抽象转换为低级抽象，即降级，通常是直接的，而逆过程可能不是。



Most pipelines enter the in-tree dialect infrastructure through the [Linalg](#) dialect, which features a versatile representation of structured computation on structured data. The dialect is [specifically designed](#) to support various transformations with minimal analysis. Operations in this dialect support both tensor and buffer operands and the bufferization process can happen without changing the operations themselves. Furthermore, Linalg provides “named” operations with specific payload, such as matrix multiplication and convolution, and “generic” operations that only define the structure. Conversions are available between the two forms. The inherent iterative structure of Linalg dialect operations allows them to be converted into vector operations as well as (affine) loops around vector or scalar operations.

大多数管道通过 [Linalg](#) 方言进入树内方言基础设施，该方言具有对结构化数据上结构化计算的灵活表示。该方言专门设计用于支持各种变换，分析最少。该方言中的操作支持张量和缓冲区操作数，缓冲区化过程可以在不改变操作本身的情况下发生。此外，[Linalg](#) 提供了具有特定有效载荷的“命名”操作，例如矩阵乘法和卷积，以及仅定义结构的“通用”操作。两种形式之间可以进行转换。[Linalg](#) 方言操作的内禀迭代结构允许它们被转换为向量操作以及围绕向量或标量操作的（仿射）循环。

The [Async](#) dialect captures a general asynchronous programming model and may appear at different levels: at a higher level where it is used to[organize large chunks of computation across and within devices, and at a lower level where it can wrap sequences of primitive instructions.

异步方言捕获了一般的异步编程模型，可能出现在不同的级别：在较高级别，它用于[组织和跨设备以及设备内部的大块计算，以及在较低级别，它可以包装原始指令的序列]。

The [Vector](#) dialect (Note that the vector *type* belongs to the built-in dialect and can be used outside of the Vector dialect.) is a mid-level abstraction for SIMD and, potentially, SIMT execution models. It leverages MLIR’s multidimensional vector type to target different platforms through dedicated lower-level dialects. An ongoing

work investigates the use of [vector abstraction to target GPU](#) devices (SIMT) through explicit representation of threads.

向量方言（请注意，向量类型属于内置方言，可以在向量方言之外使用。）是 SIMD 和潜在 SIMT 执行模型的中间层抽象。它利用 MLIR 的多维向量类型，通过专门的低级方言针对不同的平台。正在进行的工作正在研究使用向量抽象通过显式表示线程来针对 GPU 设备（SIMT）。

The [Affine](#) dialect is MLIR’s take on [polyhedral compilation](#). It encapsulates the restrictions of the related programming model and defines the corresponding operations, namely control flow structures such as affine loops and conditionals and affine counterparts of memory operations. Its primary goal is to enable polyhedral transformations, such as auto-parallelization, loop fusion and tiling for locality improvement, and loop vectorization in MLIR.

仿射方言是 MLIR 对多面体编译的见解。它封装了相关编程模型的限制并定义了相应的操作，即控制流结构，如仿射循环和条件以及内存操作的仿射对应物。其主要目标是实现多面体变换，如自动并行化、循环融合和为了局部性改进的循环填充，以及在 MLIR 中的循环向量化。

The [SCF](#) (Structured Control Flow) dialect contains the common control flow concepts expressed at a higher level than branches in a control flow graph (CFG), e.g., (parallel) “for” and “while” loops as well as conditionals. This dialect is used to represent, and sometimes transform, the structure of the computation without affecting the payload. It is a common lowering target from Affine and Linalg, which may also be used as an entry point to the MLIR code generation infrastructure from lower-level representations such as C.

SCF（结构化控制流）方言包含在控制流图（CFG）分支以上的高级控制流概念，例如（平行）“for”和“while”循环以及条件语句。此方言用于表示，有时也用于转换，计算结构而不影响有效载荷。它是从 Affine 和 Linalg 的常见降低目标，也可以用作从 C 等底层表示的 MLIR 代码生成基础设施的入口点。

Various programming models, namely [GPU/SIMT](#), [Async](#), [OpenMP](#) and [OpenACC](#) can be obtained from the SCF dialect. Each of these models is represented by a corresponding dialect, the operations in which are rarely subject to further *optimizing* transformations. However, these representations are an opportunity to implement transformations specific to the programming model, e.g., ones currently explored for the Async dialect.

各种编程模型，即 GPU/SIMT、异步、OpenMP 和 OpenACC，可以从 SCF 方言中获取。每个模型都由相应的方言表示，其中的操作很少受到进一步优化转换的影响。然而，这些表示是实现特定于编程模型的转换的机会，例如目前为异步方言探索的转换。

SCF can also be converted to the “standard” CFG representation by replacing structured control flow with branches between blocks. The branch operations are currently contained in the Standard dialect, together with numerous other operations at various abstraction levels. For example, the Standard dialect also contains pointwise operations on tensors and vectors, conversions between buffers and tensors, trigonometric operations on scalars, etc. Therefore, the Standard dialect [is in the process of being split](#) into multiple well-defined dialects.

SCF 也可以通过用块之间的分支替换结构化控制流来转换为“标准”CFG 表示。分支操作目前包含在标准方言中，以及各种抽象级别的许多其他操作。例如，标准方言还包含张量和向量的逐点操作、缓冲区和张量之间的转换、标量的三角运算等。因此，标准方言正在被拆分为多个定义良好的方言。

Ultimately, parts of the Standard dialect (operations on scalars and vectors, and branches) are converted into target-specific dialects that mostly serve as exit points from the MLIR code generation infrastructure. These include the [LLVM](#), [NVVM](#), [ROCDL](#), [AVX](#), [Neon](#), [SVE](#) and [SPIR-V](#) dialects, all of which correspond to an external format, IR or instruction set. These dialects are not subject to transformation except for canonicalization.

最终，标准方言（标量与向量的操作以及分支）被转换为特定目标的方言，这些方言主要作为从 MLIR 代码生成基础设施的出口点。这包括 LLVM、NVVM、ROCDL、AVX、Neon、SVE 和 SPIR-V 方言，所有这些都对应于外部格式、IR 或指令集。这些方言除了规范化外，不受转换的影响。

Finally, the `Shape` dialect is used to describe shapes of data independently of the payload or (mostly) structure. It appears at the entry level of the code generation pipeline and is typically lowered into address arithmetic or canonicalized away.

最后，`Shape` 方言用于独立于有效载荷或（主要是）结构来描述数据形状。它在代码生成管道的入口级别出现，通常被降低为地址算术或规范化。

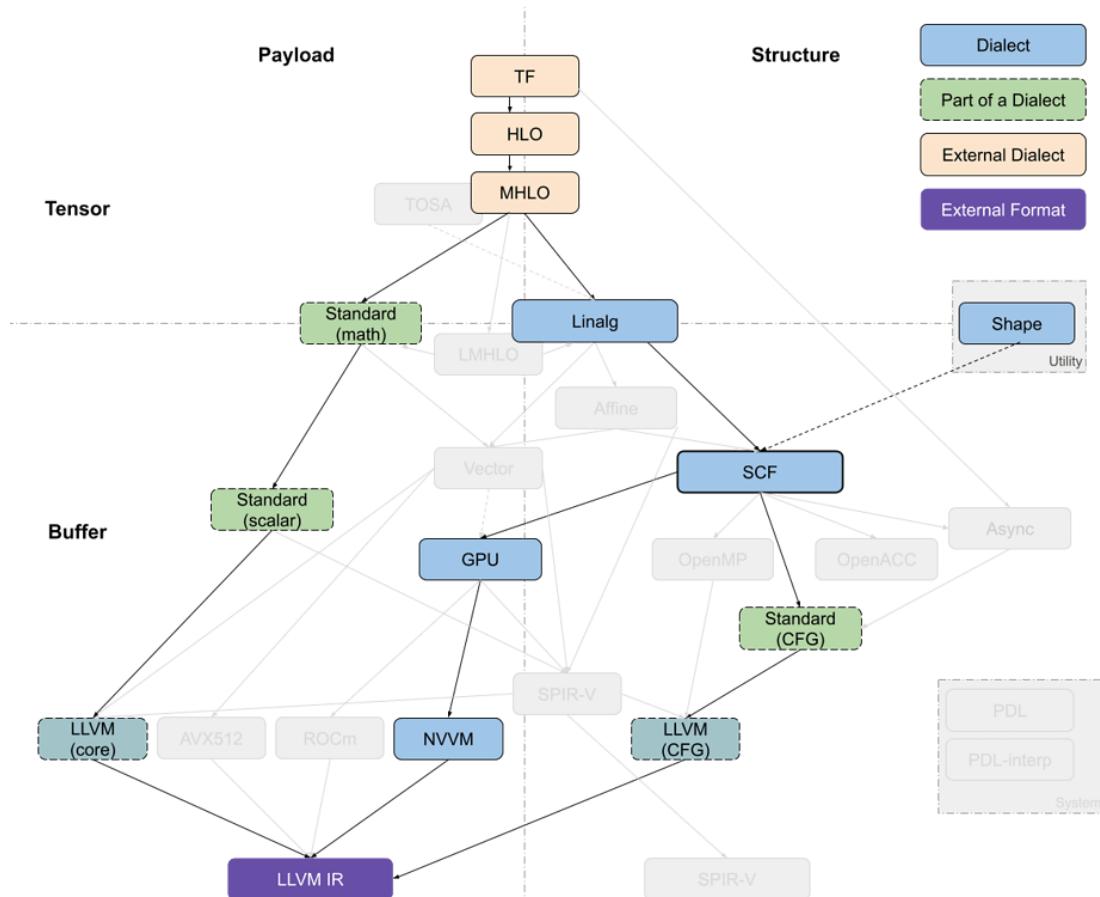
`PDL` (Pattern Description Language) 和 `PDLInterp` 方言是用于下一代模式重写基础设施 MLIR 来指定转换的。因此，它们从未出现在代码生成管道中，但可能是描述其操作所必需的。

`PDL` (模式描述语言) 和 `PDLInterp` 方言被下一代模式重写基础设施 MLIR 用于指定转换。因此，它们从未出现在代码生成管道中，但可能是描述其操作所必需的。

Some Existing Pipelines 一些现有管道

TensorFlow Kernel Generator TensorFlow 内核生成器

codegen 4 Kernel generator pipeline 代码生成器 4 内核生成器管道 1170×954 120 KB



The Tensorflow Kernel Generator project, starting at the TensorFlow (TF) dialect, has recently switched to targeting Linalg-on-tensors from [MHLO](#) (Meta HLO, more suitable for compilation thanks to, e.g., removal of implicit broadcasting, and with support for dynamic shapes; where [HLO](#) is High-Level Optimizer representation, derived from XLA) instead of [LMHLO](#) (Late MHLO, same as MHLO but on buffers rather than tensors) and performs fusion at that level before calling [bufferization](#) on Linalg. Further loop transformations such as tiling happen at the SCF level, which is then converted into target-specific GPU dialect while the payload operations are converted to the LLVM dialect with Standard as intermediary. Now-retired prototypes have experimented with using LMHLO dialect targeting Linalg-on-Buffers and performing all transformations on SCF, where it may be more complex than on tensor abstraction.

Tensorflow 内核生成器项目，从 TensorFlow (TF) 方言开始，最近已切换到针对 Linalg-on-tensors 从 MHLO (元 HLO，由于例如去除隐式广播等，更适合编译，并支持动态形状；其中 HLO 是高级优化表示，源自 XLA) 而不是 LMHLO (延迟 MHLO，与 MHLO 相同，但针对缓冲区而不是张量) 的目标，并在该级别进行融合，然后在调用 Linalg 的缓冲区上进行缓冲化。进一步的循环转换，如 tiling，发生在 SCF 级别，然后将其转换为特定于目标 GPU 方言，而有效负载操作则转换为 LLVM 方言，以标准作为中介。现在已退役的原型已尝试使用针对 Linalg-on-Buffers 的 LMHLO 方言，并在 SCF 上执行所有转换，这可能比在张量抽象上更复杂。

When producing several kernels, TensorFlow-related flows are expected to use the Async dialect to orchestrate computation.

当生成多个内核时，TensorFlow 相关的流程应使用异步方言来编排计算。

Analysis 分析

Crossing Dialects 跨越方言

In hindsight, it appears that the dialects that cross axis boundaries in this classification (GPU, Linalg and Vector) have required the most discussion and iteration before being accepted as part of the core ecosystem. Even now, users of MLIR infrastructure reported that it was challenging to understand the positioning of some dialects. For example, IREE uses parts of the GPU dialect related to on-device execution, but not the parts related to managing data and kernels from the host, which are closer related to the structure than to the payload. Similarly, the discussion on bridging the tensor and memref abstraction with corresponding operations required significant effort to converge.

回顾起来，在这个分类（GPU、Linalg 和 Vector）中跨越轴边界的地方，方言需要最多的讨论和迭代才能被接受为核心生态系统的一部分。即便现在，MLIR 基础设施的用户报告称，理解某些方言的位置仍然具有挑战性。例如，IREE 使用了与设备上执行相关的 GPU 方言的一部分，但没有使用与从主机管理数据和内核相关的部分，这些部分与结构比与有效载荷更相关。同样，关于将张量和 memref 抽象与相应的操作桥接的讨论也耗费了大量精力以达成共识。

This suggests that new dialects, or smaller IR concepts, can be easier to discuss and reach consensus if they are clearly positioned with respect to other dialects and the design space. When it is necessary to cross the gap between abstractions, it may be preferable to discuss it separately and aim for generalization between dialects (e.g., bufferization process).

这表明，如果新方言或较小的 IR 概念在与其他方言和设计空间的关系中明确定位，它们可能更容易讨论和达成共识。当需要跨越抽象之间的差距时，最好单独讨论，并旨在方言之间实现泛化（例如，缓冲过程）。

Linalg at the Center 线性代数在中心

The Linalg dialect is one of the main entry points to the MLIR code generation pipelines. Its most recent evolution makes it operate on both tensors and buffers, making bufferization an intra-dialect transformation. It has sufficiently high-level information about the operations to perform transformations without expensive analyses, especially when operating on tensors as values. Some transformations like fusion of element-wise operations and tiling, and combination thereof to generate imperfectly nested computations capture a sufficient amount of transformations needed to target a broad range of architectures. Furthermore, the concept of named operations enables payload-carrying operations that build on the computational patterns of Linalg.

Linalg 方言是 MLIR 代码生成管道的主要入口之一。其最新的演变使其能够在张量和缓冲区上运行，使缓冲区化为方言内部的转换。它对要执行的操作具有足够的高级信息，可以在不进行昂贵分析的情况下进行转换，尤其是在将张量作为值操作时。一些转换，如逐元素操作的融合和分块，以及将它们组合起来生成不完美嵌套的计算，捕获了针对广泛架构所需的大量转换。此外，命名操作的概念使得基于 Linalg 计算模式的携带有效载荷的操作成为可能。

Being exercised in virtually all compilation pipelines adds maintenance pressure and stability requirements on Linalg as production users start to rely on it. The fact that it works across buffers and tensors, and may capture both the payload and the structure of the computation makes understanding Linalg to some extent a requirement for understanding any MLIR code generation pipeline. While there are benefits in having one well-defined and maintained entry point, extra care must be taken to ensure that Linalg remains composable with other dialects and that transformation algorithms are not over-designed for it.

在几乎所有编译管道中都被使用，给 Linalg 带来了维护压力和稳定性要求，随着生产用户开始依赖它。它能够在缓冲区和张量之间工作，并可能捕获计算的有效载荷和结构，这在一定程度上使得理解 Linalg 成为理解任何 MLIR 代码生成管道的必要条件。虽然有一个定义良好且维护良好的入口点有其好处，但必须格外小心，以确保 Linalg 与其他方言保持可组合性，并且变换算法不要过度设计。

Pipeline Differences to Complementary Transformations 管道差异与互补变换

Several parallel compilation pipelines appear to emerge in the ecosystem. The case of targeting GPUs is particularly illustrative: optimizing transformations, parallelism detection and device mapping decisions can happen in various dialects (Linalg, Affine, SCF), which may end up partially reimplementing each other's functionality; device mapping can derive SIMD threads from explicit loops or from vectors, using the SCF or the Vector dialect. Finally, GPU libraries have support for higher-level operations such as contractions and convolutions, that could be directly targeted from the entry point of the code generation pipeline.

几个并行编译管道似乎在生态系统中出现。针对 GPU 的情况尤其具有说明性：优化转换、并行性检测和设备映射决策可以在各种方言（Linalg、Affine、SCF）中发生，这可能导致部分重新实现彼此的功能；设备映射可以从显式循环或向量中推导出 SIMD 线程，使用 SCF 或 Vector 方言。最后，GPU 库支持更高层次的运算，如收缩和卷积，这些运算可以直接从代码生成管道的入口点进行定位。

It is more important than ever to make sure representations and transformations compose and complement each other to deliver on MLIR's promise to unify the compilation infrastructure, as opposed to building independent parallel flows. It does not necessarily mean immediately reusing all components, but avoid poorly composable patterns as much as possible. The utility of domain- and target-specific compilers is undeniable, but the project may need to invest into cross-cutting representations, using the generic mechanisms of attributes and interfaces. Circling back to targeting GPUs as an example, a device mapping strategy expressed as attributes that can be attached to different operations (e.g., Linalg generics or parallel SCF loops) that can be transformed through interfaces without needing to know the details of a specific operation.

确保表示和变换相互组合和补充，以实现 MLIR 统一编译基础设施的承诺，比以往任何时候都更加重要，而不是构建独立的并行流程。这并不意味着立即重用所有组件，但应尽可能避免不良的可组合模式。领域和目标特定编译器的效用无可否认，但项目可能需要投资于跨领域的表示，使用属性和接口的通用机制。以针对 GPU 为例，一种设备映射策略可以表示为可以附加到不同操作（例如，Linalg 泛型或并行 SCF 循环）的属性，这些操作可以通过接口进行转换，而无需了解特定操作的详细信息。

Build Small Reusable Abstractions 构建小型可重用抽象

One can observe an unsurprising tendency of performing most transformations in the higher levels of the code generation pipeline: the necessary validity information is readily available or easily extractable at these levels without the need for complex analyses. Yet, higher levels of abstraction often have more stringent restrictions of what is representable. When pursuing benefits of such abstractions, it is important to keep in mind the expressivity and the ability to perform at least some transformations at lower levels as means to quickly increase expressivity without reimplementing the top-level abstraction (dynamic shapes in HLO are a good example).

一个人可以观察到在代码生成管道的高层进行大多数转换的令人惊讶的趋势：必要的有效性信息在这些层面上易于获取或提取，无需复杂的分析。然而，抽象层次越高，对可表示内容的限制通常越严格。在追求这种抽象的好处时，重要的是要记住表达性和在较低层次上执行至少一些转换的能力，作为快速提高表达性的手段，而不需要重新实现顶层抽象（HLO 中的动态形状是一个很好的例子）。

Need for More Structure 需要更多结构

Another emerging tendency is larger dialects and progressive lowering performed without leaving the dialect boundaries. Some examples include Linalg operations on tensors that get transformed into using buffers, GPU dialect block reductions that get decomposed into shuffles, and math operations in Standard that can be expanded into smaller operations or approximated using other Standard operations. We are reaching a point where a dialect contains several loosely connected subsets of operations. This has led to the proposal of splitting the Standard dialect into multiple components. However, the splitting may not always be desirable or even feasible: the *same* Linalg operations accept tensors and buffers so it is challenging to have separate TLinalg and BLinalg without lots of duplication. This calls for additional ways of structuring the operations within a dialect in a way that is understandable programmatically.

另一种新兴趋势是更大方言和渐进式降低，而无需跨越方言边界。一些例子包括将张量操作转换为使用缓冲区，将 GPU 方言块减少分解为洗牌，以及可以在标准中扩展为较小操作或使用其他标准操作进行近似的标准数学操作。我们正达到一个点，其中方言包含几个松散连接的操作子集。这导致了将标准方言拆分为多个组件的提议。然而，拆分可能并不总是可取或甚至可行：相同的 Linalg 操作接受张量和缓冲区，因此在没有大量重复的情况下，要分别有 TLinalg 和 BLinalg 是具有挑战性的。这需要以可编程方式理解的方式在方言内对操作进行结构化。

Host/Device Or Program/Kernel As Additional Axis 主机/设备或程序/内核作为附加轴

Finally, uses of MLIR in the scope of larger, mostly ML-related, flows prompt for cleaner separation between the aspects that pertain to the overall organization of computations (e.g., operations that reflect the mapping of a model on a distributed system and interaction with the framework that embeds MLIR, typically on a “host”) and organization of individual computations (e.g., operations that correspond to internals of a “kernel” or another large unit of computation, potentially offloaded to a “device”). For “host” parts, code generation may also be necessary and often requires different transformations than for “kernel” parts.

最终，在更大、主要与机器学习相关的流程中使用 MLIR，促使计算整体组织方面的方面（例如，反映模型在分布式系统上的映射以及与嵌入 MLIR 的框架交互的操作，通常在“主机”上）与单个计算组织（例如，对应“内核”或其他大型计算单元内部的操作，可能卸载到“设备”上）之间的分离更加清晰。对于“主机”部分，代码生成可能也是必要的，并且通常需要与“内核”部分不同的转换。

Specific examples of this separation include the GPU dialect that contains operations for both controlling the execution from host and operations executed on the GPU proper, which may benefit from separation into independent dialects, and the Async dialect that is being used on two levels: organizing the execution of independent “kernels” and parallelizing execution within a “kernel” by targeting LLVM coroutines.

具体示例包括包含控制主机执行和 GPU 上执行操作的 GPU 方言，这可能从分离成独立的方言中受益，以及正在两个级别上使用的 Async 方言：通过针对 LLVM 协程组织独立“内核”的执行和通过针对“内核”内的并行执行。

MLIR 方言: 概述

MLIR 方言是 MLIR 生态系统中的专业子集，定义了特定领域的操作、类型和转换。每个方言都针对特定领域或抽象级别的计算进行表示和优化。这种模块化允许开发人员通过组合和转换这些方言来构建自定义编译管道。

以下是对关键 MLIR 方言的介绍，如 **Math**、**Linalg**、**SCF** 和 **Affine**，它们在高性能和领域特定计算中发挥着关键作用。

1. Math 方言

目的:

Math 方言提供了一组在计算任务中常用的数学操作，如三角函数、指数函数、对数函数和算术操作。它作为高级抽象与低级数值计算之间的桥梁。

特性:

- 包括基本的数学函数，如 `sin`、`cos`、`exp`、`log` 和 `sqrt`。
- 通过抽象常用数学例程简化计算。
- 支持针对数学操作的优化，如常量折叠或基于库的替换。

应用:

- 常用于机器学习、科学计算以及需要大量数值计算的领域。
- 为转换为硬件优化的数学库提供了基础。

2. Linalg 方言

目的:

Linalg 方言旨在表示结构化的线性代数计算，如矩阵乘法、卷积和归约。它提供了与张量和线性代数操作紧密对齐的抽象。

特性:

- 操作包括:
 - `linalg.matmul` : 矩阵-矩阵乘法。
 - `linalg.conv` : 用于图像处理或神经网络的卷积。
 - `linalg.generic` : 用于用户定义线性代数模式的通用循环。
- 组合性:
 - Linalg 操作可以组合和嵌套以形成复杂的计算。
 - 显式建模迭代和内存访问模式。
- 优化:
 - 高级转换（如分块、融合）。
 - 与低级方言（如 Affine）集成，以进行详细的硬件映射。

应用:

-
- 机器学习框架中的张量操作。
 - 涉及密集和稀疏线性代数的高性能计算工作负载。
-

3. SCF 方言（结构化控制流）

目的:

SCF 方言提供了基本的控制流结构，如循环和条件语句。这些结构是通用的，可以作为高级控制流和迭代建模的构建块。

特性:

- 操作:
 - `scf.for` : 表示带有起始值、结束值和步长的标准 for 循环。
 - `scf.if` : 用于控制流的条件分支。
 - `scf.parallel` : 用于多线程或 SIMD 风格执行的并行循环。
- 高级控制流抽象简化了后续的优化和转换。

应用:

- 用作将高级循环转换为低级结构的中间表示。
 - 作为集成循环优化（如循环展开或向量化）的跳板。
-

4. Affine 方言

目的:

Affine 方言提供了用于建模静态控制流、内存访问模式和循环嵌套的精确且可分析的结构。它特别适用于优化性能关键的计算。

特性:

- **Affine 表达式:**
 - 使用带有整数系数的线性表达式来表示循环边界和内存访问。
 - 示例: `affine.for %i = 0 to 100 step 2`。
- **Affine 映射:**
 - 抽象地表示索引和内存地址之间的映射。
 - 适用于循环分块或数组重索引等转换。
- **优化:**
 - 支持循环融合、交换和分块等转换。
 - 允许依赖性分析以进行并行执行。

应用:

-
- 需要显式控制循环和内存的高性能计算。
-

- 作为降低到硬件特定表示（如 LLVM IR）的基础层。
-

方言之间的交互

MLIR 方言设计为无缝互操作。例如：

- **Linalg** 方言中的高级操作可以降低为 **SCF** 方言中的循环，或降低为 **Affine** 方言中的静态循环以进行进一步优化。
 - **Math** 方言中的数学操作可以嵌入到 Linalg 或 Affine 操作中进行数值计算。
 - SCF 和 Affine 方言提供了互补的控制流抽象：SCF 用于通用循环，Affine 用于静态、可分析的模式。
-

使用 MLIR 方言的优势

1. **模块化**: 开发人员可以选择或定义适合其领域的方言，促进重用和专业化。
2. **可扩展性**: 可以在不改变核心基础设施的情况下引入新方言。
3. **优化管道**: 方言支持从高级抽象到低级硬件特定指令的分阶段降低和优化。
4. **领域特定效率**: 预定义的操作和模式满足机器学习和科学计算等特定领域的需求。

MLIR 的方言方法为开发优化的、可移植且可维护的编译器基础设施提供了强大的框架。

Linalg

Linalg is designed to solve the High-level Hierarchical Optimization (HHO box) in MLIR and to interoperate nicely within a *Mixture Of Expert Compilers* environment (i.e. the *CGSel* box). Linalg 旨在解决 MLIR 中的高级分层优化（HHO 框）问题，并在混合专家编译器环境中良好地交互（即 CGSel 框）。

Set of Key Transformations—一组关键变换

The following key transformations have been central to driving the design of Linalg. They are all implemented in terms of the properties of the `linalg.generic` OpInterface and avoid the pitfall of relying on hardcoded one-off op knowledge. 以下关键转换是推动 Linalg 设计的关键。它们都基于 `linalg.generic` OpInterface 的性质实现，避免了依赖于硬编码的单次操作知识的陷阱。

The textual form description of these transformations is left for future work. Still, it is useful to list the key transformations that are performed on the Linalg IR and that have influenced its design: 这些变换的文本形式描述留待以后的工作。尽管如此，列出在 Linalg IR 上执行的关键变换以及对其设计产生影响的变换是有用的：

1. Progressive Buffer Allocation. 演进式缓冲区分配。
2. Parametric Tiling. 参数铺砖。
3. Promotion to Temporary Buffer in Fast Memory. 提升至快速内存中的临时缓冲区。
4. Tiled Producer-Consumer Fusion with Parametric Tile-And-Fuse. 基于参数化分块融合的镶嵌式生产者-消费者融合
5. Map to Parallel and Reduction Loops and Hardware. 并行和归约循环及硬件的映射
6. Vectorization: Rewrite in Vector Form. 向量化：以向量形式重写。
7. Lower to Loops (Affine, Generic, and Parallel). 下到循环（仿射、通用和并行）。
8. Lower to Library Calls or Special Instructions, Intrinsics or ISA. 下至库函数调用或特殊指令，内建函数或指令集架构。
9. Partially Lower to Iterations Over a Finer-Grained Linalg Op. 部分降低到更细粒度的 Linalg 操作迭代。

MLIR Language Reference

text text text

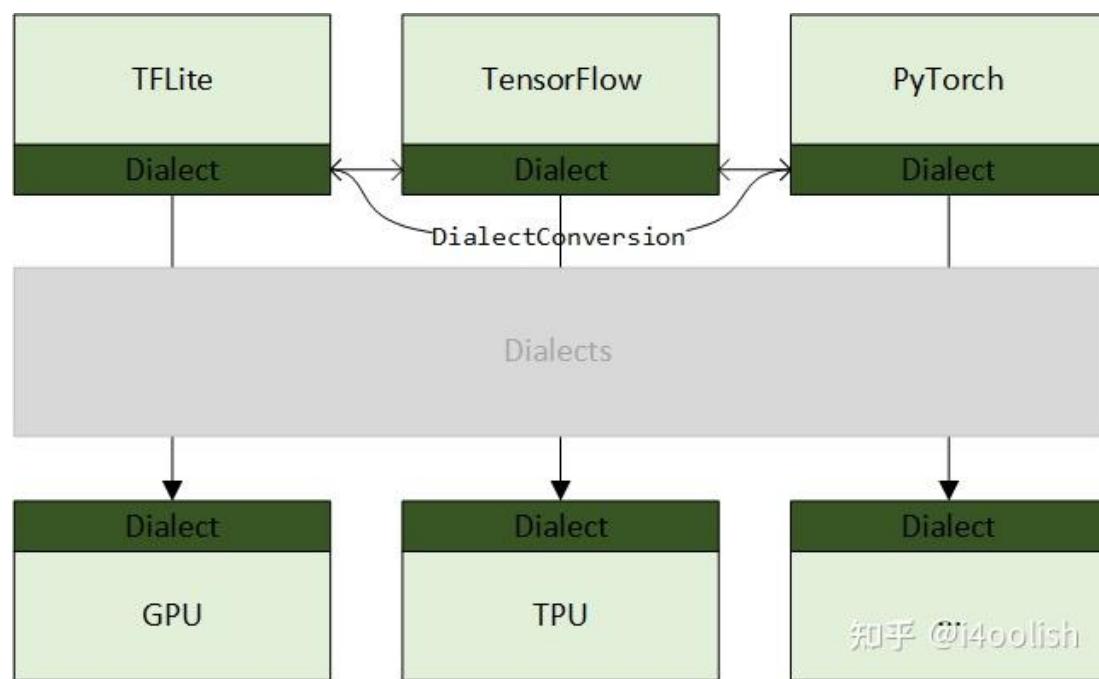
MLIR 基本上是基于一种类图的数据结构，这种结构由称为操作 (Operations) 的节点和称为值 (Values) 的边组成。每个值都是由一个操作或块参数 (Block Argument) 产生的，并且具有由类型系统定义的值类型。操作包含在块 (Blocks) 中，块包含在区域 (Regions) 中。操作在其所属的块中是有顺序的，块在其所属的区域中也是有顺序的，尽管这种顺序在某些类型的区域中可能具有或不具有语义意义。操作还可以包含区域，从而能够表示层次结构。

操作可以表示许多不同的概念，从高层概念（如函数定义、函数调用、缓冲区分配、缓冲区的视图或切片以及进程创建）到低层概念（如与目标无关的算术、与目标相关的指令、配置寄存器和逻辑门）。这些不同的概念由 MLIR 中的不同操作表示，并且 MLIR 中可用的操作集可以任意扩展。

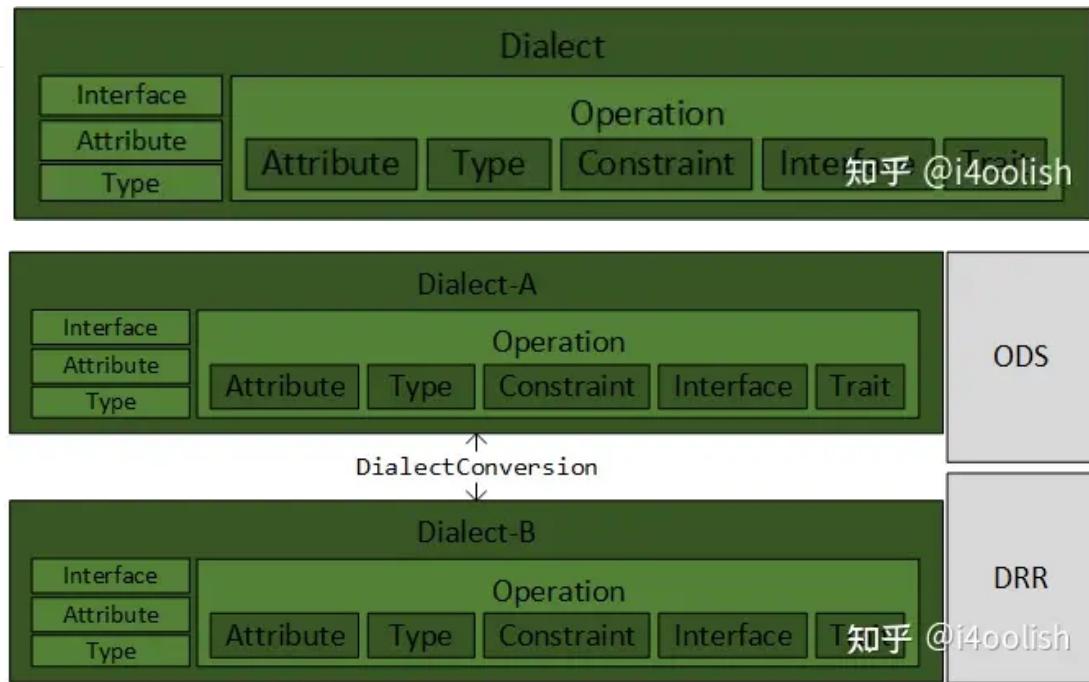
MLIR 还提供了一个可扩展的框架，用于对操作进行转换，使用了编译器 Passes 的常见概念。对任意操作集启用任意转换集会带来显著的扩展挑战，因为每个转换都必须可能考虑到任何操作的语义。MLIR 通过允许操作语义使用特性 (Traits) 和接口 (Interfaces) 进行抽象描述，解决了这一复杂性，从而使转换能够更通用地操作这些操作。特性通常描述了对有效中间表示 (IR) 的验证约束，从而能够捕获并检查复杂的不变量。（参见 Op vs Operation）

MLIR 的一个明显应用是表示基于静态单赋值形式 (SSA) 的中间表示 (IR)，如 LLVM 核心 IR，使用适当的操作类型来定义模块、函数、分支、内存分配和验证约束，以确保 SSA 主导性 (Dominance) 属性。

MLIR 包含了一组定义了这些结构的方言 (Dialects)。然而，MLIR 的设计足够通用，也可以表示其他类似编译器的数据结构，例如语言前端中的抽象语法树、目标特定后端中生成的指令或高级综合工具中的电路。



text



关于MLIR基本模块学习过程如下:

1. Dialect, Attribute, Type, Operation; 想象如果自己去实现, 该怎么设计类;
2. DialectConversion; 想象在自己实现的前四个模块上, 如何实现DialectConversion;
3. Interface, Constraint, Trait; 同样, 想象自己会怎么增加这些功能;
4. Transformation, Concalization;
5. Region, Block: 基于1. 设计的Operation, 以及4. 增加的Transformation, 想象如何对Operation进行抽象, 提取出Region和Block的概念;
6. Pass;
7. 最后才是ODS和DRR。

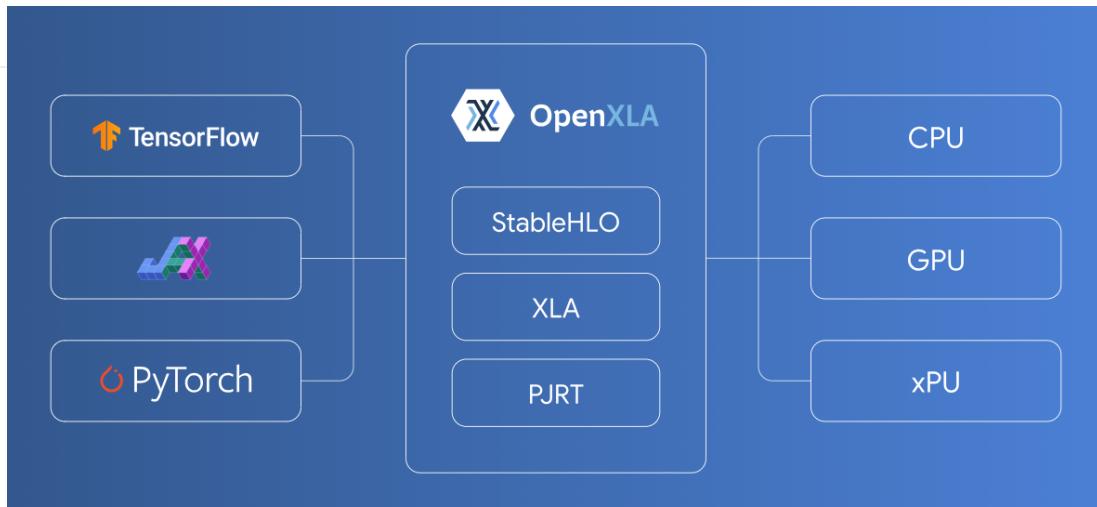
[MLIR中Dialects分类及关联](#)

OpenXLA

[openxla](#)

An open ecosystem of performant, portable, and extensible machine learning (ML) infrastructure components that simplify ML development by defragmenting the tools between frontend frameworks and hardware backends. Built by industry leaders in AI modeling, software, and hardware.

一个由高性能、可移植和可扩展的机器学习 (ML) 基础设施组件组成的开放生态系统，通过对前端框架和硬件后端之间的工具进行碎片整理来简化 ML 开发。由 AI 建模、软件和硬件领域的行业领导者构建。



XLA

XLA (Accelerated Linear Algebra) is an open source compiler for machine learning. The XLA compiler takes models from popular frameworks such as PyTorch, TensorFlow, and JAX, and optimizes the models for high-performance execution across different hardware platforms including GPUs, CPUs, and ML accelerators.

XLA（加速线性代数）是一个用于机器学习的开源编译器。XLA 编译器采用来自 PyTorch、TensorFlow 和 JAX 等流行框架的模型，并优化模型，以便在不同的硬件平台（包括 GPU、CPU 和 ML 加速器）上实现高性能执行。

Key benefits 主要优点

- **Build anywhere** : XLA is already integrated into leading ML frameworks such as TensorFlow, PyTorch, and JAX.
 - 随处构建：XLA 已集成到领先的 ML 框架中，例如 TensorFlow、PyTorch 和 JAX。
- **Run anywhere** : It supports various backends including GPUs, CPUs, and ML accelerators, and includes a pluggable infrastructure to add support for more.
 - 随处运行：它支持各种后端，包括 GPU、CPU 和 ML 加速器，并包含可插拔基础设施以增加对更多后端的支持。
- **Maximize and scale performance** : It optimizes a model's performance with production-tested optimization passes and automated partitioning for model parallelism.
 - 最大化和扩展性能：它通过经过生产测试的优化过程和模型并行性的自动分区来优化模型的性能。
- **Eliminate complexity** : It leverages the power of [MLIR](#) to bring the best capabilities into a single compiler toolchain, so you don't have to manage a range of domain-specific compilers.
 - 消除复杂性：它利用了 MLIR 将最佳功能引入单个编译器工具链，因此您无需管理一系列特定于域的编译器。
- **Future ready** : As an open source project, built through a collaboration of leading ML hardware and software vendors, XLA is designed to operate at the cutting-edge of the ML industry.
 - 面向未来：作为一个开源项目，通过领先的 ML 硬件和软件供应商的合作构建，XLA 旨在在 ML 行业的前沿运行。

StableHLO

StableHLO is an operation set for high-level operations (HLO) in machine learning (ML) models. Essentially, it's a portability layer between different ML frameworks and ML compilers: ML frameworks that produce StableHLO programs are compatible with ML compilers that consume StableHLO programs.

StableHLO 是机器学习 (ML) 模型中高级操作 (HLO) 的操作集。本质上，它是不同 ML 框架和 ML 编译器之间的可移植层：生成 StableHLO 程序的 ML 框架与使用 StableHLO 程序的 ML 编译器兼容。



Shardy

Shardy is an MLIR-based tensor partitioning system for all dialects. Built from the collaboration of both the GSPMD and PartIR teams, it incorporates the best of both systems, and the shared experience of both teams and users.

Shardy 是一个基于 MLIR 的张量划分系统，适用于所有方言。它由 GSPMD 和 PartIR 团队合作构建，融合了两个系统的优点以及团队和用户的共享体验。

PJRT

PJRT is a hardware and framework independent interface for ML compilers and runtimes. It is currently included with the XLA distribution. See the XLA GitHub and documentation for more information on how to use and integrate PJRT.

PJRT 是用于 ML 编译器和运行时的独立于硬件和框架的接口。目前它包含在 XLA 发行版中。有关如何使用和集成 PJRT 的更多信息，请参阅 XLA GitHub 和文档。

TF(TensorFlow) Dialect

This dialect maps to TensorFlow operations. 该方言映射到 TensorFlow 操作。

Invariants: 不变量:

- All values are of Tensor type (in particular, scalars are represented using zero-dimensional tensors);
- 所有值都是 Tensor 类型（特别是标量使用零维张量表示）；

TODO: Make invariants more structured so that we can reference them in ops.TODO:

使不变量更加结构化，以便我们可以在操作中引用它们。

PIR 基本概念和开发¶

在 3.0 版本下，飞桨研发了基于 MLIR 范式的新一代中间表示技术，即 Paddle IR（下简称 PIR）。这项技术对底层的核心概念如 Operation、Attribute 等进行了系统性的抽象，为开发者提供了灵活的基础组件；同时，通过引入 Dialect 这一概念，飞桨能够全面、分层次管理框架各模块对中间表示的需求，并支持开发者根据需求定制化扩展 Dialect，显著提升了框架的扩展性。PIR 遵循 SSA（即 Static Single Assignment）原则，统一了顶层结构，实现“算子顺序性”和“计算图语义”的兼容表示。此外，PIR 还提供了更加简洁、低成本的 Pass 开发体系，并内置了一系列丰富且功能完备的 Pass 优化策略，为大模型的极致性能优化提供了强有力支撑。

在深度学习框架 IR 概念中，「顺序性」和「图语义」是两个非常高频常用的概念。旧的中间表示体系由「顺序性」ProgramDesc 和「图语义」Graph 两个核心类共同承载。用户在静态图 API 或者动转静模块下，产生的中间表示是 Op-by-Op 的 Program，如果要应用更高层面的优化策略（比如算子融合、inplace 策略、剪枝等），框架会将由 Program 构造出 Graph，其由数据节点、算子节点和彼此关联的边构成。

在新的 Paddle IR 中，飞桨在底层抽象了一套高度可扩展的基础组件，包括 Type、Attribute、Op、Trait 和 Interface，并引入了 Dialect 的概念，支持开发者灵活扩展、自由定制，提供了完备鲁邦的语义表达能力；在模型表示层，通过多 Dialect 模块化管理，统一多端表示，实现了训推一体的全架构统一表示，无缝衔接组合算子和编译器，支持自动优化和多硬件适配；在图变换层，通过统一底层模块，简化基础概念，向用户提供了低成本开发、易用高性能、丰富可插拔的 Pass 优化机制。飞桨的新一代的 IR 表示坚持 SSA（静态单赋值）原则，模型等价于一个有向无环图。并以 Value、Operation 对计算图进行抽象，Operation 为节点，Value 为边。

- Operation 表示计算图中的节点：一个 Operation 表示一个算子，它里面包含了零个或多个 Region；Region 表示一个闭包，它里面包含了零个或多个 Block；Block 表示一个符合 SSA 的基本块，里面包含了零个或多个 Operation；三者循环嵌套，可以实现任意复杂的语法结构
- Value 表示计算图中的有向边：用来将两个 Operaton 关联起来，描述了程序中的 UD 链（即 Use-Define 链）；OpResult 表示定义端，定义了一个 Value，OpOperand 表示使用端，描述了对一个 Value 的使用。

二、设计初衷¶

计算图中间表示（Intermediate Representation，即 IR）是深度学习框架性能优化、推理部署、编译器等方向的重要基石。近些年来，越来越多的框架和研究者将编译器技术引入到深度学习的神经网络模型优化中，并在此基础上借助编译器的理念、技术和工具对神经网络进行自动优化和代码生成。飞桨历史上在架构层面并存着多套不同的中间表示体系，其表达能力各不相同、Pass 开发维护成本较高，代码复用性较差，缺乏统一规范，存在严重的框架稳定性问题。

Kimi

这两个C文件（`passthrough.h.txt` 和 `passthrough.c.txt`）构成了一个名为 `cudawrap` 的库，它的作用是提供一个封装层，用于加载和访问 CUDA 相关的库（如 CUDA Runtime、cuDNN 和 cuBLAS）中的函数。这个封装层允许程序在运行时动态地加载这些库，并提供了一种机制来调用这些库中的函数。以下是对这两个文件的详细分析：

`passthrough.h` 文件

这个头文件定义了 `cudawrap` 库的数据结构和宏，用于封装对 CUDA 相关库的调用。

1. 数据结构定义:

- `entry_t`：一个结构体，包含一个函数指针 `fn_ptr` 和一个函数名 `name`，用于存储库中函数的地址和名称。

2. 宏定义:

- `ENTER` 和 `LEAVE`：用于调试目的，记录函数的进入和离开时间，以及计算函数执行时间。
- `TIME_GEMM_START` 和 `TIME_GEMM_STOP`：用于性能分析，记录和计算 GEMM（General Matrix Multiply）操作的开始和结束时间。
- `CUDART_ENTRY_ENUM`、`CUDNN_ENTRY_ENUM` 和 `CUBLAS_ENTRY_ENUM`：宏，用于生成枚举值，代表不同的库函数。
- `CUDART_FIND_ENTRY`、`CUDNN_FIND_ENTRY` 和 `CUBLAS_FIND_ENTRY`：宏，用于查找特定函数的入口。
- `REAL_CUDA_RT_API_CALL`、`REAL_CUDNN_CALL` 和 `REAL_CUBLAS_CALL`：宏，用于实际调用库中的函数。

3. 枚举定义:

- `cudart_entry_enum_t`、`cudnn_entry_enum_t` 和 `cublas_entry_enum_t`：枚举类型，列出了所有需要封装的 CUDA Runtime、cuDNN 和 cuBLAS 函数。

`passthrough.c` 文件

这个源文件实现了 `passthrough.h` 中定义的数据结构和函数。

1. 全局变量初始化:

- `_cudart_initied_`、`_cudnn_initied_` 和 `_cublas_initied_`：用于标记对应的库是否已经初始化。

2. 库函数入口数组:

- `cudart_lib_entry`、`cudnn_lib_entry` 和 `cublas_lib_entry`：数组，存储了对应库的所有函数入口。

3. 加载库函数:

- `load_cudart_lib`、`load_cudnn_lib` 和 `load_cublas_lib`：函数，用于动态加载 CUDA Runtime、cuDNN 和 cuBLAS 库，并查找所需的函数地址。

4. 初始化检查函数:

- `cudart_lib_initied`、`cudnn_lib_initied` 和 `cublas_lib_initied`：函数，用于检查对应的库是否已经初始化。

总结

`cudawrap` 库的主要作用是提供一个中间层，使得应用程序可以更加灵活地使用 CUDA 相关的库。通过动态加载和查找函数，它允许应用程序在不同的环境（如不同的 CUDA 版本或配置）下运行，而无需重新编译。此外，它还提供了性能分析和调试支持，帮助开发者优化和调试 CUDA 相关的代码。

使用方法

要使用 `cudawrap` 库，你需要按照以下步骤操作：

1. **包含头文件**: 在你的 C 或 C++ 源文件中包含 `passthrough.h` 头文件。
2. **链接库文件**: 在编译你的程序时，确保链接了 `cudawrap` 库以及它依赖的 CUDA 相关库（如 `libcudart.so, libcudnn.so, libcublas.so`）。
3. **初始化库**: 在程序中调用 `load_cudart_lib()`, `load_cudnn_lib()`, 和 `load_cublas_lib()` 函数来加载对应的 CUDA 库，并初始化 `cudawrap` 库。
4. **调用封装函数**: 使用 `cudawrap` 提供的宏来调用 CUDA 库中的函数。

下面是一个简单的例子，展示了如何使用 `cudawrap` 库来调用 CUDA Runtime API 中的 `cudaMalloc` 和 `cudaFree` 函数：

```

#include "passthrough.h"

int main() {
    // 初始化 CUDA 运行时库
    if (load_cudart_lib() != 0) {
        fprintf(stderr, "Failed to load CUDA runtime library.\n");
        return -1;
    }

    // 分配 CUDA 设备内存
    void *devPtr;
    size_t size = 1024 * 1024; // 分配 1MB 内存
    REAL_CUDA_RT_API_CALL(cudart_lib_entry, , cudaMalloc, (void**)&devPtr, size);

    if (devPtr == NULL) {
        fprintf(stderr, "Failed to allocate device memory.\n");
        // 这里应该进行清理工作
        return -1;
    }

    // ... 在这里进行其他 CUDA 操作 ...

    // 释放 CUDA 设备内存
    REAL_CUDA_RT_API_CALL(cudart_lib_entry, , cudaFree, (devPtr));

    // 清理工作，关闭程序
    return 0;
}

```

在这个例子中，我们首先调用 `load_cudart_lib()` 来加载 CUDA 运行时库并初始化 `cudawrap` 库。然后，我们使用 `REAL_CUDA_RT_API_CALL` 宏来封装对 `cudaMalloc` 和 `cudaFree` 函数的调用。这个宏会查找对应的函数入口，并执行函数调用。

请注意，这个例子假设 `cudawrap` 库已经被正确编译并链接到你的程序中。在实际使用中，你可能需要根据你的环境和编译器来调整编译和链接选项。此外，错误处理和资源管理（如内存泄漏）也是实际编程中需要考虑的重要方面。

Hijack CUBLAS

`hijack_cublas` 是一个库，它提供了对 cuBLAS 库函数的封装和可能的替代实现。这个库的目的是在不同的环境下，如模拟 CUDA 环境或在不支持 CUDA 的系统上，提供 cuBLAS 函数的接口。以下是对 `hijack_cublas` 的进一步分析：

hijack_cublas.h 文件

这个头文件定义了 `hijack_cublas` 库的主要结构和枚举类型，以及函数原型。

1. 枚举类型：

- `cublasStatus_t` : 定义了 cuBLAS API 可能返回的状态码。
- `cublasOperation_t` : 定义了矩阵操作类型，如 `CUBLAS_OP_N` (无转置)、`CUBLAS_OP_T` (转置) 和 `CUBLAS_OP_C` (共轭转置)。

2. 结构体定义:

- `cublasContext` : 定义了一个结构体，包含一个初始化标志和一个 CUDA 流。
- `cublasHandle_t` : `cublasContext` 结构体的指针类型，用于作为 cuBLAS 库函数的上下文句柄。

3. 外部变量声明:

- `cublas_entry[]` : 声明了一个 `entry_t` 类型的数组，用于存储 cuBLAS 库函数的入口。

4. 函数类型定义:

- `cublas_sym_t` : 定义了一个函数指针类型，指向一个返回 `cublasStatus_t` 的函数。

`hijack_cublas.c` 文件

这个源文件实现了 `hijack_cublas.h` 中声明的函数，提供了 cuBLAS 函数的具体实现。

1. cuBLAS 函数实现:

- `cublasCreate_v2` : 创建一个 cuBLAS 上下文（句柄）。
- `cublasDestroy_v2` : 销毁一个 cuBLAS 上下文（句柄）。
- `cublasSetStream_v2` : 设置 cuBLAS 上下文的 CUDA 流。

2. 异步 GEMM 实现:

- `cublasSgemm_async_fn` : 一个异步执行 SGEMM 操作的函数，它使用 OpenMP 进行多线程计算。

3. SGEMM 函数实现:

- `cublasSgemm_v2` : 执行矩阵乘法操作 (SGEMM)。这个函数根据编译时定义的条件 (如 `USE_MoCCUDA` 和 `USE_GCD`) 来决定是调用真实的 cuBLAS 函数还是模拟的实现。

4. 条件编译:

- 根据定义的宏 (如 `USE_MoCCUDA`、`USE_GCD` 等)，`hijack_cublas` 可以选择使用不同的实现策略。例如，如果定义了 `USE_MoCCUDA`，则会使用模拟的 cuBLAS 实现，而不是调用真实的 CUDA 函数。

5. 调试和性能分析:

- 文件中包含了 `ENTER` 和 `LEAVE` 宏，用于调试和性能分析，记录函数的进入和离开时间。

总结

`hijack_cublas` 库的作用是在不具备完整 CUDA 支持的环境下，提供 cuBLAS 库的接口和可能的模拟实现。它允许开发者在编写代码时，不需要关心后端是真实的 CUDA 环境还是模拟环境，从而提高了代码的可移植性。此外，它还提供了异步执行和多线程支持，以优化性能。通过条件编译，`hijack_cublas` 可以灵活地适应不同的使用场景和需求。

ChatGPT

The provided **C** and **header** files implement a dynamic loader for CUDA libraries (**libcudart**, **cudnn**, and **cublas**) and a wrapper for their functionality. Here's a detailed breakdown:

Purpose

The main role of these files is to **dynamically load CUDA libraries at runtime** instead of linking them statically. This approach:

1. Enables the use of CUDA functionalities without requiring the application to depend on specific versions of CUDA libraries during compilation.
 2. Provides a mechanism for runtime flexibility, allowing switching between different library versions.
 3. Facilitates debugging, logging, and performance tracking of library function calls.
-

passthrough.c Analysis

Key Features

1. Dynamic Library Loading:

- Functions like `dlopen()` and `dlsym()` dynamically load the CUDA libraries and resolve symbols (function pointers).
- If the library or a specific function is not found, it emits errors and gracefully exits.

2. Function Resolution:

- Arrays (`cudart_lib_entry`, `cudnn_lib_entry`, and `cublas_lib_entry`) store the function names and their resolved function pointers (`fn_ptr`).
- Functions like `load_cudart_lib()`, `load_cudnn_lib()`, and `load_cublas_lib()` populate these arrays.

3. Initialization Flags:

- Static variables like `_cudart_initied_`, `_cudnn_initied_`, and `_cublas_initied_` prevent redundant re-initialization.

4. Error Handling:

- Robust error checking for missing libraries and functions.

CUDA Libraries

- `libcudart` : Basic CUDA Runtime functionalities (e.g., `cudaMalloc`, `cudaMemcpy`).
 - `cudnn` : Deep Neural Network primitives.
 - `cublas` : Basic Linear Algebra operations.
-

passthrough.h Analysis

Structure Definitions

- `entry_t` : A structure for maintaining a function pointer (`fn_ptr`) and the corresponding function name (`name`).

Wrapper Macros

1. Function Wrapping:

- `REAL_CUDA_RT_API_CALL`, `REAL_CUDNN_CALL`, and `REAL_CUBLAS_CALL` resolve symbols and execute them.
- Example: A call to `cudaMalloc()` gets resolved dynamically using the corresponding function pointer.

2. Logging and Debugging:

- `ENTER` and `LEAVE` macros log function entry and exit with execution timings, but only if compiled with `DEBUG` or `FUNC_TIMINGS` flags.
- `TIME_GEMM_START` and `TIME_GEMM_STOP` specifically measure performance of GEMM (General Matrix Multiplication) operations, crucial for BLAS and deep learning workloads.

Enumerations

- Enumerates all supported library functions (`CUDART_ENTRY_ENUM`, `CUDNN_ENTRY_ENUM`, and `CUBLAS_ENTRY_ENUM`) for easy indexing in the arrays.

Initialization Functions

- `load_cudart_lib()`, `load_cudnn_lib()`, and `load_cublas_lib()` are declared for library loading at runtime.

Typical Workflow

1. Initialization:

- A CUDA library (e.g., `libcudart`) is dynamically loaded when a function is called for the first time.
- Initialization flags ensure one-time setup.

2. Function Execution:

- For example, a wrapped call to `cudaMalloc()` would:
 - Resolve the function pointer via `dlsym()`.
 - Execute the function through the resolved pointer.

3. Debugging and Logging (Optional):

- Execution times and other statistics are logged if debugging macros are enabled.

Role in CUDAWRAP

These files act as the **core implementation for a dynamic wrapper** around CUDA libraries, suitable for environments where:

1. **Runtime Flexibility:** The application can dynamically switch between different versions of CUDA libraries by setting paths via environment variables (`SWITCHCUDART`, `SWITCHCUDNN`, `SWITCHCUBLAS`).
2. **Debugging Support:** Tracks performance and function calls without recompiling the CUDA libraries.
3. **Ease of Integration:** Reduces compilation dependencies, making applications portable across systems with varying CUDA versions.

Would you like to explore specific sections or use cases in more detail?

How to use

To use the **CUDAWRAP** library defined in the provided files, follow these steps. Here's a detailed explanation with an example of how to integrate it into a project and use its features.

1. Setting Up the Environment

The **CUDAWRAP** library uses environment variables to dynamically load CUDA libraries. You need to set these variables to point to the correct library paths. For example:

```
export SWITCHCUDART=/usr/local/cuda/lib64/libcudart.so
export SWITCHCUDNN=/usr/local/cuda/lib64/libcudnn.so
export SWITCHCUBLAS=/usr/local/cuda/lib64/libcublas.so
```

If the environment variables are not set, the library defaults to paths like `PATHTOCUDA_ROOT/lib64/libcudart.so`.

2. Compilation

You need to compile the `passthrough.c` and link it into your project. Use the following steps:

1. Compile the `passthrough.c` :

```
gcc -fPIC -c passthrough.c -o passthrough.o -ldl
```

- `-fPIC` : Generates position-independent code (necessary for shared libraries).
- `-ldl` : Links the dynamic loader library (`dlopen`, `dlsym`).

2. Link the compiled object into your application:

```
gcc main.c passthrough.o -o myapp -ldl
```

3. Example Code

`main.c` – Using CUDAWRAP to call CUDA functions.

```

#include <stdio.h>
#include "passthrough.h" // Include the header

int main() {
    // Step 1: Load the CUDA Runtime Library (libcudart)
    if (load_cudart_lib() != 0) {
        fprintf(stderr, "Failed to load libcudart library!\n");
        return 1;
    }

    // Step 2: Initialize CUDA
    int deviceCount = 0;

    // Dynamically call the cudaGetDeviceCount function
    typedef cudaError_t (*cudaGetDeviceCount_t)(int *);
    cudaGetDeviceCount_t cudaGetDeviceCount =
        (cudaGetDeviceCount_t)CUDART_FIND_ENTRY(cudart_lib_entry, cudaGetDeviceCount);

    if (cudaGetDeviceCount == NULL) {
        fprintf(stderr, "Function cudaGetDeviceCount not found!\n");
        return 1;
    }

    cudaError_t err = cudaGetDeviceCount(&deviceCount);
    if (err != cudaSuccess) {
        fprintf(stderr, "CUDA Error: %s\n", cudaGetString(err));
        return 1;
    }

    printf("Number of CUDA devices: %d\n", deviceCount);

    // Step 3: Use another CUDA function
    void *devPtr;
    size_t size = 1024;

    typedef cudaError_t (*cudaMalloc_t)(void **, size_t);
    cudaMalloc_t cudaMalloc =
        (cudaMalloc_t)CUDART_FIND_ENTRY(cudart_lib_entry, cudaMalloc);

    if (cudaMalloc == NULL) {
        fprintf(stderr, "Function cudaMalloc not found!\n");
        return 1;
    }

    err = cudaMalloc(&devPtr, size);
    if (err != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed: %s\n", cudaGetString(err));
    }
}

```

```

        return 1;
    }

    printf("Successfully allocated %zu bytes on the device\n", size);

    // Cleanup (if necessary)
    typedef cudaError_t (*cudaFree_t)(void *);
    cudaFree_t cudaFree =
        (cudaFree_t)CUDART_FIND_ENTRY(cudart_lib_entry, cudaFree);

    if (cudaFree != NULL) {
        cudaFree(devPtr);
    }

    return 0;
}

```

4. Explanation of Example

1. **Loading the Library:** The `load_cudart_lib()` function dynamically loads the CUDA runtime library and populates `cudart_lib_entry` with function pointers.
2. **Finding a Function:** Use `CUDART_FIND_ENTRY` macro to retrieve the pointer for the desired function (e.g., `cudaGetDeviceCount`, `cudaMalloc`).
3. **Calling the Function:** Cast the function pointer to the appropriate type and call it like a regular function.
4. **Handling Errors:** Check for `NULL` function pointers and handle errors using CUDA's error functions (`cudaGetErrorString`).

5. Logging and Debugging

Enable logging and performance tracking by compiling with the `DEBUG` or `FUNC_TIMINGS` flags. For example:

```
gcc -DDEBUG main.c passthrough.o -o myapp -ldl
```

This will log entry and exit times for CUDA functions, which is helpful for debugging and profiling.

6. Benefits of Using CUDAWRAP

- **Runtime Flexibility:** Swap between CUDA library versions without recompilation.
- **Debugging:** Trace function calls and measure performance.
- **Compatibility:** Avoid hard-coding library paths, making your application portable across systems with different CUDA installations.

Would you like further clarification on any part or help setting up a specific CUDA example?

Hijack CUBLAS

The `hijack_cublas.h` file builds upon the **passthrough** dynamic loading mechanism to provide wrappers for the cuBLAS library. Here's how you can understand and use this extension effectively.

Purpose of `hijack_cublas.h`

This header file:

1. Extends the **CUDAWRAP** framework to specifically work with the cuBLAS library.
 2. Introduces cuBLAS-specific types and enumerations (e.g., `cublasStatus_t`, `cublasOperation_t`).
 3. Implements mechanisms to hijack cuBLAS calls for logging, debugging, or dynamically replacing the cuBLAS library.
-

Key Components

1. Enums for cuBLAS Status and Operations

- `cublasStatus_t` : Represents return statuses for cuBLAS functions (e.g., success, allocation failure, or execution errors).
- `cublasOperation_t` : Represents matrix operations (`CUBLAS_OP_N` for normal, `CUBLAS_OP_T` for transpose, etc.).

2. `cublasContext`

- Placeholder structure that mimics a cuBLAS handle (`cublasHandle_t`).
- Contains:
 - `__initied__` : Indicates if the handle has been initialized.
 - `stream` : CUDA stream associated with the handle.

Note: This is a stub and may require proper initialization in a real-world scenario.

3. Function Wrapping

- `cublas_sym_t` : A generic type for function pointers to cuBLAS functions.
 - `cublas_entry[]` : Similar to `cudart_lib_entry` in `passthrough.h`, it stores cuBLAS function names and pointers, allowing dynamic resolution.
-

Integration into a Project

To use `hijack_cublas.h` for cuBLAS calls:

1. Include Headers

```
#include "passthrough.h"
#include "hijack_cublas.h"
```

2. Initialize cuBLAS Library

Call `load_cublas_lib()` to dynamically load the cuBLAS library and initialize the `cublas_entry` table:

```
if (load_cublas_lib() != 0) {
    fprintf(stderr, "Failed to load cuBLAS library!\n");
    return -1;
}
```

3. Use cuBLAS Functions Dynamically

Retrieve and execute cuBLAS functions via `CUBLAS_FIND_ENTRY` macro:

```
// Example: cublasCreate_v2
typedef cublasStatus_t (*cublasCreate_t)(cublasHandle_t *);
cublasCreate_t cublasCreate = (cublasCreate_t)CUBLAS_FIND_ENTRY(cublas_lib_entry, cublasCreate_v2);

cublasHandle_t handle;
cublasStatus_t status = cublasCreate(&handle);

if (status != CUBLAS_STATUS_SUCCESS) {
    fprintf(stderr, "cublasCreate failed: %d\n", status);
    return -1;
}
```

Example Application

Here's a full example demonstrating how to use `hijack_cublas.h` with matrix multiplication (`cublasSgemm_v2`).

```

#include <stdio.h>
#include "hijack_cublas.h"

// Example: Matrix multiplication using cuBLAS

int main() {
    // Step 1: Load cuBLAS library
    if (load_cublas_lib() != 0) {
        fprintf(stderr, "Failed to load cuBLAS library!\n");
        return -1;
    }

    // Step 2: Create cuBLAS handle
    typedef cublasStatus_t (*cublasCreate_t)(cublasHandle_t *);
    cublasCreate_t cublasCreate = (cublasCreate_t)CUBLAS_FIND_ENTRY(cublas_lib_entry, cublasCreate_\
    cublasHandle_t handle;
    if (cublasCreate(&handle) != CUBLAS_STATUS_SUCCESS) {
        fprintf(stderr, "Failed to create cuBLAS handle!\n");
        return -1;
    }

    // Step 3: Perform SGEMM (Single-precision General Matrix Multiply)
    float alpha = 1.0f, beta = 0.0f;
    int m = 3, n = 3, k = 3;

    float A[m * k] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    float B[k * n] = {9, 8, 7, 6, 5, 4, 3, 2, 1};
    float C[m * n] = {0};

    typedef cublasStatus_t (*cublasSgemm_t)(cublasHandle_t, cublasOperation_t, cublasOperation_t,
                                            int, int, int, const float *, const float *, int,
                                            const float *, const float *, int, float *);
    cublasSgemm_t cublasSgemm = (cublasSgemm_t)CUBLAS_FIND_ENTRY(cublas_lib_entry, cublasSgemm_v2);

    if (cublasSgemm == NULL) {
        fprintf(stderr, "Function cublasSgemm_v2 not found!\n");
        return -1;
    }

    cublasStatus_t status = cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N,
                                         m, n, k, &alpha, A, m, B, k, &beta, C, m);

    if (status != CUBLAS_STATUS_SUCCESS) {
        fprintf(stderr, "cublasSgemm failed: %d\n", status);
        return -1;
    }
}

```

```
// Print result
printf("Matrix C:\n");
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        printf("%f ", C[i * n + j]);
    }
    printf("\n");
}

// Cleanup
typedef cublasStatus_t (*cublasDestroy_t)(cublasHandle_t);
cublasDestroy_t cublasDestroy = (cublasDestroy_t)CUBLAS_FIND_ENTRY(cublas_lib_entry, cublasDestr

if (cublasDestroy) cublasDestroy(handle);

return 0;
}
```

Key Steps in Example

1. **Loading:** `load_cublas_lib` initializes the function pointer table.
2. **Retrieving Function Pointers:** Use `CUBLAS_FIND_ENTRY` to locate specific cuBLAS functions.
3. **Calling Functions:** Use resolved function pointers like normal cuBLAS functions.
4. **Matrix Multiplication:** Performs single-precision GEMM (`cublasSgemm_v2`).
5. **Cleanup:** Destroy cuBLAS handle using `cublasDestroy_v2` .

Would you like a deeper dive into any specific cuBLAS function or other areas of this framework?

This script builds and installs the `libblocksruntime` library, a dependency for the `libdispatch` (also known as Grand Central Dispatch). Here's a detailed breakdown:

1. Navigate to the Source Directory

```
cd ${WDIR}/blocks-runtime/
```

- Changes the working directory to `${WDIR}/blocks-runtime/`. `${WDIR}` is likely a variable defined earlier in the environment, pointing to the base working directory.

2. Initialize Build Tools

```
libtoolize ; aclocal ; autoheader ; autoconf ; automake --add-missing
```

- This sequence prepares the build environment for the `libblocksruntime` library:
 - `libtoolize` : Prepares the source tree for `libtool`, which helps manage the creation of shared and static libraries.
 - `aclocal` : Generates `aclocal.m4` by gathering macro definitions, setting up the environment for `autoconf`.
 - `autoheader` : Creates a template for `config.h`, used for platform-specific configuration.
 - `autoconf` : Generates the `configure` script from `configure.ac`, which customizes the build process.
 - `automake --add-missing` : Generates `Makefile.in` and adds any missing auxiliary files (like `install-sh` or `missing`).

These steps ensure the project is ready for a standard `./configure && make` build process.

3. Prepare the Build Directory

```
rm -rf build ; mkdir -p build ; cd build
```

- `rm -rf build` : Deletes any existing `build` directory, ensuring a clean slate.
- `mkdir -p build` : Creates a new `build` directory if it doesn't already exist.
- `cd build` : Changes into the `build` directory where the actual compilation will take place.

4. Configure the Build

```
CC="${MocCC} ${MocCFLAGS} -fcommon" CXX="${MocCXX} ${MocCXXFLAGS} -fcommon" \
../configure --prefix="${WDIR}/install/libblocksruntime" \
--disable-shared --enable-static --with-pic
```

- Sets the **C and C++ compiler flags**:
 - `cc` and `cxx` are set to `${MOCCC}` and `${MOCXX}` respectively, along with their corresponding flags (`${MOCFLAGS}` and `${MOCXXFLAGS}`), plus the `-fcommon` flag.
 - `-fcommon` ensures variables in multiple translation units are treated as common, improving compatibility with older codebases.
- Runs the `../configure` script with the following options:
 - `--prefix` : Specifies the installation directory as `${WDIR}/install/libblocksruntime` .
 - `--disable-shared` : Disables building shared libraries.
 - `--enable-static` : Enables building static libraries.
 - `--with-pic` : Ensures position-independent code is generated, which is often required for static libraries used in shared contexts.

This step customizes the build for the specific environment.

5. Build and Install

```
make -j$(nproc) install V=1
```

- `make -j$(nproc)` : Compiles the source code using as many parallel jobs as there are available CPU cores (`$(nproc)`).
- `install` : Installs the compiled library to the directory specified by the `--prefix` option in the `configure` script.
- `V=1` : Ensures verbose output during the `make` process, which shows detailed compilation commands.

Key Outputs

- The static version of the `libblocksruntime` library is built and installed to `${WDIR}/install/libblocksruntime` .
- Shared libraries are explicitly disabled to focus on creating static libraries for linking.

Purpose

This script is part of setting up a dependency for a larger project, ensuring that `libblocksruntime` is built in a controlled and predictable way, customized for the environment.

https://en.wikipedia.org/wiki/Grand_Central_Dispatch

text

`libdispatch`, also known as **Grand Central Dispatch (GCD)**, is a library designed to provide efficient, system-level support for concurrent and parallel programming. It was first introduced by Apple as part of macOS and iOS but has since been made open-source and ported to other platforms like Linux.

What is `libdispatch` For?

1. Concurrency and Parallelism

- `libdispatch` helps developers write concurrent programs more easily by managing threads and work queues internally.
- Instead of explicitly creating and managing threads, `libdispatch` allows you to schedule tasks (blocks of code) to execute asynchronously or synchronously.

2. Efficient Resource Utilization

- It uses a pool of threads and intelligently distributes tasks to available system resources, optimizing CPU and memory usage.
- Balances workloads across cores to maximize performance.

3. Thread Safety

- Simplifies thread-safe programming by encapsulating tasks in dispatch queues, avoiding race conditions and the need for extensive locks.

4. Event Handling

- Allows for event-driven programming by integrating tasks with file descriptors, timers, or other signals.
-

Key Features of `libdispatch`

1. Dispatch Queues

- A queue where tasks are scheduled to execute.
- Types:
 - **Serial Queue:** Tasks are executed one at a time in order.
 - **Concurrent Queue:** Tasks are executed in parallel, as resources allow.
 - **Main Queue:** A special serial queue for tasks that must run on the main thread, typically UI updates.

2. Work Items

- Blocks of code that you enqueue for execution.
- Can be written in C, Objective-C, or Swift.

3. Dispatch Sources

- Event sources that trigger the execution of tasks, such as timers, file descriptors, or custom triggers.

4. Group Management

- Dispatch groups allow you to group tasks and wait for their collective completion.

5. Timers

- Built-in support for creating and managing timers in a queue-based system.

When to Use **libdispatch**

1. Asynchronous Tasks

- When tasks can be performed independently of each other, allowing the program to remain responsive (e.g., downloading a file while processing user input).

2. Parallel Processing

- Tasks that can run simultaneously to leverage multicore processors, such as image processing, data analysis, or sorting algorithms.

3. Thread-Safe Code

- Simplifies concurrent access to shared resources without explicit locks.

4. Event-Driven Programming

- Ideal for applications that need to respond to events like network requests, user input, or timers.

5. Performance-Critical Applications

- Helps offload heavy computational tasks without blocking the main thread.

How to Use libdispatch

In C:

```
#include <dispatch/dispatch.h>

void my_function() {
    printf("Task executed\n");
}

int main() {
    // Create a dispatch queue
    dispatch_queue_t queue = dispatch_queue_create("com.example.myqueue", DISPATCH_QUEUE_CONCURRENT);

    // Add tasks to the queue
    dispatch_async(queue, ^{
        my_function();
    });

    dispatch_async(queue, ^{
        printf("Another task executed\n");
    });

    // Wait for some time to let tasks finish
    sleep(1);

    // Clean up
    dispatch_release(queue);
    return 0;
}
```

In Swift (if using on macOS or iOS):

```

import Dispatch

let queue = DispatchQueue(label: "com.example.myqueue", attributes: .concurrent)

// Schedule tasks
queue.async {
    print("Task 1 executed")
}

queue.async {
    print("Task 2 executed")
}

// Dispatch to the main queue
DispatchQueue.main.async {
    print("UI updates can go here")
}

```

Advantages of `libdispatch`

1. **Simplifies Multithreading:** Abstracts low-level threading details.
2. **Efficient:** Optimizes resource use, particularly in multicore systems.
3. **Portable:** Works on macOS, iOS, Linux, and more.

Limitations

- Not a substitute for more complex threading needs like thread-local storage or thread prioritization.
- For very fine-grained control of threads, alternatives like POSIX threads or C++ `std::thread` might be better.

In short, `libdispatch` is an essential tool for building modern, high-performance applications, especially when concurrency and responsiveness are crucial.

macOS GCD - Grand Central Dispatch

[macOS GCD - Grand Central Dispatch](#)

Grand Central Dispatch (GCD), also known as libdispatch (libdispatch.dylib), is available in both macOS and iOS. It's a technology developed by Apple to optimize application support for concurrent (multithreaded) execution on multicore hardware. Grand Central Dispatch (GCD), 也称为libdispatch (libdispatch.dylib), 在 macOS 和 iOS

中均可用。它是 Apple 开发的一项技术，旨在优化多核硬件上并发（多线程）执行的应用程序支持。

GCD provides and manages FIFO queues to which your application can submit tasks in the form of block objects. Blocks submitted to dispatch queues are executed on a pool of threads fully managed by the system. GCD automatically creates threads for executing the tasks in the dispatch queues and schedules those tasks to run on the available cores. GCD 提供并管理 FIFO 队列，您的应用程序可以向其中以块对象的形式提交任务。提交到调度队列的块在完全由系统管理的线程池上执行。GCD 自动创建线程来执行调度队列中的任务，并安排这些任务在可用内核上运行。

In summary, to execute code in parallel, processes can send blocks of code to GCD, which will take care of their execution. Therefore, processes don't create new threads; GCD executes the given code with its own pool of threads (which might increase or decrease as necessary). 总之，要并行执行代码，进程可以将代码块发送到 GCD，GCD 将负责它们的执行。因此，进程不会创建新线程；而是创建新线程。GCD 使用自己的线程池执行给定的代码（可能会根据需要增加或减少）。

This is very helpful to manage parallel execution successfully, greatly reducing the number of threads processes create and optimising the parallel execution. This is ideal for tasks that require great parallelism (brute-forcing?) or for tasks that shouldn't block the main thread: For example, the main thread on iOS handles UI interactions, so any other functionality that could make the app hang (searching, accessing a web, reading a file...) is managed this way. 这对于成功管理并行执行非常有帮助，大大减少了进程创建的线程数量并优化了并行执行。这非常适合需要大量并行性（强制？）的任务或不应阻塞主线程的任务：例如，iOS 上的主线程处理 UI 交互，因此任何其他可能导致应用程序挂起的功能（搜索、访问网络、读取文件...）都是以这种方式进行管理的。

[GCD Part 1: Queues and methods](#)

[Grand Central Dispatch \(GCD\) on FreeBSD](#)

[GCD Handbook](#)

```
sudo apt install libunwind-dev
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  libunwind-dev
0 upgraded, 1 newly installed, 0 to remove and 2 not upgraded.
Need to get 441 kB of archives.
After this operation, 2,916 kB of additional disk space will be used.
Get:1 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 libunwind-dev amd64 1.6.2-3 [441 kB]
Fetched 441 kB in 1s (775 kB/s)
Selecting previously unselected package libunwind-dev:amd64.
(Reading database ... 296316 files and directories currently installed.)
Preparing to unpack .../libunwind-dev_1.6.2-3_amd64.deb ...
Unpacking libunwind-dev:amd64 (1.6.2-3) ...
Setting up libunwind-dev:amd64 (1.6.2-3) ...
Processing triggers for man-db (2.11.2-2) ...
```

The `Makefile` appears to be structured to build a project with specific compiler configurations tailored for a system, potentially the Fugaku supercomputer. Here's an analysis and an organized breakdown of its components:

Analysis of Key Sections in the `Makefile`

1. General Description:

- o The initial comments describe the `Makefile`'s purpose:
 - `make depend` : Generate dependencies using `makedepend`.
 - `make` : Build the executable (presumably `mycc`).
 - `make clean` : Remove intermediate (`*.o`) and executable files.

2. Compiler Configuration:

- o Defines the compilers:
 - `cc` : Default C compiler (`clang`).
 - `cxx` : Default C++ compiler (`clang++`).
 - `FC` : Default Fortran compiler (`flang`).

3. Compile-Time Flags:

- o Optimizations:
 - `OPTI` : Includes architecture-specific optimizations for the A64FX CPU (Fugaku) and link-time optimization.
- o Debug flags:
 - `DEBUG` : Disabled by default, can enable `-DDEBUG` and debugging flags (`-O0`, `-g`).
- o CUDA handling:
 - `RedirectCUDA` : Redirects CUDA functionality to a mock or alternative implementation.
 - `RedirCUDAPaths` : Adds custom paths for CUDA and cuDNN.
- o GCD:
 - `GrandCentralDispatch` : Enables Grand Central Dispatch (disabled by default).

Sorted Contents of the `Makefile`

1. Metadata and Instructions

```
# 'make depend' uses makedepend to automatically generate dependencies
# 'make'       build executable file 'mycc'
# 'make clean' removes all .o and executable files
```

- Instructions for the user about the purpose and usage of the `Makefile`.

2. Compiler Configuration

```
# Define the C compiler to use
CC ?= clang
CXX ?= clang++
FC ?= flang
```

- Specifies compilers for C, C++, and Fortran.

3. Optimization and Debug Flags

```
# Define any compile-time flags
OPTI ?= -Ofast -ffast-math -mcpu=a64fx -flto=thin
DEBUG ?= #-DDEBUG -O0 -g
```

- `OPTI` : Enables optimizations for speed, targeting the A64FX processor.
- `DEBUG` : Placeholder for debugging flags, commented out by default.

4. CUDA and Other Flags

```
RedirectCUDA ?= -DUSE_MoCCUDA=1 #=0
RedirectCUDA += #-DUSE_MoCREGISTERFNS
RedirCUDAPaths ?= -DPATHToMoCDeviceProp="$(shell pwd)"
RedirCUDAPaths += -DPATHToCUDA_ROOT="$(CUDA_TOOLKIT_ROOT_DIR)" -DPATHToCuDNN_ROOT="$(CuDNN_ROOT)"

GrandCentralDispatch ?= -DUSE_GCD=1 #=0
```

- Configurations for:
 - Redirecting CUDA calls (`RedirectCUDA`).
 - Setting custom paths for CUDA and cuDNN.
 - Enabling/disabling Grand Central Dispatch (`GrandCentralDispatch`).

Next Steps

The file likely includes build rules, dependency handling, and cleaning targets in subsequent sections. To fully analyze and organize the `Makefile`, I will inspect further lines and continue sorting its content into logical modules.

Additional Analysis and Organization of the `Makefile`

The content adds more configuration for building a complex project. Here's the breakdown of the additional sections:

5. Compiler Flags

```
CFLAGS ?= $(OPTI) $(DEBUG) $(RedirectCUDA) $(RedirCUDAPaths) $(GrandCentralDispatch) \
-Wall -Wno-unused-label -fPIC -fopenmp
CXXFLAGS ?= $(CFLAGS) -Wno-deprecated-declarations -std=c++14
```

- **CFLAGS** : Includes optimization, debugging, CUDA redirection, architecture-specific flags, and OpenMP support.
- **CXXFLAGS** : Builds upon `CFLAGS` with C++-specific flags, including compatibility for C++14.

6. Include Directories

```
LOCALINCL = -I$(shell pwd)/src/cudart -I$(shell pwd)/src/cudnn -I$(shell pwd)/src/cublas \
-I$(shell pwd)/src/utils -I$(shell pwd)/src/cudawrap
CINCLUDES ?= $(LOCALINCL) \
-DFUJITSU -I$(Torch_BUILD_ROOT)/ssl2/include \
$(shell pkg-config --keep-system-cflags --cflags libunwind) \
$(shell pkg-config --keep-system-cflags --cflags hwloc) \
-I$(LIBDIS_ROOT)/include
CXXINCLUDES ?= $(LOCALINCL) \
-I$(Torch_BUILD_ROOT)/aten/src \
-I$(Torch_BUILD_ROOT)/torch/include \
-I$(CUDA_TOOLKIT_ROOT_DIR)/include
FINCLUDES ?= $(LOCALINCL)
```

- **LOCALINCL** : Defines local include paths for CUDA, cuDNN, utilities, and CUDA wrappers.
- **CINCLUDES**, **CXXINCLUDES**, and **FINCLUDES** :
 - Extend local includes to support Torch library paths, CUDA Toolkit, and other dependencies (e.g., `libunwind`, `hwloc`).

7. Linker Flags and Library Paths

```
LFLAGS ?= -fuse-ld=lld \
-L$(Torch_BUILD_ROOT)/ssl2/lib -Wl,-rpath=$(Torch_BUILD_ROOT)/ssl2/lib \
$(shell pkg-config --keep-system-libs --libs-only-L libunwind) \
$(shell pkg-config --keep-system-libs --libs-only-L hwloc) \
-L$(LIBDIS_ROOT)/lib -L$(LIBPQ_ROOT)/lib -L$(LIBKQU_ROOT)/lib64 -L$(LIBBRT_ROOT)/lib \
-L$(VENV_SITEPACKAGES)/torch/lib -Wl,-rpath=$(VENV_SITEPACKAGES)/torch/lib
```

- Specifies:
 - The linker to use (`lld`).
 - Library search paths and runtime paths (e.g., `Torch`, `libunwind`, `hwloc`).

- Dependencies on additional libraries in specific paths (e.g., `LIBDIS_ROOT`, `LIBPWQ_ROOT`).
-

8. Libraries to Link

```
LIBS ?= -lopenblas \
$(shell pkg-config --keep-system-libs --libs-only-l libunwind) \
$(shell pkg-config --keep-system-libs --libs-only-l hwloc) \
-l:libdispatch.a -l:libpthread_workqueue.a -l:libkqueue.a -l:libBlocksRuntime.a \
-liberty -lstdc++ -lc10 -lpthread -lm
```

- Links required libraries, including:
 - Mathematical libraries (e.g., `openblas`).
 - System utilities (`libunwind`, `pthread`).
 - Blocks Runtime and other specific libraries for parallel and GPU operations.
-

Next Steps

The subsequent lines likely include definitions for:

- Source files.
- Build rules for object files and the final executable.
- Dependency generation. I will inspect the remaining lines to complete the analysis.

Final Breakdown of the `Makefile`

The remaining sections define source files, object file transformations, and build rules for the target executable. Here's an organized summary:

9. Source Files

```
CSRCS ?= src/cudawrap/passthrough.c \
src/cudart/hijack_cuda_runtime_api.c \
src/cudnn/hijack_cudnn.c \
src/cUBLAS/hijack_cUBLAS.c \
src/utils/utils.c src/utils/addTensor.c src/utils/async.c

CXXSRC ?= src/torch/hijack_torch_cuda.cpp

FSRCS ?= src/utils/frt_utils.f90
```

- `CSRCS` : C source files for CUDA wrapping and utilities.
 - `CXXSRC` : C++ source files for Torch and CUDA handling.
 - `FSRCS` : Fortran source files for utility functions.
-

10. Object File Definitions

```
.SUFFIXES: .c .cpp .f90
OBJS = $(CSRCS:.c=.o)
OBJS += $(CXXSRCS:.cpp=.o)
OBJS += $(FSRCS:.f90=.o)
```

- Transforms the source files into object files by replacing their extensions (`.c` , `.cpp` , `.f90`) with `.o` .

11. Target Executable and Libraries

```
# Define the executable file
MAIN = lib/libMoccuda.so
CUDNN_shared = dep/install/cudnn/lib64/libcudnn.so
CUDNN_static = dep/install/cudnn/lib64/libcudnn_static.a
```

- `MAIN` : The target output is a shared library (`libMoccuda.so`).
- `CUDNN_shared` and `CUDNN_static` : Paths for shared and static cuDNN libraries.

12. Build Rules

```
.PHONY: depend clean

all:      $(MAIN)
        @echo "Our CUDA wrapper" $(MAIN) "has been compiled,"
        @echo "use LD_PRELOAD=$(shell pwd)/lib/libMoccuda.so <binary> to apply it."

$(MAIN): $(OBJS)
```

- `all` : Default target, builds the `MAIN` target and outputs instructions for using the shared library.
- Dependencies:**
 - Relyes on compiled object files (`$(OBJS)`).

Overall Organization of the `Makefile`

- 1. Metadata and Instructions:**
 - Overview of `Makefile` usage.
- 2. Compiler and Build Configurations:**
 - Compiler settings (`cc` , `CXX` , `fc`).
 - Optimization (`OPTI`), debugging (`DEBUG`), and CUDA-related flags.
- 3. Include Paths:**
 - Defines paths for header files and dependencies.
- 4. Linker Flags and Libraries:**

- Specifies library paths (`LFLAGS`) and libraries to link (`LIBS`).

5. Source File Definitions:

- Lists source files for different languages (`CSRCS` , `CXXSRCS` , `FSRCS`).

6. Object File Transformations:

- Generates object files using suffix replacement.

7. Target and Build Rules:

- Builds the `MAIN` target (shared library) and provides a usage message.

Would you like assistance refining or modifying this `Makefile` ?

text

text

<https://spack.io/>

<https://github.com/spack/spack>

Welcome to Spack! Spack is a package manager for supercomputers, Linux, and macOS. It makes installing scientific software easy. Spack isn't tied to a particular language; you can build a software stack in Python or R, link to libraries written in C, C++, or Fortran, and easily swap compilers or target specific microarchitectures.

Learn more here.

This Bash script is designed to automate the generation of wrapper libraries for various CUDA-related libraries, which are commonly used in GPU-accelerated computing. These wrappers seem to integrate with **Score-P**, a performance analysis and tracing framework. The script processes libraries like `libcudart.so`, `libcublas.so`, and others to facilitate performance instrumentation or runtime monitoring of CUDA applications. Here's a detailed breakdown:

该 Bash 脚本旨在自动生成各种 CUDA 相关库的包装器库，这些库通常用于 GPU 加速计算。这些包装器似乎与 Score-P（一个性能分析和跟踪框架）集成。该脚本处理 `libcudart.so`、`libcublas.so` 等库，以促进 CUDA 应用程序的性能检测或运行时监控。以下是详细的细分：

Purpose of the Script

1. Wrapper Libraries Creation:

- Uses the `scorep-libwrap-init` tool to generate Score-P wrapper libraries for CUDA libraries.
- Enables capturing calls to CUDA libraries for debugging, profiling, or performance analysis.

2. Custom Adjustments:

- Edits wrapper source files and build configurations to include specific headers and modify compilation flags (e.g., adding `-fPIC` for position-independent code).

3. Instruments Specific CUDA Libraries:

- Covers libraries such as:
 - `libcudart.so` : Runtime API for CUDA.
 - `libcublas.so` : Basic Linear Algebra Subprograms (BLAS).
 - `libcufft.so` : Fast Fourier Transform (FFT).
 - `libcurand.so` : Random number generation.
 - `libcudnn.so` : Deep neural network support.
 - `libcusparse.so` : Sparse matrix operations.
 - `libnvToolsExt.so` : Profiling and debugging tools.
 - `libnccl.so` : Multi-GPU communication.

Key Sections of the Script

1. Preparations

- **Directory Setup:**

- Creates a `wrapper` directory for holding generated wrappers.

- **Environment Variables:**

- Defines paths to CUDA Toolkit, cuDNN, and NCCL libraries:

```
CUDA_TOOLKIT_ROOT_DIR=/usr/local/cuda-10.0/targets/x86_64-linux
CUDNN_ROOT_DIR=/work/opt/cuda/cudnn-10.0-linux-x64-v7.6.5.32
NCCL_ROOT_DIR=/work/opt/cuda/nccl_2.5.6-1+cuda10.0_x86_64
```

2. Wrapper Creation Loop

For each CUDA library:

- Executes `scorep-libwrap-init` :

```
scorep-libwrap-init --name ${LIBN} -x c++ \
--cppflags "-I${CUDA_TOOLKIT_ROOT_DIR}/include/" \
--ldflags "-L${CUDA_TOOLKIT_ROOT_DIR}/lib/" \
--libs "-l${LIBN}" --update wrapper/wrap_${LIBN}
```

- Generates initial wrapper sources and build files for the target library.
- Customizes source code (`libwrap.h`, `.filter` files) and the Makefile:
 - Includes appropriate CUDA headers like `cublas_v2.h` for `cublas`, `cudnn.h` for `cudnn`, etc.
 - Edits filtering rules to exclude certain symbols (e.g., `cuda*`, `curandGenerateBinomial*`).
- Builds the wrapper:

```
make libscorep_libwrap_${LIBN}_runtime.la
```

3. Special Handling

Some libraries require additional modifications:

- For `libcudart.so`, it adds a custom `set_launch_func` to handle kernel launches.
- For `libcufft.so`, includes `cufft.h` and `cuffttx.h`.

Output

The script generates a series of dynamic libraries (`.la` files) in the `wrapper` directory, such as:

- `libscorep_libwrap_cudart_runtime.la`
- `libscorep_libwrap_cublas_runtime.la`
- `libscorep_libwrap_cufft_runtime.la` ... and so on.

These libraries allow Score-P to monitor and trace GPU-related operations in programs that use these CUDA libraries.

Potential Use Cases

- **Performance Profiling:**
 - Developers can profile CUDA-based applications to analyze bottlenecks or resource usage.
- **Debugging and Instrumentation:**
 - The generated wrappers enable fine-grained debugging of CUDA calls.
- **Integration with HPC Environments:**

- Supports performance analysis for GPU-heavy workloads in high-performance computing (HPC) clusters.
-

Suggestions for Improvement

- **Documentation:**

- Add comments to clarify what specific modifications in each `sed` command are meant to achieve.

- **Portability:**

- Replace hardcoded paths (e.g., `/usr/local/cuda-10.0/`) with more configurable options or environment checks.

- **Error Handling:**

- Include error checks after key commands (`mkdir`, `scorep-libwrap-init`, `make`).

This script is highly specialized and targets developers working with GPU performance optimization or debugging using CUDA and Score-P.

这段 Bash 脚本旨在自动生成针对各种 CUDA 相关库的封装库（wrapper libraries），这些库通常用于 GPU 加速计算。封装库似乎与 **Score-P** 集成，这是一种性能分析和追踪框架。脚本处理诸如 `libcudart.so`、`libcublas.so` 等库，以便为 CUDA 应用程序提供性能检测或运行时监控功能。以下是详细的分解：

脚本的目的

1. 创建封装库:

- 使用 `scorep-libwrap-init` 工具为 CUDA 库生成 Score-P 封装库。
- 使其能够捕获对 CUDA 库的调用，用于调试、性能分析或运行时监控。

2. 自定义调整:

- 编辑封装源码文件和构建配置，例如包含特定头文件、修改编译标志（如添加 `-fPIC` 支持位置无关代码）。

3. 处理特定的 CUDA 库:

- 涵盖的库包括：
 - `libcudart.so` : CUDA 的运行时 API。
 - `libcublas.so` : 基本线性代数子程序 (BLAS)。
 - `libcufft.so` : 快速傅里叶变换 (FFT)。
 - `libcurand.so` : 随机数生成。
 - `libcudnn.so` : 深度神经网络支持。
 - `libcusparse.so` : 稀疏矩阵运算。
 - `libnvToolsExt.so` : 性能分析和调试工具。
 - `libncccl.so` : 多 GPU 通信。

脚本的主要部分

1. 准备阶段

- 目录创建:
 - 创建 `wrapper` 目录，用于存放生成的封装库。
- 环境变量定义:
 - 定义 CUDA Toolkit、cuDNN 和 NCCL 库的路径:

```
CUDA_TOOLKIT_ROOT_DIR=/usr/local/cuda-10.0/targets/x86_64-linux
CUDNN_ROOT_DIR=/work/opt/cuda/cudnn-10.0-linux-x64-v7.6.5.32
NCCL_ROOT_DIR=/work/opt/cuda/nccl_2.5.6-1+cuda10.0_x86_64
```

2. 循环生成封装库

对于每个 CUDA 库:

- 执行 `scorep-libwrap-init` :
- ```
scorep-libwrap-init --name ${LIBN} -x c++ \
--cppflags "-I${CUDA_TOOLKIT_ROOT_DIR}/include/" \
--ldflags "-L${CUDA_TOOLKIT_ROOT_DIR}/lib/" \
--libs "-l${LIBN}" --update wrapper/wrap_${LIBN}
```
- 为目标库生成初始的封装源码和构建文件。
  - 自定义源码（如 `libwrap.h` 和 `.filter` 文件）以及 `Makefile`:
    - 包含适当的 CUDA 头文件，例如针对 `cublas` 的 `cublas_v2.h`，针对 `cudnn` 的 `cudnn.h` 等。
    - 编辑过滤规则以排除某些符号（例如 `cuda*`、`curandGenerateBinomial*`）。
  - 编译封装库:
- ```
make libscorep_libwrap_${LIBN}_runtime.la
```

3. 特殊处理

某些库需要额外的修改:

- 对于 `libcudart.so`，添加了自定义的 `set_launch_func` 以处理内核启动。
- 对于 `libcufft.so`，包含了 `cufft.h` 和 `cufftTxt.h`。

输出

脚本在 `wrapper` 目录中生成一系列动态库（`.la` 文件），例如:

- `libscorep_libwrap_cudart_runtime.la`
- `libscorep_libwrap_cublas_runtime.la`
- `libscorep_libwrap_cufft_runtime.la` ... 等。

这些库允许 Score-P 监控和跟踪程序中与 GPU 相关的操作。

潜在用途

- 性能分析:
 - 开发者可以分析 CUDA 应用程序的性能瓶颈或资源使用情况。
 - 调试与监控:
 - 生成的封装库支持对 CUDA 调用的细粒度调试。
 - 与高性能计算环境集成:
 - 支持在高性能计算 (HPC) 集群中分析 GPU 密集型工作负载的性能。
-

改进建议

- 文档:
 - 添加注释，说明每个 `sed` 命令的具体修改目的。
 - 移植性:
 - 将硬编码路径（如 `/usr/local/cuda-10.0/`）替换为可配置选项或环境检查。
 - 错误处理:
 - 在关键命令（如 `mkdir`、`scorep-libwrap-init`、`make`）后添加错误检查。
-

此脚本高度专业化，适用于使用 CUDA 和 Score-P 进行 GPU 性能优化或调试的开发者。

Morden Cpp

Links

- [cppreference](#)
- [modern-cpp](#)
- [hackingcpp](#)
- [cppweeklynews](#)
- RVO and NRVO
- [github: Mq-b/ModernCpp-ConcurrentProgramming-Tutorial](#)

Misc

Smart Pointers

All programming languages must answer:

- how to allocate the resources?
- how to get the read only access to the resource?
- how to get the mut/write access to the resource?
- how to pass or move the access of the resource?
- when to give up the access to the resource?
- when to release the resources?
- `~`
- practical answer: resource ownership with lifetime

[Back to Basics: Smart Pointers and RAII - Inbal Levi - CppCon 2021 hacking cpp: unique_ownership](#)

- The ownership model in C++
- syntax and design of smart pointers and RAII

ownership events:

- moving an object
- passing an object as a function parameter
- returning an object from a function

smart pointers

- `unique_ptr` : single ownership
- `shared_ptr` : multiple ownership
- `weak_ptr` : non ownership

RESOURCE MANAGEMENT-RAII

- Standard library classes using RAII
 - `std::string`, `std::vector` - free memory on DTOR
 - `std::jthread` - rejoin on DTOR
 - Standard library utilities
 - `std::unique_lock` - exclusive mutex wrapper (C++11)
 - `std::shared_lock` - shared mutex wrapper (C++14)
-

- std::lock_guard - ownership of a mutex in a scope (C++11)
- std::scoped_lock - ownership of multiple mutexes (avoids deadlock) (C++11)
- experimental::scope_exit - general purpose scope guard
- Guidelines Support Library ([github: Microsoft GSL](#))
 - gsl::owner - a wrapper which prevents multiple ownership to an object

unique_ptr

[hacking cpp: unique_ptr](#)

unique_ptr<T>

C++11

- holds one object (allocated on the Heap)
- pointer destroyed ⇒ object destroyed
- **unique ownership:** only one unique_ptr can own an object
- not copyable

```
#include <memory>

void foo() {
    C++14
    unique_ptr<int> p = make_unique<int>(); // creates int on Heap
    *p = 5; // do something with it
} // ← int object is destroyed automatically!
```

unique_ptr<T>

C++11

- **not copyable**
- but **movable**

```
void foo() {
    auto p1 = make_unique<int>();

    if(...) {
        auto p2 = p1;  COMPILER ERROR: can't copy

        auto p3 = move(p1);  // p3 owns object now, p1 holds nullptr
        ...
    } // p3 and object get destroyed
} // p1 gets destroyed
```

Returning and Parameter Passing

C++11

return

- by value ⇒ if cheap to copy or move, smart pointer otherwise
- from factories ⇒ **unique_ptr** / **shared_ptr** – no owning raw pointers
- references ⇒ only, if referenced object outlives function
- multiple types ⇒ **tuple<...>** better: struct/class

parameters

- copy needed inside function ⇒ pass by **value**
- no copy needed in function ⇒ pass by **const reference**
- mutate external object ⇒ pass by **reference**
- value optional ⇒ pass as **optional<T>** (or **const T***)
- steal contents of temporary ⇒ pass by **rvalue reference**

shared_ptr

hacking cpp: shared_ownership.



PEG: Parsing Expression Grammar 解析表达式语法

- [Parsing_expression_grammar](#)
- [awesome-pest](#)

In computer science, a parsing expression grammar (PEG) is a type of analytic formal grammar, i.e. it describes a formal language in terms of a set of rules for recognizing strings in the language. The formalism was introduced by Bryan Ford in 2004[1] and is closely related to the family of top-down parsing languages introduced in the early 1970s. Syntactically, PEGs also look similar to context-free grammars (CFGs), but they have a different interpretation: the choice operator selects the first match in PEG, while it is ambiguous in CFG. This is closer to how string recognition tends to be done in practice, e.g. by a recursive descent parser.

在计算机科学中，解析表达式语法（PEG）是一种分析形式语法，即它根据一组用于识别语言中字符串的规则来描述形式语言。该形式主义由 Bryan Ford 于 2004 年提出，与 20 世纪 70 年代初推出的自顶向下解析语言系列密切相关。从语法上讲，PEG 看起来也与上下文无关语法（CFG）类似，但它们有不同的解释：选择运算符选择 PEG 中的第一个匹配项，而在 CFG 中它是不明确的。这更接近于实践中字符串识别的方式，例如通过递归下降解析器。

Unlike CFGs, PEGs cannot be ambiguous; a string has exactly one valid parse tree or none. It is conjectured that there exist context-free languages that cannot be recognized by a PEG, but this is not yet proven.[1] PEGs are well-suited to parsing computer languages (and artificial human languages such as Lojban) where multiple interpretation alternatives can be disambiguated locally, but are less likely to be useful for parsing natural languages where disambiguation may have to be global.[2]

与 CFG 不同，PEG 不能含糊不清；一个字符串只有一个有效的解析树，或者没有。据推测，存在 PEG 无法识别的上下文无关语言，但这尚未得到证实。1 PEG 非常适合解析计算机语言（以及Lojban等人工人类语言），其中可以有多种解释替代方案。在本地消除歧义，但对于解析可能必须是全局消歧的自然语言不太有用。

pest. The Elegant Parser

- [pest.rs](#)
- [pest.docs.rs](#)
- [pest.book](#)

pest is a general purpose parser written in Rust with a focus on **accessibility**, **correctness**, and **performance**. It uses [parsing expression grammars \(or PEG\)](#) as input, which are similar in spirit to regular expressions, but which offer the enhanced expressivity needed to parse complex languages.

pest 是一个用 Rust 编写的通用解析器，重点关注可访问性、正确性和性能。它使用解析表达式语法（或 PEG）作为输入，其本质上与正则表达式类似，但提供了解析复杂语言所需的增强表达能力。

```
alpha = { 'a'..'z' | 'A'..'Z' }

digit = { '0'..'9' }

ident = { (alpha | digit)+ }

ident_list = _{ !digit ~ ident ~ (" " ~ ident)+ }

// ^
// ident_list rule is silent (produces no tokens or error reports)
```

```
int main() {
    return 5;
}
```

```
- FuncDecl
- int_t: "int"
- Identifier: "main"
- FormalParams: ""
- Block > Stmt > Return > Expr > Integer: "5"
```

GPT-Academic Report

Title:

Polygeist: Raising C to Polyhedral MLIR

Abstract:

We present Polygeist, a new compilation flow that connects the MLIR compiler infrastructure to cutting edge polyhedral optimization tools. It consists of a C and C++ frontend capable of converting a broad range of existing codes into MLIR suitable for polyhedral transformation and a bi-directional conversion between MLIR and OpenScop exchange format. The Polygeist/MLIR intermediate representation featuring high-level (affine) loop constructs and n-D arrays embedded into a single static assignment (SSA) substrate enables an unprecedented combination of SSA-based and polyhedral optimizations. We illustrate this by proposing and implementing two extra transformations: statement splitting and reduction parallelization. Our evaluation demonstrates that Polygeist outperforms on average both an LLVM IR-level optimizer (Polly) and a source-to-source state-of-the-art polyhedral compiler (Pluto) when exercised on the Polybench/C benchmark suite in sequential (2.53x vs 1.41x, 2.34x) and parallel mode (9.47x vs 3.26x, 7.54x) thanks to the new representation and transformations.

Meta Translation

标题: Polygeist: 将C提升到多面体MLIR

摘要: 我们提出了Polygeist, 这是一种新的编译流程, 旨在将MLIR编译基础设施与前沿的多面体优化工具连接起来。它包括一个C和C++前端, 能够将广泛的现有代码转换为适合多面体变换的MLIR, 并实现MLIR与OpenScop交换格式之间的双向转换。Polygeist/MLIR中间表示具备高层(仿射)循环构造和嵌入到单一静态赋值(SSA)基础结构中的n维数组, 能够实现前所未有的SSA基础和多面体优化的结合。我们通过提出和实现两个额外变换: 语句分割和归约并行化, 来证明这一点。我们的评估表明, Polygeist在Polybench/C基准套件的顺序模式(2.53x对比1.41x, 2.34x)和并行模式(9.47x对比3.26x, 7.54x)上的表现平均优于LLVM IR级别优化器(Polly)和先进的源到源多面体编译器(Pluto), 这得益于新的表示和变换。

I. INTRODUCTION

I. 引言

提高计算效率一直是计算领域的主要目标之一。通过利用并行性、时间和空间局部性以及其他性能来源, 可以显著提高程序性能。当针对现代多核CPU和GPU进行相关程序变换时, 这些变换尤其繁琐且具有挑战性, 因为现代架构具有深度内存层次和并行性, 通常由优化编译器自动执行。

多面体模型能够实现精确分析和相对简单的变换规范(如循环重构、自动并行化等), 以利用硬件性能来源。因此, 有越来越多的证据表明, 多面体模型是高效转化计算密集型程序[1]、[2]、[3]和编程加速器架构[4]、[5]、[6]的最佳框架之一。因此, 编译器社区专注于构建工具, 以识别和优化可以在多面体模型中表示的程序部分(通常称为静态控制部分或SCoP)。这些工具往往分为两类。

基于编译器的工具，如 Polly [7] 和 Graphite [8]，在编译器中间表示（IR）中检测和转换 SCoP。尽管这与编译器的其余部分无缝集成，但缺乏高层次结构和信息限制了这些工具进行分析和变换的能力。这种结构通常需要通过优化后的 IR 恢复，往往是以不完全或高成本的方式实现 [9]。此外，诸如 LICM 的常见编译器优化可能会干扰这一过程 [10]。最后，低级 IR 通常缺乏由变换生成的并行性或归约的构造，从而使流程更加复杂。

源到源编译器，如 Pluto [11]、POCC [12] 和 PPCG [5]，直接在 C 或 C++ 代码上操作。虽然这可以有效地利用源代码中的高层信息，但这些工具的有效性往往因缺乏启用优化而降低，例如将危险的内存加载转换为单赋值虚拟寄存器。此外，变换结果必须用 C 表达，而 C 语言本身的复杂性众所周知 [13]、[14]，而且也缺乏例如归约循环或未被内存存储支持的寄存器值的构造。

本文提出并评估了一种多面体编译流程 Polygeist（图 1）的好处，该流程可以利用源代码中可用的高层结构以及由低级 IR 提供的编译器优化的细粒度控制。它构建在最近的 MLIR 编译器基础设施之上，该基础设施允许在同一表示中、在同一变换过程中混合多个抽象层次 [15]。可互换的 MLIR 抽象或方言包括高层构造，如循环、并行和归约模式；完整覆盖 LLVM IR 的低层表示 [16]；以及一种受到多面体启发的表示，具有带有仿射表达式注释的循环和内存访问。此外，通过将源级和 IR 级工具的优势结合在一个端到端的多面体流程中，Polygeist 保留了高层信息并利用这些信息在较低级别抽象上执行新的或改进的优化，如语句拆分和循环携带值检测，同时影响后续优化。

我们的贡献如下：

- 一个 C 和 C++ 的 MLIR 前端，保留原始源代码中的高层循环结构；
- 一个端到端的流程，实现向多面体模型的提升和降低，利用我们的抽象执行比源级和 IR 级工具更多的优化，包括归约并行化；
- 探索 Polygeist 创建的新变换机会，尤其是语句拆分；
- 以及 Polygeist 与最先进的基于源和 IR 的工具（Pluto [11] 和 Polly [14]）之间的端到端比较，以及优化案例研究。

A. Overview

A. 概述

MLIR 是一种优化编译器基础设施，受到 LLVM 的启发，重点关注扩展性和模块化。其主要创新在于支持完全可扩展的一组指令（称为操作）和类型的中间表示（IR）。实际上，MLIR 将静态单赋值（SSA）与嵌套区域相结合，使得能够将一系列概念表示为一类一流的操作，包括机器指令，例如浮点加法、结构化控制流，例如循环、硬件电路，以及大型机器学习图。操作定义了程序的运行时语义，并处理不可变值。有关值的编译时信息用类型表示，而有关操作的信息用属性表示。操作可以附带区域，而这些区域内部又包含其他操作的（基本）块。所有操作接受的通用语法在图 2 中展示了 MLIR 的结构。此外，MLIR 支持用户自定义的自定义语法。

属性、操作和类型被组织在方言中，可以视为模块化库。MLIR 提供了少量方言，以定义常见操作，如模块、函数、循环、内存或算术指令，以及像整数和浮点数这样的通用类型。我们将在接下来的章节中讨论与 Polygeist 相关的方言。

B. Affine and MemRef Dialects

B. 仿射和内存引用方言

仿射方言 [18] 旨在表示具有显式适合多面体的循环和条件构造的静态计算程序（SCoP）。其表示的核心是以下值类别的分类：

- **符号**——已知为循环不变但在编译时未知的整数值，在多面体文献中也称为程序参数，通常是数组维度或函数参数。在 MLIR 中，符号是在具有“仿射范围”语义的操作的顶级区域中定义的值，例如函数；或者数组维度、常量和仿射映射（见下文）应用结果，无论它们的定义点在哪里。
- **维度**——是符号的扩展，接受仿射循环的归纳变量。
- **非仿射**——任何其他值。符号和维度具有索引类型，即平台特定的整数，适合指针（在 C 中为 `intptr_t`）。

MLIR 提供了两个与仿射方言相关的属性：

- **仿射映射**是多维的（准）线性函数，它将一组维度和符号参数映射到一组结果。例如，

$$((d_0, d_1, d_2, s_0) \rightarrow (d_0 + d_1, s_0 \cdot d_2))$$

是一个二维准仿射映射，可以在 MLIR 中表示为 `affine_map<(d0, d1, d2)[s0] -> (d0+d1, s0 * d2)>`。维度（左侧的括号内）和符号（左侧的方括号内）被分隔开，以允许准线性表达式：符号被视为常量，因此可以与维度相乘，而两个维度的乘积是无效的。

- **整数集**是被（准）线性表达式的合取限制的整数元组集合。例如，“三角形”集合 $\{(d_0, d_1) : 0 \leq d_0 < s_0 \wedge 0 \leq d_1 \leq d_0\}$ 表示为 `affine_set<(d0, d1)[s0]: (d0 >= 0, s0-d0-1 >= 0, d1 >= 0, d0-d1 >= 0)>`。

仿射方言利用上述概念定义了一组操作。`affine.for` 是一个“for”循环，其循环不变的下界和上界表示为带有常量步长的仿射映射。`affine.parallel` 是一个“多重for”循环嵌套，其迭代可以并发执行。这两种循环都支持通过循环传递值进行归约，以及 `max(min)` 表达式的下（上）界。`affine.if` 是一个条件构造，带有一个可选的 `else` 区域，条件被定义为给定值包含在一个整数集中。最后，`affine.load` 和 `affine.store` 表示内存访问，其中地址计算表示为仿射映射。核心的 MLIR 类型——内存引用（memref），以及相应的 memref 方言也在图 3 中展示。memref 类型描述了指向内存的结构化多索引指针，例如，`memref<?xf32>` 表示一个浮点元素的一维数组；而 memref 方言提供了内存和类型操作，例如 `memref.dim` 检索 memref 对象的维度。memref 不允许内部别名，即不同的下标始终指向不同的地址。这有效地解决了限制多面体技术在 LLVM IR 级别应用的去线性化问题 [9]。在本文中，我们只考虑与兼容 C ABI（应用程序二进制接口）的连续行主存储对应的默认布局的 memrefs。实际上，memrefs 支持可作为仿射映射表达的任意布局，但在 Polygeist 的上下文中，这些并不是必需的。

C. Other Relevant Core Dialects

MLIR 提供了几十种方言。在这些方言中，只有少数几种与我们的讨论相关：

- 结构化控制流（scf）方言定义了控制流操作，例如循环和条件语句，这些操作不受仿射分类规则的限制。例如，`scf.for` 循环可以接受任何整数值作为循环边界，这些边界不一定是仿射表达式。
- 标准（std）方言包含常见的操作，例如整数和浮点算术，这被用作从更高级别方言到多个目标方言的共同降低点，可以视为 LLVM IR 的概括[16]。
- LLVM 方言直接将 LLVM IR 指令和类型映射到 MLIR，主要是为了简化它们之间的转换。
- OpenMP 方言提供了一种与方言和平台无关的 OpenMP 指令表示，例如“parallel”和“workshare loop”，可用于转换 OpenMP 构造或生成与 OpenMP 运行时交互的 LLVM IR。
- 数学方言将整数和浮点类型上的数学操作（超出简单算术）归为一组，例如，`math.pow` 或 `math.sqrt`。

III. AN (AFFINE) MLIR COMPILATION PIPELINE

III. 一个（仿射）MLIR编译流程

Polygeist编译流程由4个组件组成（见图1）。这使得Polygeist能够保留原始程序中可用的更多结构（例如，多维数组），并能够与MLIR的高级内存操作进行交互。这与任何具有指针参数的C ABI不同，并且与C函数的接口不一致。Polygeist通过为函数参数和分配提供一个属性，使其使用与C兼容的指针类型，而不是memref，默认适用于外部函数，如strcmp和scanf。当使用memref ABI参数调用指针 ABI函数时，Polygeist生成包装代码，从memref中恢复与C ABI兼容的指针，并确保结果正确。图5展示了一个示例，演示了Polygeist与C ABI在一个小程序中的互动。

在分配和释放内存时，这种ABI的差异显得尤为重要。这是因为使用malloc分配数组的几个字节后，再将其转换为memref将不会产生合法代码（因为memref本身可能不是用原始指针实现的）。因此，Polygeist识别对分配和释放函数的调用，并将其替换为memref的合法等效函数。

函数和全局变量使用C或C++ ABI中使用的相同名称发出。这确保所有外部值都被正确加载，并且多版本函数（如C++模板或重载生成的函数）拥有不同的名称和定义。

c) 指令生成：对于大多数指令，Polygeist直接发出与等效C操作相对应的MLIR操作（如整数加法的addi，函数调用的call等）。对于某些特殊指令，如对pow的调用，Polygeist选择发出Math方言中特定的MLIR操作，而不是调用外部函数（在libm中定义）。这使得这些指令能够在MLIR中更好地进行分析和优化。

涉及内存或指针算术的操作需要额外处理。MLIR没有通用的指针算术指令；相反，它要求load和store操作包含所有被查找的索引。这为执行指针算术的操作带来了问题。为了解决这个问题，我们引入了一个临时子索引操作，用于memref，以跟踪附加的地址偏移。在Polygeist中的后续优化过程中，将子索引中的偏移量转发给任何使用它们的load或store。

d) 局部变量：局部变量通过在函数顶部分配一个memref在栈上进行处理。这使得C或C++的所需语义能够相对容易地实现。然而，由于许多局部变量和参数包含memref类型，这立即导致了memref的memref-这对大多数MLIR优化来说是一个障碍，因为在Polygeist之外这是不合法的。为了解决这个问题，我们实现了一个重量级的内存到寄存器（mem2reg）转换过程，该过程消除了MLIR构造中的不必要的加载、存储和分配。根据经验，这消除了Polybench套件中所有memref的memref。

B. Raising to Affine

将C或C++直接翻译为MLIR能够保留关于循环结构和n维数组的高级信息，但不会生成其他的仿射操作。之后，Polygeist将内存、条件和循环操作提升至其仿射方言对应物，如果它能够证明这些操作是合法的仿射操作。如果相应的前端代码被包裹在 #pragma scop 中，Polygeist假设在该区域内提升所有操作是始终合法的，且无需额外检查。对于任何未被证明或假设为仿射的操作，则保持不变。我们对仿射映射进行简化，以去除具有零次或一次迭代的循环，并删除在编译时已知条件的条件语句的分支。

a) Memory operations and loop bounds:

a) 内存操作和循环边界：为了转换一个操作，Polygeist用恒等仿射映射（affine_map<() [s0]->(s0)>[%bound]）替换其边界和下标操作数。它然后将计算映射操作数的操作，例如加法（addi）、乘法（muli），折叠到映射本身中。从循环归纳变量传递得出的值变成了映射维度，其他值则变成符号。例

如, `affine_map<()>[s0]>(s0)>[%bound]` 在 `%bound = addi %N, %i` 的情况下, 其中 `%i` 是一个归纳变量, 被折叠为 `affine_map<(d0)>[s0]>(s0 + d0)>(%i)[%N]`。该过程在无法再折叠操作或满足仿射值分类规则时终止。

b) 条件语句: 前端为两种输入代码模式 (if 条件和三元表达式) 生成条件操作。条件通过引入整数集合并将操作数折叠到其中进行转换, 类似于仿射映射, 此外, 操作之间的加法和运算分隔集合约束, 而非运算则对其进行反转 (`affine.if` 仅接受 ≥ 0 和 $= 0$ 的约束)。Polygeist 处理嵌套条件时遵循 C 风格的短路语义, 其中后续条件在前面条件的主体内进行检查, 当法律允许时, 将条件提升到最外层条件之外, 并用布尔操作或选择操作替换它们。在 `#pragma scop` 内, 这总是合法的。

为三元表达式生成的条件语句通常涉及其区域内的内存加载, 由于副作用, 导致无法提升。我们重用 `mem2reg` 通道, 在可能的情况下将其替换为等效的早期加载, 以实现提升。从经验上看, 这足以处理 Polybench/C 套件中的所有三元表达式 [19]。否则, 三元表达式将需要被下游多面体通道打包成单个语句。

C. Connecting MLIR to Polyhedral Tools

输入程序中使用 MLIR Affine 方言表达的区域适用于多面体模型。然而, 现有工具无法直接使用 MLIR。因此, 我们选择实现与 OpenScop [20] 的双向转换, 这是一种可被众多面体工具直接使用的交换格式, 包括 Pluto [11], 并且进一步可转换为 isl [21] 表示。这使得 Polygeist 能够与在多面体编译研究中创建的工具无缝连接, 而无需修改这些工具以支持 MLIR。

大多数多面体工具设计用于处理以语句为基础的 C 或 FORTRAN 输入, 而它们在 MLIR 中并没有直接对应的等价物。因此, 我们设计了一种机制, 从一系列 MLIR 操作创建类似语句的结构。例如:

```
void setArray(int N, double val, double * array) {...}
int main(int argc, char ** argv) { ...
    cmp = strcmp(str1, str2) ...
    double array [10];
    setArray(10, 42.0, array) }
```

可转换为:

```

func @setArray(%N: i32, %val: f64, %array: memref<?xf64>) {
    %0 = index_cast %N : i32 to index
    affine.for %i = 0 to %0 {
        affine.store %val, %array[%i] : memref<?xf64>
    }
    return
}

func @main(%argc: i32, %argv: !llvm.ptr<ptr<i8>>) -> i32 {
    ...
    %cmp = llvm.call @strcmp(%str1, %str2) :
    (!llvm.ptr<i8>, !llvm.ptr<i8>) -> !llvm.i32
    ...
    %array = memref.alloca() : memref<10xf64>
    %arraycst = memref.cast %array : memref<10xf64> to memref<?xf64>
    %val = constant 42.0 : f64
    call @setArray(%N, %val, %arraycst) :
    (i32, f64, memref<?xf64>) -> ()
}

```

图 5. 示例演示了 Polygeist 的 ABI。对于预期要由 Polygeist 编译的函数（如 setArray），指针参数被替换为 memref。对于需要外部调用约定（如 main(strcmp）的函数，Polygeist 退回到发出 llvm.ptr 并生成转换代码的方式。

我们进一步证明，这使得 Polygeist 能够通过控制语句粒度，积极影响多面体调度器的行为（详见第 III-D 节）。

a) 简单语句形成：观察到符合多面体模型的C语句主要是变量赋值，我们可以推导出一种机制，从MLIR操作链中识别语句。存储到内存中的操作是该语句的最后一个操作。该操作的逆向切片，即传递计算其操作数的操作，属于该语句。切片扩展在生成被分类为仿射维度或符号的值的操作处停止，这些值可以直接用于仿射表达式。此类值包括循环归纳变量或循环不变常量。一些操作可能出现在多个语句中，如果该值被多次使用。然而，我们需要操作与语句之间的映射是双向的，以便在调度器重新结构化程序后发出MLIR，而不考虑SSA值的可见性规则。如果一个多次使用的操作没有副作用，Polygeist会简单地复制它。对于重复操作的复制不合法的情况，Polygeist将其结果存储在堆栈分配的内存引用中，并将所有后续使用替换为内存加载。图6展示了在操作%20中使用的值%0的转换。这创建了一个新的语句。

b) 跨区域依赖关系：在某些情况下，一个语句可能由不同（嵌套）循环中的MLIR操作组成，例如，从内存加载到SSA寄存器的操作发生在外部循环中，而它在内部循环中被使用。该语句在循环层次结构中的位置并不明确。更重要的是，无法将其传达给多面体调度器。Polygeist通过在定义区域中将值存储在堆栈分配的内存引用中，并在使用区域中将其加载回去，从而解决了这一问题。图6展示了在操作%10中使用的值%0的这种转换。与基本情况类似，这在外部循环中创建了一个新的语句，可以独立调度。该方法可以视为reg2mem转换，这是在前端执行的mem2reg的反向操作。它仅适用于一部分值，并且在多面体调度完成后可以撤销。此外，为了减少依赖关系和内存占用，Polygeist执行简单的值分析，并在同一值在另一个内存位置可用并能够从那里读取时，避免创建堆栈分配的缓冲区。

c) SCoP 形成：为了定义一个 SCoP，我们将单独的语句划分为函数，这样它们就可以被表示为具有已知内存占用的透明调用，类似于 Pencil [22]。这个过程也使得语句间的 SSA 依赖关系变得清晰。这些依赖关系存在于使用相同 SSA 值的调用之间，但这些调用没有定义任何值。我们提升所有局部栈分配，并将它们放置在周围函数的入口块中，以保持它们在循环重构后仍然可见。图 7 展示了得到的 IR。

多面体表示的其余组件如下派生：语句的域被定义为所包含循环的迭代空间，由各自的下界和上界约束，并与任何“if”条件相交。这个过程利用了 MLIR 直接将边界和条件表示为仿射构造的特性。每个语句的访问关系作为 `affine.load`（读取）和 `affine.store`（必须写入）操作的仿射映射的并集获得，关系的右侧被注释为与被访问的 `memref` 对应的“数组”，这对应于 SSA 值。初始调度是使用 $(2d + 1)$ 形式分配的，奇数维度表示输入程序中循环的词法顺序，而偶数维度等于循环引导变量。OpenScop 中的仿射构造表示为线性等式 ($= 0$) 或不等式 (≥ 0) 系数的列表，这正好与 MLIR 中的内部表示相匹配，使转换变得简单明了。

d) 代码生成回 MLIR：Pluto 调度器因此生成了 OpenScop 中的新调度。从仿射调度生成循环结构是一个已解决但相当复杂的问题 [13], [14]。Polygeist 依赖于 CLoG [13] 来生成初始的循环级 AST，然后将其转换为仿射方言的循环和条件。在代码生成时无需简化仿射表达式，因为 MLIR 可以直接接受它们，并能够在后续阶段进行简化。语句作为具有重写操作数的函数调用引入，然后进行内联。

D. Controlling Statement Granularity

D. 控制语句粒度

回想一下，Polygeist 是通过原始操作的序列重建“语句”的（见第三节C）。我们最初设计了一种恢复与 C 输入中相似的语句结构的方法，但这并不是必需的。实际上，只要可以将语句组织成循环并进行拓扑排序（即语句之间没有使用-定义循环），语句可以由任何子集的 MLIR 操作形成。为了向仿射调度器暴露这些语句之间的依赖关系，我们重用了通过临时存储器的思想：每个语句将其他语句所需的值写入专用内存位置，随后语句从这些位置读取。临时存储器受到部分数组扩展的限制[23]，以最小化它们对仿射调度器的影响，因为单元素的临时存储器数组会产生人为的标量依赖关系。这种语句粒度的变化为仿射调度器提供了前所未有的灵活性，使其能够为同一 C 语句的不同部分选择不同的调度。以图8（顶部）中的语句 S 为例，该语句被三个针对 i、j 和 k 的循环包围。这种收缩模式在计算程序中很常见（此特定示例可以在相关基准中找到， $B=C$ ，见第五节E）。最佳利用局部性的循环顺序是 (k, i, j) ，这对从 B 的读取产生时效局部性（该值在现在最内层的 j 循环的所有迭代中被重用）并对从 C 的读取产生空间局部性（连续的值被连续的迭代读取，从而增加了 L1 缓存命中率的可能性）。然而，Pluto 从未提出这样的顺序，因为在 k 维度上由于对 $A[i][j]$ 的重复读/写访问而存在减少依赖关系，Pluto 趋向于选择依赖关系较少的循环作为最外层。虽然这种依赖关系本身是不可避免的，但它可以转移到图8（底部左侧）的单独语句 T 中。这种方法为调度器提供了更多的选择自由度，内存成本相对较低，而不是扩展整个 A 数组。它还将减少问题分解为一个“规范”语句，使得下游处理（例如向量化）更容易进行。

在 C 级别实现这一转换将需要操作 C 抽象语法树（AST）并推理 C（甚至 C++）的语义。这通常超出了诸如 Pluto 这样的源到源多面体优化器的能力，因为这些优化器将语句视为黑箱。虽然在 LLVM IR 级别实现这一转换也是可能的，例如在 Polly 中，在那里语句也被重建且暂时分配的注入较为容易，但驱动转换的启发式是基于循环结构和多维访问模式的，而在如此低的级别恢复这些信息是困难的[9]。

潜在分割的空间是巨大的——每个 MLIR 操作都有可能成为一个语句。因此，我们设计了一种启发式方法来处理类似于图8的收缩情况。减少语句分割适用于以下语句：

- 至少被 3 个循环包围；
- 左侧 = 右侧，并且使用所有循环，除了最内层的；
- 右侧具有两个或更多不同的访问模式。这涵盖了那些可能通过不同循环顺序改善局部性的语句，并且风险较低，不会产生不希望的分裂。这个启发式方法仅作为 Polygeist 能够启用的新转换类型的示例。

E. Post-Transformations and Backend

E. 后变换及后端

Polygeist 使得在 quasi-syntactic 和 SSA 层面上操作成为可能，从而实现了在任何一个层面上单独进行分析和优化时极其困难（甚至不可能）的任务。除了语句拆分，我们提出了两种展示 Polygeist 潜力的技术。

a) 转换带有延续值的循环（归约）：Polygeist 利用 MLIR 对循环携带值的一级支持，来检测、表达和转换类似归约的循环。这种支持不需要源代码注释，而与使用注释以启用检测的源级工具 [24] 不同，也不需要复杂的修改来进行并行代码的生成，与 Polly [25] 不同，后者受到 LLVM 缺失一级并行构造的限制。我们也未对多面体调度器进行修改，而是依赖后处理来进行归约并行化，包括最外层的并行归约循环。

总体方法遵循了 [26] 中提出的定义，针对 MLIR 的基于区域的中间表示进行了调整，如图 9 所示。Polygeist 通过扫描循环的单块主体，识别每次迭代中被修改的内存位置，即具有循环不变下标且没有交错别名存储的加载/存储对。这些被转换为循环携带值或二次归纳变量，加载/存储对被提升到循环外并重新用于读取初始值和存储最终值。在循环主体中，循环携带值可以通过一系列无副作用操作进行更新。如果这个链被认为是结合的和交换的，那么该循环即为归约。即使在没有合适的归约操作的情况下，循环携带值也能被检测到。具有此类值的循环将对 mem2reg 产生贡献，降低内存占用，但不适合进行并行化。

b) 延迟并行化：Polygeist 并不依赖于通过仿射调度器获取的依赖距离信息，而是对生成的代码进行单独的多面体分析以检测循环并行性。该分析本身是一种经典的多面体依赖分析 [27]、[28]，在 MLIR 区域结构之上实现。在进行 SSA 基于优化之后执行这一步，特别是 mem2reg 和归约检测，能够并行化更多的循环。特别是，归约循环和那些变量值仅在单次迭代内相关的循环（类似于活跃范围重排序 [29]，但没有昂贵的额外多面体分析）都得到处理（在循环中定义的 SSA 值的活跃范围从不超出循环）。

IV. EVALUATION

我们的评估有两个目标。1) 我们希望证明，Polygeist 生成的代码在没有额外优化的情况下，其性能与像 Clang 这样的现代编译器没有任何不可解释的差异。2) 我们探讨 Polygeist 的内部表示如何在同一编译流中支持结合使用仿射变换和基于 SSA 的变换，并评估与现有的源代码和编译器基础上的多面体工具相比的潜在优势。

A. Experimental Setup

我们在一台 AWS c5.metal 实例上进行实验，该实例禁用了超线程和 Turbo Boost。系统为 Ubuntu 20.04，运行于双插槽 Intel Xeon Platinum 8275CL CPU，主频为 3.0 GHz，每个插槽有 24 个核心，L1、L2 和 L3 缓存分别为 0.75 MB、35 MB 和 35.75 MB，并配备 256 GB RAM。我们使用“EXTRALARGE”数据集运行 Poly-Bench [19] 中的所有 30 个基准测试。Pluto 无法从 adi 基准测试中提取 SCoP。我们对每个基准测试进行了总共 5 次试验，采用 PolyBench 报告的执行时间；除非另有说明，结果以中位数的形式给出。以下各节中报告的每个测量或结果均与双精度数据相关。所有实验均在核心 1-8 上运行，以确保所有线程位于同一插槽，并且不会与调度在核心 0 上的进程发生潜在冲突。

在所有情况下，我们采用两阶段编译：(i) 使用 clang 以 -O3 级别编译，不包括展开和向量化；或使用 Polygeist 从 C 发出 LLVM IR；(ii) 使用 clang 以 -O3 级别发出最终二进制文件。由于某些优化并非幂等的，因此第二轮优化可能会显著提升（而很少情况会抑制）性能。这就是我们选择仅在最后优化阶段进行向量化和展开的原因。由于 Polygeist 在 MLIR 级别应用了一些优化（例如，mem2reg），我们将其与两阶段编译流程（CLANG）进行比较，作为更公平的基线。我们还评估了单阶段编译，以评估两阶段流程的效果（CLANGSING）。

B. Baseline Performance

B. 基线性能

Polygeist必须生成尽可能接近现有编译流程的运行时代码，以建立一个稳固的基线。换句话说，Polygeist不应该引入开销或加速，除非明确指示，以便能够测量额外优化的效果。我们通过比较Polygeist生成的程序与同一提交版本（2021年4月）下Clang生成的程序的运行时来评估这一点。图10总结了以下几种流程的结果：

- CLANG：使用Clang编译程序，运行两阶段优化的过程；
- CLANGSING：使用Clang编译程序，运行一阶段优化的过程；
- MLIR-CLANG：使用Polygeist前端的编译流程，在MLIR内进行预处理优化，但不执行多面体调度或后处理。

C. Compilation Flows

我们将Polygeist与源代码级和中间表示级优化器（Pluto和Polly）进行比较，配置如下：

- PLUTO：使用-polycc 3及-noparallel和-tile标志的Pluto编译器自动转换；
- PLUTOPAR：与上述相同，但使用-parallel标志；
- POLLY：Polly [7]的LLVM通道，具有仿射调度和切片，没有基于模式的优化 [30]；
- POLLYPAR：与上述相同，但进行了自动并行化；
- POLYGEIST：我们的工作流，结合了Pluto和额外的变换；
- POLYGEISTPAR：与上述相同，但使用-parallel的Pluto调度，Polygeist并行化和归约。

在源代码和LLVM IR级别之间运行时，我们期望Polygeist能够从这两者的优点中受益，从而生成的代码能够与竞争对手相媲美或更优。在使用Pluto时，无论是独立地还是在Polygeist中，我们都会禁用向量化提示和循环展开的发出，以确保这两种变换都完全由LLVM优化器控制，该优化器也在Polly流程中运行。我们在Clang编译的最新阶段运行Polly，使用-mllvm -polly及其他标志以启用仿射调度、切片和并行化。Polly与Clang使用相同的LLVM提交版本。我们禁用在其他地方不可用的基于模式的优化 [30]。图11和图12分别总结了顺序和并行流程的结果。

A. Benchmarking

降低循环的转换，特别是并行化，可能会导致部分结果累积的顺序发生变化。这在 IEEE 754 语义下是不允许的，但使用 -ffast-math 选项的编译器支持这种情况。我们发现，Polybench 的分配函数会妨碍 Clang/LLVM 的别名分析，从而对性能产生负面影响 2 LLVM 提交
20d5c42e0ef5d252b434bcb610b04f1cb79fe771 3 Pluto 提交 dae26e77b94b2624a540c08ec7128f20cd7b7985，举例来说，在 adi 中。因此，我们修改了所有基准测试，使用已知会产生非别名指针的 malloc。

B. Baseline Comparison

我们没有观察到CLANG和CLANGSING配置之间的运行时间存在显著差异，在基准测试中，它们的几何平均对称差为0.43%。因此，在本文的其余部分中，我们仅考虑CLANG作为基线。同样，我们也没有观察到CLANG和MLIR-CLANG配置之间的运行时间存在显著差异，其几何平均对称差为0.24%。

我们发现短运行基准的运行时间存在变动，特别是在jacobi-1d上。这可以归因于数据初始化和基准测试代码之间的相互作用，以及其他操作系统进程的影响。从分析中排除运行时间少于0.05秒的基准（如 jacobi-1d、gesummv、atax、bicg），我们分别获得上述两个比较的几何平均对称差为0.32%和0.17%。这

些结果表明，我们的流程在基线之上没有未解释的（不）优势。

C. Performance Differences in Sequential Code

整体而言，Polygeist 相较于 Pluto ($2.34\times$) 和 Polly ($1.41\times$) 提供了更大的加速，几何平均值为 $2.53\times$ ，尽管改进并不是系统性的。Polygeist 和 Polly 之间的一些差异是由于所采用的多面体调度器，例如在 lu 和 mvt 中。得益于语句分割，Polygeist 在 2mm、3mm 及其他某些情况下生成的代码速度超过了 Pluto 和 Polly，详见第 V-E 节。

在给定相同语句和调度的情况下，代码生成层面的优化是造成其他性能差异的原因。seidel-2d 是最明显的例子：Pluto 执行的整数指令比 Polygeist 多出 $2.7 \cdot 10^{11}$ 次。假设这些指令都是索引/地址计算，混合了 add（吞吐量为 $1/2$ 或 $1/4$ ）和 imul/shl（吞吐量为 1 ），我们可以预计在 3GHz 下会有 ≈ 59 秒的差异，这与实验观察结果一致。Polygeist 在其后优化阶段优化掉了一部分指令，并从 memref 发出适当机器大小类型的均匀地址计算，使下游编译器能够进行更激进的界限分析和简化。相反，由于 Polygeist 为了编译器性能原因放弃了对 CLOOG 代码的简化，jacobi-2d 的性能较差，在 40 个分支中有多达 75 个语句副本，而 Clang 处理这些代码需耗时多达 5 秒，但其结果却实现了更好的向量化。需要进一步的工作来解决这个问题，通过直接从 Polygeist 发出向量指令。

D. Performance Differences In Parallel Code

同样，对于并行代码，某些性能差异是由于不同的调度器。例如，在 cholesky 和 lu 中，Pluto 和 Polygeist 的性能都优于 Polly，剩余的差距可以归因于代码生成级别的差异。相反，在 gemver 和 mvt 中，Polly 相较于两者皆有优势。

图 10 显示了在 CLANG、CLANGSING 和 MLIR-CLANG 配置下，Polybench 的五次运行中程序运行时间的均值和 95% 置信区间（对数尺度），值越低越好。由 Polygeist 产生的未优化代码的运行时间与 Clang 的运行时间相当。不同程度的单重和双重优化之间没有观察到显著差异。运行时间较短的 jacobi-1d 显示出高的组内变异性。

图 11 显示了在顺序配置下相对于 CLANG 的中位数加速比（对数尺度），值越高越好。Polygeist 在平均情况下的加速比（几何均值加速比为 $2.53\times$ ）优于 Pluto ($2.34\times$) 和 Polly ($1.41\times$)。Pluto 无法处理 adi，因此该案例被排除在总结统计之外。

图 12 显示了在并行配置下相对于 CLANG 的中位数加速比（对数尺度），值越高越好。Polygeist 在平均情况下的加速比（几何均值加速比为 $9.47\times$ ）优于 Pluto ($7.54\times$) 和 Polly ($3.26\times$)。Pluto 无法处理 adi，因此该案例也被排除在总结统计之外。

在 ludcmp 和 syr(2)k 中，SSA 级别的优化使得 Polygeist 生成的代码比 Pluto 更快，并且至少与 Polly 同样快。这些结果表明，Polygeist 确实利用了仿射优化和 SSA 基础优化的优点。

Polygeist 是唯一在 deriche（加速比 $6.9\times$ ）和 symm（加速比 $7.7\times$ ）上实现加速的编译流程。通过检查输出代码，我们观察到只有 Polygeist 成功地对这两个基准进行了并行化。考虑到图 13 中的输入代码，可以观察到 i 循环重用了 ym1 变量，这被仿射调度器解释为妨碍并行化的循环携带依赖。Polygeist 在将 ym1 提升为一个由 j 循环携带的 SSA 寄存器后，进行自己的并行性分析，其使用定义范围不妨碍并行化。

类似地，Polygeist 的并行化器识别出两个具有并行归约循环的基准，这些循环不包含在其他并行循环中：gramschmidt 和 durbin。相比于 Polly 的 $34\times$ 和 Pluto 的 $54\times$ ，gramschmidt 在 Polygeist 的帮助下获得了 $56\times$ 的加速。durbin 则出现了 $6\times$ 的减速，因为新的并行循环迭代次数相对较少，并且嵌套在一个顺序循环内，导致同步成本超过了并行化带来的好处。第 V-F 节将更详细地探讨 durbin 基准。

Polybench 是一组（大多数已知）具有并行特性的代码，因此在 CPU 上进行归约并行化的需求不大，因为只有一度的并行性就已足够。然而，当针对固有的目标架构如 GPU 时，利用归约并行性可能对于实现峰值性能至关重要 [31], [24]。

E. Case Study: Statement Splitting

E. 案例研究：语句拆分

我们确定了 5 个基准测试，其中应用了语句拆分启发式方法：2mm、3mm、相关性、协方差和 trmm。为了评估变换的效果，我们执行了这些基准测试，并禁用了语句拆分，用 `-nosplit` 后缀标记，如图 14 所示。在顺序版本中，2mm 的速度慢了 4.1%（3.13 秒对比 3.26 秒），但其他基准测试分别看到了 25%、50%、51% 和 27% 的加速。对于并行版本，速度提升分别为 36%、20%、44%、40% 和 -9%。

对多面体调度器输出的检查表明，它确实生成了预期的调度。例如，在相关性基准中，语句 $A[i][j] += B[k][i] * B[k][j]$ Polygeist 能够在拆分后找到 (k, i, j) 的循环顺序。通过在顺序代码上使用硬件性能计数器，我们确认整体缓存缺失率确实分别降低了 75%、50%、20%、27% 和 -26%。然而，由总匣周期估算的内存流量在 2mm 中增加了 9%，而其他基准测试则分别减少了 18%、32%、32% 和 21%。该指标与同一运行中观察到的性能差异强相关 ($r = 0.99$, $p = 3 \cdot 10^{-11}$)。这种行为可能是由于调度器产生了不同的融合结构，例如，在 2mm 中没有融合最外层的循环，这也影响了局部性。并行代码中也可以观察到类似的结果。进一步的研究有必要利用 Polygeist 创造的语句拆分机会，并与融合进行相互作用。

F. Case Study: Reduction Parallelization in durbin

F. 案例研究：Durbin 中的归约并行化

在这个基准测试中，Polygeist 利用其归约优化创建了一个其他工具无法实现的并行循环。对于默认运行的相对较小的输入， $N = 4000$ 的迭代嵌套在另一个具有 N 次迭代的顺序循环中，整体性能出现下降。我们假设创建并行线程和同步它们的成本超过了额外并行性带来的收益，并通过增加 N 来验证我们的假设。根据图 15 中的结果，可以观察到，Polygeist 在 $N \geq 16000$ 时开始产生加速效果 (> 1)，而 Polly 仅在 $N \geq 224000$ 时才会如此，并且幅度要小得多：6.62× 对比 1.01×。在没有归约并行化的情况下，Polygeist 的表现与 Polly 相同。Pluto 未能对任何最内层循环进行并行化，并且显示没有加速。这支持了我们的假设，并强调了能够并行化归约的重要性。

VI. RELATED WORK

VI. 相关工作

a) MLIR 前端：自从 MLIR 被纳入 LLVM 框架后，已经创建了多个前端用于从特定领域语言生成 MLIR。Teckyl [2] 将面向生产力的张量理解 [1] 符号连接到 MLIR 的 Linalg 方言。Flang, LLVM 的 Fortran 前端，使用 FIR 方言建模 Fortran 特定的构造 [32]。COMET，一个针对化学领域的特定领域编译器，引入了一个从基于张量的语言到 MLIR 的特定领域前端 [33]。NPComp 旨在提供必要的基础设施以便编译数值 Python 和 PyTorch 程序，从而利用 MLIR 基础设施 [34]。PET-to-MLIR 通过解析 pet 的内部表示，将一部分多面体 C 代码转换为 MLIR 的 Affine 方言。除了当前不处理特定构造（如 if 语句、符号边界和外部函数调用）外，解析 pet 的表示限制了前端的可用性，因为它无法与非多面体代码（如初始化、验证或打印例程）接口 [35]。相对而言，Polygeist 能够从非多面体代码生成 MLIR（尽管不一定在 Affine 方言中）。CIRCT 是一个新的

项目，旨在将MLIR开发方法应用于电子设计自动化行业，隶属于LLVM框架 [17]。Stripe使用MLIR的Affine方言作为机器学习模型中的循环变换的基础，包括拆分和向量化，并接受自定义DSL作为输入 [36]。

b) Compilers Leveraging Multiple Representations:

b) 利用多种表示的编译器：SUIF编译器基础设施开创了一种组合内部表示，支持更高级的转换，包括循环优化和并行化 [37]，尤其是归约并行化 [38]。Polygeist利用SUIF中不可用的MLIR抽象：常规和仿射循环，OpenMP归约构造等。它还受益于仅作为外部扩展存在于SUIF中的SSA+区域形式 [39]，可以用于中间表示的简化。PIPS在针对OpenMP时支持循环转换和过程间优化 [40], [41]。Polygeist与这两者的不同之处在于它发出机器代码而不是源代码，从而能够发出并行运行时和其他在源语言（如C）中没有表示的指令。

c) 结合“经典”和多面体流程：少数论文关注于将“经典”的、大多数是AST级别的转化与多面体转化结合起来。PolyAST通过结合仿射调度器与AST级别的启发式方法进行融合和切片 [42]，开创了这种方法，尽管仅使用多面体转化也展示了类似的结果 [43]。CUDA-CHiLL [44] 也实验了一种类似的方法。可以说，许多自动化的多面体流程将循环融合和/或切片作为一个独立步骤来执行，这可以被认为是一种经典转化。Pluto [11] 使用几个“语法”后处理过程来利用模板中的空间局部性和并行性 [45]。已提出多个工具，以经典循环转化如融合和排列作为操作，驱动多面体循环转化，包括URUK [46]、CHiLL [47] 和Clay [48]。Polygeist与这些工具的不同之处在于它在其IR中保留了此类转化的结果，并且与多面体构造一起，允许不同抽象层次之间的交互。

d) 额外的（后）多面体转化：Polly [25] 提出了处理归约循环的支持，但并未实现代码生成。在语法层面，通过手动注释使用PENCIL指令向PET添加了归约支持 [24]。据报道，R-Stream使用了一种变体的语句拆分以影响调度器的行为并优化内存消耗 [49]。POLYSIMD在PPCG多面体流程周围使用变量重命名以改善向量化 [50]。Polygeist利用SSA和多面体信息自动化这些操作。

e) Integration of Polyhedral Optimizers into Compilers:

e) 聚合优化器与编译器的集成：

聚合优化传递在生产（GCC [8]，LLVM [7]，IBM XL [51]）和研究（R-Stream [49]，ROSE [52]）编译器中可用。在大多数情况下，必须从低级表示中提取聚合抽象，然后在专门的代码生成步骤中进行转换和降级 [13], [14]。这个提取过程并不是保证成功的，可能无法恢复源级别上可用的高级信息 [9]。此外，常见的编译器优化，例如LICM，已知会对此产生干扰 [10]。Polygeist 通过设计维护足够的高级信息，尤其是循环和多维数组结构，以规避这些问题。

源到源的聚合编译器，如Pluto [11] 和PPCG [5]，在C或C++级别上运行。它们缺乏与其他编译器优化的交互，以及对代码的全局视角，这阻碍了例如常量传播和内联等可能改善聚合优化结果的过程。Polygeist位于AST和LLVM IR级别之间，使得更高级别和低级别抽象之间的交互成为可能，而这种交互在其他情况下仅限于编译器指示，即仅仅是优化提示。此外，Polygeist可以依靠MLIR的逐步提升 [53]，以比聚合框架 [54] 更少的努力，针对更高于C代码的抽象。

VII. DISCUSSION

VII. 讨论

A. 限制

- a) 前端：尽管Polygeist技术上可以接受任何有效的C或C++代码，因为它是基于Clang构建的，但仍然存在以下限制。由于高层次MLIR方言中缺乏结构类型，仅支持值类型相同的结构体，或在特定函数内（如fprintf中的FILE）使用的结构体。所有分配内存的函数必须使用Polygeist编译，而不是使用C++编译器，这样才能确保生成的是memref而非指针。
- b) 优化器：优化器的限制源自于所涉及工具的限制。特别是，MLIR的仿射值分类导致了全有或全无的建模，如果任何循环中包含一个非仿射访问或负步长，就会将该循环降级为非仿射。对不是由Polygeist前端生成的代码使用Polygeist的后端运行时，将限制在具有正索引的循环上。最后，MLIR目前尚未提供对非凸集合的广泛支持（通常以并集的形式表示）。MLIR内部正在进行相关工作以解决此类问题。
- c) 实验：尽管我们的实验清楚地展示了在Polygeist中实现的技术的好处——语句拆分和后期（归约）并行化——但仍有不容忽视的效果是由于调度器的差异：Polygeist中的Pluto和Polly中的isl。使用Pluto 5的Polly版本与现代LLVM不兼容，而后者又是利用MLIR所必需的。将isl调度器连接到Polygeist可能会使结果更接近Polly，但仍然无法更直接地进行比较，因为SCoP检测、语句形成和仿射调度之间的相互作用。

B. Opportunities and Future Work

将MLIR与现有的多面体流程连接起来，为编译器优化研究开辟了许多新途径，连接了多面体和传统的基于SSA的编译器转换。这使得多面体调度器可以访问一些重要的分析信息，如别名分析以及精确的数据布局和目标机器描述等有用信息。可以说，这些信息已经被Polly利用，但LLVM IR和仿射循环之间的表示不匹配使得高效利用这些信息变得困难。MLIR以足够高的抽象层级暴露了类似的信息，使其可用于仿射变换。

通过在单个模块中混合不同的抽象，MLIR对整个转换过程提供了更细粒度的控制。Polygeist的扩展可以确保通过直接发出向量指令来实现循环向量化，而不是依赖于常常只是编译器建议的pragma指令。该流程还可以控制更低层次的机制，如预取或发出专用的硬件指令。反之，多面体分析可以确保下游的处理过程，例如，地址计算永远不会导致越界访问，以及其他信息。

在Polygeist的支持下，控制语句粒度的未来工作是必要的。除了影响仿射调度外，该技术还使得轻松的重新物化和局部转置缓冲成为可能，这在GPU上是至关重要的[55]，以及软件流水线化；所有这些都不需要生成已知复杂的C源代码[56]。另一方面，语句数量可能会对编译时间产生影响，因为语句数量是依赖分析和调度算法复杂性边界的重要因素。

C. Alternatives

替代方案

与其让多面体工具解析并生成MLIR，不如直接从MLIR发出C（或C++）代码，并在C源代码上使用基于C的多面体工具，但这种方法降低了编译流程的表达能力。一些MLIR构造，例如并行归约循环，可以直接在多面体模型中表达，而在C中却需要一个复杂且不保证可行的提升步骤。另一些构造，例如预向量化的仿射内存操作，则根本无法在C中表达。Polygeist支持在MLIR到MLIR的流程中透明地处理这些构造，但我们将此类处理的细节留待未来研究。

Polygeist流程也可以与其他多面体格式相连接，特别是与isl相连。我们选择OpenScop作为本研究的对象，因为它得到更多工具的支持。isl使用调度树来表示初始和转化后的程序调度。调度树与嵌套操作的IR模型非常接近，使得转换变得简单：`for` 循环对应于带节点（每个维度一个循环），`if` 条件对应于过滤节点，函数级常量可以包含在上下文节点中。树结构与MLIR区域保持一致。逆向转换可以通过isl的AST生成工具获得。

VIII. CONCLUSION

我们提出了Polygeist，一种将现有C或C++代码导入MLIR的编译工作流程，并允许多面体工具（如Pluto）对MLIR程序进行优化。这使得MLIR能够受益于数十年来在多面体编译方面的研究。我们证明了Polygeist生成的代码在性能上与Clang相当，从而实现了针对MLIR构建的变换与现有多面体框架之间的公正比较。最后，我们展示了Polygeist所带来的优化机会，考虑了两种互补的变换：语句拆分和归约并行化。在这两种情况下，Polygeist的性能均优于最先进的多面体编译器和源到源优化器。

关于如何在硬件或软件配置与我们所使用的不同的系统上运行，这可能需要修改的评论。如预期的那样，命令描述与docker文件的内容相似。虽然docker文件无疑更方便，是设置编译器的好方法，但即便使用Docker，对于评估系统的核心数量的期望也需要类似的更改。

要编译Polygeist，首先需要编译其若干依赖项。我们在基于Ubuntu 20.04的AWS c5.metal实例上运行了实验。我们根据该系统调整了构建说明。虽然许多说明是通用的，与机器或操作系统无关，但某些步骤可能并非如此（我们将在下文中描述它们可能出现的位置）。

```
$ sudo apt update
$ sudo apt install apt-utils
$ sudo apt install tzdata build-essential \
libtool autoconf pkg-config flex bison \
libgmp-dev clang-9 libclang-9-dev texinfo \
cmake ninja-build git texlive-full numactl

# 更改默认编译器以使Pluto正常工作
$ sudo update-alternatives --install \
/usr/bin/llvm-config llvm-config \
/usr/bin/llvm-config-9 100

$ sudo update-alternatives --install \
/usr/bin/FileCheck FileCheck-9 \
/usr/bin/FileCheck 100

$ sudo update-alternatives --install \
/usr/bin/clang clang \
/usr/bin/clang-9 100

$ sudo update-alternatives --install \
/usr/bin/clang++ clang++ \
/usr/bin/clang++-9 100
```

首先，让我们下载一个实用工具库，其中将包含一些编译和基准测试所需的脚本和其他文件：

```
$ cd
$ git clone \
https://github.com/wsmoses/Polygeist-Script\ scripts
```

现在可以按照下面的步骤编译和构建Pluto:

```
$ cd $ git clone \ https://github.com/bondhugula/pluto $ cd pluto/ $ git checkout e5a039096547e0a3d34686295c
$ git submodule init $ git submodule update $ ./autogen.sh $ ./configure $ make -j nproc
```

接下来，可以通过执行以下操作来构建LLVM、MLIR和前端:

从这里开始，我们需要通过复制脚本库中的版本并替换我们刚刚构建的版本来修改omp.h。

```
$ cd $ export OMP_FILE= find \ $HOME/mlir-clang/build -iname omp.h
```

```
$ cp $HOME/scripts/omp.h $OMP_FILE
```

现在让我们构建MLIR多面体分析，以及它所需的特定版本的LLVM。我们将开始下载所需的代码并构建其依赖项。

```
$ cd $ git clone --recursive \ https://github.com/kumasento/polymer -b pact
```

```
$ cd polymer/ $ cd llvm/ $ mkdir build $ cd build/
```

```
$ cmake ..llvm \ -DLLVM_ENABLE_PROJECTS="llvm;clang;mlir" \
-DLLVM_TARGETS_TO_BUILD="host" \ -DLLVM_ENABLE_ASSERTIONS=ON \
-DCMAKE_BUILD_TYPE=Release \ -DLLVM_INSTALL_UTILS=ON \ -G Ninja
```

```
$ ninja -j nproc
```

```
$ ninja check-mlir
```

现在我们可以构建MLIR多面体分析并导出相应的构建产物。

```
$ cd ~/polymer $ mkdir build $ cd build $ export BUILD=$PWD/..llvm/build
```

```
$ cmake .. \ -DCMAKE_BUILD_TYPE=DEBUG \ -DMLIR_DIR=$BUILD/lib/cmake/mlir \
-DLLVM_DIR=$BUILD/lib/cmake/llvm \ -DLLVM_ENABLE_ASSERTIONS=ON \
-DLLVM_EXTERNAL_LIT=$BUILD/bin/llvm-lit \ -G Ninja
```

```
$ ninja -j nproc
```

```
$ export LD_LIBRARY_PATH=\ $pwd /pluto/lib:$LD_LIBRARY_PATH
```

```
$ ninja check-polymer
```

最后，我们准备开始基准测试。我们首先运行一个脚本，禁用涡轮提升和超线程，并关闭机器上仍然不必要的服务。该脚本特定于AWS实例上的核心数量（所有核心中除第一个插槽上的非超线程核心外均被禁用），以及使用的镜像（镜像上仍存在的所有不必要的服务被禁用），因此如果打算在其他机器上使用，可能需要进行修改。

```
$ cd ~/scripts/ $ sudo bash ./hyper.sh
```

我们现在可以运行基准测试脚本。该脚本本身对核心和布局有一些假设（例如，设置 taskset -c 1-8 numactl -i all）。如果使用不同的机器，可能需要根据实际情况调整这些设置。
\$ cd ~/scripts/ \$ cd polybench-c-4.2.1-beta/ \$./run.sh #
输出通过标准输出生成。该脚本的输出将包含每次试验的运行时间，描述所使用的编译设置以及运行了哪个基准测试。

本文件中包含的观点和结论是作者的观点，不应被解读为代表美国空军或美国政府的官方政策，无论是明示还是暗示。美国政府被授权为了政府目的复印和分发该文件的印刷本，尽管这里有任何版权说明。

APPENDIX

在本附录中，我们描述了如何构建Polygeist并评估其在Polybench基准测试套件上的性能（以及基准编译器）。我们提供了两种艺术品评估机制：一个Docker容器以及逐步命令描述的安装过程，连同

GPT-Academic Report

Title:

Polygeist: Raising C to Polyhedral MLIR

Abstract:

We present Polygeist, a new compilation flow that connects the MLIR compiler infrastructure to cutting edge polyhedral optimization tools. It consists of a C and C++ frontend capable of converting a broad range of existing codes into MLIR suitable for polyhedral transformation and a bi-directional conversion between MLIR and OpenScop exchange format. The Polygeist/MLIR intermediate representation featuring high-level (affine) loop constructs and n-D arrays embedded into a single static assignment (SSA) substrate enables an unprecedented combination of SSA-based and polyhedral optimizations. We illustrate this by proposing and implementing two extra transformations: statement splitting and reduction parallelization. Our evaluation demonstrates that Polygeist outperforms on average both an LLVM IR-level optimizer (Polly) and a source-to-source state-of-the-art polyhedral compiler (Pluto) when exercised on the Polybench/C benchmark suite in sequential (2.53x vs 1.41x, 2.34x) and parallel mode (9.47x vs 3.26x, 7.54x) thanks to the new representation and transformations.

Meta Translation

标题: Polygeist: 将C提升到多面体MLIR

摘要: 我们提出了Polygeist, 这是一种新的编译流程, 旨在将MLIR编译基础设施与前沿的多面体优化工具连接起来。它包括一个C和C++前端, 能够将广泛的现有代码转换为适合多面体变换的MLIR, 并实现MLIR与OpenScop交换格式之间的双向转换。Polygeist/MLIR中间表示具备高层(仿射)循环构造和嵌入到单一静态赋值(SSA)基础结构中的n维数组, 能够实现前所未有的SSA基础和多面体优化的结合。我们通过提出和实现两个额外变换: 语句分割和归约并行化, 来证明这一点。我们的评估表明, Polygeist在Polybench/C基准套件的顺序模式(2.53x对比1.41x, 2.34x)和并行模式(9.47x对比3.26x, 7.54x)上的表现平均优于LLVM IR级别优化器(Polly)和先进的源到源多面体编译器(Pluto), 这得益于新的表示和变换。

I. INTRODUCTION

Improving the efficiency of computation has always been one of the prime goals of computing. Program performance can be improved significantly by reaping the benefits of parallelism, temporal and spatial locality, and other performance sources. Relevant program transformations are particularly tedious and challenging when targeting modern multicore CPUs and GPUs with deep memory hierarchies and parallelism, and are often performed automatically by optimizing compilers. The polyhedral model enables precise analyses and a relatively easy specification of transformations (loop restructuring, automatic parallelization, etc.) that take advantage of hardware performance sources. As a result, there is growing evidence that the polyhedral model is one of the best frameworks for efficient transformation of compute-intensive programs [1], [2], [3], and for programming accelerator architectures [4], [5], [6]. Consequently, the compiler community has focused on building tools that identify and optimize parts of the program that can be represented within the polyhedral model (commonly referred to as static-control parts, or SCoP's). Such tools tend to fall into two categories. Compiler-based tools like

Polly [7] and Graphite [8] detect and transform SCoPs in compiler intermediate representations (IRs). While this offers seamless integration with rest of the compiler, the lack of high-level structure and information hinders the tools' ability to perform analyses and transformations. This structure needs to be recovered from optimized IR, often Fig. 1. The Polygeist compilation flow consists of 4 stages. The frontend traverses Clang AST to emit MLIR SCF dialect (Section III-A), which is raised to the Affine dialect and pre-optimized (Section III-B). The IR is then processed by a polyhedral scheduler (Sections III-C, III-D) before postoptimization and parallelization (Section III-E). Finally, it is translated to LLVM IR for further optimization and binary generation by LLVM. imperfectly or at a significant cost [9]. Moreover, common compiler optimizations such as LICM may interfere with the process [10]. Finally, low-level IRs often lack constructs for, e.g., parallelism or reductions, produced by the transformation, which makes the flow more complex. Source-to-source compilers such as Pluto [11], POCC [12] and PPCG [5] operate directly on C or C++ code. While this can effectively leverage the high-level information from source code, the effectiveness of such tools is often reduced by the lack of enabling optimizations such as those converting hazardous memory loads into single-assignment virtual registers. Furthermore, the transformation results must be expressed in C, which is known to be complex [13], [14] and is also missing constructs for, e.g., reduction loops or register values not backed by memory storage. This paper proposes and evaluates the benefits of a polyhedral compilation flow, Polygeist (Figure 1), that can leverage both the high-level structure available in source code and the fine-grained control of compiler optimization provided by lowlevel IRs. It builds on the recent MLIR compiler infrastructure that allows the interplay of multiple abstraction levels within the same representation, during the same transformations [15]. Intermixable MLIR abstractions, or dialects, include highlevel constructs such as loops, parallel and reduction patterns; low-level representations fully covering LLVM IR [16]; and a polyhedral-inspired representation featuring loops and memory accesses annotated with affine expressions. Moreover, by combining the best of source-level and IR-level tools in an end-to-end polyhedral flow, Polygeist preserves high-level information and leverages them to perform new or improved %result = "dialect.operation"(%operand, %operand) {attribute = #dialect<"value">} ({ // Inside a nested region.
 ^basic_block(%block_argument: !dialect.type): "another.operation"() : () -> () } : (!dialect.type) ->
 !dialect.result_type Fig. 2. Generic MLIR syntax for an operation with two operands, one result, one attribute and a single-block region. optimizations, such as statement splitting and loop-carried value detection, on a lower-level abstraction as well as to influence downstream optimizations. We make the following contributions: • a C and C++ frontend for MLIR that preserves high-level loop structure from the original source code; • an end-to-end flow with raising to and lowering from the polyhedral model, leveraging our abstraction to perform more optimizations than both source-and IR-level tools, including reduction parallelization; • an exploration of new transformation opportunities created by Polygeist, in particular, statement splitting; • and an end-to-end comparison between Polygeist and state-of-the-art source-and IR-based tools (Pluto [11] and Polly [14]) along with optimization case studies.

I. 引言

提高计算效率一直是计算领域的主要目标之一。通过利用并行性、时间和空间局部性以及其他性能来源，可以显著提高程序性能。当针对现代多核 CPU 和 GPU 进行相关程序变换时，这些变换尤其繁琐且具有挑战性，因为现代架构具有深度内存层次和并行性，通常由优化编译器自动执行。

多面体模型能够实现精确分析和相对简单的变换规范（如循环重构、自动并行化等），以利用硬件性能来源。因此，有越来越多的证据表明，多面体模型是高效转化计算密集型程序 [1]、[2]、[3] 和编程加速器架构 [4]、[5]、[6] 的最佳框架之一。因此，编译器社区专注于构建工具，以识别和优化可以在多面体模型中表示的程序部分（通常称为静态控制部分或 SCoP）。这些工具往往分为两类。

基于编译器的工具，如 Polly [7] 和 Graphite [8]，在编译器中间表示（IR）中检测和转换 SCoP。尽管这与编译器的其余部分无缝集成，但缺乏高层次结构和信息限制了这些工具进行分析和变换的能力。这种结构通常需要通过优化后的 IR 恢复，往往是以不完全或高成本的方式实现 [9]。此外，诸如 LICM 的常见

编译器优化可能会干扰这一过程 [10]。最后，低级 IR 通常缺乏由变换生成的并行性或归约的构造，从而使流程更加复杂。

源到源编译器，如 Pluto [11]、POCC [12] 和 PPCG [5]，直接在 C 或 C++ 代码上操作。虽然这可以有效地利用源代码中的高层信息，但这些工具的有效性往往因缺乏启用优化而降低，例如将危险的内存加载转换为单赋值虚拟寄存器。此外，变换结果必须用 C 表达，而 C 语言本身的复杂性众所周知 [13]、[14]，而且也缺乏例如归约循环或未被内存存储支持的寄存器值的构造。

本文提出并评估了一种多面体编译流程 Polygeist（图 1）的好处，该流程可以利用源代码中可用的高层结构以及由低级 IR 提供的编译器优化的细粒度控制。它构建在最近的 MLIR 编译器基础设施之上，该基础设施允许在同一表示中、在同一变换过程中混合多个抽象层次 [15]。可互换的 MLIR 抽象或方言包括高层构造，如循环、并行和归约模式；完整覆盖 LLVM IR 的低层表示 [16]；以及一种受到多面体启发的表示，具有带有仿射表达式注释的循环和内存访问。此外，通过将源级和 IR 级工具的优势结合在一个端到端的多面体流程中，Polygeist 保留了高层信息并利用这些信息在较低级别抽象上执行新的或改进的优化，如语句拆分和循环携带值检测，同时影响后续优化。

我们的贡献如下：

- 一个 C 和 C++ 的 MLIR 前端，保留原始源代码中的高层循环结构；
- 一个端到端的流程，实现向多面体模型的提升和降低，利用我们的抽象执行比源级和 IR 级工具更多的优化，包括归约并行化；
- 探索 Polygeist 创建的新变换机会，尤其是语句拆分；
- 以及 Polygeist 与最先进的基于源和 IR 的工具（Pluto [11] 和 Polly [14]）之间的端到端比较，以及优化案例研究。

A. Overview

MLIR is an optimizing compiler infrastructure inspired by LLVM [16] with a focus on extensibility and modularity [15]. Its main novelty is the IR supporting a fully extensible set of instructions (called operations) and types. Practically, MLIR combines SSA with nested regions, allowing one to express a range of concepts as first-class operations including machine instructions such as floating-point addition, structured control flow such as loops, hardware circuitry [17], and large machine learning graphs. Operations define the runtime semantics of a program and process immutable values. Compile-time information about values is expressed in types, and information about operations is expressed in attributes. Operations can have attached regions, which in turn contain (basic) blocks of further operations. The generic syntax, accepted by all operations, illustrates the structure of MLIR in Figure 2. Additionally, MLIR supports user-defined custom syntax. Attributes, operations, and types are organized in dialects, which can be thought of as modular libraries. MLIR provides a handful of dialects that define common operations such as modules, functions, loops, memory or arithmetic instructions, and ubiquitous types such as integers and floats. We discuss the dialects relevant to Polygeist in the following sections.

A. 概述

MLIR 是一种优化编译器基础设施，受到 LLVM 的启发，重点关注扩展性和模块化。其主要创新在于支持完全可扩展的一组指令（称为操作）和类型的中间表示（IR）。实际上，MLIR 将静态单赋值（SSA）与嵌套区域相结合，使得能够将一系列概念表示为一类一流的操作，包括机器指令，例如浮点加法、结构化控制流，例如循环、硬件电路，以及大型机器学习图。操作定义了程序的运行时语义，并处理不可变值。有关值的编译时信息用类型表示，而有关操作的信息用属性表示。操作可以附带区域，而这些区域内部又包含其他操作的（基本）块。所有操作接受的通用语法在图 2 中展示了 MLIR 的结构。此外，MLIR 支持用户自定义的自定义语法。

属性、操作和类型被组织在方言中，可以视为模块化库。MLIR 提供了少量方言，以定义常见操作，如模块、函数、循环、内存或算术指令，以及像整数和浮点数这样的通用类型。我们将在接下来的章节中讨论与 Polygeist 相关的方言。

B. Affine and MemRef Dialects

The Affine dialect [18] aims at representing SCoP's with explicit polyhedral-friendly loop and conditional constructs. The core of its representation is the following classification of value categories:

- **Symbols**—integer values that are known to be loopinvariant but unknown at compile-time, also referred to as program parameters in polyhedral literature, typically array dimensions or function arguments. In MLIR, symbols are values defined in the top-level region of an operation with "affine scope" semantics, e.g., functions; or array dimensions, constants, and affine map (see below) application results regardless of their definition point.
- **Dimensions**—are an extension of symbols that also accepts induction variables of affine loops.
- **Non-affine**—any other values. Symbols and dimensions have index type, which is a platform-specific integer that fits a pointer (`intptr_t` in C). MLIR provides two attributes relevant for the Affine dialect:
- **Affine maps** are multi-dimensional (quasi-)linear functions that map a list of dimension and symbol arguments to a list of results. For example, $(d_0, d_1, d_2, s_0) \rightarrow (d_0 + d_1, s_0 \cdot d_2)$ is a two-dimensional quasi-affine map, which can be expressed in MLIR as `affine_map<(d0,d1,d2)[s0]> -> (d0+d1, s0 * d2)`. Dimensions (in parentheses on the left) and symbols (in brackets on the left) are separated to allow quasi-linear expressions: symbols are treated as constants, which can therefore be multiplied with dimensions, whereas a product of two dimensions is invalid.
- **Integer sets** are collections of integer tuples constrained by conjunctions of (quasi-)linear expressions. For example, a "triangular" set $\{(d_0, d_1) : 0 \leq d_0 < s_0 \wedge 0 \leq d_1 \leq d_0\}$ is represented as `affine_set<(d0,d1)[s0]> : (d0 >= 0, s0-d0-1 >= 0, d1 >= 0, d0-d1 >= 0)`.

The Affine dialect makes use of the concepts above to define a set of operations. An `affine.for` is a "for" loop with loop-invariant lower and upper bounds expressed as affine maps with a constant step. An `affine.parallel` is a "multifor" loop nest, iterations of which may be executed concurrently. Both kinds of loops support reductions via loopcarried values as well as max(min) expression lower(upper) bounds. An `affine.if` is a conditional construct, with an optional else region, and a condition defined as inclusion of the given values into an integer set. Finally, `affine.load` and `affine.store` express memory accesses where the address computation is expressed as an affine map. A core MLIR type-`memref`, which stands for memory reference—and the corresponding `memref` dialect are also featured in Figure 3. The `memref` type describes a structured multi-index pointer into memory, e.g., `memref<?xf32>` denotes a 1-d array of floating-point elements; and the `memref` dialect provides memory and type manipulation operations, e.g., `memref.dim` retrieves the dimensionality of a `memref` object. `memref` does not allow internal aliasing, i.e., different subscripts always point to different addresses. This effectively defines away the delinearization problem that hinders the application of polyhedral techniques at the LLVM IR level [9]. Throughout this paper, we only consider `memrefs` with the default layout that corresponds to contiguous row-major storage compatible with C ABI (Application Binary Interface). In practice, `memrefs` support arbitrary layouts expressible as affine maps, but these are not necessary in Polygeist context.

B. 仿射和内存引用方言

仿射方言 [18] 旨在表示具有显式适合多面体的循环和条件构造的静态计算程序（SCoP）。其表示的核心是以下值类别的分类：

- **符号**——已知为循环不变但在编译时未知的整数值，在多面体文献中也称为程序参数，通常是数组维度或函数参数。在 MLIR 中，符号是在具有“仿射范围”语义的操作的顶级区域中定义的值，例如函数；或者数组维度、常量和仿射映射（见下文）应用结果，无论它们的定义点在哪里。
- **维度**——是符号的扩展，接受仿射循环的归纳变量。
- **非仿射**——任何其他值。符号和维度具有索引类型，即平台特定的整数，适合指针（在 C 中为 `intptr_t`）。

MLIR 提供了两个与仿射方言相关的属性：

- **仿射映射**是多维的（准）线性函数，它将一组维度和符号参数映射到一组结果。例如，

$$((d_0, d_1, d_2, s_0) \rightarrow (d_0 + d_1, s_0 \cdot d_2))$$

是一个二维准仿射映射，可以在 MLIR 中表示为 `affine_map<(d0, d1, d2)[s0] -> (d0+d1, s0 * d2)>`。维度（左侧的括号内）和符号（左侧的方括号内）被分隔开，以允许准线性表达式：符号被视为常量，因此可以与维度相乘，而两个维度的乘积是无效的。

- 整数集是被（准）线性表达式的合取限制的整数元组集合。例如，“三角形”集合 `((d_0, d_1) : 0 \leq d_0 < s_0 \wedge 0 \leq d_1 \leq d_0)` 表示为 `affine_set<(d0, d1)[s0]: (d0 \geq 0, s0-d0-1 \geq 0, d1 \geq 0, d0-d1 \geq 0)>`。

仿射方言利用上述概念定义了一组操作。`affine.for` 是一个“for”循环，其循环不变的下界和上界表示为带有常量步长的仿射映射。`affine.parallel` 是一个“多重for”循环嵌套，其迭代可以并发执行。这两种循环都支持通过循环传递值进行归约，以及 `max(min)` 表达式的下（上）界。`affine.if` 是一个条件构造，带有一个可选的 `else` 区域，条件被定义为给定值包含在一个整数集中。最后，`affine.load` 和 `affine.store` 表示内存访问，其中地址计算表示为仿射映射。核心的 MLIR 类型——内存引用 (`memref`)，以及相应的 `memref` 方言也在图 3 中展示。`memref` 类型描述了指向内存的结构化多索引指针，例如，`memref<?xf32>` 表示一个浮点元素的一维数组；而 `memref` 方言提供了内存和类型操作，例如 `memref.dim` 检索 `memref` 对象的维度。`memref` 不允许内部别名，即不同的下标始终指向不同的地址。这有效地解决了限制多面体技术在 LLVM IR 级别应用的去线性化问题 [9]。在本文中，我们只考虑与兼容 C ABI（应用程序二进制接口）的连续行主存储对应的默认布局的 `memrefs`。实际上，`memrefs` 支持可作为仿射映射表达的任意布局，但在 Polygeist 的上下文中，这些并不是必需的。

C. Other Relevant Core Dialects

MLIR 提供了几十种方言。出于那些，只有少数几种与我们的讨论相关：

- The Structured Control Flow (scf) dialect defines the control flow operations such as loops and conditionals that are not constrained by affine categorization rules. For example, the scf.for loop accepts any integer value as loop bounds, which are not necessarily affine expressions.
- The Standard (std) dialect contains common operations such as integer and float arithmetic, which is used as a common lowering point from higher-level dialects before fanning out into multiple target dialects and can be seen as a generalization of LLVM IR [16].
- The LLVM dialect directly maps from LLVM IR instructions and types to MLIR, primarily to simplify the translation between them.
- The OpenMP dialect provides a dialect-and platformagnostic representation of OpenMP directives such as "parallel" and "workshare loop", which can be used to transform OpenMP constructs or emit LLVM IR that interacts with the OpenMP runtime.
- The Math dialect groups together mathematical operations on integer and floating type beyond simple arithmetic, e.g., `math.pow` or `math.sqrt`.

MLIR 提供了几十种方言。在这些方言中，只有少数几种与我们的讨论相关：

- 结构化控制流 (scf) 方言定义了控制流操作，例如循环和条件语句，这些操作不受仿射分类规则的限制。例如，`scf.for` 循环可以接受任何整数值作为循环边界，这些边界不一定是仿射表达式。
- 标准 (std) 方言包含常见的操作，例如整数和浮点算术，这被用作从更高级别方言到多个目标方言的共同降低点，可以视为 LLVM IR 的概括 [16]。
- LLVM 方言直接将 LLVM IR 指令和类型映射到 MLIR，主要是为了简化它们之间的转换。
- OpenMP 方言提供了一种与方言和平台无关的 OpenMP 指令表示，例如“parallel”和“workshare loop”，可用于转换 OpenMP 构造或生成与 OpenMP 运行时交互的 LLVM IR。
- 数学方言将整数和浮点类型上的数学操作（超出简单算术）归为一组，例如，`math.pow` 或 `math.sqrt`。

III. AN (AFFINE) MLIR COMPILATION PIPELINE

The Polygeist pipeline consists of 4 components (Figure 1): 4). This allows Polygeist to preserve more of the structure available within the original program (e.g., multi-dimensional arrays) and enables interaction with MLIR's high-level memory operations. This diverges from the C ABI for any functions with pointer arguments and wouldn't interface correctly with C functions. Polygeist addresses this by providing an attribute for function arguments and allocations to use a C-compatible pointer type rather than memref, applied by default to external functions such as strcmp and scanf. When calling a pointer-ABI function with a memref-ABI argument, Polygeist generates wrapper code that recovers the C ABI-compatible pointer from memref and ensures the correct result. Figure 5 shows an example demonstrating how the Polygeist and C ABI may interact for a small program. When allocating and deallocating memory, this difference in ABI becomes significant. This is because allocating several bytes of an array with malloc then casting to a memref will not result in legal code (as memref itself may not be implemented with a raw pointer). Thus, Polygeist identifies calls to allocation and deallocation functions and replaces them with legal equivalents for memref. Functions and global variables are emitted using the same name used by the C or C++ ABI. This ensures that all external values are loaded correctly, and multi-versioned functions (such as those generated by C++ templates or overloading) have distinct names and definitions. c)

Instruction Generation: For most instructions, Polygeist directly emits an MLIR operation corresponding to the equivalent C operation (addi for integer add, call for function call, etc.). For some special instructions such as a call to pow, Polygeist chooses to emit a specific MLIR operation in the Math dialect, instead of a call to an external function (defined in libm). This permits such instructions to be better analyzed and optimized within MLIR. Operations that involve memory or pointer arithmetic require additional handling. MLIR does not have a generic pointer arithmetic instruction; instead, it requires that load and store operations contain all of the indices being looked up. This presents issues for operations that perform pointer arithmetic. To remedy this, we introduce a temporary subindex operation for memref's keeps track of the additional address offsets. A subsequent optimization pass within Polygeist, forwards the offsets in a subindex to any load or store which uses them. d)

Local Variables: Local variables are handled by allocating a memref on stack at the top of a function. This permits the desired semantics of C or C++ to be implemented with relative ease. However, as many local variables and arguments contain memref types, this immediately results in a memref of a memref-a hindrance for most MLIR optimizations as it is illegal outside of Polygeist. As a remedy, we implement a heavyweight memory-to-register (mem2reg) transformation pass that eliminates unnecessary loads, stores, and allocations within MLIR constructs. Empirically this eliminates all memrefs of memref in the Polybench suite.

III. 一个（仿射）MLIR编译流程

Polygeist编译流程由4个组件组成（见图1）。这使得Polygeist能够保留原始程序中可用的更多结构（例如，多维数组），并能够与MLIR的高级内存操作进行交互。这与任何具有指针参数的C ABI不同，并且与C函数的接口不一致。Polygeist通过为函数参数和分配提供一个属性，使其使用与C兼容的指针类型，而不是memref，默认适用于外部函数，如strcmp和scanf。当使用memref ABI参数调用指针 ABI函数时，Polygeist生成包装代码，从memref中恢复与C ABI兼容的指针，并确保结果正确。图5展示了一个示例，演示了Polygeist与C ABI在一个小程序中的互动。

在分配和释放内存时，这种ABI的差异显得尤为重要。这是因为使用malloc分配数组的几个字节后，再将其转换为memref将不会产生合法代码（因为memref本身可能不是用原始指针实现的）。因此，Polygeist识别对分配和释放函数的调用，并将其替换为memref的合法等效函数。

函数和全局变量使用C或C++ ABI中使用的相同名称发出。这确保所有外部值都被正确加载，并且多版本函数（如C++模板或重载生成的函数）拥有不同的名称和定义。

c) 指令生成：对于大多数指令，Polygeist直接发出与等效C操作相对应的MLIR操作（如整数加法的addi，函数调用的call等）。对于某些特殊指令，如对pow的调用，Polygeist选择发出Math方言中特定的MLIR操作，而不是调用外部函数（在libm中定义）。这使得这些指令能够在MLIR中更好地进行分析和优化。

涉及内存或指针算术的操作需要额外处理。MLIR没有通用的指针算术指令；相反，它要求load和store操作包含所有被查找的索引。这为执行指针算术的操作带来了问题。为了解决这个问题，我们引入了一个临时子索引操作，用于memref，以跟踪附加的地址偏移。在Polygeist中的后续优化过程中，将子索引中的偏移量转发给任何使用它们的load或store。

d) 局部变量：局部变量通过在函数顶部分配一个memref在栈上进行处理。这使得C或C++的所需语义能够相对容易地实现。然而，由于许多局部变量和参数包含memref类型，这立即导致了memref的memref这对大多数MLIR优化来说是一个障碍，因为在Polygeist之外这是不合法的。为了解决这个问题，我们实现了一个重量级的内存到寄存器(mem2reg)转换过程，该过程消除了MLIR构造中的不必要的加载、存储和分配。根据经验，这消除了Polybench套件中所有memref的memref。

B. Raising to Affine

The translation from C or C++ to MLIR directly preserves high-level information about loop structure and n-D arrays, but does not generate other Affine operations. Polygeist subsequently raises memory, conditional, and looping operations into their Affine dialect counterparts if it can prove them to be legal affine operations. If the corresponding frontend code was enclosed within `#pragma scop`, Polygeist assumes it is always legal to raise all operations within that region without additional checks. Any operations which are not proven or assumed to be affine remain untouched. We perform simplifications on affine maps to remove loops with zero or one iteration and drop branches of a conditional with a condition known at compile time.

将C或C++直接翻译为MLIR能够保留关于循环结构和n维数组的高级信息，但不会生成其他的仿射操作。之后，Polygeist将内存、条件和循环操作提升至其仿射方言对应物，如果它能够证明这些操作是合法的仿射操作。如果相应的前端代码被包裹在 `#pragma scop` 中，Polygeist假设在该区域内提升所有操作是始终合法的，且无需额外检查。对于任何未被证明或假设为仿射的操作，则保持不变。我们对仿射映射进行简化，以去除具有零次或一次迭代的循环，并删除在编译时已知条件的条件语句的分支。

a) Memory operations and loop bounds:

To convert an operation, Polygeist replaces its bound and subscript operands with identity affine maps (`affine_map<() [s0]->(s0)>[%bound]`). It then folds the operations computing the map operands, e.g., addi, muli, into the map itself. Values that are transitively derived from loop induction variables become map dimensions and other values become symbols. For example, `affine_map<() [s0]->(s0)> [%bound]` with `%bound = addi %N, %i`, where `%i` is an induction variable, is folded into `affine_map<(d0)[s0] ->(s0 + d0)>(%i)[%N]`. The process terminates when no operations can be folded or when Affine value categorization rules are satisfied. b)

Conditionals: Conditional operations are emitted by the frontend for two input code patterns: if conditions and ternary expressions. The condition is transformed by introducing an integer set and by folding the operands into it similarly to the affine maps, with in addition and operations separating set constraints and not operations inverting them (affine.if only accepts ≥ 0 and $= 0$ constraints). Polygeist processes nested conditionals with C-style shortcircuit semantics, in which the subsequent conditions are checked within the body of the preceding conditionals, by hoisting conditions outside the outermost conditional when legal and replacing them with a boolean operation or a select. This is always legal within `#pragma scop`. Conditionals emitted for ternary expressions often involve memory loads in their regions, which prevent hoisting due to side effects. We reuse our

mem2reg pass to replace those to equivalent earlier loads when possible to enable hoisting. Empirically, this is sufficient to process all ternary expressions in the Polybench/C suite [19]. Otherwise, ternary expressions would need to be packed into a single statement by the downstream polyhedral pass.

a) 内存操作和循环边界：为了转换一个操作，Polygeist 用恒等仿射映射 ($\text{affine_map} < () [s0] \rightarrow (s0) > [%\text{bound}]$) 替换其边界和下标操作数。它然后将计算映射操作数的操作，例如加法 (addi)、乘法 (muli)，折叠到映射本身中。从循环归纳变量传递得出的值变成了映射维度，其他值则变成符号。例如， $\text{affine_map} < () [s0] \rightarrow (s0) > [%\text{bound}]$ 在 $\%\text{bound} = \text{addi } \%N, \%i$ 的情况下，其中 $\%i$ 是一个归纳变量，被折叠为 $\text{affine_map} < (d0) [s0] \rightarrow (s0 + d0) > (%i) [%N]$ 。该过程在无法再折叠操作或满足仿射值分类规则时终止。

b) 条件语句：前端为两种输入代码模式（if 条件和三元表达式）生成条件操作。条件通过引入整数集合并将操作数折叠到其中进行转换，类似于仿射映射，此外，操作之间的加法和运算分隔集合约束，而非运算则对其进行反转（ affine.if 仅接受 ≥ 0 和 $= 0$ 的约束）。Polygeist 处理嵌套条件时遵循 C 风格的短路语义，其中后续条件在前面条件的主体内进行检查，当法律允许时，将条件提升到最外层条件之外，并用布尔操作或选择操作替换它们。在 #pragma scop 内，这总是合法的。

为三元表达式生成的条件语句通常涉及其区域内的内存加载，由于副作用，导致无法提升。我们重用 mem2reg 通道，在可能的情况下将其替换为等效的早期加载，以实现提升。从经验上看，这足以处理 Polybench/C 套件中的所有三元表达式 [19]。否则，三元表达式将需要被下游多面体通道打包成单个语句。

C. Connecting MLIR to Polyhedral Tools Part-1

Regions of the input program expressed using MLIR Affine dialect are amenable to the polyhedral model. Existing tools, however, cannot directly consume MLIR. We chose to implement a bi-directional conversion to and from OpenScop [20], an exchange format readily consumable by numerous polyhedral tools, including Pluto [11], and further convertible to isl [21] representation. This allows Polygeist to seamlessly connect with tools created in polyhedral compilation research without having to amend those tools to support MLIR. Most polyhedral tools are designed to operate on C or FORTRAN inputs build around statements, which do not have a direct equivalent in MLIR. Therefore, we design a mechanism to create statement-like structure from chains of MLIR void setArray(int N, double val, double array) {} ... int main(int argc, char * argv) { ... cmp = strcmp(str1, str2) ... double array [10]; setArray(10, 42.0, array) } \Downarrow func @setArray(%N: i32, %val: f64, %array: memref<%xf64>) { %0 = index_cast %N : i32 to index affine.for %i = 0 to %0 { affine.store %val, %array[%i] : memref<%xf64>} return } func @main(%argc: i32, %argv: !llvm.ptr) -> i32 { ... %cmp = llvm.call @strcmp(%str1, %str2) : (!llvm.ptr, !llvm.ptr) -> !llvm.i32 ... %array = memref.alloca() : memref<10xf64> %arraycst = memref.cast %array : memref<10xf64> to memref<%xf64> %val = constant 42.0 : f64 call @setArray(%N, %val, %arraycst) : (i32, f64, memref<%xf64>) -> () } Fig. 5. Example demonstrating Polygeist ABI. For functions expected to be compiled with Polygeist such as setArray, pointer arguments are replaced with memrefs. For functions that require external calling conventions (such as main/strcmp), Polygeist falls back to emitting llvm.ptr and generates conversion code. operations. We further demonstrate that this gives Polygeist an ability to favorably affect the behavior of the polyhedral scheduler by controlling statement granularity (Section III-D).

输入程序中使用 MLIR Affine 方言表达的区域适用于多面体模型。然而，现有工具无法直接使用 MLIR。因此，我们选择实现与 OpenScop [20] 的双向转换，这是一种可被众多面体工具直接使用的交换格式，包括 Pluto [11]，并且进一步可转换为 isl [21] 表示。这使得 Polygeist 能够与在多面体编译研究中创建的工具无缝连接，而无需修改这些工具以支持 MLIR。

大多数多面体工具设计用于处理以语句为基础的 C 或 FORTRAN 输入，而它们在 MLIR 中并没有直接对应的等价物。因此，我们设计了一种机制，从一系列 MLIR 操作创建类似语句的结构。例如：

```

void setArray(int N, double val, double * array) {...}
int main(int argc, char ** argv) { ...
    cmp = strcmp(str1, str2) ...
    double array [10];
    setArray(10, 42.0, array) }

```

可转换为：

```

func @setArray(%N: i32, %val: f64, %array: memref<?xf64>) {
    %0 = index_cast %N : i32 to index
    affine.for %i = 0 to %0 {
        affine.store %val, %array[%i] : memref<?xf64>
    }
    return
}

func @main(%argc: i32, %argv: !llvm.ptr<ptr<i8>>) -> i32 {
    ...
    %cmp = llvm.call @strcmp(%str1, %str2) :
    (!llvm.ptr<i8>, !llvm.ptr<i8>) -> !llvm.i32
    ...
    %array = memrefalloca() : memref<10xf64>
    %arraycst = memref.cast %array : memref<10xf64> to memref<?xf64>
    %val = constant 42.0 : f64
    call @setArray(%N, %val, %arraycst) :
    (i32, f64, memref<?xf64>) -> ()
}

```

图 5. 示例演示了 Polygeist 的 ABI。对于预期要由 Polygeist 编译的函数（如 setArray），指针参数被替换为 memref。对于需要外部调用约定（如 main(strcmp）的函数，Polygeist 退回到发出 llvm.ptr 并生成转换代码的方式。

我们进一步证明，这使得 Polygeist 能够通过控制语句粒度，积极影响多面体调度器的行为（详见第 III-D 节）。

C. Connecting MLIR to Polyhedral Tools Part-2

a) Simple Statement Formation: Observing that C statements amenable to the polyhedral model are (mostly) variable assignments, we can derive a mechanism to identify statements from chains of MLIR operations. A store into memory is the last operation of the statement. The backward slice of this operation, i.e., the operations transitively computing its operands, belong to the statement. The slice extension stops at operations producing a value categorized as affine dimension or symbol, directly usable in affine expressions. Such values are loop induction variables or loop-invariant constants. Some operations may end up in multiple statements if the value is used more than once. However, we need the mapping between operations and statements to be bidirectional in order to emit MLIR after the scheduler has restructured the program without considering SSA value visibility rules. If an operation with multiple uses is side effect free, Polygeist simply duplicates it. For operations whose duplication is illegal, Polygeist stores their results in stack-allocated memref's and replaces all further uses with

memory loads. Figure 6 illustrates the transformation for value %0 used in operation %20. This creates a new statement. b) Region-Spanning Dependencies: In some cases, a statement may consist of MLIR operations across different (nested) loops, e.g., a load from memory into an SSA register happens in an outer loop while it is used in inner loops. The location of such a statement in the loop hierarchy is unclear. More importantly, it cannot be communicated to the polyhedral scheduler. Polygeist resolves this by storing the value in a stack-allocated memref in the defining region and loading it back in the user regions. Figure 6 illustrates this transformation for value %0 used in operation %10. Similarly to the basic case, this creates a new statement in the outer loop that can be scheduled independently. This approach can be seen as a reg2mem conversion, the inverse of mem2reg performed in the frontend. It only applies to a subset of values, and may be undone after polyhedral scheduling has completed. Furthermore, to decrease the number of dependencies and memory footprint, Polygeist performs a simple value analysis and avoids creating stack-allocated buffers if the same value is already available in another memory location and can be read from there.

a) 简单语句形成：观察到符合多面体模型的C语句主要是变量赋值，我们可以推导出一种机制，从MLIR操作链中识别语句。存储到内存中的操作是该语句的最后一个操作。该操作的逆向切片，即传递计算其操作数的操作，属于该语句。切片扩展在生成被分类为仿射维度或符号的值的操作处停止，这些值可以直接用于仿射表达式。此类值包括循环归纳变量或循环不变常量。一些操作可能出现在多个语句中，如果该值被多次使用。然而，我们需要操作与语句之间的映射是双向的，以便在调度器重新结构化程序后发出MLIR，而不考虑SSA值的可见性规则。如果一个多次使用的操作没有副作用，Polygeist会简单地复制它。对于重复操作的复制不合法的情况，Polygeist将其结果存储在堆栈分配的内存引用中，并将所有后续使用替换为内存加载。图6展示了在操作%20中使用的值%0的转换。这创建了一个新的语句。

b) 跨区域依赖关系：在某些情况下，一个语句可能由不同（嵌套）循环中的MLIR操作组成，例如，从内存加载到SSA寄存器的操作发生在外部循环中，而它在内部循环中被使用。该语句在循环层次结构中的位置并不明确。更重要的是，无法将其传达给多面体调度器。Polygeist通过在定义区域中将值存储在堆栈分配的内存引用中，并在使用区域中将其加载回去，从而解决了这一问题。图6展示了在操作%10中使用的值%0的这种转换。与基本情况类似，这在外部循环中创建了一个新的语句，可以独立调度。该方法可以视为reg2mem转换，这是在前端执行的mem2reg的反向操作。它仅适用于一部分值，并且在多面体调度完成后可以撤销。此外，为了减少依赖关系和内存占用，Polygeist执行简单的值分析，并在同一值在另一个内存位置可用并能够从那里读取时，避免创建堆栈分配的缓冲区。

C. Connecting MLIR to Polyhedral Tools Part-3

c) SCoP Formation: To define a SCoP, we outline individual statements into functions so that they can be represented as opaque calls with known memory footprints, similarly to Pencil [22]. This process also makes the inter-statement SSA dependencies clear. These dependencies exist between calls that use the same SSA value, but there are no values defined by these calls. We lift all local stack allocations and place them at the entry block of the surrounding function in order to keep them visible after loop restructuring. Figure 7 demonstrates the resulting IR. The remaining components of the polyhedral representation are derived as follows: the domain of the statement is defined to be the iteration space of its enclosing loops, constrained by their respective lower and upper bounds, and intersected with any "if" conditions. This process leverages the fact that MLIR expresses bounds and conditions directly as affine constructs. The access relations for each statement are obtained as unions of affine maps of the affine.load (read) and affine.store (must-write) operations, with RHS of the relation annotated by an "array" that corresponds to the SSA value of the accessed memref. Initial schedules are assigned using the $(2d + 1)$ formalism, with odd dimensions representing the lexical order of loops in the input program and even dimensions being equal to loop induction variables. Affine constructs in OpenScop are represented as lists of linear equality ($= 0$) or inequality (≥ 0) coefficients, which matches exactly the internal representation in MLIR, making the conversion straightforward. d) Code Generation Back to MLIR: The Pluto scheduler produces

new schedules in OpenScop as a result. Generating loop structure back from affine schedules is a solved, albeit daunting, problem [13], [14]. Polygeist relies on CLooG [13] to generate an initial loop-level AST, which it then converts to Affine dialect loops and conditionals. There is no need to simplify affine expressions at code generation since MLIR accepts them directly and can simplify them at a later stage. Statements are introduced as function calls with rewritten operands and then inlined.

c) SCoP 形成：为了定义一个 SCoP，我们将单独的语句划分为函数，这样它们就可以被表示为具有已知内存占用的透明调用，类似于 Pencil [22]。这个过程也使得语句间的 SSA 依赖关系变得清晰。这些依赖关系存在于使用相同 SSA 值的调用之间，但这些调用没有定义任何值。我们提升所有局部栈分配，并将它们放置在周围函数的入口块中，以保持它们在循环重构后仍然可见。图 7 展示了得到的 IR。

多面体表示的其余组件如下派生：语句的域被定义为其所包含循环的迭代空间，由各自的下界和上界约束，并与任何“if”条件相交。这个过程利用了 MLIR 直接将边界和条件表示为仿射构造的特性。每个语句的访问关系作为 affine.load（读取）和 affine.store（必须写入）操作的仿射映射的并集获得，关系的右侧被注释为与被访问的 memref 对应的“数组”，这对应于 SSA 值。初始调度是使用 $(2d + 1)$ 形式分配的，奇数维度表示输入程序中循环的词法顺序，而偶数维度等于循环引导变量。OpenScop 中的仿射构造表示为线性等式 ($= 0$) 或不等式 (≥ 0) 系数的列表，这正好与 MLIR 中的内部表示相匹配，使转换变得简单明了。

d) 代码生成回 MLIR：Pluto 调度器因此生成了 OpenScop 中的新调度。从仿射调度生成循环结构是一个已解决但相当复杂的问题 [13], [14]。Polygeist 依赖于 CLooG [13] 来生成初始的循环级 AST，然后将其转换为仿射方言的循环和条件。在代码生成时无需简化仿射表达式，因为 MLIR 可以直接接受它们，并能够在后续阶段进行简化。语句作为具有重写操作数的函数调用引入，然后进行内联。

D. Controlling Statement Granularity

Recall that Polygeist reconstructs "statements" from sequences of primitive operations (Section III-C). We initially designed an approach that recovers the statement structure similar to that in the C input, but this is not a requirement. Instead, statements can be formed from any subsets of MLIR operations as long as they can be organized into loops and sorted topologically (i.e., there are no use-def cycles between statements). To expose the dependencies between such statements to the affine scheduler, we reuse the idea of going through scratchpad memory: each statement writes the values required by other statements to dedicated memory locations, and the following statements read from those. The scratchpads are subject to partial array expansion [23] to minimize their effect on the affine scheduler as single-element scratchpad arrays create artificial scalar dependencies. This change in statement granularity gives the affine scheduler unprecedented flexibility allowing it to choose different schedules for different parts of the same C statement. Consider, for example, the statement S in Figure 8(top) surrounded by three loops iterating over i, j and k. Such contraction patterns are common in computational programs (this particular example can be found in the correlation benchmark with B=C, see Section V-E). The loop order that best exploits the locality is (k, i, j), which results in temporal locality for reads from B (the value is reused in all iterations of the now-innermost j loop) and in spatial locality for reads from C (consecutive values are read by consecutive iterations, increasing the likelihood of L1 cache hits). Yet, Pluto never proposes such an order because of a reduction dependency along the k dimension due to repeated read/write access to A[i][j] as Pluto tends to pick loops with fewer dependencies as outermost. While the dependency itself is inevitable, it can be moved into a separate statement T in Figure 8(bottom left). This approach provides scheduler with more freedom of choice for the first statement at a lesser memory cost than expanding the entire A array. It also factors out the reduction into a "canonical" statement that is easier to process for the downstream passes, e.g., vectorization. Implementing this transformation at the C level would require manipulating C AST and reasoning about C (or even C++) semantics. This is typically out of reach for source-to-source polyhedral optimizers such as Pluto that treat statements as black boxes. While it is possible to implement this transformation at the LLVM IR

level, e.g., in Polly, where statements are also reconstructed and injection of temporary allocations is easy, the heuristic driving the transformation is based on the loop structure and multi-dimensional access patterns which are difficult to recover at such a low level [9]. The space of potential splittings is huge—each MLIR operation can potentially become a statement. Therefore, we devise a heuristic to address the contraction cases similar to Figure 8. Reduction statement splitting applies to statements: • surrounded by at least 3 loops; • with LHS = RHS, and using all loops but the innermost; • with two or more different access patterns on the RHS. This covers statements that could have locality improved by a different loop order and with low risk of undesired fission. This heuristic merely serves as an illustration of the kind of new transformations Polygeist can enable.

D. 控制语句粒度

回想一下，Polygeist 是通过原始操作的序列重建“语句”的（见第三节C）。我们最初设计了一种恢复与 C 输入中相似的语句结构的方法，但这并不是必需的。实际上，只要可以将语句组织成循环并进行拓扑排序（即语句之间没有使用-定义循环），语句可以由任何子集的 MLIR 操作形成。为了向仿射调度器暴露这些语句之间的依赖关系，我们重用了通过临时存储器的思想：每个语句将其他语句所需的值写入专用内存位置，随后语句从这些位置读取。临时存储器受到部分数组扩展的限制[23]，以最小化它们对仿射调度器的影响，因为单元素的临时存储器数组会产生人为的标量依赖关系。这种语句粒度的变化为仿射调度器提供了前所未有的灵活性，使其能够为同一 C 语句的不同部分选择不同的调度。以图8（顶部）中的语句 S 为例，该语句被三个针对 i、j 和 k 的循环包围。这种收缩模式在计算程序中很常见（此特定示例可以在相关基准中找到， $B=C$ ，见第五节E）。最佳利用局部性的循环顺序是 (k, i, j)，这对从 B 的读取产生时效局部性（该值在现在最内层的 j 循环的所有迭代中被重用）并对从 C 的读取产生空间局部性（连续的值被连续的迭代读取，从而增加了 L1 缓存命中率的可能性）。然而，Pluto 从未提出这样的顺序，因为在 k 维度上由于对 $A[i][j]$ 的重复读/写访问而存在减少依赖关系，Pluto 趋向于选择依赖关系较少的循环作为最外层。虽然这种依赖关系本身是不可避免的，但它可以转移到图8（底部左侧）的单独语句 T 中。这种方法为调度器提供了更多的选择自由度，内存成本相对较低，而不是扩展整个 A 数组。它还将减少问题分解为一个“规范”语句，使得下游处理（例如向量化）更容易进行。

在 C 级别实现这一转换将需要操作 C 抽象语法树（AST）并推理 C（甚至 C++）的语义。这通常超出了诸如 Pluto 这样的源到源多面体优化器的能力，因为这些优化器将语句视为黑箱。虽然在 LLVM IR 级别实现这一转换也是可能的，例如在 Polly 中，在那里语句也被重建且暂时分配的注入较为容易，但驱动转换的启发式是基于循环结构和多维访问模式的，而在如此低的级别恢复这些信息是困难的[9]。

潜在分割的空间是巨大的——每个 MLIR 操作都有可能成为一个语句。因此，我们设计了一种启发式方法来处理类似于图8的收缩情况。减少语句分割适用于以下语句：• 至少被 3 个循环包围；• 左侧 = 右侧，并且使用所有循环，除了最内层的；• 右侧具有两个或更多不同的访问模式。这涵盖了那些可能通过不同循环顺序改善局部性的语句，并且风险较低，不会产生不希望的分裂。这个启发式方法仅作为 Polygeist 能够启用的新转换类型的示例。

E. Post-Transformations and Backend

Polygeist allows one to operate on both quasi-syntactic and SSA level, enabling analyses and optimizations that are extremely difficult, if not impossible, to perform at either level in isolation. In addition to statement splitting, we propose two techniques that demonstrate the potential of Polygeist. a) Transforming Loops with Carried Values (Reductions): Polygeist leverages MLIR's first-class support for loopcarried values to detect, express and transform reduction-like loops. This support does not require source code annotations, unlike source-level tools [24] that use annotations to enable detection, nor complex modifications for parallel code emission, unlike Polly [25], which suffers from LLVM missing first-class parallel constructs. We do not modify the polyhedral scheduler either, relying on post-processing for reduction parallelization, including outermost parallel reduction loops. The overall approach follows the definition proposed in [26] with adaptations to MLIR's region-based IR, and is

illustrated in Figure 9. Polygeist identifies memory locations modified on each iteration, i.e. load/store pairs with loop-invariant subscripts and no interleaving aliasing stores, by scanning the single-block body of the loop. These are transformed into loop-carried values or secondary induction variables, with the load/store pair lifted out of the loop and repurposed for reading the initial and storing the final value. Loop-carried values may be updated by a chain of side effect-free operations in the loop body. If this chain is known to be associative and commutative, the loop is a reduction. Loop-carried values are detected even in absence of reduction-compatible operations. Loops with such values contribute to mem2reg, decreasing memory footprint, but are not subject to parallelization.

b) Late Parallelization: Rather than relying on the dependence distance information obtained by the affine scheduler, Polygeist performs a separate polyhedral analysis to detect loop parallelism in the generated code. The analysis itself is a classical polyhedral dependence analysis [27], [28] implemented on top of MLIR region structure. Performing it after SSA-based optimizations, in particular mem2reg and reduction detection, allows parallelizing more loops. In particular, reduction loops and loops with variables whose value is only relevant within a single iteration similar to live-range reordering [29] but without expensive additional polyhedral analyses (live-range of an SSA value defined in a loop never extends beyond the loop).

E. 后变换及后端

Polygeist 使得在 quasi-syntactic 和 SSA 层面上操作成为可能，从而实现了在任何一个层面上单独进行分析和优化时极其困难（甚至不可能）的任务。除了语句拆分，我们提出了两种展示 Polygeist 潜力的技术。

a) 转换带有延续值的循环（归约）：Polygeist 利用 MLIR 对循环携带值的一级支持，来检测、表达和转换类似归约的循环。这种支持不需要源代码注释，而与使用注释以启用检测的源级工具 [24] 不同，也不需要复杂的修改来进行并行代码的生成，与 Polly [25] 不同，后者受到 LLVM 缺失一级并行构造的限制。我们也未对多面体调度器进行修改，而是依赖后处理来进行归约并行化，包括最外层的并行归约循环。

总体方法遵循了 [26] 中提出的定义，针对 MLIR 的基于区域的中间表示进行了调整，如图 9 所示。Polygeist 通过扫描循环的单块主体，识别每次迭代中被修改的内存位置，即具有循环不变下标且没有交错别名存储的加载/存储对。这些被转换为循环携带值或二次归纳变量，加载/存储对被提升到循环外并重新用于读取初始值和存储最终值。在循环主体中，循环携带值可以通过一系列无副作用操作进行更新。如果这个链被认为是结合的和交换的，那么该循环即为归约。即使在没有合适的归约操作的情况下，循环携带值也能被检测到。具有此类值的循环将对 mem2reg 产生贡献，降低内存占用，但不适合进行并行化。

b) 延迟并行化：Polygeist 并不依赖于通过仿射调度器获取的依赖距离信息，而是对生成的代码进行单独的多面体分析以检测循环并行性。该分析本身是一种经典的多面体依赖分析 [27]、[28]，在 MLIR 区域结构之上实现。在进行 SSA 基于优化之后执行这一步，特别是 mem2reg 和归约检测，能够并行化更多的循环。特别是，归约循环和那些变量值仅在单次迭代内相关的循环（类似于活跃范围重排序 [29]，但没有昂贵的额外多面体分析）都得到处理（在循环中定义的 SSA 值的活跃范围从不出循环）。

IV. EVALUATION

Our evaluation has two goals. 1) We want to demonstrate that the code produced by Polygeist without additional optimization does not have any inexplicable performance differences than a state-of-the-art compiler like Clang. 2) We explore how Polygeist's internal representation can support a mix of affine and SSA-based transformation in the same compilation flow, and evaluate the potential benefits compared to existing source and compiler-based polyhedral tools.

我们的评估有两个目标。1) 我们希望证明，Polygeist 生成的代码在没有额外优化的情况下，其性能与像 Clang 这样的现代编译器没有任何不可解释的差异。2) 我们探讨 Polygeist 的内部表示如何在同一编译流程中支持结合使用仿射变换和基于 SSA 的变换，并评估与现有的源代码和编译器基础上的多面体工具相比的潜在优势。

A. Experimental Setup

We ran our experiments on an AWS c5.metal instance with hyper-threading and Turbo Boost disabled. The system is Ubuntu 20.04 running on a dual-socket Intel Xeon Platinum 8275CL CPU at 3.0 GHz with 24 cores each, with 0.75, 35, 35.75 MB L1, L2, L3 cache per socket, respectively, and 256 GB RAM. We ran all 30 benchmarks from Poly-Bench [19], using the "EXTRALARGE" dataset. Pluto is unable to extract SCoP from the adi benchmark. We ran a total of 5 trials for each benchmark, taking the execution time reported by PolyBench; the median result is taken unless stated otherwise. Every measurement or result reported in the following sections refers to double-precision data. All experiments were run on cores 1-8, which ensured that all threads were on the same socket and did not potentially conflict with processes scheduled on core 0. In all cases, we use two-stage compilation: (i) using clang at -O3 excluding unrolling and vectorization; or Polygeist to emit LLVM IR from C; (ii) using clang at -O3 to emit the final binary. As several optimizations are not idempotent, a second round of optimization can potentially significantly boost (and rarely, hinder) performance. This is why we chose to only perform vectorization and unrolling at the last optimization stage. Since Polygeist applies some optimizations at the MLIR level (e.g., mem2reg), we compare against the twostage compilation pipeline as a more fair baseline (CLANG). We also evaluate a single-stage compilation to assess the effect of the two-stage flow (CLANGSING).

我们在一台 AWS c5.metal 实例上进行实验，该实例禁用了超线程和 Turbo Boost。系统为 Ubuntu 20.04，运行于双插槽 Intel Xeon Platinum 8275CL CPU，主频为 3.0 GHz，每个插槽有 24 个核心，L1、L2 和 L3 缓存分别为 0.75 MB、35 MB 和 35.75 MB，并配备 256 GB RAM。我们使用“EXTRALARGE”数据集运行 Poly-Bench [19] 中的所有 30 个基准测试。Pluto 无法从 adi 基准测试中提取 SCoP。我们对每个基准测试进行了总共 5 次试验，采用 PolyBench 报告的执行时间；除非另有说明，结果以中位数的形式给出。以下各节中报告的每个测量或结果均与双精度数据相关。所有实验均在核心 1-8 上运行，以确保所有线程位于同一插槽，并且不会与调度在核心 0 上的进程发生潜在冲突。

在所有情况下，我们采用两阶段编译：(i) 使用 clang 以 -O3 级别编译，不包括展开和向量化；或使用 Polygeist 从 C 发出 LLVM IR；(ii) 使用 clang 以 -O3 级别发出最终二进制文件。由于某些优化并非幂等的，因此第二轮优化可能会显著提升（而很少情况会抑制）性能。这就是我们选择仅在最后优化阶段进行向量化和展开的原因。由于 Polygeist 在 MLIR 级别应用了一些优化（例如，mem2reg），我们将其与两阶段编译流程（CLANG）进行比较，作为更公平的基线。我们还评估了单阶段编译，以评估两阶段流程的效果（CLANGSING）。

B. Baseline Performance

Polygeist must generate code with runtime as close as possible to that of existing compilation flows to establish a solid baseline. In other words, Polygeist should not introduce overhead nor speedup unless explicitly instructed otherwise, to allow for measuring the effects of additional optimizations. We evaluate this by comparing the runtime of programs produced by Polygeist with those produced by Clang at the same commit (Apr 2021) 2.

Figure 10 summarizes the results with the following flows:

- CLANG: A compilation of the program using Clang, when running two stages of optimization;
- CLANGSING: A compilation of the program using Clang, when running one stage of optimization;
- MLIR-CLANG: A compilation flow using the Polygeist frontend and preprocessing optimizations within MLIR, but not running polyhedral scheduling nor postprocessing.

B. 基线性能

Polygeist必须生成尽可能接近现有编译流程的运行时代码，以建立一个稳固的基线。换句话说，Polygeist不应该引入开销或加速，除非明确指示，以便能够测量额外优化的效果。我们通过比较Polygeist生成的程序与同一提交版本（2021年4月）下Clang生成的程序的运行时来评估这一点。图10总结了以下几种流程的结果：

- CLANG：使用Clang编译程序，运行两阶段优化的过程；
- CLANGSING：使用Clang编译程序，运行一阶段优化的过程；
- MLIR-CLANG：使用Polygeist前端的编译流程，在MLIR内进行预处理优化，但不执行多面体调度或后处理。

C. Compilation Flows

We compare Polygeist with a source-level and an IR-level optimizer (Pluto and Polly) in the following configurations:

- PLUTO: Pluto compiler auto-transformation [11] using `polycc 3` with `-noparallel` and `-tile` flags;
- PLUTOPAR: Same as above but with `-parallel` flag;
- POLLY: Polly [7] LLVM passes with affine scheduling and tiling, and no pattern-based optimizations [30];
- POLLYPAR: Same as above with auto-parallelization;
- POLYGEIST: Our flow with Pluto and extra transforms;
- POLYGEISTPAR: Same as above but with `-parallel` Pluto schedule, Polygeist parallelization and reductions.

Running between source and LLVM IR levels, we expect Polygeist to benefit from both worlds, thus getting code that is on par or better than competitors. When using Pluto, both standalone and within Polygeist, we disable the emission of vectorization hints and loop unrolling to make sure both transformations are fully controlled by the LLVM optimizer, which also runs in Polly flows. We run Polly in the latest stage of Clang compilation, using `-mllvm -polly` and additional flags to enable affine scheduling, tiling and parallelization as required. Polly is taken at the same LLVM commit as Clang. We disable pattern-based optimizations [30] that are not available elsewhere. Figures 11 and 12 summarize the results for sequential and parallel flows, respectively.

我们将Polygeist与源代码级和中间表示级优化器（Pluto和Polly）进行比较，配置如下：

- PLUTO：使用`-polycc 3`及`-noparallel`和`-tile`标志的Pluto编译器自动转换；
- PLUTOPAR：与上述相同，但使用`-parallel`标志；
- POLLY：Polly [7]的LLVM通道，具有仿射调度和切片，没有基于模式的优化 [30]；
- POLLYPAR：与上述相同，但进行了自动并行化；
- POLYGEIST：我们的工作流，结合了Pluto和额外的变换；
- POLYGEISTPAR：与上述相同，但使用`-parallel`的Pluto调度，Polygeist并行化和归约。

在源代码和LLVM IR级别之间运行时，我们期望Polygeist能够从这两者的优点中受益，从而生成的代码能够与竞争对手相媲美或更优。在使用Pluto时，无论是独立地还是在Polygeist中，我们都会禁用向量化提示和循环展开的发出，以确保这两种变换都完全由LLVM优化器控制，该优化器也在Polly流程中运行。我们在Clang编译的最新阶段运行Polly，使用`-mllvm -polly`及其他标志以启用仿射调度、切片和并行化。Polly与Clang使用相同的LLVM提交版本。我们禁用在其他地方不可用的基于模式的优化 [30]。图11和图12分别总结了顺序和并行流程的结果。

A. Benchmarking

The transformation of reduction loops, in particular parallelization, may result in a different order of partial result accumulation. This is not allowed under IEEE 754 semantics, but is supported by compilers with `-ffast-math` option. We found that Polybench allocation function hinders Clang/LLVM alias analysis, negatively affecting performance 2 LLVM commit 20d5c42e0ef5d252b434bcb610b04f1cb79fe771 3 Pluto commit dae26e77b94b2624a540c08ec7128f20cd7b7985 in, e.g., `adi`. Therefore, we modified all benchmarks to use `malloc` that is known to produce non-aliasing pointers.

降低循环的转换，特别是并行化，可能会导致部分结果累积的顺序发生变化。这在 IEEE 754 语义下是不允许的，但使用 `-ffast-math` 选项的编译器支持这种情况。我们发现，Polybench 的分配函数会妨碍 Clang/LLVM 的别名分析，从而对性能产生负面影响 2 LLVM 提交 20d5c42e0ef5d252b434bcb610b04f1cb79fe771 3 Pluto 提交 dae26e77b94b2624a540c08ec7128f20cd7b7985，举例来说，在 `adi` 中。因此，我们修改了所有基准测试，使用已知会产生非别名指针的 `malloc`。

B. Baseline Comparison

We did not observe a significant difference between the runtimes of CLANG and CLANGSING configurations, with a geometric mean of 0.43% symmetric difference 4 across benchmarks. Therefore, we only consider CLANG as baseline throughout the remainder of this paper. We did not observe a significant difference between the runtimes of CLANG and MLIR-CLANG configurations either, with a geometric mean of 0.24% symmetric difference. We found a variation in runtimes of short-running benchmarks, in particular `jacobi-1d`. This can be attributed to the interaction with the data initialization and benchmarking code, and with other OS processes. Excluding the benchmarks running in under 0.05s (`jacobi-1d`, `gesummv`, `atax`, `bicg`) from the analysis, we obtain 0.32% and 0.17% geomean symmetric differences respectively for the two comparisons above. These results suggest that our flow has no unexplained (dis)advantages over the baseline.

我们没有观察到CLANG和CLANGSING配置之间的运行时间存在显著差异，在基准测试中，它们的几何平均对称差为0.43%。因此，在本文的其余部分中，我们仅考虑CLANG作为基线。同样，我们也没有观察到CLANG和MLIR-CLANG配置之间的运行时间存在显著差异，其几何平均对称差为0.24%。

我们发现短运行基准的运行时间存在变动，特别是在`jacobi-1d`上。这可以归因于数据初始化和基准测试代码之间的相互作用，以及其他操作系统进程的影响。从分析中排除运行时间少于0.05秒的基准（如`jacobi-1d`、`gesummv`、`atax`、`bicg`），我们分别获得上述两个比较的几何平均对称差为0.32%和0.17%。这些结果表明，我们的流程在基线之上没有未解释的（不）优势。

C. Performance Differences in Sequential Code

Overall, Polygeist leads to larger speedups, with $2.53\times$ geometric mean, than both Pluto ($2.34\times$) and Polly ($1.41\times$), although improvements are not systematic. Some difference between Polygeist and Polly is due to the employed polyhedral schedulers, e.g., in `lu` and `mvt`. Polygeist produces code faster than both Pluto and Polly in `2mm`, `3mm` and others thanks to statement splitting, see Section V-E. Given identical statements and schedules, codegen-level optimization accounts for other performance difference. `seidel-2d` is the clearest example: Pluto executes $2.7 \cdot 10^{11}$ more integer instructions than Polygeist. Assuming these to be index/address computations, a mix of add (throughput 1/2 or 1/4) and imul/shl (throughput 1), we can expect a $\approx 59s$ difference at 3GHz, consistent with experimental observations. Polygeist optimizes away a part of those in its post-optimization phase and emits homogeneous address computation from memref with proper machine size type, enabling more aggressive bound analysis and simplification in the downstream compiler. Conversely, `jacobi-2d` has poorer performance because Polygeist gives up on simplifying CLooG code, with up to 75 statement copies in 40 branches, for compiler performance reasons, as opposed to Clang that takes up to 5s to process it but results in better vectorization. Further work is necessary to address this issue by emitting vector instructions directly from Polygeist.

整体而言，Polygeist相较于Pluto ($2.34\times$) 和 Polly ($1.41\times$) 提供了更大的加速，几何平均值为 $2.53\times$ ，尽管改进并不是系统性的。Polygeist 和 Polly 之间的一些差异是由于所采用的多面体调度器，例如在 `lu` 和 `mvt` 中。得益于语句分割，Polygeist 在 `2mm`、`3mm` 及其他某些情况下生成的代码速度超过了 Pluto 和 Polly，详见第 V-E 节。

在给定相同语句和调度的情况下，代码生成层面的优化是造成其他性能差异的原因。seidel-2d 是最明显的例子：Pluto 执行的整数指令比 Polygeist 多出 $2.7 \cdot 10^{11}$ 次。假设这些指令都是索引/地址计算，混合了 add（吞吐量为 1/2 或 1/4）和 imul/shl（吞吐量为 1），我们可以预计在 3GHz 下会有 ≈ 59 秒的差异，这与实验观察结果一致。Polygeist 在其后优化阶段优化掉了一部分指令，并从 memref 发出适当机器大小类型的均匀地址计算，使下游编译器能够进行更激进的界限分析和简化。相反，由于 Polygeist 为了编译器性能原因放弃了对 CLooG 代码的简化，jacobi-2d 的性能较差，在 40 个分支中有多达 75 个语句副本，而 Clang 处理这些代码需耗时多达 5 秒，但其结果却实现了更好的向量化。需要进一步的工作来解决这个问题，通过直接从 Polygeist 发出向量指令。

D. Performance Differences In Parallel Code

Similarly to sequential code, some performance differences are due to different schedulers. For example, in cholesky and lu, both Pluto and Polygeist outperform Polly, and the remaining gap can be attributed to codegen-level differences. Conversely, in gemver and mvt Polly has a benefit over both Fig. 10. Mean and 95% confidence intervals (log scale) of program run time across 5 runs of Polybench in CLANG, CLANGSING and MLIR-CLANG configurations, lower is better. The run times of code produced by Polygeist without optimization is comparable to that of Clang. No significant variation is observed between single and double optimization. Short-running jacobi-1d shows high intra-group variation. Fig. 11. Median speedup over CLANG for sequential configurations (log scale), higher is better. Polygeist outperforms ($2.53 \times$ geomean speedup) both Pluto ($2.34 \times$) and Polly ($1.41 \times$) on average. Pluto can't process adi, which is therefore excluded from summary statistics. Fig. 12. Median speedup over CLANG for parallel configurations (log scale), higher is better. Polygeist outperforms ($9.47 \times$ geomean speedup) both Pluto ($7.54 \times$) and Polly ($3.26 \times$) on average. Pluto can't process adi, which is therefore excluded from summary statistics. Pluto and Polygeist. On ludcmp and syr(2)k, SSA-level optimizations let Polygeist produce code which is faster than Pluto and at least as fast as Polly. These results demonstrate that Polygeist indeed leverages the benefits of both the affine and SSA-based optimizations. Polygeist is the only flow that obtains speedup on deriche ($6.9 \times$) and symm ($7.7 \times$). Examining the output code, we observe that only Polygeist manages to parallelize these two benchmarks. Considering the input code in Figure 13, one can observe that the i loop reuses the ym1 variable, which is interpreted as parallelism-preventing loop-carried dependency by polyhedral schedulers. Polygeist performs its own parallelism analysis after promoting ym1 to an SSA register (carried by the j loop) whose use-def range does not prevent parallelization. Similarly, the Polygeist parallelizer identifies two benchmarks with parallel reduction loops that are not contained in other parallel loops: gramschmidt and durbin. gramschmidt benefits from a $56 \times$ speedup with Polygeist, compared to $34 \times$ with Polly and $54 \times$ with Pluto. durbin sees a $6 \times$ slowdown since the new parallel loop has relatively for (i=0; i<_pb_w; i++) {="" ym1="SCALAR_VAL(0.0); ...="" for="" (j=0;" j<_pb_h;" j++) {="" *="" }="" %z="constant" 0.0="" :="" f64="" affine.parallel="" %i="" ...="" {="" affine.for="" %j="" ...="" iter_args("%ym1=%z)->f64 {"="" %0="" affine.load="" %y1[%i,%j]="" //="" ...="" affine.yield="" %0="" }}="" Fig.="" 13.="" Excerpt="" from="" the="" deriche="" benchmark.="" The="" outer="" loop="" reuses="" ym1="" which="" makes="" it="" appear="" non-parallel="" to="" affine="" schedulers="" (left).="" Polygeist="" detects="" parallelism="" thanks="" to="" its="" mem2reg="" optimization,="" reduction-like="" loop-carried="" %ym1="" value="" detection="" and="" late="" parallelization="" (right).="" few="" iterations="" and="" is="" nested="" inside="" a="" sequential="" loop,="" leading="" to="" synchronization="" costs="" that="" outweigh="" the="" parallelism="" benefit.="" Section="" V-F="" explores="" the="" durbin="" benchmark="" in="" more="" detail.="" Polybench="" is="" a="" collection="" of="" codes="" (mostly)="" known="" to="" be="" parallel="" and,="" as="" such,="" has="" little="" need="" for="" reduction="" parallelization="" on="" CPU="" where="" one="" degree="" of="" parallelism="" is="" sufficient.="" When="" targeting="" inherently="" target="" architectures="" as="" GPUs,="" however,="" exploiting="" reduction="" parallelism="" could="" be="" vital="" for="" achieving="" peak="" performance="" [31],="" [24].

同样，对于并行代码，某些性能差异是由于不同的调度器。例如，在 cholesky 和 lu 中，Pluto 和 Polygeist 的性能都优于 Polly，剩余的差距可以归因于代码生成级别的差异。相反，在 gemver 和 mvt 中，Polly 相较于两者皆有优势。

图 10 显示了在 CLANG、CLANGSING 和 MLIR-CLANG 配置下，Polybench 的五次运行中程序运行时间的均值和 95% 置信区间（对数尺度），值越低越好。由 Polygeist 产生的未优化代码的运行时间与 Clang 的运行时间相当。不同程度的单重和双重优化之间没有观察到显著差异。运行时间较短的 jacobi-1d 显示出高的组内变异性。

图 11 显示了在顺序配置下相对于 CLANG 的中位数加速比（对数尺度），值越高越好。Polygeist 在平均情况下的加速比（几何均值加速比为 $2.53\times$ ）优于 Pluto ($2.34\times$) 和 Polly ($1.41\times$)。Pluto 无法处理 adi，因此该案例被排除在总结统计之外。

图 12 显示了在并行配置下相对于 CLANG 的中位数加速比（对数尺度），值越高越好。Polygeist 在平均情况下的加速比（几何均值加速比为 $9.47\times$ ）优于 Pluto ($7.54\times$) 和 Polly ($3.26\times$)。Pluto 无法处理 adi，因此该案例也被排除在总结统计之外。

在 ludcmp 和 syr(2)k 中，SSA 级别的优化使得 Polygeist 生成的代码比 Pluto 更快，并且至少与 Polly 同样快。这些结果表明，Polygeist 确实利用了仿射优化和 SSA 基础优化的优点。

Polygeist 是唯一在 deriche (加速比 $6.9\times$) 和 symm (加速比 $7.7\times$) 上实现加速的编译流程。通过检查输出代码，我们观察到只有 Polygeist 成功地对这两个基准进行了并行化。考虑到图 13 中的输入代码，可以观察到 i 循环重用了 ym1 变量，这被仿射调度器解释为妨碍并行化的循环携带依赖。Polygeist 在将 ym1 提升为一个由 j 循环携带的 SSA 寄存器后，进行自己的并行性分析，其使用定义范围不妨碍并行化。

类似地，Polygeist 的并行化器识别出两个具有并行归约循环的基准，这些循环不包含在其他并行循环中：gramschmidt 和 durbin。相比于 Polly 的 $34\times$ 和 Pluto 的 $54\times$ ，gramschmidt 在 Polygeist 的帮助下获得了 $56\times$ 的加速。durbin 则出现了 $6\times$ 的减速，因为新的并行循环迭代次数相对较少，并且嵌套在一个顺序循环内，导致同步成本超过了并行化带来的好处。第 V-F 节将更详细地探讨 durbin 基准。

Polybench 是一组（大多数已知）具有并行特性的代码，因此在 CPU 上进行归约并行化的需求不大，因为只有一度的并行性就已足够。然而，当针对固有的目标架构如 GPU 时，利用归约并行性可能对于实现峰值性能至关重要 [31], [24]。

E. Case Study: Statement Splitting

We identified 5 benchmarks where the statement splitting heuristic applied: 2mm, 3mm, correlation, covariance and trmm. To assess the effect of the transformation, we executed these benchmarks with statement splitting disabled, suffixed with -nosplit in Figure 14. In sequential versions, 2mm is 4.1% slower (3.13s vs 3.26s), but the other benchmarks see speedups of 25%, 50%, 51% and 27%, respectively. For parallel versions, the speedups are of 36%, 20%, 44%, 40% and -9% respectively. Examination of polyhedral scheduler outputs demonstrates that it indeed produced the desired schedules. For example, in the correlation benchmark which had the statement $A[i][j] += B[k][i] * B[k][j]$ Polygeist was able to find the (k, i, j) loop order after splitting. Using hardware performance counters on sequential code we confirm that the overall cache miss ratio has indeed decreased by 75%, 50%, 20%, 27%, and -26%, respectively. However, the memory traffic estimated by the number of bus cycles has increased by 9% for 2mm, and decreased by 18%, 32%, 32%, and 21% for the other benchmarks. This metric strongly correlates with the observed performance difference in the same run ($r = 0.99$, $p = 3 \cdot 10^{-11}$). This behavior is likely due to the scheduler producing a different fusion structure, e.g., not fusing outermost loops in 2mm, which also affects locality. Similar results can be observed for parallel code. Further research is necessary to exploit the statement splitting opportunities, created by Polygeist, and interplay with fusion.

E. 案例研究：语句拆分

我们确定了 5 个基准测试，其中应用了语句拆分启发式方法：2mm、3mm、相关性、协方差和 trmm。为了评估变换的效果，我们执行了这些基准测试，并禁用了语句拆分，用 `-nosplit` 后缀标记，如图 14 所示。在顺序版本中，2mm 的速度慢了 4.1%（3.13 秒对比 3.26 秒），但其他基准测试分别看到了 25%、50%、51% 和 27% 的加速。对于并行版本，速度提升分别为 36%、20%、44%、40% 和 -9%。

对多面体调度器输出的检查表明，它确实生成了预期的调度。例如，在相关性基准中，语句 $A[i][j] += B[k][i] * B[k][j]$ Polygeist 能够在拆分后找到 (k, i, j) 的循环顺序。通过在顺序代码上使用硬件性能计数器，我们确认整体缓存缺失率确实分别降低了 75%、50%、20%、27% 和 -26%。然而，由总匣周期估算的内存流量在 2mm 中增加了 9%，而其他基准测试则分别减少了 18%、32%、32% 和 21%。该指标与同一运行中观察到的性能差异强相关 ($r = 0.99, p = 3 \cdot 10^{-11}$)。这种行为可能是由于调度器产生了不同的融合结构，例如，在 2mm 中没有融合最外层的循环，这也影响了局部性。并行代码中也可以观察到类似的结果。进一步的研究有必要利用 Polygeist 创造的语句拆分机会，并与融合进行相互作用。

F. Case Study: Reduction Parallelization in durbin

In this benchmark, Polygeist uses its reduction optimization to create a parallel loop that other tools cannot. For the relatively small input run by default, $N = 4000$ iterations inside another sequential loop with N iterations, the overall performance decreases. We hypothesize that the cost of creating parallel threads and synchronizing them outweighs the benefit of the additional parallelism and test our hypothesis by increasing N . Considering the results in Figure 15, one observes that Polygeist starts yielding speedups (> 1) for $N \geq 16000$ whereas Polly only does so at $N \geq 224000$, and to a much lesser extent: $6.62 \times$ vs $1.01 \times$. Without reduction parallelization, Polygeist follows the same trajectory as Polly. Pluto fails to parallelize any innermost loop and shows no speedup. This evidences in favor of our hypothesis and highlights the importance of being able to parallelize reductions.

F. 案例研究：Durbin 中的归约并行化

在这个基准测试中，Polygeist 利用其归约优化创建了一个其他工具无法实现的并行循环。对于默认运行的相对较小的输入， $N = 4000$ 的迭代嵌套在另一个具有 N 次迭代的顺序循环中，整体性能出现下降。我们假设创建并行线程和同步它们的成本超过了额外并行性带来的收益，并通过增加 N 来验证我们的假设。根据图 15 中的结果，可以观察到，Polygeist 在 $N \geq 16000$ 时开始产生加速效果 (> 1)，而 Polly 仅在 $N \geq 224000$ 时才会如此，并且幅度要小得多： $6.62 \times$ 对比 $1.01 \times$ 。在没有归约并行化的情况下，Polygeist 的表现与 Polly 相同。Pluto 未能对任何最内层循环进行并行化，并且显示没有加速。这支持了我们的假设，并强调了能够并行化归约的重要性。

VI. RELATED WORK

- a) MLIR Frontends: Since the adoption of MLIR under the LLVM umbrella, several frontends have been created for generating MLIR from domain-specific languages. Teckyl [2] connects the productivity-oriented Tensor Comprehensions [1] notation to MLIR's Linalg dialect. Flang—the LLVM's Fortran frontend—models Fortran-specific constructs using the FIR dialect [32]. COMET, a domain-specific compiler for chemistry, introduces an MLIR-targeting domain-specific frontend from a tensor-based language [33]. NPComp aims at providing the necessary infrastructure to compile numerical Python and PyTorch programs taking advantage of the MLIR infrastructure [34]. PET-to-MLIR converts a subset of polyhedral C code to MLIR's Affine dialect by parsing pet's internal representation. In addition to currently not handling specific constructs (ifs, symbolic bounds, and external function calls), parsing pet's representation limits the frontend's usability as it cannot interface with non-polyhedral code such as initialization, verification, or printing routines [35]. In contrast, Polygeist generates MLIR from non-polyhedral code (though not necessarily in the Affine dialect). CIRCT is a new project under the

LLVM umbrella that aims to apply MLIR development methodology to the electronic design automation industry [17]. Stripe uses MLIR Affine dialect as a substrate for loop transformations in machine learning models, including tiling and vectorization, and accepts a custom DSL as input [36].

VI. 相关工作

a) MLIR前端: 自从MLIR被纳入LLVM框架后, 已经创建了多个前端用于从特定领域语言生成MLIR。Teckyl [2] 将面向生产力的张量理解 [1] 符号连接到MLIR的Linalg方言。Flang, LLVM的Fortran前端, 使用FIR方言建模Fortran特定的构造 [32]。COMET, 一个针对化学领域的特定领域编译器, 引入了一个从基于张量的语言到MLIR的特定领域前端 [33]。NPComp旨在提供必要的基础设施以便编译数值Python和PyTorch程序, 从而利用MLIR基础设施 [34]。PET-to-MLIR通过解析pet的内部表示, 将一部分多面体C代码转换为MLIR的Affine方言。除了当前不处理特定构造 (如if语句、符号边界和外部函数调用) 外, 解析pet的表示限制了前端的可用性, 因为它无法与非多面体代码 (如初始化、验证或打印例程) 接口 [35]。相对而言, Polygeist能够从非多面体代码生成MLIR (尽管不一定在Affine方言中)。CIRCT是一个新的项目, 旨在将MLIR开发方法应用于电子设计自动化行业, 隶属于LLVM框架 [17]。Stripe使用MLIR的Affine方言作为机器学习模型中的循环变换的基础, 包括拆分和向量化, 并接受自定义DSL作为输入 [36]。

b) Compilers Leveraging Multiple Representations:

The SUIF compiler infrastructure pioneered a combined internal representation that supports higher-level transformations, including loop optimization and parallelization [37] and, in particular, reduction parallelization [38]. Polygeist leverages MLIR abstractions unavailable in SUIF: regular and affine for loops, OpenMP reduction constructs, etc. It also benefits from the SSA+regions form, which is only available as external extension in SUIF [39], for IR simplification. PIPS supports loop transformations and inter-procedural optimization when targeting OpenMP [40], [41]. Polygeist differs from both by emitting machine code rather than source code, which allows it to emit parallel runtime and other directives that have no representation in the source language such as C. c) Combining "Classical" and Polyhedral Flows: Few papers have focused on combining "classical", mostly ASTlevel, and polyhedral transformations. PolyAST pioneered the approach by combining an affine scheduler with ASTlevel heuristics for fusion and tiling [42], although similar results were demonstrated with only polyhedral transformations [43]. An analogous approach was experimented in CUDA-CHILL [44]. Arguably, many automated polyhedral flows perform loop fusion and/or tiling as a separate step that can be assimilated to classical transformations. Pluto [11] uses several "syntactic" postprocessing passes to exploit spatial locality and parallelism in stencils [45]. Several tools have been proposed to drive polyhedral loop transformations with scripts using classical loop transformations such as fusion and permutation as operations, including URUK [46], CHILL [47] and Clay [48]. Polygeist differs from all of these because it preserves the results of such transformations in its IR along with polyhedral constructs and enables interaction between different levels of abstraction. d) Additional (Post-)Polyhedral Transformations: Support for handling reduction loops was proposed in Polly [25], but the code generation is not implemented. At the syntactic level, reduction support was added to PET via manual annotation with PENCIL directives [24]. R-Stream reportedly uses a variant of statement splitting to affect scheduler's behavior and optimize memory consumption [49]. POLYSIMD uses variable renaming around PPCG polyhedral flow to improve vectorization [50]. Polygeist automates these leveraging both SSA and polyhedral information.

b) 利用多种表示的编译器: SUIF编译器基础设施开创了一种组合内部表示, 支持更高级的转换, 包括循环优化和并行化 [37], 尤其是归约并行化 [38]。Polygeist利用SUIF中不可用的MLIR抽象: 常规和仿射循环, OpenMP归约构造等。它还受益于仅作为外部扩展存在于SUIF中的SSA+区域形式 [39], 可以用于中

间表示的简化。PIPS在针对OpenMP时支持循环转换和过程间优化 [40], [41]。Polygeist与这两者的不同之处在于它发出机器代码而不是源代码，从而能够发出并行运行时和其他在源语言（如C）中没有表示的指令。

c) 结合“经典”和多面体流程：少数论文关注于将“经典”的、大多数是AST级别的转化与多面体转化结合起来。PolyAST通过结合仿射调度器与AST级别的启发式方法进行融合和切片 [42]，开创了这种方法，尽管仅使用多面体转化也展示了类似的结果 [43]。CUDA-CHILL [44] 也实验了一种类似的方法。可以说，许多自动化的多面体流程将循环融合和/或切片作为一个独立步骤来执行，这可以被认为是一种经典转化。Pluto [11] 使用几个“语法”后处理过程来利用模板中的空间局部性和并行性 [45]。已提出多个工具，以经典循环转化如融合和排列作为操作，驱动多面体循环转化，包括URUK [46]、CHILL [47] 和Clay [48]。Polygeist与这些工具的不同之处在于它在其IR中保留了此类转化的结果，并且与多面体构造一起，允许不同抽象层次之间的交互。

d) 额外的（后）多面体转化：Polly [25] 提出了处理归约循环的支持，但并未实现代码生成。在语法层面，通过手动注释使用PENCIL指令向PET添加了归约支持 [24]。据报道，R-Stream使用了一种变体的语句拆分以影响调度器的行为并优化内存消耗 [49]。POLYSIMD在PPCG多面体流程周围使用变量重命名以改善向量化 [50]。Polygeist利用SSA和多面体信息自动化这些操作。

e) Integration of Polyhedral Optimizers into Compilers:

Polyhedral optimization passes are available in production (GCC [8], LLVM [7], IBM XL [51]) and research (R-Stream [49], ROSE [52]) compilers. In most cases, the polyhedral abstraction must be extracted from a lower-level representation before being transformed and lowered in a dedicated code generation step [13], [14]. This extraction process is not guaranteed and may fail to recover high-level information available at the source level [9]. Furthermore, common compiler optimizations such as LICM are known to interfere with it [10]. Polygeist maintains a sufficient amount of high-level information, in particular loop and n-D array structure, to circumvent these problems by design. Source-to-source polyhedral compilers such as Pluto [11] and PPCG [5] operate on a C or C++ level. They lack interaction with other compiler optimizations and a global vision of the code, which prevents, e.g., constant propagation and inlining that could improve the results of polyhedral optimization. Being positioned between the AST and LLVM IR levels, Polygeist enables the interaction between higher and lower-level abstractions that is otherwise reduced to compiler pragmas, i.e. mere optimization hints. Furthermore, Polygeist can rely on MLIR's progressive raising [53] to target abstractions higher level than C code with less effort than polyhedral frameworks [54].

e) 聚合优化器与编译器的集成:

聚合优化传递在生产（GCC [8], LLVM [7], IBM XL [51]）和研究（R-Stream [49], ROSE [52]）编译器中可用。在大多数情况下，必须从低级表示中提取聚合抽象，然后在专门的代码生成步骤中进行转换和降级 [13], [14]。这个提取过程并不是保证成功的，可能无法恢复源级别上可用的高级信息 [9]。此外，常见的编译器优化，例如LICM，已知会对此产生干扰 [10]。Polygeist通过设计维护足够的高级信息，尤其是循环和多维数组结构，以规避这些问题。

源到源的聚合编译器，如Pluto [11] 和PPCG [5]，在C或C++级别上运行。它们缺乏与其他编译器优化的交互，以及对代码的全局视角，这阻碍了例如常量传播和内联等可能改善聚合优化结果的过程。Polygeist位于AST和LLVM IR级别之间，使得更高级别和低级别抽象之间的交互成为可能，而这种交互在其他情况下仅限于编译器指示，即仅仅是优化提示。此外，Polygeist可以依靠MLIR的逐步提升 [53]，以比聚合框架 [54] 更少的努力，针对更高于C代码的抽象。

VII. DISCUSSION

A. Limitations a) Frontend: While Polygeist could technically accept any valid C or C++ thanks to building off Clang, it has the following limitations. Only structs with values of the same type or are used within specific functions (such as FILE within fprintf) are supported due to the lack of a struct-type in high-level MLIR dialects. All functions that allocate memory must be compiled with Polygeist and not a C++ compiler to ensure that a memref is emitted rather than a pointer. b) Optimizer: The limitations of the optimizer are inherited from those of the tools involved. In particular, the MLIR affine value categorization results in all-or-nothing modeling, degrading any loop to non-affine if it contains even one nonaffine access or a negative step. Running Polygeist's backend on code not generated by Polygeist's frontend, which reverses loops with negative steps, is limited to loops with positive indices. Finally, MLIR does not yet provide extensive support for non-convex sets (typically expressed as unions). Work is ongoing within MLIR to address such issues. c) Experiments: While our experiments clearly demonstrate the benefits of the techniques implemented in Polygeist-statement splitting and late (reduction) parallelization -non-negligible effects are due to scheduler difference: Pluto in Polygeist and isl in Polly. The version of Polly using Pluto 5 is not compatible with modern LLVM necessary to leverage MLIR. Connecting isl scheduler to Polygeist may have yielded results closer to Polly, but still not comparable more directly because of the interplay between SCoP detection, statement formation and affine scheduling.

VII. 讨论

A. 限制

- a) 前端: 尽管Polygeist技术上可以接受任何有效的C或C++代码, 因为它是基于Clang构建的, 但仍然存在以下限制。由于高层次MLIR方言中缺乏结构类型, 仅支持值类型相同的结构体, 或在特定函数内(如fprintf中的FILE)使用的结构体。所有分配内存的函数必须使用Polygeist编译, 而不是使用C++编译器, 这样才能确保生成的是memref而非指针。
- b) 优化器: 优化器的限制源自于所涉及工具的限制。特别是, MLIR的仿射值分类导致了全有或全无的建模, 如果任何循环中包含一个非仿射访问或负步长, 就会将该循环降级为非仿射。对不是由Polygeist前端生成的代码使用Polygeist的后端运行时, 将限制在具有正索引的循环上。最后, MLIR目前尚未提供对非凸集合的广泛支持(通常以并集的形式表示)。MLIR内部正在进行相关工作以解决此类问题。
- c) 实验: 尽管我们的实验清楚地展示了在Polygeist中实现的技术的好处——语句拆分和后期(归约)并行化——但仍不容忽视的效果是由于调度器的差异: Polygeist中的Pluto和Polly中的isl。使用Pluto 5的Polly版本与现代LLVM不兼容, 而后者又是利用MLIR所必需的。将isl调度器连接到Polygeist可能会使结果更接近Polly, 但仍然无法更直接地进行比较, 因为SCoP检测、语句形成和仿射调度之间的相互作用。

B. Opportunities and Future Work

Connecting MLIR to existing polyhedral flows opens numerous avenues for compiler optimization research, connecting polyhedral and conventional SSA-based compiler transformations. This gives polyhedral schedulers access to important analyses such as aliasing and useful information such as precise data layout and target machine description. Arguably, this information is already leveraged by Polly, but the representational mismatch between LLVM IR and affine loops makes it difficult to exploit them efficiently. MLIR exposes similar information at a sufficiently high level to make it usable in affine transformations. By mixing abstractions in a single module, MLIR provides finer-grain control over the entire transformation process. An extension of Polygeist can, e.g., ensure loop vectorization by directly emitting vector instructions instead of relying on pragmas, which are often merely a recommendation for the compiler. The flow can also control lower-level mechanisms like prefetching or emit specialized hardware instructions. Conversely, polyhedral analyses can guarantee downstream passes that, e.g., address computation never produces out-ofbounds accesses and other information. Future work is necessary on controlling statement granularity made possible by Polygeist. Beyond

affecting affine schedules, this technique enables easy rematerialization and local transposition buffers, crucial on GPUs [55], as well as software pipelining; all without having to produce C source which is known to be complex [56]. On the other hand, this may have an effect on the compilation time as the number of statements is an important factor in the complexity bound of the dependence analysis and scheduling algorithms.

将MLIR与现有的多面体流程连接起来，为编译器优化研究开辟了许多新途径，连接了多面体和传统的基于SSA的编译器转换。这使得多面体调度器可以访问一些重要的分析信息，如别名分析以及精确的数据布局和目标机器描述等有用信息。可以说，这些信息已经被Polly利用，但LLVM IR和仿射循环之间的表示不匹配使得高效利用这些信息变得困难。MLIR以足够高的抽象层级暴露了类似的信息，使其可用于仿射变换。

通过在单个模块中混合不同的抽象，MLIR对整个转换过程提供了更细粒度的控制。Polygeist的扩展可以确保通过直接发出向量指令来实现循环向量化，而不是依赖于常常只是编译器建议的pragma指令。该流程还可以控制更低层次的机制，如预取或发出专用的硬件指令。反之，多面体分析可以确保下游的处理过程，例如，地址计算永远不会导致越界访问，以及其他信息。

在Polygeist的支持下，控制语句粒度的未来工作是必要的。除了影响仿射调度外，该技术还使得轻松的重新物化和局部转置缓冲成为可能，这在GPU上是至关重要的[55]，以及软件流水线化；所有这些都不需要生成已知复杂的C源代码[56]。另一方面，语句数量可能会对编译时间产生影响，因为语句数量是依赖分析和调度算法复杂性边界的重要因素。

C. Alternatives

Instead of allowing polyhedral tools to parse and generate MLIR, one could emit C (or C++) code from MLIR 6 and use C-based polyhedral tools on the C source, but this approach decreases the expressiveness of the flow. Some MLIR constructs, such as parallel reduction loops, can be directly expressed in the polyhedral model, whereas they would require a non-trivial and non-guaranteed raising step in C. Some other constructs, such as prevectorized affine memory operations, cannot be expressed in C at all. Polygeist enables transparent handling of such constructs in MLIR-to-MLIR flows, but we leave the details of such handling for future work. The Polygeist flow can be similarly connected to other polyhedral formats, in particular isl. We choose OpenScop for this work because it is supported by a wider variety of tools. isl uses schedule trees [57] to represent the initial and transformed program schedule. Schedule trees are sufficiently close to the nested-operation IR model making the conversion straightforward: "for" loops correspond to band nodes (one loop per band dimension), "if" conditionals correspond to filter nodes, function-level constants can be included into the context node. The tree structure remains the same as that of MLIR regions. The inverse conversion can be obtained using isl's AST generation facility [14].

替代方案

与其让多面体工具解析并生成MLIR，不如直接从MLIR发出C（或C++）代码，并在C源代码上使用基于C的多面体工具，但这种方法降低了编译流程的表达能力。一些MLIR构造，例如并行归约循环，可以直接在多面体模型中表达，而在C中却需要一个复杂且不保证可行的提升步骤。另一些构造，例如预向量化的仿射内存操作，则根本无法在C中表达。Polygeist支持在MLIR到MLIR的流程中透明地处理这些构造，但我们将此类处理的细节留待未来研究。

Polygeist流程也可以与其他多面体格式相连接，特别是与isl相连。我们选择OpenScop作为本研究的对象，因为它得到更多工具的支持。isl使用调度树来表示初始和转化后的程序调度。调度树与嵌套操作的IR模型非常接近，使得转换变得简单：`for`循环对应于带节点（每个带维度一个循环），`if`条件对应于过滤节点，函数级常量可以包含在上下文节点中。树结构与MLIR区域保持一致。逆向转换可以通过isl的AST生成工具获得。

VIII. CONCLUSION Part-1

We present Polygeist, a compilation workflow for importing existing C or C++ code into MLIR and allows polyhedral tools, such as Pluto, to optimize MLIR programs. This enables MLIR to benefit from decades of research in polyhedral compilation. We demonstrate that the code generated by Polygeist has comparable performance with Clang, enabling unbiased comparisons between transformations built for MLIR and existing polyhedral frameworks. Finally, we demonstrate the optimization opportunities enabled by Polygeist considering two complementary transformations: statement splitting and reduction parallelization. In both cases, Polygeist achieves better performance than state-of-the-art polyhedral compiler and source-to-source optimizer. comments regarding how this may need to be modified to run on a system with hardware or software configuration that is distinct from what we used. As expected, the command description mirrors much of the content of the docker file. While a docker file is certainly more convenient and a good way of getting the compiler set up, similar changes to expectations of how many cores the system has in the evaluation will be required even with Docker. To compile Polygeist, one must first compile several of its dependencies. We ran our experiments on an AWS c5.metal instance based on Ubuntu 20.04. We've tailored our build instructions to such a system. While many of the instructions are general and independent of machine, or OS, some steps may not be (and we describe what locations they may occur below). \$ sudo apt update \$ sudo apt install apt-utils \$ sudo apt install tzdata build-essential \ libtool autoconf pkg-config flex bison \ libgmp-dev clang-9 libclang-9-dev texinfo \ cmake ninja-build git texlive-full numactl # Change default compilers to make Pluto happy \$ sudo update-alternatives --install \ /usr/bin/llvm-config llvm-config \ /usr/bin/llvm-config-9 100 \$ sudo update-alternatives --install \ /usr/bin/FileCheck FileCheck-9 \ /usr/bin/FileCheck 100 \$ sudo update-alternatives --install \ /usr/bin/clang clang \ /usr/bin/clang-9 100 \$ sudo update-alternatives --install \ /usr/bin/clang++ clang++ \ /usr/bin/clang++-9 100 To begin, let us download a utility repository, which will contain several scripts and other files useful for compilation and benchmarking: \$ cd \$ git clone \ <https://github.com/wsmoses/Polygeist-Script> \ scripts One can now compile and build Pluto as shown below:

我们提出了Polygeist，一种将现有C或C++代码导入MLIR的编译工作流程，并允许多面体工具（如 Pluto）对MLIR程序进行优化。这使得MLIR能够受益于数十年来在多面体编译方面的研究。我们证明了 Polygeist生成的代码在性能上与Clang相当，从而实现了针对MLIR构建的变换与现有多面体框架之间的公正比较。最后，我们展示了Polygeist所带来的优化机会，考虑了两种互补的变换：语句拆分和归约并行化。在这两种情况下，Polygeist的性能均优于最先进的多面体编译器和源到源优化器。

关于如何在硬件或软件配置与我们所使用的不同的系统上运行，这可能需要修改的评论。如预期的那样，命令描述与docker文件的内容相似。虽然docker文件无疑更方便，是设置编译器的好方法，但即便使用Docker，对于评估系统的核心数量的期望也需要类似的更改。

要编译Polygeist，首先需要编译其若干依赖项。我们在基于Ubuntu 20.04的AWS c5.metal实例上运行了实验。我们根据该系统调整了构建说明。虽然许多说明是通用的，与机器或操作系统无关，但某些步骤可能并非如此（我们将在下文中描述它们可能出现的位置）。

```
$ sudo apt update
$ sudo apt install apt-utils
$ sudo apt install tzdata build-essential \
libtool autoconf pkg-config flex bison \
libgmp-dev clang-9 libclang-9-dev texinfo \
cmake ninja-build git texlive-full numactl

# 更改默认编译器以使Pluto正常工作
$ sudo update-alternatives --install \
/usr/bin/llvm-config llvm-config \
/usr/bin/llvm-config-9 100

$ sudo update-alternatives --install \
/usr/bin/FileCheck FileCheck-9 \
/usr/bin/FileCheck 100

$ sudo update-alternatives --install \
/usr/bin/clang clang \
/usr/bin/clang-9 100

$ sudo update-alternatives --install \
/usr/bin/clang++ clang++ \
/usr/bin/clang++-9 100
```

首先，让我们下载一个实用工具库，其中将包含一些编译和基准测试所需的脚本和其他文件：

```
$ cd
$ git clone \
https://github.com/wsmoses/Polygeist-Script\ scripts
```

现在可以按照下面的步骤编译和构建Pluto：

VIII. CONCLUSION Part-2

```
$ cd $ git clone \ https://github.com/bondhugula/pluto $ cd pluto/ $ git checkout e5a039096547e0a3d34686295c
$ git submodule init $ git submodule update $ ./autogen.sh $ ./configure $ make -j nproc ext one can build
LLVM, MLIR, and the frontend by performing the following: From here, we need to modify omp.h by
copying the version from the scripts repository and replacing the version we just built. 8$ cd $ 
export OMP_FILE= find \ $HOME/mlir-clang/build -iname omp.h$ cp $HOME/scripts/omp.h $OMP_FILE Let us
now build the MLIR polyhedral analyses, along with the specific version of LLVM it requires. We shall begin by
downloading the requisite code and building its dependencies. $ cd $ git clone --recursive \
https://github.com/kumasento/polymer -b pact $ cd polymer/ $ cd llvm/ $ mkdir build $ cd build/ $ cmake ../llvm
\ -DLLVM_ENABLE_PROJECTS="llvm;clang;mlir" \ -DLLVM_TARGETS_TO_BUILD="host" \ -
DLLVM_ENABLE_ASSERTIONS=ON \ -DCMAKE_BUILD_TYPE=Release \ -
DLLVM_INSTALL_UTILS=ON \ -G Ninja $ ninja -j nproc$ ninja check-mlir We can now build the MLIR
polyhedral analyses and export the corresponding build artifacts. $ cd ~/polymer $ mkdir build $ cd
```

```

build $ export BUILD=$PWD/.. llvm/build $ cmake .. \ -DCMAKE_BUILD_TYPE=DEBUG \ -
-DMLIR_DIR=$BUILD/lib/cmake/mlir \ -DLLVM_DIR=$BUILD/lib/cmake/llvm \ -DLLVM_ENABLE_ASSERTIONS=ON \ -
DLLVM_EXTERNAL_LIT=$BUILD/bin/llvm-lit \ -G Ninja $ ninja -j nproc$ export LD_LIBRARY_PATH=\
pwd /pluto/lib:$LD_LIBRARY_PATH $ ninja check-polymer Finally, we are ready to begin benchmarking. We
begin by running a script that disables turbo boost & hyperthreading and remaining nonessential services on the
machine. The script is specific to both the number of cores on the AWS instance (all cores except the non
hyperthreaded cores on the first socket were disabled), as well as the image used (all nonessential services still
present on the image were disabled) and thus may require modification if intending to be used on a different
machine. $ cd ~/scripts/ $ sudo bash ./hyper.sh

$ cd $ git clone \ https://github.com/bondhugula/pluto $ cd pluto/ $ git checkout e5a039096547e0a3d34686295c
$ git submodule init $ git submodule update $ ./autogen.sh $ ./configure $ make -j nproc

```

接下来，可以通过执行以下操作来构建LLVM、MLIR和前端：

从这里开始，我们需要通过复制脚本库中的版本并替换我们刚刚构建的版本来修改omp.h。

```

$ cd $ export OMP_FILE= find \ $HOME/mlir-clang/build -iname omp.h
$ cp $HOME/scripts/omp.h $OMP_FILE

```

现在让我们构建MLIR多面体分析，以及它所需的特定版本的LLVM。我们将开始下载所需的代码并构建其依赖项。

```

$ cd $ git clone --recursive \ https://github.com/kumasento/polymer -b pact
$ cd polymer/ $ cd llvm/ $ mkdir build $ cd build/
$ cmake .. llvm \ -DLLVM_ENABLE_PROJECTS="llvm;clang;mlir" \ -
DLLVM_TARGETS_TO_BUILD="host" \ -DLLVM_ENABLE_ASSERTIONS=ON \ -
DCMAKE_BUILD_TYPE=Release \ -DLLVM_INSTALL_UTILS=ON \ -G Ninja
$ ninja -j nproc
$ ninja check-mlir

```

现在我们可以构建MLIR多面体分析并导出相应的构建产物。

```

$ cd ~/polymer $ mkdir build $ cd build $ export BUILD=$PWD/.. llvm/build
$ cmake .. \ -DCMAKE_BUILD_TYPE=DEBUG \ -DMLIR_DIR=$BUILD/lib/cmake/mlir \ -
DLLVM_DIR=$BUILD/lib/cmake/llvm \ -DLLVM_ENABLE_ASSERTIONS=ON \ -
DLLVM_EXTERNAL_LIT=$BUILD/bin/llvm-lit \ -G Ninja
$ ninja -j nproc
$ export LD_LIBRARY_PATH=\ pwd /pluto/lib:$LD_LIBRARY_PATH
$ ninja check-polymer

```

最后，我们准备开始基准测试。我们首先运行一个脚本，禁用涡轮提升和超线程，并关闭机器上仍然不必要的服务。该脚本特定于AWS实例上的核心数量（所有核心中除第一个插槽上的非超线程核心外均被禁用），以及使用的镜像（镜像上仍存在的所有不必要的服务被禁用），因此如果打算在其他机器上使用，可能需要进行修改。

```
$ cd ~/scripts/ $ sudo bash ./hyper.sh
```

VIII. CONCLUSION Part-3

We can now run the benchmarking script. The script itself has assumptions about cores and layout (setting taskset -c 1-8 numactl -i all for example). If using a different machine, these settings may need to be tweaked as appropriate. cd ~/scripts/ \$ cd polybench-c-4.2.1-beta/ \$./run.sh # Output comes through stdout The output of this script will contain the runtime of each trial, describing what compilation setting was used, as well as which benchmark was run.

我们现在可以运行基准测试脚本。该脚本本身对核心和布局有一些假设（例如，设置 taskset -c 1-8 numactl -i all）。如果使用不同的机器，可能需要根据实际情况调整这些设置。cd ~/scripts/ \$ cd polybench-c-4.2.1-beta/ \$./run.sh # 输出通过标准输出生成 该脚本的输出将包含每次试验的运行时间，描述所使用的编译设置以及运行了哪个基准测试。

#

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

本文件中包含的观点和结论是作者的观点，不应被解读为美国空军或美国政府的官方政策，无论是明示还是暗示。美国政府被授权为了政府目的复印和分发该文件的印刷本，尽管这里有任何版权说明。

APPENDIX

In this artifact appendix, we describe how to build Polygeist and evaluate its performance (as well as baseline compilers) on the Polybench benchmark suite. We provide two mechanisms for artifact evaluation: a Docker container 7 , and a command-by-command description of the installation process, along with

在本附录中，我们描述了如何构建Polygeist并评估其在Polybench基准测试套件上的性能（以及基准编译器）。我们提供了两种艺术品评估机制：一个Docker容器以及逐步命令描述的安装过程，连同

Polygeist: Raising C to Polyhedral MLIR

Published in: 2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)

发表于: 2021 年第 30 届国际并行架构与编译技术会议 (PACT)

Abstract: 摘要:

We present Polygeist, a new compilation flow that connects the MLIR compiler infrastructure to cutting edge polyhedral optimization tools. It consists of a C and C++ frontend capable of converting a broad range of existing codes into MLIR suitable for polyhedral transformation and a bi-directional conversion between MLIR and OpenScop exchange format. The Polygeist/MLIR intermediate representation featuring high-level (affine) loop constructs and n-D arrays embedded into a single static assignment (SSA) substrate enables an unprecedented combination of SSA-based and polyhedral optimizations. We illustrate this by proposing and implementing two extra transformations: statement splitting and reduction parallelization. Our evaluation demonstrates that Polygeist outperforms on average both an LLVM IR-level optimizer (Polly) and a source-to-source state-of-the-art polyhedral compiler (Pluto) when exercised on the Polybench/C benchmark suite in sequential (2.53x vs 1.41x, 2.34x) and parallel mode (9.47x vs 3.26x, 7.54x) thanks to the new representation and transformations.

我们介绍 Polygeist，这是一种新的编译流程，它将 MLIR 编译器基础设施与前沿的多面体优化工具连接起来。它包括一个 C 和 C++ 前端，能够将广泛现有的代码转换为适用于多面体变换的 MLIR，并实现 MLIR 与 OpenScop 交换格式的双向转换。具有高级（仿射）循环构造和嵌入到单个静态赋值（SSA）基底的 Polygeist/MLIR 中间表示，实现了基于 SSA 和多面体优化的前所未有的组合。我们通过提出并实现两个额外的转换来展示这一点：语句拆分和减少并行化。我们的评估表明，在 Polybench/C 基准测试套件上，Polygeist 在顺序（2.53 倍 vs 1.41 倍，2.34 倍）和并行模式（9.47 倍 vs 3.26 倍，7.54 倍）上平均优于 LLVM IR 级别的优化器（Polly）和源到源的前沿多面体编译器（Pluto），这得益于新的表示和转换。

[William S. Moses](#)

[Polygeist: Affine C in MLIR \[MLIR Open Design Meeting 02/11/2021\]](#)

<https://www.youtube.com/@billymoses7764>

[getting_started/Use_Polygeist](#)

[Retargeting and Respecializing GPU Workloads for Performance Portability](#)

重新定位和针对性能可移植性重新专业化的 GPU 工作负载

In order to come close to peak performance, accelerators like GPUs require significant architecture-specific tuning that understand the availability of shared memory, parallelism, tensor cores, etc. Unfortunately, the pursuit of higher performance and lower costs have led to a significant diversification of architecture designs, even from the same vendor. This creates the need for performance portability across different GPUs, especially important for programs in a particular programming model with a certain architecture in mind. Even when the program can be seamlessly executed on a different architecture, it may suffer a performance penalty due to it not being sized appropriately to the available hardware resources such as fast memory and registers, let alone not using newer advanced features of the architecture. We propose a new approach to improving performance of (legacy) CUDA programs for modern machines by automatically adjusting the amount of work each parallel thread does, and the amount of memory and register resources it requires. By operating within the MLIR compiler infrastructure, we are able to also target AMD GPUs by performing automatic translation from CUDA and simultaneously adjust the program granularity to fit the size of target GPUs. Combined with autotuning assisted by the platform-specific

compiler, our approach demonstrates 27% geomean speedup on the Rodinia benchmark suite over baseline CUDA implementation as well as performance parity between similar NVIDIA and AMD GPUs executing the same CUDA program.

为了接近峰值性能，像 GPU 这样的加速器需要针对特定架构进行显著的调整，这些调整理解共享内存、并行性、张量核心等的可用性。不幸的是，追求更高的性能和更低的成本导致了架构设计的显著多样化，即使是来自同一供应商的产品也是如此。这产生了在不同 GPU 之间实现性能可移植性的需求，这对于特定编程模型和特定架构的程序尤为重要。即使程序可以在不同的架构上无缝执行，它也可能因为未适当调整以适应可用的硬件资源（如快速内存和寄存器）而遭受性能损失，更不用说没有使用架构的新先进特性。我们提出了一种新方法，通过自动调整每个并行线程执行的工作量以及它所需的内存和寄存器资源，来提高（遗留）CUDA 程序在现代机器上的性能。通过在 MLIR 编译器基础设施中操作，我们能够通过自动从 CUDA 进行翻译来针对 AMD GPU 进行目标定位，同时调整程序粒度以适应目标 GPU 的大小。结合平台特定编译器辅助的自动调整，我们的方法在 Rodinia 基准测试套件上相对于基线 CUDA 实现实现了 27% 的几何平均速度提升，以及执行相同 CUDA 程序时类似 NVIDIA 和 AMD GPU 之间的性能对等。

Frontend Performance Differences

- 8% performance boost on Floyd-Warshall occurs if Polygeist generates a single MLIR module for both benchmarking and timing code by default
- MLIR doesn't properly generate LLVM datalayout, preventing vectorization for MLIR-generated code (patched in our lowering)
- Different choice of allocation function can make a 30% impact on some tests (adi)
- LLVM strength-reduction is fragile and sometimes misses reversed loop induction variable (remaining gap in adi)
- 如果 Polygeist 默认为基准测试和计时代码生成单个 MLIR 模块，则 Floyd-Warshall 的性能将提升 8%
- MLIR 无法正确生成 LLVM 数据布局，从而阻止了 MLIR 生成的代码的矢量化（在我们的降低版本中进行了修补）
- 不同的分配函数选择可能会对某些测试（adi）产生 30% 的影响
- LLVM 强度降低很脆弱，有时会错过反向环感应变量（ADI 中的剩余间隙）

Polgeist MLIR Compiler Frontend

Polygeist 的核心功能 Polygeist 的主要目标是 bridging the gap between C/C++ and MLIR。它具有以下核心功能：

C/C++前端:能够解析和分析广泛的 C 和 C++ 代码。MLIR 生成:将 C/C++ 代码转换为适合多面体变换的 MLIR 表示。多面体优化:利用 MLIR 的多面体优化能力进行高级循环优化。并行优化:支持自动并行化和并行构造的优化。GPU 后端支持:包括 CUDA 和 ROCm 后端，实现 GPU 加速。这些功能使 Polygeist 成为连接传统 C/C++ 代码和现代 MLIR 编译架构的关键工具。

Polygeist 的工作原理 Polygeist 的工作流程可以简要概括为以下几个步骤：

解析 C/C++ 代码:使用 Clang 的前端能力解析输入的 C/C++ 代码。AST 分析:对抽象语法树(AST)进行深入分析，提取程序的结构和语义信息。MLIR 生成:基于 AST 分析结果，生成对应的 MLIR 表示。多面体建模:将 MLIR 表示转换为多面体模型，为后续优化铺平道路。优化应用:应用多面体优化、并行优化等高级优化技术。代码生成:将优化后的 MLIR 转换回 LLVM IR 或直接生成目标代码。通过这一系列步骤，Polygeist 能够充分利用 MLIR 的强大功能，同时保持对原始 C/C++ 代码的兼容性。

Polygeist 的优势与应用 Polygeist 为 C/C++程序带来了诸多优势:

高级优化:通过多面体模型,可以进行更复杂和有效的循环优化。并行化:自动检测和利用并行机会,提高程序性能。可移植性:通过 MLIR 表示,可以更容易地将程序移植到不同的硬件平台。GPU 加速:内置的 CUDA 和 ROCm 后端支持,简化 GPU 编程。与 LLVM 生态系统集成:作为 LLVM 项目的一部分,可以无缝集成到现有的 LLVM 工具链中。这些优势使 Polygeist 在高性能计算、科学计算、机器学习等领域具有广泛的应用前景。

实际应用案例 以下是 Polygeist 在实际项目中的应用案例:

科学计算优化:在一个大规模数值模拟项目中,使用 Polygeist 对核心计算 kernel 进行优化,通过多面体变换和自动并行化,性能提升了 30%。

机器学习框架:某开源机器学习框架使用 Polygeist 优化其 C++后端,实现了更高效的张量运算,在某些模型上推理速度提升了 20%。

图形渲染引擎:一个游戏引擎项目利用 Polygeist 的 GPU 后端支持,简化了 CUDA 代码的生成过程,大大提高了开发效率。

这些案例展示了 Polygeist 在提升程序性能和简化开发流程方面的巨大潜力。

```
cgeist input.c -S -emit-mlir | mlir-opt --canonicalize --cse > output.mlir
```

2022 LLVMHPC William S. Moses, Polygeist: C++ Frontend for MLIR

[text](#)

The MLIR Framework

- MLIR is a recent compiler infrastructure designed for reuse and extensibility
- Rather than providing a predefined set of instructions and types, MLIR operates on collections of dialects that contain sets of interoperable user-defined operations, attributes and types
- Anyone can define their own optimizable dialect/operation, with a large set of existing dialects (structured control flow, affine, GPU, quantum, fully homomorphic encryption, circuits, LLVM, and more!)
- MLIR 是一种为重用和可扩展性而设计的最新编译器基础架构
- MLIR 不是提供一组预定义的指令和类型,而是对包含一组可互操作的用户定义操作、属性和类型的方言集合进行操作
- 任何人都可以使用大量现有方言(结构化控制流、仿射、GPU、量子、完全同态加密、电路、LLVM 等!)定义自己可优化的方言/操作。

The Polyhedral Model

- Represent programs as a collection of computations and constraints on a multi-dimensional grid (polyhedron)
- Makes it easy to analyze and specify program transformations best exploit the available hardware
- Loop restructuring for spatial/temporal locality, automatic parallelization, etc.
- One of the best frameworks for optimizing compute-intensive programs like machine learning kernels or scientific simulations as well as for programming accelerators.

Preserve the parallel structure

- Maintain GPU parallelism in a form understandable to the compiler
- Enables optimization between caller and kernel
- Enable parallelism-specific optimization
- 将程序表示为多维网格(多面体)上的计算和约束集合
- 便于分析和指定最佳利用可用硬件的程序转换
- 用于空间/时间局部性、自动并行化等的循环重构
- 优化机器学习内核或科学模拟等计算密集型程序以及编程加速器的最佳框架之一。

Synchronization via Memory

- Synchronization (sync_threads) ensures all threads within a block finish executing codeA before executing codeB
- The desired synchronization behavior can be reproduced by defining sync_threads to have the union of the memory semantics of the code before and after the sync.
- This prevents code motion of instructions which require the synchronization for correctness, but permits other code motion (e.g. index computation).
- 同步 (sync_threads) 确保块中的所有线程在执行 CodeB 之前完成对 CodeA 的执行
- 可以通过定义 sync_threads 来重现所需的同步行为，以便在同步之前和之后具有代码的内存语义的并集。
- 这可以防止需要同步以确保正确性的指令的代码移动，但允许其他代码移动（例如索引计算）。
- High-level synchronization representation enables new optimizations, like sync elimination.
- A synchronize instruction is not needed if the set of read/writes before the sync don't conflict with the read/writes after the sync.
- 高级同步表示支持新的优化，例如同步消除。
- 如果同步前的读/写集与同步后的读/写集不冲突，则不需要 synchronize 指令。

```

__global__ void bpnn_layerforward(...) {
    __shared__ float node[HEIGHT];
    __shared__ float weights[HEIGHT][WIDTH];

    if ( tx == 0 )
        node[ty] = input[index_in] ;

    // Unnecessary Barrier #1
    // None of the read/writes below the sync
    // (weights, hidden)
    // intersect with the read/writes above the sync
    // (node, input)
    __syncthreads();

    // Unnecessary Store #1
    weights[ty][tx] = hidden[index];

    __syncthreads();
    // Unnecessary Load #1
    weights[ty][tx] = weights[ty][tx] * node[ty];
    // ...
}

}

```

GPU Transpilation

- A unified representation of parallelism enables programs in one parallel architecture (e.g. CUDA) to be compiled to another (e.g. CPU/OpenMP)
- **Most CPU backends do not have an equivalent block synchronization**
- Efficiently lower a top-level synchronization by distributing the parallel for loop around the sync, and interchanging control flow

```

parallel_for %i = 0 to N {
    codeA(%i);
    sync_threads;
    codeB(%i);
}
; =>
parallel_for %i = 0 to N {
    codeA(%i);
}

parallel_for %i = 0 to N {
    codeB(%i);
}

```

GPU Synchronization Lowering: Control Flow

Synchronization within control flow (for, if, while, etc) can be lowered by splitting around the control flop and interchanging the parallelism.

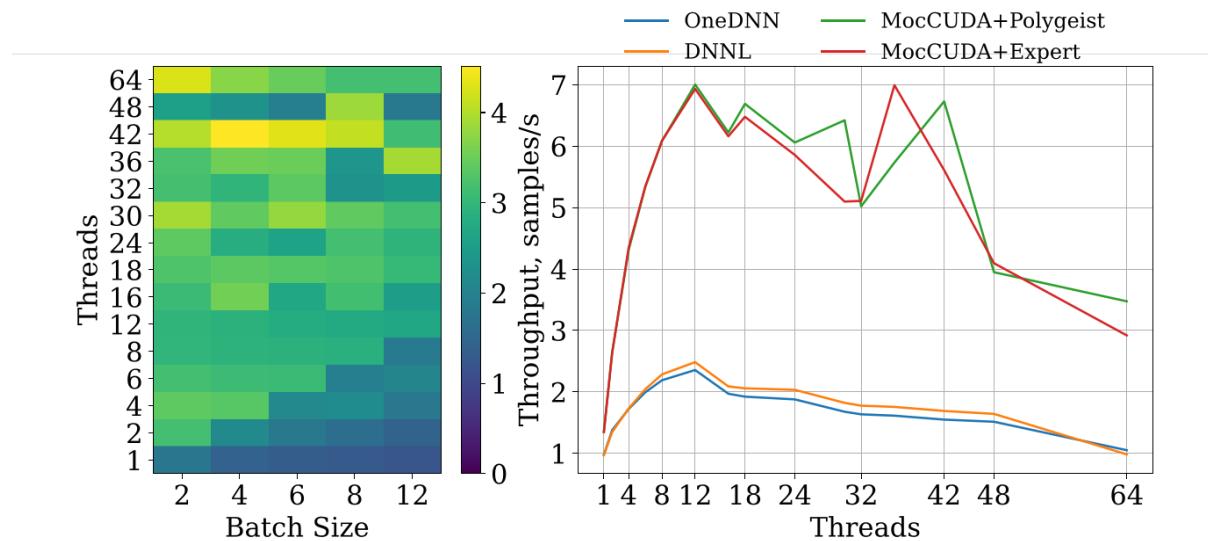
```

parallel_for %i = 0 to N {
    for %j = ... {
        codeB1(%i, %j);
        sync_threads;
        codeB2(%i, %j);
    }
}
; Interchange =>
for %j = ... {
    parallel_for %i = 0 to N {
        codeB1(%i, %j);
        sync_threads;
        codeB2(%i, %j);
    }
}
; Split =>
for %j = ... {
    parallel_for %i = 0 to N {
        codeB1(%i, %j);
    }
    parallel_for %i = 0 to N {
        codeB2(%i, %j);
    }
}

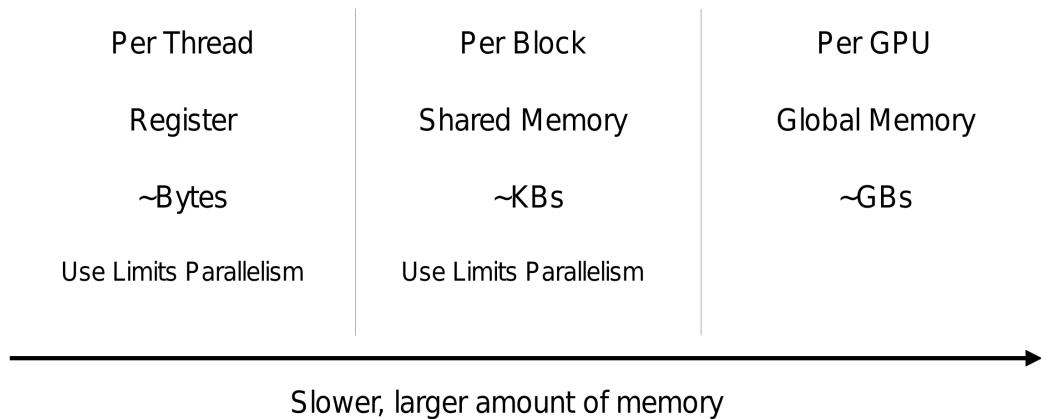
```

GPU Transpilation Performance

- CUDA programs transcompiled by Polygeist not only match the performance of handwritten OpenMP programs, but achieve a speedup!
 - 58% geomean speedup on Rodinia
 - 2.7x geomean speedup on PyTorch versus built-in CPU backend (also using our MocCUDA compatibility layer)



GPU Memory Hierarchy



Affine transformation

[text](#)

[text](#)

[text](#)

In Euclidean geometry, an affine transformation or affinity (from the Latin, *affinis*, "connected with") is a geometric transformation that preserves lines and parallelism, but not necessarily Euclidean distances and angles.

在欧几里得几何中，仿射变换或亲和力（来自拉丁语 *affinis*, “连接”）是一种几何变换，它保留线条和平行度，但不一定是欧几里得距离和角度。

More generally, an affine transformation is an automorphism of an affine space (Euclidean spaces are specific affine spaces), that is, a function which maps an affine space onto itself while preserving both the dimension of any affine subspaces (meaning that it sends points to points, lines to lines, planes to planes, and so on) and the ratios of the lengths of parallel line segments. Consequently, sets of parallel affine subspaces remain parallel after an affine transformation. An affine transformation does not necessarily preserve angles between lines or distances between points, though it does preserve ratios of distances between points lying on a straight line.

更一般地说，仿射变换是仿射空间的自同态（欧几里得空间是特定的仿射空间），即一个函数，它将仿射空间映射到自身，同时保留任何仿射子空间的维度（意味着它将点发送到点、线到线、平面到平面等）和平行长度的比率线段。因此，并行仿射子空间集在仿射变换后保持平行。仿射变换不一定保留线条之间的角度或点之间的距离，尽管它确实保留了位于直线上的点之间的距离比率。

If X is the point set of an affine space, then every affine transformation on X can be represented as the composition of a linear transformation on X and a translation of X . Unlike a purely linear transformation, an affine transformation need not preserve the origin of the affine space. Thus, every linear transformation is affine, but not every affine transformation is linear.

如果 X 是仿射空间的点集，则 X 上的每个仿射变换都可以表示为 X 上的线性变换和 X 的平移的组合。与纯线性变换不同，仿射变换不需要保留仿射空间的原点。因此，每个线性变换都是仿射变换，但并非每个仿射变换都是线性变换。

Examples of affine transformations include translation, scaling, homothety, similarity, reflection, rotation, hyperbolic rotation, shear mapping, and compositions of them in any combination and sequence.

仿射变换的示例包括平移、缩放、同质性、相似性、反射、旋转、双曲旋转、剪切映射以及它们以任意组合和顺序的组合。

Viewing an affine space as the complement of a hyperplane at infinity of a projective space, the affine transformations are the projective transformations of that projective space that leave the hyperplane at infinity invariant, restricted to the complement of that hyperplane.

将仿射空间视为投影空间无穷远处的超平面的补码，仿射变换是该射影空间的投影变换，它使无穷远处的超平面保持不变，仅限于该超平面的补码。

A generalization of an affine transformation is an affine map[1] (or affine homomorphism or affine mapping) between two (potentially different) affine spaces over the same field k . Let (X, V, k) and (Z, W, k) be two affine spaces with X and Z the point sets and V and W the respective associated vector spaces over the field k . A map $f : X \rightarrow Z$ is an affine map if there exists a linear map $mf : V \rightarrow W$ such that $mf(x - y) = f(x) - f(y)$ for all x, y in X .[2]

仿射变换的泛化是同一域 k 上两个（可能不同的）仿射空间之间的仿射映射1（或仿射同态或仿射映射）。设 (X, V, k) 和 (Z, W, k) 是两个仿射空间，其中 X 和 Z 是点集， V 和 W 是场 k 上各自的关联向量空间。如果存在线性映射 $mf: V \rightarrow W$ ，使得 $xf(x-y) = f(x) - f(y)$ 对于 X, Z 中的所有 $x, y \in Z$ 是仿射映射

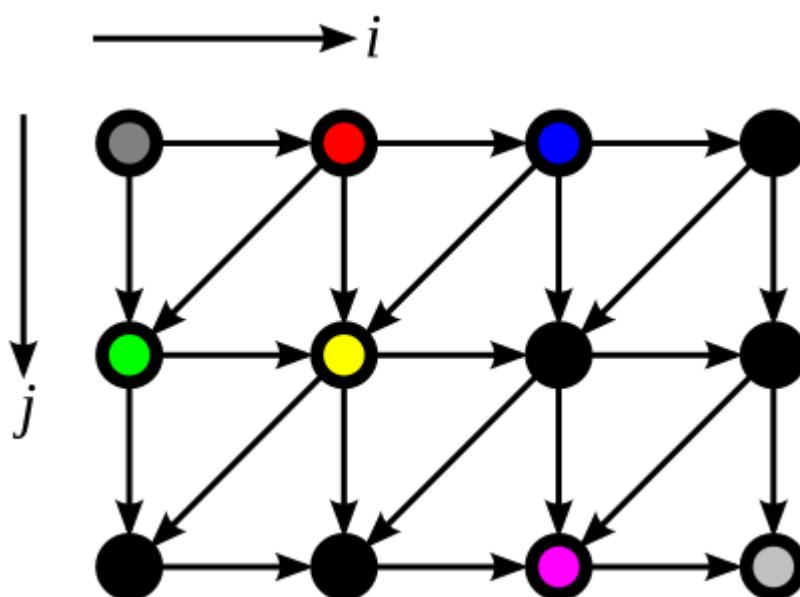
Polyhedral Model

[cs6120-Advanced Compilers text](#)

[text](#)

The polyhedral model (also called the polytope method) is a mathematical framework for programs that perform large numbers of operations -- too large to be explicitly enumerated -- thereby requiring a compact representation. Nested loop programs are the typical, but not the only example, and the most common use of the model is for loop nest optimization in program optimization. The polyhedral method treats each loop iteration within nested loops as lattice points inside mathematical objects called polyhedra, performs affine transformations or more general non-affine transformations such as tiling on the polytopes, and then converts the transformed polytopes into equivalent, but optimized (depending on targeted optimization goal), loop nests through polyhedra scanning.

多面体模型（也称为多面体方法）是一个数学框架，用于执行大量运算的程序 -- 太大而无法显式列举 -- 因此需要紧凑的表示。嵌套循环程序是典型的但不是唯一的例子，该模型最常见的用途是程序优化中的循环嵌套优化。多面体方法将嵌套循环中的每个循环迭代视为称为多面体的数学对象内的格子点，执行射影变换或更通用的非射影变换，例如在多面体上平铺，然后通过多面体扫描将转换后的多面体转换为等效但优化（取决于目标优化目标）的循环嵌套。



Polyhedral model in programming

[Frameworks supporting the polyhedral model](#)

多面体模型及其在循环编译优化中的应用

多面体模型是一个用于表示和优化程序的数学框架，尤其适用于涉及循环的程序。它是编译器技术中执行优化的强大工具，尤其在高性能计算中具有重要作用。下面我们将详细解析该概念及其在循环优化中的应用。

多面体模型概述

多面体模型是一个用于描述循环及其迭代的数学抽象，采用多维几何体（即多面体）来表示循环的迭代空间。通过对这些多面体的优化，可以提升程序的执行效率。

在多面体模型中，循环通过以下关键要素进行表示：

1. **循环边界**: 定义循环变量可能取的范围。
2. **依赖关系**: 描述循环迭代之间的关系，特别是数据依赖（如读后写或写后读），这些关系必须在转换过程中保持一致。
3. **迭代空间**: 表示所有可能的循环迭代的多维空间，通常以多面体形式可视化，每个点代表一个循环的迭代。

多面体模型的工作原理

1. **循环表示**: 在多面体模型中，循环通过一组数学不等式表示，这些不等式定义了循环变量的边界。例如，嵌套循环可以用一组不等式来描述，这些不等式构成一个多面体，其中每个点对应循环的一个迭代。
2. **依赖分析**: 在循环程序中，不同迭代之间可能会存在数据依赖。多面体模型可以帮助正式分析这些依赖，判断哪些迭代可以并行执行，哪些需要按照特定顺序执行以保证程序正确性。
3. **转换操作**: 多面体模型允许对循环进行各种转换，例如：
 - **循环融合**: 将两个循环合并为一个循环，以提高数据局部性。
 - **循环分裂**: 将一个循环拆分为两个，以提高并行性或数据局部性。
 - **循环块化（Tiling）**: 将循环迭代空间分成更小的块（tile），以提高缓存使用效率和局部性。
 - **循环交换**: 改变嵌套循环的执行顺序，以改善内存访问模式或并行性。

多面体模型在循环编译优化中的应用

1. **并行化**: 多面体模型可以帮助识别并行执行的机会。通过分析迭代空间和数据依赖关系，模型能够判断哪些循环迭代可以安全地并行执行，这对于多核处理器的利用至关重要。
2. **缓存优化**: 多面体模型通过转换循环来优化内存访问模式，提升数据局部性。例如，通过应用循环块化或分块技术，模型确保多个循环迭代访问的数据可以适配到缓存中，从而减少内存延迟。
3. **矢量化**: 多面体模型可以帮助判断哪些循环部分可以进行矢量化，从而更好地利用现代处理器的 SIMD（单指令多数据）指令集。
4. **乱序执行**: 通过分析循环依赖关系，多面体模型可以帮助编译器确定哪些迭代可以乱序执行，从而提高指令级并行性，并在现代处理器上获得更好的性能。
5. **自动并行化**: 该模型在自动并行化编译器中扮演重要角色，通过分析循环结构和依赖关系，它可以决定哪些循环或循环部分可以安全并行化。
6. **针对特定架构的优化**: 多面体模型可以生成针对特定硬件的优化代码，如 GPU，通过应用适合该架构内存层次结构和处理能力的转换来提升性能。

多面体模型的优点

- **数学严谨性**: 多面体模型提供了一种精确且正式的方式来描述循环，使编译器可以更容易地推理循环优化。
- **表达能力强**: 它能够在一个统一的框架中表达多种循环优化，如并行化、局部性优化和矢量化。

- **自动化:** 它能够自动优化代码，减少手动调整的需求。

多面体模型的挑战

- **复杂性:** 多面体模型数学上较为复杂，在编译器中实现起来可能具有一定难度，尤其是在处理具有复杂依赖关系的程序时。
- **可扩展性:** 对于大规模应用程序，管理多面体表示的开销可能会非常大，尤其是在内存和计算上。

示例

考虑一个简单的嵌套循环：

```
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        A[i][j] = B[i][j] + C[i][j];
    }
}
```

在多面体模型中，`i` 和 `j` 的迭代空间被表示为一个二维多面体。依赖分析表明，内循环的每次迭代依赖于前几次迭代的数据。基于此，模型可以应用循环融合或块化等优化技术，从而提高缓存使用效率或并行性。

结论

多面体模型是一个用于优化循环的复杂工具。通过将循环表示为多面体并分析其依赖关系，模型能够实现包括并行化、循环转换（如块化、融合等）和内存优化在内的多种优化。尽管它的实现较为复杂，但它已成为高性能编译技术的基础，尤其在科学计算和高性能应用中具有重要意义。

The **Polyhedral Model** is a mathematical framework used to represent and optimize programs, particularly those involving loops. It is a powerful tool for performing optimization in compiler technology, especially in high-performance computing. Let's break down the concept and its application to loop optimization.

Polyhedral Model Overview

The Polyhedral Model is a mathematical abstraction used to describe loops and their iterations in terms of multi-dimensional shapes, or polyhedra. These polyhedra represent the iteration spaces of loops, and their optimization helps improve the performance of programs.

In the polyhedral model, loops are represented using the following key components:

1. **Loop Bounds:** The range of indices that a loop variable can take.
2. **Dependences:** The relationships between iterations of loops, specifically data dependencies (such as read-after-write or write-after-read), which must be respected during transformations.
3. **Iteration Spaces:** The multi-dimensional space that represents all possible iterations of the loops. These are often visualized as polyhedra (multi-dimensional geometric objects) where each point corresponds to an iteration of the loop.

How the Polyhedral Model Works

1. **Representation of Loops:** In the polyhedral model, loops are represented in a high-level mathematical form. For example, a nested loop can be described by a set of inequalities that define the bounds of the loop's variables. This set of inequalities is a polyhedron, where each point corresponds to an iteration of the loop.
2. **Dependence Analysis:** In loop programs, data can be dependent across different iterations. The polyhedral model enables the formal analysis of such dependencies, helping identify if iterations can be executed in parallel or need to be executed in a specific order to preserve correctness.
3. **Transformations:** The polyhedral model allows for a variety of transformations on the loops, such as:
 - **Loop Fusion:** Combining two loops into one to improve data locality.
 - **Loop Fission:** Splitting a loop into two to enhance parallelism or data locality.
 - **Loop Tiling (Blocking):** Dividing the loop iteration space into smaller blocks (tiles) to improve cache usage and locality.
 - **Loop Interchange:** Changing the order of nested loops to improve memory access patterns or parallelism.

Applications in Loop Compilation and Optimization

1. **Parallelism:** The polyhedral model helps identify opportunities for parallel execution. By analyzing the iteration space and data dependencies, it can determine whether iterations can be safely executed in parallel, which is crucial for leveraging multi-core processors.
2. **Cache Optimization:** The polyhedral model can help optimize memory access patterns by transforming loops to improve data locality. For example, by applying loop blocking or tiling, the model can ensure that data accessed by multiple iterations of the loop fit within the cache, reducing memory latency.
3. **Vectorization:** The polyhedral model aids in determining which parts of the loop can be vectorized, allowing for better utilization of SIMD (Single Instruction, Multiple Data) instructions on modern processors.
4. **Out-of-Order Execution:** By analyzing loop dependencies, the polyhedral model helps compilers determine which iterations can be executed out of order, improving instruction-level parallelism and performance on modern processors.
5. **Automatic Parallelization:** The model plays a crucial role in compilers that automatically parallelize code. By analyzing the loop structure and dependencies, it can decide which loops or parts of loops can be parallelized safely.
6. **Optimization for Specific Architectures:** The polyhedral model can be used to generate optimized code for specific hardware, such as GPUs, by applying transformations that are well-suited for the architecture's memory hierarchy and processing capabilities.

Advantages of the Polyhedral Model

- **Mathematical Rigor:** The polyhedral model provides a formal and precise way to describe loops, making it easier for compilers to reason about loop optimizations.
- **Expressive Power:** It can express a wide range of loop optimizations, such as parallelism, locality, and vectorization, in a unified framework.
- **Automation:** It enables automatic optimization of code, reducing the need for manual tuning.

Challenges of the Polyhedral Model

- **Complexity:** The polyhedral model is mathematically complex and can be difficult to implement in compilers, especially for very large programs with intricate dependencies.
- **Scalability:** For large-scale applications, the overhead of managing polyhedral representations can be significant, especially in terms of memory and computation.

Example

Consider a simple nested loop:

```
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        A[i][j] = B[i][j] + C[i][j];
    }
}
```

The polyhedral model represents the iteration space of `i` and `j` as a 2D polyhedron. Dependence analysis would show that each iteration of the inner loop depends on data from the previous iterations. Based on this, the polyhedral model can apply optimizations like loop fusion or tiling to improve cache usage or parallelism.

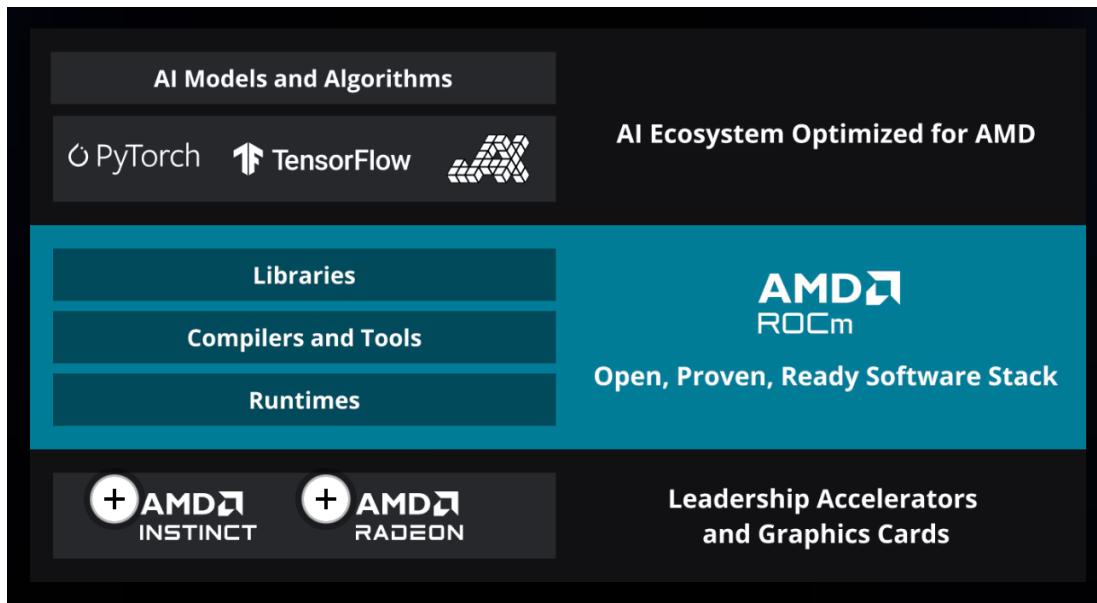
Conclusion

The **Polyhedral Model** is a sophisticated tool used in compilers for optimizing loops. By representing loops as polyhedra and analyzing their dependencies, it allows for a range of optimizations, including parallelization, loop transformations (tiling, fusion, etc.), and memory optimizations. Although it can be complex to implement, it has become a foundational technique for high-performance compilation, especially in scientific computing and high-performance applications.

Optimized GPU Software Stack 优化的 GPU 软件堆栈

AMD ROCm™ is an open software stack including drivers, development tools, and APIs that enable GPU programming from low-level kernel to end-user applications. ROCm is optimized for Generative AI and HPC applications, and is easy to migrate existing code into.

AMD ROCm™ 是一个开放软件堆栈，包括驱动程序、开发工具和 API，支持从低级内核到最终用户应用程序的 GPU 编程。ROCm 针对生成式 AI 和 HPC 应用程序进行了优化，并且易于将现有代码迁移到其中。



ROCm 3 是用于图形处理单元(GPU) 编程的Advanced Micro Devices (AMD) 软件堆栈。 ROCm 跨越多个领域：图形处理单元上的通用计算(GPGPU)、高性能计算(HPC)、异构计算。它提供了多种编程模型：HIP（基于 GPU 内核的编程） 、 OpenMP （基于指令的编程） 和OpenCL 。

[wiki/ROCM](#)

[AMD ROCm documentation](#)

ROS Packages

Package	Purpose
learn_turtlesim	<i>You gotta start somewhere.</i>
learn_arduino	<i>Poking at the real world.</i>
learn_rviz	<i>Static visualisations of robot models.</i>
learn_tf	<i>Dynamic visualisations of robot models.</i>
learn_joy	<i>Interactive visualisations of robot models.</i>
learn_imu	<i>WIP - Read and display data from an IMU.</i>
learn_webcam	<i>Stream video from a USB webcam.</i>
learn_kinect	<i>Stream video and more from an Xbox 360 Kinect.</i>
jaws_description	<i>The first Jaws modelled as a URDF.</i>
jaws_visualization	<i>WIP - A visual demonstration of Jaws.</i>

ROS: Robot Operating System

- index.ros.org/
- [wiki.ros: Tutorials](http://wiki.ros.org/Tutorials)
- [wiki.ros: cn Introduction](http://wiki.ros.org/cn/Introduction)
- [github: ros-infrastructure](https://github.com/ros-infrastructure)
- [github: ROS core stacks](https://github.com/ROS-Industrial/ROS_core_stacks)
- [github: ros-dpg](https://github.com/ROS-Industrial/ros-dpg)
- [rep: learn-ros](http://rep.ros.org/learn-ros.html)
- [blog: ros-tutorials](http://blog.ros-tutorials.org/)

ROS（机器人操作系统）提供库和工具来帮助软件开发人员创建机器人应用程序。它提供硬件抽象、设备驱动程序、库、可视化工具、消息传递、包管理等。

ROS Releases/Distributions

- [ros: Distributions](http://ros.org/reps/rep-0001.html#distributions)
- [ros2: Releases](http://ros.org/reps/rep-0001.html#releases)
- [ros - os version match ROS:](http://ros.org/reps/rep-0001.html#os-version-match-ros)
- ROS Box Turtle 2010.03.02
- ROS C Turtle 2010.08.02
- ROS Diamondback 2011.03.01
- ROS Electric Emys 2011.08.30
- ROS Fuerte Turtle 2012.04.23
- ROS Groovy Galapagos 2012.12.31
- ROS Hydro Medusa 2013.09.04
- ROS Indigo Igloo 2014.07.22
- ROS Jade Turtle 2015.05.23
- ROS Kinetic Kame 2016.05.23
- ROS Lunar Loggerhead 2017.05.23
- ROS Melodic Morenia 2018.05.23
- ROS Noetic Ninjemys 2020.05.23

ROS2:

- ROS 2 Ardent Apalone 2017.12.08
- ROS 2 Bouncy Bolson 2018.05.31
- ROS 2 Crystal Clemmys 2018.12.12
- ROS 2 Dashing Diademata 2019.05.31
- ROS 2 Eloquent Elusor 2019.12.12
- ROS 2 Foxy Fitzroy 2020.06.05
- ROS 2 Galactic Geochelone 2021.05.23
- ROS 2 Humble Hawksbill 2022.05.23
- ROS 2 Iron Irwini 2023.05.23
- ROS 2 Jazzy Jalisco 2024.05.23

There is a new ROS 2 distribution released yearly on May 23rd (World Turtle Day).

ROS Noetic Ninjemys

- [wiki](#)

ROS Noetic Ninjemys is the thirteenth ROS distribution release. It was released on May 23rd, 2020.

Concepts

- REP: ROS Enhancement Proposals
 - REPs are documents that define standards, conventions, and best practices for the ROS ecosystem.
They are similar to RFCs (Request for Comments) in the internet protocol community or PEPs (Python Enhancement Proposals) in the Python community.
 - Filesystem Level
 - Packages
 - Metapackages
 - Package Manifests
 - Repositories
 - Message (msg) types: Message Description, stored in `my_package/msg/MyMessageType.msg`
 - Service (srv) types: Service Description, stored in `my_package/srv/MyServiceType.srv`
 - Graph Level
 - Nodes: process that performs computation
 - Master: provides name registration and lookup to the rest of the Computation Graph
 - Parameter Server: allows data to be stored by key in a central location
 - Messages: data structure for communication
 - Topics: messages are routed via a transport system with publish/subscribe semantics
 - a node sends a message by publishing it to a given topic
 - a node receives a message by subscribing to the appropriate topic
 - Services: request/reply is done via a service
 - a node offers a service under a specific name
 - a client uses the service by sending the request message and awaiting the reply
 - Bags: a format for saving and playing back ROS message data
 - mechanism for storing ROS message data, such as sensor data
 - Community Level
 - Distributions
 - Repositories
 - ROS Wiki
 - ...
 - names: Package Resource Names and Graph Resource Names
 - Graph Resource Names:
 - provides a hierarchical naming structure that is used for all resources in ROS Computation Graph
 - Graph Resource Names are an important mechanism in ROS for providing `encapsulation`.
 - Each resource is defined within a namespace, which it may share with many other resources.
 - In general, resources can create resources within their namespace and they can access resources within or above their own namespace.
 - Connections can be made between resources in distinct namespaces, but this is generally done by integration code above both namespaces.
-

- This encapsulation isolates different portions of the system from accidentally grabbing the wrong named resource or globally hijacking names.

- 每个资源都在一个命名空间中定义，该命名空间可以与许多其他资源共享。
- 通常，资源可以在其命名空间中创建资源，并且可以访问其自己的命名空间内或之上的资源。
- 可以在不同命名空间中的资源之间建立连接，但这通常是通过两个命名空间上方的集成代码完成的。
- 这种封装将系统的不同部分与意外获取错误的命名资源或全局劫持名称隔离开来。
- / : global namespace
- four types of Graph Resource Names:
 - base, relative, global, and private
 - base
 - relative/name
 - /global/name
 - ~private/name
- Package Resource Names
 - "std_msgs/String" refers to the "String" message type in the "std_msgs" Package.

Higher-Level Concepts

- [wiki](#)
- Coordinate Frames/Transforms
 - The `tf` package provides a distributed, ROS-based framework for calculating the positions of multiple coordinate frames over time.
- Actions/Tasks
 - The `actionlib` package defines a common, topic-based interface for preemptible tasks in ROS.
- Message Ontology
 - `common_msgs` stack provides a set of common message types for interacting with robots.
 - `actionlib_msgs` : messages for representing actions
 - `diagnostic_msgs` : messages for sending diagnostic data.
 - `geometry_msgs` : messages for representing common geometric primitives.
 - `nav_msgs` : messages for navigation.
 - `sensor_msgs` : messages for representing sensor data.
- Plugins
 - `pluginlib` package provides tools for writing and dynamically loading plugins using the ROS build system.
- Filters
 - `filters` package provides a set of filters for processing data streams.
- Robot Model
 - The `urdf` package defines an XML format for representing a robot model and provides a C++ parser.

Client Libraries

A ROS client library is a collection of code that eases the job of the ROS programmer. It takes many of the ROS concepts and makes them accessible via code. In general, these libraries let you write ROS nodes, publish and subscribe to topics, write and call services, and use the Parameter Server. Such a library can be implemented in any programming language, though the current focus is on providing robust C++ and Python support.

ROS 客户端库是简化 ROS 程序员工作的代码集合。它采用了许多 ROS 概念，并使其可以通过代码访问。通常，这些库允许您编写 ROS 节点、发布和订阅主题、编写和调用服务以及使用 Parameter Server。这样的库可以用任何编程语言实现，尽管目前的重点是提供强大的 C++ 和 Python 支持。

- [roscpp](#)
- [rospy](#)
- [roslisp](#)
- ...

Technical Overview

Tools

rosdep

rosdep is a command-line tool for installing system dependencies.

```
# install
sudo apt-get install python3-rosdep
# or
pip install rosdep
# source install
git clone https://github.com/ros-infrastructure/rosdep
cd rosdep
source setup.sh

# init rosdep, needs to call only once after installation
sudo rosdep init
# update
rosdep update
# install system dependencies

# install dependency of a package
rosdep install AMAZING_PACKAGE
# install dependency of all packages in the workspace
# cd into the catkin workspace, run:
rosdep install --from-paths src --ignore-src -r -y
```

catkin

Low-level build system macros and infrastructure for ROS.

- [wiki.ros: catkin](#)
- [wiki.ros: catkin conceptual overview](#)
- [catkin](#)
- [ros: rep-0128](#)

[catkin](#) is the official build system of ROS and the successor to the original ROS build system, [rosbuild](#). [catkin](#) combines [CMake](#) macros and Python scripts to provide some functionality on top of CMake's normal workflow. [catkin](#) was designed to be more conventional than [rosbuild](#), allowing for better distribution of packages, better cross-compiling support, and better portability. [catkin](#)'s workflow is very similar to [CMake](#)'s but adds support for automatic 'find package' infrastructure and building multiple, dependent projects at the same time.

[catkin](#) 是 ROS 的官方构建系统，也是原始 ROS 构建系统 [rosbuild](#) 的继承者。[catkin](#) 结合了 [CMake](#) 宏和 Python 脚本，在 [CMake](#) 的正常工作流程之上提供了一些功能。[Catkin](#) 的设计比 [rosbuild](#) 更传统，允许更好的包分发、更好的交叉编译支持和更好的可移植性。[catkin](#) 的工作流程与 [CMake](#) 的工作流程非常相似，但增加了对自动“查找包”基础设施的支持，并同时构建多个依赖的项目。

```
# debian 12 bookworm
catkin/stable 0.8.10-9 all
# python3
python3-catkin/stable, now 0.8.10-9 al
```

Usage:

```

cd path/to/your/catkin_workspace
# will build any packages in /catkin_workspace/src
catkin_make

# equivalent to
cd path/to/your/catkin_workspace
cd src
catkin_init_workspace
cd ..
mkdir build
cd build
cmake ../src -DCMAKE_INSTALL_PREFIX=../install -DCATKIN_DEVEL_PREFIX=../devel
make

# build specific package
catkin_make -DCATKIN_WHITELIST_PACKAGES="package1;package2"

# revert back to building all packages:
catkin_make -DCATKIN_WHITELIST_PACKAGES=""

# generate build and devel dir under workspace root

# install
catkin_make install

# specific source
catkin_make --source my_src
catkin_make install --source my_src

```

rosinstall_generator

generates `.rosinstall` files containing information about repositories with ROS packages/stacks.

```

# usage
rosinstall_generator PACKAGE DEPENDENCY1 DEPENDENCY2 > PACKAGE.rosinstall

# example
rosinstall_generator desktop --rosdistro noetic --deps --tar > noetic-desktop.rosinstall

```

vcstool

Command-line tools for maintaining a workspace of projects from multiple version-control systems.

vcstool provides commands to manage several local SCM repositories (supports git, mercurial, subversion, bazaar) based on a single workspace definition file (`.repos` or `.rosinstall`).

```
vcs help

# example
vcs import --input noetic-desktop.rosinstall ./src
```

text

text

<https://github.com/pokusew/ros2-build>

Install ROS2 HUMBLE on debian12 bookworm

Source installation ROS 2 Humble

```
vcstool
catkin_pkg
pyparsing
pip install empy==3.3.2 # 4.2 has error when compiling
lark

conda install pytest
```

colcon - collective construction??

<https://colcon.readthedocs.io/en/released/>

vcstool

<https://wiki.ros.org/vcstool>

To install ROS 2 Humble Hawksbill on Debian 12, follow this detailed step-by-step guide:

1. Add the ROS 2 Package Repository

a. Configure your sources list

Open a terminal and add the ROS 2 repository to your system's sources list:

```
sudo sh -c 'echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] https://raw.githubusercontent.com/ros/rosdistro/main/ros2-distro/ros2 $ROS_DISTRO main" | sudo tee /etc/apt/sources.list.d/ros2-distro.list'
```

Note: Since Debian is not officially supported by ROS 2, you might need to replace `$(lsb_release -cs)` with the equivalent Ubuntu codename for the target ROS 2 Humble distribution (e.g., `jammy` for Ubuntu 22.04).

b. Add the GPG key

Import the GPG key used to sign the ROS 2 packages:

```
sudo apt update
sudo apt install -y curl gnupg2 lsb-release
curl -sSL https://raw.githubusercontent.com/ros/rosdistro/main/ros.key | sudo tee /usr/share/keyrings/ros2-distro.list
```

2. Install ROS 2 Humble

a. Update package index

Update your local package index to include the ROS 2 repository:

```
sudo apt update
```

b. Install ROS 2

Install the ROS 2 Humble desktop full package:

```
sudo apt install ros-humble-desktop
```

If you want a lighter installation, you can use one of the following commands instead:

- For the base packages:

```
sudo apt install ros-humble-ros-base
```

- For specific tools or packages, list available ROS 2 packages:

```
apt search ros-humble
```

3. Set Up the Environment

a. Source the ROS 2 setup file

Add the ROS 2 setup script to your shell configuration file (`.bashrc` or `.zshrc`):

```
echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

If you use `zsh`, modify `.zshrc`:

```
echo "source /opt/ros/humble/setup.zsh" >> ~/.zshrc
source ~/.zshrc
```

b. Verify the installation

Check the ROS 2 version:

```
ros2 --version
```

4. Install Additional Tools (Optional)

a. ROS 2 Command Line Tools

Install development tools and utilities for building and debugging:

```
sudo apt install python3-colcon-common-extensions python3-rosdep python3-argcomplete
```

b. Initialize `rosdep`

Set up `rosdep` for managing dependencies:

```
sudo rosdep init
rosdep update
```

5. Test the Installation

a. Launch the ROS 2 demo

Run the talker and listener example to ensure ROS 2 is working:

1. Open a terminal and start a talker:

```
ros2 run demo_nodes_cpp talker
```

2. Open another terminal and start a listener:

```
ros2 run demo_nodes_cpp listener
```

You should see messages being published by the talker and received by the listener.

6. Troubleshooting

- If `ros2` commands are not recognized, check if ROS 2 is correctly sourced:

```
source /opt/ros/humble/setup.bash
```

- If any dependencies are missing, use `rosdep` to install them:

```
rosdep install --from-paths src --ignore-src -r -y
```

This should allow you to successfully install and configure ROS 2 Humble on Debian 12!

The error occurs because the ROS 2 Humble binaries are built for Ubuntu 22.04 (`jammy`), which relies on certain dependencies (like `libpython3.10`) that are not available in Debian 12 (`bookworm`) by default.

To resolve this issue, you can try the following approaches:

Option 1: Use Docker for ROS 2

Using Docker is the easiest and cleanest way to run ROS 2 on Debian 12 without dealing with compatibility issues.

1. Install Docker:

```
sudo apt update
sudo apt install -y docker.io
sudo usermod -aG docker $USER
```

2. Pull the ROS 2 Humble Docker image:

```
docker pull osrf/ros:humble-desktop
```

3. Run the Docker container:

```
docker run -it --rm osrf/ros:humble-desktop
```

This method isolates ROS 2 from your system, ensuring all dependencies work seamlessly.

Option 2: Build ROS 2 Humble from Source

If you need ROS 2 natively installed, you can build it from source with the following steps:

1. Install Required Tools

```
sudo apt update  
sudo apt install -y build-essential cmake git python3-colcon-common-extensions python3-rosdep python3-rosinstall
```

2. Install ROS 2 Dependencies

```
sudo rosdep init  
rosdep update
```

3. Clone the ROS 2 Humble Source Code

Create a workspace and clone the ROS 2 repositories:

```
mkdir -p ~/ros2_humble/src  
cd ~/ros2_humble  
wget https://raw.githubusercontent.com/ros2/ros2/humble/ros2.repos  
vcs import src < ros2.repos
```

4. Resolve Dependencies

Install all required dependencies:

5. Build ROS 2

Build the workspace:

```
colcon build --symlink-install
```

6. Source the ROS 2 Setup

Add the setup script to your shell configuration:

```
echo "source ~/ros2_humble/install/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Option 3: Manually Add Missing Dependencies

Manually resolving dependency issues can be complex but may work in some cases:

1. **Enable Debian Backports or Experimental Repositories:**

```
sudo nano /etc/apt/sources.list
```

Add the following line:

```
deb http://deb.debian.org/debian bookworm-backports main contrib non-free
```

2. **Install libpython3.10 :** Download the necessary package from Ubuntu's `jammy` repository and install it manually.

Example:

```
wget http://archive.ubuntu.com/ubuntu/pool/main/p/python3.10/libpython3.10_3.10.6-1_amd64.deb
sudo dpkg -i libpython3.10_3.10.6-1_amd64.deb
```

3. **Repeat for Other Dependencies:** Identify the missing dependencies and manually download and install them.

Recommendation

The most reliable option for running ROS 2 Humble on Debian 12 is **using Docker**. Building from source is feasible but requires more effort to resolve all dependency issues. Let me know if you'd like assistance with a specific approach!

Install ROS from source

ROS Noetic on Debian 12

Install from source:

- [wiki.ros: Installation](#)
- [csdn blog: Debian12 通过源码安装 ros-noetic](#)
- [Compiling ros1 noetic from source on Ubuntu 22.04](#)

Debian 12, Bookworm

Dependencies:

```
# use apt or pip to install:  
python3-  
rosdep  
rosinstall-generator  
vcstools  
vcstool  
  
# init rosdep  
sudo rosdep init  
rosdep update  
  
# use apt  
build-essential  
libboost1.74-all-dev  
  
# 版本不能太高, debian12 bookworm default v1.0.0  
liblog4cxx10v5_0.10.0  
liblog4cxx-dev_0.10.0  
  
# need lower version -> v1.11.2, can download from pkgs.org  
libogre-1.12-dev  
ogre-1.12-tools  
  
liburdfdom-tools  
liburdfdom-headers-dev  
liburdfdom-dev  
  
libbz2-dev  
libgpgme-dev  
  
liborocos-kdl-dev/stable 1.5.1-2+b4 amd64  
Kinematics and Dynamics Library development files  
  
liborocos-kdl1.5/stable,now 1.5.1-2+b4 amd64 [installed,automatic]  
Kinematics and Dynamics Library runtime
```

Installation:

```

# create catkin workspace
mkdir ~/ros_catkin_ws
cd ~/ros_catkin_ws

# download source code for ros noetic, use vcstool, build all od Desktop
# generates rosinstall file for noetic-desktop
rosinstall_generator desktop --rosdistro noetic --deps --tar > noetic-desktop.rosinstall
mkdir src
# use vcs to download all source code base on xx.rosinstall
vcs import --input noetic-desktop.rosinstall ./src

# resolve dependencies (in official doc)
# in debian 12 bookworm, please manually install the dependencies in the former section
rosdep install --from-paths ./src --ignore-packages-from-source --rosdistro noetic -y

# build, use catkin_make_isolate
./src/catkin/bin/catkin_make_isolated --install \
-DCMAKE_BUILD_TYPE=Release \
-DPYTHON_EXECUTABLE=/usr/bin/python3 # use your python3 path
--install-space path/to/install # default is ~/ros_catkin_ws/install_isolated

# after build success, all files have been installed in ~/ros_catkin_ws/install_isolated
# source the setup.sh file, or add it to your ~/.bashrc
source ~/ros_catkin_ws/install_isolated/setup.sh

```

Other problems:

```

roscore

# if stack on roscore, do:
pip uninstall rosgraph
sudo apt-get install python3-rosgraph python3-rosgraph-msgs

```

Lifetime Example

Problem

rust: 如下程序的生命周期'input 的含义是什么，标注了会怎么样，如果不标注会怎么样，请举例说明.

```
pub enum ExprSymbol<'input> {
    NumSymbol(&'input str),
    Op(Box<ExprSymbol<'input>>, Opcode, Box<ExprSymbol<'input>>),
    Error,
}
```

GPT

在你的代码中，生命周期标注 `'input` 的意义在于它限制了枚举 `ExprSymbol` 的实例中引用的字符串（或其他带有生命周期的部分）的生命周期。具体来说：

含义

生命周期 `'input` 是用来表明：

1. `ExprSymbol` 类型中的所有带有生命周期的字段，其生命周期必须与 `'input` 匹配。
2. 具体来说，`NumSymbol(&'input str)` 的字符串引用必须存在至少与 `ExprSymbol` 的生命周期一样长。

如果标注生命周期

生命周期 `'input` 的标注可以明确指定这个枚举中引用数据的关系，使得编译器能够追踪数据的生命周期并防止悬空引用等问题。

例如：

```
fn parse_expr<'a>(input: &'a str) -> ExprSymbol<'a> {
    ExprSymbol::NumSymbol(input) // 确保 `input` 的生命周期与返回值一致
}
```

在这种情况下：

- `parse_expr` 返回的 `ExprSymbol` 结构体中的字符串引用 `&'a str` 的生命周期与传入的 `input` 一致。
- 如果在调用 `parse_expr` 后 `input` 被释放，则会导致编译错误，因为 `ExprSymbol` 中仍然存储了对 `input` 的引用。

如果不标注生命周期

如果不标注生命周期，Rust 编译器会无法推断 `ExprSymbol` 中字符串引用的生命周期。这会导致以下情况：

1. 如果 `ExprSymbol` 的所有成员中不包含引用，则可以移除生命周期标注，Rust 会认为它没有任何依赖生命周期的数据，编译正常。
2. 如果有引用（如 `&str`），编译器会报错，因为它需要显式的生命周期来描述数据引用的生命周期关系。

例如：

```
pub enum ExprSymbol {
    NumSymbol(&str), // 编译器会报错：需要显式生命周期
    Op(Box<ExprSymbol>, Opcode, Box<ExprSymbol>),
    Error,
}
```

编译器会提示类似错误：

```
error[E0106]: missing lifetime specifier
--> src/main.rs:2:17
 |
2 |     NumSymbol(&str),
|           ^ expected named lifetime parameter
```

原因是 Rust 无法自动推断 `&str` 的生命周期。

举例说明

标注生命周期

```
fn example<'a>(input: &'a str) -> ExprSymbol<'a> {
    ExprSymbol::NumSymbol(input)
}

fn main() {
    let input = String::from("123");
    let symbol = example(&input); // input 的生命周期和 symbol 保持一致
    println!("{}:", symbol);
    // input 和 symbol 生命周期一致，因此在 input 释放前 symbol 是有效的
}
```

不标注生命周期（尝试移除）

```
pub enum ExprSymbol {
    NumSymbol(&str), // 错误：缺少生命周期标注
    Op(Box<ExprSymbol>, Opcode, Box<ExprSymbol>),
    Error,
}
```

这种情况下，编译器会要求添加生命周期标注，因为它需要确保 `&str` 的生命周期一致，避免悬空引用或内存安全问题。

总结

1. **标注生命周期** 是为了明确类型中的引用与其生命周期的关系，让 Rust 编译器能够确保内存安全。
2. 如果不标注生命周期，Rust 无法推断带有引用的结构体的生命周期，会导致编译错误。
3. 在你的例子中，生命周期 `'input` 确保了 `ExprSymbol<'input>` 中的所有字符串引用必须与 `'input` 生命周期一致，从而避免在 `ExprSymbol` 活跃时底层数据被释放的问题。

Problems and Solutions

Rust Resources

Rust

- [The Rust Reference](#)
- [Rust Ref: zh](#)
- [~~](#)
- [The Rust Programming Language](#)
- [The Rust Programming Language: Experiment Type](#)
- [Rust 程序设计语言 \(zh\) - 2022](#)
- [Rust 程序设计语言 \(zh\) - 2024-05-02](#)
- [~~](#)
- [effective rust: 35 Specific Ways to Improve Your Rust Code](#)
- [effective-rust-cn](#)
- [~~](#)
- [The Rustonomicon: The Dark Arts of Unsafe Rust](#)
- [Rust 语言圣经\(Rust Course\)](#)
- [rust-by-example](#)
- [The Cargo Book](#)
- [The Little Book of Rust Macros](#)
- [~~](#)
- [Rust RFCs - RFC Book](#)
- [~~](#)
- [releases](#)
- [github releases](#)
- [~~](#)
- [Rust 语言中文社区](#)
- [clippy](#)
- [Rust Design Patterns](#)
- [Newtype Index Pattern](#)
- [Embrace the newtype pattern -- Effective Rust](#)
- [Idiomatic tree and graph like structures in Rust](#)

Rust Compiler

- [cranelift: a fast, secure, relatively simple and innovative compiler backend](#)
- [rustc_codegen_cranelift](#)
- [wasmtime: About A lightweight WebAssembly runtime that is fast, secure, and standards-compliant](#)

Rust OS

- [NUDT-OS-Book](#)
- [rcore-os](#)
- [rCore-Tutorial-Book-v3](#)
- [haibo_chen](#)

Others

- [rust-for-linux](#)
 - [一文读懂什么是进程、线程、协程](#)
 - [rust-cli](#)
-

- [Joel on Software](#)

git submodule update --init --recursive

Commands

```
# rustup: Install, manage, and update Rust toolchains.
rustup install/default/update/show

rustup self uninstall

# cargo: Rust's package manager and build system.
cargo new <project-name> # create a new Rust project

cargo build # build the current package
cargo run # build and run the current package
cargo check # check the current package for errors without building
cargo test # run the tests in the current package
cargo fmt # check formatting of the current package

cargo build --release # build the current package with optimizations

cargo doc --open # build all dependences doc and open in browser

RUST_BACKTRACE=1 cargo run # checkout backtrace
```

OwnerShip

1. Rust 中的每一个值都有一个 **所有者**（*owner*）。
2. 值在任一时刻有且只有一个所有者。
3. 当所有者（变量）离开作用域，这个值将被丢弃。

引用

- 在任意给定时间，**要么**只能有一个可变引用，**要么**只能有多个不可变引用。
- 引用必须总是有效的。

Slice 类型

`slice` 允许你引用集合中一段连续的元素序列，而不用引用整个集合。`slice` 是一种引用，所以它没有所有权。

```

fn first_word(s: &String) -> &str {
    let bytes = s.as_bytes();

    for (i, &item) in bytes.iter().enumerate() {
        if item == b' ' {
            return &s[0..i];
        }
    }

    &s[..]
}

```

```

fn main() {
    let my_string = String::from("hello world");

    // `first_word` 适用于 `String` (的 slice)，部分或全部
    let word = first_word(&my_string[0..6]);
    let word = first_word(&my_string[..]);
    // `first_word` 也适用于 `String` 的引用,
    // 这等价于整个 `String` 的 slice
    let word = first_word(&my_string);

    let my_string_literal = "hello world";

    // `first_word` 适用于字符串字面值，部分或全部
    let word = first_word(&my_string_literal[0..6]);
    let word = first_word(&my_string_literal[..]);

    // 因为字符串字面值已经 **是** 字符串 slice 了,
    // 这也是适用的，无需 slice 语法!
    let word = first_word(my_string_literal);
}

```

Struct

字段初始化简写语法 (field init shorthand)

```
fn build_user(email: String, username: String) -> User {
    User {
        active: true,
        username,
        email,
        sign_in_count: 1,
    }
}
```

结构体更新语法 (struct update syntax)

使用旧实例的大部分值但改变其部分值来创建一个新的结构体实例

```
fn main() {
    // --snip--
    let user2 = User {
        email: String::from("another@example.com"),
        ..user1
    };
}
```

示例 5-7 中的代码也在 `user2` 中创建了一个新实例，但该实例中 `email` 字段的值与 `user1` 不同，而 `username`、`active` 和 `sign_in_count` 字段的值与 `user1` 相同。`..user1` 必须放在最后，以指定其余的字段应从 `user1` 的相应字段中获取其值，但我们可以选择以任何顺序为任意字段指定值，而不用考虑结构体定义中字段的顺序。

请注意，结构更新语法就像带有 `=` 的赋值，因为它移动了数据，就像我们在“[变量与数据交互的方式（一）：移动](#)”部分讲到的一样。在这个例子中，总体上说我们在创建 `user2` 后就不能再使用 `user1` 了，因为 `user1` 的 `username` 字段中的 `String` 被移到 `user2` 中。如果我们给 `user2` 的 `email` 和 `username` 都赋予新的 `String` 值，从而只使用 `user1` 的 `active` 和 `sign_in_count` 值，那么 `user1` 在创建 `user2` 后仍然有效。`active` 和 `sign_in_count` 的类型是实现 `Copy` trait 的类型，所以我们在“[变量与数据交互的方式（二）：克隆](#)”部分讨论的行为同样适用。

元组结构体 (tuple structs)

```
struct Color(i32, i32, i32);
struct Point(i32, i32, i32);

fn main() {
    let black = Color(0, 0, 0);
    let origin = Point(0, 0, 0);
}
```

类单元结构体 (unit-like structs)

- 没有任何字段的结构体
- 类单元结构体常常在你想要在某个类型上实现 trait 但不需要在类型中存储数据的时候发挥作用。

```
struct AlwaysEqual;

fn main() {
    let subject = AlwaysEqual;
}
```

sdsd

dsd

Rust Book

Rust 也为系统编程世界带来了现代化的开发工具：

- `Cargo` 是内置的依赖管理器和构建工具，它能轻松增加、编译和管理依赖，并使依赖在 Rust 生态系统中保持一致。
- `Rustfmt` 格式化工具确保开发者遵循一致的代码风格。
- `rust-analyzer` 为集成开发环境（IDE）提供了强大的代码补全和内联错误信息功能。
- `rustup` 下载 Rust，这是一个管理 Rust 版本和相关工具的命令行工具

Rust Language Reference

Ref

Traits 特性

```
unsafe? trait IDENTIFIER GenericParams? ( : TypeParamBounds? )? WhereClause? {
    InnerAttribute*
    AssociatedItem*
}
```

- A `trait` describes an abstract interface that types can implement.
- This interface consists of `associated items`, which come in three varieties.
- Trait declaration defines a trait in the `type namespace` of the module or block where it is located.
 - `Associated items` are defined as members of the trait within their respective namespaces.
 - `Associated types` are defined in the type namespace.
 - `Associated constants` and `associated functions` are defined in the value namespace.
- All traits define an implicit type parameter `self` that refers to “the type that is implementing this interface”.
- Traits may also contain additional type parameters. These type parameters, including `self`, may be constrained by other traits and so forth as usual.
- Traits are implemented for specific types through separate [implementations](#).
- Trait functions may omit the function body by replacing it with a semicolon. This indicates that the implementation must define the function.
- If the trait function defines a body, this definition acts as a default for any implementation which does not override it. Similarly, associated constants may omit the equals sign and expression to indicate implementations must define the constant value. Associated types must never define the type, the type may only be specified in an implementation.
- `Trait functions` are not allowed to be `const`.

-
- `trait` 描述了一个类能实现的抽象接口。
 - 这个 接口 由一些 相关项目 构成，包括：
 - `functions` 函数
 - `types` 类型
 - `constants` 常数
 - trait 的声明在该声明所在的 module 或 block 中的命名空间定义了一个 trait
 - 所有特征都定义了一个隐式类型参数 Self，它指的是“实现该接口的类型”。特征还可能包含其他类型参数。这些类型参数，包括 Self，可能会像往常一样受到其他特征等的约束。
 - 对一个特定类型的 trait 实现是与 trait 的定义相分开的
 - 特征函数可以通过用分号替换来省略函数体。这表明实现必须定义该函数。
 - 如果特征函数定义了一个函数体，则此定义将充当任何不覆盖它的实现的默认值。
 - 类似地，关联的常量可以省略等号和表达式以指示实现必须定义常量值。
 - 关联类型绝不能定义该类型，该类型只能在实现中指定。
 - trait function 不能为 const ?

```
// Examples of associated trait items with and without definitions.

trait Example {
    const CONST_NO_DEFAULT: i32;
    const CONST_WITH_DEFAULT: i32 = 99;
    type TypeNoDefault;
    fn method_without_default(&self);
    fn method_with_default(&self) {}
}
```

Trait bounds

泛型 项可能会使用 traits 作为 type parameters 的限制 (bounds)。

Trait and lifetime bounds

Trait 和 lifetime bounds 提供了一种方法来限制哪些类型和生命周期用作其参数。可以在 where 子句中为任何类型提供界限。对于某些常见情况，还有更简短的形式：

特征和生命周期界限为通用项提供了一种方法来限制将哪些类型和生命周期用作其参数。可以在 where 子句中为任何类型提供界限。对于某些常见情况，还有更简短的形式：

- Bounds written after declaring a generic parameter:
 - `fn f<A: Copy>()` is the same as `fn f<A>() where A: Copy {}`.
 - 声明通用参数后写入的界限：
 - `fn f<A: Copy>()` 与 `fn f () where A: Copy {}`。
- In trait declarations as supertraits:
 - `trait Circle : Shape {}` is equivalent to `trait Circle where Self : Shape {}`.
 - 在作为 supertraits 的特征声明中：
 - `trait Circle : Shape {}` 相当于 `trait circle where Self : Shape {}`。
- In trait declarations as bounds on associated types:
 - `trait A { type B: Copy; }` is equivalent to `trait A where Self::B: Copy { type B; }`.
 - 在特征声明中作为关联类型的边界：
 - `trait A { type B: Copy; }` 相当于 `trait A where Self::B: Copy { type B; }`。

Bounds on an item must be satisfied when using the item. When type checking and borrow checking a generic item, the bounds can be used to determine that a trait is implemented for a type. For example, given `Ty: Trait`

使用物品时必须满足物品的限制。当类型检查和借用检查通用项时，边界可用于确定是否为类型实现了特征。例如，给定 `Ty: Trait`

- In the body of a generic function, methods from `Trait` can be called on `Ty` values. Likewise associated constants on the `Trait` can be used.
 - 在泛型函数体内，可以在 `Ty` 上调用 `Trait` 中的方法 价值观。同样可以使用 `Trait` 上的关联常量。
- Associated types from `Trait` can be used.
 - 可以使用 `Trait` 中的关联类型。
- Generic functions and types with a `T: Trait` bounds can be used with `Ty` being used for `T`.
 - 带有 `T: Trait` 边界可以与 `Ty` 一起使用 用于 `T`。

Trait objects 特性对象

A `trait object` is an opaque value of another type that implements a set of traits. The set of traits is made up of an `object safe` `base trait` plus any number of `auto traits`.

特征对象是实现一组特征的另一种类型的不透明值。该特征集由 对象安全基本特征(`object safe base trait`) 加上任意数量的 自动特征 (`auto traits`) 组成。

Trait objects implement the `base trait`, its `auto traits`, and any `supertraits` of the base trait.

Trait 对象实现基本特征、其自动特征和任何基本特征的超级特征。

Trait objects are written as the keyword `dyn` followed by a set of `trait bounds`, but with the following restrictions on the trait bounds.

All traits except the first trait must be auto traits, there may not be more than one lifetime, and opt-out bounds (e.g. `?sized`) are not allowed. Furthermore, paths to traits may be parenthesized.

Trait 对象被编写为关键字 `dyn` 后跟一组 特征边界， 但对特征边界有以下限制。

除第一个特征之外的所有特征都必须是自动特征， 不能有超过一个生命周期，并且不允许选择退出边界 (例如`?Sized`) 。

此外， 特征的路径可以用括号括起来。

For example, given a trait `Trait`, the following are all trait objects:例如， 给定一个特征 `Trait`， 以下都是特征对象:

- `dyn Trait`
- `dyn Trait + Send`
- `dyn Trait + Send + Sync`
- `dyn Trait + 'static`
- `dyn Trait + Send + 'static`
- `dyn Trait +`
- `dyn 'static + Trait` .
- `dyn (Trait)`

Two trait object types alias each other if the base traits alias each other and if the sets of auto traits are the same and the lifetime bounds are the same. For example, `dyn Trait + Send + UnwindSafe` is the same as `dyn Trait + UnwindSafe + Send` .

Due to the opaqueness of which concrete type the value is of, trait objects are `dynamically sized types`. Like all DSTs, trait objects are used behind some type of pointer; for example `&dyn SomeTrait` or `Box<dyn SomeTrait>` . Each instance of a pointer to a trait object includes:

由于值属于哪种具体类型的不透明性， 特征对象是 动态调整大小的类型。像所有 DSTs 一样， 特征对象用在某种类型的指针后面； 例如 `&dyn SomeTrait` 或 `Box<dyn SomeTrait>` 。指向特征对象的指针的每个实例包括：

- a pointer to an instance of a type `T` that implements `SomeTrait`
 - 指向实现 `SomeTrait` 的类型 `T` 实例的指针
- a `virtual method table`, often just called a `vtable`, which contains, for each method of `SomeTrait` and its `supertraits` that `T` implements, a pointer to `T`'s implementation (i.e. a function pointer).
 - 虚拟方法表， 通常简称为 vtable， 其中包含 T 实现的 `SomeTrait` 及其超特征的每个方法， 一个指向 T 实现的指针（即函数指针）。

The purpose of trait objects is to permit “late binding” of methods. Calling a method on a trait object results in virtual dispatch at runtime: that is, a function pointer is loaded from the trait object vtable and invoked indirectly. The actual implementation for each vtable entry can vary on an object-by-object basis.

特征对象的目的是允许方法的“后期绑定”。在特征对象上调用方法会导致运行时虚拟分派：也就是说，从特征对象 vtable 加载函数指针并间接调用。每个 vtable 条目的实际实现可能因对象而异。

An example of a trait object:

```
trait Printable {
    fn stringify(&self) -> String;
}

impl Printable for i32 {
    fn stringify(&self) -> String { self.to_string() }
}

fn print(a: Box<dyn Printable>) {
    println!("{}", a.stringify());
}

fn main() {
    print(Box::new(10) as Box<dyn Printable>);
}
```

In this example, the trait `Printable` occurs as a trait object in both the type signature of `print`, and the cast expression in `main`.

在此示例中，特征 `Printable` 作为特征对象出现在 `print` 的类型签名和 `main` 中的强制转换表达式中。

Trait Object Lifetime Bounds

特征对象生命周期界限

Since a trait object can contain references, the lifetimes of those references need to be expressed as part of the trait object. This lifetime is written as `Trait + 'a`. There are `defaults` that allow this lifetime to usually be inferred with a sensible choice.

由于特征对象可以包含引用，因此这些引用的生命周期需要表达为特征对象的一部分。此 `lifetime` 可写为 `Trait + 'a`。有一些默认值允许通常通过明智的选择来推断此生命周期。

Attributes 属性

- [attributes](#)

```
// inner attribute
#[Attr]
// outer attribute
#[Arrt]
```

An `attribute` is a general, free-form metadatum that is interpreted according to name, convention, language, and compiler version. Attributes are modeled on Attributes in [ECMA-335](#), with the syntax coming from [ECMA-334](#) (C#).

Inner attributes, written with a bang (`!`) after the hash (`#`), apply to the item that the attribute is declared within.

- *Outer attributes*, written without the bang after the hash, apply to the thing that follows the attribute.

The attribute consists of a path to the attribute, followed by an optional delimited token tree whose interpretation is defined by the attribute. Attributes other than macro attributes also allow the input to be an equals sign (`=`) followed by an expression. See the [meta item syntax](#) below for more details.

An attribute may be unsafe to apply. To avoid undefined behavior when using these attributes, certain obligations that cannot be checked by the compiler must be met. To assert these have been, the attribute is wrapped in `unsafe(...)`, e.g. `#[unsafe(no_mangle)]`.

The following attributes are unsafe:

- `export_name`
- `link_section`
- `no_mangle`

Attributes can be classified into the following kinds:

- [Built-in attributes](#)
- [Macro attributes](#)
- [Derive macro helper attributes](#)
- [Tool attributes](#)

Attributes may be applied to many things in the language:

- All [item declarations](#) accept outer attributes while [external blocks](#), [functions](#), [implementations](#), and [modules](#) accept inner attributes.
- Most [statements](#) accept outer attributes (see [Expression Attributes](#) for limitations on expression statements).
- [Block expressions](#) accept outer and inner attributes, but only when they are the outer expression of an [expression statement](#) or the final expression of another block expression.
- [Enum](#) variants and [struct](#) and [union](#) fields accept outer attributes.
- [Match expression arms](#) accept outer attributes.
- [Generic lifetime or type parameter](#) accept outer attributes.
- Expressions accept outer attributes in limited situations, see [Expression Attributes](#) for details.
- [Function](#), [closure](#) and [function pointer](#) parameters accept outer attributes. This includes attributes on variadic parameters denoted with `...` in function pointers and [external blocks](#).

Examples:

```
// General metadata applied to the enclosing module or crate.
#![crate_type = "lib"]

// A function marked as a unit test
#[test]
fn test_foo() {
    /* ... */
}

// A conditionally-compiled module
#[cfg(target_os = "linux")]
mod bar {
    /* ... */
}

// A lint attribute used to suppress a warning/error
#[allow(non_camel_case_types)]
type int8_t = i8;

// Inner attribute applies to the entire function.
fn some_unused_variables() {
    #![allow(unused_variables)]

    let x = ();
    let y = ();
    let z = ();
}
```

Built-in attributes index

The following is an index of all built-in attributes.

- Conditional compilation
 - `cfg` — Controls conditional compilation.
 - `cfg_attr` — Conditionally includes attributes.
- Testing
 - `test` — Marks a function as a test.
 - `ignore` — Disables a test function.
 - `should_panic` — Indicates a test should generate a panic.
- Derive
 - `derive` — Automatic trait implementations.
 - `automatically_derived` — Marker for implementations created by `derive`.
- Macros
 - `macro_export` — Exports a `macro_rules` macro for cross-crate usage.
 - `macro_use` — Expands macro visibility, or imports macros from other crates.
 - `proc_macro` — Defines a function-like macro.
 - `proc_macro_derive` — Defines a derive macro.

- `proc_macro_attribute` — Defines an attribute macro.
- Diagnostics
 - `allow`, `expect`, `warn`, `deny`, `forbid` — Alters the default lint level.
 - `deprecated` — Generates deprecation notices.
 - `must_use` — Generates a lint for unused values.
 - `diagnostic::on_unimplemented` — Hints the compiler to emit a certain error message if a trait is not implemented.
- ABI, linking, symbols, and FFI
 - `link` — Specifies a native library to link with an `extern` block.
 - `link_name` — Specifies the name of the symbol for functions or statics in an `extern` block.
 - `link_ordinal` — Specifies the ordinal of the symbol for functions or statics in an `extern` block.
 - `no_link` — Prevents linking an extern crate.
 - `repr` — Controls type layout.
 - `crate_type` — Specifies the type of crate (library, executable, etc.).
 - `no_main` — Disables emitting the `main` symbol.
 - `export_name` — Specifies the exported symbol name for a function or static.
 - `link_section` — Specifies the section of an object file to use for a function or static.
 - `no_mangle` — Disables symbol name encoding.
 - `used` — Forces the compiler to keep a static item in the output object file.
 - `crate_name` — Specifies the crate name.
- Code generation
 - `inline` — Hint to inline code.
 - `cold` — Hint that a function is unlikely to be called.
 - `no_builtins` — Disables use of certain built-in functions.
 - `target_feature` — Configure platform-specific code generation.
 - `track_caller` - Pass the parent call location to `std::panic::Location::caller()`.
 - `instruction_set` - Specify the instruction set used to generate a functions code
- Documentation
 - `doc` — Specifies documentation. See [The Rustdoc Book](#) for more information. Doc comments are transformed into `doc` attributes.
- Preludes
 - `no_std` — Removes std from the prelude.
 - `no_implicit_prelude` — Disables prelude lookups within a module.
- Modules
 - `path` — Specifies the filename for a module.
- Limits
 - `recursion_limit` — Sets the maximum recursion limit for certain compile-time operations.
 - `type_length_limit` — Sets the maximum size of a polymorphic type.
- Runtime
 - `panic_handler` — Sets the function to handle panics.
 - `global_allocator` — Sets the global memory allocator.
 - `windows_subsystem` — Specifies the windows subsystem to link with.
- Features
 - `feature` — Used to enable unstable or experimental compiler features. See [The Unstable Book](#) for features implemented in `rustc`.
- Type System
 - `non_exhaustive` — Indicate that a type will have more fields/variants added in future.
- Debugger

- `debugger_visualizer` — Embeds a file that specifies debugger output for a type.
- `collapse_debuginfo` — Controls how macro invocations are encoded in debuginfo.

Derive 派生

- [derivable-traits](#)

`derive` 属性允许为数据结构自动生成新的程序项。它使用 `MetaListPaths` 元项属性句法（为程序项）指定一系列要实现的 trait 或指定要执行的派生宏的路径。

例如，下面的派生属性将为结构体 `Foo` 创建一个实现 `PartialEq` trait 和 `Clone` trait 的实现(`impl item`)，类型参数 `T` 将被派生出的实现(`impl`)加上 `PartialEq` 或¹ `clone` 约束：

```
#[test]
#[should_panic(expected = "values don't match")]
fn mytest() {
    assert_eq!(1, 2, "values don't match");
}
```

1、自动派生常用的 traits：Rust 提供了一些常用的 traits，可以通过 `#[derive()]` 属性自动为结构体或枚举实现这些 traits。一些常见的可派生 traits 包括：

- `Debug`：通过实现 `Debug` trait，可以使用 `println!("{}:{}", my_struct)` 来打印结构体的调试信息。
- `Clone`：通过实现 `Clone` trait，可以使用 `my_struct.clone()` 创建结构体的克隆副本。
- `PartialEq` 和 `Eq`：通过实现 `PartialEq` trait，可以进行结构体的部分相等性比较，而 `Eq` trait 则实现了完全相等性比较。
- `PartialOrd` 和 `Ord`：通过实现 `PartialOrd` trait，可以对结构体进行部分有序性比较，而 `Ord` trait 实现了完全有序性比较。

2、自定义 traits 的自动派生：除了派生常见的 traits，您还可以自定义 traits，并使用 `#[derive()]` 属性为结构体或枚举自动生成实现代码。例如，如果您定义了一个名为 `MyTrait` 的自定义 trait，并希望为结构体自动实现它，可以这样写：

```
trait MyTrait {
    // trait 方法定义
}

#[derive(MyTrait)]
struct MyStruct {
    // 结构体字段
}
```

注意，对于自定义 traits 的派生，您需要在编写 `MyTrait` trait 时手动实现 `#[derive(MyTrait)]` 的逻辑，或者使用第三方库提供的派生宏。

3、手动实现 traits：某些 traits 无法通过 `#[derive()]` 属性自动派生，因为它们可能需要更多的信息或自定义实现。在这种情况下，您需要手动为结构体或枚举实现这些 traits。手动实现 traits 通常涉及为每个 trait 方法提供具体的实现代码。以下是手动实现 `MyTrait` trait 的示例：

```

trait MyTrait {
    // trait 方法定义
}

struct MyStruct {
    // 结构体字段
}

impl MyTrait for MyStruct {
    // MyTrait 方法的具体实现
}

```

#[derive(Debug)]

```

#[derive(Debug)]
struct Rectangle {
    width: u32,
    height: u32,
}

fn main() {
    let rect1 = Rectangle {
        width: 30,
        height: 50,
    };

    println!("rect1 is {rect1:?}");
}

```

在 {} 中加入 `:?` 指示符告诉 `println!` 我们想要使用叫做 `Debug` 的输出格式。`Debug` 是一个 trait，它允许我们以一种对开发者有帮助的方式打印结构体，以便当我们调试代码时能看到它的值。

当我们有一个更大的结构体时，能有更易读一点的输出就好了，为此可以使用 `{:#?}` 替换 `println!` 字符串中的 `{:?:}`。在这个例子中使用 `{:#?}` 风格将会输出如下：

```

$ cargo run
Compiling rectangles v0.1.0 (file:///projects/rectangles)
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.48s
Running `target/debug/rectangles`
rect1 is Rectangle {
    width: 30,
    height: 50,
}

```

另一种使用 `Debug` 格式打印数值的方法是使用 `dbg!` 宏。`dbg!` 宏接收一个表达式的所有权（与 `println!` 宏相反，后者接收的是引用），打印出代码中调用 `dbg!` 宏时所在的文件和行号，以及该表达式的结果值，并返回该值的所有权。

注意：调用 `dbg!` 宏会打印到标准错误控制台流（`stderr`），与 `println!` 不同，后者会打印到标准输出控制台流（`stdout`）。

```
#[derive(Debug)]
struct Rectangle {
    width: u32,
    height: u32,
}

fn main() {
    let scale = 2;
    let rect1 = Rectangle {
        width: dbg!(30 * scale),
        height: 50,
    };

    dbg!(&rect1);
}
```

```
$ cargo run
Compiling rectangles v0.1.0 (file:///projects/rectangles)
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.61s
Running `target/debug/rectangles`
[src/main.rs:10:16] 30 * scale = 60
[src/main.rs:14:5] &rect1 = Rectangle {
    width: 60,
    height: 50,
}
```

Method and Impl

```

#[derive(Debug)]
struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }
}

fn main() {
    let rect1 = Rectangle {
        width: 30,
        height: 50,
    };

    println!(
        "The area of the rectangle is {} square pixels.",
        rect1.area()
    );
}

```

为了使函数定义于 `Rectangle` 的上下文中，我们开始了一个 `impl` 块（`impl` 是 *implementation* 的缩写），这个 `impl` 块中的所有内容都将与 `Rectangle` 类型相关联。接着将 `area` 函数移动到 `impl` 大括号中，并将签名中的第一个（在这里也是唯一一个）参数和函数体中其他地方的对应参数改成 `self`。然后在 `main` 中将我们先前调用 `area` 方法并传递 `rect1` 作为参数的地方，改成使用 **方法语法** (*method syntax*) 在 `Rectangle` 实例上调用 `area` 方法。方法语法获取一个实例并加上一个点号，后跟方法名、圆括号以及任何参数。

在 `area` 的签名中，使用 `&self` 来替代 `rectangle: &Rectangle`，`&self` 实际上是 `self: &Self` 的缩写。在一个 `impl` 块中，`Self` 类型是 `impl` 块的类型的别名。方法的第一个参数必须有一个名为 `self` 的 `Self` 类型的参数，所以 Rust 让你在第一个参数位置上只用 `self` 这个名字来简化。注意，我们仍然需要在 `self` 前面使用 `&` 来表示这个方法借用了 `Self` 实例，就像我们在 `rectangle: &Rectangle` 中做的那样。方法可以选择获得 `self` 的所有权，或者像我们这里一样不可变地借用 `self`，或者可变地借用 `self`，就跟其他参数一样。

这里选择 `&self` 的理由跟在函数版本中使用 `&Rectangle` 是相同的：我们并不想获取所有权，只希望能够读取结构体中的数据，而不是写入。如果想要在方法中改变调用方法的实例，需要将第一个参数改为 `&mut self`。通过仅仅使用 `self` 作为第一个参数来使方法获取实例的所有权是很少见的；这种技术通常用在当方法将 `self` 转换成别的实例的时候，这时我们想要防止调用者在转换之后使用原始的实例。

使用方法替代函数，除了可使用方法语法和不需要在每个函数签名中重复 `self` 的类型之外，其主要好处在于组织性。我们将某个类型实例能做的所有事情都一起放入 `impl` 块中，而不是让将来的用户在我们的库中到处寻找 `Rectangle` 的功能。

自动引用和解引用（automatic referencing and dereferencing）

Rust 并没有一个与 `->` 等效的运算符；相反，Rust 有一个叫 **自动引用和解引用**（*automatic referencing and dereferencing*）的功能。方法调用是 Rust 中少数几个拥有这种行为的地方。

它是这样工作的：当使用 `object.something()` 调用方法时，Rust 会自动为 `object` 添加 `&`、`&mut` 或 `*` 以便使 `object` 与方法签名匹配。

```
p1.distance(&p2);
(&p1).distance(&p2);
```

第一行看起来简洁的多。这种自动引用的行为之所以有效，是因为方法有一个明确的接收者——`self` 的类型。在给出接收者和方法名的前提下，Rust 可以明确地计算出方法是仅仅读取（`&self`），做出修改（`&mut self`）或者是获取所有权（`self`）。事实上，Rust 对方法接收者的隐式借用让所有权在实践中更友好。

关联函数

所有在 `impl` 块中定义的函数被称为 **关联函数**（*associated functions*），因为它们与 `impl` 后面命名的类型相关。我们可以定义不以 `self` 为第一参数的关联函数（因此不是方法），因为它们并不作用于一个结构体的实例。我们已经使用了一个这样的函数：在 `String` 类型上定义的 `String::from` 函数。

不是方法的关联函数经常被用作返回一个结构体新实例的构造函数。这些函数的名称通常为 `new`，但 `new` 并不是一个关键字。例如我们可以提供一个叫做 `square` 关联函数，它接受一个维度参数并且同时作为宽和高，这样可以更轻松的创建一个正方形 `Rectangle` 而不必指定两次同样的值：

```
impl Rectangle {
    fn square(size: u32) -> Self {
        Self {
            width: size,
            height: size,
        }
    }
}
```

关键字 `Self` 在函数的返回类型中代指在 `impl` 关键字后出现的类型，在这里是 `Rectangle`

使用结构体名和 `::` 语法来调用这个关联函数：比如 `let sq = Rectangle::square(3);`。这个函数位于结构体的命名空间中：`::` 语法用于关联函数和模块创建的命名空间。[第七章](#)会讲到模块。

Option 枚举和其相对于空值的优势

这一部分会分析一个 `Option` 的案例，`Option` 是标准库定义的另一个枚举。`Option` 类型应用广泛因为它编码了一个非常普遍的场景，即一个值要么有值要么没值。

例如，如果请求一个非空列表的第一项，会得到一个值，如果请求一个空的列表，就什么也不会得到。从类型系统的角度来表达这个概念就意味着编译器需要检查是否处理了所有应该处理的情况，这样就可以避免在其他编程语言中非常常见的 bug。

编程语言的设计经常要考虑包含哪些功能，但考虑排除哪些功能也很重要。Rust 并没有很多其他语言中有的空值功能。**空值** (*Null*) 是一个值，它代表没有值。在有空值的语言中，变量总是这两种状态之一：空值和非空值。

Tony Hoare, `null` 的发明者，在他 2009 年的演讲“Null References: The Billion Dollar Mistake”中曾经说到：

```
I call it my billion-dollar mistake. At that time, I was designing the first comprehensive type system for objects. I designed it because I didn't understand references. I didn't know what they were. So I thought I'd just invent my own. It looked like it solved all the problems that references could solve, but in a much more convenient way. And I called it the null object.
```

我称之为我十亿美元的错误。当时，我在为一个面向对象语言设计第一个综合性的面向引用的类型系统。我的目标是通过编译器的自动检测来避免空指针引用错误。

空值的问题在于当你尝试像一个非空值那样使用一个空值，会出现某种形式的错误。因为空和非空的属性无处不在，非常容易出现这类错误。

然而，空值尝试表达的概念仍然是有意义的：空值是一个因为某种原因目前无效或缺失的值。

问题不在于概念而在于具体的实现。为此，Rust 并没有空值，不过它确实拥有一个可以编码存在或不存在概念的枚举。这个枚举是 `Option<T>`，而且它[定义于标准库中](#)，如下：

```
enum Option<T> {
    None,
    Some(T),
}
```

`Option<T>` 枚举是如此有用以至于它甚至被包含在了 `prelude` 之中，你不需要将其显式引入作用域。另外，它的成员也是如此，可以不需要 `Option::` 前缀来直接使用 `Some` 和 `None`。即便如此 `Option<T>` 也仍是常规的枚举，`Some(T)` 和 `None` 仍是 `Option<T>` 的成员。

`<T>` 语法是一个我们还未讲到的 Rust 功能。它是一个泛型类型参数，第十章会更详细的讲解泛型。目前，所有你需要知道的就是 `<T>` 意味着 `Option` 枚举的 `Some` 成员可以包含任意类型的数据，同时每一个用于 `T` 位置的具体类型使得 `Option<T>` 整体作为不同的类型。这里是一些包含数字类型和字符串类型 `option` 值的例子：

```
let some_number = Some(5);
let some_char = Some('e');

let absent_number: Option<i32> = None;
```

`some_number` 的类型是 `Option<i32>`。`some_char` 的类型是 `Option<char>`，是不同于 `some_number` 的类型。因为我们在 `Some` 成员中指定了值，Rust 可以推断其类型。对于 `absent_number`，Rust 需要我们指定 `Option` 整体的类型，因为编译器只通过 `None` 值无法推断出 `Some` 成员保存的值的类型。这里我们告诉 Rust 希望 `absent_number` 是 `Option<i32>` 类型的。

当有一个 `Some` 值时，我们就知道存在一个值，而这个值保存在 `Some` 中。当有个 `None` 值时，在某种意义上，它跟空值具有相同的意义：并没有一个有效的值。那么，`Option<T>` 为什么就比空值要好呢？

简而言之，因为 `Option<T>` 和 `T`（这里 `T` 可以是任何类型）是不同的类型，编译器不允许像一个肯定有效的值那样使用 `Option<T>`。例如，这段代码不能编译，因为它尝试将 `Option<i8>` 与 `i8` 相加：

```

let x: i8 = 5;
let y: Option<i8> = Some(5);

let sum = x + y;

$ cargo run
    Compiling enums v0.1.0 (file:///projects/enums)
error[E0277]: cannot add `Option<i8>` to `i8`
--> src/main.rs:5:17
|
5 |     let sum = x + y;
|           ^ no implementation for `i8 + Option<i8>`
|
= help: the trait `Add<Option<i8>>` is not implemented for `i8`
= help: the following other types implement trait `Add<Rhs>`:
    `&'a i8` implements `Add<i8>`
    `&i8` implements `Add<&i8>`
    `i8` implements `Add<&i8>`
    `i8` implements `Add`


For more information about this error, try `rustc --explain E0277`.
error: could not compile `enums` (bin "enums") due to 1 previous error

```

很好！事实上，错误信息意味着 Rust 不知道该如何将 `Option<i8>` 与 `i8` 相加，因为它们的类型不同。当在 Rust 中拥有一个像 `i8` 这样类型的值时，编译器确保它总是有一个有效的值。我们可以自信使用而无需做空值检查。只有当使用 `Option<i8>`（或者任何用到的类型）的时候需要担心可能没有值，而编译器会确保我们在使用值之前处理了为空的情况。

换句话说，在对 `Option<T>` 进行运算之前必须将其转换为 `T`。通常这能帮助我们捕获到空值最常见的问题之一：假设某值不为空但实际上为空的情况。

消除了错误地假设一个非空值的风险，会让你对代码更加有信心。为了拥有一个可能为空的值，你必须要显式的将其放入对应类型的 `Option<T>` 中。接着，当使用这个值时，必须明确的处理值为空的情况。只要一个值不是 `Option<T>` 类型，你就 **可以** 安全的认定它的值不为空。这是 Rust 的一个经过深思熟虑的设计决策，来限制空值的泛滥以增加 Rust 代码的安全性。

那么当有一个 `Option<T>` 的值时，如何从 `Some` 成员中取出 `T` 的值来使用它呢？`Option<T>` 枚举拥有大量用于各种情况的方法：你可以查看[它的文档](#)。熟悉 `Option<T>` 的方法将对你的 Rust 之旅非常有用。

总的来说，为了使用 `Option<T>` 值，需要编写处理每个成员的代码。你想要一些代码只当拥有 `Some(T)` 值时运行，允许这些代码使用其中的 `T`。也希望一些代码只在值为 `None` 时运行，这些代码并没有一个可用的 `T` 值。`match` 表达式就是这么一个处理枚举的控制流结构：它会根据枚举的成员运行不同的代码，这些代码可以使用匹配到的值中的数据。

PS: C/C++如果程序员不够小心注意，很容易造成有空指针的情况，并且忘了检查，就会导致对空指针的使用！！（CSC-2024 C++ 惨痛的教训 - 依赖关系分析遍中没有对函数可能传入空指针进行检查，导致对空指针的操作！）

为什么要引入空指针？如果不得不引入某种为“空”的状态，最好 wrap 一下！就像 rust 中的 Option 一样！

Rust Programming Language

Ch3: Common Programming Concepts

- Shadowing: 隐藏, 定义一个与之前变量同名的新变量, 第一个变量被第二个变量隐藏。

Ch9: 错误处理

Rust 要求你承认错误的可能性, 并在你的代码编译前采取一些行动。

- 可恢复的 recoverable: 向用户报告错误
 - 使用 `Result` 处理
- 不可恢复的 unrecoverable: 立即 panic 停止程序
 - 使用 `panic!` 处理
 - 通过 backtrace 查看调用栈信息

使用 `Result` 处理可恢复的 (recoverable) 错误

```
use std::fs::File;

fn main() {
    let greeting_file_result = File::open("hello.txt"); // Result<T, E>

    let greeting_file = match greeting_file_result {
        Ok(file) => file,
        Err(error) => panic!("Problem opening the file: {error:?}"),
    };
}
```

匹配不同的错误

```
use std::fs::File;
use std::io::ErrorKind;

fn main() {
    let greeting_file_result = File::open("hello.txt");

    let greeting_file = match greeting_file_result {
        Ok(file) => file,
        Err(error) => match error.kind() {
            ErrorKind::NotFound => match File::create("hello.txt") {
                Ok(fc) => fc,
                Err(e) => panic!("Problem creating the file: {e:?}"),
            },
            other_error => {
                panic!("Problem opening the file: {other_error:?}");
            }
        },
    };
}
```

使用 闭包 和 `unwrap_or_else` 方法

```
use std::fs::File;
use std::io::ErrorKind;

fn main() {
    let greeting_file = File::open("hello.txt").unwrap_or_else(|error| {
        if error.kind() == ErrorKind::NotFound {
            File::create("hello.txt").unwrap_or_else(|error| {
                panic!("Problem creating the file: {:?}", error);
            })
        } else {
            panic!("Problem opening the file: {:?}", error);
        }
    });
}
```

失败时 `panic` 的简写: `unwrap` 和 `expect`

如果 Result 值是成员 Ok, `unwrap` 会返回 Ok 中的值。如果 Result 是成员 Err, `unwrap` 会为我们调用 `panic!`

```
use std::fs::File;

fn main() {
    let greeting_file = File::open("hello.txt").unwrap();
}
```

expect 与 unwrap 的使用方式一样：返回文件句柄或调用 panic! 宏。expect 在调用 panic! 时使用的错误信息将是我们传递给 expect 的参数，而不像 unwrap 那样使用默认的 panic! 信息。

```
use std::fs::File;

fn main() {
    let greeting_file = File::open("hello.txt")
        .expect("hello.txt should be included in this project");
}
```

传播错误 propagating

```
use std::fs::File;
use std::io::{self, Read};

fn read_username_from_file() -> Result<String, io::Error> {
    let username_file_result = File::open("hello.txt");

    let mut username_file = match username_file_result {
        Ok(file) => file,
        Err(e) => return Err(e),
    };

    let mut username = String::new();

    match username_file.read_to_string(&mut username) {
        Ok(_) => Ok(username),
        Err(e) => Err(e),
    }
}
```

传播错误的简写：? 运算符

Result 值之后的 ? 被定义为与示例 9-6 中定义的处理 Result 值的 match 表达式有着完全相同的工作方式。如果 Result 的值是 Ok，这个表达式将会返回 Ok 中的值而程序将继续执行。如果值是 Err，Err 将作为整个函数的返回值，就好像使用了 return 关键字一样，这样错误值就被传播给了调用者。

```

use std::fs::File;
use std::io::{self, Read};

fn read_username_from_file() -> Result<String, io::Error> {
    let mut username_file = File::open("hello.txt")?;
    let mut username = String::new();
    username_file.read_to_string(&mut username)?;
    Ok(username)
}

```

? 运算符所使用的错误值被传递给了 from 函数，它定义于标准库的 From trait 中，其用来将错误从一种类型转换为另一种类型。当 ? 运算符调用 from 函数时，收到的错误类型被转换为由当前函数返回类型所指定的错误类型。这在当函数返回单个错误类型来代表所有可能失败的方式时很有用，即使其可能会因很多种原因失败。

我们可以将示例 9-7 中的 read_username_from_file 函数修改为返回一个自定义的 OurError 错误类型。如果我们也定义了 impl From for OurError 来从 io::Error 构造一个 OurError 实例，那么 read_username_from_file 函数体中的 ? 运算符调用会调用 from 并转换错误而无需在函数中增加任何额外的代码。

我们甚至可以在 ? 之后直接使用链式方法调用进一步缩短代码，如示例 9-8 所示：

```

use std::fs::File;
use std::io::{self, Read};

fn read_username_from_file() -> Result<String, io::Error> {
    let mut username = String::new();

    File::open("hello.txt")?.read_to_string(&mut username)?;

    Ok(username)
}

```

more simplify:

```

use std::fs;
use std::io;

fn read_username_from_file() -> Result<String, io::Error> {
    fs::read_to_string("hello.txt")
}

```

在 Option<T> 值上使用 ? 运算符

错误信息也提到 ? 也可用于 Option 值。如同对 Result 使用 ? 一样，只能在返回 Option 的函数中对 Option 使用 ?。在 Option 上调用 ? 运算符的行为与 Result 类似：如果值是 None，此时 None 会从函数中提前返回。如果值是 Some，Some 中的值作为表达式的返回值同时函数继续。示例 9-11 中有一个从给定文本中返回第一行最后一个字符的函数的例子：

```
fn last_char_of_first_line(text: &str) -> Option<char> {
    text.lines().next()?.chars().last()
}
```

这个函数返回 `Option` 因为它可能会在这个位置找到一个字符，也可能没有字符。这段代码获取 `text` 字符串 `slice` 作为参数并调用其 `lines` 方法，这会返回一个字符串中每一行的迭代器。因为函数希望检查第一行，所以调用了迭代器 `next` 来获取迭代器中第一个值。如果 `text` 是空字符串，`next` 调用会返回 `None`，此时我们可以使用 `?` 来停止并从 `last_char_of_first_line` 返回 `None`。如果 `text` 不是空字符串，`next` 会返回一个包含 `text` 中第一行的字符串 `slice` 的 `Some` 值。

`?` 会提取这个字符串 `slice`，然后可以在字符串 `slice` 上调用 `chars` 来获取字符的迭代器。我们感兴趣的是第一行的最后一个字符，所以可以调用 `last` 来返回迭代器的最后一项。这是一个 `Option`，因为有可能第一行是一个空字符串，例如 `text` 以一个空行开头而后面的行有文本，像是 "`nhi`"。不过，如果第一行有最后一个字符，它会返回在一个 `Some` 成员中。`?` 运算符作用于其中给了我们一个简洁的表达这种逻辑的方式。如果我们不能在 `Option` 上使用 `?` 运算符，则不得不使用更多的方法调用或者 `match` 表达式来实现这些逻辑。

注意你可以在返回 `Result` 的函数中对 `Result` 使用 `?` 运算符，可以在返回 `Option` 的函数中对 `Option` 使用 `?` 运算符，但是不可以混合搭配。`?` 运算符不会自动将 `Result` 转化为 `Option`，反之亦然；在这些情况下，可以使用类似 `Result` 的 `ok` 方法或者 `Option` 的 `ok_or` 方法来显式转换。

幸运的是 `main` 函数也可以返回 `Result<(), E>`，示例 9-12 中的代码来自示例 9-10 不过修改了 `main` 的返回值为 `Result<(), Box<dyn Error>>` 并在结尾增加了一个 `Ok()` 作为返回值。这段代码可以编译：

```
use std::error::Error;
use std::fs::File;

fn main() -> Result<(), Box<dyn Error>> {
    let greeting_file = File::open("hello.txt")?;

    Ok(())
}
```

`Box<dyn Error>` 类型是一个 **trait 对象** (*trait object*) 第十八章 [顾及不同类型值的 trait 对象](#) 部分会做介绍。目前可以将 `Box<dyn Error>` 理解为“任何类型的错误”。在返回 `Box<dyn Error>` 错误类型 `main` 函数中对 `Result` 使用 `?` 是允许的，因为它允许任何 `Err` 值提前返回。即便 `main` 函数体从来只会返回 `std::io::Error` 错误类型，通过指定 `Box<dyn Error>`，这个签名也仍是正确的，甚至当 `main` 函数体中增加更多返回其他错误类型的代码时也是如此。

当 `main` 函数返回 `Result<(), E>`，如果 `main` 返回 `Ok()` 可执行程序会以 `0` 值退出，而如果 `main` 返回 `Err` 值则会以非零值退出；成功退出的程序会返回整数 `0`，运行错误的程序会返回非 `0` 的整数。Rust 也会从二进制程序中返回与这个惯例相兼容的整数。

`main` 函数也可以返回任何实现了 `std::process::Termination` trait 的类型，它包含了一个返回 `ExitCode` 的 `report` 函数。请查阅标准库文档了解更多为自定义类型实现 `Termination` trait 的细节。

现在我们讨论过了调用 `panic!` 或返回 `Result` 的细节，是时候回到它们各自适合哪些场景的话题了。

创建自定义类型进行有效性验证

```

pub struct Guess {
    value: i32,
}

impl Guess {
    pub fn new(value: i32) -> Guess {
        if value < 1 || value > 100 {
            panic!("Guess value must be between 1 and 100, got {value}.");
        }

        Guess { value }
    }

    pub fn value(&self) -> i32 {
        self.value
    }
}

```

Ch10: Generic Types, Traits, and lifetimes

- Generic Data Types
- Traits: Defining Shared Behavior
- Validating References with Lifetimes

Traits

- trait 定义了某个特定类型拥有可能与其他类型共享的功能。可以通过 trait 以一种抽象的方式定义共同行为。
- 可以使用 trait bounds 指定泛型是任何拥有特定行为的类型。

需要注意的限制是，只有在 trait 或类型至少有一个属于当前 crate 时，我们才能对类型实现该 trait。

- 例如，可以为 aggregator crate 的自定义类型 Tweet 实现如标准库中的 Display trait，这是因为 Tweet 类型位于 aggregator crate 本地的作用域中。
- 类似地，也可以在 aggregator crate 中为 Vec<T> 实现 Summary，这是因为 Summary trait 位于 aggregator crate 本地作用域中。

但是不能为外部类型实现外部 trait。例如，不能在 aggregator crate 中为 Vec<T> 实现 Display trait。这是因为 Display 和 Vec<T> 都定义于标准库中，它们并不位于 aggregator crate 本地作用域中。这个限制是被称为 **相干性** (coherence) 的程序属性的一部分，或者更具体的说是 **孤儿规则** (orphan rule)，其得名于不存在父类型。这条规则确保了其他人编写的代码不会破坏你代码，反之亦然。没有这条规则的话，两个 crate 可以分别对相同类型实现相同的 trait，而 Rust 将无从得知应该使用哪一个实现。

Validating References with Lifetimes: 使用生命周期确保引用有效

- 生命周期避免了悬垂引用 dangling reference

- 借用检查器 borrow checker: 比较作用域以确保所有的借用都是有效的
- 函数中的泛型生命周期
- 生命周期注解语法
- 函数签名中的生命周期注解
- 深入理解生命周期
- 结构体定义中的生命周期注解
- 生命周期省略 Lifetime Elision
- 方法定义中的生命周期注解
- 静态生命周期
- 结合泛型类型参数、trait bounds 和生命周期
- 总结

```
fn longest(x: &str, y: &str) -> &str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

```
$ cargo run
Compiling chapter10 v0.1.0 (file:///projects/chapter10)
error[E0106]: missing lifetime specifier
--> src/main.rs:9:33
|
9 | fn longest(x: &str, y: &str) -> &str {
|         ----      ^ expected named lifetime parameter
|
= help: this function's return type contains a borrowed value, but the signature does not say when
help: consider introducing a named lifetime parameter
|
9 | fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
|         +++++     ++          ++          ++
|
For more information about this error, try `rustc --explain E0106`.
error: could not compile `chapter10` (bin "chapter10") due to 1 previous error
```

提示文本揭示了返回值需要一个泛型生命周期参数，因为 Rust 并不知道将要返回的引用是指向 x 或 y。事实上我们也不知道，因为函数体中 if 块返回一个 x 的引用而 else 块返回一个 y 的引用！

生命周期注解语法

函数签名中的生命周期注解

我们希望函数签名表达如下限制：这两个参数和返回的引用存活的一样久。（两个）参数和返回的引用的生命周期是相关的。就像示例 10-21 中在每个引用中都加上了 'a 那样。

```
fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

现在函数签名表明对于某些生命周期 `'a`，函数会获取两个参数，它们都是与生命周期 `'a` 存在的至少一样长的字符串 slice。函数会返回一个同样也与生命周期 `'a` 存在的至少一样长的字符串 slice。它的实际含义是 `longest` 函数返回的引用的生命周期与函数参数所引用的值的生命周期的较小者一致。这些关系就是我们希望 Rust 分析代码时所使用的。

记住通过在函数签名中指定生命周期参数时，我们并没有改变任何传入值或返回值的生命周期，而是指出任何不满足这个约束条件的值都将被借用检查器拒绝。注意 `longest` 函数并不需要知道 `x` 和 `y` 具体会存在多久，而只需要知道有某个可以被 `'a` 替代的作用域将会满足这个签名。

当在函数中使用生命周期注解时，这些注解出现在函数签名中，而不存在于函数体中的任何代码中。生命周期注解成为了函数约定的一部分，非常像签名中的类型。让函数签名包含生命周期约定意味着 Rust 编译器的工作变得更简单了。如果函数注解有误或者调用方法不对，编译器错误可以更准确地指出代码和限制的部分。如果不这么做的话，Rust 编译器会对我们期望的生命周期关系做更多的推断，这样编译器可能只能指出离出问题地方很多步之外的代码。

当具体的引用被传递给 `longest` 时，被 `'a` 所替代的具体生命周期是 `x` 的作用域与 `y` 的作用域相重叠的那一部分。换一种说法就是泛型生命周期 `'a` 的具体生命周期等同于 `x` 和 `y` 的生命周期中较小的那个。因为我们用相同的生命周期参数 `'a` 标注了返回的引用值，所以返回的引用值就能保证在 `x` 和 `y` 中较短的那个生命周期结束之前保持有效。

深入理解生命周期

当从函数返回一个引用，返回值的生命周期参数需要与一个参数的生命周期参数相匹配。如果返回的引用没有指向任何一个参数，那么唯一的可能就是它指向一个函数内部创建的值。然而它将会是一个悬垂引用，因为它将会在函数结束时离开作用域。

综上，生命周期语法是用于将函数的多个参数与其返回值的生命周期进行关联的。一旦它们形成了某种关联，Rust 就有了足够的信息来允许内存安全的操作并阻止会产生悬垂指针亦或是违反内存安全的行为。

结构体定义中的生命周期注解

```

struct ImportantExcerpt<'a> {
    part: &'a str,
}

fn main() {
    let novel = String::from("Call me Ishmael. Some years ago...");
    let first_sentence = novel.split('.').next().unwrap();
    let i = ImportantExcerpt {
        part: first_sentence,
    };
}

```

这个结构体有唯一一个字段 `part`，它存放了一个字符串 slice，这是一个引用。类似于泛型参数类型，必须在结构体名称后面的尖括号中声明泛型生命周期参数，以便在结构体定义中使用生命周期参数。这个注解意味着 `ImportantExcerpt` 的实例不能比其 `part` 字段中的引用存在的更久。

这里的 `main` 函数创建了一个 `ImportantExcerpt` 的实例，它存放了变量 `novel` 所拥有的 `String` 的第一个句子的引用。`novel` 的数据在 `ImportantExcerpt` 实例创建之前就存在。另外，直到 `ImportantExcerpt` 离开作用域之后 `novel` 都不会离开作用域，所以 `ImportantExcerpt` 实例中的引用是有效的。

生命周期省略（Lifetime Elision）

省略规则并不提供完整的推断：如果 Rust 在明确遵守这些规则的前提下变量的生命周期仍然是模棱两可的话，它不会猜测剩余引用的生命周期应该是什么。编译器会在可以通过增加生命周期注解来解决错误问题的地方给出一个错误提示，而不是进行推断或猜测。

函数或方法的参数的生命周期被称为 **输入生命周期** (*input lifetimes*)，而返回值的生命周期被称为 **输出生命周期** (*output lifetimes*)。

编译器采用三条规则来判断引用何时不需要明确的注解。第一条规则适用于输入生命周期，后两条规则适用于输出生命周期。如果编译器检查完这三条规则后仍然存在没有计算出生命周期的引用，编译器将会停止并生成错误。这些规则适用于 `fn` 定义，以及 `impl` 块。

- 第一条规则是编译器为每一个引用参数都分配一个生命周期参数。换句话说就是，函数有一个引用参数的就有一个生命周期参数：`fn foo<'a>(x: &'a i32)`，有两个引用参数的函数就有两个不同的生命周期参数，`fn foo<'a, 'b>(x: &'a i32, y: &'b i32)`，依此类推。
- 第二条规则是如果只有一个输入生命周期参数，那么它被赋予所有输出生命周期参数：`fn foo<'a>(x: &'a i32) -> &'a i32`。
- 第三条规则是如果方法有多个输入生命周期参数并且其中一个参数是 `&self` 或 `&mut self`，说明是个对象的方法 (method)(译者注：这里涉及 rust 的面向对象参见 17 章)，那么所有输出生命周期参数被赋予 `self` 的生命周期。第三条规则使得方法更容易读写，因为只需更少的符号。

方法定义中的生命周期注解

```
impl<'a> ImportantExcerpt<'a> {
    fn announce_and_return_part(&self, announcement: &str) -> &str {
        println!("Attention please: {announcement}");
        self.part
    }
}
```

静态生命周期

'static，其生命周期能够存活于整个程序期间。所有的字符串字面值都拥有 'static 生命周期

结合泛型类型参数、trait bounds 和生命周期

```
use std::fmt::Display;

fn longest_with_an_announcement<'a, T>(
    x: &'a str,
    y: &'a str,
    ann: T,
) -> &'a str
where
    T: Display,
{
    println!("Announcement! {ann}");
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

泛型类型参数意味着代码可以适用于不同的类型。trait 和 trait bounds 保证了即使类型是泛型的，这些类型也会拥有所需要的行为。由生命周期注解所指定的引用生命周期之间的关系保证了这些灵活多变的代码不会出现悬垂引用。而所有的这一切发生在编译时所以不会影响运行时效率！

Ch11: Writing Automated Tests

- How to write
- Controlling How Tests Are Run
- Test Organizations

How to write tests

编写测试函数：

- 设置任何所需的数据或状态
- 运行需要测试的代码
- 断言其结果是我们所期望的

Rust 提供的测试功能: `test` Attribute, some Macros, and `should_panic` Attribute.

Example:

```
pub fn add(left: usize, right: usize) -> usize {
    left + right
}

#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn it_works() {
        let result = add(2, 2);
        assert_eq!(result, 4);
    }
}
```

- `#[cfg(test)]` : 标记模块为测试模块, 只有在运行 `cargo test` 时才会编译和运行测试代码。
- `use super::*;` : 导入父模块, 这样可以直接调用父模块中的函数。
- `#[test]` : 标记函数为测试函数, 只有在运行 `cargo test` 时才会运行测试函数。
- `assert_eq!(result, 4);` : 断言函数的返回值等于 4。
- `cargo test` : 运行测试。
- `assert!(expression)` : 断言表达式为 true。
- `assert_ne!(exp1, exp2)` : 断言表达式 `exp1 != exp2`。
- `assert_eq!(exp1, exp2)` : 断言表达式 `exp1 == exp2`。
- `assert_approx_eq!(exp1, exp2, epsilon)` : 断言表达式 `exp1` 与 `exp2` 差值小于等于 `epsilon`。
- `assert_ne_precise!(exp1, exp2)` : 断言表达式 `exp1` 与 `exp2` 差值大于 `f32::EPSILON` 或 `f64::EPSILON`。
- `assert_xx(abc, def, info)` : 断言 `abc` 与 `def` xx, `info` 为附加失败信息。

```

pub struct Guess {
    value: i32,
}

impl Guess {
    pub fn new(value: i32) -> Guess {
        if value < 1 {
            panic!(
                "Guess value must be less than or equal to 100, got {value}."
            );
        } else if value > 100 {
            panic!(
                "Guess value must be greater than or equal to 1, got {value}."
            );
        }
        Guess { value }
    }
}

#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    #[should_panic]
    fn greater_than_100() {
        Guess::new(200);
    }

    #[test]
    #[should_panic(expected = "less than or equal to 100")]
    fn greater_than_100_second() {
        Guess::new(101);
    }
}

```

- `#[should_panic]` : 标记测试函数期望 panic。
- `#[should_panic(expected = "less than or equal to 100")]` : 标记测试函数期望 panic 信息包含 "less than or equal to 100"。

将 `Result<T, E>` 用于测试

```

pub fn add(left: usize, right: usize) -> usize {
    left + right
}

#[cfg(test)]
mod tests {
    use super::*;

    // ANCHOR: here
    #[test]
    fn it_works() -> Result<(), String> {
        let result = add(2, 2);

        if result == 4 {
            Ok(())
        } else {
            Err(String::from("two plus two does not equal four"))
        }
    }
    // ANCHOR_END: here
}

```

- `it_works` 函数返回 `Result<(), String>` 类型
- 测试函数体在测试通过时返回 `Ok(())`，测试失败时返回 `Err(String)`

Controlling How Tests Are Run

- 可以将一部分命令行参数传递给 `cargo test`，而将另外一部分传递给生成的测试二进制文件。
- 为了分隔这两种参数，需要先列出传递给 `cargo test` 的参数，接着是分隔符 `--`，再之后是传递给测试二进制文件的参数。
- 运行 `cargo test --help` 会提示 `cargo test` 的有关参数，而运行 `cargo test -- --help` 可以提示在分隔符之后使用的有关参数。
- 并行或连续的运行测试
 - `cargo test -- --test-threads=1`
- 显示函数输出
 - `cargo test -- --show-output`
- 通过指定名字来运行部分测试
 - `cargo test one_hundred`：运行名为 `one_hundred` 的测试函数
 - `cargo test add`：指定部分测试的名称，运行所有名称中包含 `add` 的测试函数
- 除非特别指定否则忽略某些测试
 - `#[ignore]`：标记测试函数为忽略的测试函数。
 - `cargo test -- --ignored`：运行所有被标记为忽略的测试函数。
 - `cargo test -- --include-ignored`：运行所有测试函数，包括被标记为忽略的测试函数。

Ch12: An I/O Project

- parse cmd args
- read the file
- reconstruct: dispatch
- tests driver'
- env variables
- stderr

二进制项目的关注分离

`main` 函数负责多个任务的组织问题在许多二进制项目中很常见。所以 Rust 社区开发出一类在 `main` 函数开始变得庞大时进行二进制程序的关注分离的指导。这些过程有如下步骤:

- 将程序拆分成 `main.rs` 和 `lib.rs` 并将程序的逻辑放入 `lib.rs` 中。
- 当命令行解析逻辑比较小时，可以保留在 `main.rs` 中。
- 当命令行解析开始变得复杂时，也同样将其从 `main.rs` 提取到 `lib.rs` 中。

经过这些过程之后保留在 `main` 函数中的责任应该被限制为:

- 使用参数值调用命令行解析逻辑
- 设置任何其他的配置
- 调用 `lib.rs` 中的 `run` 函数
- 如果 `run` 返回错误，则处理这个错误

这个模式的一切就是为了关注分离: `main.rs` 处理程序运行，而 `lib.rs` 处理所有的真正的任务逻辑。因为不能直接测试 `main` 函数，这个结构通过将所有的程序逻辑移动到 `lib.rs` 的函数中使得我们可以测试它们。仅仅保留在 `main.rs` 中的代码将足够小以便阅读就可以验证其正确性。

Ch13: Functional Language Features: Iterators and Closures

Closures: Anonymous Functions that capture their Environment

Iterators

Ch14: More about Cargo and Creates.io

Ch15: Smart Pointers

- Using `Box<T>` to Point to Data on the Heap
- Treating Smart Pointers Like Regular References with the `Deref` Trait
- Running Code on Cleanup with `Drop` Trait
- `Rc<T>`, the Reference Counted Smart Pointer
- `RefCell<T>` and the Interior Mutability Pattern 内部可变性模式
- Reference Cycles Can Leak Memory 循环引用会导致内存泄漏

智能指针，另一方面，是一种类似于指针的数据结构，但还具有额外的元数据和功能。智能指针的概念并非特有于 Rust：智能指针起源于 C++，也存在于其他语言中。Rust 标准库中定义了各种智能指针，它们提供的功能超越了引用所能提供的。为了探讨一般概念，我们将查看几个不同的智能指针示例，包括引用计数智能指针类型。此指针允许您通过跟踪所有者的数量来允许数据有多个所有者，当没有所有者时，清理数据。

Rust 拥有 `所有权` 和 `借用` 概念，在引用和智能指针之间有额外的区别：引用仅 `借用` 数据，而在许多情况下，智能指针 `拥有` 它们所指向的数据。

智能指针通常使用结构体实现。智能指针不同于结构体的地方在于其实现了 `Deref` 和 `Drop` trait。`Deref` trait 允许智能指针结构体实例表现的像引用一样，这样就可以编写既用于引用、又用于智能指针的代码。`Drop` trait 允许我们自定义当智能指针离开作用域时运行的代码。

这里将会讲到的是来自标准库中最常用的一些：

- `Box<T>`，用于在堆上分配值
- `Rc<T>`，一个引用计数类型，其数据可以有多个所有者
- `Ref<T>` 和 `RefMut<T>`，通过 `RefCell<T>` 访问。（`RefCell<T>` 是一个在运行时而不是在编译时执行借用规则的类型）。

另外我们会涉及 `内部可变性 (interior mutability)` 模式，这是不可变类型暴露出改变其内部值的 API。

我们也会讨论 `引用循环 (reference cycles)` 会如何泄漏内存，以及如何避免。

Box<T>

`Box<T>` 类型是一个智能指针，因为它实现了 `Deref` trait，它允许 `Box<T>` 值被当作引用对待。当 `Box<T>` 值离开作用域时，由于 `Box<T>` 类型 `Drop` trait 的实现，`box` 所指向的堆数据也会被清除。这两个 trait 对于在本章余下讨论的其他智能指针所提供的功能中，将会更为重要。让我们更详细的探索一下这两个 trait。

通过 Deref trait 将智能指针当作常规引用处理

为了启用 `*` 运算符的解引用功能，需要实现 `Deref` trait。

每次当我们在代码中使用 `*` 时，`*` 运算符都被替换成了先调用 `deref` 方法再接着使用 `*` 解引用的操作，且只会发生一次，不会对 `*` 操作符无限递归替换，解引用出上面 `i32` 类型的值就停止了，这个值与示例 15-9 中 `assert_eq!` 的 `5` 相匹配。

函数和方法的隐式 Deref 强制转换

`Deref 强制转换 (deref coercions)` 将实现了 `Deref` trait 的类型的引用转换为另一种类型的引用。例如，`Deref` 强制转换可以将 `&String` 转换为 `&str`，因为 `String` 实现了 `Deref` trait 因此可以返回 `&str`。`Deref` 强制转换是 Rust 在函数或方法传参上的一种便利操作，并且只能作用于实现了 `Deref` trait 的类型。当这种特定类型的引用作为实参传递给和形参类型不同的函数或方法时将自动进行。这时会有一系列的 `deref` 方法被调用，把我们提供的类型转换成了参数所需的类型。

`Deref` 强制转换的加入使得 Rust 程序员编写函数和方法调用时无需增加过多显式使用 `&` 和 `*` 的引用和解引用。这个功能也使得我们可以编写更多同时作用于引用或智能指针的代码。

Deref 强制转换如何与可变性交互

类似于如何使用 `Deref` trait 重载不可变引用的 `*` 运算符, Rust 提供了 `DerefMut` trait 用于重载可变引用的 `*` 运算符。

Rust 在发现类型和 trait 实现满足三种情况时会进行 Deref 强制转换:

- 当 `T: Deref<Target=U>` 时从 `&T` 到 `&U`。
- 当 `T: DerefMut<Target=U>` 时从 `&mut T` 到 `&mut U`。
- 当 `T: Deref<Target=U>` 时从 `&mut T` 到 `&U`。

头两个情况除了第二种实现了可变性之外是相同的: 第一种情况表明如果有一个 `&T`, 而 `T` 实现了返回 `U` 类型的 `Deref`, 则可以直接得到 `&U`。第二种情况表明对于可变引用也有着相同的行为。

第三个情况有些微妙: Rust 也会将可变引用强转为不可变引用。但是反之是 不可能 的: 不可变引用永远也不能强转为可变引用。因为根据借用规则, 如果有一个可变引用, 其必须是这些数据的唯一引用 (否则程序将无法编译)。将一个可变引用转换为不可变引用永远也不会打破借用规则。将不可变引用转换为可变引用则需要初始的不可变引用是数据唯一的不可变引用, 而借用规则无法保证这一点。因此, Rust 无法假设将不可变引用转换为可变引用是可能的。

使用 Drop Trait 运行清理代码

...

Rc 引用计数智能指针

RefCell 和内部可变性模式

Ch16: Fearless Concurrency

Ch17: Object Oriented Programming Features in Rust

- 面向对象语言特点
- 顾及不同类型值的 trait 对象
- 面向对象设计模式的实现

Ch18: Patterns and Matching

Ch19: Advanced Features

Ch20: Building a Multithreaded WebServer

Jekyll

Transform your plain text into static websites and blogs.\ 将纯文本转换为静态网站和博客。

Jekyll is a **static site generator** that runs on the Ruby programming language.

- Simple 简单的
 - No more databases, comment moderation, or pesky updates to install—just your content.
 - 不再需要安装数据库、评论审核或烦人的更新，只需安装您的内容。
 - [How Jekyll works](#)
- Static 静止的
 - [Markdown](#), [Liquid](#), HTML & CSS go in. Static sites come out ready for deployment.
 - 加入[Markdown](#)、[Liquid](#)、HTML 和 CSS。静态站点出来后即可部署。
 - [Jekyll template guide](#)
- Blog-aware 博客意识
 - Permalinks, categories, pages, posts, and custom layouts are all first-class citizens here.
 - 永久链接、类别、页面、帖子和自定义布局在这里都是一等公民。
 - [Migrate your blog](#)

Running in seconds, [Quickstart](#):

```
# make sure satisfy the [prerequisites](https://jekyllrb.com/docs/installation/#requirements)
# on Debian:
sudo apt-get install ruby-full build-essential
# on Ubuntu:
sudo apt-get install ruby-full build-essential zlib1g-dev

# Install the jekyll and bundler gems
gem install bundler jekyll
jekyll new my-awesome-site
cd my-awesome-site
bundle exec jekyll serve # --livereload
# => Now browse to http://localhost:4000
```

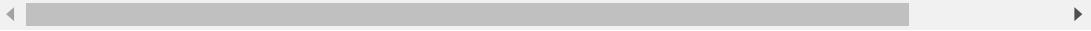
Problems:

```
gem install bundler jekyll
# Error:
# /usr/bin/ruby3.1 -I /usr/lib/ruby/vendor_ruby -r ./siteconf20241228-103530-c2f02s.rb extconf.rb
# mkmf.rb can't find header files for ruby at /usr/lib/ruby/include/ruby.h

# Solution:
sudo apt install ruby-dev

# Error
# sudo gem install jekyll
# Net::OpenTimeout: Failed to open TCP connection to github.com:443 (Connection timed out - user space limit reached)

# Solution:
# network error, just use proxy.
```



Note

- [How to Create a Static Website with Jekyll](#)
- [Andrej Karpathy blog - 2014](#)
- [Why you \(yes, you\) should blog - 2017](#)
- [Hexo Theme Nexmoe](#)
- [pages.github](#)
- [hhw-google-blogger](#)

Static Site Generator

A static site generator builds a website using plain HTML files. When a user visits a website created by a static site generator, it is loaded no differently than if you had created a website with plain HTML. By contrast, a dynamic site running on a server side language, such as PHP, must be built every time a user visits the site.

静态站点生成器使用纯 HTML 文件构建网站。当用户访问由静态网站生成器创建的网站时，其加载方式与您使用纯 HTML 创建网站时没有什么不同。相比之下，在服务器端语言（例如 PHP）上运行的动态站点必须在用户每次访问该站点时构建。

You can treat a static site generator as a very simple sort of CMS (content management system). Instead of having to include your entire header and footer on every page, for example, you can create a header.html and footer.html and load them into each page. Instead of having to write in HTML, you can write in Markdown, which is much faster and more efficient.

您可以将静态站点生成器视为一种非常简单的 CMS（内容管理系统）。例如，您不必在每个页面上包含整个页眉和页脚，您可以创建 header.html 和 footer.html 并将它们加载到每个页面中。您不必使用 HTML 编写，而是可以使用 Markdown 编写，这样更快、更高效。

Here are some of the main advantages of static site generators over dynamic sites:

以下是静态站点生成器相对于动态站点的一些主要优点：

- **Speed:** your website will perform much faster, as the server does not need to parse any content. It only needs to read plain HTML.
 - **速度:** 您的网站将执行得更快，因为服务器不需要解析任何内容。它只需要读取纯 HTML。
- **Security:** your website will be much less vulnerable to attacks, since there is nothing that can be exploited server side.
 - **安全性:** 您的网站将更不容易受到攻击，因为服务器端没有任何东西可以被利用。
- **Simplicity:** there are no databases or programming languages to deal with. A simple knowledge of HTML and CSS is enough.
 - **简单性:** 无需处理数据库或编程语言。了解 HTML 和 CSS 的简单知识就足够了。
- **Flexibility:** you know exactly how your site works, as you made it from scratch.
 - **灵活性:** 您确切地知道您的网站是如何运作的，因为您是从头开始创建的。

Of course, dynamic sites have their advantages as well. The addition of an admin panel makes for ease of updating, especially for those who are not tech-savvy. Generally, a static site generator would not be the best idea for making a CMS for a client. Static site generators also don't have the possibility of updating with real time content. It's important to understand how both work to know what would work best for your particular project.

当然，动态网站也有其优点。添加管理面板可以轻松更新，特别是对于那些不精通技术的人来说。一般来说，静态站点生成器并不是为客户制作 CMS 的最佳主意。静态站点生成器也无法更新实时内容。了解两者的工作原理非常重要，这样才能知道哪种方法最适合您的特定项目。

Ruby

- Ruby: A dynamic, open source programming language.
- RubyGems: A package manager for the Ruby programming language.
- Bundler: A tool for managing Ruby dependencies.

Ruby

A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.

一种动态的开源编程语言，重点关注简单性和生产力。它有一个优雅的语法：读起来自然，写起来也容易。

Features:

- Simple syntax,
- Basic OO features (classes, methods, objects, and so on),
- Special OO features (mixins, singleton methods, renaming, and so on),
- Operator overloading,
- Exception handling,
- Iterators and closures,
- Garbage collection,
- Dynamic loading (depending on the architecture),
- High transportability (runs on various Unices, Windows, DOS, macOS, OS/2, Amiga, and so on).

wikipedia:

Ruby is an interpreted, high-level, general-purpose programming language. It was designed with an emphasis on programming productivity and simplicity. In Ruby, everything is an object, including primitive data types. It was developed in the mid-1990s by Yukihiro "Matz" Matsumoto in Japan.

Ruby是一种解释型、高级、通用编程语言。它的设计重点是编程效率和简单性。在 Ruby 中，一切都是对象)，包括原始数据类型。它是由日本松本幸弘“Matz”于 20 世纪 90 年代中期开发的。

Ruby is dynamically typed and uses garbage collection) and just-in-time compilation. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. According to the creator, Ruby was influenced by Perl, Smalltalk, Eiffel), Ada), BASIC, Java), and Lisp).[10]#[citenote-about-10) [[3]](https://en.wikipedia.org/wiki/Ruby(programming_language)#cite_note-confreaks-3)

Ruby 是动态类型的，并使用垃圾收集)和即时编译。它支持多种编程范例，包括过程式编程、面向对象编程和函数式编程。根据创建者的说法，Ruby 受到Perl、Smalltalk、Eiffel)、Ada)、BASIC、Java)和Lisp)的影响。[10]#[citenote-about-10) [[3]](https://en.wikipedia.org/wiki/Ruby(programming_language)#cite_note-confreaks-3)

RubyGems

- [wikipedia](#)

RubyGems is a package manager for the Ruby programming language that provides a standard format for distributing Ruby programs and libraries (in a self-contained format called a "gem"), a tool designed to easily manage the installation of gems, and a server for distributing them. It was created by Chad Fowler, Jim Weirich, David Alan Black, Paul Brannan and Richard Kilmer in 2004.[2]

RubyGems 是 Ruby 编程语言的包管理器，它提供了用于分发 Ruby 程序和库的标准格式（以称为“gem”的独立格式）、一个旨在轻松管理 gems 安装的工具以及一个用于分发它们。它由 Chad Fowler 、 Jim Weirich 、 David Alan Black 、 Paul Brannan 和 Richard Kilmer 于 2004 年创建。 [2].

The interface for RubyGems is a command-line tool called gem which can install and manage libraries (the gems).[3] RubyGems integrates with Ruby run-time loader to help find and load installed gems from standardized library folders. Though it is possible to use a private RubyGems repository, the public repository is most commonly used for gem management.

RubyGems 的界面是一个名为 gem 的命令行工具，它可以安装和管理库（gems）。 [3] RubyGems 与 Ruby 运行时加载器集成，以帮助从标准化库文件夹中查找并加载已安装的 gem。尽管可以使用私有 RubyGems 存储库，但公共存储库最常用于 gem 管理。

The public repository helps users find gems, resolve dependencies and install them. RubyGems is bundled with the standard Ruby package as of Ruby 1.9.[4]

公共存储库可帮助用户查找 gem、解决依赖关系并安装它们。从 Ruby 1.9 开始，RubyGems 与标准 Ruby 包捆绑在一起。 [4]

Bundler

Bundler provides a consistent environment for Ruby projects by tracking and installing the exact gems and versions that are needed.

Bundler 通过跟踪和安装所需的确切 gem 和版本，为 Ruby 项目提供一致的环境。

Bundler is an exit from dependency hell, and ensures that the gems you need are present in development, staging, and production. Starting work on a project is as simple as bundle install.

Bundler 是摆脱依赖地狱的出口，并确保您需要的 gem 存在于开发、登台和生产中。开始项目工作就像 bundle install 一样简单。

Ruby-101 in jekyll

Gems

- **Gems** are code you can include in Ruby projects. Gems package specific functionality. You can share gems across multiple projects or with other people. Gems can perform actions like:
- Gems 是可以包含在 Ruby 项目中的代码。 Gems 封装特定功能。您可以跨多个项目或与其他人共享 gem。

Gemfile

- A **Gemfile** is a list of gems used by your site. Every Jekyll site has a Gemfile in the main folder.
- Gemfile 是您的站点使用的 gem 列表。每个 Jekyll 站点的主文件夹中都有一个 Gemfile。

Bundler

- **Bundler** is a gem that installs all gems in your Gemfile.
- To install gems in your Gemfile using Bundler, run the following in the directory that has the Gemfile:

```
bundle install  
bundle exec jekyll serve
```

- To bypass Bundler if you aren't using a Gemfile, run `jekyll serve`.

Ruby Version Manager ([RVM](#))

al-folio

- [running-local-al-folio - 2022](#)

Customize

```
.  
├── assets/: contains the assets that are displayed in the website  
│   └── json/  
│       └── resume.json: CV in JSON format (https://jsonresume.org/)  
├── _bibliography/  
│   └── papers.bib: bibliography in BibTeX format  
├── _config.yml: the configuration file of the template  
├── _data/: contains some of the data used in the template  
│   ├── cv.yml: CV in YAML format, used when assets/json/resume.json is not found  
│   ├── repositories.yml: users and repositories info in YAML format  
│   └── socials.yml: your social media and contact info in YAML format  
├── _includes/: contains code parts that are included in the main HTML file  
│   └── news.liquid: defines the news section layout in the about page  
├── _layouts/: contains the layouts to choose from in the frontmatter of the Markdown files  
├── _news/: the news that will appear in the news section in the about page  
├── _pages/: contains the pages of the website  
│   └── 404.md: 404 page (page not found)  
├── _posts/: contains the blog posts  
├── _projects/: contains the projects  
└── _sass/: contains the SASS files that define the style of the website  
    ├── _base.scss: base style of the website  
    ├── _cv.scss: style of the CV page  
    ├── _distill.scss: style of the Distill articles  
    ├── _layout.scss: style of the overall layout  
    ├── _themes.scss: themes colors and a few icons  
    └── _variables.scss: variables used in the SASS files
```


Great question! Let's go over each component and how they fit together.

1. GNOME

GNOME (GNU Network Object Model Environment) is a **desktop environment**. A desktop environment provides the graphical interface that users interact with, including the desktop, windows, icons, file managers, and settings applications. GNOME is one of the most popular desktop environments for Linux and is used in Debian, Ubuntu, Fedora, and other distributions.

Key components provided by GNOME include:

- **Window management:** Managing open applications and windows.
- **Applications:** Built-in utilities like the file manager, terminal, settings (i.e., `gnome-control-center`), and more.
- **User interface:** Consistent design and behavior for how things look and feel in the system.

GNOME uses a display protocol (like X11 or Wayland) to render graphics to the screen, which brings us to the next components.

2. X11 (X Window System)

X11, often called the **X Window System** or just **X**, is a **display server protocol** that provides the fundamental layer for graphical display on Unix-like operating systems. It was created in the 1980s and has been the standard display protocol for Linux desktops for a long time.

X11's primary responsibilities include:

- Managing **windows** (open applications) and their placement.
- Handling **input events** (mouse, keyboard).
- **Communication** between the hardware (graphics card) and applications.

GNOME traditionally ran on top of X11, which would handle all the graphical processing and window management on the display. However, X11 has some limitations, particularly with modern graphics hardware, security, and handling multiple screens, which led to the development of a more modern protocol: Wayland.

3. Wayland

Wayland is a **newer display server protocol** designed to replace X11. It was created to address some of X11's limitations and to provide a simpler, more secure, and efficient system for rendering graphics in modern desktop environments.

Wayland's advantages over X11 include:

- **Improved security:** Each application (client) has its own display server instance, reducing security risks.
- **Better performance:** It's designed to take advantage of modern graphics capabilities, resulting in smoother graphics and lower latency.
- **Simplicity:** Wayland's architecture is much simpler than X11, reducing the number of potential points of failure.

In Wayland, the compositor (in this case, GNOME's compositor called **Mutter**) manages windows, input events, and rendering directly, whereas in X11, this was handled by a separate **X server**.

4. Relationship Between GNOME, X11, and Wayland

- **GNOME** is the desktop environment that users interact with.
- **X11 or Wayland** is the display protocol that GNOME uses to render graphics to the screen. GNOME can run on top of either X11 or Wayland, depending on the configuration.
- In a **Wayland session**, GNOME directly communicates with the Wayland compositor (Mutter), which manages the display.
- In an **X11 session**, GNOME interacts with the X server, which acts as a middle layer between the hardware and the applications.

Each display protocol has its own strengths and weaknesses. Many Linux distributions, including Debian, have been moving towards using Wayland as the default due to its performance and security benefits. However, some applications and configurations still rely on X11, so X11 remains an option and fallback in many systems.

How This Relates to Your Issue

Your error message shows that `gnome-control-center` (the GNOME Settings app) encountered problems because it was running under Wayland but tried to interact with components in an X11 manner, resulting in a crash. This kind of error can happen when applications expect one protocol but the system defaults to another.

Switching to an X11 session (or forcing the GNOME Settings app to use X11) is a workaround to avoid these incompatibilities if certain applications or configurations are unstable under Wayland.

file explorer: nautilus See also: dolphin, caja, thunar, vifm.

- Launch Nautilus: nautilus
- Launch Nautilus as root user: nautilus admin:/
- Launch Nautilus and display a specific directory: nautilus path/to/directory
- Launch Nautilus with a specific file or directory selected: nautilus --select path/to/file_or_directory
- Launch Nautilus in a separated window: nautilus --new-window
- Close all Nautilus instances: nautilus --quit
- Display help: nautilus --help

settings: gnome-control-center

dpkg: Debian package manager.

- Install a package: dpkg -i path/to/file.deb
- Remove a package: dpkg -r package
- List installed packages: dpkg -l pattern
- List a package's contents: dpkg -L package
- List contents of a local package file: dpkg -c path/to/file.deb
- Find out which package owns a file: dpkg -S path/to/file

Current	Legacy
Files	Nautilus
Web	Epiphany
Text Editor	Gedit
Videos	Totem
Main Menu	Alacarte
Document Viewer	Evince
Disk Usage Analyzer	Baobab
Image Viewer	EoG (Eye of GNOME)
Passwords and Keys	Seahorse
GNOME Translation Editor	Gtranslator

outer disk mount: /media/hhw/Elements SE

hyprrland

[Github-Hyprrland](#)

yprrland is a 100% independent, dynamic tiling Wayland compositor that doesn't sacrifice on its looks. Hyprrland 是一个 100% 独立的动态平铺 Wayland 合成器，它不会牺牲其外观。

It provides the latest Wayland features, is highly customizable, has all the eyecandy, the most powerful plugins, easy IPC, much more QoL stuff than other compositors and more... 它提供最新的 Wayland 功能，高度可定制，拥有所有吸引眼球、最强大的插件、简单的 IPC，比其他合成器更多的 QoL 东西等等.....

Wayland

[text](#) [text](#)

Wayland is a replacement for the X11 window system protocol and architecture with the aim to be easier to develop, extend, and maintain. Wayland 是 X11 窗口系统协议和架构的替代品，旨在更易于开发、扩展和维护。

Wayland is the language (protocol) that applications can use to talk to a display server in order to make themselves visible and get input from the user (a person). A Wayland server is called a "compositor". Applications are Wayland clients. Wayland 是应用程序可以用来与显示服务器通信的语言（协议），以便自身可见并从用户（一个人）那里获得输入。Wayland 服务器称为“合成器”。应用程序是 Wayland 客户端。

Wayland also refers to a system architecture. It is not just a server-client relationship between a compositor and applications. There is no single common Wayland server like Xorg is for X11, but every graphical environment brings with it one of many compositor implementations. Window management and the end user experience are often tied to the compositor rather than swappable components. Wayland 还指系统架构。它不仅仅是合成器和应用程序之间的服务器-客户端关系。没有像 Xorg 那样用于 X11 的单一通用 Wayland 服务器，但每个图形环境都带来了许多合成器实现之一。窗口管理和最终用户体验通常与合成器而不是可交换组件相关联。

A core part of Wayland architecture is libwayland: an inter-process communication library that translates a protocol definition in XML to a C language API. This library does not implement Wayland, it merely encodes and decodes Wayland messages. The actual implementations are in the various compositor and application toolkit projects. Wayland 架构的核心部分是 libwayland：一个进程间通信库，可将 XML 中的协议定义转换为 C 语言 API。该库不实现 Wayland，它只是对 Wayland 消息进行编码和解码。实际实现位于各种合成器和应用程序工具包项目中。

Wayland does not restrict where and how it is used. A Wayland compositor could be a standalone display server running on Linux kernel modesetting and evdev input devices or on many other operating systems, or a nested compositor that itself is an X11 or Wayland application (client). Wayland can even be used in application-internal communication as is done in some web browsers. Wayland 不限制它的使用地点和方式。Wayland 合成器可以在 Linux 内核模式设置和 evdev 输入设备或许多其他操作系统上运行的独立显示服务器，也可以是本身是 X11 或 Wayland 应用程序（客户端）的嵌套合成器。Wayland 甚至可以用于应用程序内部通信，就像在某些 Web 浏览器中所做的那样。

Xwayland

[text](#)

Xwayland is an X server and a Wayland client. It plays the role of a proxy between legacy X clients which do not support the Wayland protocols and the Wayland server.

Xwayland 是一个 X 服务器和一个 Wayland 客户端。它在不支持 Wayland 协议的遗留 X 客户端和 Wayland 服务器之间扮演代理的角色。

configure Debian 12 system to use Wayland for all components that support it

Yes, you can configure your Debian 12 system to use Wayland for all components that support it. This might help resolve compatibility issues by ensuring everything is consistently running on Wayland without falling back to X11. Here's how to do it:

1. Ensure GNOME is Running on Wayland

By default, Debian 12 GNOME should be set to use Wayland unless it's explicitly disabled. Here's how to confirm or switch to a Wayland session:

1. **Log Out** of your current GNOME session.
2. **On the login screen**, select your username, and before logging back in, look for a **gear icon** near the Sign In button.
3. Click the gear icon and select **GNOME** or **GNOME on Wayland** (not GNOME on Xorg).
4. Log back in.

If you don't see the option for **GNOME on Wayland**, you may need to ensure that Wayland is enabled in the GNOME Display Manager (GDM) configuration file.

2. Enable Wayland in GDM (GNOME Display Manager)

If Wayland is disabled in GDM, follow these steps to re-enable it:

1. Open a terminal and edit the GDM configuration file:

```
sudo nano /etc/gdm3/custom.conf
```

2. Look for the line:

```
#WaylandEnable=false
```

3. Uncomment it and set it to `true`, or remove the line entirely to allow Wayland by default:

```
WaylandEnable=true
```

4. Save and close the file.

5. Restart GDM to apply the changes:

```
sudo systemctl restart gdm3
```

3. Set Applications to Prefer Wayland

Most GNOME applications and modern GTK apps should automatically use Wayland if the system is in a Wayland session. However, some applications (especially older or custom ones) may default to X11. Here's how to ensure that applications favor Wayland:

- **Check environment variables:** Wayland applications typically recognize `WAYLAND_DISPLAY`. To make sure applications use Wayland, you can set this variable globally:

1. Open the global profile configuration:

```
sudo nano /etc/environment
```

2. Add the following line:

```
WAYLAND_DISPLAY=wayland-0
```

3. Save and close the file.

4. Log out and log back in for changes to take effect.

- **Force specific applications to use Wayland:** Some applications (like Firefox and Electron-based apps) have specific flags to prefer Wayland. Here are a few examples:

- **Firefox:** Run Firefox with:

```
MOZ_ENABLE_WAYLAND=1 firefox
```

- **Electron apps:** For Electron apps (like Visual Studio Code), start them with:

```
code --enable-features=UseOzonePlatform --ozone-platform=wayland
```

4. Verify that Everything is Running on Wayland

Once you've made these changes, you can verify that GNOME and your applications are running on Wayland:

- **Check GNOME session:** Open a terminal and run:

```
echo $XDG_SESSION_TYPE
```

If the output is `wayland`, then GNOME is running on Wayland.

- **Check individual applications:** For example, in `gnome-control-center` or Firefox, you can check which display protocol they're using:

- **In `gnome-control-center`:** Run `gnome-control-center` from the terminal and see if there are any `GdkX11`-related warnings (which would indicate it's trying to use X11). If no warnings appear, it's likely running on Wayland.
- **In Firefox:** Type `about:support` in the address bar and look for `window Protocol` under **Graphics**. It should say `Wayland`.

5. Restart the System

To ensure all components pick up the changes, restart your computer:

```
sudo reboot
```

After this, your GNOME session and most applications should fully run on Wayland, which should help avoid issues related to mixed X11 and Wayland components. If you encounter specific applications that still don't work well with Wayland, you might need to run them with the `GDK_BACKEND=x11` flag temporarily or wait for further compatibility improvements.

second

Yes, it is possible to configure your system to use **Wayland** for all components, which would resolve the problem where your GNOME desktop environment and applications are conflicting with X11.

If you want to set **Wayland** as the default display protocol for all components on your Debian 12 system, follow these steps:

1. Ensure Wayland is Installed

First, ensure that Wayland and the necessary components for GNOME are installed. This includes the **Wayland compositor** and the **GNOME Wayland backend**.

Run the following command to check if the necessary Wayland components are installed:

```
sudo apt install wayland-protocols gnome-shell wayland
```

2. Ensure GNOME is Running on Wayland

By default, GNOME should run on **Wayland** on systems that support it, unless there is a configuration forcing it to run on **X11**. You can verify that GNOME is using Wayland by checking the `GDK_BACKEND` environment variable, which should be set to `wayland`.

Step-by-Step to Force GNOME to Use Wayland:

- 1. Check Current Session** To check whether GNOME is running on Wayland or X11, open a terminal and run the following command:

```
echo $XDG_SESSION_TYPE
```

- If the output is `wayland`, GNOME is already using Wayland.
- If the output is `x11`, GNOME is using X11, and you'll need to configure it to use Wayland.

- 2. Log Out of the Current Session** Log out of your current GNOME session.

- 3. Choose GNOME on Wayland (at the Login Screen)**

- On the login screen, after entering your username, you should see a settings icon or a gear icon (usually in the lower-right corner or near the session selection).
- Select **GNOME on Wayland**.
- Then log back in.

This will start GNOME under **Wayland** rather than X11.

3. Disable X11 as a Fallback Option (Optional)

To make sure that your system does not fall back to X11 automatically, you can disable the ability to use X11 for GNOME.

1. Edit the GDM Configuration:

Open the file `/etc/gdm3/custom.conf` in a text editor with superuser privileges. For example, use `nano`:

```
sudo nano /etc/gdm3/custom.conf
```

2. Disable Wayland and Ensure X11 is Not Available:

Find the line:

```
#WaylandEnable=false
```

and make sure it is **commented out** (no `#` before it). This ensures that Wayland is enabled for GNOME.

Now, find the line:

```
#WaylandEnable=false
```

and make sure it is **disabled** (no `#` before it).

3. Restart GDM (GNOME Display Manager):

After making the changes, restart the GDM service (this controls the graphical login):

```
sudo systemctl restart gdm3
```

4. Reboot Your System:

Finally, reboot your computer to make sure everything is set properly:

```
sudo reboot
```

4. Verifying the System is Using Wayland

After logging back in, you can confirm that all components are using Wayland by running:

```
echo $XDG_SESSION_TYPE
```

It should output `wayland` to confirm the system is running on Wayland.

5. Check for Potential Problems with Wayland

While most modern GNOME applications and components should work well under Wayland, there might still be some legacy X11-specific issues or applications that might not fully support Wayland yet. In that case:

- **Fallback to X11 for specific apps:** If you encounter issues with specific applications, you can still run them under X11 by explicitly setting the `GDK_BACKEND=x11` variable for those applications only.
- **System-wide Wayland:** If you want to use Wayland across all applications and ensure that they are compatible with it, consider looking into compatibility layers or tools that support running X11 applications under Wayland. For example, you could use `xwayland`, which allows X11 applications to run on Wayland.

Conclusion

Once you've configured GNOME to use **Wayland** and ensured that all components are set up to run under it, your system should no longer experience the issues related to X11-compatibility. However, always be prepared for occasional compatibility issues with legacy applications, although this is becoming less of a problem over time as more software supports Wayland natively.

If you can't open the Settings application in Debian 12, the issue could be due to several reasons, including a missing or broken package, or a problem with the desktop environment (usually GNOME on Debian). Here are some steps to troubleshoot and fix the issue:

1. Try Opening Settings from the Terminal

Sometimes, running an application from the terminal provides error messages that can help diagnose the issue.

1. Open a terminal.
2. Type the following command to try launching Settings:

```
gnome-control-center
```

If there is an error message, note it down, as it can provide clues about what is going wrong.

2. Restart GNOME Shell

If `gnome-control-center` fails to start, it might be related to the GNOME Shell. Restarting GNOME might help:

1. Press `Alt + F2`.
2. Type `r` and press Enter to reload the GNOME Shell.

Then try opening the Settings app again.

3. Check and Reinstall the `gnome-control-center` Package

The Settings app in GNOME is provided by the `gnome-control-center` package. If it's missing or corrupt, reinstalling it might fix the issue.

1. Open a terminal.
2. Run the following commands:

```
sudo apt update  
sudo apt install --reinstall gnome-control-center
```

3. After reinstalling, try launching the Settings app again.

4. Check for Pending Updates

Outdated packages or system libraries can sometimes cause issues. Make sure your system is up-to-date:

```
sudo apt update && sudo apt upgrade
```

5. Clear GNOME Settings Cache

Sometimes, cache files for GNOME can get corrupted. Deleting them can reset configuration settings and fix problems.

1. Open a terminal.
2. Run these commands to clear the cache:

```
rm -rf ~/.cache/gnome-control-center
rm -rf ~/.config/gnome-control-center
```

3. Try launching Settings again.

6. Check Logs for Errors

If the above steps don't work, checking system logs can give further hints.

1. Open a terminal.
2. View recent log messages related to GNOME Control Center:

```
journalctl -xe | grep gnome-control-center
```

Look for any errors related to `gnome-control-center` in the output.

7. Reboot the System

As a final step, reboot the system to clear any temporary issues:

```
sudo reboot
```

After rebooting, try opening the Settings app again.

If the problem persists, the issue may be with GNOME itself or with deeper system settings.

second

`gnome-control-center`

```
(gnome-control-center:4566): GLib-GObject-WARNING **: 10:08:38.727: invalid cast from
'GdkWaylandToplevel' to 'GdkX11Surface'
```

```
(gnome-control-center:4566): GLib-GObject-WARNING **: 10:08:38.727: invalid cast from
'GdkWaylandDisplay' to 'GdkX11Display' [1] 4566 segmentation fault gnome-control-center
```

ans:

The error messages you're seeing indicate that `gnome-control-center` is attempting to cast Wayland-specific objects as X11 objects, which is causing it to crash with a segmentation fault. This is a common issue when there are conflicts between X11 and Wayland components, especially if the system is running Wayland but some applications are expecting X11.

Here are several ways to potentially fix this issue:

1. Start GNOME on X11 Instead of Wayland

If you're running GNOME with Wayland, you can switch to an X11 session to see if that resolves the issue.

1. **Log out** of your current session.

2. On the login screen, select your username, and before logging back in, look for a settings icon (usually a gear icon) near the login button.

3. Choose **GNOME on Xorg** or **GNOME on X11** instead of **GNOME on Wayland**.
4. Log back in and try opening the Settings app again.

2. Force `gnome-control-center` to Use X11 Temporarily

If you prefer to stay on Wayland but want to see if the issue is related to Wayland compatibility, you can try launching `gnome-control-center` with the `GDK_BACKEND` environment variable set to X11.

Open a terminal and run:

```
GDK_BACKEND=x11 gnome-control-center
```

This command forces `gnome-control-center` to use X11, which may help it avoid the Wayland-related segmentation fault.

3. Update Your System and GNOME Packages

This may be a known issue with your specific version of GNOME or Debian 12. Make sure your system and GNOME packages are fully updated.

```
sudo apt update && sudo apt upgrade
```

Then, try running the Settings app again.

4. Disable Wayland for GNOME (as a More Permanent Fix)

If you find that `gnome-control-center` works well under X11, you may want to disable Wayland entirely for GNOME.

1. Open the file `/etc/gdm3/custom.conf` in a text editor with superuser privileges, for example:

```
sudo nano /etc/gdm3/custom.conf
```

2. Look for the line:

```
#WaylandEnable=false
```

3. Uncomment it by removing the `#` so it reads:

```
WaylandEnable=false
```

4. Save the file and exit the editor.

5. Reboot the system:

```
sudo reboot
```

This will disable Wayland, so GNOME will use X11 by default. After rebooting, try launching the Settings app again.

5. Reinstall Related GNOME and Wayland Packages

It's possible that some Wayland libraries or GNOME components are misconfigured or outdated. Try reinstalling them:

```
sudo apt install --reinstall gnome-shell gnome-control-center libwayland-client0 libwayland-server0
```

Then restart your system and try opening Settings again.

If the issue continues, it might be a deeper compatibility issue with the Wayland implementation on your system, and sticking with X11 for GNOME might be the most stable solution until updates fix the problem.

Fcitx

Setup_Fcitx_5

由于很多不同地方的过渡阶段，没有适合每个地方的完美解决方案。请根据您的环境选择您自己的解决方案。基本上，您要做的是为桌面会话设置以下环境变量。

```
XMODIFIERS=@im=fcitx
GTK_IM_MODULE=fcitx
QT_IM_MODULE=fcitx
```

```
export XMODIFIERS=@im=fcitx
export GTK_IM_MODULE=fcitx
export QT_IM_MODULE=fcitx
```

Useful commands:

```
fcitx-diagnose # print diagnostic information
fcitx-configtool # open the configuration tool
# fcitx-remote
```

CSDN: Debian Linux Input Method

debian系统对中文输入法的支持少之又少，很多人会选择使用搜狗，但是对于大多数来说，会有各种各样的问题，所以这里将会介绍使用系统自带的fcitx输入法。

首先软件源更新（选）：

1.在终端中输入： sudo gedit /etc/apt/sources.list

2.在打开的文本中删除全部内容，粘贴上以下文本：

```
deb http://mirrors.163.com/debian/ jessie-updates main non-free contrib
deb http://mirrors.163.com/debian/ jessie-backports main non-free contrib
deb-src http://mirrors.163.com/debian/ jessie main non-free contrib
deb-src http://mirrors.163.com/debian/ jessie-updates main non-free contrib
deb-src http://mirrors.163.com/debian/ jessie-backports main non-free contrib
deb http://mirrors.163.com/debian-security/ jessie/updates main non-free contrib
deb-src http://mirrors.163.com/debian-security/ jessie/updates main non-free contrib
deb http://ftp.cn.debian.org/debian wheezy main contrib non-free
```

(此处包含163以及debian官方软件源)

3.点击保存，并关闭，回到终端，进行软件源同步

输入指令： sudo apt-get update

apt-get install fcitx-ui-classic && apt-get install fcitx-ui-light

- 5.点击菜单，找到应用: 输入法，并打开。
- 6.在用户设置中点击 fcitx 选项，并点击确定。
- 7.根据输入法配置中的提示，打开终端，输入指令（根据自身要求选择）：

```
sudo apt-get install fcitx-sunpinyin fcitx-googlepinyin fcitx-pinyin
```

(这里有三种拼音输入法：fcitx-sunpinyin，fcitx-googlepinyin 和 fcitx-pinyin，不需要的可以删掉)

```
sudo apt-get fcitx-table-wubi fcitx-table-wbpy
```

(两种五笔输入法：fcitx-table-wubi 和fcitx-table-wbpy)

```
sudo apt-get fcitx-table-cangjie
```

(繁体中文输入，只有一种)

- 8.安装通用的输入法码表：fcitx-table* 套件（必装！）

```
sudo apt-get fcitx-table*
```

- 9.应用程序支持（必装！）

```
sudo apt-get install fcitx-frontend-gtk2 fcitx-frontend-gtk3 fcitx-frontend-qt4
```

(fcitx-frontend-gtk2 和 fcitx-frontend-gtk3 必装，同时 fcitx-frontend-qt4 也建议一起装上)

- 10.最后重启，根据自己的快捷键启动输入法（默认 Ctrl+空格）

在右下脚会有小键盘，右键 --配置 可以选择各种输入选项——————

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明

原文链接：<https://blog.csdn.net/ieso/article/details/105274943>

LSB (Linux Standard Base) is a standardization effort by the Linux Foundation aimed at increasing compatibility among different Linux distributions. Its primary goal is to ensure that software applications can run on various distributions without requiring modification, promoting software interoperability.

Key Concepts of LSB

1. Standardization:

- LSB defines a set of standards for Linux distributions to follow, covering the core components of a Linux system, such as file system hierarchy, system libraries, and application programming interfaces (APIs).
- By adhering to these standards, developers can write applications that are more portable across compliant distributions.

2. Compatibility:

- LSB aims to minimize the differences among distributions by providing a consistent environment. This includes defining package formats, directory structures, and other elements that help applications function uniformly across various distributions.

3. Core Components:

- The LSB specification includes guidelines on system libraries, command-line utilities, file locations, and other core components necessary for application development.

LSB Modules

The LSB is divided into various modules, each addressing specific areas of the Linux environment. Some of the key modules include:

1. Core Module:

- Defines essential libraries and utilities required for applications to run.
- Includes specifications for basic system libraries, such as glibc, and common command-line tools.

2. Graphics Module:

- Focuses on graphics-related libraries and interfaces, ensuring compatibility for graphical applications.
- This module may include support for X Window System libraries and graphics rendering libraries.

3. Desktop Module:

- Specifies standards for desktop environments, such as GNOME or KDE, to ensure applications can integrate seamlessly into the user interface.
- Includes guidelines for desktop files, menus, and application launchers.

4. Printing Module:

- Addresses standards for printing in Linux, including common protocols and interfaces.
- Ensures that applications can interact with printers consistently across different distributions.

5. Web Module:

- Defines standards for web applications, including required libraries and services.
- Aims to facilitate the development of web-based applications that run across various Linux environments.

Implementation and Compliance

- **Compliance Testing:** Distributions that claim LSB compliance undergo testing to ensure they meet the established standards. This helps developers and users trust that their applications will work as intended on compliant systems.
- **Packaging:** Many distributions provide tools for packaging software that adheres to LSB standards, making it easier to distribute and install applications.

Benefits of LSB

- **Portability:** Developers can write code that runs on any LSB-compliant distribution, reducing the need for separate versions of software.
- **Easier Development:** With a standardized environment, developers can focus on creating applications rather than dealing with the nuances of different distributions.
- **Community Collaboration:** LSB fosters collaboration among various Linux distributions, encouraging a unified approach to development and application deployment.

Current Status

While LSB was widely adopted in the past, its relevance has diminished somewhat as distributions have evolved and some developers have opted for alternative packaging methods (like Snap or Flatpak) that focus on containerization. Nevertheless, understanding LSB remains important for those working with Linux systems, especially in environments where compatibility and standardization are critical.

text

text

Arch Linux 是一个独立开发的 x86-64 通用 GNU/Linux 发行版，它致力于通过遵循滚动发布模型来提供大多数软件的最新稳定版本。默认安装是一个最小的基本系统，由用户配置为仅添加有意需要的内容。

Debian12 bookworm Release Notes

5.1.5. Fcitx versions no longer co-installable 5.1.5. Fcitx 版本不再可共同安装

The packages fcitx and fcitx5 provide version 4 and version 5 of the popular Fcitx Input Method Framework. Following upstream's recommendation, they can no longer be co-installed on the same operating system. Users should determine which version of Fcitx is to be kept if they had co-installed fcitx and fcitx5 previously. 软件包 fcitx 和 fcitx5 提供流行的 Fcitx 输入法框架的版本 4 和版本 5。根据上游的建议，它们不能再共同安装在同一个操作系统上。如果用户之前已经共同安装了 fcitx 和 fcitx5，则应确定要保留哪个版本的 fcitx。

Before the upgrade, users are strongly encouraged to purge all related packages for the unwanted Fcitx version (*fcitx- for Fcitx 4, and fcitx5- for Fcitx 5*). When the upgrade is finished, consider executing the `im-config` again to select the desired input method framework to be used in the system. 在升级之前，强烈建议用户清除不需要的 Fcitx 版本的所有相关软件包（Fcitx 4 的 `fcitx-` 和 Fcitx 5 的 `fcitx5-`）。升级完成后，考虑再次执行 `im-config` 选择要在系统中使用的所需输入法框架。

text

```
sudo yum install -y httpd  
  
sudo rpm -i httpd-2.4.6-80.el7.centos.x86_64.rpm
```

Rocky Linux 8

[How To Install and Use Docker on Rocky Linux 8](#)

[How To Install Nginx on Rocky Linux 8](#)

Nautilus: Debian File Manager

nautilus-tips-tweaks

1. Enable quick file preview Quick preview is rather a handy feature for a file manager. KDE's Dolphin file manager provides it as a built-in feature.

You can preview files such as PDF, text, images, audio, etc. You can scroll documents while in preview.

In Nautilus, you need to install gnome-sushi to get this feature.

```
sudo apt install gnome-sushi
```

Now, close all instances of file manager and open it again. To see the preview, select a file and press the Space key.

Printing Drivers

Debian12

```
# hplip-gui: HP Linux Printing and Imaging - GUI utilities (Qt-based)
sudo apt install hplip-gui

# deps on:
hplip: HP Linux Printing and Imaging
cups: Common Unix Printing System - PPD/driver support, web interface.
```

HPLIP is composed of:

- System services to handle communications with the printers
- HP CUPS backend driver (hp:) with bi-directional communication with HP printers (provides printer status feedback to CUPS and enhanced HPIJS functionality such as 4-side full-bleed printing support)
- HP CUPS backend driver for sending faxes (hpfax:)
- hpcups CUPS Raster driver to turn rasterized input from the CUPS filter chain into the printer's native format (PCL, LIDIL, ...). (hpcups is shipped in a separate package)
- HPIJS Ghostscript IJS driver to rasterize output from PostScript(tm) files or from any other input format supported by Ghostscript, and also for PostScript(tm) to fax conversion support (HPIJS is shipped in a separate package)
- Command line utilities to perform printer maintenance, such as ink-level monitoring or pen cleaning and calibration
- GUI and command line utility to download data from the photo card interfaces in MFP devices
- GUI and command line utilities to interface with the fax functions
- A GUI toolbox to access all these functions in a friendly way
- HPAIO SANE backend (hpaio) for flatbed and Automatic Document Feeder (ADF) scanning using MFP devices

```
groups <username> # to check groups of a user
usermod -G <groupname> <username> # to add a user to a group
usermod -G <groupname> -d <directory> <username> # to change the home directory of a user
usermod -s <shell> <username> # to change the shell of a user

# give user sudo access
sudo vim /etc/sudoers
<username> ALL=(ALL) ALL # add the line above to the end of the file

# add user to sudo|wheel group
sudo usermod -aG sudo <username>
```

```
su
```

Switch shell to another user.

More information: <https://manned.org/su>.

- Switch to superuser (requires the root password):

```
su
```

- Switch to a given user (requires the user's password):

```
su username
```

- Switch to a given user and simulate a full login shell:

```
su - username
```

- Execute a command as another user:

```
su - username -c "command"
```

adduser

User addition utility.

More information: <https://manned.org/adduser>.

- Create a new user with a default home directory and prompt the user to **set** a password:

```
adduser username
```

- Create a new user without a home directory:

```
adduser --no-create-home username
```

- Create a new user with a home directory at the specified path:

```
adduser --home path/to/home username
```

- Create a new user with the specified shell **set** as the login shell:

```
adduser --shell path/to/shell username
```

- Create a new user belonging to the specified group:

```
adduser --ingroup group username
```

users

Display a list of logged **in** users.

See also: `useradd`, `userdel`, `usermod`.

More information: <https://www.gnu.org/software/coreutils/users>.

- Print logged **in** usernames:

```
users
```

- Print logged **in** usernames according to a given file:

```
users /var/log/wtmp
```

```
usermod

Modify a user account.

See also: users, useradd, userdel.

More information: https://manned.org/usermod.

- Change a username:
  sudo usermod -l|--login new_username username

- Change a user ID:
  sudo usermod -u|--uid id username

- Change a user shell:
  sudo usermod -s|--shell path/to/shell username

- Add a user to supplementary groups (mind the lack of whitespace):
  sudo usermod -a|--append -G|--groups group1,group2,... username

- Change a user home directory:
  sudo usermod -m|--move-home -d|--home path/to/new_home username
```

```

gpasswd

Administer /etc/group and /etc/gshadow.
More information: https://manned.org/gpasswd.

- Define group administrators:
  sudo gpasswd -A user1,user2 group

- Set the list of group members:
  sudo gpasswd -M user1,user2 group

- Create a password for the named group:
  gpasswd group

- Add a user to the named group:
  gpasswd -a user group

- Remove a user from the named group:
  gpasswd -d user group

(base) → ~ gpasswd
Usage: gpasswd [option] GROUP

Options:
  -a, --add USER           add USER to GROUP
  -d, --delete USER        remove USER from GROUP
  -h, --help                display this help message and exit
  -Q, --root CHROOT_DIR    directory to chroot into
  -r, --remove-password    remove the GROUP's password
  -R, --restrict            restrict access to GROUP to its members
  -M, --members USER,...   set the list of members of GROUP
  -A, --administrators ADMIN,...  set the list of administrators for GROUP

Except for the -A and -M options, the options cannot be combined.

```

delete user

In Debian 12, the `deluser` and `userdel` commands are used to delete users. However, if you can't find these commands, it might be due to a missing package or a typo. Here's how you can delete a user in Debian 12:

1. Using `deluser` (Recommended)

The `deluser` command is part of the `passwd` package and is the preferred method for removing users in Debian.

Steps:

1. Install the `passwd` package (if not already installed):

```
sudo apt update
sudo apt install passwd
```

2. Delete the user:

```
sudo deluser username
```

Replace `username` with the actual username you want to delete.

3. Delete the user's home directory and mail spool (optional):

```
sudo deluser --remove-home username
```

2. Using `userdel`

The `userdel` command is part of the `shadow-utils` package and is a lower-level tool for removing users.

Steps:

1. Install the `shadow-utils` package (if not already installed):

```
sudo apt update
sudo apt install shadow-utils
```

2. Delete the user:

```
sudo userdel username
```

3. Delete the user's home directory and mail spool (optional):

```
sudo rm -rf /home/username
sudo rm -rf /var/mail/username
```

Troubleshooting

- Command not found: If you get a "command not found" error, ensure that the `passwd` or `shadow-utils` package is installed.
- Typo: Double-check the spelling of the commands (`deluser` and `userdel`).

Summary

- Use `deluser` for a more user-friendly experience.
- Use `userdel` for a more manual approach.

- Always ensure the necessary packages are installed.

Let me know if you encounter any issues!

```
# /etc/apt/apt.conf.d/01proxy
# /etc/apt/apt.conf
Acquire::http::proxy "http://user:password@host:port/";

Acquire {
    HTTP::proxy "http://proxy_server:port/";
    HTTPS::proxy "http://proxy_server:port/";
}

apt search --names-only <package_name>
```

```

apt 2.6.1 (amd64)

Usage: apt-get [options] command
        apt-get [options] install|remove pkg1 [pkg2 ...]
        apt-get [options] source pkg1 [pkg2 ...]

apt-get is a command line interface for retrieval of packages
and information about them from authenticated sources and
for installation, upgrade and removal of packages together
with their dependencies.

Most used commands:
update - Retrieve new lists of packages
upgrade - Perform an upgrade
install - Install new packages (pkg is libc6 not libc6.deb)
reinstall - Reinstall packages (pkg is libc6 not libc6.deb)
remove - Remove packages
purge - Remove packages and config files
autoremove - Remove automatically all unused packages
dist-upgrade - Distribution upgrade, see apt-get(8)
dselect-upgrade - Follow dselect selections
build-dep - Configure build-dependencies for source packages
satisfy - Satisfy dependency strings
clean - Erase downloaded archive files
autoclean - Erase old downloaded archive files
check - Verify that there are no broken dependencies
source - Download source archives
download - Download the binary package into the current directory
changelog - Download and display the changelog for the given package

apt is a commandline package manager and provides commands for
searching and managing as well as querying information about packages.
It provides the same functionality as the specialized APT tools,
like apt-get and apt-cache, but enables options more suitable for
interactive use by default.

Most used commands:
list - list packages based on package names
search - search in package descriptions
show - show package details
install - install packages
reinstall - reinstall packages
remove - remove packages
autoremove - automatically remove all unused packages
update - update list of available packages
upgrade - upgrade the system by installing/upgrading packages

```

```
full-upgrade - upgrade the system by removing/installing/upgrading packages
edit-sources - edit the source information file
satisfy - satisfy dependency strings
```

```
apt-get purge docker-ce -y
apt list --installed | grep docker
```

```
sudo curl -x "http://127.0.0.1:7890" https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.gpg
```


Useful Links Collection

- 导航 s
 - CS 自学指南
 - 网站导航
 - linux.do 网站导航
- Blogs
 - linux.cn
 - liaoxuefeng
 - Andrej Karpathy blog - 2014
 - Why you (yes, you) should blog - 2017
 - Hacker News Daily
- Tools
 - pdf-to-txt
 - table to markdown
- Tutorials
 - GDB 调试多线程程序的总结
 - Linux 101
- Courses
 - CS615 -- System Administration
- Articles
 - What Every Programmer Should Know About Memory - Traditional Chinese
- Misc
 - gnu home zh
 - The best free stock photos, royalty free images & videos shared by creators.
- ToRead:
 - zhihu - Performance and Compatibility in the HongMeng Production Microkernel
 - zhihu - Programming Books to read
- Awesome Lists:
 - Awesome-Selfhosted
- Radio to Txt
 - blog: 几款免费的语音转文字工具推荐
 - 飞书妙记
 - buzz: Transcribe and translate audio offline on your personal computer. Powered by OpenAI's Whisper.
- Online Convert
 - onlineconvertfree
 - caj2pdf
 - caj2pdf
 - gentleltd
 - github: caj2pdf
- text tools

- delete all enters
 - doc
 - [zealdocs](#)
 - video download
 - [yt](#)
 - pkgs
 - [pkgs: Packages for Linux and Unix](#)
 - [repology: the packaging hub](#)
-

docling

<https://ds4sd.github.io/docling/>

这是一个由 IBM 开源的 Python 工具，专门用于将各类文档转化为适合生成式 AI 使用的工具。它能够将 PDF、DOCX、PPTX、图片、HTML、Markdown 等多种流行文档格式，导出为 Markdown 和 JSON 格式，支持多种 OCR 引擎（PDF）、统一的文档对象（DoclingDocument），轻松集成检索增强生成（RAG）和问答应用，适用于需要将文档作为生成式 AI 模型输入的场景。

best-of-ml-python

该项目提供了一个高质量的机器学习 Python 库列表，包含超过 900 个开源项目，并按照项目质量评分进行排名，每周更新一次。所有开源项目被分成了 30 多个分类，包括机器学习框架、数据可视化、自然语言处理、OCR、模型部署等，便于不同应用领域的开发者快速找到所需的机器学习工具和资源。

bananas: cross-platform screen sharing

bananas 官网：<https://getbananas.net>

云朵备份发布地址：<https://github.com/likeflyme/cloudbak>

云朵备份官网：<https://www.cloudbak.org/>

使用教程：<https://www.cloudbak.org/use/create-session.html>

魔镜：项目地址<https://github.com/idoottop/MagicMirror/blob/main/docs/cn/readme.md>

作者提供的安装包，提取密码：`4ro2`：<https://del-wang.lanzout.com/b01qdt5nba>

一键马赛克：<https://github.com/Ritr/publicTools>

Traymond 增强版：<https://github.com/tabris17/traymond>

壹印：<https://github.com/ggchivalrous/yiyin>

仿生阅读：<https://github.com/yitong2333/Bionic-Reading/blob/main/README-CN.md>

Bochs

欢迎来到Bochs IA-32仿真器项目

Bochs是一个用C++编写的高度可移植的开源 [IA-32\(x86\)PC仿真器](#)，可以在大多数流行的平台上运行。它包括对英特尔x86 CPU、普通I/O设备和自定义BIOS的仿真。Bochs可以编译模拟许多不同的x86 CPU，从早期的386到最新的x86-64英特尔和AMD处理器，甚至可能还没有进入市场。

Bochs能够在仿真中运行大多数操作系统，包括Linux、DOS或Microsoft Windows。Bochs最初由Kevin Lawton编写，目前由该项目维护。

Bochs可以被编译并以多种模式使用，其中一些模式仍在开发中。bochs的“典型”用途是提供完整的x86 PC仿真，包括x86处理器、硬件设备和内存。这允许你在工作站的模拟器中运行操作系统和软件，就像你有一台机器中拥有一台机器一样。例如，假设你的工作站是一个Unix/X11工作站，但你想运行Win'95应用程序。Bochs将允许你在Unix/X11工作站上运行Win'95和相关软件，在你的工作站上显示一个窗口，模拟PC上的显示器。

Bochs是一个**仿真器**（emulator）--不是**虚拟化软件**（virtualization software）。它可以在许多架构上移植：X86、ARM、MIPS等。这意味着它必须能够模拟每一条CPU指令。

这使Bochs区别于虚拟化解决方案，例如VirtualBox、VMWare等。那些项目提供了很好的用户体验和快速的性能，其代价是硬件限制、一些非确定性和一些必要的黑客攻击来使程序工作。

Bochs的仿真提供了一个可控的、精确的执行环境，但代价是速度/性能。这在某些情况下可能是有利的，例如：

- 在开发操作系统或引导程序时
- 当处理非常老的、关键任务的软件时
- 当对系统级代码进行逆向工程时

欲了解更多信息，请参见用户指南中的介绍部分

Compression

Types of Compression

以下是不同类型的压缩文件（zip、gz、tar.gz、tar.bz2、tar.xz）的区别和特点：

1. Zip 文件 (.zip)

- 格式：单文件归档格式，可包含多个文件和目录。
- 压缩：默认使用 DEFLATE 算法，但也支持其他算法。
- 常见用途：广泛用于跨平台的文件共享和分发。
- 特点：
 - 支持压缩和归档一步完成。
 - 允许随机访问单个文件，无需解压整个归档文件。
 - 支持密码保护和加密。
- 工具：Unix 类系统上的 `zip` 和 `unzip` 命令，各种图形工具，以及许多操作系统的原生支持。

2. Gzip 文件 (.gz)

- 格式：单文件压缩格式。
- 压缩：使用 DEFLATE 算法。
- 常见用途：通常用于压缩单个文件或与 tar 结合压缩多个文件/目录。
- 特点：
 - 压缩速度快，压缩比好。
 - 不能单独归档多个文件或目录，通常与 tar 一起使用。
- 工具：`gzip` 和 `gunzip` 命令。

3. 带 Gzip 压缩的 Tar 包 (.tar.gz 或 .tgz)

- 格式：结合 tar（归档）和 gzip（压缩）。
- 压缩：使用 gzip 进行压缩。
- 常见用途：Unix 类系统中常用于软件打包和分发。
- 特点：
 - 将多个文件和目录归档为一个文件。
 - 顺序访问，需要解压整个归档文件才能访问文件。
 - 广泛支持并用于软件分发。
- 工具：带有 `-czvf` 和 `-xzvf` 选项的 `tar` 命令。

4. 带 Bzip2 压缩的 Tar 包 (.tar.bz2)

- 格式：结合 tar（归档）和 bzip2（压缩）。
- 压缩：使用 Burrows-Wheeler 算法，压缩比更好但压缩速度较慢。
- 常见用途：用于压缩比比压缩速度更重要的场景。
- 特点：
 - 将多个文件和目录归档为一个文件。
 - 压缩比比 gzip 更好，但速度较慢。
 - 顺序访问。

- 工具: 带有 `-cjvf` 和 `-xjvf` 选项的 `tar` 命令。

5. 带 XZ 压缩的 Tar 包 (`.tar.xz`)

- 格式: 结合 tar (归档) 和 xz (压缩)。
- 压缩: 使用 LZMA2 算法, 提供非常高的压缩比和良好的速度。
- 常见用途: 在需要最大压缩比的情况下越来越受欢迎, 用于软件分发和备份。
- 特点:
 - 将多个文件和目录归档为一个文件。
 - 在提到的格式中压缩比最高。
 - 顺序访问。
- 工具: 带有 `-cJvf` 和 `-xJvf` 选项的 `tar` 命令。

主要区别

1. 压缩算法:
 - Zip: 默认使用 DEFLATE。
 - Gzip: DEFLATE。
 - Bzip2: Burrows-Wheeler。
 - XZ: LZMA2。
2. 压缩比:
 - Zip: 中等。
 - Gzip: 中等到好。
 - Bzip2: 高。
 - XZ: 非常高。
3. 压缩速度:
 - Zip: 快。
 - Gzip: 快。
 - Bzip2: 较慢。
 - XZ: 比 gzip 慢但比 bzip2 快。
4. 归档能力:
 - Zip: 是 (一步完成)。
 - Gzip: 否 (与 tar 一起使用)。
 - Bzip2: 否 (与 tar 一起使用)。
 - XZ: 否 (与 tar 一起使用)。
5. 文件访问:
 - Zip: 随机访问。
 - Gzip/Tar.gz: 顺序访问。
 - Bzip2/Tar.bz2: 顺序访问。
 - XZ/Tar.xz: 顺序访问。

总结

- Zip: 适合通用的、跨平台的文件共享, 支持随机访问。
- Gzip: 常用于压缩单个文件或与 tar 一起用于多个文件, 速度和压缩比平衡良好。
- Tar.gz: Unix 类系统中常见的软件打包方式, 压缩比中等。
- Tar.bz2: 压缩比比 gzip 好但速度较慢, 适合对压缩比要求更高的场景。
- Tar.xz: 压缩比最高, 适合需要最大压缩比的场景, 尽管速度较慢。

根据你的具体需求选择压缩比、速度和文件访问要求合适的格式。

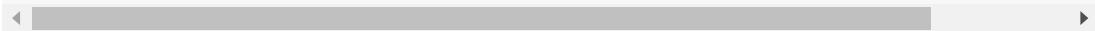
featurize

autodl

To use `scp` to copy a directory but exclude a specific subdirectory in Debian 12, you can achieve this by combining `tar` with SSH. The `scp` command itself does not directly support excluding files or directories, but you can use the following approach as a workaround:

Command:

```
tar --exclude='dir_father/dir_child1' -cf - dir_father | ssh user@remote_host 'cd /destination/path'
```



Explanation:

1. `tar` on the local machine:

- o `--exclude='dir_father/dir_child1'` : Skips the `dir_child1` directory during the `tar` process.
- o `-cf - dir_father` : Creates a tar archive of `dir_father` but writes it to standard output (`-`).

2. `ssh`:

- o Connects to the remote host.

3. `tar` on the remote machine:

- o `cd /destination/path` : Ensures the extracted files are placed in the correct destination.
- o `tar -xf -` : Reads the tar archive from standard input and extracts it.

Replace the following placeholders with actual values:

- `user` : Your username on the remote host.
- `remote_host` : The hostname or IP of the remote machine.
- `/destination/path` : The desired destination directory on the remote machine.

Notes:

- Ensure you have the necessary permissions on both the local and remote directories.
- If your `dir_child1` contains sensitive data or large files, double-check the exclusion path to avoid accidental transfer.

Let me know if you need further clarification!

Drawing

神经网络画图

- [gwding/draw_convnet \(github.com\)](https://github.com/gwding/draw_convnet)
- <https://cbovar.github.io/ConvNetDraw/>

Others

- [canva: Visual Suite for Everyone](#)
 - can draw simple icons from words, e.g. "NUDT SysY Compiler"
- [milanote: the tool for organizing creative projects](#)

Flatpack

- [flatpak](#)

The future of apps on Linux.

Change Sources:

```
flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.flatpakrepo

flatpak remotes --show-details # 显示flatpak官方源

flatpak remote-modify flathub --url=https://mirror.sjtu.edu.cn/flathub

# run gimp
flatpak run org.gimp.GIMP//stable
```

serach font on google, download the font file and install.

gui managers:

```
sudo apt install font-manager

fc-cache -fv
fc-list | grep "IBM Plex Serif"
```

Fuzz Testing

[wiki: Fuzz testing](#)

In [programming](#) and [software development](#), **fuzzing** or **fuzz testing** is an automated [software testing](#) technique that involves providing invalid, unexpected, or [random data](#) as inputs to a [computer program](#). The program is then monitored for exceptions such as [crashes](#)) "Crash (computing)", failing built-in code [assertions](#)) "Assertion (software development)", or potential [memory leaks](#). Typically, fuzzers are used to test programs that take structured inputs. This structure is specified, such as in a [file format](#) or [protocol](#) and distinguishes valid from invalid input. An effective fuzzer generates semi-valid inputs that are "valid enough" in that they are not directly rejected by the parser, but do create unexpected behaviors deeper in the program and are "invalid enough" to expose [corner cases](#) that have not been properly dealt with.

在编程和软件开发中，模糊测试或模糊测试是一种自动化软件测试技术，涉及提供无效、意外或随机的数据作为计算机程序的输入。然后，将监视程序是否存在异常，例如崩溃、内置代码断言失败或潜在的内存泄漏。通常，模糊测试程序用于测试采用结构化输入的程序。此结构是指定的，例如以文件格式或协议指定，并区分有效输入和无效输入。有效的模糊测试器会生成“足够有效”的半有效输入，因为它们不会被解析器直接拒绝，但确实会在程序的更深处产生意想不到的行为，并且“足够无效”以暴露未正确处理的极端情况。

- [afl](#)

AFL: American Fuzzy Lop

- [afl](#)
- [aflplusplus](#)
- [afl.rs](#)
- [rust-fuzz: AFL for Rust](#)

American fuzzy lop is a security-oriented [fuzzer](#) that employs a novel type of compile-time instrumentation and genetic algorithms to automatically discover clean, interesting test cases that trigger new internal states in the targeted binary. This substantially improves the functional coverage for the fuzzed code. The compact [synthesized corpora](#) produced by the tool are also useful for seeding other, more labor- or resource-intensive testing regimes down the road.

American fuzzy lop 是一个以安全为导向的 模糊器，采用新型编译时检测和遗传算法来自动发现干净、有趣的测试用例，从而触发目标二进制文件中的新内部状态。这大大提高了模糊代码的功能覆盖率。该工具生成的紧凑的合成语料库也可用于为将来的其他劳动或资源密集型测试制度奠定基础。

Compared to other instrumented fuzzers, *afl-fuzz* is designed to be practical: it has modest performance overhead, uses a variety of highly effective fuzzing strategies and effort minimization tricks, requires [essentially no configuration](#), and seamlessly handles complex, real-world use cases - say, common image parsing or file compression libraries.

与其他仪器化模糊器相比,*afl-fuzz*被设计为实用:它具有适度的性能开销,使用各种高效的模糊策略和努力最小化技巧,基本上不需要配置,并且可以无缝处理复杂的现实世界用例 - 例如,常见的图像解析或文件压缩库。

AFL.RS

[Fuzz testing](#) is a software testing technique used to find security and stability issues by providing pseudo-random data as input to the software. [AFLplusplus](#) is a popular, effective, and modern fuzz testing tool based on [AFL](#). This library, `afl.rs`, allows one to run AFLplusplus on code written in [the Rust programming language](#).

模糊测试是一种软件测试技术，用于通过提供伪随机数据作为软件的输入来发现安全性和稳定性问题。
AFLplusplus 是一种基于 AFL 的流行、有效、现代的模糊测试工具。这个库 `afl.rs` 允许人们在用 Rust 编程语言编写的代码上运行 AFLplusplus。

AFL: American Fuzzy Lop

**Issue 1: Debian12 default python3.11, if use conda env
python3.12, and build afl, afl need libpython3.12.so.1.0,**

solve: use python3.11 to build afl

recommend: change conda base env python version to 3.11

Issues and Solutions

Geographic Information System

GIS Software

[QIGS install on debian/ubuntu](#)

[equatorstudios](#)

GIS Libs

- [samgeo](#)
 - A Python package for segmenting geospatial data with the Segment Anything Model (SAM)
 - [doc: samgeo](#)

Shp File

How to Open and Work with Shapefiles and GeoJSON Files

To open and work with GeoJSON or Shapefiles using free and open-source software (FOSS), there are several excellent tools you can use. Here's a step-by-step guide:

1. QGIS (Quantum GIS)

QGIS is one of the most popular open-source Geographic Information System (GIS) software. It fully supports Shapefiles and GeoJSON files.

Steps:

1. Download QGIS:
 - Download and install QGIS from <https://qgis.org>.
 2. Open a Shapefile:
 - Launch QGIS.
 - Go to Layer > Add Layer > Add Vector Layer.
 - Browse to the Shapefile (.shp) file (you'll also need its associated .dbf and .shx files).
 - Click "Add" to display the Shapefile.
 3. Inspect and Edit:
 - Use the "Attributes Table" to view data.
 - Apply styling, run spatial analysis, or export the data.
 4. Open GeoJSON:
 - Similarly, go to Layer > Add Layer > Add Vector Layer, and browse to the .geojson file.
-

2. GDAL/OGR Tools (Command-line)

The GDAL library provides command-line tools to manipulate spatial data formats, including Shapefiles and GeoJSON.

Steps:

1. Install GDAL:
 - For Windows, download the GDAL binaries from [OSGeo4W](#).
 - For Linux/Mac, use package managers (`sudo apt install gdal-bin` or `brew install gdal`).
 2. View Shapefile Info:
 - Run `ogrinfo` to inspect the file:
-

```
ogrinfo -al -so your_file.shp
```

3. Convert Shapefile to GeoJSON:

- Use `ogr2ogr` to convert formats:

```
ogr2ogr -f "GeoJSON" output.geojson input.shp
```

3. GeoServer

GeoServer is a web-based server to visualize and share spatial data.

Steps:

1. Install GeoServer:

- Download it from <http://geoserver.org>.
- Run the GeoServer web interface.

2. Upload Shapefile/GeoJSON:

- Create a workspace and data store.
- Upload the Shapefile or GeoJSON to the data store.

3. View or Serve:

- Preview the data as a map.
- Serve it as a WMS/WFS for online use.

4. Python Libraries: GeoPandas and Fiona

If you're comfortable with Python, libraries like GeoPandas and Fiona are excellent tools.

Steps:

1. Install GeoPandas and Fiona:

```
pip install geopandas
pip install fiona
```

2. Read a Shapefile:

```
import geopandas as gpd
shapefile = gpd.read_file('your_file.shp')
print(shapefile.head())
```

3. Read a GeoJSON:

```
geojson = gpd.read_file('your_file.geojson')
print(geojson)
```

4. Plot or Analyze Data:

```
shapefile.plot()
```

5. Online Tools

You can use online FOSS tools like Mapshaper (for lightweight Shapefile and GeoJSON editing) at <https://mapshaper.org>.

Steps:

1. Visit the site and upload your file.
2. Edit, simplify, or convert between formats directly in the browser.

Summary

- QGIS: Best for comprehensive spatial data management.
- GDAL/OGR: Best for command-line and batch operations.
- GeoServer: Best for web-based visualization and sharing.
- GeoPandas/Fiona: Best for Python developers.
- Mapshaper: Lightweight and quick edits online.

All of these options are free and open-source!

GRUB

GRUB (GRand Unified Bootloader) 是默认的 Linux 分发版引导加载程序。它负责加载内核并将其控制权交给内核。它可以定制以加载不同的操作系统、不同的内核版本，甚至不同的启动配置。

要定制 GRUB，你可以编辑文件 `/etc/default/grub`。该文件包含 GRUB 的配置选项列表。你可以添加或删除选项以满足需求。

要使更改永久化，你可以使用命令 `update-grub`。这将生成一个新的文件 `/boot/grub/grub.cfg`，GRUB 在下次启动时将使用该文件。

要查看当前的 GRUB 配置，你可以运行命令 `grub-editenv`。这将在交互式 shell 中打开 GRUB 环境变量。

要向 GRUB 添加新的操作系统，可以在目录 `/etc/grub.d` 中创建新条目。该目录包含一系列由 GRUB 执行的 shell 脚本，用于生成 GRUB 配置。每个脚本负责向菜单中添加一个新条目。

GRand Unified Bootloader（大一统引导程序）

GRUB 试图为 IBM PC 兼容机提供一个引导加载程序，它既能为初学者或对技术不感兴趣的用户提供方便，又能灵活地帮助专家在不同的环境中使用。目前，它最适用于至少使用一种类似 UNIX 的免费操作系统的用户，但也可用于大多数 PC 操作系统。

这个项目的起因实际上是我们想在 IBM PC 兼容系统的 Mach4 上以符合多重引导标准的方式引导 GNU HURD 操作系统。然后，我尝试在 FreeBSD 使用的标准引导加载程序中添加对额外功能的支持。为了让所有功能都能正常工作，我必须做的事情越来越多，直到显然有必要从头开始另起炉灶。

GRUB 从一开始的多模块引导加载器发展到现在已经有很长的路要走了。它所使用的一些技术在自由软件世界中是独一无二的，还有一些技术显然也优于大多数专有操作系统。这里和多引导建议中的文档对未来的操作系统和 PC 引导加载程序编写者应该非常有用。

grub 命令

grub 命令的功能是用于交互式地管理 GRUB 引导程序。GRUB 是一个系统引导程序，可以服务于 Linux、Windows、FreeBSD 等常见操作系统，配置方式分为交互式和非交互式两种模式，用户只需要键入 `grub` 命令即可进入到“`grub>`”提示状态，然后通过常用命令及参数进行配置工作。原文链接：

<https://www.linuxcool.com/grub>

[Linux 黑话解释：Linux 中的 GRUB 是什么？ | Linux 中国 - 知乎 \(zhihu.com\)](#)

[GNU GRUB - GNU Project - Free Software Foundation \(FSF\)](#)

GitBook

Product documentation (your users will love)

Forget building your own custom docs platform. With GitBook you get beautiful documentation for your users, and a branch-based Git workflow for your team.

- [gitbook.com](#)
- [gitbook-ng.github.io](#)
- [gitbook-documentation zh](#)
- [gitbook-cli](#)
- [github: GitbookIO/gitbook](#)
- [GitbookIO/integrations](#)
- [Gitbook 打造的 Gitbook 说明文档](#)

GitBook 是基于 Node.js 的开源命令行工具，用于输出漂亮的电子书。

遗憾的是，GitBook 开源项目已经停止维护，专注打造的 [gitbook.com](#) 网站在国内访问受限。

Extensions

- [awesome-gitbook-plugins](#)
- [include-codeblock](#)
- [edit-link](#)
- [sharing](#)
- [terminull](#)
- [intopic-toc](#)
- [disqus](#)
- [github](#)
- [back-to-top-button](#)
- [download-pdf-link](#)
- [mermaid-newface](#)

Install

```
# install nvm
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash

# install node version 10.24.1
nvm install 10.24.1

# install gitbook-cli (with change npm source)
npm config set registry https://registry.npmmirror.com
npm install gitbook-cli -g

# install gitbook
gitbook -v

# 下载添加的插件 & build
gitbook install
gitbook build # generate static files under `_book` directory

# start server: localhost:4000
gitbook serve
```

格式

格式主要注重简单和易读性

GitBook 约定了下面这些文件的作用:

- README: 书本的介绍
- SUMMARY: 章节结构, 用来生成书本内容的预览表。
- LANGS: 多语言书籍
- GLOSSARY: 术语描述的清单

至少需要一个 README 和 SUMMARY 文件来构建一本书。

Hyper-V

提供硬件虚拟化

- [Windows 10 上的 Hyper-V 简介 | Microsoft Learn](#)
- [创建虚拟网络 | Microsoft Learn](#)

Hyper-V 是微软开发的一项虚拟化技术，在 Windows Server 和 Windows 10 操作系统中均可使用。简单来说，它可以让一个物理服务器运行多个虚拟机，并将物理资源（如 CPU、内存、磁盘空间等）分配给这些虚拟机，从而实现虚拟化。

使用 Hyper-V 可以在同一台计算机上运行多个操作系统，这些操作系统可以是 Windows、Linux 或其他操作系统。每个虚拟机可以独立运行，就像在物理硬件上运行一样。Hyper-V 还提供了一个虚拟机管理工具，可以管理虚拟机的配置、启动、停止等操作。

Hyper-V 具有很多优点，如资源共享、快速部署、灵活性和可扩展性等。它可以帮助组织节省成本并提高效率。同时，Hyper-V 也支持很多高级功能，如动态内存、虚拟机快照、远程管理等，使其更加强大和便捷。

总的来说，Hyper-V 是一项非常重要的虚拟化技术，它为组织和企业提供了强大的资源管理和利用能力，可以帮助实现更加高效的 IT 运营。

- 虚拟交换机: Hyper-V 使用虚拟交换机将虚拟机连接到物理网络。您可以创建新的虚拟交换机或使用现有的交换机。
- 虚拟机设置: 您可以配置虚拟机的各种设置，例如内存分配，处理器分配和网络适配器设置。
- 集成服务: Hyper-V 的集成服务可以提升虚拟机的性能和功能。您可以根据需要启用或禁用集成服务。
- Virtual switch: Hyper-V uses a virtual switch to connect virtual machines to your physical network. You can create a new virtual switch or use an existing one.
- Virtual machine settings: You can configure various settings for your virtual machine, such as memory allocation, processor allocation, and network adapter settings.
- Integration services: Hyper-V integration services enhance the performance and functionality of virtual machines. You can enable or disable integration services as needed.

《逻辑的引擎》 Martin Davis

Turing, His Machine and Computability Jurg Kohlas Register Machines are Turing Machines Colin B. Price

Markdown

vscode extenssion Markdown All in One

Key	Command
Ctrl/Cmd + B	Toggle bold
Ctrl/Cmd + I	Toggle italic
Alt+S (on Windows)	Toggle strikethrough1
Ctrl + Shift +]	Toggle heading (uplevel)
Ctrl + Shift + [Toggle heading (downlevel)
Ctrl/Cmd + M	Toggle math environment
Alt + C	Check/Uncheck task list item
Ctrl/Cmd + Shift + V	Toggle preview
Ctrl/Cmd + K V	Toggle preview to side

Tips



- [Complete list of github markdown emoji markup](#)
- [developer-icons](#)
- [devicon](#)

text

Node Package Manager (NPM)

```
# change npm registry source
npm config set registry https://registry.npmmirror.com

# check in verbose mode
npm install --verbose

# clean cache and reinstall
npm cache clean --force
npm install

# set proxy for npm
npm config set proxy http://proxy.example.com:8080
npm config set https-proxy http://proxy.example.com:8080

# update npm
npm install -g npm
```

Node Version Manager (NVM)

nvm

PNPM (Fast, disk space efficient package manager)

<https://pnpm.io/installation>

PowerShell

常用命令

Proxy settings

```
# C:\Users\ludiser\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1

function set_proxy0 { $env:HTTP_PROXY="http://127.0.0.1:7890" }
function set_proxy1 { $env:HTTPS_PROXY="https://127.0.0.1:7890" }
function unset_proxy0 { $env:HTTP_PROXY="" }
function unset_proxy1 { $env:HTTPS_PROXY="" }
```

删除

```
# 删除指定文件
Remove-Item * -Include *.json -Recurse
# 删除文件而保留文件夹
# 「This example deletes all of the files that have names that include a dot (.) 」
Remove-Item * -Include *.* -Exclude *.md -Recurse
# 删除包含指定字符的文件夹
# 一定要注意加上通配符「*bin*」，否则只会删除bin这样的文件夹
Remove-Item * -Recurse -Include *bin*
```

- 获取版本信息 `$PSVersionTable.PSVersion`
- 获取主机信息 `Get-Host`
- 下载文件 `Invoke-WebRequest / iwr`
- 命令 `get-command`
- 进程 `get-process`
- 指令重命名 `Set-Alias xxxx0 xxxx1`
- 清屏 `cls`
- 查找程序路径 `Get-Command -Name xxname`

Regular Expressions

- [wikipedia](#)
- [runoob](#)

```
[^a-z]      // 除了小写字母以外的所有字符
[^\\\/\^]    // 除了(\)(/)(^)之外的所有字符
[^"']       // 除了双引号(")和单引号(')之外的所有字符

a.*b        // a后面可以有任意字符，直到遇到b
```

存储技术比较

以下是SD（安全数字）、eMMC（嵌入式多媒体卡）和NVMe（非易失性存储器快车）这三种不同存储技术的比较：

SD（安全数字）

1. **类型**：可移动闪存卡。
2. **常见用途**：相机、智能手机、平板电脑及其他便携设备。
3. **外形尺寸**：小，有标准SD、miniSD和microSD等多种尺寸。
4. **速度**：根据类别和类型（如SDHC、SDXC、UHS-I、UHS-II）而有很大差异。
5. **接口**：使用SD接口，速度比PCIe和SATA接口慢。
6. **容量**：通常从几GB到几百GB不等。
7. **性能**：一般低于eMMC和NVMe，适合需求较低的存储任务。

eMMC（嵌入式多媒体卡）

1. **类型**：嵌入式不可移动闪存存储。
2. **常见用途**：智能手机、平板电脑、低端笔记本电脑、物联网设备及一些汽车应用。
3. **外形尺寸**：直接焊接在设备主板上。
4. **速度**：比SD卡快但比SSD（尤其是NVMe SSD）慢，通常在200-400 MB/s左右。
5. **接口**：使用类似SD的接口，但为嵌入式使用进行了优化。
6. **容量**：通常在16 GB到128 GB之间，但也有更高容量的产品。
7. **性能**：提供适中的性能，足以满足大多数消费类移动应用的需求，但不适合高性能计算任务。

NVMe（非易失性存储器快车）

1. **类型**：高性能SSD存储技术。
2. **常见用途**：高端笔记本电脑、台式机、服务器及企业存储解决方案。
3. **外形尺寸**：多种，包括M.2、U.2和PCIe扩展卡。
4. **速度**：显著快于SD和eMMC，读写速度超过3000 MB/s。
5. **接口**：使用PCIe接口，支持更高的数据传输速率和更低的延迟。
6. **容量**：从256 GB到数TB不等。
7. **性能**：高性能，适用于游戏、视频编辑和服务器工作负载等高要求应用。

主要差异

1. **外形尺寸**：
2. **SD**：可移动卡。
3. **eMMC**：嵌入式且不可移动。
4. **NVMe**：可以嵌入或用作扩展卡，有多种外形尺寸。
5. **性能**：
6. **SD**：性能最低，适用于基本存储需求。
7. **eMMC**：性能适中，满足大多数移动和消费类应用。
8. **NVMe**：性能最高，适用于高要求应用和高速数据传输。

9. 接口：

10. **SD**： SD接口。

11. **eMMC**： 类似SD但优化用于嵌入式使用。

12. **NVMe**： PCIe接口，速度更快，延迟更低。

13. 使用场景：

14. **SD**： 适合相机和便携设备等可移动存储需求。

15. **eMMC**： 适合成本效益高的消费电子嵌入式存储。

16. **NVMe**： 最佳选择用于计算机和企业应用中的高性能存储。

17. 容量：

18. **SD**： 容量通常低于eMMC和NVMe。

19. **eMMC**： 容量适中，常见于消费电子产品。

20. **NVMe**： 容量较大，满足更高的存储需求。

总结来说，SD卡适用于对性能要求较低的可移动存储，eMMC适合消费电子产品中的嵌入式应用，NVMe则提供最高性能，适用于高要求应用和高速数据传输。

SD (Secure Digital), eMMC (embedded MultiMediaCard), and NVMe (Non-Volatile Memory Express) are different types of storage technologies used in various devices. Here's a detailed comparison of each:

SD (Secure Digital)

1. **Type** : Removable flash memory card.
2. **Common Uses** : Cameras, smartphones, tablets, and other portable devices.
3. **Form Factor** : Small, with various sizes including standard SD, miniSD, and microSD.
4. **Speed** : Varies significantly by class and type (e.g., SDHC, SDXC, UHS-I, UHS-II).
5. **Interface** : Uses the SD interface, which is slower compared to PCIe and SATA interfaces.
6. **Capacity** : Typically ranges from a few GB to several hundred GB.
7. **Performance** : Generally lower than eMMC and NVMe, suitable for less demanding storage tasks.

eMMC (embedded MultiMediaCard)

1. **Type** : Embedded non-removable flash storage.
2. **Common Uses** : Smartphones, tablets, low-end laptops, IoT devices, and some automotive applications.
3. **Form Factor** : Soldered directly onto the device's motherboard.
4. **Speed** : Faster than SD cards but slower than SSDs (especially NVMe SSDs). Typically around 200-400 MB/s.
5. **Interface** : Uses an interface similar to SD, but optimized for embedded use.
6. **Capacity** : Usually ranges from 16 GB to 128 GB, though higher capacities are available.
7. **Performance** : Provides moderate performance, sufficient for most consumer mobile applications but not for high-performance computing tasks.

NVMe (Non-Volatile Memory Express)

1. **Type** : High-performance SSD storage technology.
2. **Common Uses** : High-end laptops, desktops, servers, and enterprise storage solutions.

- 3. **Form Factor** : Various, including M.2, U.2, and PCIe add-in cards.
- 4. **Speed** : Significantly faster than both SD and eMMC, with speeds exceeding 3,000 MB/s for reads and writes.
- 5. **Interface** : Uses the PCIe interface, allowing for much higher data transfer rates and lower latency.
- 6. **Capacity** : Ranges from 256 GB to several TB.
- 7. **Performance** : High performance, suitable for demanding applications like gaming, video editing, and server workloads.

Key Differences

1. Form Factor :

- 2. **SD** : Removable cards.
- 3. **eMMC** : Embedded and non-removable.
- 4. **NVMe** : Can be embedded or used as add-in cards, with various form factors.

5. Performance :

- 6. **SD** : Lowest performance, suitable for basic storage needs.
- 7. **eMMC** : Moderate performance, sufficient for most mobile and consumer applications.
- 8. **NVMe** : Highest performance, suitable for demanding applications and high-speed data transfer.

9. Interface :

- 10. **SD** : SD interface.
- 11. **eMMC** : Similar to SD but optimized for embedded use.
- 12. **NVMe** : PCIe interface, much faster and lower latency.

13. Usage Scenarios :

- 14. **SD** : Ideal for removable storage needs like cameras and portable devices.
- 15. **eMMC** : Suitable for cost-effective embedded storage in consumer electronics.
- 16. **NVMe** : Best for high-performance storage in computers and enterprise applications.

17. Capacity :

- 18. **SD** : Typically lower capacities compared to eMMC and NVMe.
- 19. **eMMC** : Moderate capacities, often seen in consumer electronics.
- 20. **NVMe** : Higher capacities, catering to more intensive storage requirements.

In summary, SD cards are ideal for removable storage with moderate performance needs, eMMC is suitable for embedded applications with moderate performance, and NVMe offers the highest performance for demanding applications and high-speed data transfer.

TWRP: Team Win Recovery Project

[TeamWin - TWRP](#)

TWRP是一个开源的第三方Recovery系统，用于替代Android设备的原生Recovery系统，提供更多功能和选项，如备份、恢复、刷机等，为用户提供更灵活和定制化的操作体验。TWRP支持多种Android设备，提供了直观的图形界面和触摸屏支持，使用户能够轻松进行批量操作，更方便地管理和维护自己的设备。

Autocorrelation Function (ACF)

The Autocorrelation Function (ACF) is a statistical tool used to measure the correlation between a time series and its lagged versions. In other words, it quantifies how similar a time series is to itself at different points in time. The ACF is widely used in time series analysis, particularly in the context of identifying patterns, trends, and seasonality, as well as in model building for forecasting.

Key Concepts:

1. **Lag:** The lag k represents the time difference between the observations in the time series. For example, if you have a monthly time series, a lag of 1 means you are comparing each month with the previous month, a lag of 2 means you are comparing each month with the month before the previous one, and so on.
2. **Correlation:** The correlation between two variables measures how closely they are related. In the context of the ACF, the correlation is between the time series and its lagged versions.
3. **Autocorrelation Coefficient:** The ACF at lag k is the correlation coefficient between the time series and its lagged version at lag k . It is denoted as ρ_k .

Mathematical Definition:

The autocorrelation function at lag k is given by:

$$\rho_k = \frac{\text{Cov}(X_t, X_{t-k})}{\text{Var}(X_t)}$$

where:

- X_t is the value of the time series at time t .
- X_{t-k} is the value of the time series at time $t-k$ (i.e., k time periods earlier).
- $\text{Cov}(X_t, X_{t-k})$ is the covariance between X_t and X_{t-k} .
- $\text{Var}(X_t)$ is the variance of the time series.

Interpretation:

- **Lag 0 ($k=0$):** The autocorrelation at lag 0 is always 1 because it represents the correlation of the time series with itself, which is perfect.
- **Positive Autocorrelation:** If $\rho_k > 0$, it indicates that the time series values at lag k are positively correlated. This means that high values in the time series tend to be followed by high values, and low values tend to be followed by low values.
- **Negative Autocorrelation:** If $\rho_k < 0$, it indicates that the time series values at lag k are negatively correlated. This means that high values tend to be followed by low values, and vice versa.
- **Zero Autocorrelation:** If $\rho_k \approx 0$, it indicates that there is no significant linear relationship between the time series and its lagged version at lag k .

Example:

Consider a simple time series representing the monthly sales of a product over a year:

$$\text{Sales} = \{100, 120, 110, 130, 140, 150, 160, 170, 180, 190, 200, 210\}$$

To calculate the ACF at lag 1, we compare each month's sales with the sales from the previous month:

Lag 1: {120, 110, 130, 140, 150, 160, 170, 180, 190, 200, 210}

We then calculate the correlation coefficient between the original series and the lagged series. If the correlation coefficient is positive and close to 1, it indicates a strong positive autocorrelation at lag 1, meaning that sales in one month are highly correlated with sales in the previous month.

Visual Representation:

The ACF is often visualized using a plot called the **Autocorrelation Plot** or **ACF Plot**. This plot shows the autocorrelation coefficients for different lags on the y-axis and the lags on the x-axis. The plot typically includes a dashed line representing the significance level, which helps in determining whether the autocorrelation coefficients are statistically significant.

Conclusion:

The Autocorrelation Function is a powerful tool in time series analysis that helps in understanding the internal structure of the data, identifying patterns, and selecting appropriate models. By examining the ACF, analysts can determine whether a time series is stationary, has a trend, or exhibits seasonality, which are crucial steps in building accurate forecasting models.

Differences Between ACF and PACF

The Autocorrelation Function (ACF) and the Partial Autocorrelation Function (PACF) are both essential tools in time series analysis, but they serve different purposes and provide different insights into the structure of the time series data. Here are the key differences between ACF and PACF:

1. Definition and Purpose

- **Autocorrelation Function (ACF):**
 - **Definition:** The ACF measures the correlation between a time series and its lagged versions. It quantifies how similar the time series is to itself at different points in time.
 - **Purpose:** The ACF helps in identifying the overall pattern of correlation in the time series, including both short-term and long-term dependencies.
- **Partial Autocorrelation Function (PACF):**
 - **Definition:** The PACF measures the correlation between a time series and its lagged versions, after removing the effects of shorter lags. It isolates the relationship between the time series and a specific lag, controlling for the influence of other lags.
 - **Purpose:** The PACF helps in identifying the direct relationship between the time series and a specific lag, after accounting for the effects of intermediate lags. It is particularly useful for determining the order of the autoregressive (AR) component in ARIMA models.

2. Mathematical Interpretation

- **ACF:**
 - The ACF at lag k is the correlation coefficient between the time series and its lagged version at lag k .
 - It is given by:
$$\rho_k = \frac{\text{Cov}(X_t, X_{t-k})}{\text{Var}(X_t)}$$
- **PACF:**
 - The PACF at lag k is the coefficient ϕ_{kk} in the autoregressive model of order k .
 - It is given by:

$$X_t = \phi_{k1}X_{t-1} + \phi_{k2}X_{t-2} + \cdots + \phi_{kk}X_{t-k} + \epsilon_t$$

- The PACF at lag k is the partial correlation between X_t and X_{t-k} , controlling for the effects of $X_{t-1}, X_{t-2}, \dots, X_{t-(k-1)}$.

3. Visual Interpretation

- **ACF Plot:**
 - The ACF plot shows the autocorrelation coefficients for different lags on the y-axis and the lags on the x-axis.
 - It helps in identifying the overall pattern of correlation in the time series, including both short-term and long-term dependencies.
 - Significant spikes in the ACF plot indicate the presence of autocorrelation at those lags.

- **PACF Plot:**

-
- The PACF plot shows the partial autocorrelation coefficients for different lags on the y-axis and the lags on the x-axis.
 - It helps in identifying the direct relationship between the time series and a specific lag, after accounting for the effects of intermediate lags.
 - Significant spikes in the PACF plot indicate the presence of partial autocorrelation at those lags, which is particularly useful for determining the order of the AR component in ARIMA models.

4. Use in Model Identification

- **ACF:**

- The ACF is useful for identifying the order of the moving average (MA) component in ARIMA models.
- A rapidly decaying ACF suggests that the time series is dominated by the MA component.

- **PACF:**

- The PACF is useful for identifying the order of the autoregressive (AR) component in ARIMA models.
- A rapidly decaying PACF suggests that the time series is dominated by the AR component.

Example

Consider a time series with the following characteristics:

- **ACF Plot:** The ACF plot shows significant spikes at lags 1, 2, and 3, but the spikes decay rapidly after lag 3.
- **PACF Plot:** The PACF plot shows significant spikes at lags 1 and 2, but the spikes decay rapidly after lag 2.

Interpretation:

- The significant spikes in the ACF plot at lags 1, 2, and 3 suggest that the time series has a moving average component of order 3 (i.e., MA(3)).
- The significant spikes in the PACF plot at lags 1 and 2 suggest that the time series has an autoregressive component of order 2 (i.e., AR(2)).

Conclusion

The ACF and PACF are complementary tools in time series analysis. The ACF helps in identifying the overall pattern of correlation in the time series, while the PACF helps in isolating the direct relationship between the time series and specific lags, after accounting for the effects of intermediate lags. Together, they are essential for identifying the appropriate ARIMA model for forecasting.

ARIMA Time Series Prediction Model

The ARIMA (AutoRegressive Integrated Moving Average) model is a widely used statistical method for time series forecasting. It combines three components: AutoRegressive (AR), Integrated (I), and Moving Average (MA). The ARIMA model is denoted as ARIMA(p, d, q), where \$p\$, \$d\$, and \$q\$ are parameters that need to be determined.

Components of ARIMA

1. AutoRegressive (AR) Component (p):

- The AR component models the relationship between an observation and a number of lagged observations (i.e., previous values in the time series).
- The parameter \$p\$ represents the number of lag observations included in the model.
- For example, if \$p = 2\$, the model uses the two previous observations to predict the current value.

2. Integrated (I) Component (d):

- The I component represents the degree of differencing (i.e., the number of times the data have had past values subtracted).
- The parameter \$d\$ indicates the number of non-seasonal differences needed to make the time series stationary.
- Stationarity means that the statistical properties of the time series, such as mean and variance, are constant over time.

3. Moving Average (MA) Component (q):

- The MA component models the relationship between an observation and a residual error from a moving average model applied to lagged observations.
- The parameter \$q\$ represents the number of lag residual errors included in the model.
- For example, if \$q = 1\$, the model uses the error from the previous time step to predict the current value.

Determining the Proper \$p\$, \$d\$, and \$q\$

1. Determining \$d\$ (Differencing Order):

- The first step is to check if the time series is stationary. If not, differencing is applied to make it stationary.
- Use statistical tests like the Augmented Dickey-Fuller (ADF) test to check for stationarity.
- If the series is not stationary, apply differencing (i.e., subtract the previous value from the current value) and repeat the test.
- The number of times differencing is applied until the series becomes stationary is the value of \$d\$.

2. Determining \$p\$ (AR Order):

- Use the Autocorrelation Function (ACF) plot to identify the number of significant lags.
- The ACF plot shows the correlation between the time series and its lagged versions.
- The value of \$p\$ is determined by the number of significant lags in the ACF plot.

3. Determining \$q\$ (MA Order):

- Use the Partial Autocorrelation Function (PACF) plot to identify the number of significant lags.
- The PACF plot shows the correlation between the time series and its lagged versions, after removing the effects of shorter lags.
- The value of \$q\$ is determined by the number of significant lags in the PACF plot.

Example

Suppose you have a monthly sales time series and you want to build an ARIMA model. Here are the steps to determine p , d , and q :

1. Check for Stationarity:

- Apply the ADF test. If the series is not stationary, apply differencing once (i.e., $d = 1$) and check again.
- If the series becomes stationary after one differencing, $d = 1$.

2. Determine p Using ACF Plot:

- Plot the ACF for the differenced series.
- If the ACF shows significant lags at lags 1 and 2, but not beyond, then $p = 2$.

3. Determine q Using PACF Plot:

- Plot the PACF for the differenced series.
- If the PACF shows significant lags at lag 1, but not beyond, then $q = 1$.

Thus, the ARIMA model for this time series would be ARIMA(2, 1, 1).

Model Selection and Validation

- **Model Selection:** Once p , d , and q are determined, fit the ARIMA model to the data and evaluate its performance using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE).
- **Cross-Validation:** Use techniques like rolling forecasting origin or time series cross-validation to validate the model's performance on out-of-sample data.
- **Parameter Tuning:** Experiment with different combinations of p , d , and q to find the model that minimizes the chosen error metric.

Conclusion

The ARIMA model is a versatile tool for time series forecasting, but its effectiveness depends on the proper selection of the parameters p , d , and q . By carefully analyzing the ACF and PACF plots and ensuring the series is stationary, you can determine the appropriate ARIMA model for your time series data.

Partial Autocorrelation Function (PACF)

The Partial Autocorrelation Function (PACF) is a statistical tool used in time series analysis to measure the correlation between a time series and its lagged versions, after removing the effects of shorter lags. In other words, the PACF helps to isolate the relationship between the time series and a specific lag, controlling for the influence of other lags.

Key Concepts:

1. **Autocorrelation:** As discussed earlier, autocorrelation measures the correlation between a time series and its lagged versions. However, this correlation can be influenced by other lags.
2. **Partial Autocorrelation:** The partial autocorrelation at lag (k) (denoted as (ϕ_{kk})) is the correlation between the time series and its (k)-th lag, after removing the effects of all shorter lags (i.e., lags 1 through ($k-1$)).
3. **PACF Plot:** The PACF plot shows the partial autocorrelation coefficients for different lags on the y-axis and the lags on the x-axis. It helps in identifying the significant lags that are not explained by shorter lags.

Mathematical Definition:

The partial autocorrelation at lag (k) is the coefficient (ϕ_{kk}) in the following autoregressive model:

$$[X_t = \phi_{k1} X_{t-1} + \phi_{k2} X_{t-2} + \dots + \phi_{kk} X_{t-k} + \epsilon_t]$$

where:

- (X_t) is the value of the time series at time (t).
- (X_{t-k}) is the value of the time series at time ($t-k$) (i.e., (k) time periods earlier).
- (ϵ_t) is the error term at time (t).
- (ϕ_{kk}) is the partial autocorrelation coefficient at lag (k).

Interpretation:

- **Lag 0 ($k=0$):** The partial autocorrelation at lag 0 is always 1 because it represents the correlation of the time series with itself, which is perfect.
- **Positive Partial Autocorrelation:** If ($\phi_{kk} > 0$), it indicates that the time series values at lag (k) are positively correlated after controlling for shorter lags.
- **Negative Partial Autocorrelation:** If ($\phi_{kk} < 0$), it indicates that the time series values at lag (k) are negatively correlated after controlling for shorter lags.
- **Zero Partial Autocorrelation:** If ($\phi_{kk} \approx 0$), it indicates that there is no significant linear relationship between the time series and its lagged version at lag (k) after controlling for shorter lags.

Example:

Consider a simple time series representing the monthly sales of a product over a year:

$$[\text{Sales} = \{100, 120, 110, 130, 140, 150, 160, 170, 180, 190, 200, 210\}]$$

To calculate the PACF at lag 2, we need to fit an autoregressive model of order 2 and extract the coefficient (ϕ_{22}):

1. **Fit the AR(2) Model:** [$X_t = \phi_{21} X_{t-1} + \phi_{22} X_{t-2} + \epsilon_t$] Using the sales data, estimate the coefficients (ϕ_{21}) and (ϕ_{22}).

-
2. **Extract ($\backslash\phi_{\{22\}}$)**: The coefficient ($\backslash\phi_{\{22\}}$) represents the partial autocorrelation at lag 2, controlling for the effect of lag 1.

Visual Representation:

The PACF is often visualized using a plot called the **Partial Autocorrelation Plot** or **PACF Plot**. This plot shows the partial autocorrelation coefficients for different lags on the y-axis and the lags on the x-axis. The plot typically includes a dashed line representing the significance level, which helps in determining whether the partial autocorrelation coefficients are statistically significant.

Conclusion:

The Partial Autocorrelation Function (PACF) is a crucial tool in time series analysis, particularly in identifying the order of the autoregressive (AR) component in ARIMA models. By examining the PACF, analysts can determine which lags are significant after controlling for the effects of shorter lags, which is essential for building accurate forecasting models.

Typst

Links

- [Typst](#)
- [polylux: package to create presentation slides.](#)

Reference

#show rule

```
#show "ArtosFlow": name => box[
    #box(image(
        "logo.svg",
        height: 0.7em,
    ))
    #name
]
```

This report is embedded in the ArtosFlow project. ArtosFlow is a project of the Artos Institute.

With show rules, you can redefine how Typst displays certain elements. You specify which elements Typst should show differently and how they should look. Show rules can be applied to instances of text, many functions, and even the whole document. 使用 show rules，您可以重新定义 Typst 显示某些元素的方式。您可以指定 Typst 应该以不同的方式显示哪些元素以及它们应该如何显示。显示规则可以应用于文本实例、许多函数，甚至整个文档。

There is a lot of new syntax in this example: We write the show keyword, followed by a string of text we want to show differently and a colon. Then, we write a function that takes the content that shall be shown as an argument. Here, we called that argument name. We can now use the name variable in the function's body to print the ArtosFlow name. Our show rule adds the logo image in front of the name and puts the result into a box to prevent linebreaks from occurring between logo and name. The image is also put inside of a box, so that it does not appear in its own paragraph.

这个例子中有很多新的语法：我们编写 show 关键字，后跟我们想要以不同方式显示的一串文本和一个冒号。然后，我们编写一个函数，该函数将应显示为参数的内容。在这里，我们将该参数称为 name。我们现在可以使用函数体中的 name 变量来打印 ArtosFlow 名称。我们的 show 规则将 logo 图像添加到名称前面，并将结果放入一个框中，以防止 logo 和 name 之间出现换行。图像也被放在一个框内，这样它就不会出现在自己的段落中。

VScode

Shortcuts

- `Ctrl+p` 快速打开文件
- `Ctrl+shift+p` 打开命令面板
- `Ctrl+w` 关闭当前文件
- `Ctrl+k+w` 关闭所有文件
- `Ctrl+f` 快速搜索
- `Ctrl+g` 跳转到指定行
- `Ctrl+Alt+f` Find in Explorer
- `Ctrl+R` Run Recent Command

Others

文件恢复

在 vscode 界面，按住 `Ctrl+Shift+P` 打开命令面板，找到本地历史记录

VScode server

确认 server commit_id

```
# ~/.vscode-server/bin
# 下载对应的server程序
# (注意把:${commit_id}替换成对应的Commit ID)
https://update.code.visualstudio.com/commit:\${commit\_id}/server-linux-x64/stable
# vscode-server-linux-x64.tar.gz
# 放到 ~/.vscode-server/bin/${commit_id}/ 文件夹下
# 解压
tar -zvxf vscode-server-linux-x64.tar.gz --strip=1

https://update.code.visualstudio.com/commit:e7e037083ff4455cf320e344325dacb480062c3c/server-linux-x64/stable
```

Debugging Launch Configuration

- Debugging Launch Configuration

- nobepay
-

Windows Subsystem for Linux

- 工作原理
- 配置文件
- 备份还原
- 网络
- 文件系统
- 内核替换

Changing the MAC address of your laptop can help with privacy or troubleshooting network issues. Here's how to do it for different operating systems:

For Windows

1. Open Device Manager:

- o Right-click on the Start button and select **Device Manager**.

2. Locate Network Adapters:

- o Find **Network adapters** and expand the section. Right-click on your Wi-Fi adapter and select **Properties**.

3. Change MAC Address:

- o Go to the **Advanced** tab.
- o In the **Property** box, select **Network Address or Locally Administered Address**.
- o In the **Value** box, enter your new MAC address (without colons or dashes). For example, `00AABBCCDDEE`. Make sure the address is valid (it should be 12 hexadecimal digits).

4. Restart Your Adapter:

- o Disable and re-enable your network adapter for the changes to take effect.

For Linux

1. Open Terminal.

2. Identify Your Network Interface:

- o Run the command:

```
ip link
```

- o Note the name of your network interface (e.g., `wlan0`).

3. Bring Down the Interface:

- o Replace `wlan0` with your interface name:

```
sudo ip link set wlan0 down
```

4. Change the MAC Address:

- o Run:

```
sudo ip link set wlan0 address XX:XX:XX:XX:XX:XX
```

- o Replace `XX:XX:XX:XX:XX:XX` with your desired MAC address.

5. Bring Up the Interface:

- o Run:

```
sudo ip link set wlan0 up
```

6. Verify the Change:

- Check your new MAC address:

```
ip link show wlan0
```

For macOS

1. **Open Terminal.**
2. **Find Your Network Interface:**

- Run the command:

```
ifconfig
```

- Identify your Wi-Fi interface (usually `en0`).

3. **Change the MAC Address:**

- Run the following command (replace `en0` with your interface name and `XX:XX:XX:XX:XX:XX` with your desired MAC address):

```
sudo ifconfig en0 ether XX:XX:XX:XX:XX:XX
```

4. **Verify the Change:**

- Check your new MAC address:

```
ifconfig en0 | grep ether
```

Important Notes

- **Temporary Change:** The changes you make are temporary and will revert after a reboot. To make permanent changes, you would need to create scripts or use specific settings.
- **Network Policies:** Be cautious when changing your MAC address, as some networks have policies against this and may block access.
- **Unique MAC Address:** Ensure the new MAC address is unique on your local network to avoid conflicts.

By following these steps, you can successfully change the MAC address on your laptop!

Create Article for DNS Server Config:

[The Sisyphean Task Of DNS Client Config on Linux](#)

DNS (Domain Name Service): turn names into IP addresses

TimeLine/History/Development:

4.3 BSD: use /etc/resolv.conf to configure DNS servers

but DHCP program need to determine the DNS servers in the network, it may edit and change the /etc/resolv.conf file. When more than one program what to edit the /etc/resolv.conf file, it may cause conflict and cause problems.

resolvconf: a loose convention for managing DNS, common 2 versions are Deian and Openresolv.

To use a random MAC address on Debian 12, you can utilize a few different methods, including temporary changes or setting it to generate random addresses automatically. Here's how you can do it:

Method 1: Temporary Change Using `macchanger`

1. **Install `macchanger`:** Open a terminal and install `macchanger` using the following command:

```
sudo apt update
sudo apt install macchanger
```

2. **Identify Your Network Interface:** Find the name of your network interface (e.g., `wlan0`, `eth0`) by running:

```
ip link
```

3. **Bring Down the Interface:** Replace `wlan0` with your interface name:

```
sudo ip link set wlan0 down
```

4. **Change the MAC Address:** Use `macchanger` to set a random MAC address:

```
sudo macchanger -r wlan0
```

5. **Bring Up the Interface:**

```
sudo ip link set wlan0 up
```

6. **Verify the Change:**

```
ip link show wlan0
```

Method 2: Permanent Random MAC Address on Startup

To have a random MAC address every time you boot, you can modify the Network Manager configuration.

1. **Open NetworkManager Configuration:** Edit the connection file for your network interface. The path may vary, but you can typically find it in `/etc/NetworkManager/system-connections/`. You can list the available connections with:

```
sudo ls /etc/NetworkManager/system-connections/
```

Then open the specific connection file using a text editor (replace `YourConnection` with the actual connection name):

```
sudo nano /etc/NetworkManager/system-connections/YourConnection
```

- 2. Modify the Configuration:** Look for the `[802-11-wireless]` section and add or modify the following lines:

```
[connection]
...
ethernet.cloned-mac-address=random
```

or for Wi-Fi connections:

```
[802-11-wireless]
...
802-11-wireless.cloned-mac-address=random
```

- 3. Save and Exit:** Save the file and exit the editor.

- 4. Restart NetworkManager:** Restart the NetworkManager to apply the changes:

```
sudo systemctl restart NetworkManager
```

Method 3: Manual Random MAC Address Generation

If you want to generate a random MAC address manually, you can use a simple script or command.

- 1. Generate a Random MAC Address:** You can use the following command to generate a random MAC address:

```
printf '02:%x:%x:%x:%x:%x\n' $((RANDOM%256)) $((RANDOM%256)) $((RANDOM%256)) $((RANDOM%256))
```

- 2. Set the Random MAC Address:** Use `ip` to change the MAC address as shown in Method 1, substituting `xx:xx:xx:xx:xx:xx` with the output of the above command.

Notes

- **Temporary Changes:** The MAC address will revert to the original on reboot unless you use the NetworkManager method.
- **Network Policies:** Be aware that changing your MAC address may violate certain network policies.
- **Conflict Avoidance:** Ensure that the generated MAC address does not conflict with other devices on your network.

By following these methods, you can effectively use a random MAC address on your Debian 12 system!

In Debian 12's Wi-Fi identity management, here are the key terms and their meanings:

1. **SSID (Service Set Identifier)**: This is the name of the Wi-Fi network. It's what you see when you scan for available networks. Each SSID is unique within a given area.
2. **BSSID (Basic Service Set Identifier)**: This refers to the MAC address of the access point (AP) providing the network. While SSID is the name, BSSID identifies the actual hardware of the AP.
3. **MAC Address**: The Media Access Control (MAC) address is a unique identifier assigned to network interfaces for communications on a network. Each device on a network has a distinct MAC address.
4. **Cloned Address Options**:
 - **Preserve**: This option keeps the device's original MAC address.
 - **Permanent**: This sets a fixed MAC address for the device, which will remain the same across reboots and disconnections.
 - **Random**: This generates a new, random MAC address each time the device connects to the network, enhancing privacy by making tracking more difficult.
 - **Stable**: This generates a MAC address that remains consistent but is not the original. It's typically based on the device's original MAC but altered to provide some level of anonymity.

These options help manage network identity and privacy in various scenarios.

在 Debian 12 的 Wi-Fi 身份管理中，以下是关键术语及其含义：

1. **SSID（服务集标识符）**：这是 Wi-Fi 网络的名称。当你扫描可用网络时，会看到这个名称。每个 SSID 在特定区域内是唯一的。
2. **BSSID（基本服务集标识符）**：这指的是提供网络的接入点（AP）的 MAC 地址。虽然 SSID 是名称，但 BSSID 识别的是实际硬件的接入点。
3. **MAC 地址**：媒体访问控制（MAC）地址是分配给网络接口的唯一标识符，用于网络通信。网络上的每个设备都有一个独特的 MAC 地址。
4. **克隆地址选项**：
 - **保留**：此选项保持设备的原始 MAC 地址。
 - **永久**：这会为设备设置一个固定的 MAC 地址，该地址在重启和断开连接后将保持不变。
 - **随机**：这会在设备连接到网络时生成一个新的随机 MAC 地址，提高隐私性，使追踪更困难。
 - **稳定**：这会生成一个保持一致的 MAC 地址，但不是原始地址。通常是基于设备的原始 MAC 地址但进行了修改，以提供一定程度的匿名性。

这些选项有助于在不同场景中管理网络身份和隐私。

Attention Is Turing Complete

Not yet complete.

[Attention Is Turing Complete](#)

TLDR:

本文证明了 Attention 机制是图灵完备的，本文构造了一个有1层Encoder和3层Decoder的Transformer，用其模拟了图灵机的计算过程（即计算格局的变化）。

Abstract

Alternatives to recurrent neural networks, in particular, architectures based on self-attention, are gaining momentum for processing input sequences. In spite of their relevance, the computational properties of such networks have not yet been fully explored. We study the computational power of the Transformer, one of the most paradigmatic architectures exemplifying self-attention. We show that the Transformer with hard-attention is Turing complete exclusively based on their capacity to compute and access internal dense representations of the data. Our study also reveals some minimal sets of elements needed to obtain this completeness result.

循环神经网络的替代方案，特别是基于自注意力的架构，在处理输入序列方面正在获得动力。尽管它们具有相关性，但此类网络的计算特性尚未得到充分探索。我们研究Transformer的计算能力，它是体现自我注意力的最具范式的架构之一。我们证明，具有硬注意力的Transformer 完全基于其计算和访问数据内部密集表示的能力，是图灵完备的。我们的研究还揭示了获得这种完整性结果所需的一些最小元素集。

Introduction

序列到序列神经网络 我们对序列到序列 (seq-to-seq) 神经网络架构感兴趣,我们接下来将正式化。对于某些 $d > 0$, seq-to-seq 网络 N 接收向量 $x \in Q$ 的序列 $X = (x, \dots, x)$ 作为输入,并生成向量 $y \in Q$ 的序列 $Y = (y, \dots, y)$ 作为输出。大多数这种类型的架构都需要一个种子向量 s 和一些停止标准来确定输出的长度。后者通常基于特定输出向量的生成,称为序列结束标记。相反,在我们的形式化中,我们允许网络产生一个固定数量的 $r \geq 0$ 的输出向量。因此,为方便起见,我们将一般的 seq-to-seq 网络视为函数 N ,使得值 $N(X, s, r)$ 对应于 $Y = (y, y, \dots, y)$ 形式的输出序列。通过此定义,我们可以将 seq-to-seq 网络解释为字符串的语言识别器,如下所示。

定义 1 seq-to-seq 语言识别器是一个元组 $A = (\Sigma, f, N, s, F)$,其中 Σ 是有限字母表, $f: \Sigma \rightarrow Q$ 是嵌入函数, N 是 seq-to-seq 网络, $s \in Q$ 是种子向量, $F \subseteq Q$ 是一组最终向量。我们说 A 接受字符串 $w \in \Sigma$,如果存在一个整数 $r \in N$,使得 $N(f(w), s, r) = (y, \dots, y)$ 和 $y \in F$ 。

A 接受的语言(用 $L(A)$ 表示)是 A 接受的所有字符串的集合。

我们对识别器施加了两个额外的限制。

- 嵌入函数 $f: \Sigma \rightarrow Q$ 应该由图灵机以 Σ 大小的多项式时间计算。这涵盖了计算符号的 input 嵌入的两种最典型方法:one-hot 编码和由固定前馈网络计算的嵌入。
- 此外,集合 F 也应该在多项式时间内可识别;给定一个向量 f ,隶属度 $F \in F$ 应该由在多项式时间内相对于 f 的大小(以位为单位)工作的图灵机决定。这涵盖了使用固定序列结束向量检查相等性的常用方法。

我们施加这些限制是为了防止通过在 input embedding 或 stopping 条件中编码任意计算来作弊的可能性,同时足够宽容地构建有意义的 embeddings 和 stopping criterions。

图灵机计算 让我们回顾一下,(确定性的)图灵机是 $M = (Q, \Sigma, \delta, q, F)$ 形式的元组,其中:

seq-to-seq 神经网络架构的图灵完备性

seq-to-seq 神经网络架构的 N 类定义了类 L ,该类由使用 N 中的网络的语言识别器接受的所有语言组成。从这些概念中, N 类的图灵完备性的形式化自然而然地随之而来。

定义 2

如果 L 包含所有可判定语言(即图灵机可识别的所有语言),则 seq-to-seq 神经网络架构的 N 类为图灵完备。

1. The Transformer architecture

在本节中,我们提出了 Transformer 架构的形式化(Vaswani et al., 2017),从函数和参数的一些具体选择中抽象出来。我们的形式化并不是为了产生 Transformer 的有效实现,而是提供一个简单的设置,通过该设置可以以正式的方式建立其数学属性。

Transformer 在很大程度上基于接下来介绍的注意力机制。考虑评分函数 $\text{score} : Q \times Q \rightarrow Q$ 和归一化函数 $\rho : Q \rightarrow Q$,对于 $d, n > 0$ 。假设 $q \in Q$,并且 $K = (k_1, \dots, k_n)$ 和 $V = (v_1, \dots, v_n)$ 是 Q 中元素的元组。由 $\text{Att}(q, K, V)$ 表示的 q -attention over (K, V) 是 $Q \in$ 向量 a ,定义如下。

$$(s_1, \dots, s_n) = \rho(\text{score}(q, k_1), \text{score}(q, k_2), \dots, \text{score}(q, k_n))$$

$$a = s_1 v_1 + s_2 v_2 + \dots + s_n v_n$$

通常, q 称为查询, K 称为键, V 称为值。我们对评分函数没有任何限制,但我们确实对归一化函数施加了一些限制,以确保它在持仓上产生概率分布。我们要求归一化函数满足以下条件:对于每个 $x = (x_1, \dots, x_n) \in Q$,有一个函数 f 从 Q 到 Q ,使得 $\rho(x)$ 的第 i 个分量 $\rho(x)_i$ 相等

$j=1 f(x)_j = f(x)_i$ 。我们注意到,例如,可以通过简单地选择 $f(x)$ 作为指数函数 e 来以这种方式定义 softmax 函数,但允许其他可能性,接下来我们将解释。

在证明可能性结果时,我们需要选择特定的评分和归一化函数。评分函数的通常选择是由具有输入 (q, k) 的前馈网络定义的非线性函数,有时称为加性注意力(Bahdanau et al., 2014)。另一种可能性是使用点积 $\langle q, k \rangle$,称为乘法注意力(Vaswani et al., 2017)。

我们实际上使用了两者的组合:乘法注意力加上定义为 $\sigma(g(\cdot))$ 形式的函数组成的前馈网络,其中 g 是仿射变换, σ 是方程(1)中定义的分段线性 S 形激活。对于归一化函数,softmax 是一个标准选项。尽管如此,在我们的证明中,我们使用 hardmax 函数,如果 x 是 x 中的最大值,则通过设置 $f(x) = 1$ 来获得,否则设置 $f(x) = 0$ 。因此,对于最大值出现 r 次的向量 x ,如果 x 是 x 的最大值,则 $\text{hardmax}(x) = 1$,否则为 $\text{hardmax}(x) = 0$ 。每当 hardmax 用作归一化函数时,我们都称其为 hard attention。

让我们观察一下,hardmax 的选择对于我们的证明以当前形式工作至关重要,因为它允许模拟“访问”向量序列中特定位置的过程。Hard attention 以前专门用于处理图像(Xu et al., 2015;Elsayed et al., 2019),但是,据我们所知,它尚未在自我注意架构的上下文中用于处理序列。有关我们对正结果函数的选择的进一步讨论,请参见第 5 节。按照惯例,对于函数 $F : Q \rightarrow Q$ 和序列 $X = (x_1, x_2, \dots, x_n)$,其中 $x_i \in Q$,我们写 $F(X)$ 来表示序列 $(F(x_1), F(x_2), \dots, F(x_n))$ 。

Transformer 编码器和解码器

Transformer 的单层编码器是一个参数函数 $\text{Enc}(\theta)$,其中 θ 是参数,它接收 Q 中向量的序列 $X = (x_1, x_2, \dots, x_n)$ 作为输入,返回一个序列 $\text{Enc}(X; \theta) = (z_1, z_2, \dots, z_n)$ 中与 X 长度相同的向量。一般来说,我们认为 θ 中的参数是参数化函数 $Q(\cdot)$ 、 $K(\cdot)$ 、 $V(\cdot)$ 和 $O(\cdot)$,它们都从 Q 到 Q 。然后,单层编码器定义如下

$$a = \text{Att}(Q(x), K(X), V(X)) + x \quad (4) \quad z = O(a) + a \quad (5)$$

请注意,在等式 4 中,我们将函数 Q 和 V 分别应用于 X 中的每个条目。在实践中, $Q(\cdot)$ 、 $K(\cdot)$ 、 $V(\cdot)$ 通常是指定为维度 $(d \times d)$ 矩阵的线性变换,而 $O(\cdot)$ 是前馈网络。 $+ x$ and $+ a$ summands 通常称为残差连接(He et al., 2016;他等人)。当用作参数的特定函数不重要时,我们只需编写 $Z = \text{Enc}(X)$ 。

Transformer 编码器被简单地定义为单层编码器(具有独立参数)的重复应用,加上两个最终变换函数 $K(\cdot)$ 和 $V(\cdot)$ 应用于最终层输出序列中的每个向量。因此,L 层 Transformer 编码器由以下递归定义($1 \leq l \leq L-1$ 且 $X = X$):

$$X = \text{Enc}(X; \theta), K = K(X), V = V(X). \quad (6)$$

$V = V(X)$ 我们写 $(K, V) = T\text{Enc}(X)$ 来表示 (K, V) 是 L 层变压器编码器在输入序列 X 上的结果。

解码器

单层解码器类似于单层编码器,但需要额外注意一对外部键值向量 (K, V) 。单层解码器的输入是序列 $Y = (y, \dots, y)$ 加上外部对 (K, V) ,输出是与 Y 长度相同的序列 $Z = (z, \dots, z)$ 。在定义解码器层时,我们用 Y 表示序列 (y, \dots, y) ,为 $1 \leq i \leq k$ 。该层的输出 $Z = (z, \dots, z)$ 也被参数化,这次由四个函数 $Q(\cdot)$ 、 $K(\cdot)$ 、 $V(\cdot)$ 和 $O(\cdot)$ 从 Q to Q 组成,并且对于每个 $1 \leq i \leq k$ 定义如下:

$$p = \text{Att}(Q(y), K(Y), V(Y)) + y \quad (7) \quad a = \text{Att}(p, K, V) + p \quad (8) \quad z = O(a) + a \quad (9)$$

请注意,对 $(K(Y), V(Y))$ 的第一个(自我)关注仅在索引 i 之前考虑 Y 的子序列,并用于生成查询 p to 关注外部对 (K, V) 。我们将 Y 和 (K, V) 上的单解码器层的输出表示为 $\text{Dec}((K, V), Y; \theta)$ 。Transformer 解码器是单层解码器的重复应用,加上一个转换函数 $F : Q \rightarrow Q$ 应用于解码序列的最终向量。因此,解码器的输出是 $Q \in$ 单个向量 z 。正式地,L 层 Transformer 解码器定义为

$$Y = \text{Dec}((K, V), Y; \theta), z = F(y) \quad (1 \leq l \leq L-1 \text{ 和 } Y = Y). \quad (10)$$

我们使用 $z = T\text{Dec}((K, V), Y)$ 来表示 z 是这个 L 层变压器解码器在输入 Y 和 (K, V) 上的输出。

Transformer 的一个重要限制是 Transformer 解码器的输出总是对应于某些有限字母 Γ 中的字母编码。从形式上讲,需要存在一个有限字母 Γ 和一个嵌入函数 $g : \Gamma \rightarrow Q$,使得 Transformer 解码器的最终变换函数 F 将 Q 中的任何向量映射到 Γ 中字母嵌入的有限集 $g(\Gamma)$ 中的向量。

完整的 Transformer

Transformer 网络接收一个输入序列 X 、一个种子向量 y 和一个值 $r \in N$ 。它的输出是一个序列 $Y = (y, \dots, y)$,定义为

$$y = T\text{Dec}(T\text{Enc}(X), (y, y, \dots, y)), \text{ 对于 } 0 \leq t \leq r-1. \quad (11)$$

$$\mathbf{y}_{t+1} = T\text{Dec}(T\text{Enc}(X), (\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_t)), \quad \text{for } 0 \leq t \leq r-1.$$

我们将变压器的输出序列表示为 $Y = (y, y, \dots, y) = \text{Trans}(X, y, r)$ 。

3.1 比例下的不变性

如上所述,Transformer 网络在捕获语言的能力方面相当弱。这是因为 Transformer 是顺序不变的,即它们无法访问 input 中元素的相对顺序。更正式地说,两个相互排列的 Importing 序列会产生完全相同的输出。这是注意力函数的以下属性的结果:如果 $K = (k, \dots, k)$, $V = (v, \dots, v)$ 且 $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ 是排列,则对于每个查询 q , $\text{Att}(q, K, V) = \text{Att}(q, \pi(K), \pi(V))$ 。

$$\mathbf{a}_i = \text{Att}(Q(\mathbf{x}_i), K(X), V(X)) + \mathbf{x}_i$$

$$\mathbf{z}_i = O(\mathbf{a}_i) + \mathbf{a}_i$$

4. Transformer 的位置编码的图灵完备性

定理 6

具有位置编码的 Transformer 网络类是图灵完备的。此外,图灵完备性即使在受限设置中也成立,其中 $n \in \mathbb{N}$ 的位置嵌入 $\text{pos}(n)$ 中唯一的非恒定值是 n 、 $1/n$ 和 $1/n$,并且 Transformer 网络具有单个编码器层和三个解码器层。

实际上,这个结果的证明显示了更有力的东西:不仅 Transformers 可以识别图灵机接受的所有语言,即所谓的可识别或可判定语言;它们可以识别所有递归可枚举或半可判定的语言,这些语言 L 存在枚举 L 中所有字符串的 TM。

我们现在提供定理 6 的完整证明。为了可读性,一些中间引理的证明被归入附录。设 $M = (Q, \Sigma, \delta, q, F)$ 是一个图灵机,其磁带向右是无限的,并假设特殊符号 $\# \in \Sigma$ 用于标记磁带中的空白位置。

我们对 M 在处理 Importing 字符串时的工作原理做出以下假设:

M 从状态 q 开始,指向磁带的第一个读取空白符号 $\#$ 的单元格。输入将立即写入第一个单元格的右侧。

Q 具有特殊状态 q_{used} 来读取完整的输入。

最初 (步骤 0), M 过渡到状态 q ,并将其头部向右移动。

当处于状态时, q_{it} 向右移动,直到读取符号 $\#$ 。

不接受状态 (F 中的状态) 没有转换。

很容易证明,每个通用的图灵机都等同于满足上述假设的图灵机。我们证明了可以构建一个 transformer 网络 $Transhat$ 能够在每个可能的输入字符串上模拟 M ;或者,更正式地说, $L(M) = L(Trans)$ 。构造以某种方式参与其中,并使用了几个辅助定义和中间结果。为了便于阅读,我们将构造和证明分为三个部分。我们首先给出我们使用的策略的高级视图。然后,我们详细介绍了实现我们的策略所需的编码器和解码器的架构,最后我们正式证明我们架构的每个部分都可以实际实现。

在 $Transwe$ 的编码器部分接收字符串 $w = ss \dots s$ 。我们首先使用嵌入函数将每个 s 表示为 one-hot 向量,并为每个索引添加位置编码。编码器产生输出 (K, V) ,其中 $K = (k, \dots, k)$ 和 $V = (v, \dots, v)$ 是键和值的序列,使得 v 包含沙子的信息 k 包含第 i 个位置编码的信息。我们稍后会证明,这允许我们关注每个特定位置,并将每个输入符号从编码器复制到解码器 (参见引理 7)。

在 $Transwe$ 的解码器部分模拟 M 在 $w = ss \dots s$ 。为此,我们定义以下序列 (对于 $i \geq 0$) :

q :计算步骤 i 中 M 的状态 s :步骤 i 中 M 头部读取的符号 v :步骤 i 中 M 写入的符号 m :步骤 i 中 M 头部在过渡中移动的方向

$v = [q_1, s_1, x_1, q_2, s_2, x_2, x_3, x_4, x_5, s_3, x_6, s_4, x_7, x_8, x_9, x_{10}, x_{11}]$,

空间变化的纳米光子神经网络

Spatially varying nanophotonic neural networks

Abstract

The explosive growth in computation and energy cost of artificial intelligence has spurred interest in alternative computing modalities to conventional electronic processors. Photonic processors, which use photons instead of electrons, promise optical neural networks with ultralow latency and power consumption. However, existing optical neural networks, limited by their designs, have not achieved the recognition accuracy of modern electronic neural networks. In this work, we bridge this gap by embedding parallelized optical computation into flat camera optics that perform neural network computations during capture, before recording on the sensor. We leverage large kernels and propose a spatially varying convolutional network learned through a low-dimensional reparameterization. We instantiate this network inside the camera lens with a nanophotonic array with angle-dependent responses. Combined with a lightweight electronic back-end of about 2K parameters, our reconfigurable nanophotonic neural network achieves 72.76% accuracy on CIFAR-10, surpassing AlexNet (72.64%), and advancing optical neural networks into the deep learning era.

人工智能的计算和能源成本的爆炸式增长激发了人们对传统电子处理器的替代计算模式的兴趣。光子处理器使用光子而不是电子，有望实现具有超低延迟和低功耗的光学神经网络。然而，现有的光学神经网络受其设计的限制，尚未达到现代电子神经网络的识别精度。在这项工作中，我们通过将并行光学计算嵌入到平面相机光学元件中来弥合这一差距，这些光学元件在捕获期间执行神经网络计算，然后再在传感器上记录。我们利用大型内核，并提出了一个通过低维重新参数化学习的空间变化卷积网络。我们在相机镜头内使用具有角度相关响应的纳米光子阵列来实例化这个网络。结合约 2K 参数的轻量级电子后端，我们的可重构纳米光子神经网络在 CIFAR-10 上实现了 72.76% 的准确率，超过了 AlexNet (72.64%)，将光学神经网络带入了深度学习时代。

INTRODUCTION 介绍

Increasing demands for high-performance artificial intelligence (AI) in the last decade have levied immense pressure on computing architectures across domains, including robotics, transportation, personal devices, medical imaging and scientific imaging. Although electronic microprocessors have undergone drastic evolution over the past 50 years (1), providing us with general-purpose central processing units and custom accelerator platforms (e.g., graphical processing unit and Digital Signal Processor (DSP) ASICs), this growth rate is far outpaced by the explosive growth of AI models. Specifically, the Moore's law delivers a doubling in transistor counts every 2 years (2), whereas deep neural networks (DNNs) (3), arguably the most influential algorithms in AI, have doubled in size every 6 months (4). However, the end of voltage scaling has made the power consumption, and not the number of transistors, the principal factor limiting further improvements in computing performance (5). Overcoming this limitation and radically reducing compute latency and power consumption could drive unprecedented applications from low-power edge computation in the camera, potentially enabling computation in thin eyeglasses or microrobots and reducing power consumption in data centers used for training of neural network architectures. 在过去十年中，对高性能人工智能（AI）的需求不断增长，这给机器人、运输、个人设备、医学成像和科学成像等各个领域的计算架构带来了巨大压力。尽管电子微处理器在过去 50 年中经历了巨大的发展（1），为我们提供了通用的中央处理器和自定义加速器平台（例如图形处理单元和数字信号处理器（DSP）ASIC），但这一增长率远远超过了 AI 模型的爆炸式增长。具体来说，摩尔定律的晶体管数量每 2 年翻一番（2），而深度神经网络（DNN）（3）可以说是 AI 领域最有影响力的方法，其大小每 6 个月翻一番（4）。然而，电压缩放的结束使功耗而不是晶体管数量成为限制计算性能

进一步提高的主要因素 (5)。克服这一限制并从根本上降低计算延迟和功耗，可以推动相机中的低功耗边缘计算带来前所未有的应用，有可能在薄眼镜或微型机器人中实现计算，并降低用于神经网络架构训练的数据中心的功耗。

Optical computing has been proposed as a potential avenue to alleviate several inherent limitations of digital electronics, e.g., compute speed, heat dissipation, and power, and could potentially boost computational throughput, processing speed, and energy efficiency by orders of magnitude (6–10). Such optical computers leverage several advantages of photonics to achieve high throughput, low latency, and low power consumption (11). These performance improvements are achieved by sacrificing reconfigurability. Thus, although general-purpose optical computing has yet to be practically realized due to obstacles such as larger physical footprints and inefficient optical switches (12, 13), several notable advances have already been made toward optical/photonic processors tailored specifically for AI (14, 15). Representative examples include optical computers that perform widely used signal processing operators (16–22), e.g., spatial/temporal differentiation, integration, and convolution with performance far beyond those of contemporary electronic processors. Most notably, optical neural networks (ONNs) (6, 23–38) can perform AI inference tasks such as image recognition when implemented as fully optical or hybrid opto-electronical computers. 光学计算已被提议作为一种潜在的途径来缓解数字电子学的几个固有限制，例如计算速度、散热和功率，并可能将计算吞吐量、处理速度和能源效率提高几个数量级 (6–10)。这种光学计算机利用光子学的几个优势来实现高吞吐量、低延迟和低功耗 (11)。这些性能改进是通过牺牲可重新配置性来实现的。因此，尽管由于较大的物理占用空间和低效的光开关等障碍，通用光计算尚未实际实现 (12, 13)，但专门为 AI 量身定制的光/光子处理器已经取得了一些显著进展 (14, 15)。代表性示例包括执行广泛使用的信号处理运算符 (16–22) 的光学计算机，例如空间/时间微分、积分和卷积，其性能远远超过当代电子处理器。最值得注意的是，光学神经网络 (ONN) (6, 23–38) 在作为全光学或混合光电计算机实现时可以执行 AI 推理任务，例如图像识别。

Existing ONNs can be broadly classified into two categories based on either integrated photonics (24–30) [e.g., Mach-Zehnder interferometers (23, 26), phase change materials (24), microring resonators (29), multimode fibers (30)] for physically realizing multiply-adds floating point operations (FLOPs), or with free-space optics (6, 31–37) that implement convolutional layers with light propagation through diffractive elements [e.g., 3D-printed surfaces (6), 4F optical correlators (37), optical masks (35), and meta-surfaces (36)]. The design of these ONN architectures has been fundamentally restricted by the underlying network design, including the challenge of scaling to large numbers of neurons (within integrated photonic circuits) and the lack of scalable energy-efficient nonlinear optical operators. As a result, even the most successful ensemble ONNs (31) that use dozens of ONNs in parallel, have only achieved LeNet (39)–level accuracy on image classification, which was achieved by their electronic counterparts over 30 years ago. Moreover, most high-performance ONNs can only operate under coherent illumination, prohibiting the integration into the camera optics under natural lighting conditions. Although hybrid opto-electronic networks (35, 36, 40) working on incoherent light do exist, most of them do not yield favorable results as their optical front-end is designed for small-kernel spatially uniform convolutional layers, which this work finds does not fully exploit the design space available for optical convolution. 现有的 ONN 大致可分为两类，基于集成光子学 (24–30) [例如，马赫-曾德尔干涉仪 (23, 26)、相变材料 (24)、微环谐振器 (29)、多模光纤 (30)] 用于物理实现乘加浮点运算 (FLOP)，或具有自由空间光学器件 (6, 31–37)，它们实现了卷积层，光通过衍射元件传播 [例如，3D 打印表面 (6)、4F 光学相关器 (37)、光学掩模 (35) 和超表面 (36)]。这些 ONN 架构的设计从根本上受到底层网络设计的限制，包括扩展到大量神经元 (在集成光子电路内) 的挑战以及缺乏可扩展的节能非线性光学运算符。因此，即使是最成功的集成 ONN (31) 并行使用数十个 ONN，也只能达到 LeNet (39) 级的图像分类精度，而这是 30 多年前电子同类产品实现的。此外，大多数高性能 ONN 只能在相干照明下工作，因此无法在自然光条件下集成到相机光学元件中。尽管确实存在处理非相干光的混合光电网络 (35, 36, 40)，但它们中的大多数并没有产生有利的结果，因为它们的光学前端是为小核空间均匀的卷积层设计的，这项工作发现这并没有充分利用可用于光学卷积的设计空间。

In this work, we report a novel nanophotonic neural network that lifts the aforementioned limitations, allowing us to close the gap to the first modern DNN architectures (41) with optical compute in a flat form factor of only 4 mm length, akin to performing computation on the sensor cover glass, in lieu of the bulky compound 4-f system-based Fourier filter setup (40). We leverage the ability of a lens system to perform large-kernel spatially varying (LKS) convolutions tailored specifically for image recognition and semantic segmentation. These operations are performed during the capture before the sensor makes a measurement. We learn large kernels via low-dimensional reparameterization techniques, which circumvent spurious local extremum caused by direct optimization. To physically realize the ONN, we develop a differentiable spatially varying inverse design framework that solves for metasurfaces (42–46) that can produce the desired angle-dependent responses under spatially incoherent illumination. Because of the compact footprint and complementary metal-oxide semiconductor (CMOS) sensor compatibility, the resulting optical system is not only a photonic accelerator but also an ultracompact computational camera that directly operates on the ambient light from the environment before the analog to digital conversion. We find that this approach facilitates generalization and transfer learning to other tasks, such as semantic segmentation, reaching performance comparable to AlexNet (41) in 1000-category ImageNet (47) classification and PASCAL VOC (48) semantic segmentation. 在这项工作中，我们报道了一种新颖的纳米光子神经网络，它消除了上述限制，使我们能够缩小与第一个现代 DNN 架构 (41) 的差距，光学计算采用仅 4 mm 长度的扁平外形尺寸，类似于在传感器盖玻璃上执行计算，而不是笨重的基于化合物 4-f 系统的傅里叶滤波器设置 (40)。我们利用镜头系统的能力来执行专为图像识别和语义分割量身定制的大内核空间变化 (LKS) 卷积。这些操作是在传感器进行测量之前在捕获期间执行的。我们通过低维重新参数化技术学习大内核，该技术规避了直接优化引起的伪局部极值。为了物理实现 ONN，我们开发了一个可微分空间变化逆向设计框架，该框架求解超表面 (42–46)，该超表面可以在空间非相干照明下产生所需的角度相关响应。由于紧凑的封装和互补的金属氧化物半导体 (CMOS) 传感器兼容性，最终的光学系统不仅是一个光子加速器，而且还是一个超紧凑的计算相机，在模数转换之前直接对来自环境的环境光进行操作。我们发现这种方法有助于泛化和将学习转移到其他任务，如语义分割，在 1000 个类别的 ImageNet (47) 分类和 PASCAL VOC (48) 语义分割中达到与 AlexNet (41) 相当的性能。

Recent work (49) concurrent to ours reported a novel metasurface doublet that implements a multichannel optical convolution via angular and polarization multiplexing under spatially incoherent illuminance, and extensions (50, 51) leverage large convolutional kernels for image classification and semantic segmentation. While this work shares advantages with ours, such as multichannel operation, high performance, and the use of incoherent light, our method uses a single metasurface and relies on LKS convolution instead of uniform convolutions increasing the parameter space by an order of magnitude. 与我们同时进行的最新工作 (49) 报道了一种新的超表面双合体，它在空间非相干照度下通过角度和偏振多路复用实现多通道光学卷积，而扩展 (50, 51) 利用大型卷积核进行图像分类和语义分割。虽然这项工作与我们的工作具有共同的优势，例如多通道操作、高性能和非相干光的使用，但我们的方法使用单个超表面，并依赖于 LKS 卷积而不是均匀卷积，将参数空间增加了一个数量级。

Hence, by on-chip integration of the flat-optics front-end (>99% FLOPs) with an extremely lightweight electronic back-end (<1% flops), we achieve higher classification performance than modern fully electronic classifiers [73.80% in simulation and 72.76% experiment, compared to 72.64% by alexnet ([*41*](https://www.science.org/doi/10.1126/sciadv.adp0391#core-r41)) on cifar-10 ([*52*](https://sciadv.adp0391#core-r52)) test set], while simultaneously reducing the number of parameters four orders of magnitude, thus bringing onns into deep learning era. 因此，通过将平面光学前端 (>99% FLOPs) 与极轻的电子后端 (<1% FLOPs) 进行片上集成，我们实现了比现代全电子分类器更高的分类性能 [模拟中为 73.80%，实验中为 72.76%，而 CIFAR-10 上的 AlexNet (41) 为 72.64% ([*52*] 测试集)]，同时将电子参数的数量减少了四个数量级，从而将 ONN 带入了现代深度学习时代。

RESULTS 结果

LKSV parameterization LKSV 参数化

The working principle and optoelectronic implementation of the proposed spatially varying nanophotonic neural network (SVN3) are illustrated in Fig. 1A. The SVN3 is an optoelectronic neuromorphic computer that comprises a metalens array nanophotonic front-end and a lightweight electronic back-end (embedded in a low-cost microcontroller unit) for image classification or semantic segmentation. The metalens array front-end consists of 50 metalens elements that are made of 390-nm pitch nano-antennas and are optimized for incoherent light in a band around 525 nm. The wavefront modulation induced by each metalens can be represented by the optical convolution of the incident field and the point spread functions (PSFs) of the individual device. Therefore, the nanophotonic front-end performs parallel multichannel convolutions, at the speed of light, without any power consumption. We also refer to texts S1 and S3 for additional details on the physical forward model and the neural network design, respectively. 所提出的空间变化纳米光子神经网络（SVN3）的工作原理和光电实现如图 1A 所示。SVN3 是一种光电神经形态计算机，包括一个超透镜阵列纳米光子前端和一个用于图像分类或语义分割的轻量级电子后端（嵌入在低成本微控制器单元中）。超透镜阵列前端由 50 个超透镜元件组成，这些元件由 390 nm 间距纳米天线制成，并针对 525 nm 左右波段的非相干光进行了优化。每个超透镜诱导的波前调制可以用入射场的光学卷积和单个器件的点扩散函数（PSF）来表示。因此，纳米光子前端以光速执行并行多通道卷积，无需任何功耗。我们还参考了文本 S1 和

CRTPO: Compile-Time-Reflection-Programming

[wiki](#)

```
// The Curiously Recurring Template Pattern (CRTPO)
template <class T>
class Base {
    // methods within Base can use template to access members of Derived
};

class Derived : public Base<Derived> {
    // ...
};
```

Why?

Polymorphic chaining

```

// Base class
template <typename ConcretePrinter>
class Printer {
public:
    Printer(ostream& pstream) : m_stream(pstream) {}

    template <typename T>
    ConcretePrinter& print(T&& t) {
        m_stream << t;
        return static_cast<ConcretePrinter&>(*this);
    }

    template <typename T>
    ConcretePrinter& println(T&& t) {
        m_stream << t << endl;
        return static_cast<ConcretePrinter&>(*this);
    }

private:
    ostream& m_stream;
};

// Derived class
class CoutPrinter : public Printer<CoutPrinter> {
public:
    CoutPrinter() : Printer(cout) {}

    CoutPrinter& SetConsoleColor(Color c) {
        // ...
        return *this;
    }
};

// usage
CoutPrinter().print("Hello ").SetConsoleColor(Color.red).println("Printer!");

```

Polymorphic copy construction 多态复制构造

When using polymorphism, one sometimes needs to create copies of objects by the base class pointer. A commonly used idiom for this is adding a virtual clone function that is defined in every derived class. The CRTP can be used to avoid having to duplicate that function or other similar functions in every derived class. 使用多态性时，有时需要通过基类指针创建对象的副本。为此，常用的习惯用法是添加在每个派生类中定义的虚拟克隆函数。CRTP 可用于避免在每个派生类中重复该函数或其他类似函数。

```
// Base class has a pure virtual function for cloning
class AbstractShape {
public:
    virtual ~AbstractShape() = default;
    virtual std::unique_ptr<AbstractShape> clone() const = 0;
};

// This CRTP class implements clone() for Derived
template <typename Derived>
class Shape : public AbstractShape {
public:
    std::unique_ptr<AbstractShape> clone() const override {
        return std::make_unique<Derived>(static_cast<Derived const&>(*this));
    }

protected:
    // We make clear Shape class needs to be inherited
    Shape() = default;
    Shape(const Shape&) = default;
    Shape(Shape&&) = default;
};

// Every derived class inherits from CRTP class instead of abstract class

class Square : public Shape<Square> {};

class Circle : public Shape<Circle> {};
```

This allows obtaining copies of squares, circles or any other shapes by `shapePtr->clone()`. 这允许通过`shapePtr->clone()`获取正方形、圆形或任何其他形状的副本。

ArgParse in Python

unexpected `parser.add_argument("--train", type=bool)`

在使用 `argparse` 时，`parser.add_argument("--train", type=bool)` 的用法可能会导致一些意外行为，因为 `type=bool` 并不会将输入的字符串自动转换为布尔值。相反，它会尝试将输入的字符串作为 Python 的 `bool()` 函数的参数，这通常会导致输入的值被解释为 `True` 或 `False`，但结果可能不符合预期。

问题分析

- `bool("True")` 返回 `True`，但 `bool("False")` 也返回 `True`，因为非空字符串在 Python 中被视为 `True`。
- 因此，直接使用 `type=bool` 无法正确解析布尔值参数。

正确的用法

为了正确解析布尔值参数，可以使用 `action='store_true'` 或 `action='store_false'`，或者自定义类型转换函数。

方法 1：使用 `action='store_true'` 或 `action='store_false'`

这是最常用的方法，适用于布尔值参数的默认行为。

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("--train", action='store_true', help="是否启用训练模式")

args = parser.parse_args()

print(f"训练模式: {args.train}")
```

使用示例

```
python script.py --train
```

输出:

```
训练模式: True
```

如果不传递 `--train` 参数:

```
python script.py
```

输出:

```
训练模式: False
```

方法 2：自定义类型转换函数

如果你希望用户明确传递 `True` 或 `False`，可以使用自定义类型转换函数。

```
import argparse

def str_to_bool(value):
    if value.lower() in ['true', '1', 't', 'y', 'yes']:
        return True
    elif value.lower() in ['false', '0', 'f', 'n', 'no']:
        return False
    else:
        raise argparse.ArgumentTypeError("布尔值必须是 'True' 或 'False'")

parser = argparse.ArgumentParser()
parser.add_argument("--train", type=str_to_bool, help="是否启用训练模式")

args = parser.parse_args()

print(f"训练模式: {args.train}")
```

使用示例

```
python script.py --train True
```

输出:

```
训练模式: True
```

```
python script.py --train False
```

输出:

```
训练模式: False
```

如果传递了无效的值:

```
python script.py --train maybe
```

输出:

```
usage: script.py [-h] [--train TRAIN]
script.py: error: argument --train: 布尔值必须是 'True' 或 'False'
```

总结

- 如果希望参数默认是布尔值，推荐使用 `action='store_true'` 或 `action='store_false'`。
- 如果需要用户明确传递 `True` 或 `False`，可以使用自定义类型转换函数。
- 直接使用 `type=bool` 可能会导致意外行为，不推荐使用。

AsyncSSH

- [doc](#)

Dask

Easy Parallel Python that does what you need

Dlib

Dlib C++ Library

Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high performance computing environments. Dlib's open source licensing allows you to use it in any application, free of charge.

Dlib 是一个现代 C++ 工具包，包含机器学习算法和工具，用于用 C++ 创建复杂的软件来解决现实世界的问题。它在工业界和学术界广泛使用，包括机器人、嵌入式设备、移动电话和大型高性能计算环境。Dlib 的开源许可允许您在任何应用程序中免费使用它。

```
pkg-config --cflags --libs dlib-1
-I/usr/local/include -L/usr/local/lib -ldlib /usr/lib/x86_64-linux-gnu/libsqllite3.so
```

local build and install:

apt install:

```
get_frontal_face_detector()
```

This function returns an `object_detector` that is configured to find human faces that are looking more or less towards the camera. It is created using the `scan_fhog_pyramid` object.

python bindings

```
class dlib.image_window
```

This is a GUI window capable of showing images on the screen.

```
add_overlay(rectangles, color=rgb_pixel(255,0,0)) -> None
add_overlay(rectangle, color=rgb_pixel(255,0,0)) -> None
add_overlay(full_object_detection, color=rgb_pixel(255,0,0)) -> None
clear_overlay()
get_next_double_click(self: dlib.image_window) -> object
get_next_keypress()
is_closed() -> bool
set_image(img: numpy.ndarray[(rows, cols), int]) -> None
set_title(title: str) -> None
wait_until_closed() -> None
wait_for_keypress(key: str) -> int
```

Blocks until the user presses the given key or closes the window.

```
class dlib.face_recognition_model_v1
```

This object maps human faces into 128D vectors where pictures of the same person are mapped near to each other and pictures of different people are mapped far apart. The constructor loads the face recognition model from a file.

```
defcompute_face_descriptor(
    img: numpy.ndarray[(rows, cols, 3), uint8],
    face: full_object_detection,
    num_jitters: int=0,
    padding: float=0.25),
    -> dlib.vector
```

Takes an image and a `full_object_detection` that references a face in that image and converts it into a `128D face descriptor`. If `num_jitters>1` then each face will be randomly jittered slightly `num_jitters` times, each run through the 128D projection, and the average used as the face descriptor. Optionally allows to override default padding of 0.25 around the face.

`dlib.vector`

This object is an array of vector objects.

shape_predictor_68_face_landmarks

[facial-point-annotations](#)

```
shape_predictor
```

One Millisecond Face Alignment with an Ensemble of Regression Trees, CVPR 2014

这篇论文解决了单张图像的人脸对齐问题。我们展示了如何使用回归树集成直接从像素强度的稀疏子集估计人脸的关键点位置，从而实现超实时的高质量预测。我们提出了一种基于梯度提升的通用框架，用于学习回归树集成，优化平方误差损失和自然处理缺失或部分标注的数据。我们展示了如何利用适当的先验信息，利用图像数据的结构来帮助高效的特征选择。我们还研究了不同的正则化策略及其在防止过拟合中的重要性。此外，我们分析了训练数据量对预测精度的影响，并探讨了使用合成数据进行数据增强的效果。

PyQt5

[doc](#)

- **QtCore** – 其他模块使用的非 GUI 类
- **QtGui** – 图形用户界面组件
- **QtMultimedia** – 低级多媒体编程类
- **QtNetwork** – 网络编程类
- **QtOpenGL** – OpenGL 支持类
- **QtScript** – 评估 Qt 脚本的类
- **QtSql** – 使用 SQL 进行数据库集成的类
- **QtSvg** – 显示 SVG 文件内容的类
- **QtWebKit** – 渲染和编辑 HTML 的类
- **QtXml** – 处理 XML 的类
- **QtWidgets** – 创建经典桌面风格 UI 的类
- **QtDesigner** – 扩展 Qt Designer 的类

QWidget 派生自 **QObject** 和 **QPaintDevice** 类的类是所有用户界面对象的基类。**QDialog**和**QFrame**类也派生自 **QWidget** 类。他们有自己的子类系统。

这是常用小部件的选择列表

序号	小部件和说明
1	QLabel 用于显示文本或图像
2	QLineEdit 允许用户输入一行文本
3	QTextEdit 允许用户输入多行文本
4	QPushButton 调用操作的命令按钮
5	QRadioButton 允许从多个选项中选择一个
6	QCheckBox 允许选择多个选项
7	QSpinBox 允许增加/减少整数值
8	QScrollBar 允许访问超出显示孔径的小部件内容
9	QSlider 允许线性改变边界值。
10	QComboBox 提供可供选择的项目的下拉列表
11	QMenuBar 包含 QMenu 对象的单杠
12	QStatusBar 通常在 QMainWindow 的底部，提供状态信息。
13	QToolBar 通常在 QMainWindow 顶部或浮动。包含操作按钮
14	QListView 在 ListMode 或 IconMode 中提供可选择的项目列表
15	QPixmap 显示在 QLabel 或 QPushButton 对象上的屏幕外图像表示
16	QDialog 可以向父窗口返回信息的模态或非模态窗口

一个典型的基于 GUI 的应用程序的顶层窗口是由**QMainWindow**小部件对象。上面列出的一些小部件在此主窗口中占据指定位置，而其他小部件则使用各种布局管理器放置在中央小部件区域。



QDialog 小部件呈现一个顶级窗口，主要用于收集用户的响应。它可以配置为 Modal（它会阻止其父窗口）或 Modeless（可以绕过对话窗口）。

PyQt API 有许多预配置的 Dialog 小部件，例如 `InputDialog`、`FileDialog`、`FontDialog` 等。

`QMessageBox` 是一种常用的模式对话框，用于显示一些信息性消息，并可选择要求用户通过单击其上的任何一个标准按钮来做出响应。每个标准按钮都有一个预定义的标题、一个角色并返回一个预定义的十六进制数字。

多文档界面

一个典型的 GUI 应用程序可能有多个窗口。选项卡式和堆叠式小部件允许一次激活一个这样的窗口。然而，很多时候这种方法可能没有用，因为其他窗口的视图被隐藏了。

同时显示多个窗口的一种方法是将它们创建为独立的窗口。这称为 SDI（单文档接口）。这需要更多的内存资源，因为每个窗口可能有自己的菜单系统、工具栏等。

MDI（多文档接口）应用程序消耗较少的内存资源。子窗口相对于彼此放置在主容器内。容器小部件称为 `QMdiArea`。

`QMdiArea` 小部件一般占据 `QMainWondow` 对象的中心小部件。该区域中的子窗口是 `QMdiSubWindow` 类的实例。可以将任何 `QWidget` 设置为 `subWindow` 对象的内部小部件。MDI 区域的子窗口可以层叠或平铺排列。

PyQt5 - 拖放

拖放的提供对用户来说非常直观。它存在于许多桌面应用程序中，用户可以在其中将对象从一个窗口复制或移动到另一个窗口。

基于 MIME 的拖放数据传输基于 `QDrag` 类。`QMimeData` 对象将数据与其对应的 MIME 类型相关联。它存储在剪贴板上，然后在拖放过程中使用。

下面的 `QMimeData` 类函数可以方便地检测和使用 MIME 类型。

数据库处理

PyQt5 库包含 `QtSql` 模块。它是一个复杂的类系统，可以与许多基于 SQL 的数据库进行通信。它的 `QSqlDatabase` 通过 `Connection` 对象提供访问。以下是当前可用的 SQL 驱动程序列表

QGraphicsScene

- **Create a QGraphicsScene** : This serves as the container for the graphical elements.
 - 创建 `QGraphicsScene`: 用作图形元素的容器。
- **Load the Image with QPixmap** : Use `QPixmap` to load the image.
 - Load the Image with QPixmap: 用于 `QPixmap` 加载图像。
- **Add the Image to the Scene** : Create a `QGraphicsPixmapItem` from the `QPixmap` and add it to the scene.
 - 将图像添加到场景: `QGraphicsPixmapItem` 从创建一个 `QPixmap` 并将其添加到场景中。
- **Display the Scene in a QGraphicsView** : Use `QGraphicsView` to display the contents of the scene.
 - 在 `QGraphicsView` 中显示场景: 用于 `QGraphicsView` 显示场景的内容。

A `QRadioButton` class object presents a selectable button with a text label. It is commonly used when the user need to choose one option from a list of options. This widget is represented by a small circular button that can be toggled on (checked) or off (unchecked). This class is derived from `QAbstractButton` class.

QRadioButton 类对象表示带有文本标签的可选按钮。当用户需要从选项列表中选择一个选项时，通常使用它。此小部件由一个小的圆形按钮表示，该按钮可以打开（选中）或关闭（取消选中）。此类派生自 QAbstractButton 类。

Qt Style Sheets

- [stylesheet-examples](#)
- [Customizing Qt Widgets Using Style Sheets](#)
- [Qt Style Sheets Reference](#)
- [blog: 使用 QSS 美化 PyQt5 界面](#)
- [qt-material](#)

Qt-Material:

```
pip install qt-material

import sys
from PySide6 import QtWidgets
# from PySide2 import QtWidgets
# from PyQt5 import QtWidgets
from qt_material import apply_stylesheet

# create the application and the main window
app = QtWidgets.QApplication(sys.argv)
window = QtWidgets.QMainWindow()

# setup stylesheet
apply_stylesheet(app, theme='dark_teal.xml')

# run
window.show()
app.exec_()
```

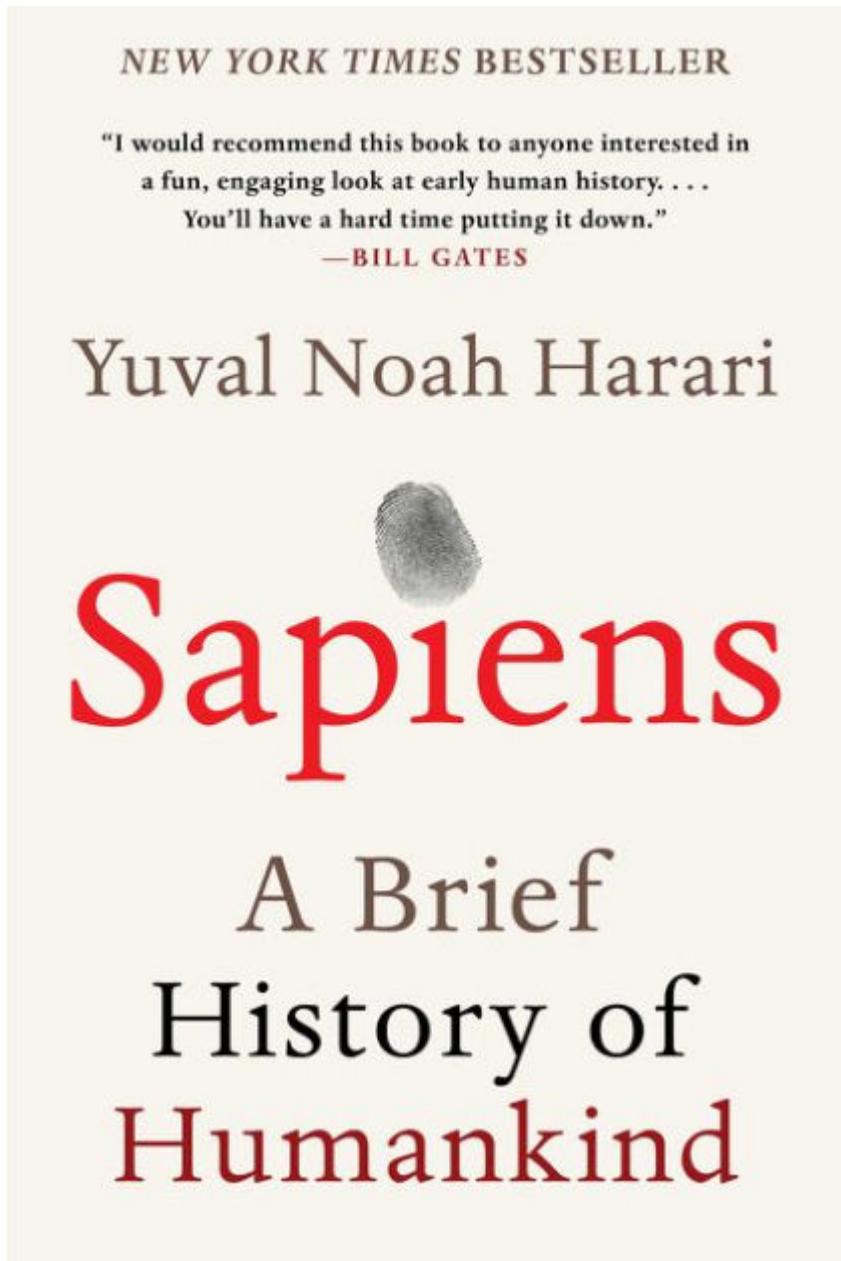
Reading Notes

- 论青年的修养-张闻天-1938
 - 论青年修养-张申府-1946
 - 研究与学风-张申府-1948
 - 青年在选择职业时的考虑-马克思
 - 语录
 - 计算之魂
-

References

- jyywiki.cn/Letter.md
- 《清华名师谈学风》-清华大学校史馆

A Brief History of Humankind



内容简介

《人类简史：从动物到上帝》是尤瓦尔·赫拉利创作的历史类著作，讲述了人类从石器时代至21世纪的演化与发展史，并将人类历史分为四个阶段：认知革命、农业革命、人类的融合统一与科学革命。

作者

尤瓦尔·赫拉利，1976年生，牛津大学历史学博士，现为耶路撒冷希伯来大学的历史系教授，青年怪才，全球瞩目的新锐历史学家。他擅长世界历史和宏观历史进程研究。在学术领域和大众出版领域都有很大的兴趣。

内容摘录

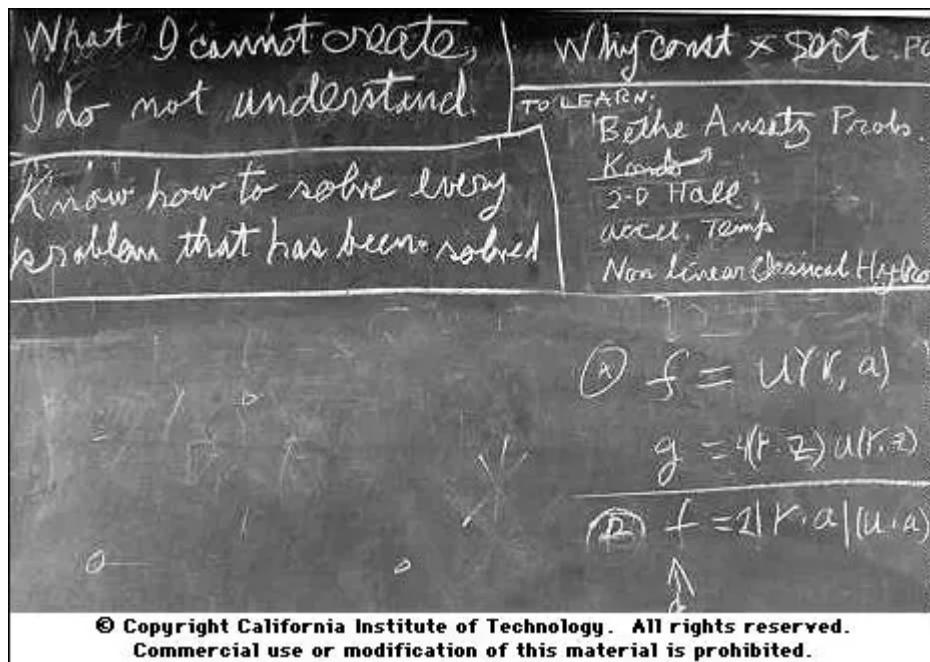
- 无论是现代国家、中世纪的教堂、古老的城市，或者古老的部落，任何大规模人类合作的根基，都在于某种只存在于集体想象中的虚构故事。例如教会的根基就在于宗教故事。
- 正因如此，人类才一起想出了“有限公司”这种概念。在法律上，这种公司是独立的个体，而不等于设立者、投资者或管理者。在过去几世纪间，这种公司已经成为经济主流，我们太习惯于这种概念，而忘了这只存在于我们的想象之中。“有限公司”的英文称为“corporation”，这点颇为讽刺，因为这个字的语源是“corpus”（拉丁文的“身体”），而这正是有限公司所没有的。虽然公司并没有真正的实体，但在法律上我们却将它称为“法人”，好像它真的是有血有肉的人一般。
- 如果要衡量某种物种演化成功与否，评判标准就在于世界上其 DNA 螺旋的拷贝数的多寡。
- 因此，我们来看看美国《独立宣言》改用生物学、科学的角度来写该是如何：我们认为下面这些真理是不言而喻的：人人演化各有不同，出生就有某些可变的特性，其中包括生命和追求快感。

注：有一个笑话说，独立宣言所说的不言自明的真理“All men are created equal.”这句话中，只有“are”是含义明确且真正为真的。All 是否包括奴隶，Men 是否包括女人，created 是神创还是演化，equal 是公平还是平等？

读书心得

其实在看《人类简史》之前，我也看过很多关于人类或者自然历史的书，但《人类简史》还是让我对人类社会的历史思考与观察的角度产生了很大的变化.....

Feynman



When Feynman said "create", he did not literally mean that in order to understand particle physics, he had to go Tony Stark on us and build his own accelerator. Instead, he meant that, starting with a blank piece of paper and the knowledge already in his mind, he could take any theoretical result and re-derive it. ("Any" is probably an exaggeration, but he could likely derive whatever he was interested in.)

Feynman thought that ability was the true marker of understanding something because the only way to be able to work something out yourself is to have a firm understanding of each step of the reasoning involved. Further, if you try this, even with relatively simple concepts you think you understand well already, you'll find that you frequently come away from the process with a much deeper appreciation for the problem.

An even more extreme position from Feynman was[1]

Once, I [David Goodstein] said to him, "Dick, explain to me, so that I can understand it, why spin one-half particles obey Fermi-Dirac statistics." Sizing up his audience perfectly, Feynman said, "I'll prepare a freshman lecture on it." But he came back a few days later to say, "I couldn't do it. I couldn't reduce it to the freshman level. That means we don't really understand it."

Feynman meant here that understanding something is not just about working through advanced mathematics. One must also have a notion that is intuitive enough to explain to an audience that cannot follow the detailed derivation.

I've seen a few more sources that spell out Feynman's position on this in detail.

The spinning plate story describes how Feynman felt that curiosity about simple things, and working them out for himself, helped him retain an attitude of play towards his professional work that got him out of a slump.

Feynman's Tips on Physics, an extension to the Feynman lectures, has a chapter about how to learn physics, emphasizing that memorizing formulas is hopeless in the long run, and that by knowing a few key things and understanding the principles, you can work out whatever details you need. I can't find this chapter online, though.

Finally, Feynman's Lost Lecture is a fantastic example of precisely what Feynman meant. In it, he describes his own elementary proof that the inverse square law for gravity leads to elliptical orbits.

当费曼说“创造”时，他并不是说：要理解粒子物理，就必须像托尼·斯塔克那样亲自造一个加速器。而是他认为，从一张白纸和头脑中已有的知识出发，他可以重新推导出任何理论结果。（“任何”可能有些夸张，但很可能会推导出他感兴趣的东西。）

费曼认为，这种能力是真正理解某件事的标志，因为唯一能让你自己解出问题的方法是，对推理过程中的每一步都有一个坚定的理解。此外，如果你尝试这样做，即使是对你已经认为理解得很简单的概念，你也会发现，你经常从过程中获得更深的认识。

费曼甚至更极端的观点是[1]：

有一次，我对他说，“迪克，解释给我听，以便我能理解，为什么半自旋粒子遵守费米-狄拉克统计规律？”他完美地评估了他的听众，然后说：“我准备一个给新生的讲座。”但是几天后他回来说，“我做不到。我无法降到新生水平。这意味着我们实际上并不了解。”

在这里，费曼的意思是，理解某件事不仅仅是通过高级数学来解决问题。还必须有一个足够直观的概念，以便向一个无法跟随详细推导的听众解释。

我找到了一些详细阐述费曼在这个问题上的观点的资料。

旋转盘的故事描述了费曼如何认为，对简单事物的好奇心以及他自己解决问题的能力，帮助他保持一种对职业工作的玩耍态度，使他从困境中走出来。

《费曼物理学指南》是一本扩展费曼讲座的书籍，其中一章讲述了如何学习物理学，强调长期来看记忆公式是徒劳的，而通过了解一些关键点以及理解原则，你可以解决所需的任何细节。不过，我在网上找不到这一章。

最后，《费曼的迷失讲座》正是费曼观点的一个绝佳例证。在其中，他描述了他自己用初等方法证明万有引力反比定律导致椭圆轨道的过程。

数学具有无可怀疑的真理性吗？

数学？数学的真理性、正确性源于什么？

什么是数学？

实际中期以前：数学哲学作为一般哲学的一个侧面

现代数学哲学

什么是数学的可靠性基础？

是数学外部的某种成分、还是数学内部的某种基本理论？

怎样把数学建立在一个可靠的基础之上？

数学基础的研究 foundations of mathematics

逻辑主义者：数学的基础在于逻辑，莱布尼茨

直觉主义者：数学的基础在于人类所固有的直觉能力，康德

希尔伯特：数学不需要任何附加的前提，数学本身就包含了绝对可靠的成分-有限的数学？？

数学基础研究的主要任务在于：

通过给非有限的数学以形式的解释，把全部数学奠基于有限的数学

希尔伯特的形式主义

数学的瑕疵-悖论

数学中的悖论-数学不是完全可靠的？

数学研究对象的实在性问题-数学本体论问题

实在论：数学的研究对象是一种独立的、不依赖人类思维的客观存在

形式主义：数学的研究对象知识一种无意义的符号，对此没有、也不必要作出解释

符号和规则中，何以蕴含如此庞杂、丰富的可能？

集中在无限性对象的实在性问题上：

- 实无限论
- 潜无限论
- 方法论的实无限论

数学的研究对象是否是一种真实的存在？

“圆”，“正方形”，“正弦函数”是真实的存在吗？

什么叫真实的存在？什么是存在？

这些概念存在于我们的“脑海”中？我们观察到的“圆形”的东西，如一元硬币，太阳，圆形井盖，和概念中的“圆形”是什么关系？

数学的真理性问题

19世纪20年代前，数学真理普遍地被认为是一种绝对真理

由于非欧几何的建立，关于数学绝对真理性的传统观念崩溃了

既然两种互不相容的几何理论都应被看成合理的数学理论，那数学命题当然就不具有绝对真理性？？

术与道

老子曰：有道无术，术尚可求也。有术无道，止于术。

庄子曰：以道驭术，术必成。离道之术，术必衰。

孙子曰：道为术之灵，术为道之体。以道统术，以术得道。

古人曰：上人用道，中人用术，下人用力。

韩愈曰：闻道有先后，术业有专攻。

研究与学风-张申府

在学问上，今日一定要加强研究，必非深造自得不可，必须养成大器，再不可浅尝辄止，皮毛肤浅。

有人说，我不要做学者专家，我为何要研究？但你只要是一个人，为社会做一点事，你总要有一些东西，总要有一个专长。因此，我们在学校里的时候就要深厚；联带着来的就要脚踏实地，实是求是，就要认真，就要实实在在；再一点就要精核，要一丝不苟，绝不等于漠忽笼统。

现在所谓研究在英文就是Research，法文，意大利文，西班牙文，以至俄文都也差不多。这也可翻译成考索，普通都以为研究很高深很难，其实，也很平常，肯作就不难。先看字面：从Research，这个字看，不过是找了又找。你肯去找，找了又找，这是不难的，只要你肯去找就行。各种学问全是找来的，各种真理全是找来的。研究古书，研究新学，全要肯找，肯找才能得。所谓“求则得之”这不是空话。譬如有一算学难题，别人解答不出，我肯去找，就有找出来的希望。十九世纪初，法国有一青年名叫葛录亚

(Galois)，未成年就死了。他对算学有兴趣，肯去找，二三年间便成为世界上第一流的算学家了。现在世界最伟大的学者是爱因斯坦，作成相对论，他的成功，也是找来的。我的另一个名字可叫调查。我国的考古学家去找，不是找出北京人来吗？研究不过如此，这并不难，不要怕，任何人都能做。再换个说法，中国人普通不说研究，他说我捉摸捉摸，因此，研究也就可以说是捉摸。捉摸是要动手动脚的，所以研究不但要用脑，要也用手，用脚。中国字同音的常同义，而且一个字常有反面的意思。有两个和捉摸音相近义相似的，就是琢磨，这就是说研究要一点一点来。如研究政治，研究经济，你都要一点一点去捉摸或琢磨，换句话说，就是要钻。前边说专家，专家就是肯钻。肯去钻，肯去捉摸，肯去琢磨，这就是研究。刚才我说中国字常有同音同义，而且一个字常有相反的意思，钻和专也就如此。另如忍字。忍是忍耐，这不坏。但由忍耐变成残忍这就坏了。忍于己是忍耐，忍于人就变成残忍了。又如创和撞也如此，这告诉人要创造非撞不可。这都是从字面上得到的意思。关于这些，不再说了。我主要的意思就是说研究不难，只要肯找了又找，只要一步一步作去，就能成功。但研究不是study，那是从书本上求知识；而是Research，就是要发现真理！

附带要讲讲研究室的任务。所谓研究室英文作 Seminary，它的字源与种子，精同义，所以它的本义就是利于产生新品种，新性质的地方，也就是可以产生新东西的地方。我们的研究室就要本着这个意旨去做。有新学问就有新行动，有新行动就能创造新事物。研究新道理不仅是知的，而且是行的，是用的。自然科学有社会功用，社会科学，经济学，政治学……更有社会功用，我们应本此精神使整个学术，整个学术界都服务于人民，都服务人类！

我十二年不教书，一个原因也许是因为有人认我一教书就会谈到政治。其实，这是一种误会，在学我要言学。近年我深深感觉中国需要一种新学风。尤其是大学或学院更要注意研究。我所谓新的学风就是深厚，笃实，精核的学风，这与研究有密切关系。中国从八十年前起办新教育，其中有抄英国的，有抄日本的，以至德国的，美国的，但抄的全是皮毛。今日除非不办教育，要办教育就应根据自己的实况，自己的需要，独立地来办。在学问上，今日一定要加强研究，必非深造自得不可，必须养成大器，再不可浅尝辄止，皮毛肤浅。有人说，我不要做学者专家，我为何要研究？但你只要是一个人，为社会做一点事，你总要有一些东西，总要有一个专长。因此，我们在学校里的时候就要深厚；联带着来的就要脚踏实地，实是求是，就要认真，就要实实在在；再一点就要精核，要一丝不苟，绝不等于漠忽笼统。我认为学校应该如此，应该造成这样的学风，应该培植这样的学生，这就需要大家共同努力。

（原载1948年4月20日《天琴》创刊号。选自河北人民出版社2005年版《张申府文集》第二卷，有删节。）

《计算之魂》吴军 人民邮电出版社

计算的本质——从机械到电子

策划人高博介绍：深度+广度

目标：

1. 理解计算科学的精髓和灵魂
2. 坚定长期投入和发展的信心——职业潜力，职业道路，发展前景广阔
要潜下心来，扎实投入

全书结构：

CP1: 算法级衡量算法优劣

CP2: CP3: 抽象与表示 计算机存储与处理数据的原理

CP4: 分类与组合

CP5: 图论及应用

CP6: 分治 结合4

CP7: 存储 时空的权衡

CP8: 并行与串行 CP9: 状态与流程 CP10: 概率算法及应用

如何对待习题与思考题：

1. 辅助理解书中内容，在对习题的思考与解答中进一步深化对书中内容的理解
- 2.

问题：

1. 面对众多的计算机科学研究方向，对选择什么感到困惑？

比如说现在火的人工智能，深度学习.....

发展潜力很大的计算机体系结构

怎样分辨，怎样选择

工程师的分级：每一级的贡献差十倍以上

六级：~码农

五级：能独立解决问题，完成工程工作。熟练掌握编程语言，必备的基本技能

四级：

什么是计算机

计算 计算机

1+1=2

数（概念or“状态”的抽象or编码） 运算符（规则，规则与数的概念无关，但规则的组合却能实现“计算”）

计算机（执行计算的机器-机械运动）

软件与硬件的分野——数与数的物理表示

算盘：为什么古代中国的算盘是最早的“计算机”？

算盘及算盘的使用包含了计算机与计算的内核：

算盘珠子：数的表示

三下五除二：计算的规则

人拨动算珠：计算机的机械运动

对谁做运算和实现什么样的运算：程序

数的表示与进制的选择：

为什么选择二进制来表示数？

巴比奇：想用复杂的物理计算装置，实现复杂的计算。

思路的改变：在布尔代数与香农的布尔代数实现运算进而实现逻辑控制

简单的基本计算装置，按照简单的规则，进行了复杂的组合，实现了复杂的计算

图灵与图灵机：什么是计算？

计算的跨越

纸带：状态与“程序”

读写头：读取与输出

读写头遵循的规则：计算的规则

能够实现一切我们所定义的“计算”——通用计算

冯诺依曼结构

1. 数据和指令的统一表示，不同解释

数据和指令的等价认识，认识的突破

2. 极致的简单才能构筑大规模的复杂

从算盘到芯片：材料学，工程上的改进

计算的边界：

可计算就是图灵机能进行的计算

现代计算机：非无限存储纸带，图灵机的物理近似

《哥德尔、艾舍尔、巴赫：集异璧之大成》

边界：

数学的边界：

计算的边界：

工程的边界：（可接受的时空成本）

日常问题求解的一般思路

- 问题建模

第一章 毫厘千里之差——大O概念

1.1 算法的规范化和量化度量

计算机算法基础分析鼻祖：高德纳

编程的自我要求：

力争一次全对，没有错误，算法在设计时就达到最佳。

遇到问题时解决问题的积极态度。

思考题1.1：

什么产品和计算机类似，是软硬件分离的？

解答：乐器，汽车...

1.2 大数和数量级的概念

我们对大数没有概念——>对计算机资源没有概念：

在自己使用计算机时，绝大部分时候，都感觉速度超级快，内存用不完，根本不会在意计算资源的耗费。

要逐步的培养起对运算量、运算速度、运算空间的“感觉”来。这要求对算法做空间和时间的复杂度分析，根据限制条件，计算量进行预算。

高德纳算法分析思想：

- 1.只考虑数据量大到接近无穷的情况。
- 2.将决定算法快慢的因素划分为 不随数据量变化的因素 与 随数据量变化的因素。

思考题1.2：

如果一个程序只运行一次，在编写它的的时候，你是采用最直观但是效率较低的算法，还是依然寻找复杂度最优的算法？

寻找最优的算法，因为此算法可能在其他地方复用。

1.3 怎样寻找最好的算法

例题1.3 总和最大区间问题

给定一个实数序列，设计一个最有效的算法，找到一个总和最大的区间

输入格式：

第一行：一个正整数N，表示序列长度

第二行：N个实数，以空格为分隔

1. 三重循环 $O(N^3)$

- 确定一个子区间需要起点p和终点q，再由p到q遍历求和，共三重循环。

```
#include <bits/stdc++.h>
using namespace std;

vector<double> arr;
int N;
double tmp;
int p, q, mp, mq; //记录左右下标
double maxSum, sum; //记录当前最大子序和与子序和

int main(){
    cin>>N;
    for(int i=0; i<N; i++){
        cin>>tmp;
        arr.push_back(tmp);
    }
    for(p=0; p<N; p++){
        for(q=p; q<N; q++){
            sum = 0;
            for(int i=p; i<=q; i++){
                sum+=arr[i];
                if(sum>maxSum){
                    maxSum=sum;
                    mp=p;
                    mq=q;
                }
            }
        }
    }
    printf("maxSum: %.2f\nmp: %d\nmq: %d\n", maxSum, mp, mq);
    for(int i=mp; i<=mq; i++){
        printf("%.2f ", arr[i]);
    }
}
return 0;
}
```

2. 两重循环 $O(N^2)$

- 子序列和没必要每次从头开始遍历累加，只需要在前一个子序列和之上加减即可。
- 在三重循环的代码上稍作修改

```

#include <bits/stdc++.h>
using namespace std;

vector<double> arr;
int N;
double tmp;
int p, q, mp, mq;
double maxSum, sum;

int main(){
    cin>>N;
    for(int i=0; i<N; i++){
        cin>>tmp;
        arr.push_back(tmp);
    }
    maxSum=arr[0];

    for(p=0; p<N; p++){
        sum=0;
        for(q=p; q<N; q++){
            sum+=arr[q];
            if(sum>maxSum){
                maxSum=sum;
                mp=p;
                mq=q;
            }
        }
    }

    printf("maxSum: %.2f\nmp: %d\nmq: %d\n", maxSum, mp, mq);
    for(int i=mp; i<=mq; i++){
        printf("%.2f ", arr[i]);
    }

    return 0;
}

```

3. 分治法 $O(N \log N)$

- 序列S的最大和子序列分如下三种情况:
- 1. 完全位于左半区间 $[p_1, q_1]$
- 1. 完全位于右半区间 $[p_2, q_2]$
- 1. 横跨中间 $[p_1, q_2]$

1和2可递归调用，3经简单分析可知，其区间为 $[p_1, q_2]$

第三种情况有问题，书上称：

2. 前后两个子序列的总和最大区间中间有间隔，我们假定这两个子序列的总和最大区间分别是 [p1,q1] 和 [p2,q2]。这时，整个序列的总和最大区间是下面三者中最大的那个：

- (1) [p1,q1]
- (2) [p2,q2];
- (3) [p1,q2]。

至于为什么，这是本节的思考题。

Page: 037

但是，横跨中间的可能的最大子区间，不一定为 [p1,q2]。举例来说明：序列：

[-2, 1, -3, 4, -1, 2, 1, -5, 4]

$$\text{mid} = (0+8)/2 = 4$$

划分为：

左半子序列： [-2, 1, -3, 4, -1]

其最大和子序列为： [4]，对应下标区间为： [3,3]，即 $p1=q1=3$ ， $lmsum = 4$

右半子序列： [2, 1, -5, 4]，

其最大和子序列为： [4]，其对应下标区间为： [8,8]，即 $p2=q2=8$ ， $rsum = 4$

进而，[p1,q2]对应的区间为 [3,8]，对应子序列和为 5

然而，可知实际最大和子序列为 [4, -1, 2, 1]，对应下标区间为 [3, 6]，对应子序和为 6
并不是 [p1,q2]

按书中描述的代码：

```

#include <bits/stdc++.h>
using namespace std;

void maxSubSum(vector<double> arr, int p, int q, int &mp, int &mq, double &maxSum);

vector<double> arr;
int N=0;
int mp=0, mq=0;
double tmp=0, maxSum=0;

int main(){
    cin>>N;
    for(int i=1; i<=N; i++){
        cin>>tmp;
        arr.push_back(tmp);
    }

    maxSubSum(arr, 0, N-1, mp, mq, maxSum);

    printf("maxSum: %.2f\nmp:%d\nmq:%d\n", maxSum, mp, mq);
    for(int i=mp; i<= mq; i++){
        printf("%.2f ", arr[i]);
    }
    return 0;
}

void maxSubSum(vector<double> arr, int p, int q, int &mp, int &mq, double &maxSum){
    //边界情况
    if(p>=q){
        mp=mp=p;
        maxSum=arr[p];
        return ;
    }

    int mid = p+(q-p)/2;
    //mid = (p+q)/2;
    int lmp, lmq, rmp, rmq;
    double lmsum, rmsum, midsum=0;
    double tmpsum=0;
    //递归情况
    maxSubSum(arr, p, mid, lmp, lmq, lmsum);
    maxSubSum(arr, mid+1, q, rmp, rmq, rmsum);

    /*
    midsum错误，原本想midsum应该起止于mp, mq, 想当然以为要包含lmsum, rmsum了;
    应该按照从中间向两边扩展
    */
}

```

```

/*
for(int i=lmp; i<= rmq; i++){
    midsum+=arr[i];
}

if(lmsum>rmsum && lmsum>midsum){
    mp=lmp;
    mq=lmq;
    maxSum=lmsum;
}
else if(rmsum >lmsum && rmsum>midsum){
    mp=rmp;
    mq=rmq;
    maxSum=rmsum;
}
else{
    mp=lmp;
    mq=rmq;
    maxSum=midsum;
}
return ;
}

```

输入:

```

9
-2 1 -3 4 -1 2 1 -5 4

```

输出:

```

maxSum:5.00
mp:3
mq:8
4.00 -1.00 2.00 1.00 -5.00 4.00

```

期望输出:

```

maxSum:6.00
mp:3
mq:6
4.00 -1.00 2.00 1.00

```

修正:

重新考虑跨越中间(mid)的情况: 从mid开始, 向序列左右扩展, 保留最大值及对应下标

对应代码为:

```

#include <bits/stdc++.h>
using namespace std;

void maxSubSum(vector<double> arr, int p, int q, int &mp, int &mq, double &maxSum);

vector<double> arr;
int N=0;
int mp=0, mq=0;
double tmp=0, maxSum=0;

int main(){
    cin>>N;
    for(int i=1; i<=N; i++){
        cin>>tmp;
        arr.push_back(tmp);
    }

    maxSubSum(arr, 0, N-1, mp, mq, maxSum);

    printf("maxSum: %.2f\nmp:%d\nmq:%d\n", maxSum, mp, mq);
    for(int i=mp; i<= mq; i++){
        printf("%.2f ", arr[i]);
    }
    return 0;
}

void maxSubSum(vector<double> arr, int p, int q, int &mp, int &mq, double &maxSum){
    //边界情况
    if(p>=q){
        mp=mp=q=p;
        maxSum=arr[p];
        return ;
    }

    int mid = p+(q-p)/2;
    //mid = (p+q)/2;
    int lmp, lmq, rmp, rmq;
    double lmsum, rmsum, midrsum=0, midlsum=0;
    double tmprsum=0, tmpltsum=0;
    //递归情况
    maxSubSum(arr, p, mid, lmp, lmq, lmsum);
    maxSubSum(arr, mid+1, q, rmp, rmq, rmsum);

    /*
    midsum错误，原本想midsum应该起止于mp, mq, 想当然以为要包含lmsum, rmsum了;
    应该按照从中间向两边扩展
    */
}

```

```

/*
int midp=mid, midq=mid+1;

for(int j=mid; j>=p; j--){
    tmplsum+=arr[j];
    if(tmplsum>midlsum){
        midlsum=tmplsum;
        midp=j;
    }
}

for(int i=mid+1; i<=q; i++){
    tmpsum+=arr[i];
    if(tmpsum>midrsum){
        midrsum=tmpsum;
        midq=i;
    }
}

double midsum=midlsum+midrsum;
if(lmsum>rmsum && lmsum>midsum){
    mp=lmp;
    mq=lmq;
    maxSum=lmsum;
}
else if(rmsum > lmsum && rmsum>midsum){
    mp=rmp;
    mq=rmq;
    maxSum=rmsum;
}
else{
    mp=midp;
    mq=midq;
    maxSum=midsum;
}
return ;
}

```

4. 正反两遍扫描法 O(N)

类似于前缀和?

```

#include <bits/stdc++.h>
using namespace std;

double maxSubSum(vector<double> arr, int l, int r);

vector<double> arr;
int N;
double tmp;
int main(){
    cin>>N;
    for(int i=1; i<=N; i++){
        cin>>tmp;
        arr.push_back(tmp);
    }
    maxSubSum(arr, 0, N-1);
    return 0;
}

double maxSubSum(vector<double> arr, int l, int r){
    int mp,mq, p, q;
    mp=mq=p=q=l;
    int i;

    double maxsum=0, foresum=0, maxforesum=0;
    for(p=l; p<=r; p++){
        if(arr[p]>0){//找到大于0的项
            //printf("p: %.2f\n",arr[p]);
            //sum=arr[p];
            foresum=0;
            maxforesum=0;
            for(i=p; i<=r; i++){
                foresum+=arr[i];

                if(foresum>maxforesum){
                    maxforesum=foresum;
                    q=i;
                    //printf("maxforesum: %.2f\n",maxforesum);
                }
            }

            if(foresum<0 || i==r){//foresum<0或者已经到序列尾部了
                if(maxforesum>maxsum){
                    maxsum=maxforesum;
                    mp=p;
                    mq=q;
                }
            }
        }
    }
}

```

```

        break;
    }
}

p=i;
}

}

printf("maxSum:%.2f\nmp:%d\nmq:%d\n",maxsum, mp, mq);
for(int i=mp; i<= mq; i++){
    printf("%.2f ", arr[i]);
}
return maxsum;
}

```

5. 动态规划dp O(N)

将问题分解考察，首先，最大子序和对应的区间为[p, q]，直观上感觉到，下标可以作为某种“状态”的标识。

状态定义：dp[i]时以下标i为结尾的最大子序列

状态转移关系/方程：

当 $dp[i-1] > 0$ 时， $dp[i] = dp[i-1] + arr[i]$
当 $dp[i-1] \leq 0$ 时， $dp[i] = arr[i]$

初始化： $dp[0]=arr[0]$ ；

代码实现如下：

```

#include <bits/stdc++.h>
using namespace std;

template<class elemType>
elemType maxSubSum(vector<elemType> arr, int r, int l);

int N;

int main(){
    cin>>N;
    vector<double> arr;
    double tmp;
    for(int i=0; i<N; i++){
        cin>>tmp;
        arr.push_back(tmp);
    }
    double maxsum=maxSubSum(arr, 0, N-1);
    return 0;
}

template<class elemType>//使用模板来提供更好的可移植性
elemType maxSubSum(vector<elemType> arr, int r, int l){
    int size = arr.size();
    //下标为i处的dp，代表：以i为最后一个元素的最大子序列和
    //当dp[i-1] > 0时, dp[i] = dp[i-1]+arr[i]
    //当dp[i-1]<=0时, dp[i]= arr[i]
    //用maxdp维护所有“以i结尾的最大子序列和”中最大的那个，即整个序列的最大子序列和
    elemType maxdp=0, dp=0;

    for(int i=0; i<size; i++){
        if(dp>0){
            dp+=arr[i];
        }
        else{
            dp=arr[i];
        }
        if(dp>maxdp){
            maxdp=dp;
        }
    }

    //printf("maxdp: %.")
    cout<<"maxdp: "<<maxdp;
    return maxdp;
}

```

6. 简单扫描 O(N)

首先扫描到一个大于零的数，以它为起点，向后累加得到thisSum，
每次累加得到的hisSum与maxSum比较并去留，
因为maxSum对应的子序列不能以负数开头，所以，当thisSum<0时，
重置累加起点和thisSum，重复上面的流程。

```

#include <bits/stdc++.h>
using namespace std;

double maxSubsequenceSum(vector<double> a, int &start, int &end);

int N;
double maxsum;
int p, q;

int main(){
    cin>>N;
    vector<double> arr;
    double tmp;
    for(int i=0; i<N; i++){
        cin>>tmp;
        arr.push_back(tmp);
    }
    double maxsum=maxSubsequenceSum(arr, p, q);
    printf("maxSum:%.2f\nmp:%d\nmq:%d\n",maxsum, p, q);

    return 0;
}

//枚举分析改进
/*
首先扫描到一个大于零的数，以它为起点，向后累加得到thisSum,
每次累加得到的hisSum与maxSum比较并去留,
因为maxSum对应的子序列不能以负数开头，所以，当thisSum<0时,
重置累加起点和thisSum，重复上面的流程
*/
//有特殊情况，需要改进，遇到<0的序列不能全盘否定，要保留最优
double maxSubsequenceSum(vector<double> a, int &start, int &end){
    int maxSum=0, thisSum=0, starttmp=0;
    start=end=0;
    int size= a.size();
    for(int i=0; i<size; i++){
        thisSum+=a[i];
        if(thisSum<0){
            thisSum=maxSum=0;
            starttmp=i+1;//遇到thisSum<0时，只修改starttmp
        }
        else if(thisSum>maxSum){
            maxSum=thisSum;
            start=starttmp;//只有thisSum>maxSum时才更新start
            end=i;
        }
    }
}

```

```

    }
    if(start==size){start=end=0;}
    return maxSum;
}

```

思考题1.3

- Q1. 将例题1.3的线性复杂度算法写成伪代码。
- Q2. 在一个数组中寻找一个区间，使得区间内的数字之和等于某个事先给定的数字。
- Q3. 在一个二维矩阵中，寻找一个矩形的区域，使其中的数字之和达到最大值。

解答：

Q2:

参考“两数之和”问题求解的思路：

```

procedure twoSum(list, target)
    map: list[i]->i

    for e, i in list, listindex:
        if target - e not in map :
            map[e] = i;

        else if target - e in map :
            print(map[e], map[target-e])
            //找到了

```

解答： step1: 建立字典/哈希表map: 数组list的前缀和 -> 前缀和对应的下标q

因为，若 $S(p,q) == target$ ，就有 $S(1, p-1) = S(1, q) - S(p,q) = S(1, q) - target$

step2: 遍历到q，得到 $S(1,q)$ ，查询map中有无 $key=S(1, q) - target$ 如果有，则返回 $p=map[key]+1$

其中，map的建立可与遍历同时进行

1.4 关于排序的讨论

思考题1.4

- Q1. 赛跑问题（GS）。
- 假定有25名短跑选手比赛争夺前三名，赛场上五条赛道，一次可以有五名选手同时比赛。比赛并不计时，只看相应的名次。假设选手的发挥是稳定的，也就是说如果约翰比张三跑得快，张三比凯利跑得快，约翰一定比凯利跑得快。最少需要几次比赛才能决出前三名？
- Q2. 区间排序。
- 如果有N个区间 $[l_1, r_1], [l_2, r_2], \dots, [l_N, r_N]$ ，只要满足下面的条件我们就说这些区间是有序的：存在 $x_i \in [l_i, r_i]$ ，其中 $i=1, 2, \dots, N$ 。

- 比如， $[1, 4]$ 、 $[2, 3]$ 和 $[1.5, 2.5]$ 是有序的，因为我们可以从这三个区间中选择1.1、2.1和2.2三个数。
同时 $[2, 3]$ 、 $[1, 4]$ 和 $[1.5, 2.5]$ 也是有序的，因为我们可以选择2.1、2.2和2.4。但是 $[1, 2]$ 、 $[2.7, 3.5]$ 和 $[1.5, 2.5]$ 不是有序的。
- 对于任意一组区间，如何将它们进行排序？

-
- 引子 计算的本质-从机械到电子
 - 第一章 毫厘千里之差-大 O 概念
 - 第二章 逆向思考-从递推到递归

论青年修养-张申府-1946

心不静，写不出系统深刻的东西来。无可如何，这是随便谈谈罢。但这样子，也许比起装腔作势，板起面孔来说话，读者可以感着更亲切一点儿。

关于青年修养，我现在有三五点意思涌上心头。现在就以次分别写在下边。

第一，我也与许多古人一样，总觉着一个青年，**为学必先立志**。就令一个人，不必专门读书，有一个大志向，也是非常之要紧。一个人但令多少有点儿知识，多少有点儿自觉，**那就要有一个志向，这样子才可以免得麻麻糊糊过一辈子**。

至于立志作什么，那却有点儿难说，但至低限度，最一般地来说，你要立志作一个好人。这话也许大空，那就反过来说也可：你总要立志不为恶。凡你平常骂人的事，你总要下决心一件也不作。这一点在今日实在最最要紧。许多人两面作风，许多人口是心非，许多人口称民主而行反民主，许多人天天骂当局，而他的行为没有一点不与当局一样，除了地位不同以外。诸如此类，都因他没有坚定志向的缘故。

说得更具体一点，一个人总要作一个合乎时代的人，因此应该对于自己的时代不可不有一点切实的认识。但这地方很容易犯一个毛病，那就是随波逐流。一个人知识行动合乎时代是必要的，但随波逐流可就大要不得。**怎样免掉这种毛病，那就要注意自觉，作得了自己主宰，有自得之处，尽量防备虚荣，并且对时代有深刻的认识**。

人怎样才能有志，尤其怎样才能有大志，这原因颇不简单。从外来原因说，这一种要靠父兄师长的告语教导。这不是人人所可得。一种靠同学朋友的切磋鼓励。这也可遇不可期。另一种就是靠阅名人传记，读大家著作；甚至看名家小说、戏剧、电影，也都会有好处。这是人人都可作得到的。**但是你这样子作时，你必懂得体察，懂得与好人看齐。换言之，要作一个好人，适当的自觉总是必要的**。

再进一步说，在今日这个时代，要作一个好人，拿旧话说，“民胞物与”总是一个必要的出发点。说得通俗一点，你要有一种治病救人的意趣。你不要把自己孤立，你要使世界因有你而不同，但你都不可总觉着你与一般人不同。

第二，**关于为学读书，我特别愿意告诉你一个“专”字**。本来，凡事，“专”都是最首要的成功诀。读书为学，也不外是。一个人要容易有成，那就最好只干一样事。古人讲学，常说博与约。但博如没有中心，必至泛滥无归，事倍功半，费力而不讨好。一个人读书，与其对一切知道一点，确不如对一点知道一切。等到你有了中心，有了主宰，有了专长之后，再对一切都知一点，那就正可以作你原来一点的必要补助。所谓由博返约，能约，也就不妨博。

前已说过，要读大家的名著。这也是为学读书上的一个必要的要诀。以专而言，与其泛览群籍，不如精读一书。但这一书必须是大家名著，不刊的经典，意味深长，使你研寻不尽者。这种书，不拘那个文明国家，自古以来都是有的。一个人读书，最好读到深造自得。大家的名典当然都是深造自得的书。不深造而有自得处，必不会开辟新纪元、创发新时代。大家名著必有不同气味，正与名乐一般。你如与它化了，你自也可以不同。

我以前尝为青年读者写过一篇以“切实，深入，专”为题的东西。深入与专，当然有联带关系。此外，最要紧也相关的，那就要说到切实。一个人作人，最怕作到飘飘然。一个人在有些地方能够飘飘若仙，未尝没有好处。但如全不着实，全不实在，全不入里，尽是肤面表毛，油腔滑调，花言巧语，那就只能说他在作人上已经失败。读书为学也如此。不拘怎样抽象的学问，最后也不能不切实际。有的人讲学尚“空灵”，其实正是为的“如实”而不执着。假使一个人一生为学，而却全与实际不相干，那就是时力精神白费了。前说读书要读大家名著，假使这种书是现代的或讲现代的，那就更好。当然我并不是说古书就不切实际。

再补充一句。一个人读书要专，要读名著，这都说过了。但一个人有丰富的人生常识也有其必要，特别是关于你的时代，你的世界，你的国家，你的社会，以及你的身体精神的常识。一个人如果没有关于生理、心理、卫生的常识，必会常在苦恼中。

第三，前边已经提到自觉，我现在要更进一步，再加上反省。一个人不识不知的生活是一种无意思的生活；一个人不长进的生活也可说是与死差不多的生活。人怎样才能不断长进？条件之一就是时时反省。怎样反省？就是你要时时自己检讨：这件事我为什么作得很成功？那件事我为什么失败了？昨天我身体那样好，今天我为什么病了？以前我这样作很顺利，现在为什么行不通了？一个有理性的人，不但要事事自觉，事事要作得有理由，而且也要成功知道理由，失败也知道理由。反省了以后，更不惮于败，这便是进步所由成。

我近来很感到反省的必要。现在许多人作事，如有错过，总是加在别人身上，或加在不能自表的客观环境身上，绝不肯回头看看自己。这样的坚决信心，这样的勇往气概并不是没有是处，但是事情弄得不好，前途也会弄得不堪设想。因此，除了相反相成，不要过分以外，我总愿教人回头看看，也愿教人有时也作一作退一步想。也许有人要感着打了他们的高兴。其实我不过愿他在一往直前上同时也要脚步放得稳一些，不要失足，失掉不必要的牺牲而已。

中国过去有许多在人的修养上特别注意的事，也许因为有了流弊，现在遂因噎废食，再不复提的。这其中一个就是一个敬字。我近年大大感到敬字的要紧。敬不必对人，尤其要紧的还在对事。我所谓敬，差不多就是小心慎重的意思，但更加了一番庄严郑重。我近年每逢什么弄坏了，即自谓不敬不敬，以自警惕，以自改正。这是与反省相联的。我相信，假使人常能如此，一定也可减少些过失。我深愿今日青年都能早点养成这个习惯。所谓修养，也不过就是养成些好习惯，尤其是沉着慎重不轻浮的习惯。

有大志，读名典，时自反省，对事专而敬。青年的应有修养，当然还不止。这三五点却是我近来时在感到之点。勉强抽空写出来，但愿大家不吝，试试看！

(四月廿六夜)

张申府（1893—1986），名崧年，河北献县人。哲学家。1920年初参与中国共产党的建党活动，是周恩来的入党介绍人。1931年至1936年任清华大学哲学系教授。

（作于1946年4月26日。原载1946年5月4日《唯民周刊》第1卷第5期。）

论青年的修养-张闻天（1938年4月28日）

同志们！你们都是从全国各方面来的青年，所以我今天就同你们讨论讨论关于青年的问题。

从全国民族抗战开始到现在这个时期内，青年的确占着一个很重要的地位。我们无论从哪一方面来看，不论是军事方面也好，政治方面也好，文化教育方面也好，都可以看到广大青年群众的活跃。他们总是站在抗战的最前线，为中华民族的解放而牺牲奋斗。同时，由于抗战规模扩大和展开，青年的责任也更加重要了。怎样使我们的青年更能在抗战中尽他们的责任，是我们大家所要商讨的问题。我今天所要讲的青年的修养问题，即是想在这方面向大家贡献一点意见。

我认为中国现代的青年，有很多共同的优点，同时也有一些共同的弱点。怎样发挥这些优点，克服那些弱点，即是青年的修养问题的内容。我现在就根据我们的经验与考察来具体讲几点，希望大家来讨论。

一、要有坚定的高尚的理想

青年有一个很大的优点，这就是他们有高尚的理想，不论这个理想是抗日救国也好，共产主义也好。他们对于现代社会是不满足的，他们希望着一个理想的社会的出现。青年的这种理想是最可宝贵的东西。正是这种理想不断地鼓舞着我们的青年向前进步。正是这种理想使他们不愿在现代社会中醉生梦死，而愿意为理想社会的实现牺牲奋斗。正是这种理想，使我们的青年，在过去，在现在，创造了许多惊天动地与可歌可泣的伟大事业。眼前的例子，也可证明这一点。比如你们，陕北公学的同学，究竟为了什么不远千里而来陕北这个比较荒凉的地区呢？陕北公学的物质条件是很困难的。住的是窑洞，吃的是小米饭。教育设备都很简陋。然而你们为什么不怕一切困难而来呢？如果你们不是为了寻找高尚的理想，我想你们是不会来的。所以高尚的理想对于青年是最可宝贵的东西。我们对于青年的理想，不但不应该鄙视，而且应该极大地爱护。旧社会里的人，常常骂青年为“理想太高”。这种骂法，我们认为是根本错误的。这只是证明那些骂青年的人的无理想之可鄙而已。孙中山先生三民主义的理想，不是也给人家骂为“大炮”吗？马克思、恩格斯、列宁、斯大林的共产主义理想，不是也给人家骂为“空谈”吗？然而我们知道马克思、恩格斯、列宁、斯大林、孙中山的伟大，也正是在于他们有高尚的理想。所以青年有高尚的理想，正是青年的优点。

但是如果我们把青年人的某些理想研究一下，那我们即可发现有些青年人的理想不是理想而是空想。空想可能常常是美丽的东西，然而究竟是空想而不是理想。那么，理想与空想的区别究竟在哪里呢？

首先，我们应该说，我们的理想是建筑在现代社会的物质基础之上的东西。

空中楼阁，究竟是幻想而不是理想，因为它是脱离现代社会的物质基础而建立在空中的。一切伟大的理想，都从现代社会的具体分析得来。闭户造车究竟只能是幻想而不是理想，因为它是个个人头脑中间随便想出来的，而不合于具体的与实际的物质条件。共产主义的理想，是人类有史以来最崇高的最伟大的理想，然而这个理想，只能从资本主义社会的物质基础上产生出来。脱离对于资本主义社会的科学分析的“大同世界”的思想，究竟是幻想而不是理想。

其次，我们的理想应该适合于人类社会发展的必然趋势。

我们说民族独立、民权自由、民生幸福的三民主义可以是我们今天的理想，因为中国社会今天正是向着这个方向发展着的。我们说共产主义社会是我们将来的理想，因为人类社会终究是要向着这个方向发展前去的。如果今天有人要恢复原始共产社会或恢复奴隶社会。或仿效君主专制或法西斯独裁的统治。那这种思想就不是真正的理想，因为这种思想是违反于人类现代社会发展的趋势的。这种思想，我们不叫它理想，而叫它做反动思想。因为这种思想不是要使社会走向前进，而是要它转向后退，就是所谓“开倒车”。一切不合于人类社会发展趋势的思想，只能是一种空想。

第三，理想与空想不同，就是理想是能够实现的，空想则是永远不得实现的。

而理想的实现，需要有在一定的物质条件下所产生的社会力量。没有这种力量，理想也就不能实现。战胜日本帝国主义、争取民族独立的理想，就要依靠在全国一切不愿意当亡国奴的各阶级、各党派、各团体的民族统一战线的力量之上。离开实现自己理想的社会力量而谈理想。那也是一种空想而不是理想。共产主义的最高理想，离开工人阶级的社会力量，就成为乌托邦，成为空想。

第四，理想之为理想，除上述条件外，还需有实现理想的具体办法。

比如，实现民族解放的理想，就要有建立、扩大与巩固抗日民族统一战线的办法。实现共产主义，就要有一定的纲领与步骤，一定的策略与战略。马克思列宁主义的策略与战略是每一共产主义者所必须学习的科学，因为离开了它们，理想虽是很好，然而仍然成为空谈而不能实现，使理想成为空想。

这就是理想与空想的基本区别。我们的青年如果赞成我所说的这些区别，那末请你们就拿这几个标准，来检查一下自己的理想：究竟你们过去所抱的理想，是理想还是空想？如果是空想，那就应该决然抛弃。如果是理想，但又不完全，那就应该修正与充实，或重新加以检讨与整理。**我们应该不客气地说，在我们青年的理想中，常常不免有些空想的成分，因此，他们有时对于自己的理想把握不定，而发生一些朝三暮四的摇摆现象。要免去这种现象，就必须把自己的理想建筑在结实坚固的科学的基础之上。**这是我所要说的第一个问题。

二、要为实现自己的理想奋斗到底

青年人在一旦觉醒，找到了自己的理想之后，都能够不顾一切为自己的理想奋斗。这种为实现自己的理想而牺牲奋斗的精神，是我们青年的一个很大优点。我们在社会中看到不少这样的人，他们也不满意于现状，他们也希望有个较好的社会出现，他们甚至同情共产主义，但是他们自己不愿意为改造社会而奋斗。他们舍不得自己的妻子儿女，他们舍不得自己的家乡故土，他们舍不得自己的生命财产，他们不得不“马马虎虎”地生活，不得不“做一天和尚撞一天钟”。但青年就不是这样。他们了解到自己的理想则已，一旦他们了解到了自己的理想，他们就会抛弃一切，奔赴自己的理想，去为自己的理想而奋斗。这种不顾个人利害而牺牲奋斗的精神，是值得钦佩与歌颂的。

但是，实现理想，实在不是一件容易的事。**理想虽是建筑在现代社会的物质基础之上，但理想是超过现代社会的东西。理想好比泥土中生长出来的花。它虽生长在泥土中，但它又不是泥土。所以理想看来常常是美丽的，而现代社会则看来是丑恶的。要把这个丑恶的现代社会变为美丽的理想社会，那决不是一天两天以至一年两年的事，而需要几十年以至上百年的奋斗与工作。不但这样，在奋斗与工作的过程中还必然要碰到无数的困难与波折，有时甚至看来似乎是不能克服以至绝望的困难。所以不论在任何困难之下，坚持自己的理想，坚持为自己理想的实现而奋斗，是绝对必要的。没有这种坚持性，任何的理想也都不能实现。**

一切伟大的革命家之所以伟大，不但因为他们有着伟大的理想，而且还因为他们始终能够为了自己的理想奋斗到底。孙中山先生“致力国民革命凡四十年”。虽是在他奋斗的过程中碰到了无数的困难，然而他始终坚持他的理想，为他的理想奋斗到最后。世界无产阶级的导师马克思、恩格斯、列宁、斯大林，他们的伟大，就是他们那种为自己的理想而坚持奋斗到底的精神。他们在丑恶的旧社会中看到美丽，他们在黑暗中看到光明，他们排除万难、克服一切困难而前进。

中国共产党人曾经完成了二万五千里长征。这件事震动了全世界。为什么二万五千里长征能够有这样伟大的影响呢？原因就在于中国共产党在这次长征中充分地表现出了它为了自己的理想而牺牲奋斗与坚持到底的精神。没有这种精神，就是一千里的长征也是不可能的。在这次长征中，我们的确曾经碰到了无数的困难。我们曾经碰到了几乎不能渡过的天险金沙江与大渡河，我们曾经碰到了人类几乎没有到过的雪山与草地，我们处在敌军的四面包围之中。困难几乎是不能克服的，然而我们那时只有一个思想，就是无论如何要克服这些困难，要为自己的理想奋斗到底。最后，我们还是完成了我们当时的任务，到达了当时的目的地。

就以今天的抗战为例吧，中国的民族抗战是一个持久的战争，困难也是很多的，波折也是很多的。如果我们碰到一些困难，遭受一些挫折，就悲观失望，就准备同敌人妥协，那中国就有亡国的危险。显然地，如果我们不能坚持抗战到底，则最后战胜日寇、实现民族独立的理想是不可能的。

正是在这个问题上，在为了自己理想的实现而奋斗到底的这个问题上，我们有些青年常常表示一些弱点。他们一开始往往以无限的热情与兴奋去奔赴自己的理想，但是一旦他们碰到困难，碰到波折，他们往往不能坚持到底，以至半途而废。过去常常有人以“五分钟热度”讥笑青年学生。这种讥笑在现在说来已经不对了，因为中国青年学生经过无数的斗争，现在已经有了极大的进步。然而在某些部分的青年中缺乏斗争的坚持性，则依然是不可否认的事实。

为什么在我们部分青年中发生这种现象呢？我以为有以下几个原因。

第一，我们的青年往往对于革命的持久性估计不足。

要把人类的高尚理想实现出来的革命，像我在前面说过的，不是一天两天以至一年两年的事，而是几十年以至上百年的事。这是一个持久的斗争。不但实现共产主义理想是如此，即最后战胜日寇也不是一年两年的事，而是一个持久战。同志们大多数是从西安步行到延安的。你们对于走陕北的山地，大概已经有了相当的经验。革命就好像走陕北的山地一样，翻过了一座山又一座山。**你们在路上不是常常想，翻过了前面一个山头一定再没有山了吧，一定可以到延安了吧。然而事实常常是相反的，山接连着山，而延安总是还在前面。**这样你们在路上就要走十天，才得到达延安。这对于你们，就是一个很好的入学考试。然而革命究竟比翻山要持久得多，困难得多。如果我们青年参加到革命中来，没有持久斗争的准备，结果是一定要失望与半途而废的。正像你们到延安来，如果你们没有这样一定要到延安的决心，你们一定会半途而返的。

第二，我们的青年往往对于革命的困难性估计不足。

要把人类高尚的理想实现出来的革命，不克服无数的困难是不可能的。革命决不像上海的大马路那样平坦好走，决不像吃饭睡觉那样容易简便。这简直同爬着没有人迹到过的崎岖的高山一样，山上没有路，没有人家，到处是荆棘与浓密的森林，到处有毒蛇猛兽的威胁，有些地方简直是不能越过的绝壁。但是我们必须前进，必须克服一切困难前进。我们有时会被荆棘树枝所刺伤，有时会被毒蛇猛兽所咬伤，我们前面的伙伴有时会用尽了一切气力面倒毙，但是我们必须前进。如果我们的青年参加到革命中来，不预先看到这种困难，没有克服一切困难的准备，那结果必然是“知难而退”。

第三，旧社会的思想习惯以及一切物质上的诱惑，也往往是使青年半途而废的一种极大的力量。

我们常常看到一有志的青年，受不起这种力量的压迫而退却了，而放弃了自己的理想。旧社会对于青年的理想，总是取着敌视的态度。他们用各种思想上的毒素，讥笑青年，陷害青年，使青年“老大”，“颓丧”，“消沉”。他们用地位、金钱、美女腐化青年，笼络青年，消磨青年的“朝气”。你们看到过无数这类的例子吧。当青年学生在学校读书的时候，往往“志向远大”，想“给国家民族谋点幸福”，“为人类解放做番事业”，然而一出学校到旧社会中混上几年，往往把过去的一切完全抛弃了。旧社会吞没了他们，旧社会融化了他们。所以，如果我们的青年不能同这种旧势力奋斗，坚持自己的理想，不为这些力量所动摇，那结果也必然是半途而废。

第四，我们的有些青年本身常常有一种“动摇性”，缺乏足够的忍耐与坚定。

我们常常称这种特点为“小资产阶级性”。他们往往今天参加革命工作，就希望革命在明天胜利。如果明天革命不能胜利，那他们就失望而消极了。他们在革命形势高涨的时候，常常趋向狂热与盲动，而在革命形势低落的时候，则又转到消沉与绝望的深渊。他们胜利时常常为胜利冲昏头脑，失败时则又因失败而垂头丧气。他们不会把今天一点一滴的切实的工作，同他们远大的理想联系起来；他们不能穷年累月地去为自己的远大的理想而进行今天看来好像是没有结果的工作；他们常常讨厌这种“琐碎”与“麻烦”的工作；他们不能清楚了解，一切革命的工作，只要能够坚持地干下去，一定可以得到一定的成绩。巨大的建筑工程没有一砖一瓦的砌筑，是永远不能成功的。伟大的理想不经过许多胜利与失败，是永远不会实现的。

这些是常常使我们的青年不能坚持自己的理想、为自己的理想奋斗到底的原因。这些弱点的克服，是每一个中国青年的严重任务。这当然不是一件容易的事，然而这件事是必须做与可能做到的。

当然，今天我们离开我们的理想还很远，然而我们总是在一天一天接近着自己的理想。社会主义的理想，过去还只是在书本上，而现在已经在占全世界六分之一的土地的苏联，变成了完全的现实。所以只要我们能够坚持自己的理想，为自己的理想奋斗到底，我们的理想的实现是可能而且是必然的。

三、要学习实现理想的办法

有了理想，有了实现理想的决心，是不是就已经够了呢？还是不够的。我们青年还要有实现理想的办法。

青年有一种很大的长处，就是有热烈的革命情绪，纯洁坦白的胸怀。正是因为他们有热烈的革命情绪，所以他们能够勇往直前，不顾一切困难与个人利益，向着自己的理想前进。正是因为他们有纯洁坦白的胸怀，很少受到旧社会的各种陈腐的思想、习惯与传统的影响，所以他们比较容易接受革命的真理。青年同那些麻木不仁的人是不相同的，同那顽固守旧的死硬派也是不相同的。**青年是有生气的，活跃的，充满着愉快与光明的一代。他们是旧社会中生长起来的美丽的花，他们也是美丽的新社会的创造者。**

在中国革命史上，我们可以举出无数例子，来证明中国青年如何常常站在民族解放的前卫地位，以他们的头颅与热血创造了无数可歌可泣的英勇的光荣的史诗。在今天伟大的民族抗战中，我们也可以到处看到青年怎样为民族抗战的最后胜利而进行着勇敢无比的斗争。如果他们没有热烈的革命情绪与纯洁坦白的胸怀，则一切这些事业的创造是不可能的。我们共产党人对于青年的这一优良的特质，尤其有深切的感觉。当我们在最黑暗最困难的时候，当在我们的周围到处是恐怖与造谣污蔑的时候，我们总是从许多青年朋友那里得到他们的同情与拥护。在今天，他们对于我们所表示的爱戴与信任，也是我们不能以言语来表示感激的。谁都可以看到，在我们延安，没有官可做，没有薪水可拿，没有很好的生活可过，还有不少人在我们的周围造我们的谣言，破坏我们，然而你们仍然不远千里而来，其原因何在呢？除了你们的那种热烈的革命情绪与纯洁坦白的胸怀，还能有其他的说明吗？

青年的这种优良的品质，是我们所应极力爱护与发扬的。

但是光是热烈的革命情绪，纯洁坦白的胸怀，还是不能实现自己的理想，完成伟大的革命事业的。**有些青年的弱点，常常表现在不会使革命的热情去服从于革命的理智，不会把纯洁坦白变为对于真理的深刻的追求。**响亮的革命口号，耀眼的革命词句，可以迷惑我们的青年，使他们发狂，使他们以此为满足。他们往往抓住问题的一方面，把它发挥，把它夸大，认为这是唯一的“真理”，排除一切其他的方面，否认一切其他的意见。他们的革命热情容易变为盲目的冲动，他们的纯洁坦白容易走向片面性、狭隘性与幼稚病。他们往往不能更冷静地去考虑问题，想出具体办法来实现自己当前的任务，来实现自己的理想。这种弱点，当然是我们的青年所应该设法克服的。

我们应该使我们的青年清楚地了解到：在理想确定之后，有了为理想而奋斗到底的决心之后，学习实现理想的具体办法就有着决定的意义。在这一方面，我想向同志们贡献几点意见。

第一，就是要了解具体情况。

我们无论做什么事，首先的问题就是了解情况。正像一个在前线上指挥军队的将军，他为了要实现他打胜仗的理想，就必须讲求作战的方法。这里，首先就是要弄清敌情，然后来决定作战的计划。我们的工作方法也是如此。比如，你们将来从学校毕业之后，你们到一个地方去工作，你们的责任决不是到那里去照你们自己的头脑中所想的来乱干一顿。而是要首先了解当时当地的具体情况，比如那里的抗战形势，那里的政治与经济，各阶级、各阶层、各党派、各民众团体以及党、政、军、民间的相互关系等。没有这种了解，我们就无法决定正确的工作方针与工作计划来实现自己的理想。

当然了解具体情况不是一件容易的事。但是，一切具体情况都是在矛盾中发展着的，都是依照普遍的辩证法的规律变动着的，所以只要我们能够把握住这些方法，那比较正确地去了解具体情况也不是不可能的。问题只是要我们的青年在接触具体情况的时候，能够更多地“想一想”，更多地研究一下，考察一

下，决不要以自己的一些片面的主观的了解为满足。世界上的一切都是非常复杂的，不是马马虎虎、冒冒失失就能够了解的。如果我们没有这种了解具体情况的要求与忍耐心，那我可以告诉大家，你们是什么事也做不好的。

当然，这并不是说我们一定要彻底了解具体情况之后才能开始工作。这种彻底了解，在一个短的时期内是不可能的。但我们要在开始工作之前，有一种初步的与起码的了解，不然，到一个地方不管一切乱干一气，那是异常危险的。

第二，在我们对于具体情况有了初步的与起码的了解之后，我们就要根据对具体情况的了解来决定我们的方针，我们的任务，斗争的形式，工作的方法方式等。

比如你们现在到上海工作，你们在解到了上海的具体情况之后，你们就必须规定你们今天在上海需要做什么，能够做什么，而且如何做法。显然地，在上海日寇军队占领的地区，我们今天的具体任务、斗争形式、工作方法方式等，决不能同其他区域相同。你们要老虑，究竟今天在上海是同在其他某些战区一样地去发展游击战争，进行公开的反日的活动，或者是这里需要执行另一种任务，即积聚自己力量准备待时而动的任务，采取另一种斗争形式，即秘密的反日斗争的活动呢？是组织公开的抗敌后援会，还是一方面组织秘密的抗日团体而同时去利用公开的合法的组织形式呢？只有正确地解决这些问题，我们才能使我们的力量巩固与发展，达到最后配合全国力量战胜日寇的理想。如果我们的任务规定得不正确，工作方法方式等都规定得不对，那我们必然会遭受严重的失败，而达不到我们的目的地。

在目前全中国人民的前面的最中心的任务是坚持抗战，最后战胜日寇。但是实现这个中心任务的方法是在各个具体的地方都不能完全相同的。我们青年要善于根据不同地区的具体情况，来决定实现这个中心任务的具体办法。由于中国政治经济发展的不平衡，由于中国的地大物博，所以实现目前中心任务的方法，都不能有千篇一律的公式。公式主义常常是我们实现总任务中的最大的障碍物。

就是各地方的具体情况，也不是固定不变的。它同样地变动着。所以你们就在一个地方的工作中仍然要时时刻刻注意新的情况的变化，而及时地改变自己的任务及工作方法方式等。至死不变的东西是没有的。这里，要求我们以极大的灵活性、机动性与创造性，来实现自己的理想。固执自己的死公式或老办法，结果必然会遭受严重的失败，而达不到目的地。

第三，在任务等规定之后，就应该立刻开始自己的实际工作。

我们是革命家，行动家，而不是空谈家。“议而不决、决而不行”的恶习惯，我们青年是决不应该学习的。我们必须把我们自己所决定的方针与计划来见诸实行。只有革命的行动才能改造世界。革命的伟人马克思曾经说过，我们的任务不但在认识世界，而主要的在改造世界。要改造世界就要有革命的实际工作，而且也只有在实际工作中，我们才更能进一步地认识世界，更正确地来决定自己的任务与工作方法方式等。也只有实际工作，能够考验我们过去的决定是否正确，充实与发展我们的决定，改正错误等。

所以我们青年在实际工作中也仍然不是盲目地乱干一顿，而要时时刻刻注意自己在实际工作中所碰到的一切问题，总结自己在实际工作中的经验。如果在实际工作中证明过去自己的决定是不正确的，是做不通的，那自己应该毫无怜惜地抛弃过去自己的决定，而根据自己对于具体情况的新的了解来重新决定自己的任务。如果有部分的错误，即应改正部分的错误。如果有不够的地方，即应以新的经验使之充实，使之发展。这里所需要的是虚心的服从真理的态度，而不是“自作聪明”去蛮干。不顾事实的蛮干，是会碰到钉子以至毁灭自己的。

所以我们的青年，应该时时刻刻在实际工作中实行工作的检查，发展自我批评，总结工作中的经验与教训，使我们的工作能够更好地得到进步与成绩。一切先进的革命的理论，我们也只能当作行动的指南而不能当作教条。一切先进的革命理论，也要在实践中充实自己与发展自己的。

我想，这三点是我们青年人在工作中所应该注意的。这三点的中心所在，即是要使我们的青年不但要有理想，有实现理想的决心，而且要学习实现理想的具体办法。

当然，青年人终究是青年人，他们一般是富于革命热情而缺乏实际经验，因此他们有时要“感情用事”，有点“冲动”，有点“幼稚病”。我们指出青年的这些弱点，并不想以此来责备青年，而只是要求青年们在自己的工作中注意到自己的这些弱点，克服这些弱点，使自己逐渐地在实际工作中成为一个比较“能干的”与“老练的”革命青年。

四、要同群众一起去实现自己的理想

我们在前面已经讲过，要实现革命的理想，一定要依靠于一定的社会的力量，这社会的力量就是千千万万的群众。历史上的伟人，固然在创造人类历史中起了很大的作用，然而如果没有群众的拥护与群众的行动，任何推动历史前进的理想都是不可能实现的。所以我们青年要实现自己的理想，就必须能够率领群众去为自己的理想奋斗。

青年在革命行动中肯负责，肯出头，肯打先锋，这是很可宝贵的品质。在中国民族解放的历史中，青年常常起着先锋的作用，就是由于青年这种优良品质的结果。但是就在这个方面，部分青年仍然表现出他们的弱点。他们肯负责，肯出头，肯打先锋的品质，往往发展到好出风头，目空一切，自高自大，包办，不耐烦，脱离群众的偏向。这种例子，我是见得很多的，不知道你们见过没有？我想一定是见过的。

在陕北公学学习的青年，一般地是觉悟程度较高的一部分中国人，他们的确有责任去领导另一部分比较落后而又占最大多数的中国人，发动他们，教育他们，训练他们，以提高他们的政治水平与组织力量。没有先锋的领导，广大的比较落后的群众，是不会自己走到先锋的地位的，这是无可否认的真理。

然而怎样可以使我们陕公的青年能够负担起这个先锋的领导的责任呢？是不是我们青年装出一个领导者的架子，自命不凡，目空一切，就能够实现这个领导作用呢？是不是只要我们的青年把我们的理想在群众前面一解释一宣布，群众就会跟了我们走，就实现了我们的领导作用呢？这些都是不能成功的。任何领导者，如果自命不凡，目空一切，装出领导者的架子，那结果他不但不能领导群众，而且会脱离群众。至于把很美丽的理想讲给群众听，群众自然可能来听一下，然而要他们为了这个理想来牺牲奋斗，那单靠这种宣传工作是不能达到目的的。

领导群众，是一件非常不容易的事。青年人决不要把这个问题看得太简单了。这里，第一个问题，就是要求**我们的青年到群众中去**。我们的青年对于“群众”两个字的观念常常是比较抽象的，有时把“群众”理想化，有时看不起“群众”，而不能活生生地去了解“群众”。所以下决心到群众中去，实是开始领导群众时的必要的一步。

我们青年到了群众中之后，就要学习如何去接近群众，去同他们生活在一起，去了解他们，以至取得他们的信任。这里，首先就是要时时刻刻去为群众服务，处处能够为他们谋利益，为他们的利益牺牲自己的一切。其次，要我们处处谦逊和气，刻苦耐劳，宽宏大量，急公好义，诲人不倦，做人家的模范与教师。第三，要能够团结与培养群众中的积极分子，发挥积极分子的作用。依靠积极分子推动全体。第四，要以民主的精神与民主的工作方法来吸收与教育群众积极参加工作，不要个人包办一切。第五，要善于孤立最顽固的少数坏人，依靠群众的公愤去打击他们与驱逐他们。只要这样。群众就会把我们当作他们“自己的人”，就会信任我们，接受我们的领导。

无疑的，当我们真正深入到群众中去时，我们必然会看到群众中有许多落后意识，落后的思想与习惯，他们的狭隘的宗法的与行会的观点。这一切我们都会碰到的。然而我们是不是因此就失望而脱离他们呢？决不是的。我们的责任，正是要提高他们的政治水平，克服他们的落后性。但是当我们认真在群众中工作时，我们更会碰到群众中的光明的一面，他们的丰富的革命性与他们的伟大的革命力量。我们在他们的深处，可以看到在旧社会的上层所没有的那种德性。他们的力量，他们的德性，可以给我们以最大的信心，使我们相信他们是我们的最伟大理想的负担者，而且他们最后一定能胜利。这就使我们在他们中间感觉到一种崇高的安慰与喜悦，这就使我们在任何困难的情形之下，总是不会脱离他们，而且总是要依靠他们的力量来克服一切困难。事实已经证明，正在证明，将要证明，一切我们当前的困难，只要我们真能发动千百万群众起来，并且依靠他们的力量，就是一定能够克服的。一切前进的革命者，是决不应该惧怕群众的，相反的，他们必须依靠群众。

领导群众的第二个问题，就是要善于使群众根据自身的经验来了解我们的领导的正确。比如抗日民族统一战线的方针，开始时不是大多数中国人都能了解与拥护的。当时我们用什么口号使群众根据自身的经验来了解我们的方针的正确呢？这就是“停止内战，一致抗日”的口号。每一个有良心的中国人，对于中国内战的痛苦都是切身感受到的，他们都能够懂得日寇的无限制的侵掠同内战是不能分离的。因此他们也许还不了解抗日民族统一战线的方针，然而对于“停止内战，一致抗日”的口号是热烈拥护的。结果，内战停止了，中国开始统一了，抗日民族统一战线得到了初步的成功。不久卢沟桥事变发生了。当时我们用什么口号使群众根据自身的经验来了解我们的方针的正确呢？这就是“为保卫平津、保卫华北、保卫全中国而战”。每一个有良心的中国人，对于日寇的这种无限制的侵掠已经到了不能再行忍耐的地步，他们都要求抵抗。全国抗战开始了，于是抗日民族统一战线有了进一步的成功。在抗战发动之后，我们用什么口号使群众根据切身的经验来了解我们的方针的正确呢？这就是“坚持抗战、争取抗战最后胜利”的口号。每一个有良心的中国人，对于这个口号是能够懂得的，因为谁都可以看到，如果中国今天半途投降妥协，那中国就会亡国，中国人就会变为亡国奴，而亡国奴是不好当的。于是抗日民族统一战线有了更进一步的成功。

这不过是举一个例子罢了。不但在提出口号上是如此，即在斗争形式的采用上也是如此。**我们要使自己不脱离群众，必须采取那种能够吸引群众参加的斗争形式，他们认为今天可以而且需要的斗争形式，以锻炼他们，提高他们的觉悟程度与组织力量。**在斗争形式上是如此，在其他问题上也莫不如此。

不估计群众今天觉悟的程度，而提出很高很“左”的口号，是决然不能动员群众参加运动的。这里需要采取各种过渡的办法来动员他们，再在运动中提高他们走向我们的理想。这里，一切急性病是有害的；这里，领导者的努力工作的精神，要同极大的忍耐心配合起来。群众的觉悟的速度，不但依靠于我们的工作，而且依靠于无数客观的政治经济的要素。革命是决不能凭主观的愿望来制造的。在没有革命形势的时候，特别在反动的黑暗的时期，群众觉悟的速度非常缓慢。那时我们如果没有等待时机的忍耐心，而想用强迫命令的方法去对待群众，那结果是必然会失败的。所谓等待时机，当然不是不要努力工作，而是把我们的工作放在准备必要的力量上，以便将来革命形势到来时，群众成千成万卷入革命浪潮中时，我们能够站在浪潮的前面负担起领导者的责任。

在群众工作的命令主义，结果没有不遭受失败的。群众在某种压力之下，可能勉强地服从，然而一切不根据群众自觉的服从，违反于他们意志的服从，最后必然会引起他们反抗并推翻那些压迫者或命令主义者。只有群众自觉的自愿的服从，才能发挥群众的积极性与创造力量到最高限度，来实现我们远大的理想。所以我们领导群众的中心问题，也就是如何使群众心悦诚服地来接受我们的领导，并且为了我们的理想的实现牺牲一切。这就要求我们在领导工作中善于使群众能够根据自身的经验来了解我们领导的正确。这里，应该反对那些光会喊革命口号、唱高调的空谈主义者，与企图以强迫命令的方法来“驱使群众为己用”的老爷们！

但是，当革命形势迅速发展、群众觉醒的速度一日千里的时候，我们要谨防自己落后于群众，变为群众的尾巴。在这种情况下，我们仍然应该站在群众的前面，组织群众的积极性向着一定的目标前进。我们要善于迅速地改变过时的口号，提出新的更高的动员口号，采取大刀阔斧的办法来动员、组织与武装群众，推动群众斗争走向最高的形式。比如在目前被敌人占领区域的广大乡村与战区的情况，就是如此。群众抗日的斗争往往一开始就采取武装斗争的形式。在这里，几天内群众所达到的觉醒程度，可以比上过去的几十年。旧的口号、旧的斗争方式与工作方法等，决不能满足群众今天的要求。群众的民族革命的高潮，会冲开一切这些陈腐的东西，找到他们的新的领导者，走向他们的目的地。

领导群众的第三个问题，就是要向群众学习。**一切革命的经验，都是从群众的实际斗争中创造出来的。**离开群众的革命实践，我们就不能有革命的理论。而群众的革命实践是长生的，是无穷的，是永远继续前进着的。它发展与充实我们的理论，它改正我们的错误，它使我们一天一天接近着绝对的真理。所以我们必须在群众斗争中去学习，在这个人生的大海中去学习。我们的学习是没有止境的。群众的革命实践不断创造出极可宝贵斗争形式、组织形式、工作方法与方式以及各种极可宝贵的经验。这些群众创造出来的东西，常常是一个领导者，即使是天才的领导者，也不能预料到的。有时一个领导者常常苦于找不到一个适当的斗争形式或组织形式以进一步开展群众运动。但是群众的革命实践常常创造了这种适

合的形式。群众对于一个问题所发表的意见，他们对于一个问题的看法与感觉，常常是一个领导者，即使是天才的领导者，也想不到的与感觉不到的。而正是他们的意见，他们的看法与感觉，可以补足一个领导者对于某个问题的理解之不足与缺陷，使某个问题得到圆满的正确的解决。

所以，群众不但需要我们去教育，而且他们也教育我们。我们的青年切不要在群众前面摆资格，自高自大，瞧不起群众，而应该很虚心地去跟他们学习，很细心地去倾听他们的意见，与他们的脉搏一起跳动。

总之，我们在领导群众中的基本原则，是无论如何要领导群众前进，而同时无论如何不要脱离群众。脱离群众的先锋主义与落后于群众的尾巴主义，都是我们所不应该赞成的。

同志们，要把这样广大的、千千万万的、觉悟程度不同的群众动员到为我们的理想而牺牲奋斗的一条战线上，真不是简单的事。要达到这个目的，你们无论如何不要脱离群众，而要成为他们的领袖。

同志们！这就是我今天想同同志们讨论的关于青年的修养的四个问题。

同志们！我前面已经说过，革命是一件最伟人的事业，也是最困难的事业。我们在革命中犯些错误，甚至是**非常严重的错误**，是免不了的。至于青年朋友们，由于生活经验与斗争经验的缺乏，犯些错误更不是什么不得了的问题。我们决不能因为怕犯错误，就不干革命。相反的，我们就要在错误中学习。

我们青年人切不要以自己已经有的一点知识与一点经验，甚至一些生吞活剥的革命的公式与口号为满足。这对于青年们是最大的危险，因为这种满足就阻碍了你们的进步，结果必然会使你们成为无用的“空头革命家”。青年们必须以列宁的“**学习学习再学习**”的口号，当作自己的座右铭。你们今天在陕北公学学习，明天就要到抗战的前线与后方的实际工作中去学习。青年应该有最热烈的学习愿望，很高的学习精神与谦逊的学习态度。只有这种不断地学习，才能丰富你们的知识与经验，才能使你们成为一个能干的革命者，才能使你们更能负担起在自己肩膀上所负担的责任。

我重复地说，工作中的错误是谁也免不了的。我们就要在错误中学习。任何错误，不论为它曾经付了如何高昂的代价，只要我们从错误中得了宝贵的经验，那这种代价也是不算可惜的。所以，我们用不着怕犯错误。但犯了错误，我们应该知道很快地去改正错误，切不要坚持错误。任何错误，如果坚持下去，那是非常危险的。青年人应该打破“爱好面子”、不肯或者害怕承认错误的恶劣倾向。青年人应该“爱好真理”。一切错误的非真理的东西，都应该决然抛弃，丝毫也不应该留恋。只有这样，才能使你们不断地进步。

所以，同志们，你们更努力地去学习革命理论吧，更大胆地去工作奋斗吧，在学习的过程中，在工作奋斗的过程中，去锻炼你们自己成为钢铁一样的战士吧！要不怕困难，不怕挫折与失败，要不怕天不怕地，要再接再厉地不屈不挠地向着你们光明的伟大的理想前进！

丢掉幻想，准备斗争

“说来也是笑话，我读过小学、中学，也当过兵，却未曾看见过世界地图，因此就不知道世界有多大。湖南省图书馆的墙壁上，挂有一张世界大地图，我每天经过那里，总是站着看一看。

过去我认为湘潭县大，湖南省更大，中国自古就称为天下，当然大得了不得。但从这个地图上看来，中国只占世界的一小部分，湖南省更小，湘潭县在地图上没有看见，韶山当然更没有影子了。世界原来有这么大！

世界既大，人就一定特别多。这样多的人怎样过生活，难道不值得我们注意吗？从韶山冲的情形来看，那里的人大都过着痛苦的生活，不是挨饿，就是挨冻。有无钱治病看着病死的；还有家庭里、乡邻间，为着大大小小的纠纷，吵嘴、打架，闹得鸡犬不宁，甚至弄得投塘、吊颈的；至于没有书读，做一世睁眼瞎子的就更多了。在韶山冲里，我就没有看见几个生活过得快活的人。韶山冲的情形是这样，全湘潭县、全湖南省、全中国、全世界的情形，恐怕也差不多！

我真怀疑，人生在世间，难道都注定要过痛苦的生活吗？决不！为什么会有这种现象呢？

这是制度不好，政治不好，是因为世界上存在人剥削人、人压迫人的制度，所以使世界大多数的人都陷入痛苦的深潭。这种不合理的现象，是不应该永远存在的，是应该彻底推翻、彻底改造的！总有一天，世界会起变化，一切痛苦的人，都会变成快活的人！幸福的人！

世界的变化，不会自己发生，必须通过革命，通过人的努力。我因此想到，我们青年的责任真是重大，我们应该做的事情真多，要走的道路真长。

从这时候起，我就决心要为全中国痛苦的人、全世界痛苦的人贡献自己全部的力量。”

我是个学生出身的人，那时，我觉得世界上干净的人只有知识分子，工人农民总是比较脏的。知识分子的衣服，别人的我可以穿，以为是干净的；工人农民的衣服，我就不愿意穿，以为是脏的。革命了，同工人农民和革命军的战士在一起了，我逐渐熟悉他们，他们也逐渐熟悉了我。这时，只是在这时，我才根本地改变了资产阶级学校所教给我的那种**资产阶级的和小资产阶级的感情**。这时，拿未曾改造的知识分子和工人农民比较，就觉得知识分子不干净了，最干净的还是工人农民，尽管他们手是黑的，脚上有牛屎，还是比资产阶级和小资产阶级知识分子都干净。

“一个人；如若不被敌人反对，那就不好了，那一定是同敌人同流合污了。如若被敌人反对，那就好了，那就证明我们同敌人划清界限了。如若敌人起劲地反对我们，把我们说得一塌糊涂，一无是处，那就更好了，那就证明我们不但同敌人划清了界限，而且证明我们的工作是卓有成效的了。”

与天奋斗，其乐无穷；

与地奋斗，其乐无穷；

与人奋斗，其乐无穷。

“让内外反动派在我们的面前发抖吧！让他们去说我们这也不行那也不行吧！中国人民不屈不挠的努力，必将稳步地达到自己的目的！”

“仗我们是不怕打的，帝国主义要想‘和平演变’我们这一代人也难；可下一代、再下一代就不好讲了。中国人讲‘君子之泽，五世而斩’，英国人说‘爵位不传三代’；到我们的第三代、第四代人身上，情形又会是个什么样子啊？我不想哪一天，在中国的土地上再出现人剥削人的现象，再出现资本家、企业主、雇工、妓女和吸食鸦片烟；如果那样，许多烈士的血就白流了……”

创造强大的人民空军，歼灭残敌，巩固国防。

1. 指点江山，激扬文字，粪土当年万户侯。
2. 要似昆仑崩绝壁，又恰像台风扫寰宇。重比翼，和云翥。
3. 军叫工农革命，旗号镰刀斧头。匡庐一带不停留，要向潇湘直进。
4. 雄关漫道真如铁，而今迈步从头越
5. 红军不怕远征难，万水千山只等闲
6. 安得倚天抽宝剑，把汝裁为三截？一截遗欧，一截赠美，一截还东国。太平世界，环球同此凉热。
7. 浴血东瓜守，驱倭棠吉归。沙场竟殒命，壮志也无违。
8. 遍地哀鸿遍地血，无非一念救苍生。
9. 宜将剩勇追穷寇，不可沽名学霸王。

青年在选择职业时的考虑

卡·马克思的中学考试德语作文

自然本身给动物规定了它应该遵守的活动范围，动物也就安分地在这个范围内活动，而不试图越出这个范围，甚至不考虑有其它范围存在。神也给人指定了共同的目标——使人类和他自己趋于高尚，但是，神要人自己去寻找可以达到这个目标的手段；神让人在社会上选择一个最适合于他、最能使他和社会变得高尚的地位。

这种选择是人比其它创造物远为优越的地方，但同时也是可能毁灭人的一生、破坏他的一切计划并使他陷于不幸的行为。因此，认真地权衡这种选择，无疑是开始走上生活道路而又不愿在最重要的事情上听天由命的青年的首要责任。

每个人眼前都有一个目标，这个目标至少在他本人看来是伟大的，而且如果最深刻的信念，即内心深处的声音，认为这个目标是伟大的，那它实际上也是伟大的，因为神决不会使世人完全没有引导者；神轻声地但坚定地作启示。

但是，这声音很容易被淹没；我们认为是热情的东西可能倏忽而生，同样可能倏忽而逝。也许，我们的幻想蓦然迸发，我们的感情激动起来，我们的眼前浮想联翩，我们狂热地追求我们认为是神本身给我们指出的目标；但是，我们梦寐以求的东西很快就使我们厌恶，于是，我们便感到自己的整个存在遭到了毁灭。

因此，我们应当认真考虑：我们对所选择的职业是不是真的怀有热情？发自我们内心的声音是不是同意选择这种职业？我们的热情是不是一种迷误？我们认为是神的召唤的东西是不是一种自我欺骗？不过，如果不对热情的来源本身加以探究，我们又怎么能认清这一切呢？

伟大的东西是闪光的，闪光会激发虚荣心，虚荣心容易使人产生热情或者一种我们觉得是热情的东西；但是，被名利迷住了心窍的人，理性是无法加以约束的，于是他一头栽进那不可抗拒的欲念召唤他去的地方；他的职业已经不再是由他自己选择，而是由偶然机会和假象去决定了。

我们的使命决不是求得一个最足以炫耀的职业，因为它不是那种可能由我们长期从事，但始终不会使我们感到厌倦、始终不会使我们劲头低落、始终不会使我们的热情冷却的职业，相反，我们很快就会觉得，我们的愿望没有得到满足，我们的理想没有实现，我们就将怨天尤人。

但是，不仅虚荣心能够引起对某种职业的突然的热情，而且我们也许会用自己的幻想把这种职业美化，把它美化成生活所能提供的至高无上的东西。我们没有仔细分析它，没有衡量它的全部分量，即它加在我们肩上的重大责任；我们只是从远处观察它，而从远处观察是靠不住的。

在这里，我们自己的理性不能给我们充当顾问，因为当它被感情欺骗，受幻想蒙蔽时，它既不依靠经验，也不依靠更深入的观察。然而，我们的目光应该投向谁呢？当我们丧失理性的时候，谁来支持我们呢？

是我们的父母，他们走过了漫长的生活道路，饱尝了人世辛酸。——我们的心这样提醒我们。

如果我们经过冷静的考察，认清了所选择的职业的全部分量，了解它的困难以后，仍然对它充满热情，仍然爱它，觉得自己适合于它，那时我们就可以选择它，那时我们既不会受热情的欺骗，也不会仓促从事。

但是，我们并不总是能够选择我们自认为适合的职业；我们在社会上的关系，还在我们有能力决定它们以前就已经在某种程度上开始确立了。

我们的体质常常威胁我们，可是任何人也不敢藐视它的权利。

诚然，我们能够超越体质的限制，但这么一来，我们也就垮得更快；在这种情况下，我们就是冒险把大厦建筑在残破的废墟上，我们的一生也就变成一场精神原则和肉体原则之间的不幸的斗争。但是，一个不能克服自身相互斗争的因素的人，又怎能抗御生活的猛烈冲击，怎能安静地从事活动呢？然而只有从安静中才能产生出伟大壮丽的事业，安静是唯一能生长出成熟果实的土壤。

尽管我们由于体质不适合我们的职业，不能持久地工作，而且很少能够愉快地工作，但是，为了克尽职守而牺牲自己幸福的思想激励着我们不顾体弱去努力工作。如果我们选择了力不胜任的职业，那么我们决不能把它做好，我们很快就会自愧无能，就会感到自己是无用的人，是不能完成自己使命的社会成员。由此产生的最自然的结果就是自卑。还有比这更痛苦的感情吗？还有比这更难于靠外界的各种赐予来补偿的感情吗？自卑是一条毒蛇，它无尽无休地搅扰、啃啮我们的胸膛，吮吸我们心中滋润生命的血液，注入厌世和绝望的毒液。

如果我们错误地估计了自己的能力，以为能够胜任经过较为仔细的考虑而选定的职业，那么这种错误将使我们受到惩罚。即使不受到外界的指责，我们也会感到比外界指责更为可怕的痛苦。

如果我们把这一切都考虑过了，如果我们的生活条件容许我们选择任何一种职业，那么我们就可以选择一种使我们获得最高尊严的职业，一种建立在我们深信其正确的思想上的职业，一种能给我们提供最广阔的场所来为人类工作，并使我们自己不断接近共同目标即臻于完美境界的职业，而对于这个共同目标来说，任何职业都只不过是一种手段。

尊严是最能使人高尚、使他的活动和他的一切努力具有更加崇高品质的东西，是使他无可非议、受到众人钦佩并高出与众人之上的东西。

但是，能给人以尊严的只有这样的职业，在从事这种职业时我们不是作为奴隶般的工具，而是在自己的领域内独立地进行创造；这种职业不需要有不体面的行动（哪怕只是表面上不体面的行动），甚至最优秀的人物也会怀着崇高的自豪感去从事它。最合乎这些要求的职业，并不总是最高的职业，但往往是最可取的职业。

但是，正如有失尊严的职业会贬低我们一样，那种建立在我们后来认为是错误的思想上的职业也一定会成为我们的沉重负担。

这里，我们除了自我欺骗，别无解救办法，而让人自我欺骗的解救办法是多么令人失望啊！

那些主要不是干预生活本身，而是从事抽象真理的研究的职业，对于还没有确立坚定的原则和牢固的、不可动摇的信念的青年是最危险的，当然，如果这些职业在我们心里深深地扎下了根，如果我们能够为它们的主导思想而牺牲生命、竭尽全力，这些职业看来还是最高尚的。

这些职业能够使具有合适才干的人幸福，但是也会使那些不经考虑、凭一时冲动而贸然从事的人毁灭。

相反，重视作为我们职业的基础的思想，会使我们在社会上占有较高的地位，提高我们自己的尊严，使我们的行为不可动摇。

一个选择了自己所珍视的职业的人，一想到他可能不称职时就会战战兢兢——这种人单是因为他在社会上所处的地位是高尚的，他也就会使自己的行为保持高尚。

在选择职业时，我们应该遵守的主要指针是人类的幸福和我们自身的完美。不应认为，这两种利益会彼此敌对、互相冲突，一种利益必定消灭另一种利益；相反，人的本性是这样的：人只有为同时代人的完美、为他们的幸福而工作，自己才能达到完美。

如果一个人只为自己劳动，他也许能够成为著名的学者、伟大的哲人、卓越的诗人，然而他永远不能成为完美的、真正伟大的人物。

历史把那些为共同目标工作因而自己变得高尚的人称为最伟大的人物；经验赞美那些为大多数人带来幸福的人是最幸福的人；宗教本身也教诲我们，人人敬仰的典范，就曾为人类而牺牲自己——有谁敢否定这类教诲呢？

如果我们选择了最能为人类而工作的职业，那么，重担就不能把我们压倒，因为这是为大家作出的牺牲；那时我们所享受的就不是可怜的、有限的、自私的乐趣，我们的幸福将属于千百万人，我们的事业——将悄然无声地存在下去，但是它会永远发挥作用，而面对我们的骨灰，高尚的人们将洒下热泪。

卡·马克思写于1835年8月