

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	1.5.7.5.6
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_coalition_formation_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
LearnLalpop	1.5.12
Lalpop	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
Hypaland	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_philosiphy	1.12.3
philosiphy	1.12.4
- -1948	1.12.5
-	1.12.6
cp0	1.12.6.1
cp1	1.12.6.2
cp2	1.12.6.3
index	1.12.6.4
- -1946	1.12.7
- -1938	1.12.8
-	1.12.9
- -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

- 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.

- 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
```

- 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

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

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

- 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-libsecret
```

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:

1. “ ”
2.
3.
4. or /
5.
6. enjoy your life
7.
8.
9.
10. /
11.
12.
13.
14.
15.
16.
17.
18.
19.
20.
21.

•
•
◦
◦
◦
•
◦ ...
•

Learn Notes

GPU to CPU

MocCUDA

MocCUDA

High-Performance GPU-to-CPU Transpilation and Optimization via High-Level Parallel Constructs

GPU CPU

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 28
ACM SIGPLAN

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.

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% geometric 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 $2.7\times$.

	CUDA	Polygeist/MLIR	CPU	CUDA Rodinia	CPU	Fugaku	GPU
OpenMP	76%		CPU	CUDA	CPU		
PyTorch		PyTorch	CUDA	PyTorch	CPU		

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.

CPU

AST	“ ”						
-----	-----	--	--	--	--	--	--

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.

MLIR	GPU	[22]	Polygeist [23] C/C++	LLVM	CUDA	MLIR	[21]	MLIR	[20]
PyTorch	CUDA			LLVM	CPU				
OpenMP	CPU		MocCUDA PyTorch		GPU			CUDA	

Overall, our paper makes the following contributions:

- A common high-level and platform-agnostic representation of SIMD-style parallelism backed by a semantic definition of barrier synchronization that ensures correctness through memory semantics, which ensures transparent application of existing optimizations.
- SIMD
- Novel parallel-specific optimizations which can exploit our high-level parallel semantics to optimize programs.
-

- 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 Polygeist C/C++ GPU CPU
- 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.
- CUDA CPU Rodinia [24] PyTorch [2] CUDA CPU Fugaku
- Resnet-50

II-CPolygeist II-C

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 Clang [23] MLIR C C++

C++	MLIR	Polygeist	MLIR SCF
MLIR	(memref)	Polygeist	[28]

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.

GPU	(3):	
• 3D for		
• ,		
• 3D for		
• CUDA / ROCm	Polygeist	

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.

OpenCL	GPU	Khronos	OpenCL
OpenCL	CUDA		
TRSM	GEMM	3 BLAS	TRSM
NVIDIA Tesla C2050	ATI Radeon 5870	GPU	OpenCL
OpenCL	CUDA	GEMM	OpenCL
GPU GEMM	50%		

Kokkos: Enabling manycore performance portability through polymorphic memory access patterns

Kokkos

Achieving Portability and Performance through OpenACC

OpenACC

<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

Fortran C C++



GPU(CUDA) to CPU

GPUOcelot

[GPUOcelot](#)

GPUOcelot: A dynamic compilation framework for PTX

GPUOcelot PTX

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

Ocelot

NVIDIA

PTX

CUDA

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

NVIDIA CUDA

HIP C++

HIP

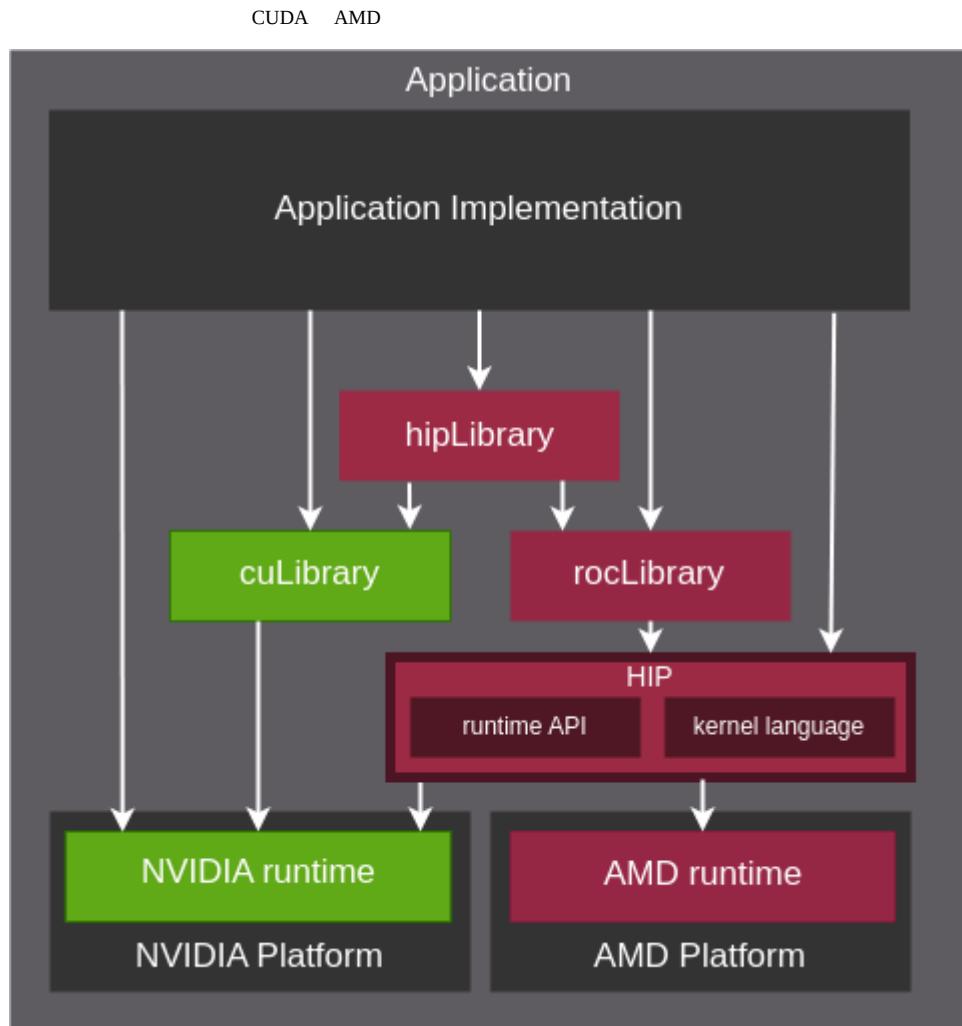
[HIP: C++ Heterogeneous-Compute Interface for Portability](#)

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 C++ API AMD NVIDIA GPU

Key features include:

- HIP is very thin and has little or no performance impact over coding directly in CUDA mode.
- HIP allows coding in a single-source C++ programming language including features such as templates, C++11 lambdas, classes, namespaces, and more.
- HIP allows developers to use the "best" development environment and tools on each target platform.
- The [HIPIFY](#) tools automatically convert source from CUDA to HIP.
- Developers can specialize for the platform (CUDA or AMD) to tune for performance or handle tricky cases.



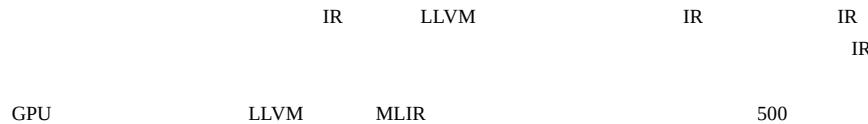
Domain-Specific Multi-Level IR Rewriting for GPU: The Open Earth Compiler for GPU-accelerated Climate Simulation

IR GPU GPU

<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.



12

- -
 -
 - = + MEU
 -
 -
 -
 -
 -
 -
 - wumpus
-

- (33 / 81) - (zhihu.com)

, , 3

1. 0 1 , \$0 \leq P(X=x_i) \leq 1\$, \$x_i\$ X
2. 1 ,

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

1. ,

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

\$x_1\$ \$x_2\$

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

“ ”

“ ”

/

/

1. /
- 2.

or

/ /

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

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

- X
- E e
- Y

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

$$\begin{aligned} & \$\boldsymbol{y} \$ (& \$\boldsymbol{Y} \$) & \$\boldsymbol{X} \$ \$\boldsymbol{E} \$ \\ \$\boldsymbol{Y} \$, & \$\boldsymbol{P}(X, \mathbf{e}, \mathbf{y}) \$ \\ -\$alpha\$ \end{aligned}$$

Independence

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

“ ”

a b a b

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

X Y

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

- “ ”
-
-

pairwise independent

$$\{\mathbf{A}_i\}_{i=1}^n$$

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

mutually independent

$$\{\mathbf{A}_i\}_{i=1}^n$$

$$\mathbf{A}_1, \dots, \mathbf{A}_n$$

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

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

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

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

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

- vs
-
-
-

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

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

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

13

- 13.1
- 13.2
 - 13.2.1
 - 13.2.2
 - 13.2.3
 - 13.2.4
- 13.3
 - 13.3.1
 - 13.3.2
 - 13.3.3
 - 13.3.4
- 13.4
 - 13.4.1
 - 13.4.2
 - 13.4.3
- 13.5
 - 13.5.1 do
 - 13.5.2

•
•
•
•
•
•
•
•

Bayesian Network

1. AIMA

$\$\\theta\$$

$$P(x_1, \dots, x_n) = \prod_{i=1}^n \theta(x_i \mid \text{parents}(X_i))$$

$\$\\theta\$ \quad " \quad \$\\theta\$$

$$P(x_i \mid \text{parents}(X_i)) \equiv \frac{P(x_i, \text{parents}(X_i))}{P(\text{parents}(X_i))}$$

$$= \frac{\sum_{\mathbf{y}} P(x_i, \text{parents}(X_i), \mathbf{y})}{\sum_{x'_i, \mathbf{y}} P(x'_i, \text{parents}(X_i), \mathbf{y})}$$

$$= \theta(x_i \mid \text{parents}(X_i))$$

" "

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i \mid \text{parents}(X_i))$$

" " "

2. $\$\\theta\\left(x\\{i\\} \\mid right. parents\\left.\\left(x\\{i\\}\\right)\\right)\\$$

$\$P\\left(x\\{i\\} \\mid right. parents\\left.\\left(x\\{i\\}\\right)\\right)\\$$

$$P(x_i \mid \text{parents}(X_i)) = \theta(x_i \mid \text{parents}(X_i))$$

" " " "

+

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i \mid \text{parents}(X_i))$$

" " "

" " "

—

" " " " "

" " "

+

=

Belief Network

- DAG
- CPT

$\$B\$ \quad \$G\$ \quad \$\\Theta\$ \quad , \quad \$B=\\langle G, \\Theta\\rangle\$$

- $\$G\$$
- $\$\Theta\$$ $\$x_i \$ \$G\$$ $\$p_i \$ \$\Theta\$$ $\$theta{x_i} \mid$
 $\pi_i = P(B) \left(x_i \mid p_i \right)$

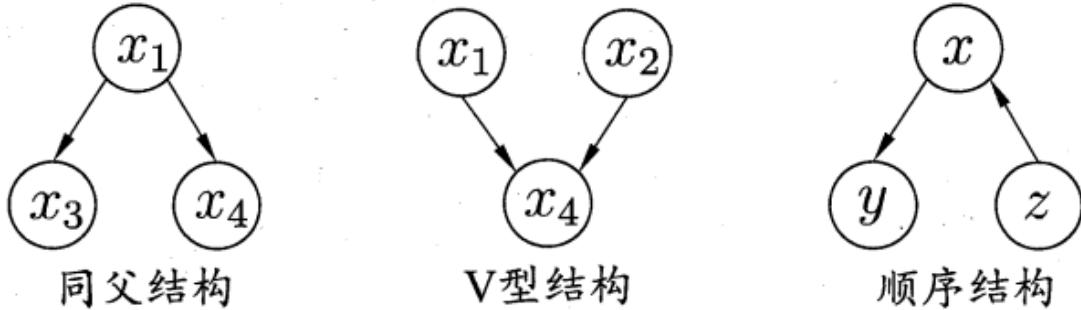


图 7.3 贝叶斯网中三个变量之间的典型依赖关系

- /
- $\$x_1 \$ \$x_3 \$ \$x_4 \$$
-
- /
- $\$x \$ \$y \$ \$z \$$
-
- V / /
- $\$x_4 \$ \$x_1 \$ \$x_2 \$$
- $\$x_4 \$ \$x_1 \$ \$x_2 \$$ “ ” Marginal Independence
- $\$x_4 \$$

D-Speration

- $\$x \$ \$y \$ \$mathbf{Z} \$ \$x \$ \$y \$ \$mathbf{Z} \$ \$x \perp\!\!\!\perp y \mid \mathbf{Z}$
1. Ancestral Subgraph $\mathbf{X} \$ \mathbf{Y} \$ \mathbf{Z}$
 2. DAG
 - DAG V V
 - PS “ ” - “ ”
 3. $\mathbf{Z} \$ \$x \$ \$y \$ \$mathbf{Z} \$ \$x \$ \$y \$$
 4. $\$x \$ \$y \$ \$mathbf{Z} \$ \$x \$ \$y \$ \$mathbf{Z} \$$

MLBOP

- $\$x_1, \dots, x_n \$ \$mathbf{Z} \$$
- V $\$x_{i-1} \rightarrow x_i \leftarrow x_{i+1} \$ \$x_i \$ \$x_j \$ \$mathbf{Z} \$$
 - $\$mathbf{Z} \$ \$mathbf{Z} \$ \$x_1 \$ \$x_n \$$
- B $\mathbf{X} \$ \mathbf{Y} \$ \mathbf{Z} \$ B$ $\$mathbf{Z} \$ \$mathbf{X} \$$
- $\$mathbf{Y} \$ \$forall x \in \mathbf{X}, y \in \mathbf{Y}, \sim(x, y) \$ \$mathbf{Z} \$$ $\$mathbf{X} \$ \$mathbf{Y} \$ \$mathbf{Z} \$$

AIMA² $\$X_i\$$ $\$\theta(x_i \mid \text{parents}(x_i))\$$

“ ”

$$P(x_1, \dots, x_n) = \prod_{i=1}^n \theta(x_i \mid \text{parents}(x_i))$$

“ ”/

 $\$P(x_i \mid \text{parents}(x_i))\$$

<=>

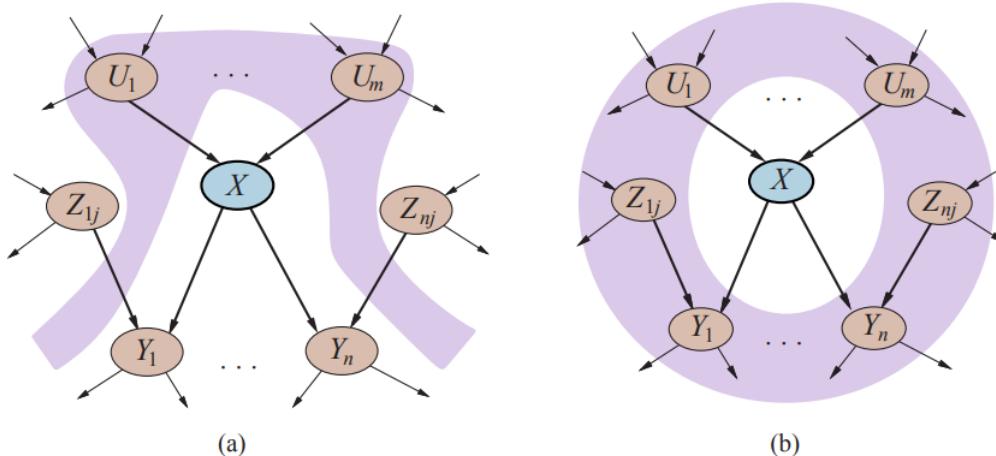
1-
2-

Figure 13.4 (a) A node X is conditionally independent of its non-descendants (e.g., the Z_{ij} s) given its parents (the U_i s shown in the lavender area). (b) A node X is conditionally independent of all other nodes in the network given its Markov blanket (the lavender area).

k

CPT

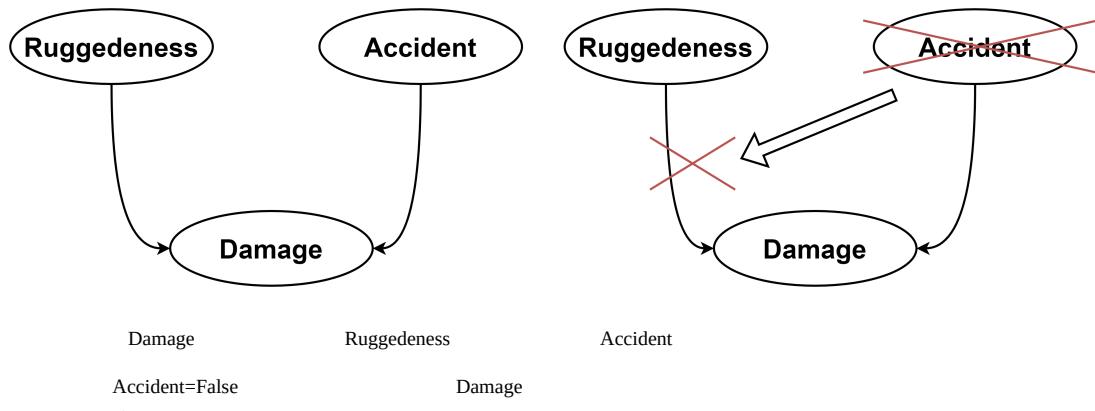
 $\$2^k\$$

“ ”

1.

2. CSI

CSI



3.

“ ”

◦

◦

\$k\$

\$O(k)\$

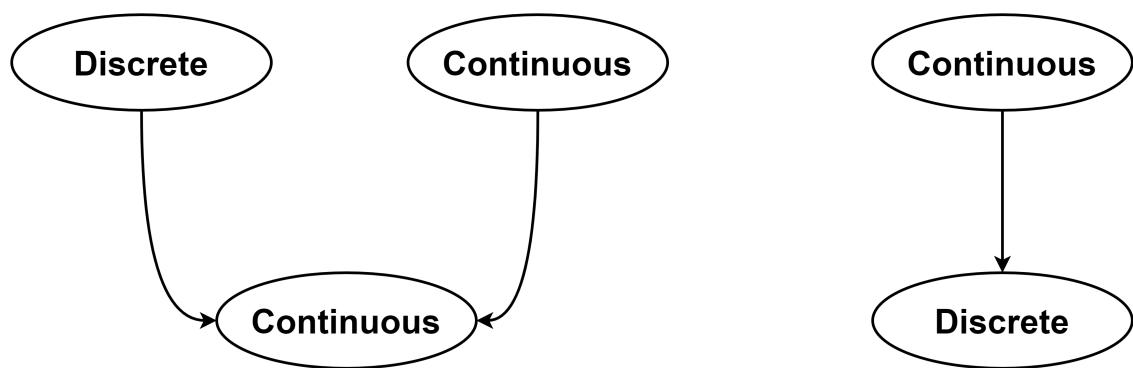
\$O(2^k)\$

1 2

-

→ → → →

/

1.
2.

- →
- $x \rightarrow y$ y
- x y
- x

$$P(y | x) = \mathcal{N}(y; a \cdot x + b, \sigma^2) = \frac{1}{\sigma(2\pi)^{1/2}} e^{-\frac{1}{2}(\frac{y-(a \cdot x + b)}{\sigma})^2}$$

y $\$\\mu=a \\cdot x + b\$$ \$x\\$ $\$\\sigma^2\$$

• →

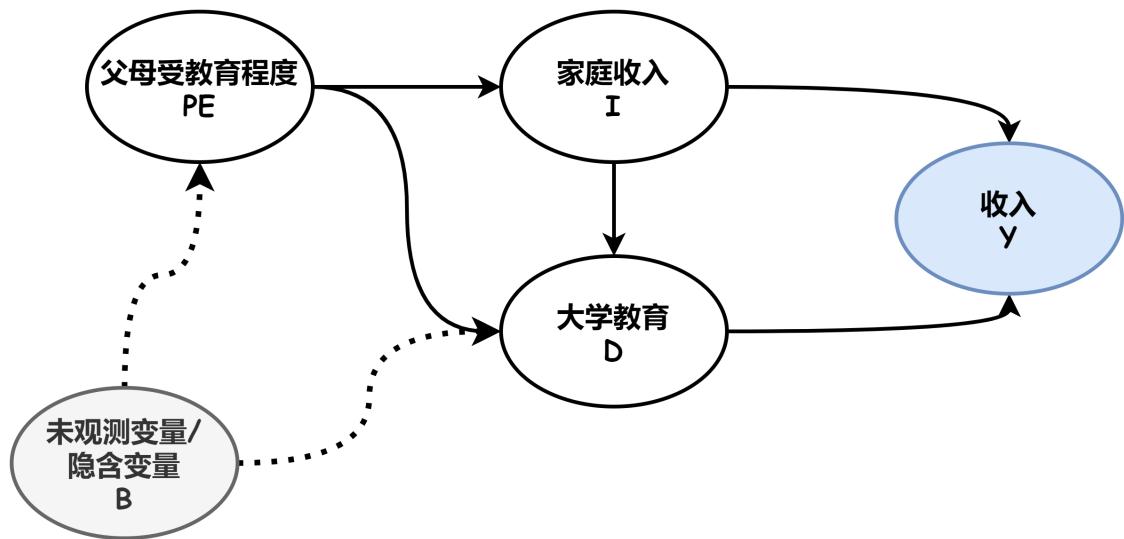
Cost → Buys

“ ” SoftMax

	5600			14000		

“ ” PE I /

“ ” —



“ ” “ ” or

D → Y Y D → Y D “ ” Y

1. $D \rightarrow Y$
2. $D \leftarrow I \rightarrow Y$
3. $D \leftarrow PE \rightarrow I \rightarrow Y$
4. $D \leftarrow B \rightarrow PE \rightarrow I \rightarrow Y$

4 D Y “ ” D Y
“ ”

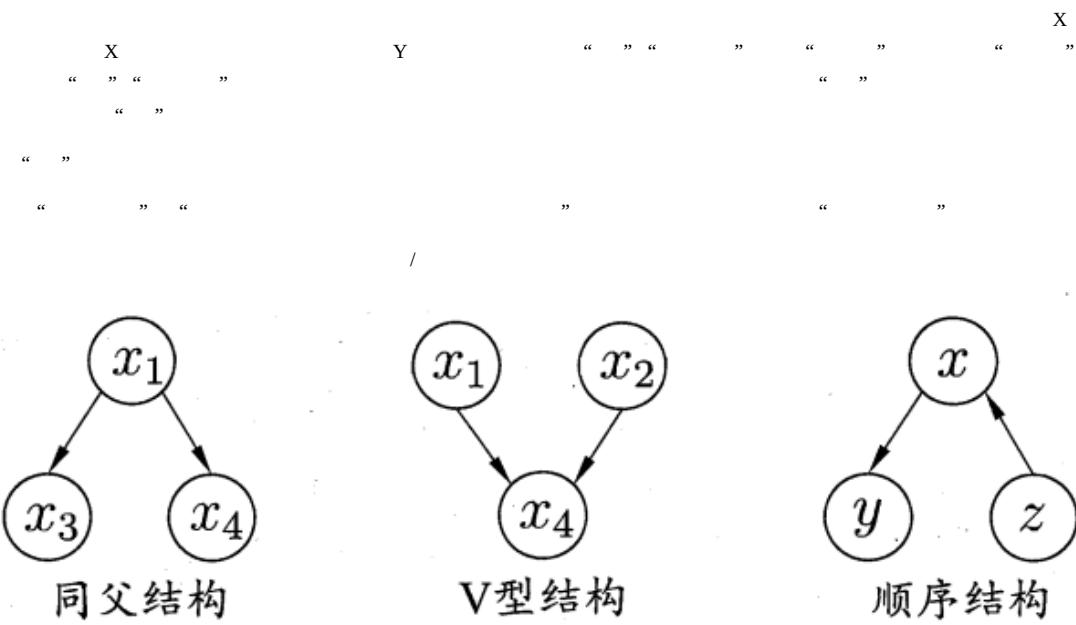
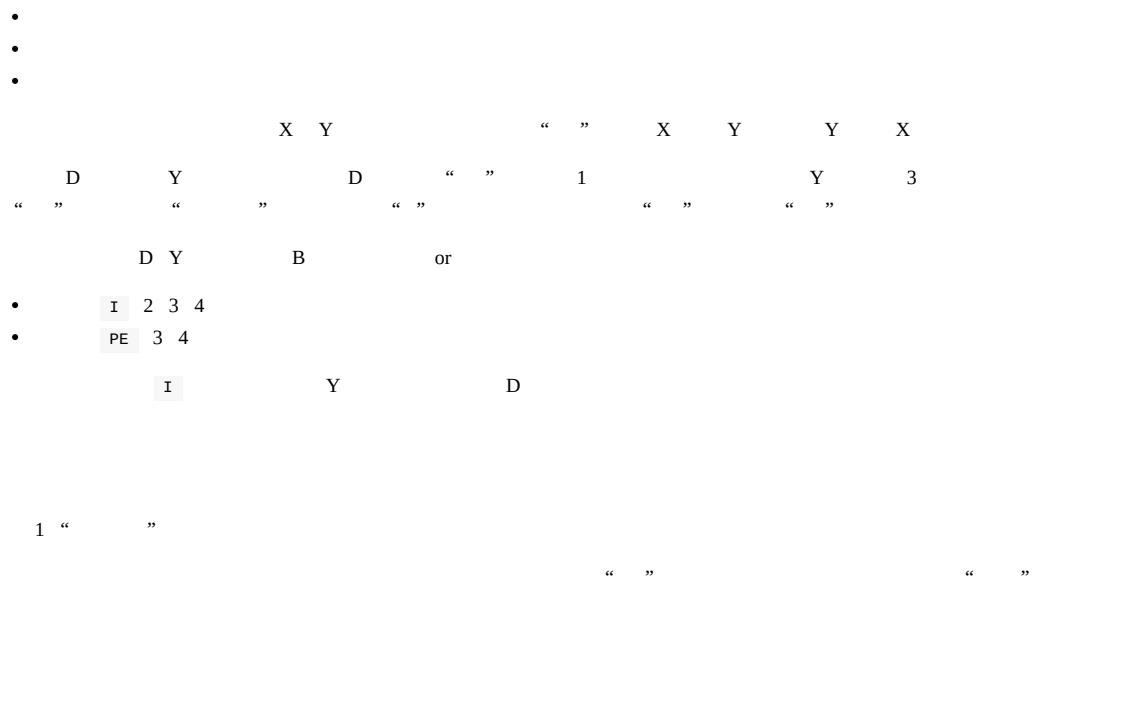
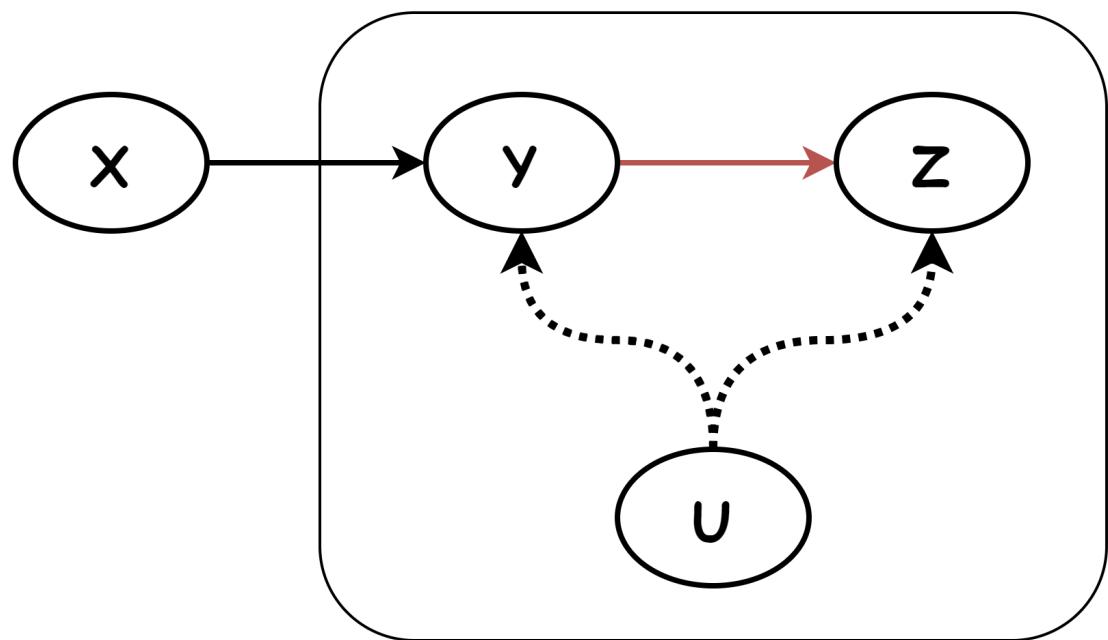


图 7.3 贝叶斯网中三个变量之间的典型依赖关系





Z Y

Z Y

X

Y \leftarrow U \rightarrow Z

- $X \rightarrow Y \rightarrow Z$
- $X \rightarrow Y$

 $Y \rightarrow Z$ Ancestral Graph [wiki](#)

•
•

1. - - - - P156 ↵

2. Artificial Intelligence: A Morden Approach-4th ↵

18

- 18.1
 - 18.1.1
 - 18.1.2
 - 18.1.3
 - 18.1.4
- 18.2
 - 18.2.1
 - 18.2.2
 - 18.2.3
 - 18.2.4
 - 18.2.5
- 18.3
 - 18.3.1
 - 18.3.2
 - 18.3.3
- 18.4
 - 18.4.1
 - 18.4.2
 - 18.4.3
 - 18.4.4

- -
 -
 -
 -
 -
-

1

- 1.1
 - 1.1.1
 - 1.1.2
 - 1.1.3 “ ”
 - 1.1.4
 - 1.1.5
- 1.2
 - 1.2.1
 - 1.2.2
 - 1.2.3
 - 1.2.4
 - 1.2.5
 - 1.2.6
 - 1.2.7
 - 1.2.8
- 1.3
 - 1.3.1 1943—1956
 - 1.3.2 1952—1969
 - 1.3.3 1966—1973
 - 1.3.4 1969—1986
 - 1.3.5 1986—
 - 1.3.6 1987—
 - 1.3.7 2001—
 - 1.3.8 2011—
- 1.4
- 1.5

2

- 2.1
- 2.2
 - 2.2.1
 - 2.2.2
 - 2.2.3
- 2.3
 - 2.3.1
 - 2.3.2
- 2.4
 - 2.4.1
 - 2.4.2
 - 2.4.3
 - 2.4.4
 - 2.4.5
 - 2.4.6
 - 2.4.7

3

- 3.1
 - 3.1.1
 - 3.1.2

- 3.2
 - 3.2.1
 - 3.2.2
 - 3.3
 - 3.3.1
 - 3.3.2
 - 3.3.3
 - 3.3.4
- 3.4
 - 3.4.1
 - 3.4.2 Dijkstra
 - 3.4.3
 - 3.4.4
 - 3.4.5
 - 3.4.6
- 3.5
 - 3.5.1
 - 3.5.2 A*
 - 3.5.3
 - 3.5.4
 - 3.5.5
 - 3.5.6
- 3.6
 - 3.6.1
 - 3.6.2
 - 3.6.3
 - 3.6.4
 - 3.6.5
 - 3.6.6

4

- 4.1
 - 4.1.1
 - 4.1.2
 - 4.1.3
 - 4.1.4
- 4.2
- 4.3
 - 4.3.1
 - 4.3.2
 - 4.3.3
- 4.4
 - 4.4.1
 - 4.4.2
 - 4.4.3
 - 4.4.4
- 4.5
 - 4.5.1
 - 4.5.2
 - 4.5.3
 - 4.5.4

5

5.1

5.2
5.2.1 5.2.2
5.2.3
5.2.4
5.3
5.3.1 5.3.2 5.3.3 5.3.4 5.4
5.5

5.6
5.6.1 5.6.2
5.7

6

6.1
6.1.1 6.1.2 6.1.3 CSP
6.2 CSP
6.2.1 6.2.2 6.2.3 6.2.4 k 6.2.5 6.2.6
6.3 CSP
6.3.1 6.3.2 6.3.3 6.3.4
6.4 CSP
6.5
6.5.1 6.5.2 6.5.3

7

7.1
7.2 wumpus
7.3
7.4
7.4.1 7.4.2 7.4.3 7.4.4
7.5
7.5.1 7.5.2 7.5.3 7.5.4
7.6
7.6.1 7.6.2 7.6.3 SAT
7.7
7.7.1 7.7.2 7.7.3 7.7.4

8

- 8.1

- 8.1.1
- 8.1.2
- 8.2
- 8.2.1
- 8.2.2
- 8.2.3
- 8.2.4
- 8.2.5
- 8.2.6
- 8.2.7
- 8.2.8
- 8.3
- 8.3.1
- 8.3.2
- 8.3.3
- 8.3.4 wumpus
- 8.4
- 8.4.1
- 8.4.2

9

- 9.1
- 9.2
- 9.2.1
- 9.2.2
- 9.3
- 9.3.1
- 9.3.2
- 9.3.3
- 9.4
- 9.4.1
- 9.4.2
- 9.4.3
- 9.4.4 Prolog
- 9.4.5
- 9.5
- 9.5.1
- 9.5.2
- 9.5.3
- 9.5.4
- 9.5.5
- 9.5.6

10

- 10.1
- 10.2
- 10.2.1
- 10.2.2
- 10.2.3
- 10.3
- 10.3.1
- 10.3.2
- 10.4
- 10.5
- 10.5.1
- 10.5.2

- 10.6
- 10.6.1
- 10.6.2

11

- 11.1
- 11.1.1
- 11.1.2
- 11.1.3
- 11.2
- 11.2.1
- 11.2.2
- 11.2.3
- 11.2.4
- 11.3
- 11.3.1
- 11.3.2
- 11.4
- 11.4.1
- 11.4.2
- 11.4.3
- 11.5
- 11.5.1
- 11.5.2
- 11.5.3
- 11.6
- 11.6.1
- 11.6.2
- 11.7

12

- 12.1
 - 12.1.1
 - 12.1.2
- 12.2
 - 12.2.1
 - 12.2.2
 - 12.2.3
- 12.3
- 12.4
- 12.5
 - 12.5.1
 - 12.5.2
- 12.6
 -
- 12.7 wumpus

13

- 13.1
- 13.2
 - 13.2.1
 - 13.2.2

- 13.2.3
- 13.2.4
- 13.3
 - 13.3.1
 - 13.3.2
 - 13.3.3
 - 13.3.4
- 13.4
 - 13.4.1
 - 13.4.2
 - 13.4.3
- 13.5
 - 13.5.1 do
 - 13.5.2

14

- 14.1
 - 14.1.1
 - 14.1.2
- 14.2
 - 14.2.1
 - 14.2.2
 - 14.2.3
- 14.3
 - 14.3.1
 - 14.3.2
- 14.4
 - 14.4.1
 - 14.4.2
 - 14.4.3
 - 14.4.4
- 14.5
 - 14.5.1
 - 14.5.2
 - 14.5.3

15

- 15.1
 - 15.1.1
 - 15.1.2
 - 15.1.3
- 15.2
 - 15.2.1
 - 15.2.2
 - 15.2.3
- 15.3
 - 15.3.1
 - 15.3.2
- 15.4
 - 15.4.1
 - 15.4.2
 - 15.4.3
 - 15.4.4
 - 15.4.5

16

- 16.1
 - 16.2
 - 16.2.1
 - 16.2.2
- 16.3
 - 16.3.1
 - 16.3.2
 - 16.3.3
 - 16.3.4
- 16.4
 - 16.4.1
 - 16.4.2
- 16.5
 - 16.5.1
 - 16.5.2
- 16.6
 - 16.6.1
 - 16.6.2
 - 16.6.3
 - 16.6.4
 - 16.6.5
 - 16.6.6
- 16.7
 - 16.7.1
 - 16.7.2

17

- 17.1
 - 17.1.1
 - 17.1.2
 - 17.1.3
 - 17.1.4 MDP
- 17.2 MDP
 - 17.2.1
 - 17.2.2
 - 17.2.3
 - 17.2.4 MDP
- 17.3
 - 17.3.1
 - 17.3.2
 - 17.3.3
 - 17.3.4
- 17.4 MDP
 - POMDP
- 17.5 POMDP
 - 17.5.1 POMDP
 - 17.5.2 POMDP

18

- 18.1
 - 18.1.1
 - 18.1.2
 - 18.1.3
 - 18.1.4
- 18.2
 - 18.2.1

- 18.2.2
- 18.2.3
- 18.2.4
- 18.2.5
- 18.3
 - 18.3.1
 - 18.3.2
 - 18.3.3
- 18.4
 - 18.4.1
 - 18.4.2
 - 18.4.3
 - 18.4.4

19

- 19.1
- 19.2
- 19.3
- 19.3.1
- 19.3.2
- 19.3.3
- 19.3.4
- 19.3.5
- 19.4
- 19.4.1
- 19.4.2
- 19.4.3
- 19.4.4
- 19.5 PAC
- 19.6
- 19.6.1
- 19.6.2
- 19.6.3
- 19.6.4
- 19.6.5
- 19.7
- 19.7.1
- 19.7.2 k-d
- 19.7.3
- 19.7.4
- 19.7.5
- 19.7.6
- 19.8
- 19.8.1
- 19.8.2
- 19.8.3
- 19.8.4
- 19.8.5
- 19.8.6
- 19.9
- 19.9.1
- 19.9.2
- 19.9.3
- 19.9.4

- 19.9.5

20

- 20.1
- 20.2
- 20.2.1
- 20.2.2
- 20.2.3
- 20.2.4
- 20.2.5
- 20.2.6
- 20.2.7
- 20.2.8
- 20.3 EM
- 20.3.1
- 20.3.2
- 20.3.3
- 20.3.4 EM
- 20.3.5

21

- 21.1
- 21.1.1
- 21.1.2
- 21.2
- 21.2.1
- 21.2.2
- 21.2.3
- 21.3
- 21.3.1
- 21.3.2
- 21.3.3
- 21.4
- 21.4.1
- 21.4.2
- 21.5
- 21.5.1
- 21.5.2
- 21.5.3
- 21.5.4
- 21.6
- 21.6.1
- 21.6.2 RNN
- 21.7
- 21.7.1
- 21.7.2
- 21.8
- 21.8.1
- 21.8.2
- 21.8.3

22

22.1	22.2	22.2.1	22.2.2	22.2.3	22.3	22.3.1	22.3.2
22.3.3	Q	22.4	22.4.1	22.4.2	22.4.3	22.4.4	22.4.5
22.5	22.6		22.7	22.7.1	22.7.2		

23

23.1	23.1.1	23.1.2 n	23.1.3	n	23.1.4 n	23.1.5	23.1.6	23.1.7	23.2	E0
	23.3	23.3.1	23.3.2		23.4	23.4.1	23.4.2	23.5		23.6

24

24.1	24.2	24.3	24.3.1	24.3.2	24.4 Transformer	24.4.1	24.4.2	24.2.3	LSTM
								Transformer	24.5
24.5.1		24.5.2		24.5.3		24.6	SOTA		

25

25.1	25.2	25.2.1		25.2.2	25.2.3	25.2.4	25.2.5	25.3	25.3.1
25.3.2	25.3.3	25.3.4		25.4	25.4.1	25.4.2			25.5
25.6	25.6.1			25.6.2	25.6.3	25.6.4	25.7		25.7.1
		25.7.2		25.7.3	25.7.4	25.7.5	25.7.6		

26

26.1	26.2	26.2.1		26.2.2	26.2.3	26.3	26.4	26.4.1	
26.4.2		26.4.3			26.5	26.5.1	26.5.2	26.5.3	26.5.4
26.6		26.7		26.7.1	26.7.2	26.8	26.8.1	26.8.2	
		26.9.1		26.9.2	26.10				26.9

27

27.1		27.1.1		27.1.2		27.1.3	27.1.4	27.2	27.2.1
27.2.2		27.3		27.3.1		27.3.2		27.3.3	27.3.4
								27.3.5	27.3.6
		27.3.7							

28

28.1		28.2	
------	--	------	--

A

A.1	\$O()\$	A.1.1	A.1.2 NP	A.2	A.3
-----	---------	-------	----------	-----	-----

```
(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 libaccinj64-11.8 libcu++-dev lib
libcublaslt11 libcudac1 libcudart11.0 libcuffft10 libcufftw10 libcuinj64-11.8 libcuhti-dev libcuhti-doc libcuhti11.8 l
libcusolver11 libcusolvermg11 libcusparse11 libgl-dev libglx-dev libnppc11 libnppial11 libnppicc11 libnppidei11 libn
libnppim11 libnppist11 libnppisu11 libnppitc11 libnpps11 libnvblas11 libnvivid1 libnvidia-cfg1 libnvidia-ml-dev lib
libnvidia-pkcs11-openssl3 libnvidia-ptxjitcompiler1 libnvjpeg11 libnvrtc-builtins11.8 libnvrtc11.2 libnvtoolsext1 li
libvdpa-dev linux-compiler-gcc-12-x86 linux-headers-6.1.0-27-amd64 linux-headers-6.1.0-27-common linux-headers-amd6
node-html5shiv nvidia-alternative nvidia-installer-cleanup nvidia-kernel-common nvidia-kernel-dkms nvidia-kernel-sup
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 libvdpa-dev nodejs
Recommended packages:
libcuda1:i386
The following NEW packages will be installed:
dkms firmware-nvidia-gsp glx-alternative-mesa glx-alternative-nvidia glx-diversions libaccinj64-11.8 libcu++-dev lib
libcublaslt11 libcudac1 libcudart11.0 libcuffft10 libcufftw10 libcuinj64-11.8 libcuhti-dev libcuhti-doc libcuhti11.8 l
libcusolver11 libcusolvermg11 libcusparse11 libgl-dev libglx-dev libnppc11 libnppial11 libnppicc11 libnppidei11 libn
libnppim11 libnppist11 libnppisu11 libnppitc11 libnpps11 libnvblas11 libnvivid1 libnvidia-cfg1 libnvidia-ml-dev lib
libnvidia-pkcs11-openssl3 libnvidia-ptxjitcompiler1 libnvjpeg11 libnvrtc-builtins11.8 libnvrtc11.2 libnvtoolsext1 li
libvdpa-dev linux-compiler-gcc-12-x86 linux-headers-6.1.0-27-amd64 linux-headers-6.1.0-27-common linux-headers-amd6
node-html5shiv nvidia-alternative nvidia-cuda-dev nvidia-installer-cleanup nvidia-kernel-common nvidia-kernel-dkms 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 [4,760 B]
Get:4 http://mirrors.ustc.edu.cn/debian bookworm/contrib amd64 nvidia-installer-cleanup amd64 20220217+3-deb12u1 [13.3
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 [5,776 B]
Get:7 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-legacy-check amd64 535.183.01-1-deb12u1 [156 kB]
Get:8 http://mirrors.ustc.edu.cn/debian bookworm/contrib amd64 nvidia-support amd64 20220217+3-deb12u1 [14.2 kB]
Get:9 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-alternative amd64 535.183.01-1-deb12u1 [152 kB]
Get:10 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvidia-ptxjitcompiler1 amd64 535.183.01-1-deb12u1
Get:11 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvidia-pkcs11-openssl3 amd64 535.183.01-1-deb12u1
Get:12 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcudac1 amd64 535.183.01-1-deb12u1 [2,999 kB]
Get:13 http://mirrors.ustc.edu.cn/debian bookworm/non-free-firmware amd64 firmware-nvidia-gsp amd64 535.183.01-1-deb12
Get:14 http://mirrors.ustc.edu.cn/debian bookworm/contrib amd64 nvidia-kernel-common amd64 20220217+3-deb12u1 [4,488 B]
Get:15 http://mirrors.ustc.edu.cn/debian bookworm/contrib amd64 nvidia-modprobe amd64 535.161.07-1-deb12u1 [21.1 kB]
Get:16 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-kernel-support amd64 535.183.01-1-deb12u1 [151
Get:17 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-kernel-dkms amd64 535.183.01-1-deb12u1 [44.9 M
Get:18 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvidia-cfg1 amd64 535.183.01-1-deb12u1 [240 kB]
Get:19 http://mirrors.ustc.edu.cn/debian bookworm/contrib amd64 nvidia-persistenced amd64 535.171.04-1-deb12u1 [27.1 k
Get:20 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcuhti11.8 amd64 11.8.87-11.8.0-5-deb12u1 [8,039 kB
Get:21 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libaccinj64-11.8 amd64 11.8.87-11.8.0-5-deb12u1 [784
Get:22 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 libcu++-dev all 1.8.1-2 [554 kB]
Get:23 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 libcub-dev all 1.17.2-2 [245 kB]
Get:24 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcublaslt11 amd64 11.11.3.6-11.8.0-5-deb12u1 [206 M
Get:25 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcublas11 amd64 11.11.3.6-11.8.0-5-deb12u1 [44.5 MB
Get:26 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcuda11.0 amd64 11.8.89-11.8.0-5-deb12u1 [167 kB]
Get:27 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcufft10 amd64 11.1.1+10.9.0.58-11.8.0-5-deb12u1 [
```

```

Get:28 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcufftw10 amd64 11.1.1+~10.9.0.58~11.8.0-5~deb12u1
Get:29 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcuinj64-11.8 amd64 11.8.87~11.8.0-5~deb12u1 [928 kB]
Get:30 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcuhti-dev amd64 11.8.87~11.8.0-5~deb12u1 [7,892 kB]
Get:31 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 node-html5shiv all 3.7.3+dfsg-5 [13.2 kB]
Get:32 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcuhti-doc all 11.8.87~11.8.0-5~deb12u1 [2,463 kB]
Get:33 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcurland10 amd64 11.1.1+~10.3.0.86~11.8.0-5~deb12u1
Get:34 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcusolver11 amd64 11.4.1.48~11.8.0-5~deb12u1 [32.0 kB]
Get:35 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcusolvermg11 amd64 11.4.1.48~11.8.0-5~deb12u1 [20.8 MB]
  37% [35 libcusolvermg11 6,765 kB/20.4 MB 33%] 8

Get:36 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcusparse11 amd64 11.7.5.86~11.8.0-5~deb12u1 [116 MB]
Ign:36 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcusparse11 amd64 11.7.5.86~11.8.0-5~deb12u1
Get:37 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 libglx-dev amd64 1.6.0-1 [15.3 kB]
Get:38 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 libgl-dev amd64 1.6.0-1 [100 kB]
Get:39 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnppc11 amd64 11.8.0.86~11.8.0-5~deb12u1 [391 kB]
Get:40 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnppial11 amd64 11.8.0.86~11.8.0-5~deb12u1 [5,547 kB]
Get:41 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnppicc11 amd64 11.8.0.86~11.8.0-5~deb12u1 [2,489 kB]
Get:42 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnppidei11 amd64 11.8.0.86~11.8.0-5~deb12u1 [2,626 kB]
Get:43 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnppif11 amd64 11.8.0.86~11.8.0-5~deb12u1 [48.0 MB]
Get:44 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnppig11 amd64 11.8.0.86~11.8.0-5~deb12u1 [15.4 MB]
Get:45 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnppim11 amd64 11.8.0.86~11.8.0-5~deb12u1 [3,195 kB]
Get:46 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnppist11 amd64 11.8.0.86~11.8.0-5~deb12u1 [16.3 MB]
Get:47 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnppisu11 amd64 11.8.0.86~11.8.0-5~deb12u1 [165 kB]
Get:48 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnppitc11 amd64 11.8.0.86~11.8.0-5~deb12u1 [1,319 kB]
Get:49 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnpps11 amd64 11.8.0.86~11.8.0-5~deb12u1 [7,502 kB]
Get:50 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvblas11 amd64 11.11.3.6~11.8.0-5~deb12u1 [176 kB]
Get:51 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvivid1 amd64 535.183.01-1~deb12u1 [1,501 kB]
Get:52 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvidia-ml1 amd64 535.183.01-1~deb12u1 [684 kB]
Get:53 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvidia-ml-dev amd64 11.8.86~11.8.0-5~deb12u1 [79.2 kB]
Get:54 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvjpeg11 amd64 11.9.0.86~11.8.0-5~deb12u1 [1,873 kB]
Get:55 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvrtc-builtins11.8 amd64 11.8.89~11.8.0-5~deb12u1
Get:56 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvrtc11.2 amd64 11.8.89~11.8.0-5~deb12u1 [16.4 MB]
Get:57 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvtoolsext1 amd64 11.8.86~11.8.0-5~deb12u1 [31.9 kB]
Get:58 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libnvvm4 amd64 11.8.89~11.8.0-5~deb12u1 [8,186 kB]
Get:59 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 libthrust-dev all 1.17.2-2 [445 kB]
Get:60 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 libvdpau-dev amd64 1.5-2 [41.5 kB]
Get:61 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 linux-compiler-gcc-12-x86 amd64 6.1.115-1 [920 kB]
Get:62 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 linux-headers-6.1.0-27-common all 6.1.115-1 [10.1 MB]
Get:63 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 linux-kbuild-6.1 amd64 6.1.115-1 [1,177 kB]
Get:64 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 linux-headers-6.1.0-27-amd64 amd64 6.1.115-1 [1,450 kB]
Get:65 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 linux-headers-amd64 amd64 6.1.115-1 [1,416 kB]
Get:66 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-cuda-dev amd64 11.8.89~11.8.0-5~deb12u1 [774 MB]
Ign:66 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-cuda-dev amd64 11.8.89~11.8.0-5~deb12u1
Get:67 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-smi amd64 535.183.01-1~deb12u1 [386 kB]
Get:36 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 libcusparse11 amd64 11.7.5.86~11.8.0-5~deb12u1 [116 MB]
Get:66 http://mirrors.ustc.edu.cn/debian bookworm/non-free amd64 nvidia-cuda-dev amd64 11.8.89~11.8.0-5~deb12u1 [774 MB]
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-libcudai_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 libcuda11.0:amd64.
Preparing to unpack .../17-libcuda11.0_11.8.89-11.8.0-5~deb12u1_amd64.deb ...
Unpacking libcuda11.0: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 libvdpa-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 libnppci1: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 libcudart11.0: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 libnvrta-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 libcufft10: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 libcuhti-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 slave links
update-alternatives: using /usr/lib/mesa-diverted to provide /usr/lib/mesa-diverted/libGL.so-master (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-p
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 libnvvcuvid1: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 slave links
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
```

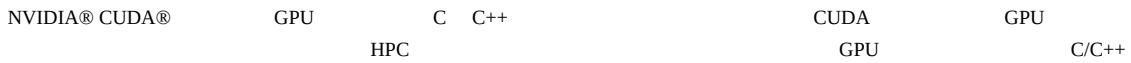


CUDA Learn

- [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](#)
- [GPU](#)
- [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.



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.

GPU GPU GPU

CUDA C++ Programming Guide v12.6

[CUDA C++ Programming Guide](#)



1.

- Thread
- Thread Block
- Grid 1D 2D 3D
 threadIdx blockIdx blockDim gridDim

2.

- Global Memory
- Shared Memory
- Local Memory
- Registers

3. Kernel

- CUDA GPU C/C++ CPU GPU

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:
- shared memories:
- barrier synchronization:

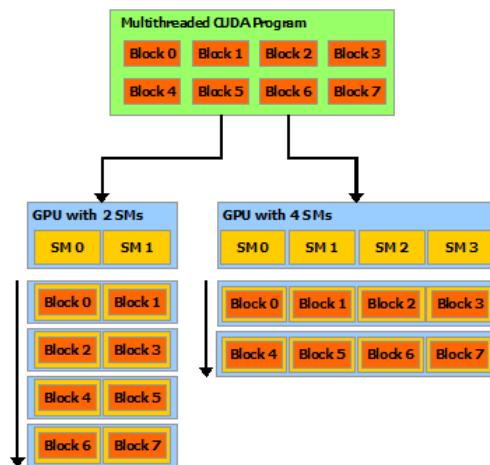
These abstractions provide fine-grained data parallelism and thread parallelism, nested within coarse-grained data parallelism and task parallelism. They guide the programmer to partition the problem into coarse sub-problems that can be solved independently in parallel by blocks of threads, and each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

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.

GPU

CUDA

3



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

GPU

(SM)

Programming Model

- Kernels:
- Thread Hierarchy:
- Memory Hierarchy:
- Heterogeneous Programming:
- Asynchronous SIMD Programming Model: SIMT
- Compute Capability:

Kernels:

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++	C++	C++	N
----------	-----	-----	---

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.



Thread Hierarchy:

- grids - blocks - threads
- core
- blocks, threads
 - `threadIdx.x, .y, .z`
 - `blockIdx.x, .y, .z`
- block :
- `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 Dx 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.

GPU	
1024	

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.

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.

4

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.

`__syncthreads()`

<code>__syncthreads()</code>	<code>__syncthreads()</code>	API
------------------------------	------------------------------	-----

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.

L1	<code>__syncthreads()</code>	
----	------------------------------	--

Memory Hierarachy:

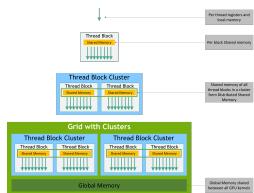
CUDA threads may access data from multiple memory spaces during their execution as illustrated by Figure 6. Each thread has private local memory. Each thread block has shared memory visible to all threads of the block and with the same lifetime as the block. Thread blocks in a thread block cluster can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

CUDA

6

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).

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



Heterogeneous Programming:

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.

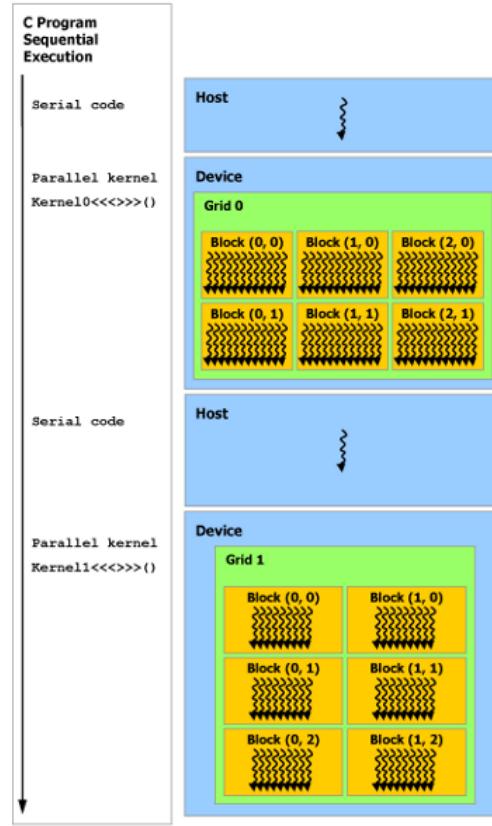
GPU	7 CUDA CUDA	C++
	CPU	

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	DRAM	
------	------	--

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.

CPU	GPU	
-----	-----	--



Serial code executes on the host while parallel code executes on the device.

Asynchronous SIMD Programming Model: SIMD

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.

CUDA	NVIDIA Ampere GPU	CUDA
CUDA		

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.

CUDA	<code>cuda::memcpy_async</code>	GPU
------	---------------------------------	-----

2.5.1. Asynchronous Operations

2.5.1.

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.

CUDA	CUDA
------	------

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`).

as-if	CUDA
<code>cuda::memcpy_async</code>	<code>cooperative_groups::memcpy_async</code>

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.

<code>cuda::barrier</code>	<code>cuda::pipeline</code>
----------------------------	-----------------------------

`cuda::pipeline`

CUDA C++

Compute Capability:

Programming Interface

Hardware Implementation

Performance Guidelines

PTX Parallel Thread Execution

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

PTX

PTX exposes the GPU as data-parallel computing device.

Numba

Overview

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.



Install CUDA

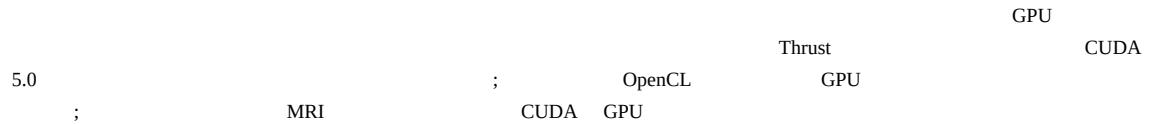
[CUDA Toolkit Archive](#)

[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.



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);

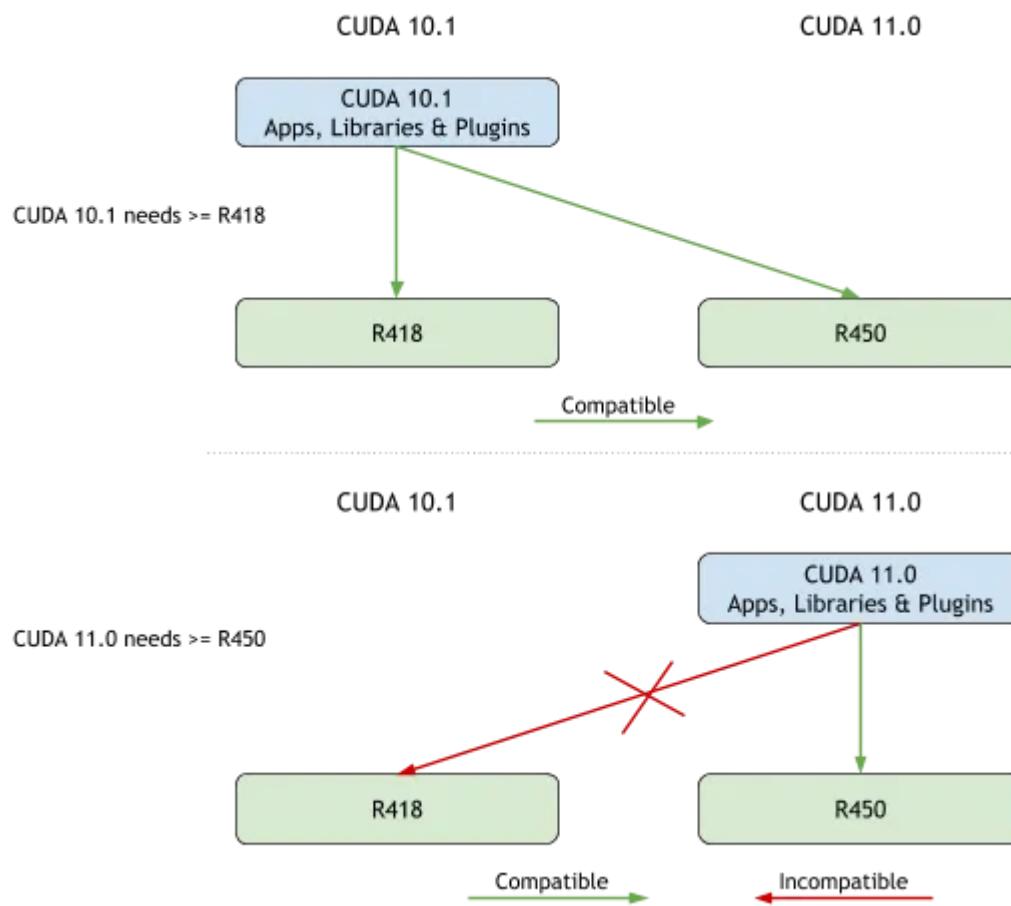
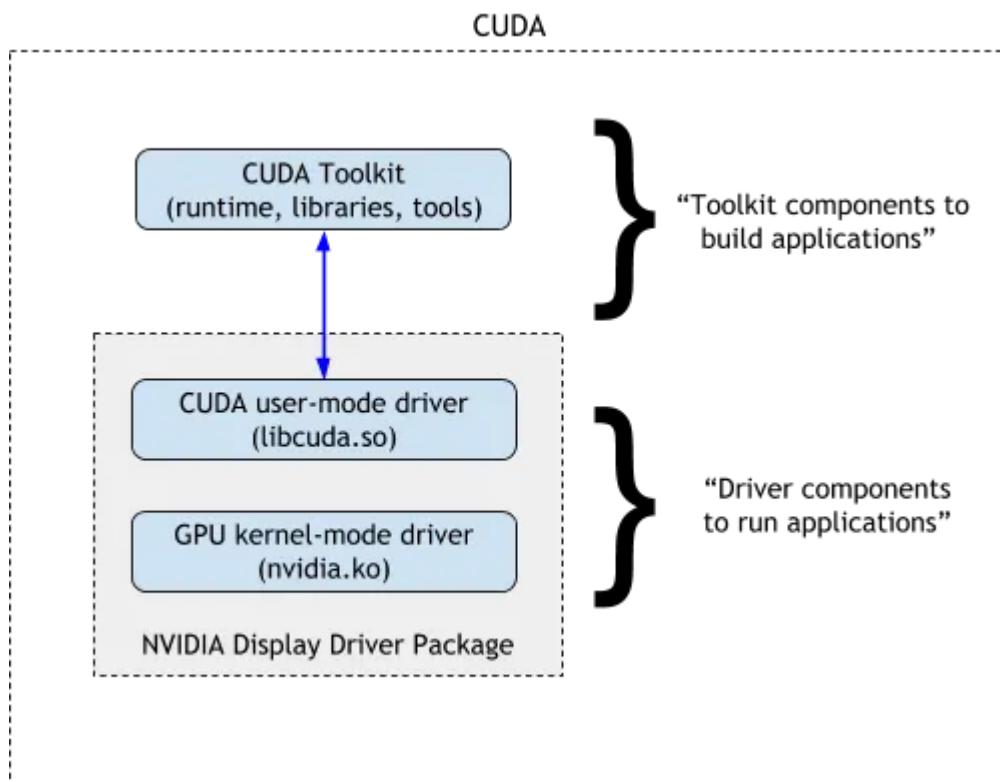
    return 0;
}

```

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

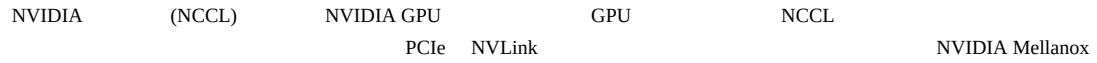
# Activate the virtual environment
echo "export PATH=/usr/local/cuda-11.8/bin:$PATH" >> venv/my_env/bin/activate
echo "LD_LIBRARY_PATH=/usr/local/cuda-11.8/lib64:$LD_LIBRARY_PATH" >> venv/my_env/bin/activate
```

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

NVIDIA Collective Communications Library (NCCL)

NVIDIA NCCL 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.



Tensor Cores

CUDA Toolkit Documentation 12.2 (nvidia.com)

NVIDIA Tensor Core () AI GPU 04_ _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 640 个 Tensor 内核：每个 SM 8 个。



Tensor Core

GEMM General Matrix Multiplication

FMA Fused Multiply–accumulate operation

Volta 架构第一代 Tensor Core

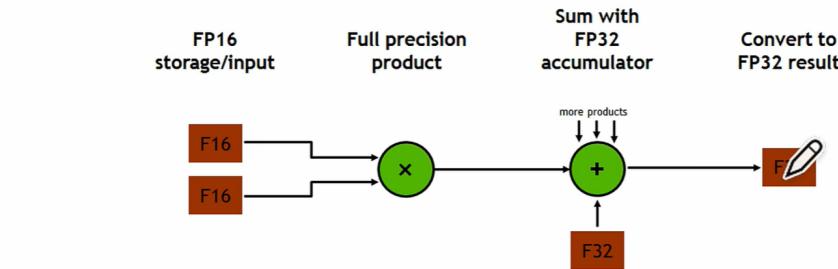
- 每个 Tensor Core 每周期能执行 **4x4x4 GEMM**，64 个 FMA。执行运算 $D = A * B + C$ ，其中 A、B、C 和 D 是 4×4 矩阵。矩阵乘法输入 A 和 B 是 FP16 矩阵，而累加矩阵 C 和 D 可以是 FP16 或 FP32 矩阵。
- Tensor Core 执行融合乘法加法，其中两个 4×4 FP16 矩阵相乘，然后将结果添加到 4×4 FP16 或 FP32 矩阵中，最终输出新的 4×4 FP16 或 FP32 矩阵。

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

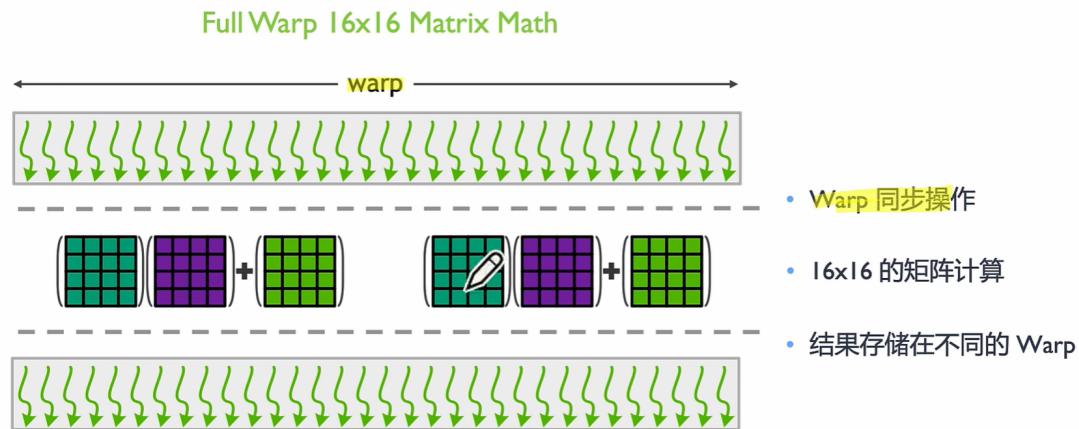
FP16 or FP32 FP16 FP16 FP16 or FP32

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

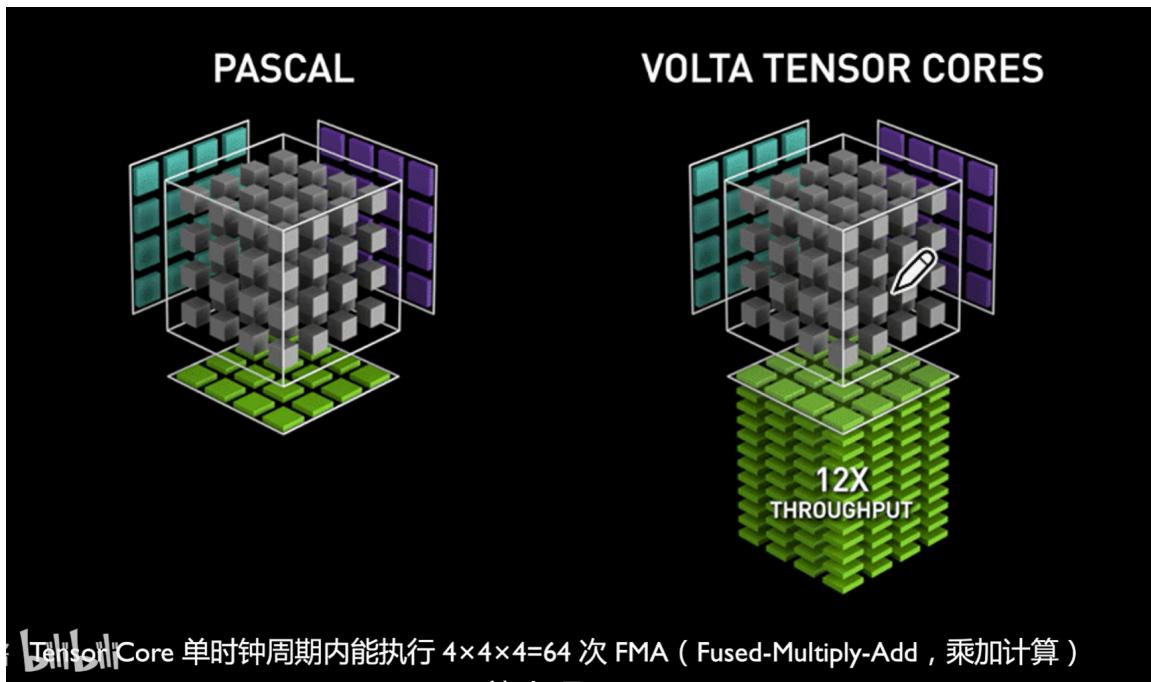
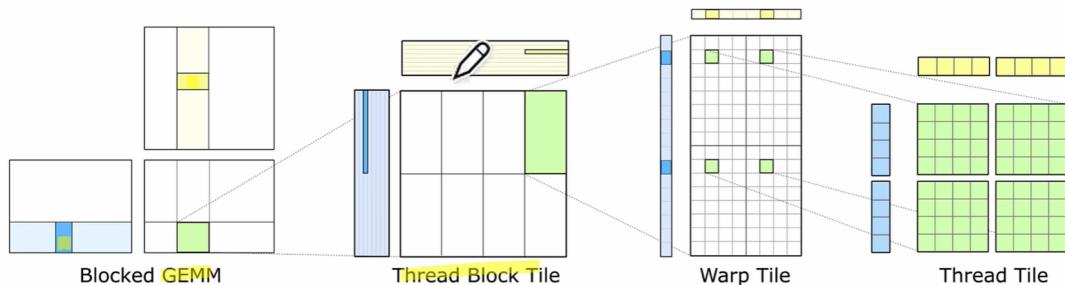
16x16x16 Warp Matrix Multiply and Accumulate(WMMA)

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

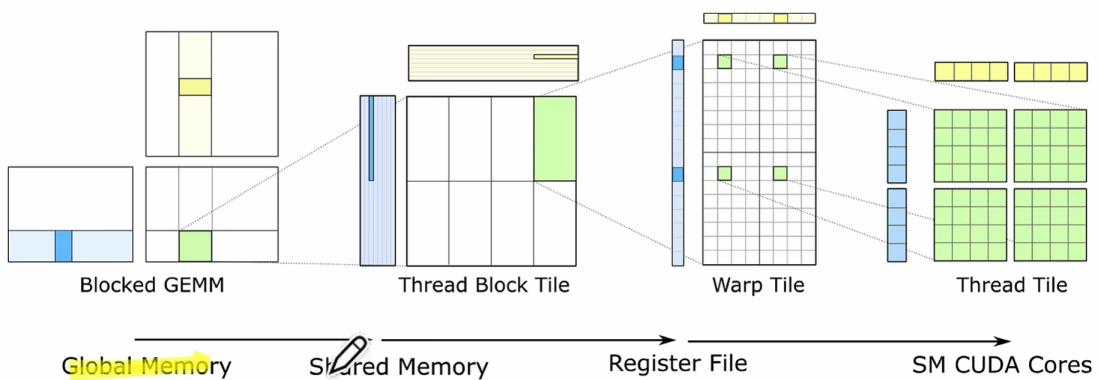
$$D = \left(\begin{array}{c|c} \text{FP16 or FP32} & \text{FP16} \\ \hline & \end{array} \right) \left(\begin{array}{c|c} \text{FP16} & \text{FP16} \\ \hline & \end{array} \right) + \left(\begin{array}{c|c} \text{FP16 or FP32} & \text{FP16} \\ \hline & \end{array} \right)$$

CNN vs GEMM

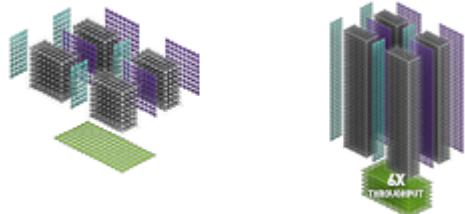
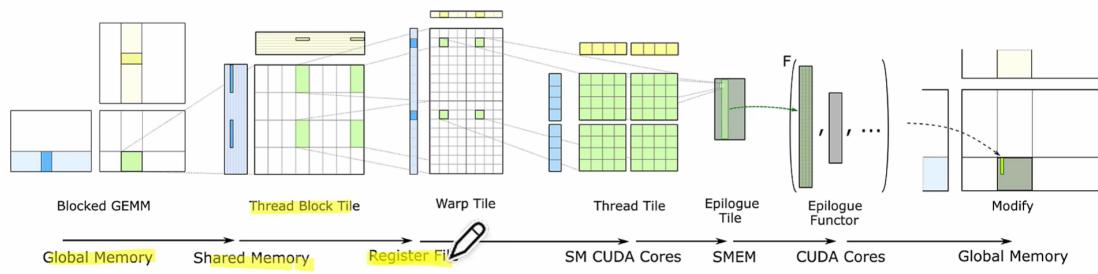
- 卷积的计算可以转换为两个矩阵相乘的求解，得到最终的卷积计算结果。
- GEMM 计算可以被成批地放在一起，作为单个大型矩阵乘法运算运行。
- 在 GPU CUDA Core 中，通过线程 Block 提供具体的计算。



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 Tensor

CUDA Tensor NVIDIA GPU

1.

- CUDA CUDA “ ” ” ” ” ”
- Tensor Tensor NVIDIA Volta

2.

- CUDA CUDA GPU
- Tensor Tensor

3.

- CUDA FP64 FP32
- Tensor FP16 FP32

4. GPU

- CUDA 2007 CUDA CUDA NVIDIA GPU
- Tensor Volta Tesla V100 Turing Ampere

5.

- CUDA
 - Tensor
- | | | | | | |
|------|--------|--------|-----|------|--------|
| CUDA | Tensor | NVIDIA | GPU | CUDA | Tensor |
| | | NVIDIA | | | |

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`, `thikness`, `linType`

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 want 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
$(lsb_release -sc) 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

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

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

YAML

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

CI

- Start, stop, and rebuild services
 -
- View the status of running services

- - Stream the log output of running services
 -
 - Run a one-off command on a service
 -

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

C++

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`
- `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.
- `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
```

-2022

Distributed swarm trajectory optimization for formation flight in dense environments

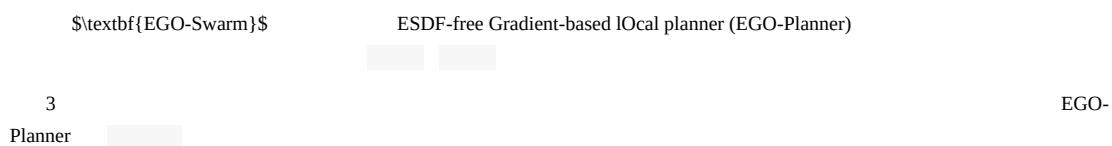


EGO-Swarm-21



- nontriviality of obstacle parameterization
- limited sensing range
- unreliable and bandwidth limited communication
- positioning drift caused by inconsistent localization

1 Introduction



- topological planning
- reciprocal collision avoidance

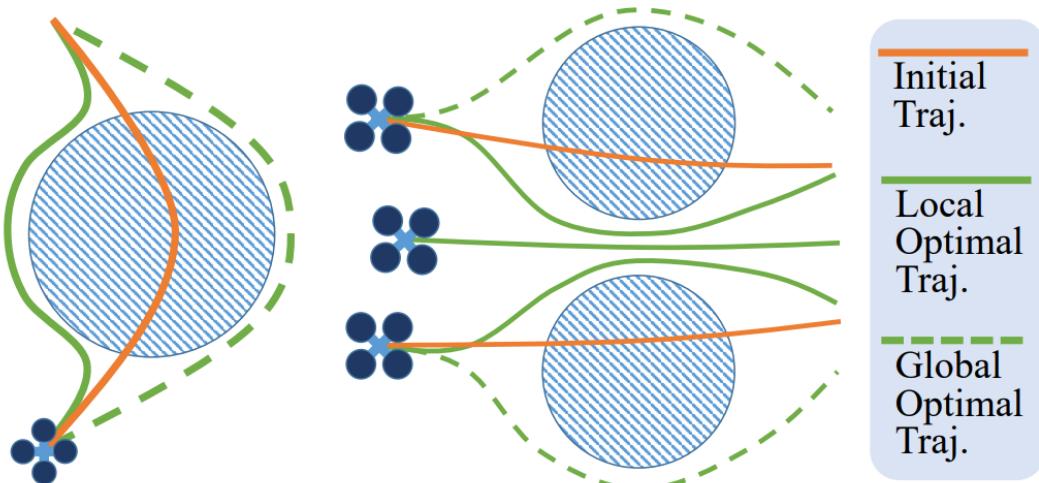


Fig. 3: 左侧:局部最优轨迹的动力学不可行性。右侧:三个机体挤满飞行通道。

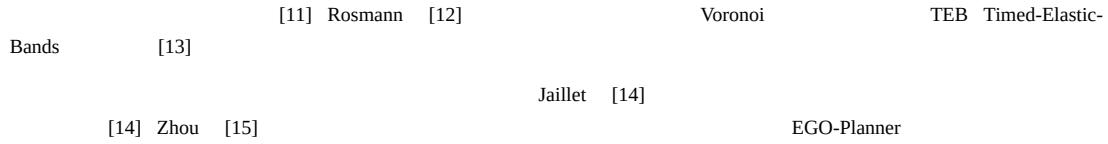
2 Related Works

A. Single Quadrotor Local Planning

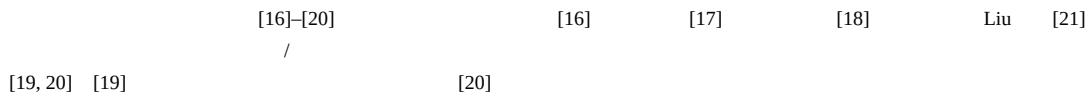


EGO-Planner-20

B. Topological Planning



C. Decentralized Drone Swarm



3 IMPLICIT TOPOLOGICAL TRAJECTORY GENERATION OF GRADIENT-BASED LOCAL PLANNING

EGO-Planner \cite{zhou2020ego}

A. An ESDF-Free Gradient-based local planner

EGO-Planner	$\$J_s\$$	$\$J_c\$$	$\$J_d\$$	$\$J_t\$$
$\$\\mathbf{Q}\$$	B-	$\$\\mathbf{\Phi}\$$		
	$\min_{\mathbf{Q}} J_{\text{EGO}} = \sum \lambda_r J_r,$			
$\$r=\{s,c,d,t\}\$$	$\$\\lambda\$$	$\$J\$$	$\$J_s\$$	$\$J_t\$$
$\$L(\mathbf{Q})\$$		$\$\\mathcal{D}\$$		
		$J_r = \sum_{\mathbf{Q} \in \Phi} \ L(\mathbf{Q}) - \mathcal{D}\ _n^n$		
$\$J_c\$$	$\$J_d\$$	$\$\\mathcal{T}\$$		
	$J_r = \sum_{\mathbf{Q} \in \Phi} \begin{cases} \left\ \frac{L(\mathbf{Q}) - (\mathcal{T} - \epsilon)}{S} \right\ _n^n & L(\mathbf{Q}) > (\mathcal{T} - \epsilon) \\ 0 & L(\mathbf{Q}) \leq (\mathcal{T} - \epsilon) \end{cases}$			
$\$S\$$	$\$n\$$	$\$\\epsilon\$$	$\backslash\text{cite}\{rosmann2012trajectory\}$	$\$L(\cdot\$$
$\$L(\cdot\$$			$\backslash\text{cite}\{zhou2020ego\}$	
EGO-Planner		$\$\\mathbf{Q}\$$		
$\$\\mathbf{p},\\mathbf{v}\$$		$\$\\mathbf{Q}\$$	$\$\\mathbf{v}\$$	
$\backslash\text{ref}\{\text{pic:ap_v_pair}\}$	$\$i\$$	$\$\\mathbf{Q}_i\$$	$\$d_{ij}\$$	
$\$\\mathbf{p},\\mathbf{v}\$$		$\backslash\text{ref}\{\text{pic:ap_v_pair}\}$???	$\$\\mathbf{\Phi}\$$
	$\$\\mathbf{Gamma}\$$	$\$\\mathbf{\Phi}\$$	$\$\\mathbf{Gamma}\$$	$\$\\mathbf{v}\$$
$\$\\mathbf{p}\$$	$\$\\mathbf{p},\\mathbf{v}\$$	$\$d_{ij}\$$		EGO-Planner
		$\backslash\text{cite}\{zhou2020ego\}$		

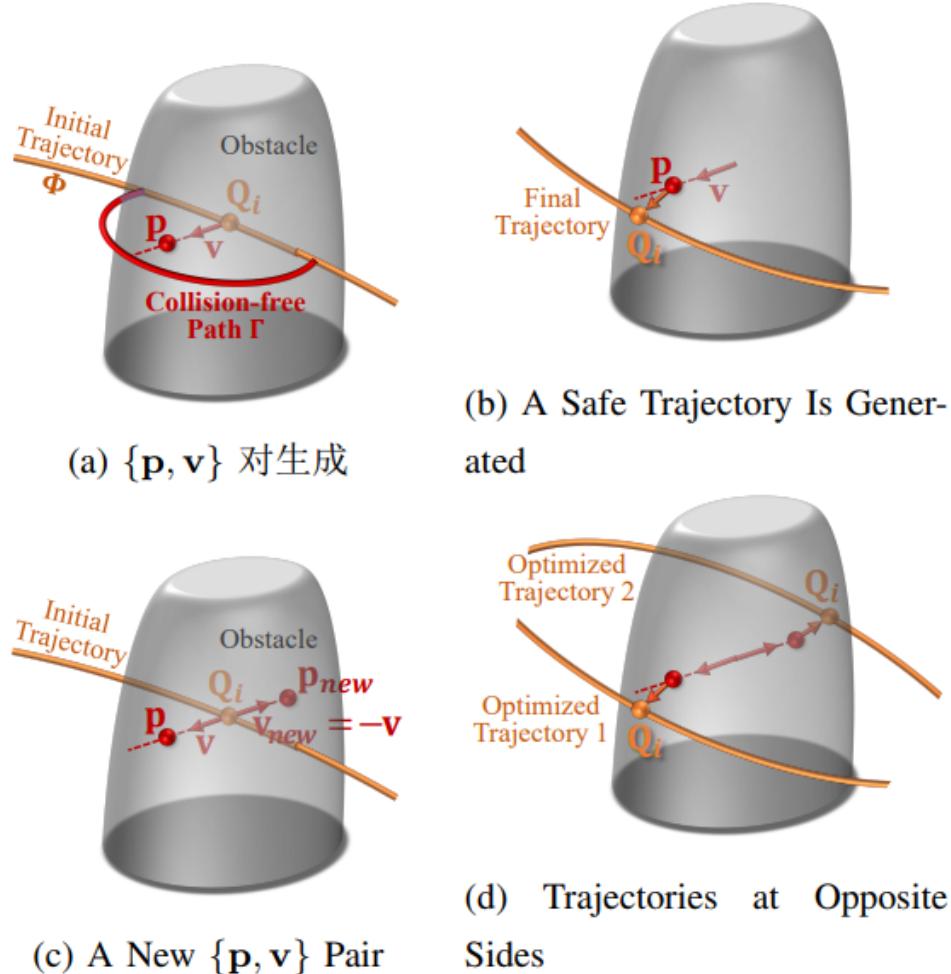


Fig. 4: EGO-Planner和拓扑轨迹生成的示意图如下所示：(a) 搜索绕过碰撞轨迹的安全路径，然后生成 $\{\mathbf{p}, \mathbf{v}\}$ 对；(b) 优化器找到满足 $d = (\mathbf{Q}_i - \mathbf{p}) \cdot \mathbf{v}$ 大于一个固定值的安全轨迹；(c) 生成一个新的 $\{\mathbf{p}, \mathbf{v}\}$ 对，其中 $\mathbf{v}_{\text{new}} := -\mathbf{v}$ ，且 \mathbf{p}_{new} 位于障碍物的对面；(d) 生成满足 $\{\mathbf{p}_{\text{new}}, \mathbf{v}_{\text{new}}\}$ 对约束条件的不同轨迹。

B. Implicit Topological Trajectory Generation

<code>\cite{jaillet2008path, zhou2020robust}</code>	<code>???</code>	Jaillet
<code>\cite{jaillet2008path}</code>	Zhou	<code>\cite{zhou2020robust}</code>
VD	VD-visiblity deformation	
“ ”	UVD	
	<code>\cite{zhou2020robust}</code>	UVD
1		
$\$\\tau_1(s) \$ \\tau_2(s)$	$\$s\\in \\left[0,1\\right]$	$\$\\tau_1(0) = \\tau_2(0) \$ \\tau_1(1) = \\tau_2(1)$
$\$s\$$	$\$\\tau_1(s) \\tau_2(s)\$$	UVD

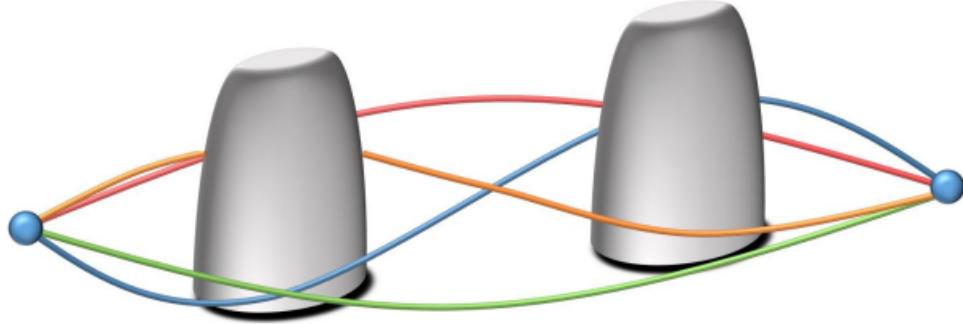


Fig. 5: Four trajectories belonging to the same 3-D homotopy stay in different local minima, which we aim to seek.

```
\cite{jaillet2008path, rosmann2012trajectory, rosmann2017integrated, zhou2020robust}
$\mathbf{v} \quad \mathbf{v}_{\text{new}} := -\mathbf{v}$
$\mathbf{v}_{\text{new}} \quad \mathbf{p}_{\text{new}}$ Def.??? 4.d
$\mathbf{p} \quad \mathbf{p}_{\text{new}}$
```

4

A. Reciprocal Collision Avoidance

$\mathbf{x}_k(t) \in \mathcal{X} \subset \mathbb{R}^3$ $\mathcal{X}_k^{\text{free}}(t) := \mathcal{X} \setminus \{ i \in \mathbb{Z} : \mathbf{x}_i(t) \leq K \}$ <i>\cite{liu2017search}</i>	$\mathbf{\Phi}_k(t) \in \mathcal{X}_k^{\text{free}}(t)$ $\mathbf{\Phi}_k(t) \in \mathcal{X}_k^{\text{free}}(t)$ <i>Sec.III-A</i>	6
--	--	---

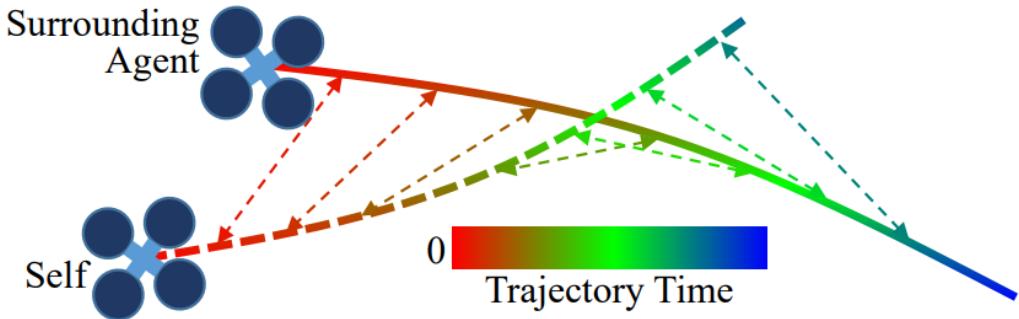


Fig. 6: Drone *Self* generates a trajectory by comparing the distance with a received trajectory from a surrounding drone of the same trajectory time.

$$\begin{aligned}
 & \mathbf{J}_{w,k} \\
 d_{k,i}(t) &= \left\| \mathbf{E}^{1/2} [\mathbf{\Phi}_k(t) - \mathbf{\Phi}_i(t)] \right\| - (\mathcal{C} + \epsilon) \\
 \mathbf{E} &:= \text{diag}(1, 1, 1/c), c > 1
 \end{aligned} \quad \begin{aligned}
 & \mathcal{C} \\
 & \mathbf{J}_{w,k} \quad (1)
 \end{aligned}$$

$$\min_Q J = J_{\text{EGO}} + \lambda_w J_w.$$

$$\Phi(t) = \mathbf{s}(t)^\top \mathbf{M}_{pb+1} \mathbf{q}_m$$

$$\mathbf{s}(t) = [1 \quad s(t) \quad s^2(t) \quad \dots \quad s^{pb}(t)]^\top$$

$$\mathbf{q}_m = [\mathbf{Q}_{m-pb} \quad \mathbf{Q}_{m-pb+1} \quad \mathbf{Q}_{m-pb+2} \quad \dots \quad \mathbf{Q}_m]^\top$$

$$\mathbf{M}\{p_b+1\} \quad p_b \quad t \quad s(t_m, t_{m+1}) \quad s(t)=(t-t_m)/triangle t$$

B. Localization Drift Compensation

Xu \rightarrow cite{xu2020decentralized} UWB

Therefore, inspired by cite{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.

\rightarrow cite{xu2020decentralized} \rightarrow cite{MeiKum1105}

$$\mathbf{S}' \subset \mathbb{R}^3 \quad \mathcal{R} \subset \mathbb{R}^2$$

$$z \begin{bmatrix} \mathbf{s}'^\top & 1 \end{bmatrix}^\top = \mathbf{K} \mathbf{T}_w^c \begin{bmatrix} \mathbf{s}^\top & 1 \end{bmatrix}^\top,$$

$$\mathbf{s}' \in \mathcal{S}' \quad \mathbf{s} \in \mathcal{S}$$

$$\mathbf{P} \in \mathcal{P} \subset \mathcal{S}$$

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

$$\mathbf{P} \in \mathcal{P} \subset \mathcal{S}$$

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

C. Agent Removal from Depth Images

\rightarrow VIO \rightarrow cite{zhou2019robust}

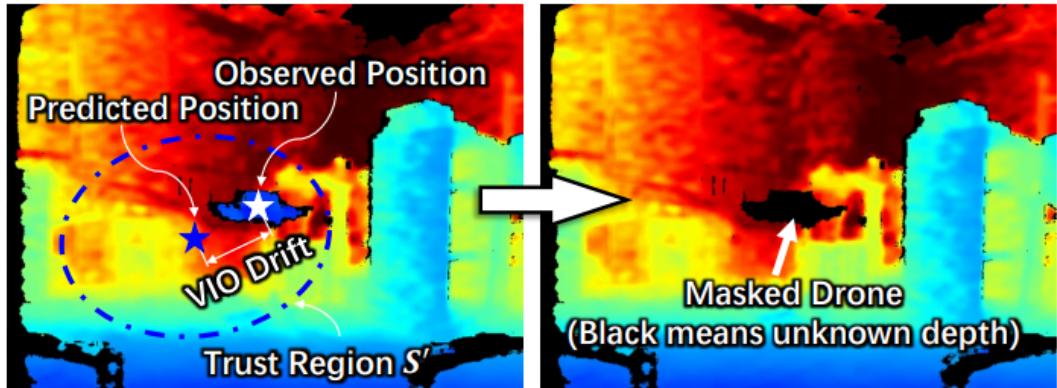


Fig. 7: 左侧:在我们使用的定位方法VIO中，通过在可信区域内比较从接收到的轨迹预测的位置与深度图中观测到的位置，估计浮动情况。右侧:为避免对地图建图产生影响，观测到的代理被屏蔽处理。

5 System Architecture

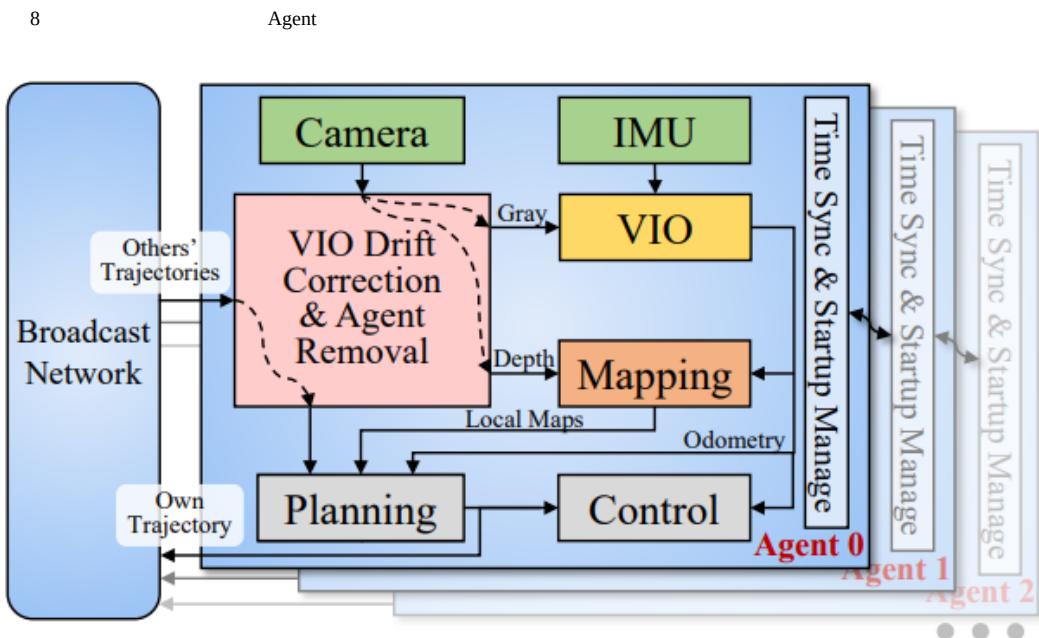


Fig. 8: System Architecture

A. Navigation System of A Single Agent

EGO-Planner \cite{zhou2020ego}

VIO

B. Communication Framework

1. Broadcast Network

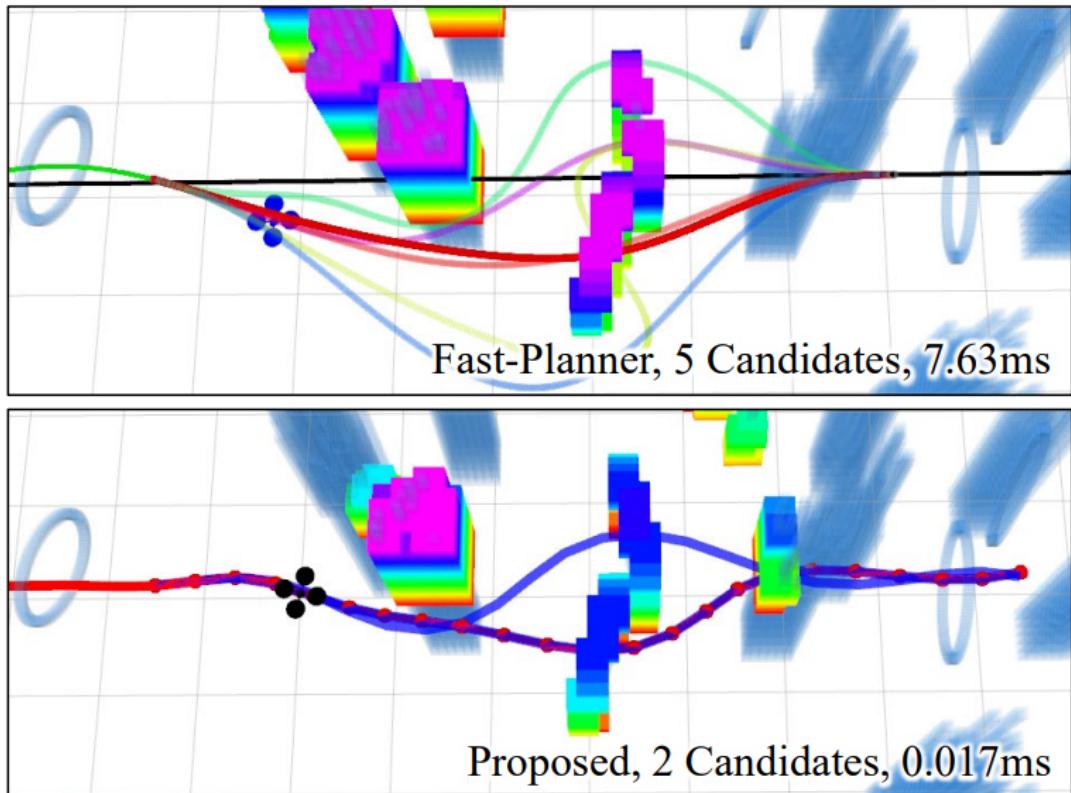
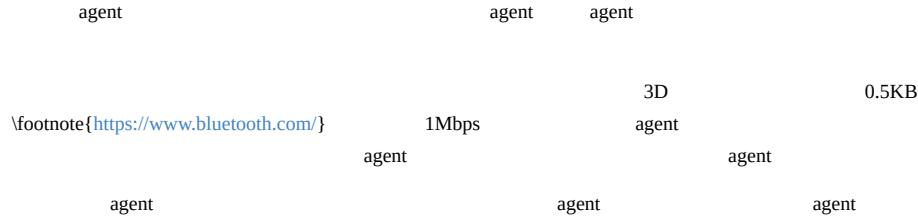


Fig. 9: 前端拓扑路径搜索的比较。所提出的规划器找到的候选拓扑不同轨迹较少，但相比快速规划器，计算量较少。需要注意的是，该时间仅用于前端局部最小值发现，而显示的轨迹是为了更好的可视化而优化的。

2 Chain Network

6
i7-9700KF CPU 7.5m
 $\$ \lambda_s=1.0, \lambda_c=\lambda_w=\lambda_t=0.5, \lambda_d=0.1 \$$ 0.1m

A. Topological Planning

EGO-Swarm Fast-Planner^{cite{zhou2020robust}}

9

EGO-Swarm

\cite{zhou2020robust} 100

Fast-Planner

PRM^{cite{kavraki1996probabilistic}}

B. Swarm Planning

1. In Empty Space

DMPC^{cite{luis2019trajectory}} ORCA^{cite{van2011reciprocalnbody}} RBP^{cite{park2020efficient}}
 $(\$d\{\rm{fly}\})\$$ $(\$t\{\rm{fly}\})\$$ $(\$t_{\rm{cal}}\$)$
 $\backslash\text{ref}\{\text{pic:multipcomp}\}$ $\text{Tab.} \textcolor{blue}{???$ $\$t\{\rm{cal}\}\$ \text{ } \ast$
 DMPC RBP ORCA EGO-Swarm

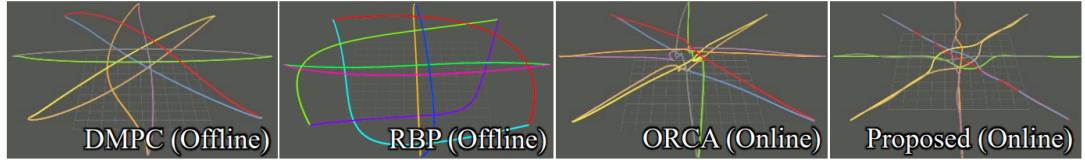


Fig. 10: 各种群组规划器在无障碍情况下的计划轨迹。

Tab.??? 10 RBP \cite{park2020efficient}

DMPC

ORCA

1. In Obstacle-rich Environments

2 0.2 2 0.42 /\$m^2\$

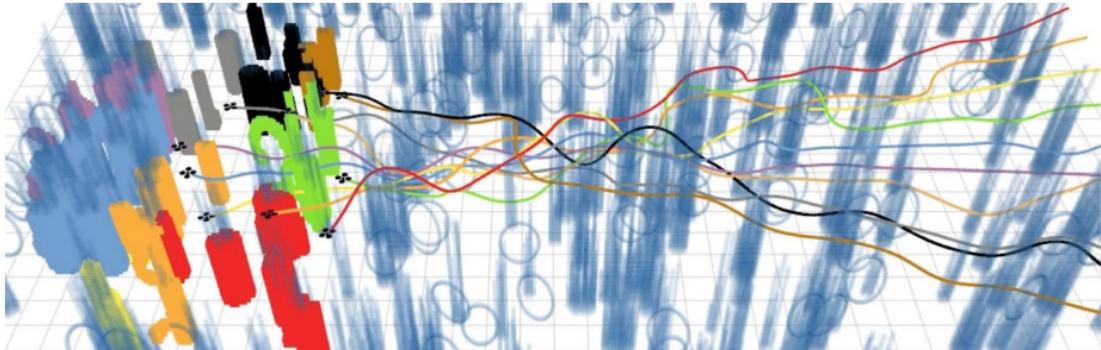


Fig. 2: 十个四旋翼无人机在模拟中飞行。有色方块代表四旋翼的局部感知。每种颜色与一个四旋翼关联。

2 $\$d\{\rm{fly}\}\$$ $\$d\{\rm{safe}\}\$$

TABLE II: EGO-Swarm在模拟不同障碍密度下的性能

Density(obs/m ²)	$d_{\text{fly}}(\text{m})$	vel.(m/s)	$d_{\text{safe}}(\text{m})$	$t_{\text{cal}}(\text{ms})$
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

50 11 5.B.1

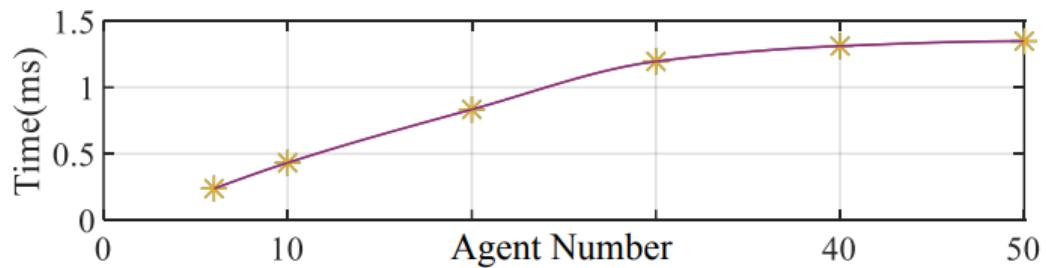


Fig. 11: The relationship between the computation time and the number of Agents in empty space.

7 Real-World exp

1 Indoor

1.5 / 12

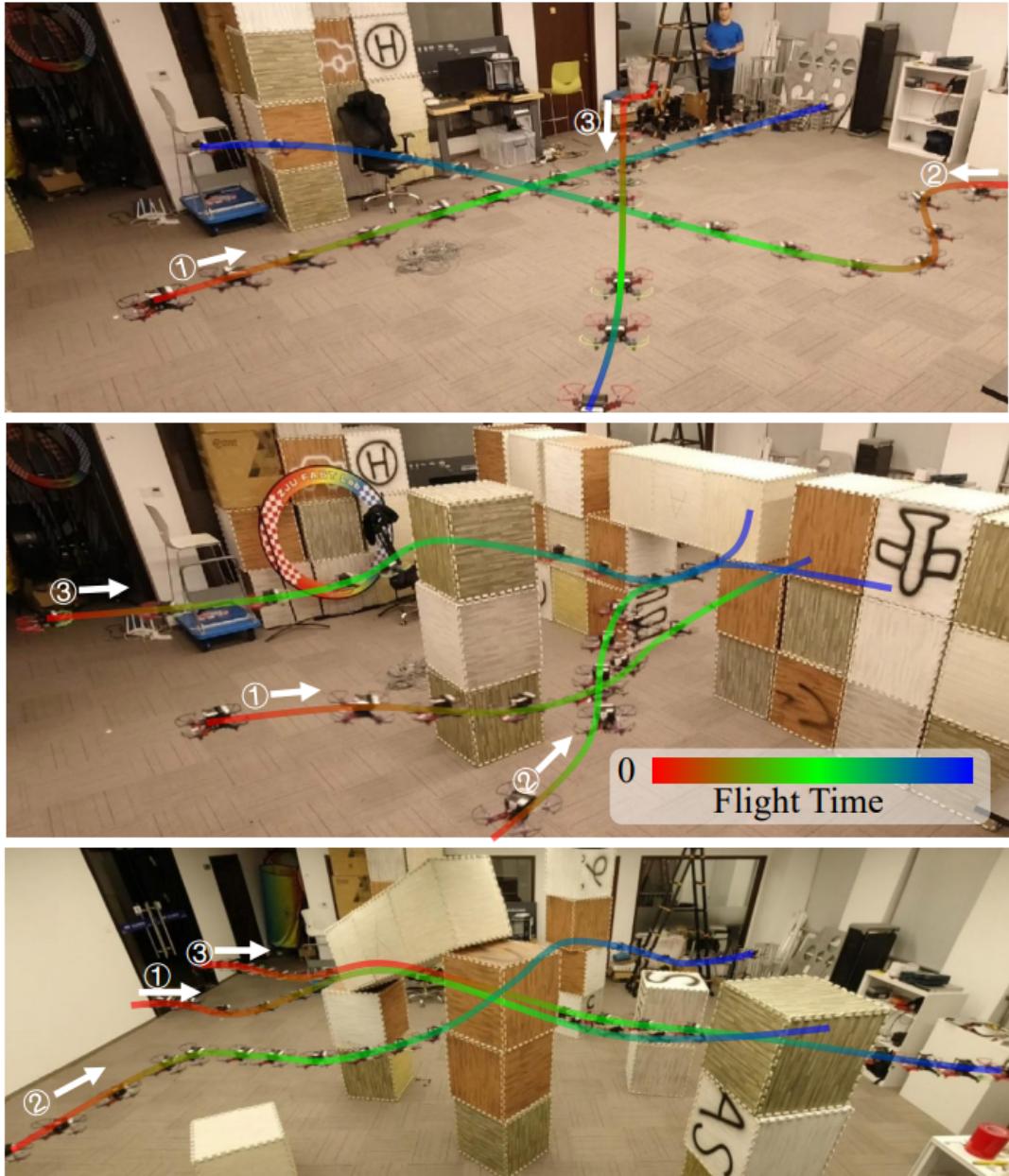


Fig. 12: Indoor Experiments

2

1

1.5 /

???

github

\footnote{\url{https://github.com/ZJU-FAST-Lab/ego-planner-swarm}}



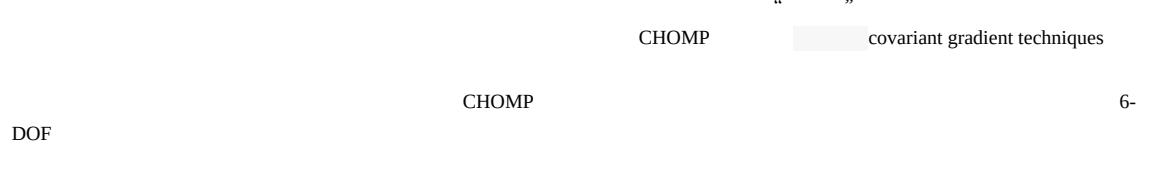
Fig. 1: 三架四旋翼飞行器在森林中的三个快照。

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

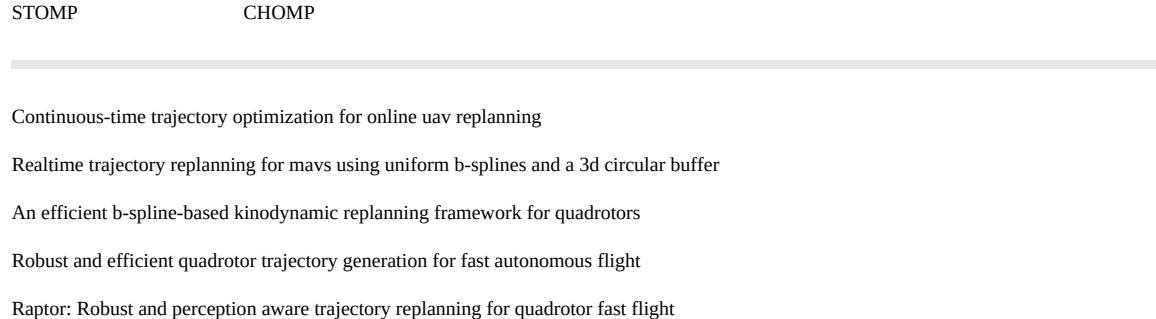
CHOMP

CHOMP: Gradient Optimization Techniques for Efficient Motion Planning



Stomp

Stomp: Stochastic trajectory optimization for motion planning



Swarm of micro flying robots in the wild

Science Robotics

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

1.

2.

GPS

3.

pixel gathering boundingbox

4.

1.

o

o

2.

i.

ii. Bezier B-Spline convex hull

iii. MINCO

3.

" "

" "

4.

5.

65m

10

100

1000

50%

-2023

Robust and Efficient Trajectory Planning for Formation Flight in Dense Environments

IEEE Transactions on Robotics

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

(differentiable graph-based metric)



PAPER

- Portability
- Adaptability
- Predictability
- Elasticity
- Resilience

PAPER

\cite{quan2022formation} \textit{PAPER}

\textit{PAPER} [4]

Micro Flying Robots-21

formation position sequence

\cite{quan2022formation}

$\sim \text{cite}\{\text{quan2022formation}\}$

ALAS

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

Department of Computer Engineering, Chosun University, Gwangju 61452, South Korea

61452

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

Abstract

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.

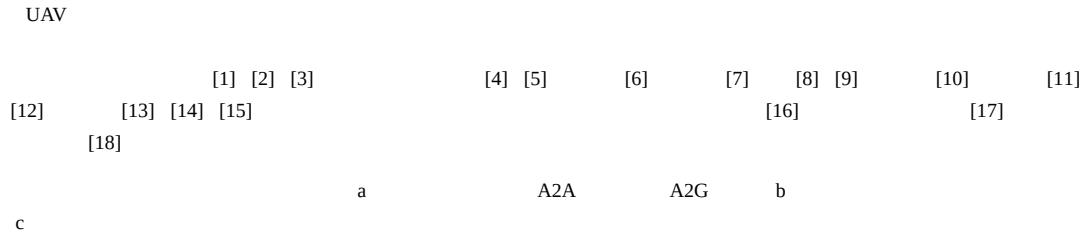
UAV

Keywords

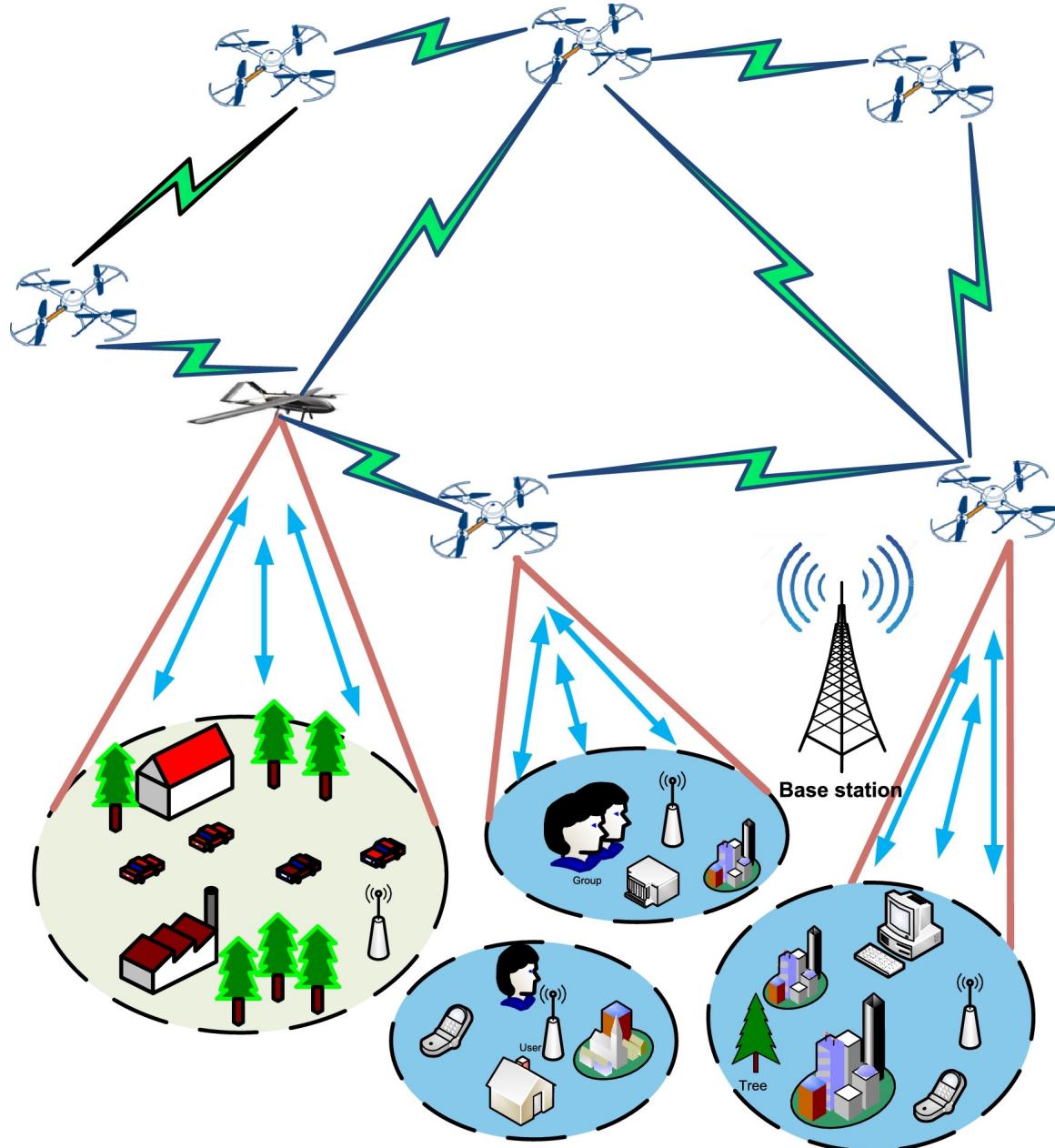
- Unmanned aerial vehicle; Multi-UAV networks; UAV autonomy; Task assignment; Task coordination; Heterogeneity;
-
- Introduction
 - 1. Challenges
 - 2. Motivation
 - 3. Contribution of the study
 - 4. Organization of the paper
- Overview of existing surveys on UAVs and our contributions
- Considerations 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.

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.



1



1.1. Challenges of UAV networks

1.1. 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].

[22]	UAVs [19] [20] [21]	[23]	[24] [25] [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].

[28] [29] [30]

[31]

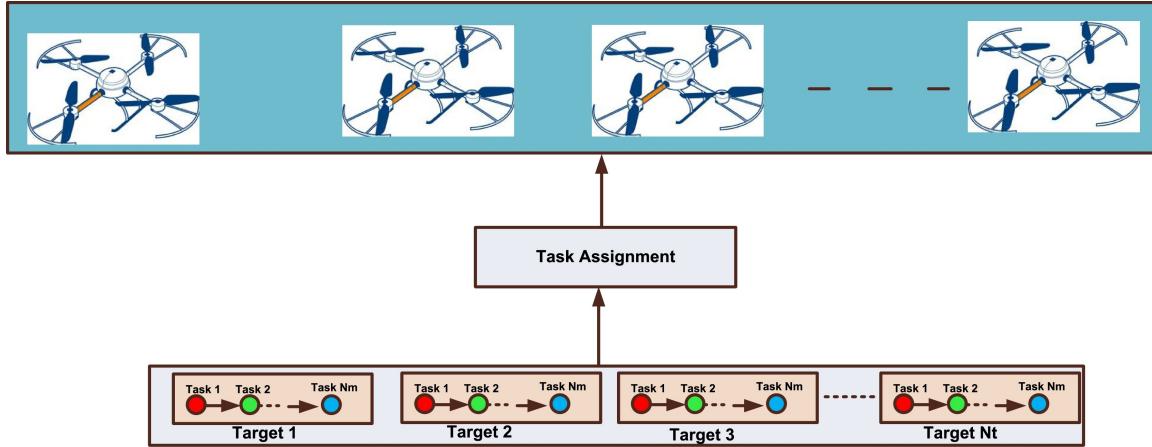
[32]

1.2. Motivation 1.2.

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.

[33]

2



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.

[34] [35]

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.

1.3. Contribution of the study

1.3.

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.

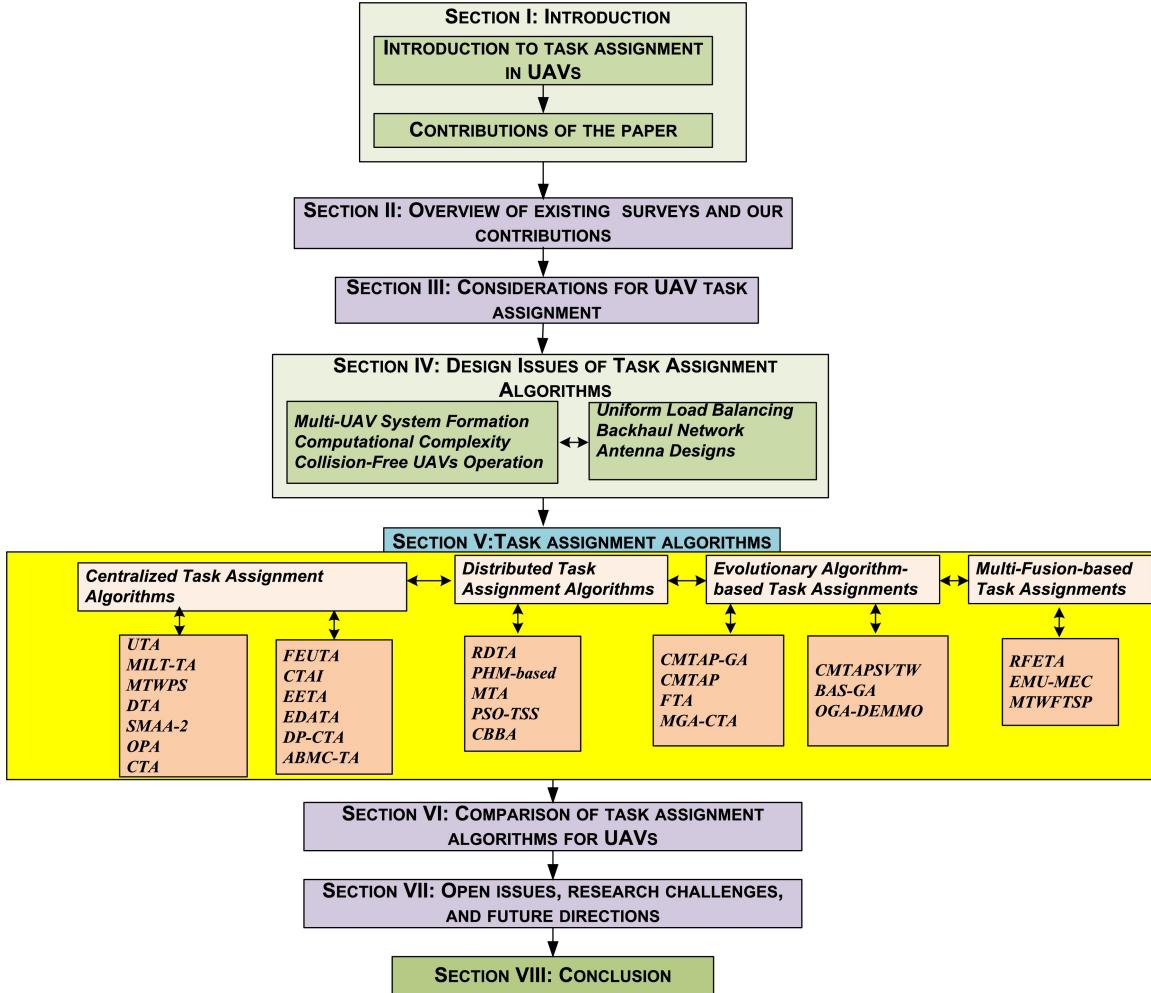
- Based on their main ideas, existing surveys on UAVs and their applications are reviewed.
 -
- The motivation behind the study of task assignment algorithms is elaborated on along with design considerations.
 -
- Existing task assignment optimizations for UAV networks are classified and compared based 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.
 -
- State-of-the-art task assignment optimizations are extensively compared and contrasted in terms of their principles, features, advantages, and limitations.
 -
- Task assignment optimizations are qualitatively compared with various features and characteristics with a comprehensive and comparative discussion.
 -
- Finally, critical open issues and research challenges faced during task assignments in multi-UAV systems are summarized and deliberated for further enhancement and improvement.
 -

1.4. Organization of the paper

1.4.

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.

- 2
- 3
- 4
- 4 5
- 6
- 7



2. Overview of existing surveys on UAVs and our contributions

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.

Table 1. Summary of existing surveys.

Reference	Year	Description	Classification of UAVs	Challenges	Topic discussed	Intelligent decision	IoT
[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	✗	✗
[44]	2016	UAVs and UAVs related issues are discussed	✓	✓	UAV based IoTs	✗	✗

Reference	Year	Description	Classification of UAVs	Challenges	Topic discussed	Intelligent decision	IoT
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].

[36]

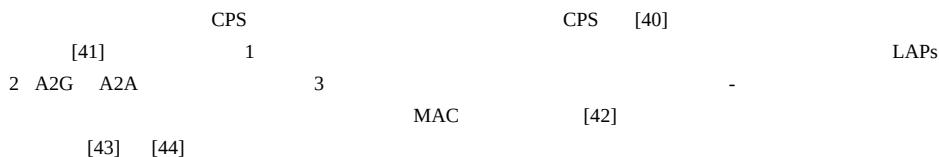
IoT

[37]

[38]

[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].



3. Considerations for UAV task assignment

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.

BS	LoS	[45]	[46]
2018	2023	TechSci	[47]
690	1410		

3.1. Single- and multi-UAV systems

3.1.

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.

[48] [49]

[50]

[51] 2

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.

UAV

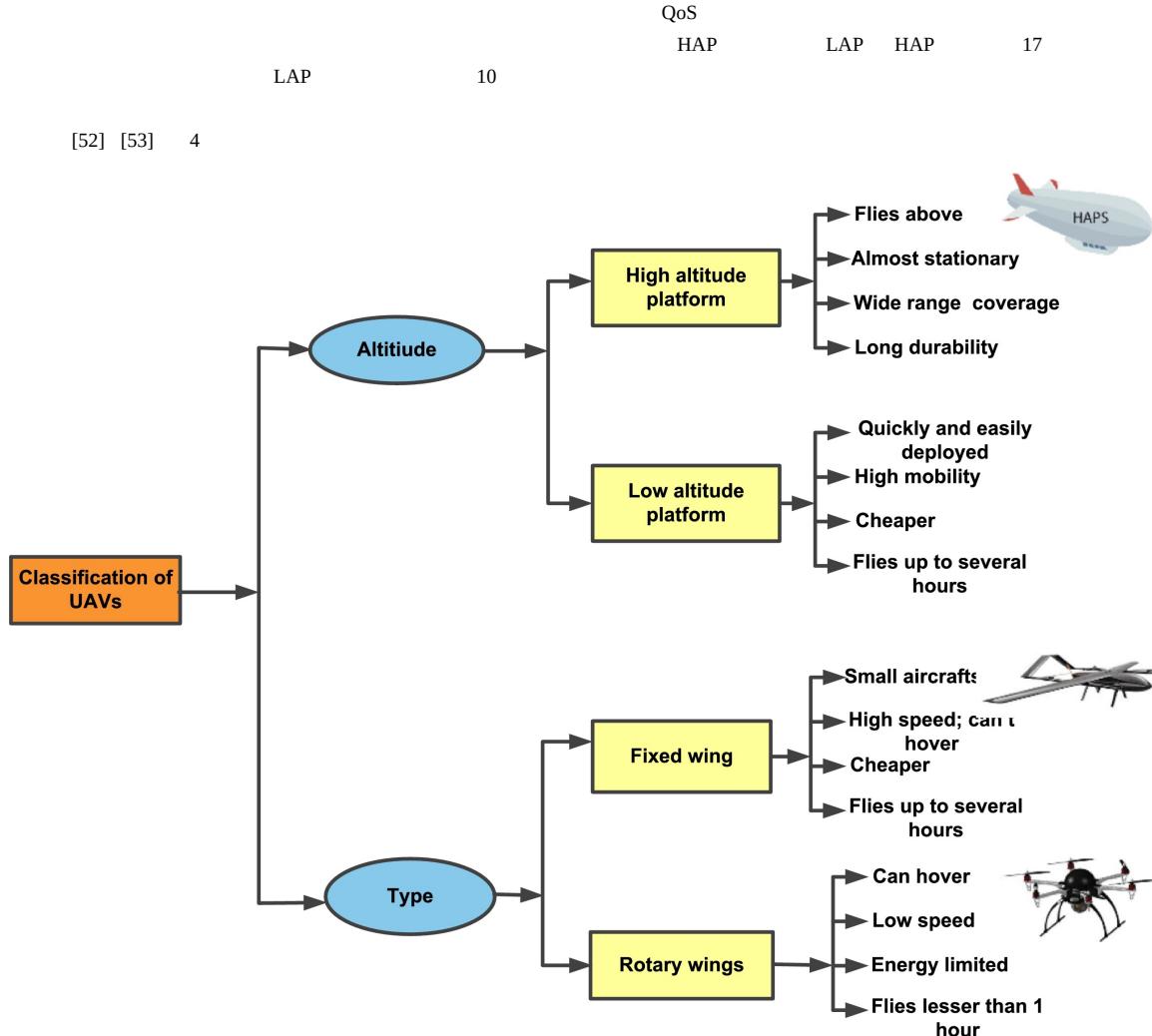
UAV

3.2. Classification of UAVs

3.2.

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.



3.3. Background study 3.3.

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.

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].

MILP [57]	WTA [58]	[54]	TSP [55]	[56]
RL [61]	PSO [62]		GA [59]	ACO [60]

[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.

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

3

Table 3. List of notations and abbreviations.

Symbol	Description
\$N_v\$	Number of UAVs
\$N_t\$	Number of targets
\$X^{k_{\{i,j\}}}\$	Decision variable in the range of 0–1
\$k\$	Order of UAV
\$L^{k_{\{i,j\}}}\$	Shortest path length
\$f^{k_{\{i,j\}}}\$	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
\$\lambda\$	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
QoS	Quality of service
HAPs	High-altitude platforms
MEC	Mobile edge computing
SO	System orchestrator

Symbol	Description
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

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.

4.1. Multi-UAV system formation

4.1.

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].

[66]

[67]

[66]

[67]

4.2. Computational complexity

4.2.

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].

[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].

[70] [71]

4.3. Collision-free UAV operation

4.3.

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](#).

UAV

[72]

[73]

[74]

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](#).



4.4. Backhaul network 4.4.

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].



4.5. Uniform load balancing

4.5.

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].



4.6. Antenna designs 4.6.

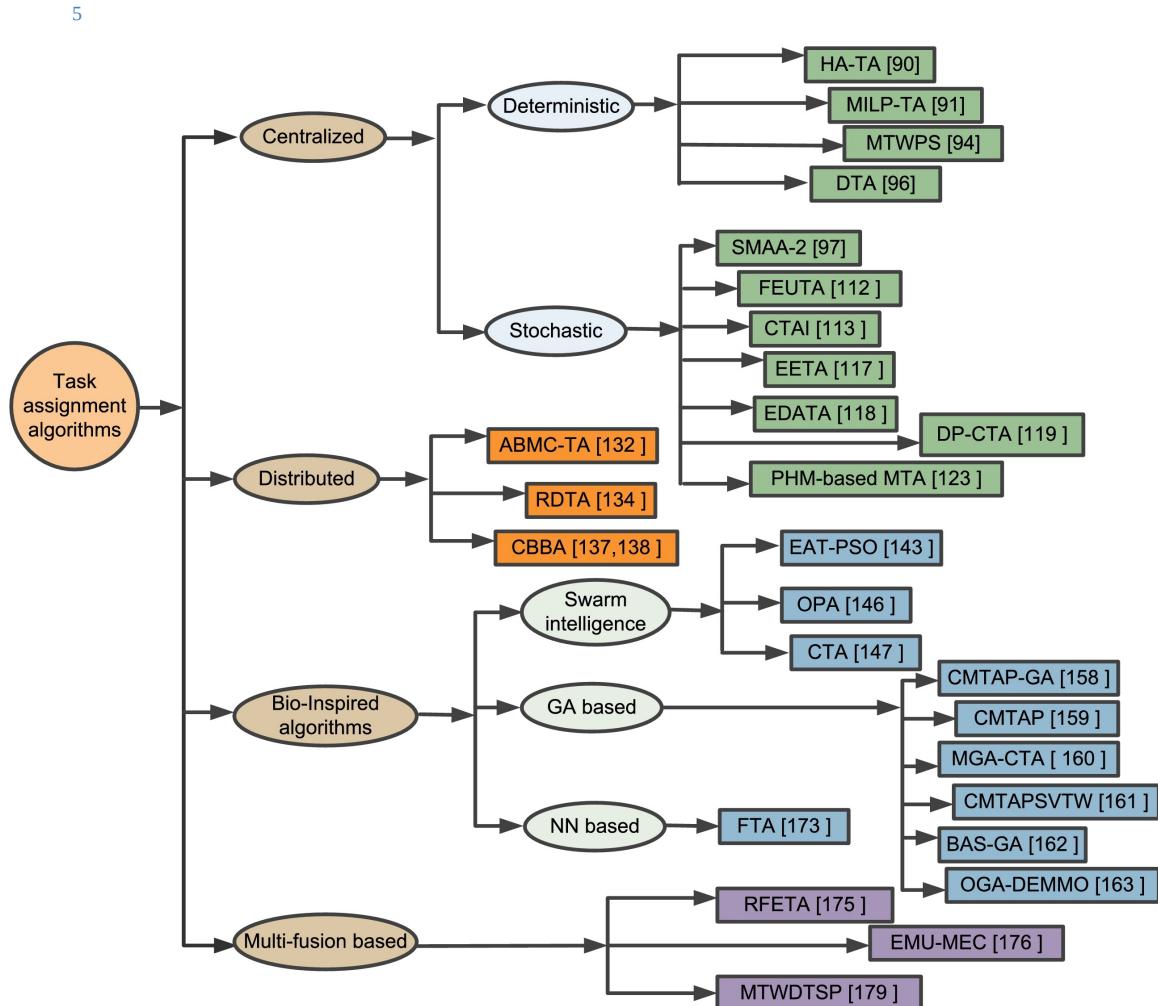
The design and number of antennas also determine the performance of UAVs, and hence they must be decided cautiously. UAVs essentially incorporate directional and [omnidirectional antennas](#). In short, omnidirectional antennas perform better in environments with high mobility but face energy drain and 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.

[85] [86]

5. Task assignment algorithms

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.



5.1. Centralized task assignment algorithms

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 [\[87\]](#), [\[88\]](#), [\[89\]](#).

[\[87\]](#) [\[88\]](#) [\[89\]](#)

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

5.1.1.1. (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.
- [\[90\]](#)
-
-
-
-

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.

- Advantages: Improves operational reliabilities and capabilities through [adaptive task assignment systems](#) and [system awareness](#); demonstrated effective performance through simulation and experiments.
-
- 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.
-
-
- 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.
-

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.

[\[91\]](#)

MILP

MILP

[\[92\]](#) [\[93\]](#)

- Advantages: Useful in offline task assignment calculations; provides optimal solution for UAV groups with combined tasks with timing and task order limitations.
-
- 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.
-
- 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.
-

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

5.1.3. (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	[94]	MTWPS
-------	------	-------

- 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.
-
- 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.
-
- 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.
-

5.1.4. Dynamic Task Assignment (DTA)

5.1.4. (DTA)

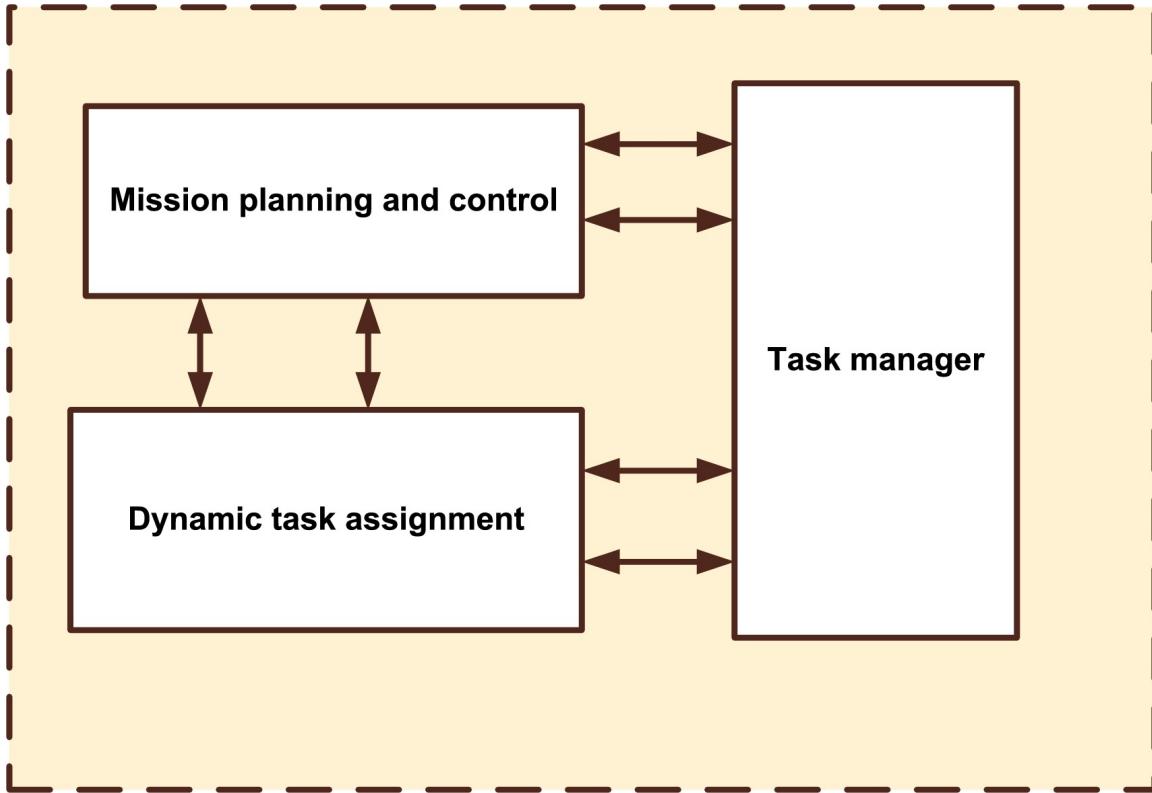
A mission planning and task assignment framework for the coordination of self-controlled UAVs to improve the robustness and scalability of multi-UAV operations was studied in [96]. An outline of the proposed framework is shown in Fig. 6. The main goal of the battlefield scenario is to eliminate enemy 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.

[96]

n

6

`agent_id`



- Advantages: Time efficiency, scalability, and robustness are achieved; achieves the mission goal with minimum cost.
-
- Limitations: Does not consider the uncertainties and aspects of dynamic environments; preliminary information regarding the size and location of enemy UAVs is unknown.
-
- Possible enhancements: Minimizing the combination of all costs is desired, requiring a solver algorithm; threats and uncertainties of the complex dynamic environments need consideration.
-

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

5.1.5 (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.

[97]

[98] [99] [100] [101] [102] [103] [104] [105] [106]

SMAA [107] [108] [109] [110] [111]

- Advantages: Uncertainties are analyzed; finds satisfactory solution even in severe situations.
-
- Disadvantages: Sensor error or target movement leads to unclear localization; probability of UAVs being destroyed while handling attacks remains.
-
- 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.

•

5.1.6. Filter-Embedded UAV Task Assignment (FEUTA)

5.1.6 (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.

SA [112]
FEUTA

- Advantages: Condenses the consequence of noise caused due to high-frequency on the planner; mitigates the effects of noise.
 -
 - Limitations: Ignores the aspects such as threats and uncertainties, which makes it unreliable for the real environment; accepts noise entirely.
 -
 - Possible enhancements: The importance of robust methods proposed for FEUTA and their connection to noise [rejection algorithms](#) must be investigated.
 -
- FEUTA

5.1.7. Coordinated Target Assignment and Intercept (CTAI)

5.1.7. 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.

[113]

[54] [114] [115] [116]

TOT TOT

- Advantages: Decomposition of motion planning issues helps obtain a near-real-time solution; risk is mitigated by increasing distance between UAV and threats.
-
- Limitations: Tracking error increases due to disturbances; only static threats are considered, and thus the results obtained are suboptimal.
-
- Possible enhancements: Timing of target intersection can be considered; aspects of the dynamic, complex 3D environment such as [wind speed](#), risks need consideration.
-

3D

5.1.8. Energy-Efficient Task Assignment (EETA)

5.1.8 (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.

MEC [117]

BSUM

- Advantages: Lessens energy depletion, and optimizes [task offloading](#), resource sharing, and trajectory without increasing delay; minimizes UAV's propulsion and computation power.
-
- Limitations: Single UAV is used, which increases the risk of mission failure.
-
- 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.
-

5.1.9. Energy and Delay Aware Task Assignment (EDATA)

5.1.9. (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] SO VAIoTS SO (EAUS) SO
SO (FTUS) EAUS DAUS FTUS (DAUS)

- Advantages: Energy consumption, operational time, and fairness of the system are optimized jointly; environmental effects on flights and performance are considered.
-
- Limitations: [GPS localization errors](#) and signal losses exist; increase in [execution time](#) for FTUS, which impacts the performance of the model.
- GPS FTUS
- 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.
- GPS

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

5.1.10. 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] DP-CTA

[120] [121] [122] DP-CTA

DP

-
-

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

- Advantages: Generates cooperative and optimal solution quickly; recovers planning of real-world air operations providing improvements.
-
- Limitations: Computation time grows with the number of targets, and the uncertainties of real complex environment are ignored.
-
- Possible enhancements: Constraints of the real dynamic environment such as sudden threats, effects of wind, and temperature on UAV flights must be focused on.
-

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

5.1.11.

(PHM)

PHM 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]

PHM

(IVHM)

IVHM

RUL

PHM

(RUL)

RUL

RUL

(FTA)

FTA

- Advantages: Applies system-level RUL methodology to accomplish task assignment including failure probability.
- RUL
- Limitations: It may not be feasible to know the probability of failure in advance; threats and uncertainties of real stochastic environments are ignored.
-
- 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.
- PHM TA

5.2. Distributed task assignment algorithms

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.

[\[124\]](#) [\[125\]](#) [\[126\]](#) [\[127\]](#)

[\[128\]](#) [\[129\]](#) [\[130\]](#) [\[131\]](#)

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

5.2.1.1. 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.

[\[132\]](#)

(C) (F) (P) (R)

[\[133\]](#)

- Advantages: Solves multi-constraint task assignment issues efficiently considering the constraints of sensors, time, danger, and fuel.
-
- 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.
-
- 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.
-

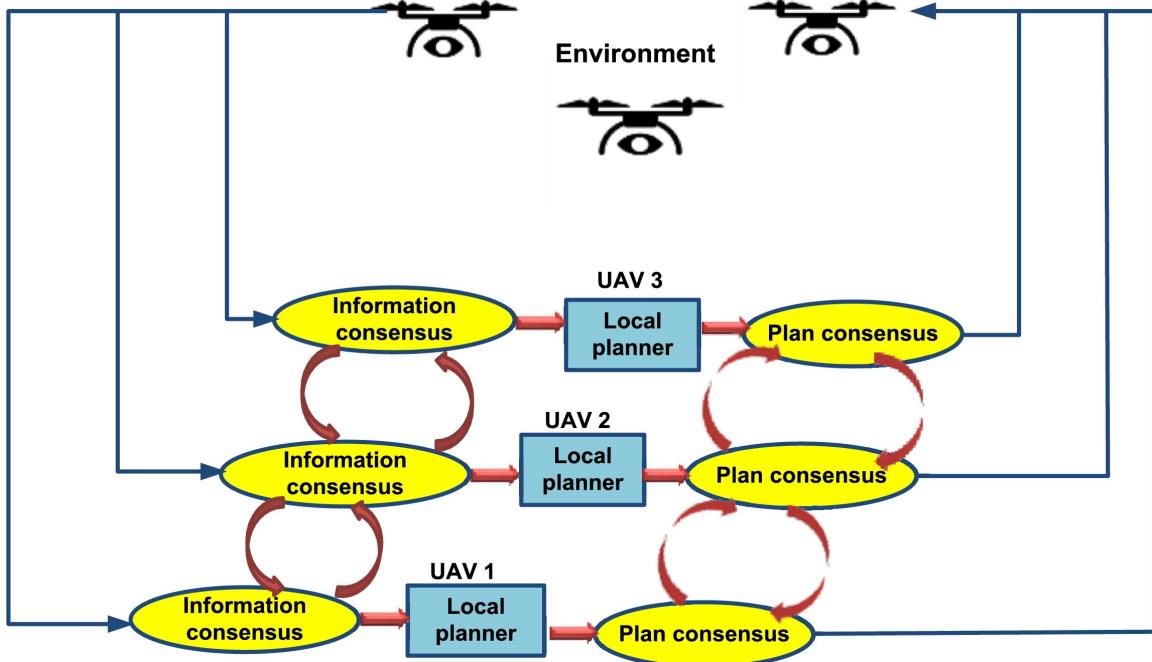
3D

5.2.2. Robust Decentralized Task Assignment (RDTA)

5.2.2.1. 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.

-
-
-
- [7](#)
-
- RDTA [\[135\]](#)
-
- [\[136\]](#)
-



- Advantages: “No Fly Zones” are considered; a modified candidate plan [selection algorithm](#) is used to improve performance and increase the robustness to the inconsistencies.
- “ ”
- 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.
-
- 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.
-

5.2.3. Consensus-based Bundle Algorithm (CBBA)

5.2.3.

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]

CBBA

CBBA

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

\$x_{ij}=1\$ \$UAV_i\$ \$j\$ \$x_i \in \{0,1\}^{\wedge N_t}\$ \$j\$ \$x_{ij}\$, \$c_{ij}(x_i, p_i)\$ \$x_i\$
\$\\$p\\$ [139] CBBA

- Advantages: Handles operational complexities of multi-UAVs; provides reliable, conflict-free UAV task assignments.
-
- Limitations: All targets may not be discovered; dynamic threats and uncertainties are ignored.
-

- Possible enhancements: Kinematic and dynamic UAV constraints on air and ground can be incorporated; UAVs need to be penalized for massive heading changes.
-

5.3. Bio-inspired task assignment algorithms

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.

ANN

[\[140\]](#) [\[141\]](#) [\[142\]](#)

Swarm intelligence-based task assignment

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.

ACO [60] PSO [62]

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

5.3.1. PSO (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].

[\[143\]](#)

EAT PSO

[\[144\]](#)

[\[145\]](#)

PSO

- Advantages: Achieves a minimum total make span, forms a seamless UAV operation schedule, and short computation time in finding a high-quality [feasible solution](#).
-
- Limitations: Allows a single UAV to occupy a position at a particular time, ignoring uncertain events and obstacles.
-
- 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.
-

5.3.2. Orchard Picking Algorithms (OPA)

5.3.2.1. (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 and denote the sets of targets and UAVs, respectively, and, where is the number of types of missiles, the total number of type missiles for targets to be destroyed is given as

[146]

$$\begin{array}{ccc} & D & \\ \text{T} & \text{U} & \text{A} & \text{V} \end{array}$$

- 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.
-
- Limitations: Does not consider the constraints of the actual battlefield; threats and uncertainties that may arise in the dynamic environment are ignored.
-
- Possible enhancements: Adaptive approach of cooperative task assignment can be introduced to balance the load of UAVs and adapt to real-time battlefield situations.
-

5.3.3. Cooperative Task Assignment (CTA)

5.3.3.1. 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]

$$\begin{array}{ccccc} & \text{ACO} & & & \\ \$x_{i,j} \in [0,1]\$ & \$x_{i,j}=1\$ & & \$x_{i,j}=0\$ & [148] \quad \text{K-means} \end{array}$$

- Advantages: Efficiently solves considerate task assignment problems of UAVs under multiple circumstances and constraints; fast convergence and a highly stable iterative curve are assured.
-
- 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.
-
- 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.
-

GA-based task assignment GA

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].

[149] [150] [151] [152]

[153]

GA

GA

[154] [155] [156] [157]

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

5.3.4.

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]

CMTAP

- Advantages: Promptly offers feasible task assignment solutions and real-time implementation for high dimensions; considers task priority and coordination, time constraints, and trajectory.
-
- 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.
-
- 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.
-

5.3.5. Cooperative Multiple Task Assignment Problem (CMTAP)

5.3.5

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.

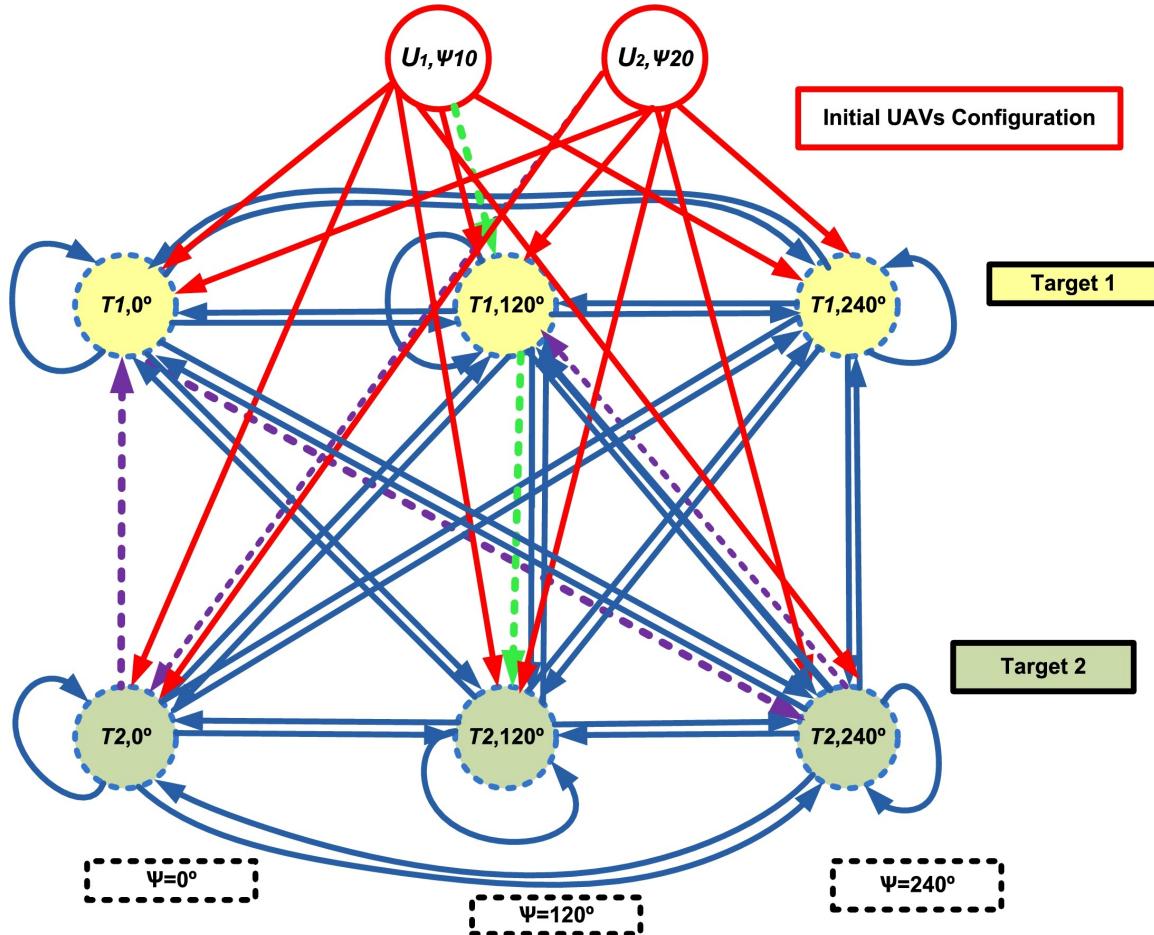
[159]

GA

1

Fig. 8 presents a graph representation of the CMTAP solution with two UAVs, $\{U_1, U_2\}$ and two targets $\{T_1, T_2\}$ to perform three tasks $N=3$. The green line indicates the path of the first UAV, U_1 , starting from the initial point, visiting the target, 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 $\psi=240^\circ$.

8 CMTAP , $\{U_1, U_2\}$, $\{T_1, T_2\}$
 $N=3$, U_1 , , T_1 , , $\psi=120^\circ$, ,
 U_2 T_2 $\psi=0^\circ$ T_1 , T_2 $\psi=240^\circ$, T_1 $\psi=240^\circ$



- Advantages: Solves computational complexity for classical combinatorial optimization methods with higher efficiency and considers task priority and timing constraints.
 -
 - 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.
 -
 - Possible enhancements: Moving, sudden targets, and actual environmental scenarios must be considered; computational efficiency needs to be realized to achieve a real-time solution.

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

5.3.6 GA (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]

- Advantages: Considers and addresses the heterogeneity of UAVs, targets, and environments; mirror representation to deal with the limited resources.
 -

- Limitations: Only one kind of resource, i.e., weapon, is considered; stationary targets considered in simulation, weapons are used in one way only.
-
- 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.
-

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

5.3.7. (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.

[161]

Dubins

- Advantages: Considers kinematic constraints (motion, external forces, propulsion), resource constraints (fuel and onboard weapons), and time constraints (time windows and sequence of tasks).
-
- Limitations: Does not cover practical mission scenarios such as weather constraints, radar, terrain barriers during simulation.
-
- 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.
-

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

5.3.8. GA (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]

GA

BAS

BAS

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.
-
- Limitations: Does not consider the kinematics of UAV which impacts on performance and constraints of the realistic dynamic environment such as threats and uncertainties.
-
- 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.
-

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

5.3.9.

(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]

OGA-DEM MO

OGA-DEM MO

- Advantages: Minimizes the execution time and total UAV consumption and improves the [optimality](#) of the algorithm and convergence energy.
- 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.
- Possible enhancements: Constraints of natural and dynamic environments such as threats, uncertainties, and other external forces can be considered.
-

Artificial Neural Network (ANN)-based Task Assignment

(ANN)

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].

ANN	DRL [164] [165] DNN [166] SOM [167] DQN [168]	ANN
	[169] [170] [171] [172]	

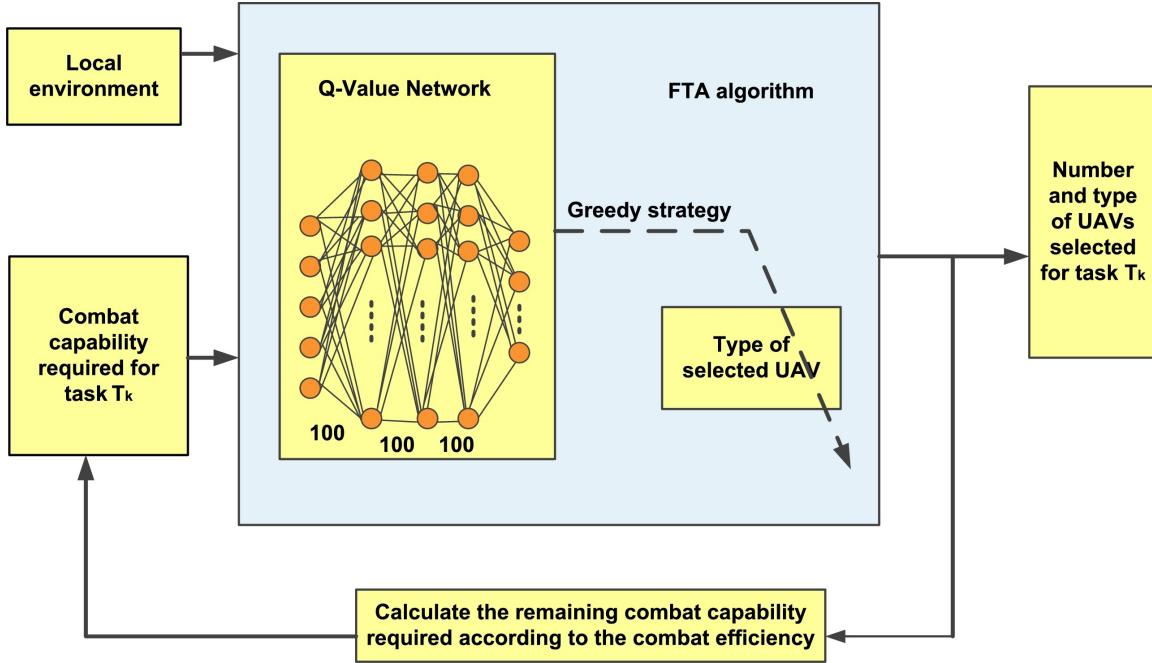
5.3.10. Fast Task Assignment (FTA)

5.3.10. (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(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 Tk. Environmental uncertainty is denoted as \$envk = {w k, r k}\$, where w k represents wind speed, and r k represents rainfall. In addition, UAVs are denoted as \$U=\left\{U_1, U_2, \dots, U_m\right\}\$, and \$\left\{a t t(U_j), \operatorname{operatorname}{def}(U_j), \operatorname{operatorname}{ele}(U_j)\right\}\$ 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]	Q	NN	R L
\$T={T 1, T 2, \dots, T N}\$,	\$\{\operatorname{operatorname}{att}(T k), \operatorname{operatorname}{def}(T k), \operatorname{operatorname}{ele}(T k)\}\$	
\$T k\$			
		\$e n v k = {w k, r k}\$	
		\$w k\$	\$r k\$
		\$U=\left\{U_1, U_2, \dots, U_m\right\}\$	
		\$\left\{a t t(U_j), \operatorname{operatorname}{def}(U_j), \operatorname{operatorname}{ele}(U_j)\right\}\$	
		\$j\$	
	Q	9	



- Advantages: Computationally fast and efficient, highly adaptive, and handles uncertainty; deals with different types of tasks.
-
- Disadvantages: Considers only stationary targets; complexities and aspects of the real battlefield environment are ignored.
-
- Possible enhancements: Moving and sudden targets and complexities of real battlefield environmental scenarios must be considered.
-

5.4. Multi-fusion-based task assignments

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.

GA ACO PSO

[174]

5.4.1. Robust Filter Embedded Task Assignment (RFETA)

5.4.1. (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]

FETA

-
-
-

RFETA

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

$$\underline{\lambda}_k \circ \sigma_k = \lambda_k + \sigma_k - \lambda_k \sigma_k$$

- Advantages: Presents robust results in uncertain environments; sensitive to errors; minimizes the effects of noise; yields fast convergence, and reduces churning.
-
- Limitations: It does not perform better if lots of information must be considered while planning.
-
- 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.
- RFETA λ

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

5.4.2. (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.

MEC [176]

BCD
[177] [178] MEC

- Advantages: Total energy consumption of sensor devices and UAVs is minimized.
-
- Limitations: Inter-cell interferences from mobile devices of other UAVs are experienced.
-
- Possible enhancements: UAV trajectory, communication, and computation resource assignment can be investigated.
-

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

5.4.3. (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]

MTWDTSP

[180] [181]

3D 2D

i

ii

iii

Dubins
iv

- Advantages: Improved convergence speed and efficiency; provides optimized task sequence.
-
- Limitations: Complexities increases with the number of UAVs and tasks; not all tasks are executed.
-

- 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.

•

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.

[4](#)

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.
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.

Protocol	Advantages	Limitations	Possible enhancements
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.
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.
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.

Protocol	Advantages	Limitations	Possible enhancements
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.
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-DEMIMO [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.
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.

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.

6. Comparison of task assignment algorithms for UAVs

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.

27

GA RHTA MILP DP

4

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.

5

MATLAB

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
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

Protocol	Main idea	Design approach
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-DEMMO [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
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.

6 6

Table 6. Comparison of task assignment algorithms in terms of key features and characteristics. 6 .

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	-
OGA-DEMMO [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). 6 .

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
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

Protocol	Year	UAV type	Task assignment type	Operational complexity	Performance evaluation tool
CMTAPSVTW [161]	2018	Heterogeneous	Bio-inspired	Low	MATLAB
BAS-GA [162]	2020	Homogeneous	Bio-inspired	Low	-
OGA-DEMIMO [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.

Table 7. Qualitative comparison of task assignment algorithms in terms of performance. 7 .

Protocol	Energy consumption	Latency	Efficiency	Adaptability	Scalability	Load balancing
HA-TA [90]	Low	Low	High	High	–	No
MILP-TA [91]	–	High	High	No	No	No
MTWPS [94]	No	Low	High	No	No	No
DTA [96]	Low	Low	High	High	High	–
SMAA-2 [97]	No	No	Improved	No	No	No
FEUTA [112]	–	Low	Improved	No	No	No
CTAI [113]	–	Low	High	No	No	No
EETA [117]	Low	Satisfactory	High	No	No	No
EDATA [118]	Low	Low	High	No	No	No
DP-CTA [119]	No	Low	Improved	High	–	No
PHM-based MTA [123]	Low	Low	High	No	No	No
ABMC-TA [132]	Low	Low	High	–	–	No
RDTA [134]	–	Low	High	High	Low	No
CBBA [137], [138]	No	Low	High	–	–	No
EAT-PSO [143]	Recharging	Low	High	No	Yes	No
available						
OPA [146]	No	Low	High	High	High	Yes
CTA [147]	No	Low	Improved	No	No	No
CTMAP-GA [158]	Low	Low	High	High	High	No
CMTAP [159]	Low	Low	Improved	High	High	No
MGA-CTA [160]	Low	Low	Improved	No	–	No
CMTAPSVTW[161]	Low	Low	High	High	High	No
BAS-GA [162]	Low	Low	High	No	Yes	No
OGA-DEMMO [163]	Low	Low	High	No	No	No
FTA [173]	No	Low	High	High	No	No
RFETA [175]	No	Low	High	High	High	No
EMU-MEC [176]	Low	Low	High	–	–	No
MTWDTSP [179]	Low	Low	High	No	No	No

7. Open issues, research challenges, and future directions

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.

7.1. UAV regulations 7.1.

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](#).

[182] [183]

7.2. Dynamic network topology

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].

FANET

[184] [185]

7.3. Localization 7.3.

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].



7.4. UAV speed 7.4.

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].



7.5. UAV types 7.5

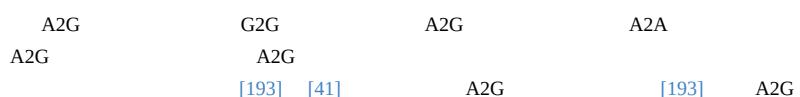
Depending on the goals and application of UAV operation, one must select an appropriate UAV type and meet numerous requirements imposed by the anticipated QoS and environmental constraints. In fact, for the proper utilization of UAVs, numerous factors, such as UAV capabilities and onboard sensor devices, 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.

[191] [192]

7.6. Channel modeling 7.6

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.

UAV-UAV UAV-GBS



7.7. Synchronization 7.7.

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].

[194]

7.8. UAV antennas 7.8

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 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].

[85] [86]

[195] [196]

7.9. Network lifetime 7.9

[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].

[197] [198] [199] [200]
[201] [202]

[203]

7.10. Coordinated communication 7.10

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.

[204]

7.11. Network density 7.11

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.

[\[205\]](#) [\[27\]](#)

7.12. Security and privacy 7.12.

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](#).

[\[206\]](#)

[\[207\]](#)

7.13. Fault-tolerant system 7.13

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.

[\[208\]](#) [\[209\]](#)

7.14. Integration with cloud computing 7.14

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.

[\[210\]](#)

7.15. Path planning 7.15

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].

/

[211]

[212] [213] [214]

3D

7.16. Integration with IoT 7.16

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.

[215]

[216]

7.17. Quality of service 7.17

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.

QoS

QoS

[217]

7.18. Scalability 7.18

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].

[218]

7.19. Mobility management 7.19

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.

[219]

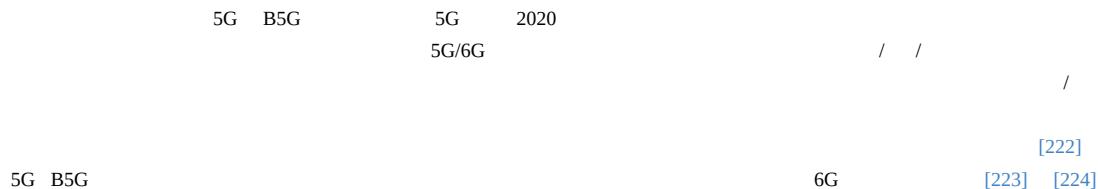
7.20. Task offloading 7.20

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.

[220] [221]

7.21. Impact of 5G, B5G, and mm-wave communication 7.21 5G B5G

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].



7.22. Intelligence-based algorithms 7.22

[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.

[225] [226] [227] [228]

7.23. Lesson learned 7.23

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.

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.

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.

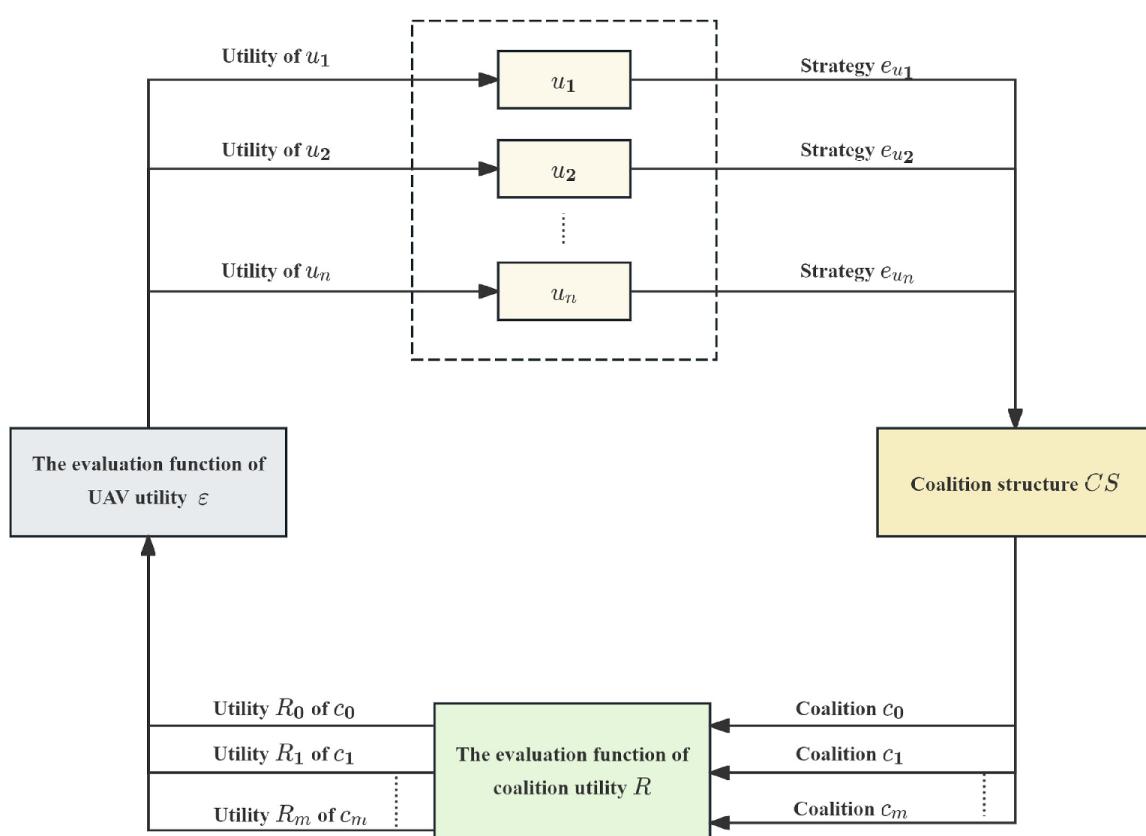
2024

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

- Keywords: task allocation, heterogeneous UAV swarms, heterogeneous resources, clustering algorithm, coalition formation game
 - ; ; ; ;
 - , , K-medoids , ,
 - , , , ,
 - , 30 100 3 , , ,
 - (:62273177,62020106003,62233009) (:BK20211566,BK20222012)
(:B20007) (:HTKJ2023KL502006) (:NI2024001)
 - doi-link
 -
 -
 - 1.
 - i.
 - ii.
 - iii.
 - 2.
 - 1. K-medoids
 - i.
 - ii.
 - 2.
 - 3.
 - 1. K-medoids
 - 2.

[7]

- [8]
 - [9]
 - [10]
 - , [11].
 - [5]
 - [12]
 - [13]



Overall_flowchart_of_the_coalition_formation_game_model

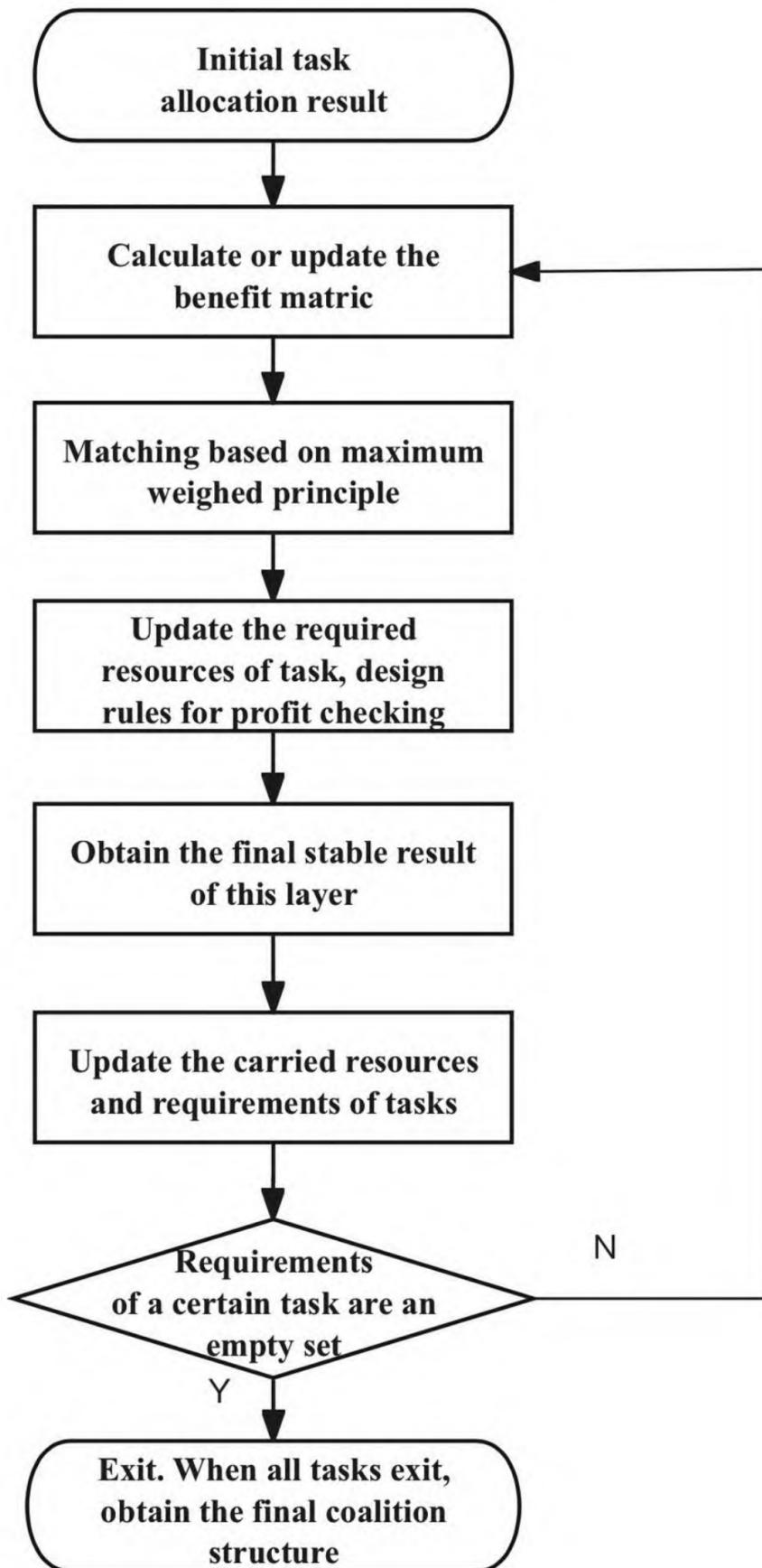


图 2 基于联盟形成博弈的任务分配算法整体流程图

Overall_flowchart_of_the_task_allocation_algorithm_based_on_coalition_formation_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. , , . 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
 , Corso Duca degli Abruzzi 24, 10129 , 2 School of Language and Literature, Harbin Institute of Technology
 (Weihai), Weihai, 264200, China , , 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
 Av. São Carlense, 400, Parque Arnold Schmidt, São Carlos 13566-590, SP,

Abstract

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](#)

MRTA ;

- Table of Contents
- Abstract
- Introduction
- Game-Theory-Based Algorithms
- Learning-Based Algorithms
- Market-Based Algorithms
- Optimization-Based Algorithms
- Hybrid Allocation Algorithms
- Discussion
- Conclusions
- Funding
- Acknowledgments
- Conflicts of Interest
- References

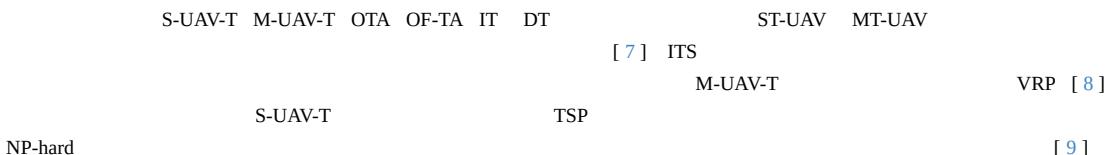
Introduction

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).

- ST-UAV MT-UAV



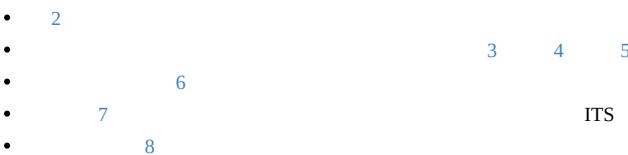
The two most used drones in the six task allocation models (S-UAV-T, M-UAV-T, O-T-A, OF-T-A, I-T, and D-T) mentioned above are S-T-UAV and M-T-UAV, and all six task allocation models involve several common objectives, including maximizing the total revenue of the task set, minimizing the flight distance, 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.



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:

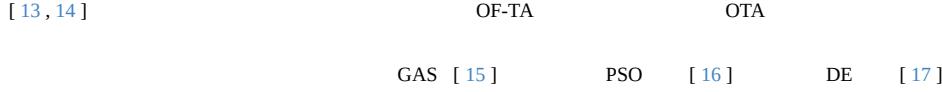
- [7 10 11 12]
- MAS
- The development of a critical review about MAS task allocation methodologies focusing on multi-UAV networks. This review paper is for engineers, researchers, and scholars who need a critical overview of these emerging topics;
 - MAS
 - The discussion of state-of-the-art allocation strategies for UAV-based ITSs, focusing on their suitability to the most established applications;
 -
 - 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.
 -

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.

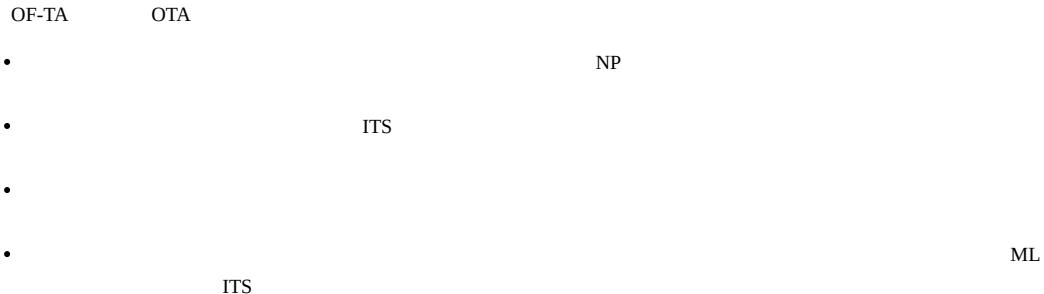


Challenges of Task Allocation Algorithms

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.



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.



Game-Theory-Based Algorithms

[18]

M-UAV-T

[19]

“ ”

$$\begin{aligned}
 & M\text{-UAV-T} & & M\text{-UAV-T} \\
 & \$A=\left\{1,2,3, \ldots, a\{M A X\}\right\} \$ & & \$a\$ \\
 & d\{a, 2\}, d\{a, 3\}, \ldots, d\{a, n\}\right\} \$ & , & \$D\{a\}=\left\{d\{a, 1\}, \right. \\
 & X\}\}\right\} \$ & & \$Z=\left\{z\{1\}, z\{2\}, z\{3\}, \ldots, z\{a_M A\right. \\
 & & & \left.\left.X\}\right\} \$ & & \left.\right\}
 \end{aligned}$$

2.1. Non-Cooperative-Game-Based Task Allocation 2.1.

2.2. Cooperative Game-Based Task Allocation 2.2.

Learning-Based Algorithms

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).

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.

$$b \leq B_{max}$$

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.

[28]

[29]

]

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

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.

[31]

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.

[\[32 \]](#)

TSSA

[\[33 \]](#)[\[34 \]](#)

Most of the proposed auction algorithms yield a poor performance for multi-dynamic tasks for multiple 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.

[\[35 \]](#)[\[36 \]](#)

The use of different auction-based algorithms to solve a heterogeneous 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.

[\[37 \]](#)

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].

[\[38 \]](#)[\[39 \]](#)[\[40 41 42\]](#)

Table 4 provides an overview of the reviewed approaches, including the main characteristics, constraints, and limitations.

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
[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

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.

(MILP)

5.1. Deterministic 5.1.

Hybrid Allocation Algorithms

Hybrid allocation strategies are also designed to improve the transportation efficiency for UAV-based ITSs by merging different types of allocation algorithms.

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]



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] HA ML

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 $\max\{t_i D_i + t_i D/d \cdot r \lambda_i + t_i D \cdot N_i\}$, where the decision variables represent drone detection, object recognition, and neutralization, respectively.



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.



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.



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.

ITS

The main conclusions drawn from this survey are summarized as follows:

- 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;
- 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;
- 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;

- 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;

ITS
/ ITS

- 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.

i ii iii)

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.

[8](#)

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

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.

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.

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.

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.

ITS

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.

ITS

Task Assignment

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

[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:

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.

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:

Jinja2

```
from jinja2 import Template

#
template = Template(open("test.jinja2").read())

#
output = template.render(target="RISCV")

#
#       HTML
print(output)
```

Lalrpop

LR(1) parser generator for Rust.

LALRPOP

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

YACC ANTLR Menhir

README

LALRPOP

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,
};

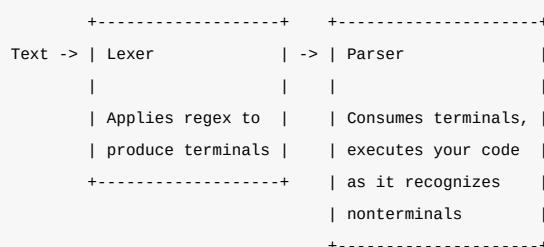
FactorOp: Opcode = {
    "*" => Opcode::Mul,
    "/" => Opcode::Div,
};

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
```



match

```
Simple match declarations      match
```

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"[0-9]+"` 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
```

```
    r"[0-9]+"      r"\w+"
```

```
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).

```
    _           " "   "22"      _           match
```

```
// fixed literals get precedence over regular expressions
match {
    r"[0-9]+",
    "22"
} else {
    r"\w+",
    -
}
```

```
match
```

```
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

```
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

The multiple code generation-related dialects in MLIR can be roughly organized along two axes: *tensor/buffer* and *payload/structure*.

/ /

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.

/ SSA - MLIR

Linalg

Standard Linalg

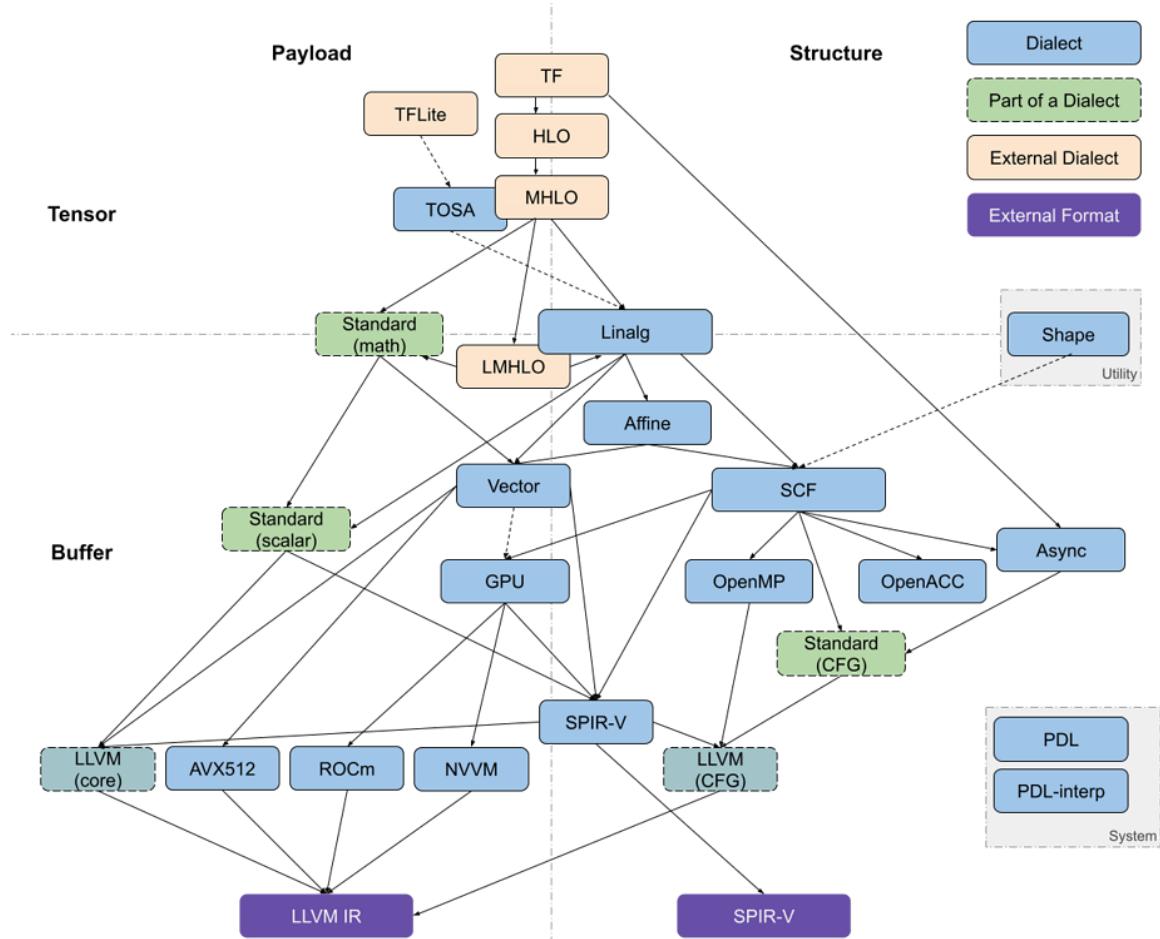
A dialect's position on the payload/structure axis indicates whether it describes *what* computation should be performed (payload) or *how* it should be performed (structure). For example, most mathematical operations in the Standard dialect specify the computation to be performed, e.g., the arctangent, 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.

/ SCF

Async

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.

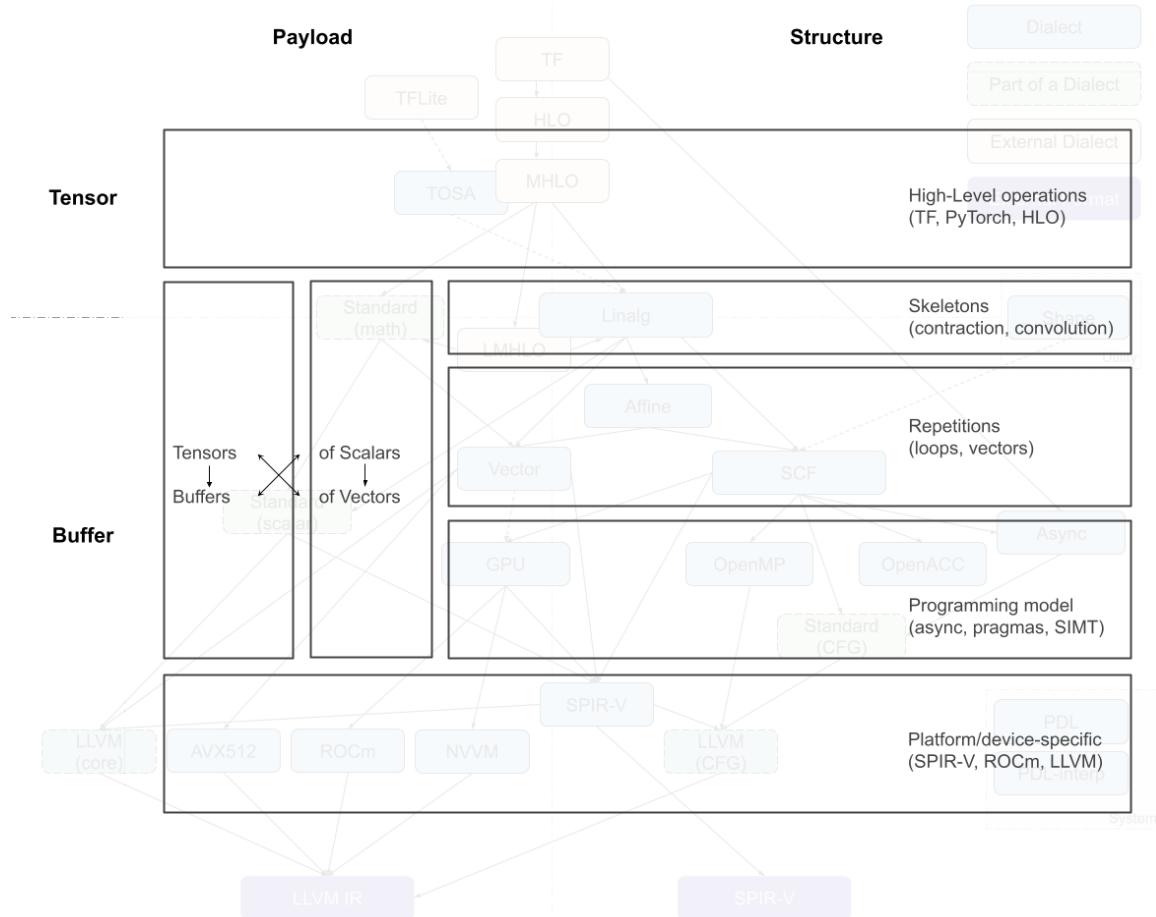
SIMD “
”



Dialects of Interest

An MLIR code generation pipeline goes through a sequence of intermediate steps, which are characterized by the most recently introduced dialects. 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



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.



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.

[
]

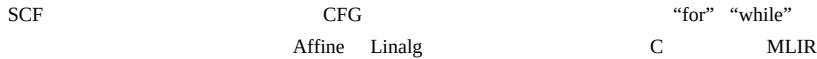
The [Vector](#) dialect (Note that the vector *type* belongs to the built-in dialect and can be used outside of the Vector dialect.) is a mid-level abstraction for SIMD and, potentially, SIMT execution models. It leverages MLIR’s multidimensional vector type to target different platforms through dedicated lower-level dialects. An ongoing work investigates the use of [vector abstraction to target GPU devices \(SIMT\)](#) through explicit representation of threads.

SIMD	SIMT	MLIR
GPU	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.



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.



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.



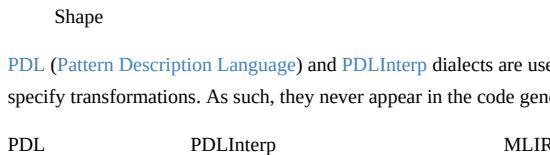
SCF can also be converted to the “standard” CFG representation by replacing structured control flow with branches between blocks. The branch operations are currently contained in the Standard dialect, together with numerous other operations at various abstraction levels. For example, the Standard dialect 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.



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.



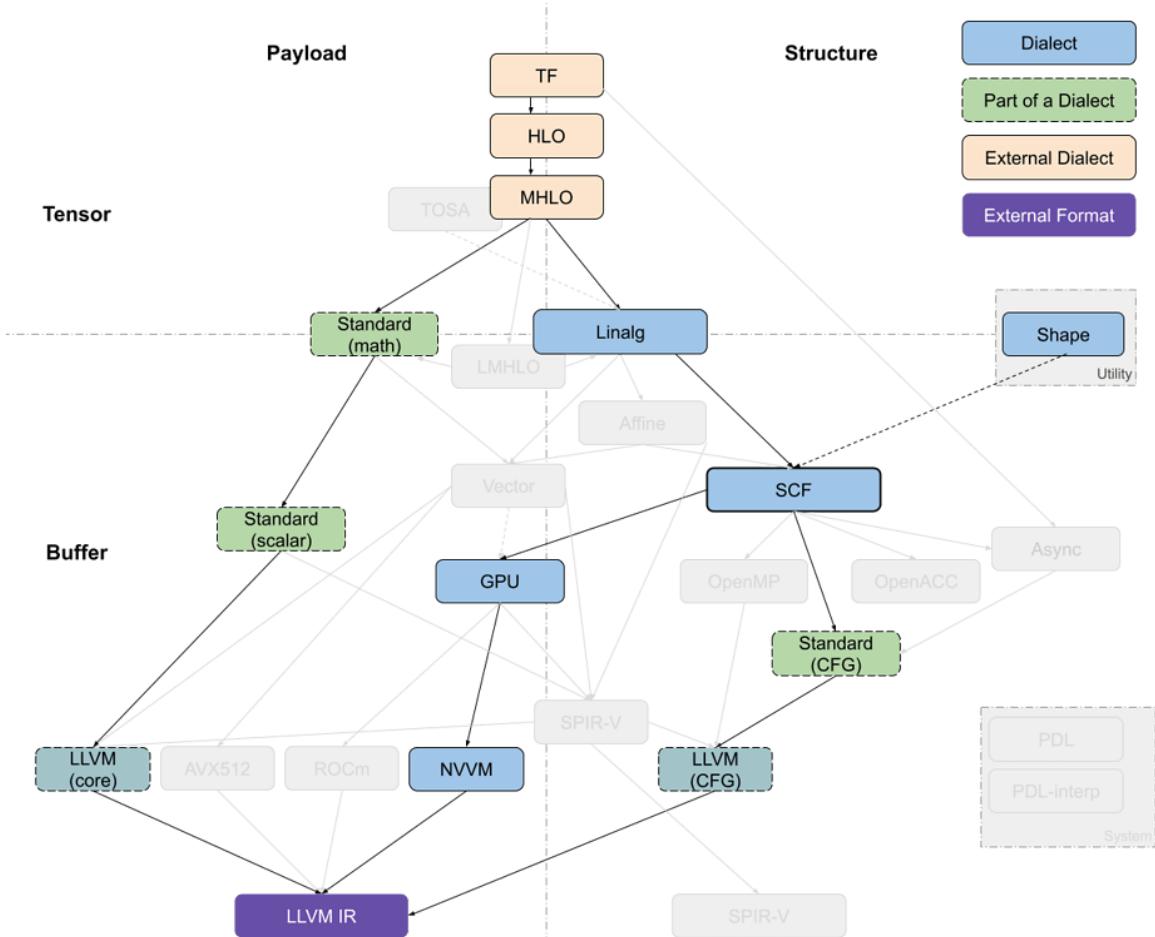
Finally, the [Shape](#) dialect is used to describe shapes of data independently of the payload or (mostly) structure. It appears at the entry level of the code generation pipeline and is typically lowered into address arithmetic or canonicalized away.



Some Existing Pipelines

TensorFlow Kernel Generator TensorFlow





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	TensorFlow	TF	Linalg-on-tensors	MHLO	HLO
	HLO	XLA	LMHLO	MHLO	MHLO
Linalg				tiling	SCF
LLVM				Linalg-on-Buffers	LMHLO
					SCF

When producing several kernels, TensorFlow-related flows are expected to use the Async dialect to orchestrate computation.

TensorFlow

Analysis

Crossing Dialects

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.



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).

IR

Linalg at the Center

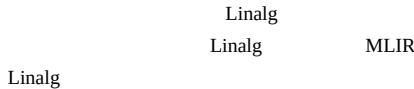
The Linalg dialect is one of the main entry points to the MLIR code generation pipelines. Its most recent evolution makes it operate on both tensors and buffers, making bufferization an intra-dialect transformation. It has sufficiently high-level information about the operations to perform transformations 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

MLIR

Linalg

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.

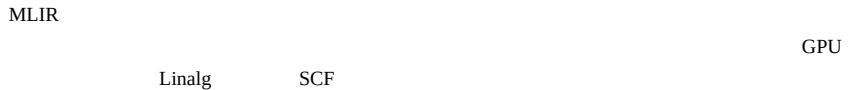


Pipeline Differences to Complementary Transformations

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.



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.



Build Small Reusable Abstractions

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).

HLO

Need for More Structure

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.

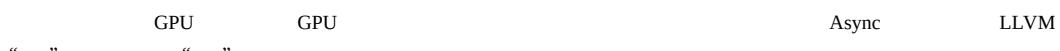


Host/Device Or Program/Kernel As Additional Axis / /

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.



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.



MLIR

MLIR MLIR

MLIR Math Linalg SCF Affine

1. Math

Math

- sin cos exp log sqrt
 -
 -
 -
 -
-

2. Linalg

Linalg

- - linalg.matmul -
 - linalg.conv
 - linalg.generic
 - - Linalg
 -
 - -
 - Affine
 -
 -
-

3. SCF

SCF

- - `scf.for` for
 - `scf.if`
 - `scf.parallel` SIMD
 -
 -
-

4. Affine

Affine

- **Affine**
 -
 - `affine.for %i = 0 to 100 step 2`
 - **Affine**
 -
 -
 -
 -
 -
 -
 - LLVM IR
-

MLIR

- | | | |
|-----------------|------------|---------------|
| • Linalg | SCF | Affine |
| • Math | Linalg | Affine |
| • SCF | Affine | SCF |
-

MLIR

- 1.
- 2.
- 3.
- 4.

MLIR

Linalg

Set of Key Transformations

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.

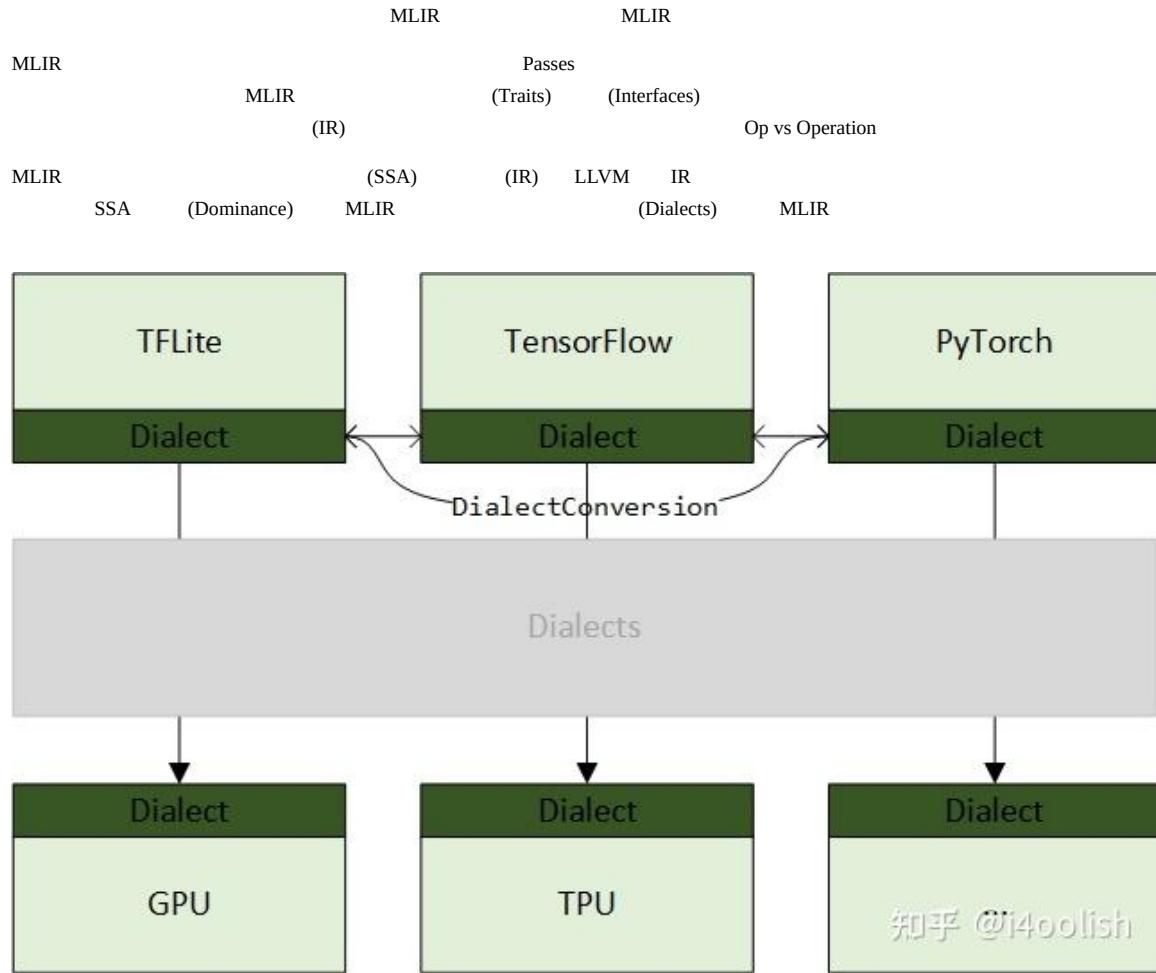
The textual form description of these transformations is left for future work. Still, it is useful to list the key transformations that are performed on the LinAlg IR and that have influenced its design:

1. Progressive Buffer Allocation.
 2. Parametric Tiling.
 3. Promotion to Temporary Buffer in Fast Memory.
 4. Tiled Producer-Consumer Fusion with Parametric Tile-And-Fuse.
 5. Map to Parallel and Reduction Loops and Hardware.
 6. Vectorization: Rewrite in Vector Form.
 7. Lower to Loops (Affine, Generic, and Parallel).
 8. Lower to Library Calls or Special Instructions, Intrinsics or ISA.
 9. Partially Lower to Iterations Over a Finer-Grained LinAlg Op.

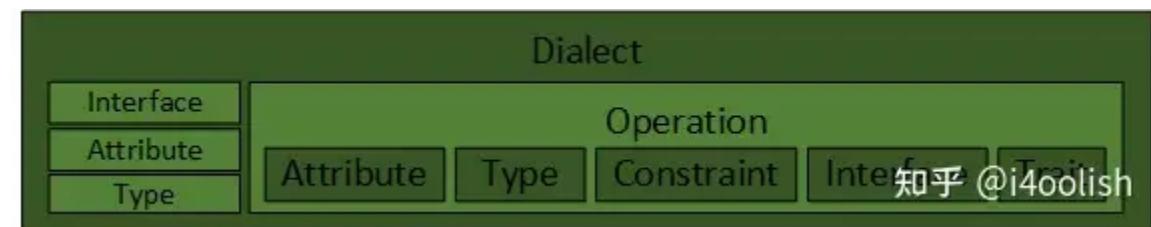
MLIR Language Reference

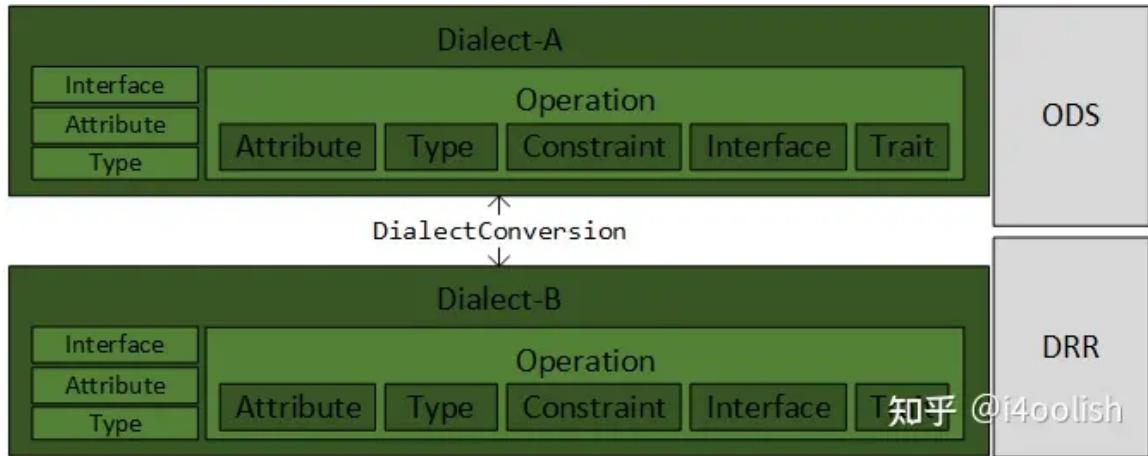
text text text

MLIR Argument)	(Operations) (Blocks)	(Values) (Regions)	(Block
-------------------	--------------------------	-----------------------	--------



text





MLIR

1. Dialect, Attribute, Type, Operation
2. DialectConversion
3. Interface, Constraint, Trait
4. Transformation, Concatenation
5. Region, Block
6. Pass
7. ODS DRR

DialectConversion

[MLIR Dialects](#)

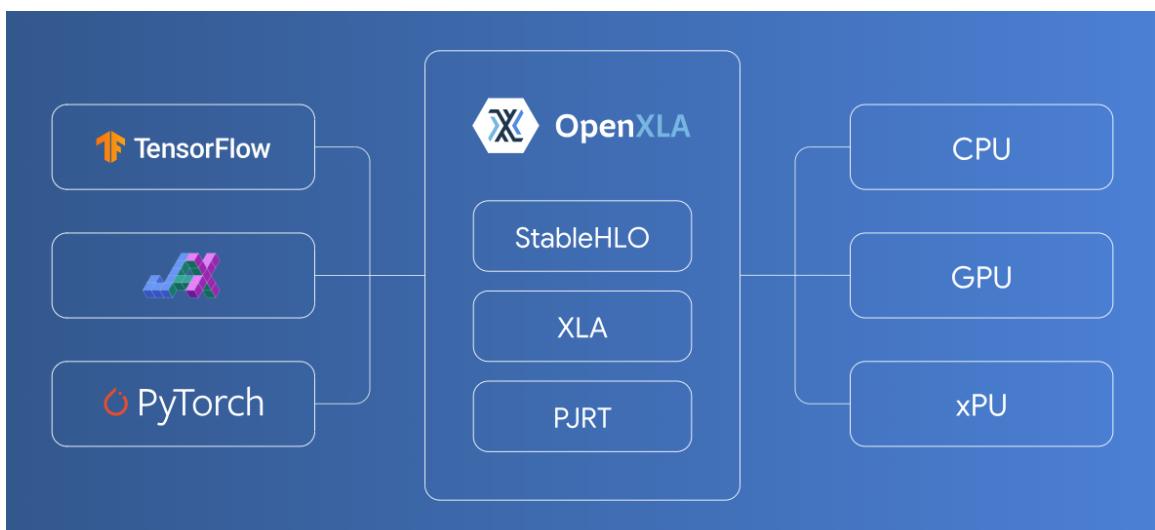
OpenXLA

[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.

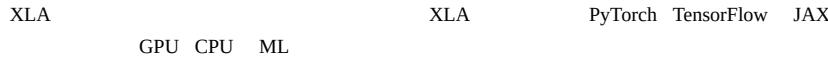
(ML)

ML AI



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.



Key benefits

- **Build anywhere** : XLA is already integrated into leading ML frameworks such as TensorFlow, PyTorch, and JAX.
 - XLA ML TensorFlow PyTorch JAX
- **Run anywhere** : It supports various backends including GPUs, CPUs, and ML accelerators, and includes a pluggable infrastructure to add support for more.
 - GPU CPU ML
- **Maximize and scale performance** : It optimizes a model's performance with production-tested optimization passes and automated partitioning for model parallelism.
 -
- **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.
 - MLIR
- **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.
 - ML XLA ML

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.



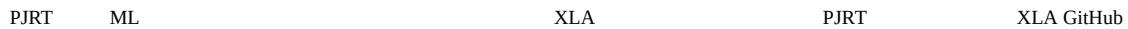
Shardy

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.



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.



TF(TensorFlow) Dialect

This dialect maps to TensorFlow operations.

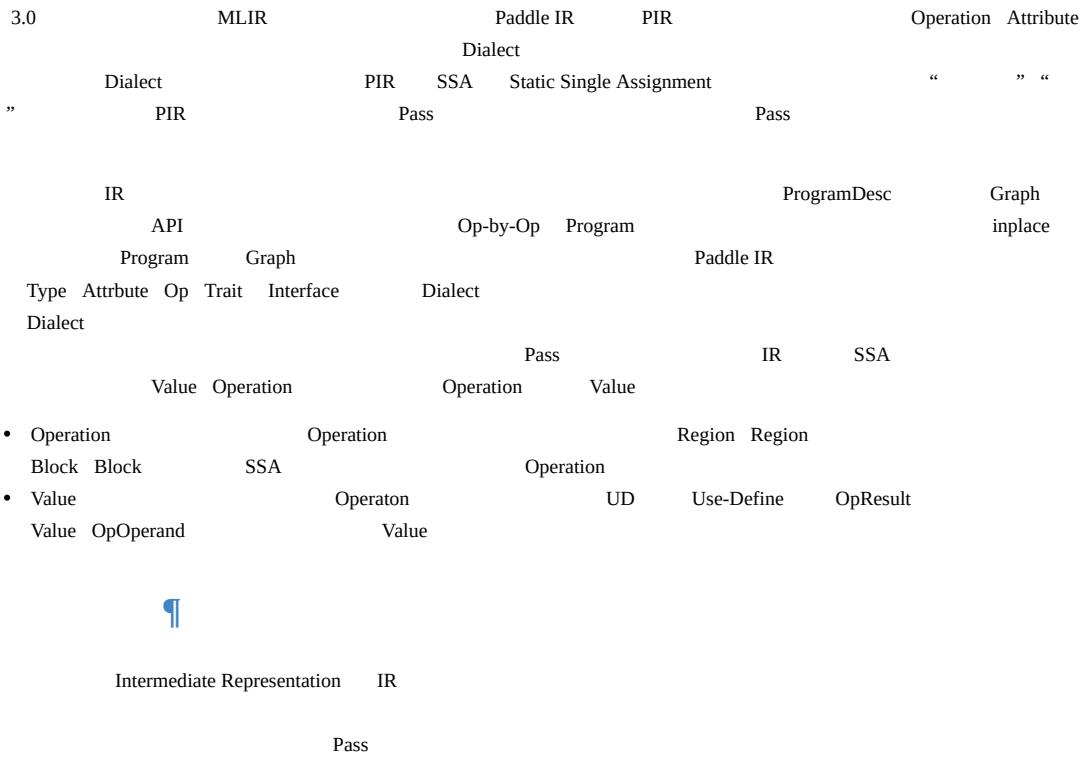
TensorFlow

Invariants:

- All values are of Tensor type (in particular, scalars are represented using zero-dimensional tensors);
- Tensor

TODO: Make invariants more structured so that we can reference them in ops.TODO

PIR



Kimi

C passthrough.h.txt passthrough.c.txt cudawrap CUDA
 CUDA Runtime cuDNN cuBLAS

passthrough.h

cudawrap CUDA
 1.

- entry_t fn_ptr name

 2.

- ENTER LEAVE
- TIME_GEMM_START TIME_GEMM_STOP GEMM General Matrix Multiply
- CUDART_ENTRY_ENUM CUDNN_ENTRY_ENUM CUBLAS_ENTRY_ENUM
- CUDART_FIND_ENTRY CUDNN_FIND_ENTRY CUBLAS_FIND_ENTRY
- REAL_CUDA_RT_API_CALL REAL_CUDNN_CALL REAL_CUBLAS_CALL

 3.

- cudart_entry_enum_t cudnn_entry_enum_t cublas_entry_enum_t CUDA Runtime
- cuDNN cuBLAS

passthrough.c

passthrough.h
 1.

- __cudart_inited__ __cudnn_inited__ __cublas_inited__

 2.

- cudart_lib_entry cudnn_lib_entry cublas_lib_entry

 3.

- load_cudart_lib load_cudnn_lib load_cublas_lib CUDA Runtime cuDNN cuBLAS

 4.

- cudart_lib_inited cudnn_lib_inited cublas_lib_inited

cudawrap CUDA
 CUDA CUDA

cudawrap
 1. C C++ passthrough.h
 2. cudawrap CUDA libcudart.so, libcudnn.so, libcublas.so
 3. load_cudart_lib(), load_cudnn_lib(), load_cublas_lib() CUDA
 cudawrap
 4. cudawrap CUDA
 cudawrap CUDA Runtime API cudaMalloc cudaFree

```

#include "passthrough.h"

int main() {
    //      CUDA
    if (load_cudart_lib() != 0) {
        fprintf(stderr, "Failed to load CUDA runtime library.\n");
        return -1;
    }

    //      CUDA
    void *devPtr;
    size_t size = 1024 * 1024; //      1MB
    REAL_CUDA_RT_API_CALL(cudart_lib_entry, , cudaMalloc, (void**)&devPtr, size);

    if (devPtr == NULL) {
        fprintf(stderr, "Failed to allocate device memory.\n");
        //
        return -1;
    }

    // ...
    CUDA ...
}

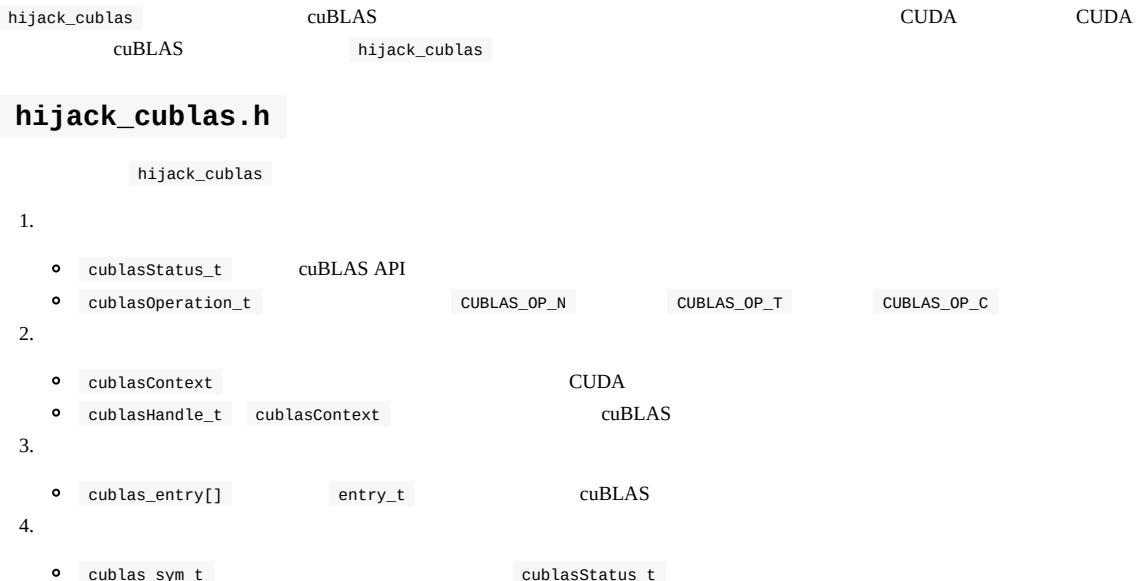
//      CUDA
REAL_CUDA_RT_API_CALL(cudart_lib_entry, , cudaFree, (devPtr));

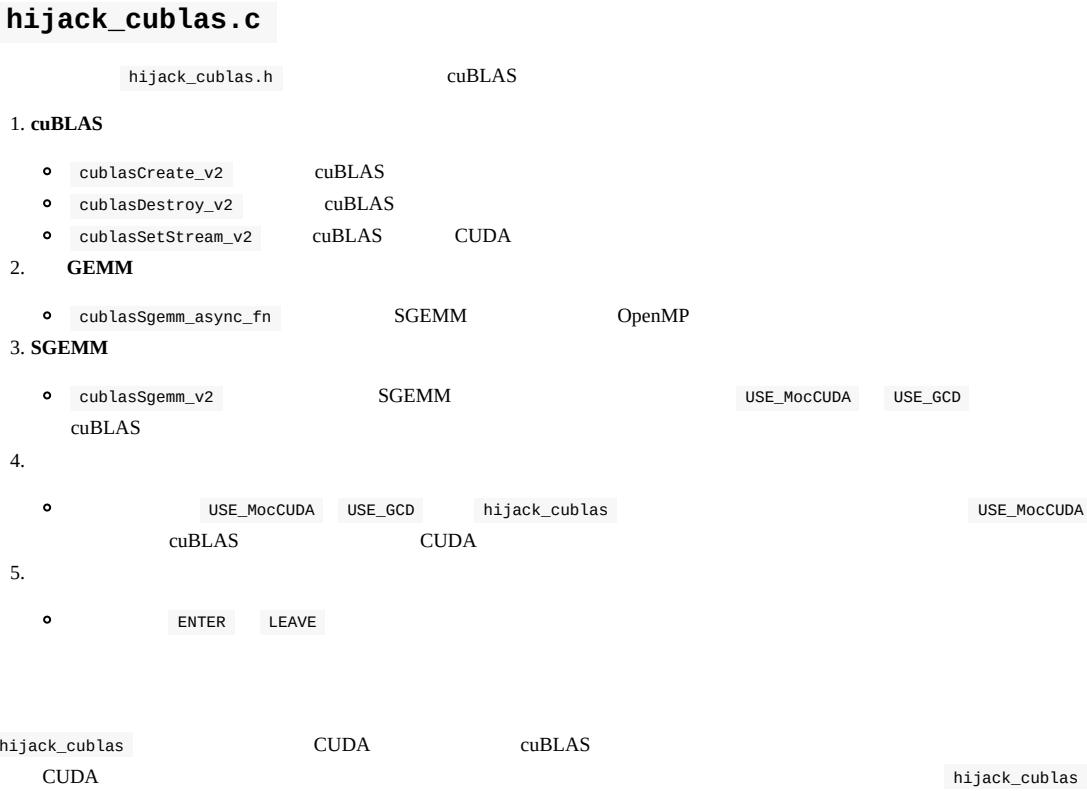
//
return 0;
}

load_cudart_lib()      CUDA      cudawrap      REAL_CUDA_RT_API_CALL
cudaMalloc      cudaFree
cudawrap

```

Hijack CUBLAS





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 (`cudaLibEntry`, `cudnnLibEntry`, and `cublasLibEntry`) store the function names and their resolved function pointers (`fn_ptr`).
- Functions like `loadCudartLib()`, `loadCudnnLib()`, and `loadCublasLib()` populate these arrays.

3. Initialization Flags:

- Static variables like `_cudaInit`, `_cudnnInit`, and `_cublasInit` 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", cudaGetErrorString(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", cudaGetErrorString(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_v2);

    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 (*cUBLASGemm_t)(cUBLASHandle_t, cUBLASOperation_t, cUBLASOperation_t,
                                           int, int, int, const float *, const float *, int,
                                           const float *, const float *, int, float *);
    cUBLASGemm_t cublasGemm = (cUBLASGemm_t)CUBLAS_FIND_ENTRY(cUBLAS_lib_entry, cublasGemm_v2);

    if (cublasGemm == NULL) {
        fprintf(stderr, "Function cublasGemm_v2 not found!\n");
        return -1;
    }

    cUBLASStatus_t status = cublasGemm(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, "cUBLASGemm 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, cublasDestroy_v2);

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 `${MocCC}` and `${MocCXX}` respectively, along with their corresponding flags (`${MocCFLAGS}` and `${MocCXXFLAGS}`), 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)

libdispatch (libdispatch.dylib) macOS iOS Apple

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 FIFO

GCD

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).

GCD GCD

GCD

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.

iOS

UI

...

[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:

- 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:

- Defines the compilers:
 - `cc` : Default C compiler (`clang`).
 - `cxx` : Default C++ compiler (`clang++`).
 - `fc` : Default Fortran compiler (`flang`).

3. Compile-Time Flags:

- Optimizations:
 - `OPTI` : Includes architecture-specific optimizations for the A64FX CPU (Fugaku) and link-time optimization.
- Debug flags:
 - `DEBUG` : Disabled by default, can enable `-DDEBUG` and debugging flags (`-O0`, `-g`).
- CUDA handling:
 - `RedirectCUDA` : Redirects CUDA functionality to a mock or alternative implementation.
 - `RedirCUDAPaths` : Adds custom paths for CUDA and cuDNN.
- 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 -fno-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$(LIBPWQ_ROOT)/lib -L$(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
CXXSRCS ?= src/torch/hijack_torch_cuda.cpp
FSRCS ?= src/utils/frt_utils.f90
```

- **CSRCS** : C source files for CUDA wrapping and utilities.
- **CXXSRCS** : 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:**
 - Relies 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` ?

Ch12

[OpenBLAS](#)

[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:

Bash	CUDA	GPU	Score-P
<code>libcudart.so</code>	<code>libcublas.so</code>	CUDA	

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 `cufftXt.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.

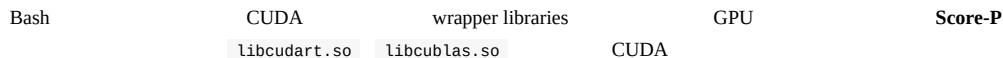
Potential Use Cases

- 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.



1.

- `scorep-libwrap-init` CUDA Score-P
- CUDA

2.

- - CUDA
 - fPIC
 - - libcudart.so CUDA API
 - libcublas.so BLAS
 - libcufft.so FFT
 - libcurand.so
 - libcudnn.so
 - libcusparse.so
 - libnvToolsExt.so
 - libncl.so GPU
-

1.

- - - wrapper
- - CUDA Toolkit cuDNN NCCL


```
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.

- CUDA
- 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_${LIBN}
```
 - - libwrap.h .filter Makefile
 - CUDA cublas cublas_v2.h cudnn cudnn.h
 - cuda* curandGenerateBinomial*
 - ```
make libscorep_libwrap_${LIBN}_runtime.la
```

**3.**

- - libcudart.so                      set\_launch\_func
    - libcufft.so        cufft.h    cufftXt.h
- 

wrapper                      .la

- `libscorep_libwrap_cudart_runtime.lib`
- `libscorep_libwrap_cUBLAS_runtime.lib`
- `libscorep_libwrap_cUFFT_runtime.lib` ...

Score-P GPU

---

- - CUDA
- - CUDA
- - HPC GPU

- - sed
- - `/usr/local/cuda-10.0/`
- - `mkdir scorep-libwrap-init make`

CUDA Score-P GPU

---

# 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;  COMPIILER 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

- [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.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | PEG    |     | Bryan Ford |       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|-----|------------|-------|
| 2004                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | 20     | 70  | PEG        | CFG   |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | PEG    | CFG |            |       |
| 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] |        |     |            |       |
| CFG                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | PEG    |     | PEG        | 1 PEG |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Lojban |     |            |       |

## 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      Rust      PEG

```
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

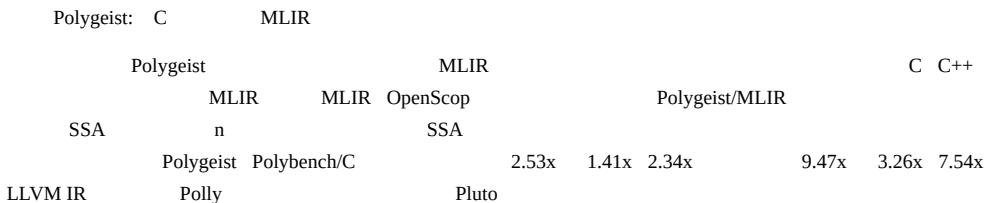
## Title:

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.

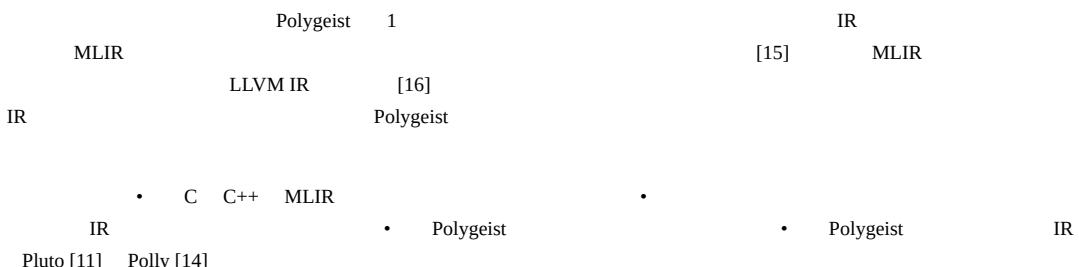
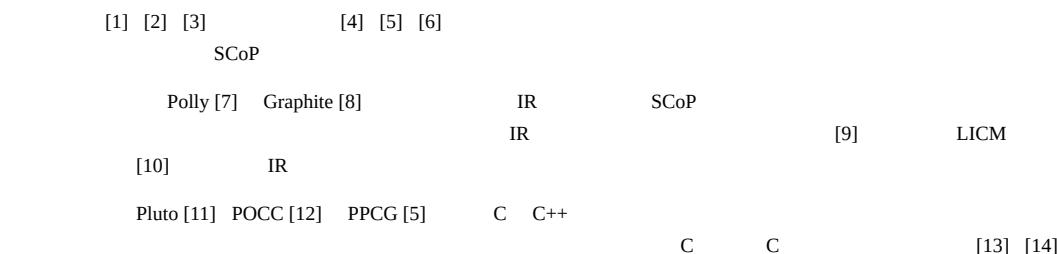
## Meta Translation



## I. INTRODUCTION

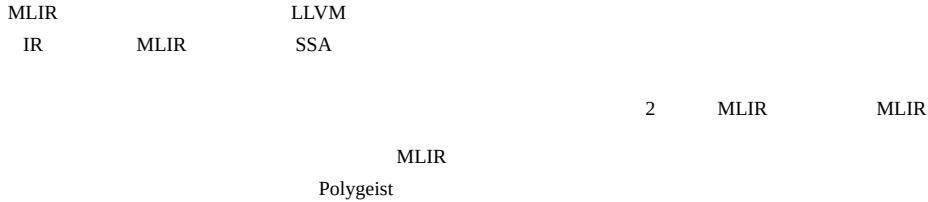
I.

CPU GPU



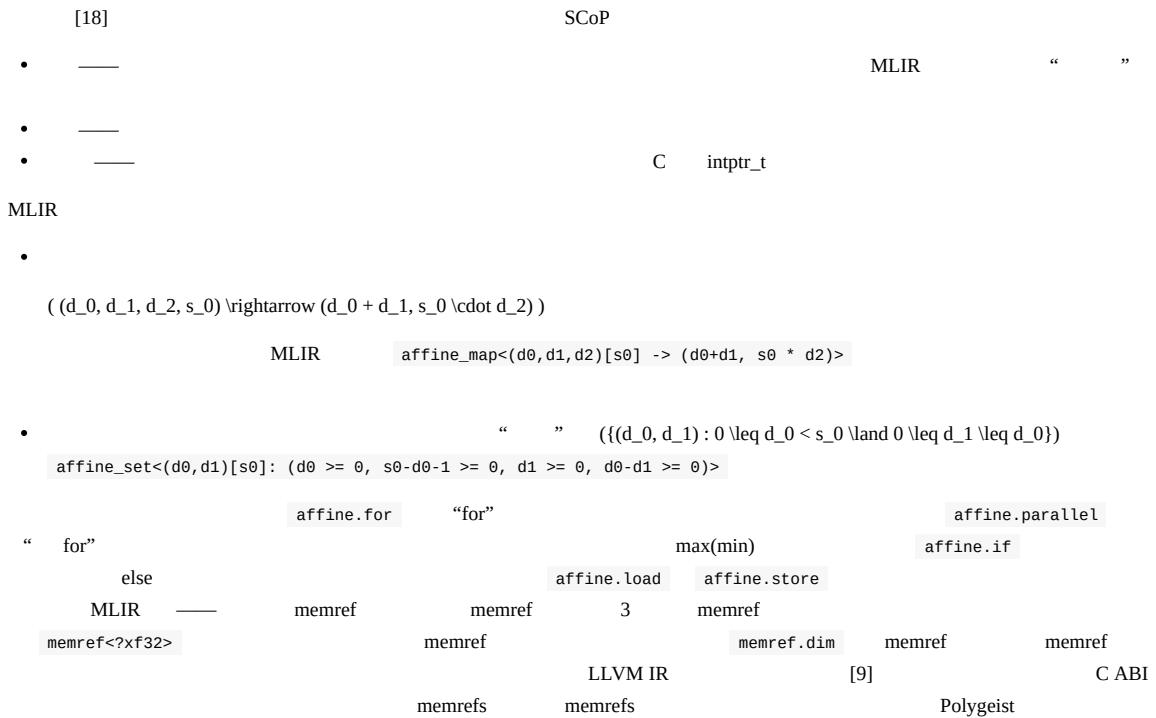
## A. Overview

A.



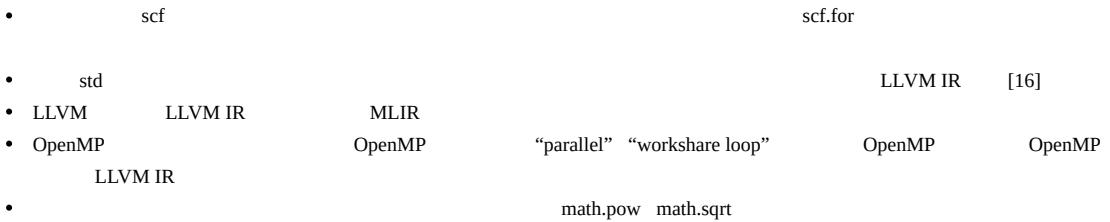
## B. Affine and MemRef Dialects

B.

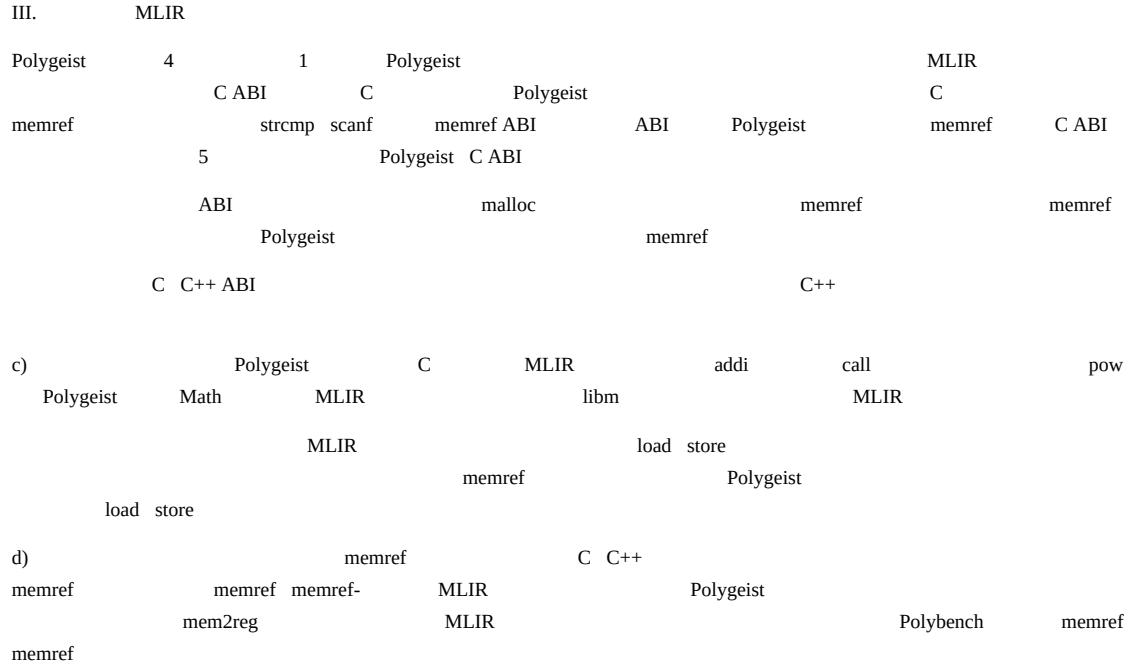


## C. Other Relevant Core Dialects

MLIR



## III. AN (AFFINE) MLIR COMPILATION PIPELINE



## B. Raising to Affine

|       |      |              |           |
|-------|------|--------------|-----------|
| C C++ | MLIR | n            | Polygeist |
|       |      | #pragma scop | Polygeist |

### a) Memory operations and loop bounds:

|                      |                      |                                   |                                          |
|----------------------|----------------------|-----------------------------------|------------------------------------------|
| a)                   | Polygeist            | affine_map<() [s0]->(s0)>[%bound] | affine_map<()                            |
|                      | addi                 | muli                              |                                          |
| [s0]->(s0)> [%bound] | %bound = addi %N, %i | %i                                | affine_map<(d0)[s0] ->(s0 + d0)>(%i)[%N] |
| b)                   | if                   |                                   |                                          |
|                      |                      | affine.if                         | $\geq 0 = 0$                             |
|                      |                      |                                   | Polygeist                                |
|                      |                      |                                   | C                                        |
|                      |                      |                                   | #pragma scop                             |
|                      |                      |                                   | mem2reg                                  |
|                      | Polybench/C          | [19]                              |                                          |

## C. Connecting MLIR to Polyhedral Tools

|             |            |          |           |               |
|-------------|------------|----------|-----------|---------------|
| MLIR Affine | Pluto [11] | isl [21] | MLIR      | OpenScop [20] |
|             |            |          | Polygeist |               |
| MLIR        |            |          |           |               |
|             | C FORTRAN  | MLIR     |           |               |
| MLIR        |            |          |           |               |

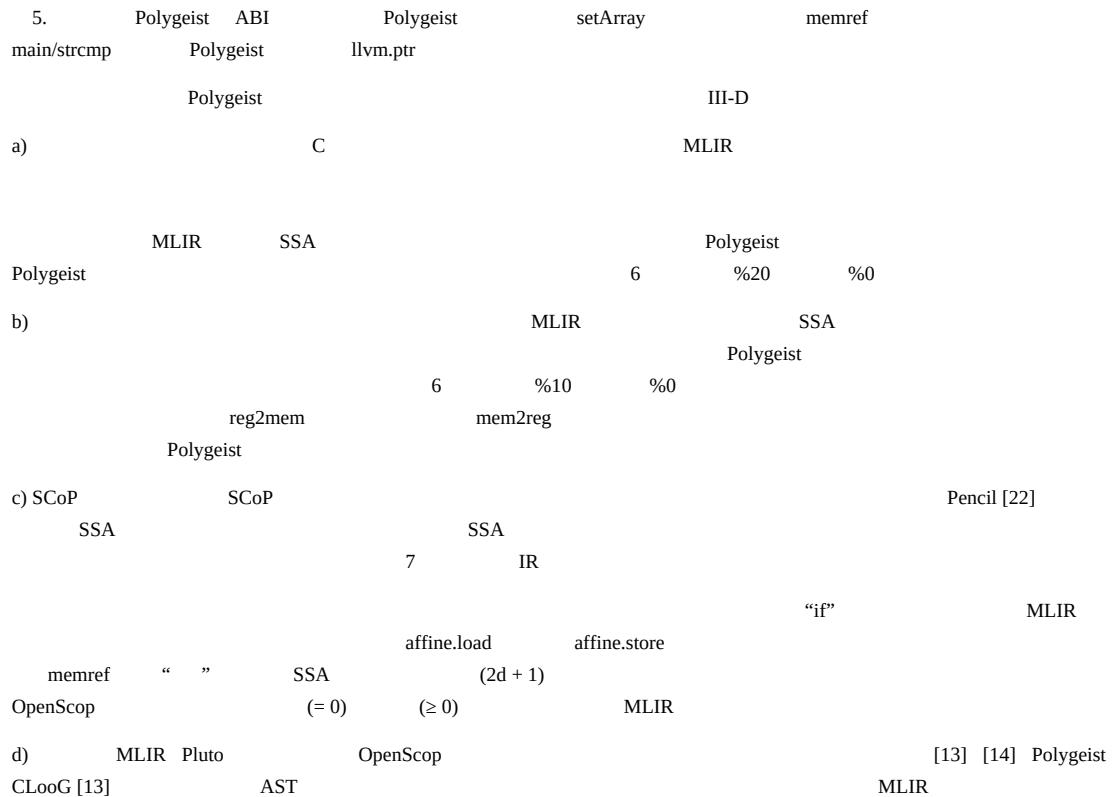
```

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) }
```

```

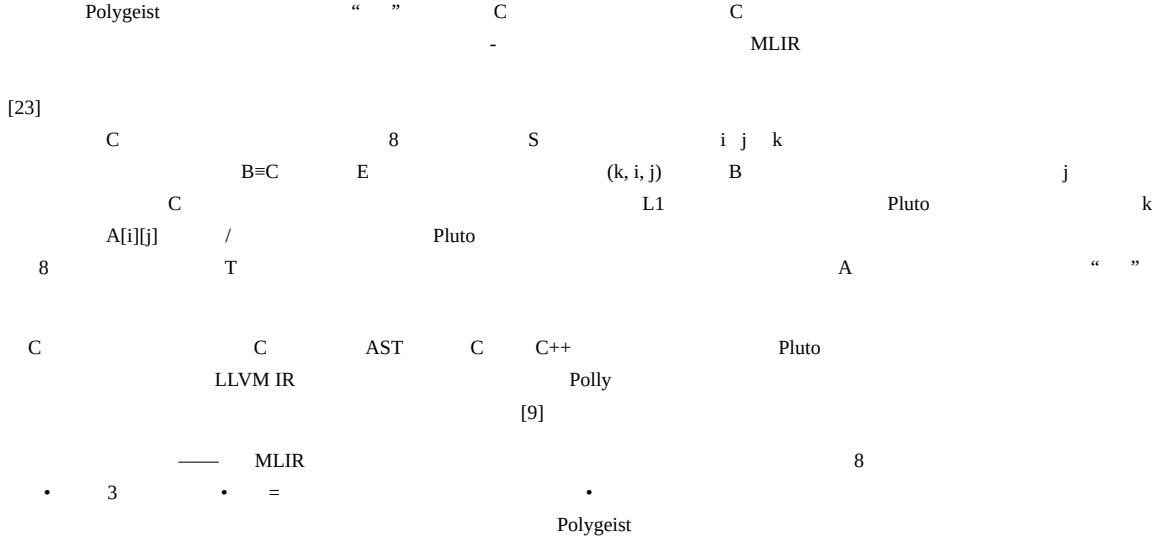
func @setArray(%N: i32, %val: f64, %array: memref<?xf64>) {
 %0 = index_cast %N : i32 to index
 affine.for %i = 0 to %0 {
 affine.store %val, %array[%i] : memref<?xf64>
 }
 return
}

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>) -> ()
}
```



## D. Controlling Statement Granularity

D.



## E. Post-Transformations and Backend

E.

Polygeist      quasi-syntactic      SSA  
                    Polygeist

a)      Polygeist      MLIR      [24]      Polly [25]      LLVM

[26]      MLIR      /      9      Polygeist  
                    /  
                    mem2reg  
b)      Polygeist      [27] [28]      MLIR      SSA      [29]      mem2reg      SSA

## IV. EVALUATION

1      Polygeist      Clang      2  
Polygeist      SSA

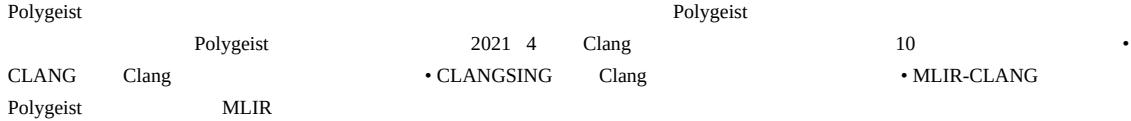
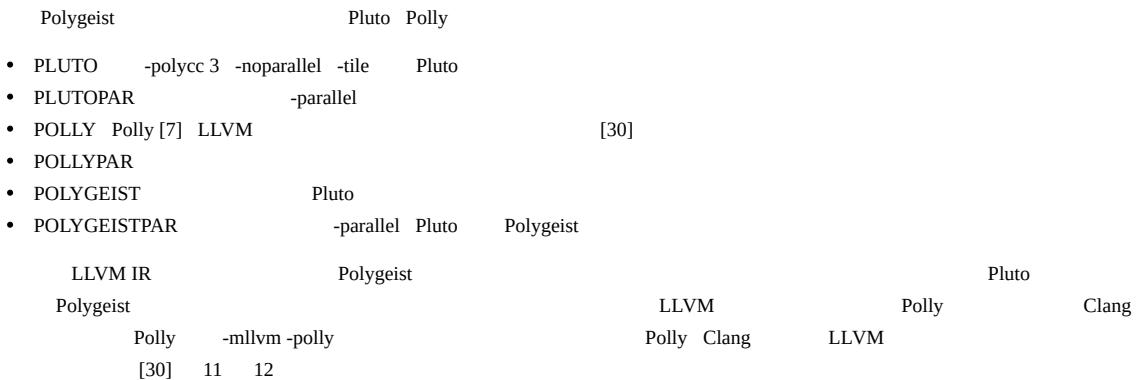
## A. Experimental Setup

|              |                 |    |             |                                    |                            |
|--------------|-----------------|----|-------------|------------------------------------|----------------------------|
|              | AWS c5.metal    |    | Turbo Boost | Ubuntu 20.04                       | Intel Xeon Platinum 8275CL |
| CPU          | 3.0 GHz         | 24 | L1 L2 L3    | 0.75 MB 35 MB 35.75 MB             | 256 GB RAM                 |
| "EXTRALARGE" | Poly-Bench [19] |    | 30          | Pluto adi                          | SCoP                       |
| PolyBench    |                 |    | 0           |                                    | 5                          |
|              |                 |    |             |                                    | 1-8                        |
|              | (i) clang -O3   |    |             | Polygeist C LLVM IR (ii) clang -O3 |                            |
|              | Polygeist MLIR  |    | mem2reg     |                                    | CLANG                      |

## CLANGSING

**B. Baseline Performance**

B.

**C. Compilation Flows****A. Benchmarking**

|                                          |            |                                         | IEEE 754 | -ffast-math |
|------------------------------------------|------------|-----------------------------------------|----------|-------------|
| Polybench                                | Clang/LLVM |                                         |          |             |
| 20d5c42e0ef5d252b434bcb610b04f1cb79fe771 | 3 Pluto    | da26e77b94b2624a540c08ec7128f20cd7b7985 |          | adi         |
|                                          | malloc     |                                         |          |             |

**B. Baseline Comparison**

| CLANG | CLANGSING                   |            | 0.43%       |
|-------|-----------------------------|------------|-------------|
| CLANG | CLANG                       | MLIR-CLANG | 0.24%       |
| 0.05  | jacobi-1d                   |            |             |
|       | jacobi-1d gesummv atax bicg |            | 0.32% 0.17% |

**C. Performance Differences in Sequential Code**

| Polygeist | Pluto     | 2.34× | Polly | 1.41× |           | 2.53×     | Polygeist | Polly |
|-----------|-----------|-------|-------|-------|-----------|-----------|-----------|-------|
|           |           |       | lu    | mvt   |           | Polygeist | 2mm       | Pluto |
| Polly     | V-E       |       |       |       |           |           | 3GHz      | Pluto |
| 2.7•10^11 | /         |       | add   | 1/2   | seidel-2d | 1         | ≈ 59      |       |
|           | Polygeist |       |       |       | imul/shl  |           |           |       |
|           | Polygeist |       |       |       | memref    |           |           |       |
| Clang     | 5         |       |       |       | CLooG     | jacobi-2d | 40        | 75    |
|           |           |       |       |       |           |           | Polygeist |       |

## D. Performance Differences In Parallel Code

|           |           |             |            | cholesky  | lu        | Pluto  | Polygeist | Polly       |           |
|-----------|-----------|-------------|------------|-----------|-----------|--------|-----------|-------------|-----------|
|           | gemver    | mvt         | Polly      |           |           |        |           |             |           |
| 10        | CLANG     | CLANGSING   | MLIR-CLANG |           | Polybench |        |           | 95%         |           |
|           | Polygeist |             | Clang      |           |           |        |           |             | jacobi-1d |
| 11        |           | CLANG       |            |           | Polygeist |        |           | 2.53×       | Pluto     |
| 2.34×     | Polly     | 1.41×       | Pluto      | adi       |           |        |           |             |           |
| 12        |           | CLANG       |            |           | Polygeist |        |           | 9.47×       | Pluto     |
| 7.54×     | Polly     | 3.26×       | Pluto      | adi       |           |        |           |             |           |
| ludcmp    | syr(2)k   | SSA         |            | Polygeist | Pluto     | Polly  | Polygeist |             |           |
| SSA       |           |             |            |           |           |        |           |             |           |
| Polygeist | deriche   |             | 6.9×       | symm      | 7.7×      |        |           | Polygeist   |           |
|           |           | 13          |            | i         | ym1       |        |           | Polygeist   | ym1       |
| j         | SSA       |             |            |           |           |        |           |             |           |
| Pluto     | 54×       | gramschmidt | Polygeist  |           | 56×       | durbin | 6×        | gramschmidt | durbin    |
|           |           |             | V-F        |           |           | durbin |           |             |           |
| Polybench |           |             | CPU        |           |           |        |           | Polly       | 34×       |
| GPU       |           |             | [31], [24] |           |           |        |           |             |           |

## E. Case Study: Statement Splitting

E.

|           |    |     |     |                              |           |           |     |          |                |
|-----------|----|-----|-----|------------------------------|-----------|-----------|-----|----------|----------------|
| 5         |    | 2mm | 3mm |                              | trmm      |           |     |          |                |
| -nosplit  | 14 | 2mm |     | 4.1%                         | 3.13      | 3.26      |     | 25%      | 50% 51%        |
| 27%       |    | 36% | 20% | 44%                          | 40%       | -9%       |     |          |                |
|           |    |     |     | A[i][j] += B[k][i] * B[k][j] | Polygeist | (k, i, j) |     |          |                |
|           |    |     |     | 75%                          | 50%       | 20%       | 27% | -26%     |                |
| 2mm       | 9% |     | 18% | 32%                          | 32%       | 21%       |     | r = 0.99 | p = 3 • 10 -11 |
|           |    |     |     | 2mm                          |           |           |     |          |                |
| Polygeist |    |     |     |                              |           |           |     |          |                |

## F. Case Study: Reduction Parallelization in durbin

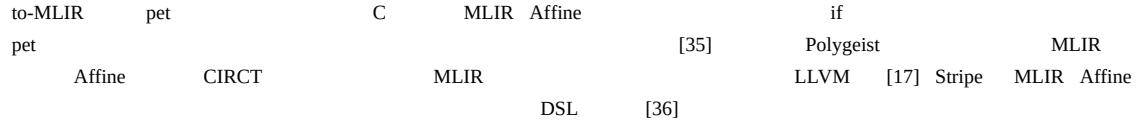
F. Durbin

|           |           |          |       |             |
|-----------|-----------|----------|-------|-------------|
| Polygeist |           | N = 4000 |       | N           |
|           |           |          |       | 15          |
| Polygeist | N ≥ 16000 | > 1      | Polly | N ≥ 224000  |
| Polygeist | Polly     | Pluto    |       | 6.62× 1.01× |

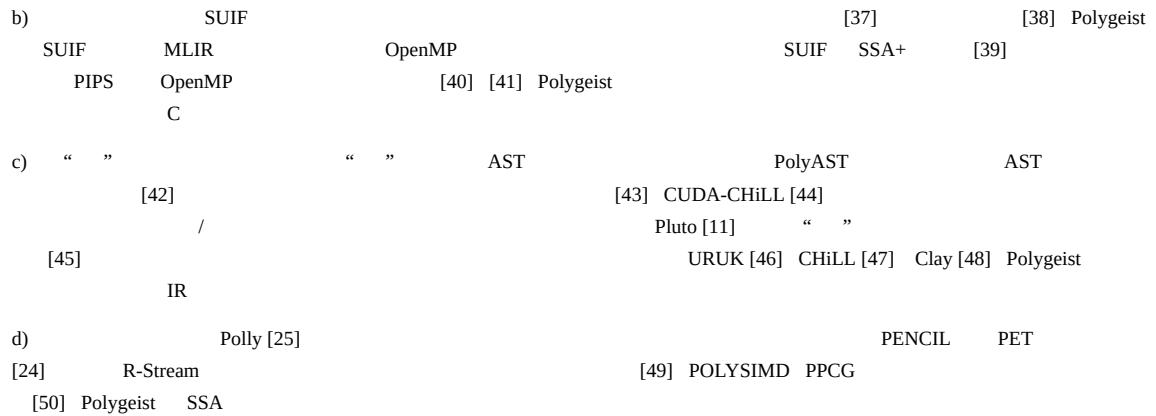
## VI. RELATED WORK

VI.

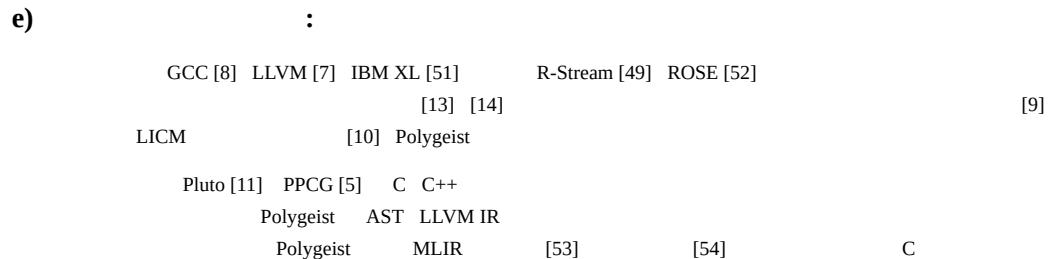
|         |        |       |      |         |            |                |
|---------|--------|-------|------|---------|------------|----------------|
| a) MLIR | MLIR   | LLVM  |      | MLIR    | Teckyl [2] | [1]            |
| MLIR    | Linalg | Flang | LLVM | Fortran | FIR        | Fortran        |
|         |        |       |      |         |            | [32] COMET     |
|         |        |       |      |         |            | Python PyTorch |
|         |        |       |      |         |            | MLIR [34] PET- |
|         |        |       |      |         |            |                |
|         |        |       | [33] | NPComp  |            |                |



## b) Compilers Leveraging Multiple Representations:



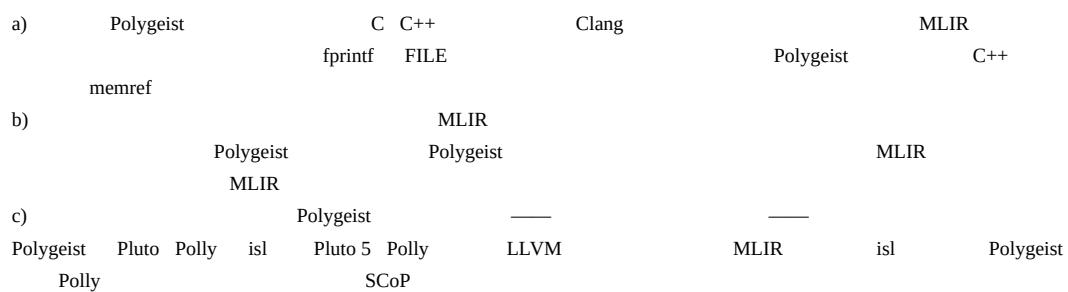
## e) Integration of Polyhedral Optimizers into Compilers:



## VII. DISCUSSION

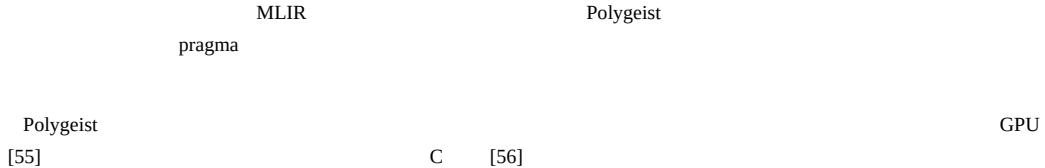
VII.

A.

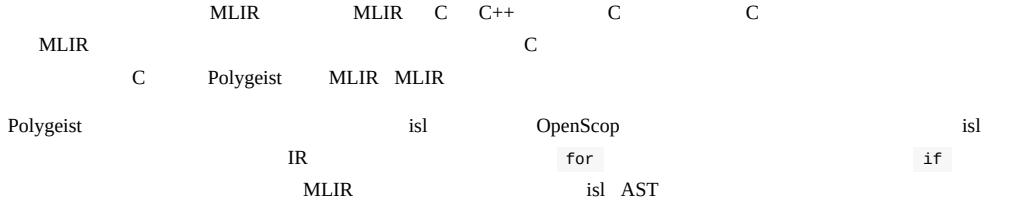


## B. Opportunities and Future Work

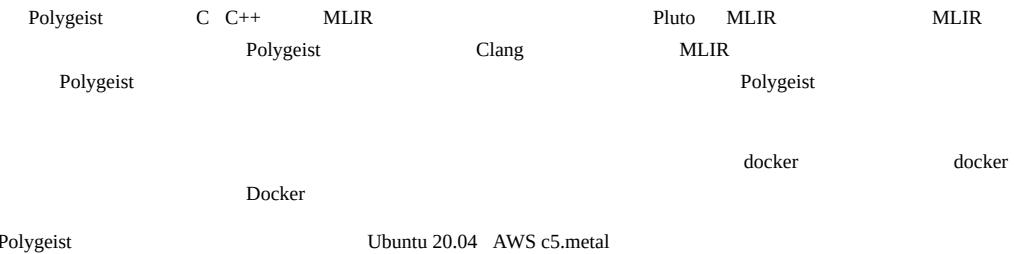




## C. Alternatives



## VIII. CONCLUSION



```

$ 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

Pluto
$ 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

```

```
$ cd
$ git clone \
https://github.com/wsmoses/Polygeist-Script\ scripts
```

## 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
```

## LLVM MLIR

## omp.h

```
$ cd $ export OMP_FILE= find \ $HOME/mlir-clang/build -iname omp.h
```

```
$ cp $HOME/scripts/omp.h $OMP_FILE
```

## MLIR

## LLVM

```
$ 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
```

## MLIR

```
$ 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
```

## AWS

```
$ cd ~/scripts/ $ sudo bash ./hyper.sh
```

```
taskset -c 1-8 numactl -i all
```

```
cd ~/scripts/ $ cd polybench-c-4.2.1-beta/ $./run.sh #
```

**APPENDIX**

Polygeist

Polybench

Docker

# 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

| Polygeist: C |       | MLIR      |             | C C++          |                   |
|--------------|-------|-----------|-------------|----------------|-------------------|
| Polygeist    |       | MLIR      |             | Polygeist/MLIR |                   |
| SSA          | n     | MLIR      | OpenScop    | SSA            |                   |
|              |       | Polygeist | Polybench/C | 2.53x          | 1.41x 2.34x       |
| LLVM IR      | Polly |           |             | Pluto          | 9.47x 3.26x 7.54x |

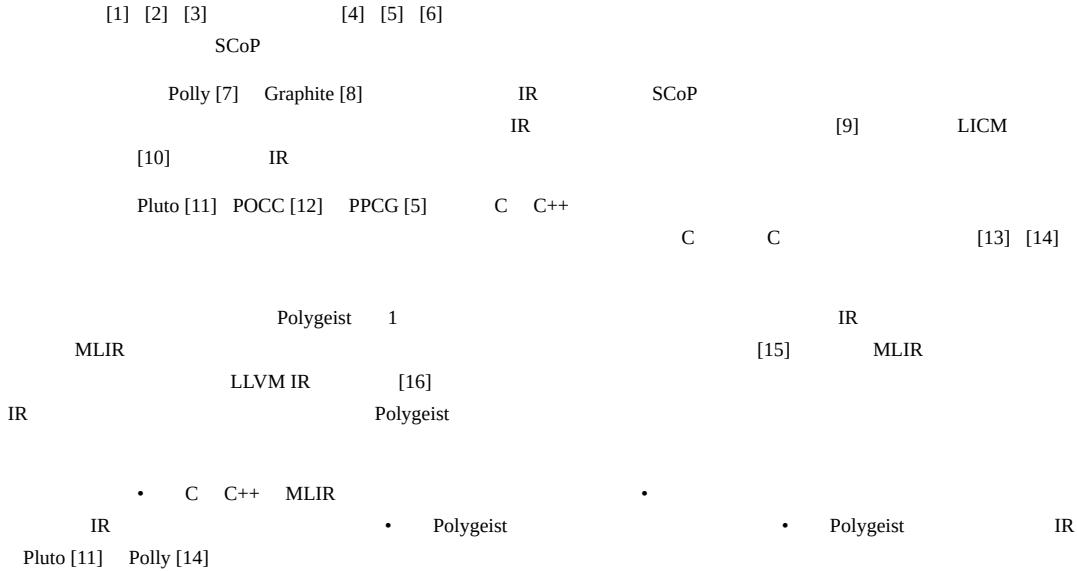
## # I. INTRODUCTION

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 SCoPs). 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.  
<sup>^</sup>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.

CPU GPU



## # 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.



## # B. Affine and MemRef Dialects

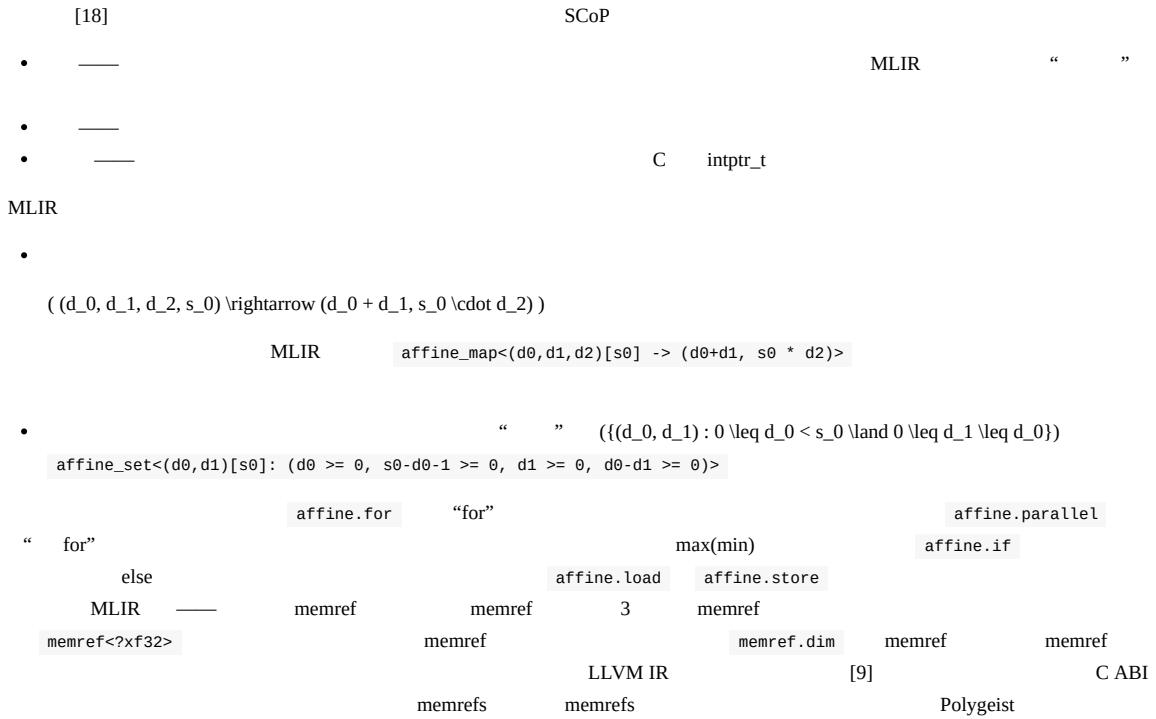
The Affine dialect [18] aims at representing SCoP's with explicit polyhedral-friendly loop and conditional constructs. The core of its representation is the following classification of value categories:

- 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,  $(d_0, d_1, d_2, s_0) \rightarrow (d_0 + d_1, s_0 * d_2)$  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.

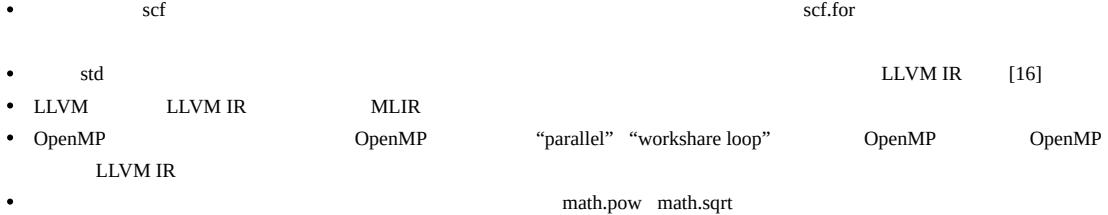


## # C. Other Relevant Core Dialects

MLIR provides several dozen dialects. Out of those, only a handful are relevant for our discussion:

- 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.

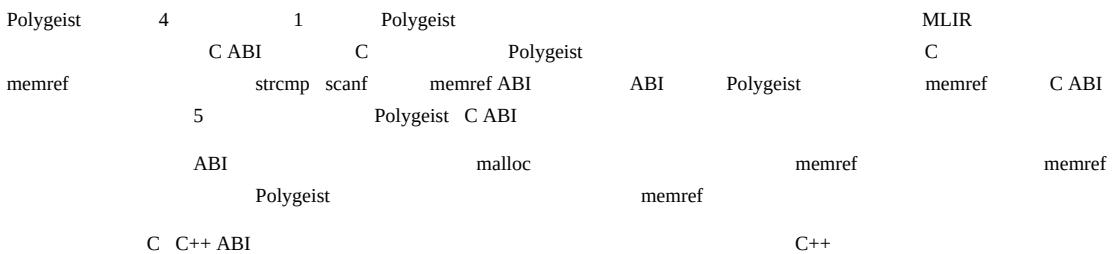
#### MLIR

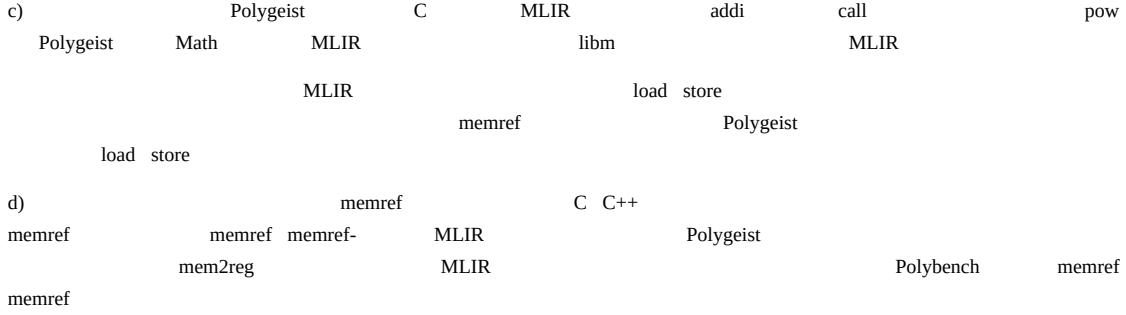


## # III. AN (AFFINE) MLIR COMPILATION PIPELINE

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. MLIR





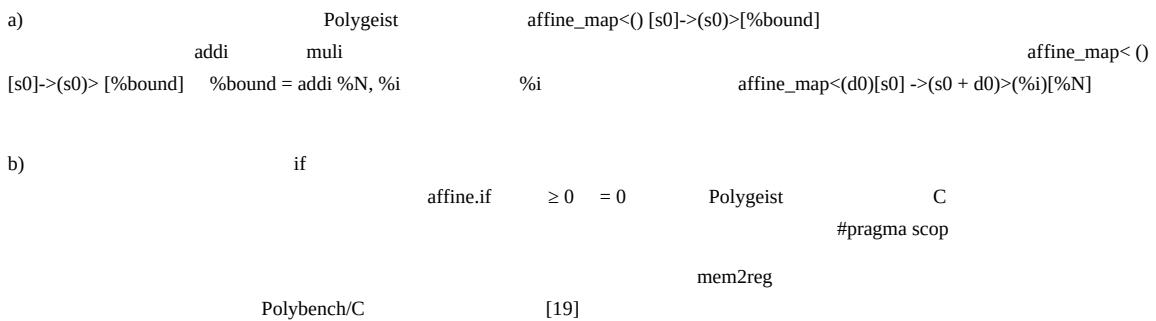
## # B. Raising to Affine

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. 1 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.



## # 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  $\geq 0$  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.



## # 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) } ↓ 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 = 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>) -> () } Fig. 5. Example demonstrating Polygeist ABI. For functions expected to be compiled with Polygeist such as setArray, pointer arguments are replaced with memref's. 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).

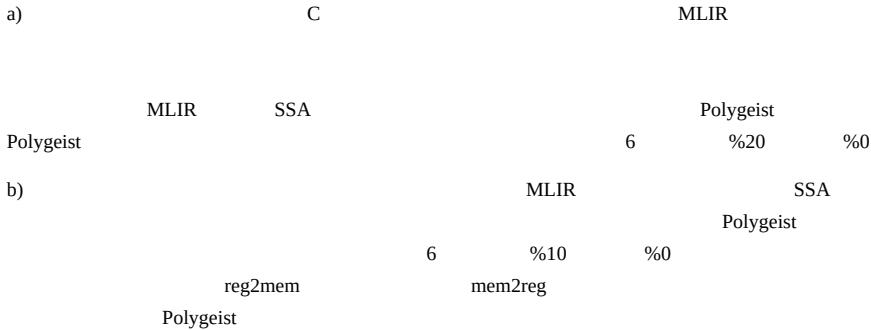


|             |               |           |           |        |
|-------------|---------------|-----------|-----------|--------|
| 5.          | Polygeist ABI | Polygeist | setArray  | memref |
| main(strcmp | Polygeist     | llvm.ptr  |           |        |
|             |               |           | Polygeist | III-D  |

## # C. Connecting MLIR to Polyhedral Tools Part-2

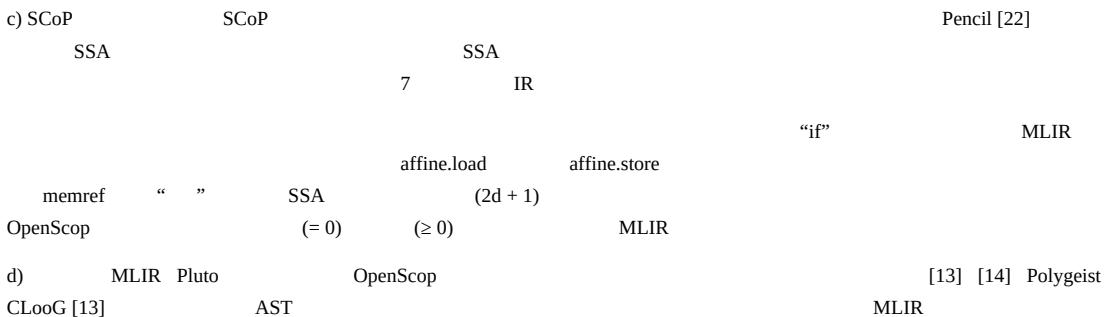
- a) Simple Statement Formation: Observing that C statements amenable to the polyhedral model are (mostly) variable assignments, we can derive a mechanism to identify statements from chains of MLIR operations. A store into memory is the last operation of the statement. The backward slice of this operation, 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.



## # 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 ( $\geq 0$ ) 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.



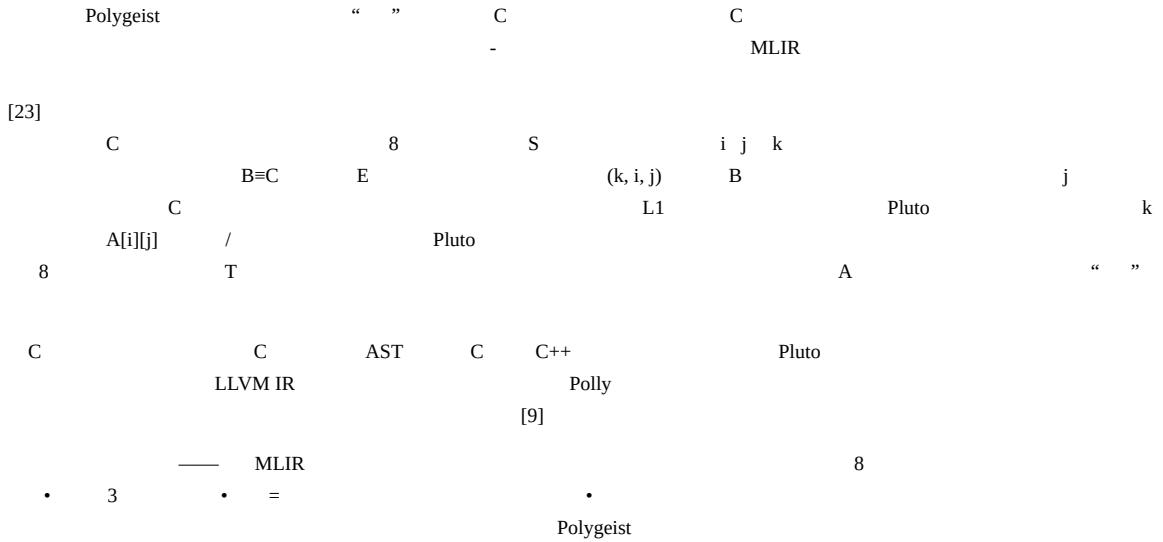
## # D. Controlling Statement Granularity

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.



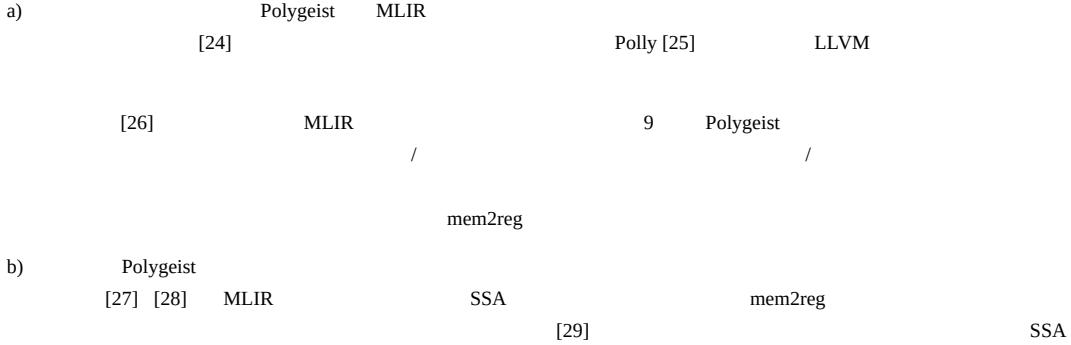
## # E. Post-Transformations and Backend

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.

Polygeist      quasi-syntactic      SSA  
                                              Polygeist



## # IV. EVALUATION

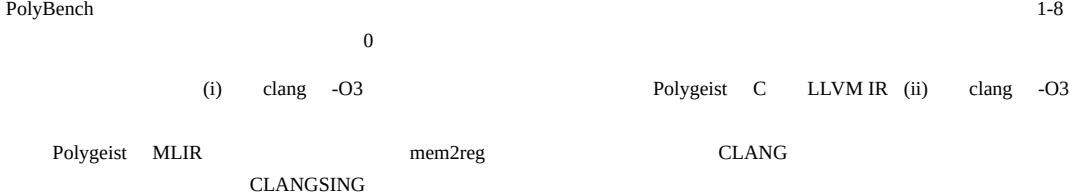
Our evaluation has two goals. 1) We want to demonstrate that the code produced by Polygeist without additional optimization does not have any inexplicable performance differences than a state-of-the-art compiler like Clang. 2) We explore how Polygeist's internal representation can support a mix of affine and SSA-based transformation in the same compilation flow, and evaluate the potential benefits compared to existing source and compiler-based polyhedral tools.



## # 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).

|              | AWS c5.metal |                 |    |    | Turbo Boost |         |       |          | Ubuntu 20.04 |  |  |   |
|--------------|--------------|-----------------|----|----|-------------|---------|-------|----------|--------------|--|--|---|
| CPU          | 3.0 GHz      | 24              | L1 | L2 | L3          | 0.75 MB | 35 MB | 35.75 MB | 256 GB RAM   |  |  |   |
| "EXTRALARGE" |              | Poly-Bench [19] |    | 30 |             | Pluto   | adi   |          | SCoP         |  |  | 5 |

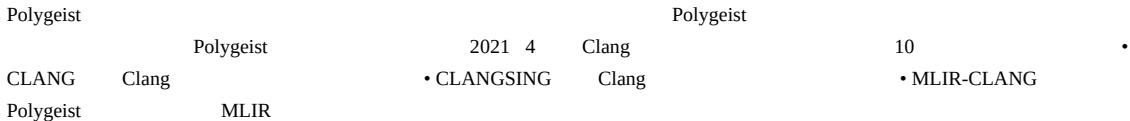


## # B. Baseline Performance

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.

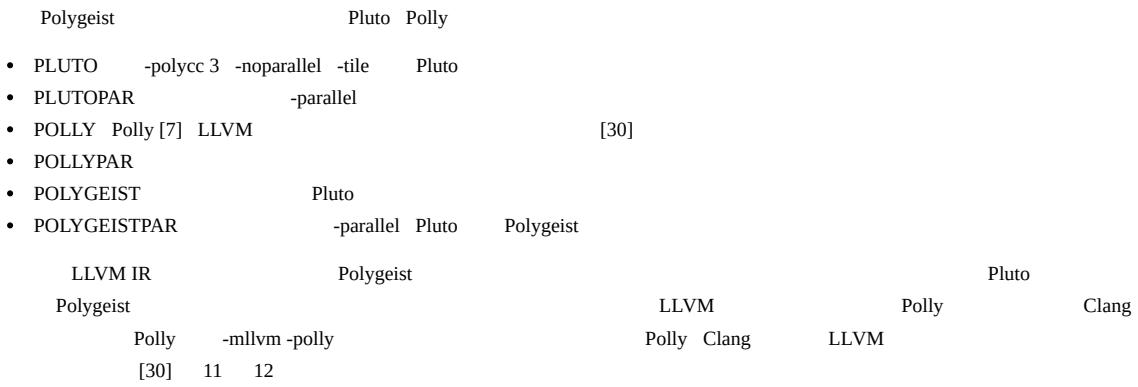


## # C. Compilation Flows

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.



## # 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.

|                                          |            |                                          |        |             |
|------------------------------------------|------------|------------------------------------------|--------|-------------|
|                                          |            | IEEE 754                                 |        | -ffast-math |
| Polybench                                | Clang/LLVM |                                          | 2 LLVM |             |
| 20d5c42e0ef5d252b434bcb610b04f1cb79fe771 | 3 Pluto    | dae26e77b94b2624a540c08ec7128f20cd7b7985 |        | adi         |
| 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.

|       |           |                   |             |
|-------|-----------|-------------------|-------------|
| CLANG | CLANGSING |                   | 0.43%       |
| CLANG |           | CLANG             | 0.24%       |
|       |           | MLIR-CLANG        |             |
|       |           | jacobi-1d         |             |
| 0.05  |           | gesummv atax bicg | 0.32% 0.17% |

## # C. Performance Differences in Sequential Code

Overall, Polygeist leads to larger speedups, with 2.53× geometric mean, than both Pluto (2.34×) and Polly (1.41×), 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•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 ≈ 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.

| Polygeist |           | Pluto | 2.34× | Polly |     | 1.41× | 2.53×     |     |     | Polygeist |       | Polly     |
|-----------|-----------|-------|-------|-------|-----|-------|-----------|-----|-----|-----------|-------|-----------|
| Polly     | V-E       |       |       | lu    | mvt |       | Polygeist | 2mm | 3mm | Pluto     | Pluto | Pluto     |
| 2.7•10^11 | /         |       |       |       |     |       | seidel-2d |     |     | Pluto     |       | Polygeist |
|           | Polygeist |       |       |       |     |       | add       | 1/2 | 1/4 | imul/shl  | 1     | 3GHz      |
|           | Polygeist |       |       |       |     |       |           |     |     | memref    |       | ≈ 59      |
| Clang     |           |       |       |       |     |       | CLOOG     |     |     | jacobi-2d | 40    | 75        |
|           |           |       |       |       |     |       |           |     |     | Polygeist |       |           |
|           |           |       |       | 5     |     |       |           |     |     |           |       |           |

## # 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× geomean speedup) both Pluto (2.34×) and Polly (1.41×) 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× geomean speedup) both Pluto (7.54×) and Polly (3.26×) 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

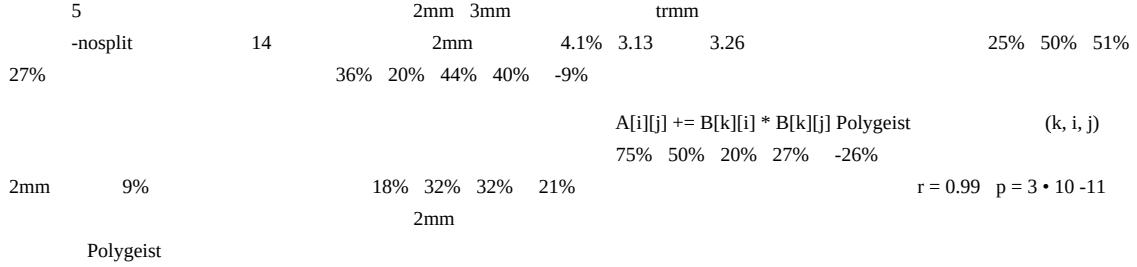
Polly. These results demonstrate that Polygeist indeed leverages the benefits of both the affine and SSA-based optimizations. Polygeist is the only flow that obtains speedup on deriche ( $6.9\times$ ) and symm ( $7.7\times$ ). Examining the output code, we observe that only Polygeist manages to parallelize 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].

|           |           |             |            | cholesky   | lu        | Pluto       | Polygeist | Polly     |
|-----------|-----------|-------------|------------|------------|-----------|-------------|-----------|-----------|
|           | gemver    | mvt         | Polly      |            |           |             |           |           |
| 10        | CLANG     | CLANGSING   | MLIR-CLANG |            | Polybench |             | 95%       |           |
|           | Polygeist |             |            | Clang      |           |             |           | jacobi-1d |
| 11        |           | CLANG       |            |            |           | Polygeist   |           | 2.53×     |
| 2.34×     | Polly     | 1.41×       | Pluto      | adi        |           |             |           | Pluto     |
| 12        |           | CLANG       |            |            |           | Polygeist   |           | 9.47×     |
| 7.54×     | Polly     | 3.26×       | Pluto      | adi        |           |             |           | Pluto     |
| ludcmp    | syr(2)k   | SSA         |            | Polygeist  | Pluto     | Polly       | Polygeist |           |
|           | SSA       |             |            |            |           |             |           |           |
| Polygeist | deriche   |             | 6.9×       | symm       | 7.7×      |             | Polygeist |           |
|           |           | 13          |            | i          | ym1       |             | Polygeist | ym1       |
|           | j         | SSA         |            |            |           |             |           |           |
|           | Polygeist |             |            |            |           | gramschmidt | durbin    | Polly     |
| Pluto     | 54×       | gramschmidt | Polygeist  |            | 56×       | durbin      | 6×        | 34×       |
|           |           |             | V-F        |            |           | durbin      |           |           |
| Polybench |           |             |            | CPU        |           |             |           |           |
| GPU       |           |             |            | [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.

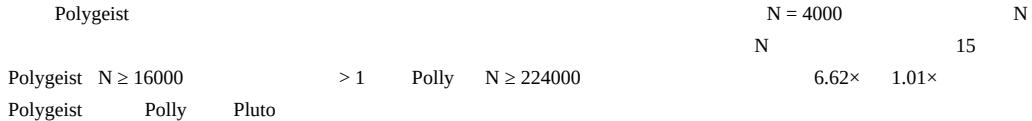
E.



## # 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.

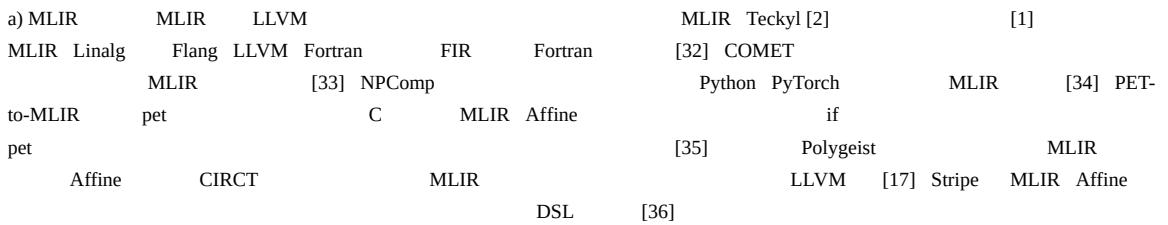
### F. Durbin



## # 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].

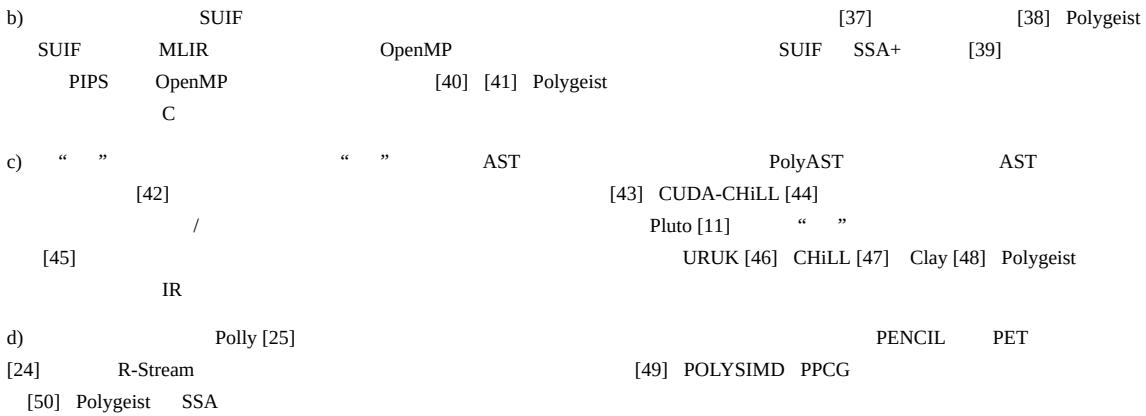
### VI.



## # b) Compilers Leveraging Multiple Representations:

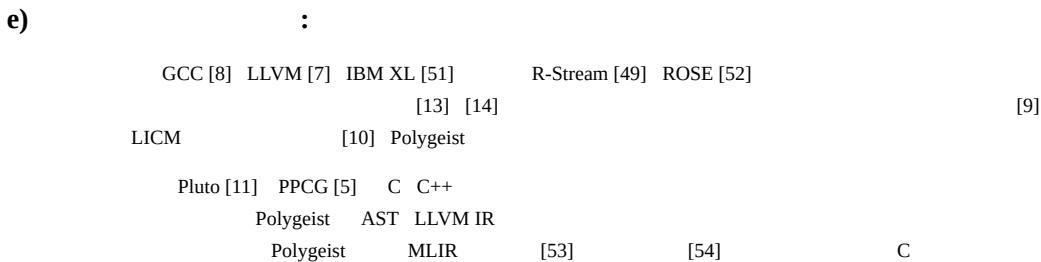
The SUIF compiler infrastructure pioneered a combined internal representation that supports higher-level transformations, including loop optimization and parallelization [37] and, in particular, reduction parallelization [38]. Polygeist leverages MLIR abstractions unavailable in SUIF: regular and affine for loops, OpenMP reduction constructs, etc. It also benefits from the SSA+regions form, which is only available as an 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.



## # 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 higherand 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].



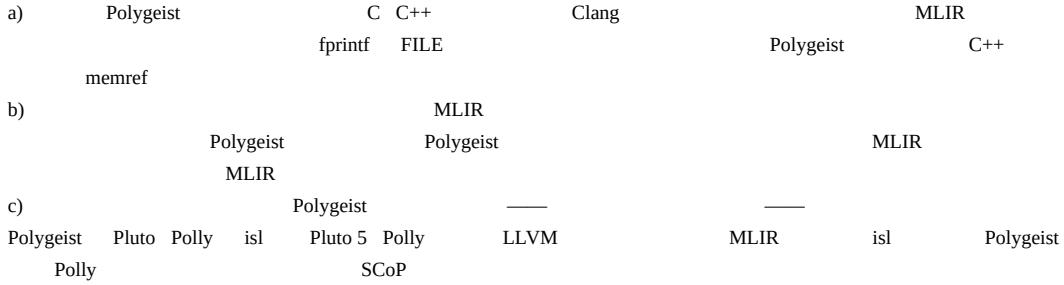
## # VII. DISCUSSION

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.

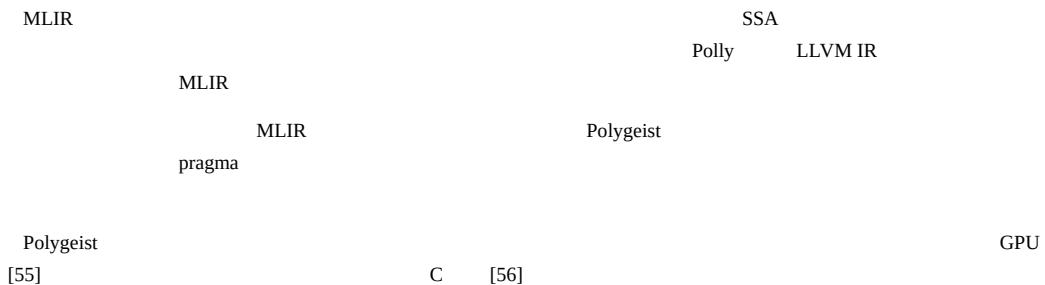
## VII.

### A.



## # B. Opportunities and Future Work

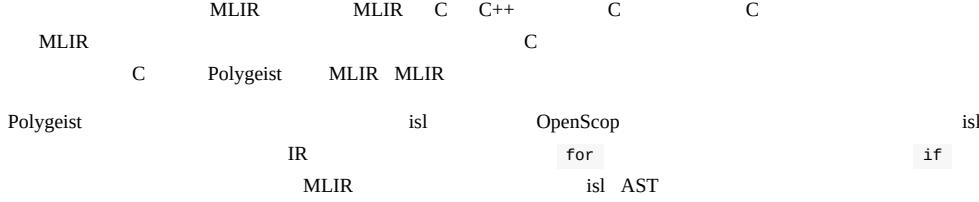
Connecting MLIR to existing polyhedral flows opens numerous avenues for compiler optimization research, connecting polyhedral and conventional SSA-based compiler transformations. This gives polyhedral schedulers access to important analyses such as aliasing and useful information such as precise data 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.



## # C. Alternatives

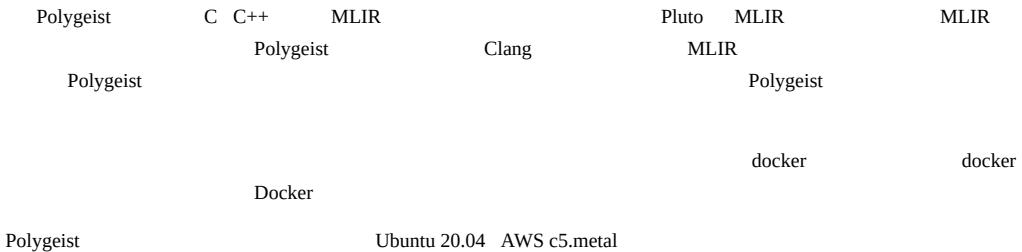
Instead of allowing polyhedral tools to parse and generate MLIR, one could emit C (or C++) code from MLIR 6 and use C-based polyhedral tools on the C source, but this approach decreases the expressiveness of the flow. Some MLIR constructs, such as parallel reduction loops, can be directly expressed in 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].



## # 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:



```
$ 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

Pluto
$ 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
```

```
$ cd
$ git clone \
https://github.com/wsmoses/Polygeist-Script\ scripts
```

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
```

LLVM MLIR

```

 omp.h

$ cd $ export OMP_FILE= find \ $HOME/mlir-clang/build -iname omp.h

$ cp $HOME/scripts/omp.h $OMP_FILE

MLIR LLVM

$ 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

MLIR

$ 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

```

AWS

```

$ 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.

```

taskset -c 1-8 numactl -i all
cd ~/scripts/ $ cd polybench-c-4.2.1-beta/ $./run.sh #

```

#

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.

## # 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 commandby-command description of the installation process, along with

|           |           |        |
|-----------|-----------|--------|
| Polygeist | Polybench | Docker |
|-----------|-----------|--------|

## Polygeist: Raising C to Polyhedral MLIR

Published in: 2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)

2021 30

PACT

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.

|                | Polygeist | MLIR          | C    | C++               |
|----------------|-----------|---------------|------|-------------------|
|                | MLIR      | MLIR OpenScop | SSA  |                   |
| Polygeist/MLIR |           | SSA           |      |                   |
| Polybench/C    | Polygeist | 2.53 vs 1.41  | 2.34 | 9.47 vs 3.26 7.54 |
| LLVM IR        | Polly     | Pluto         |      |                   |

[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](#)

GPU

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.

GPU

GPU

CUDA

| GPU  | MLIR           | CUDA    | AMD GPU | CUDA |
|------|----------------|---------|---------|------|
| CUDA | NVIDIA AMD GPU | Rodinia | CUDA    | 27%  |

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)

- Polygeist                                    MLIR                            Floyd-Warshall                    8%
- MLIR                                    LLVM                            MLIR
- adi                            30%
- LLVM                                    ADI

## Polgeist MLIR Compiler Frontend

```
Polygeist Polygeist bridging the gap between C/C++ and MLIR :
C/C++ : C C++ MLIR : C/C++ MLIR : MLIR
 : GPU : CUDA ROCm , GPU : Polygeist
C/C++ MLIR

Polygeist Polygeist :
C/C++ : Clang C/C++ AST : (AST) ,
AST , MLIR : MLIR , :
 : MLIR LLVM IR , Polygeist MLIR ,
C/C++

Polygeist Polygeist C/C++ :
 : , GPU : CUDA ROCm , GPU LLVM : LLVM ,
LLVM Polygeist

Polygeist :
 : , Polygeist kernel , , 30%
 : Polygeist C++ , , 20%
 : Polygeist GPU , CUDA ,
Polygeist
```

```
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
- MLIR                                    ,          (          GPU          LLVM !)          /

### 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
- $( \quad )$
- $/$
- $/$

## 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).

• sync\_threads                      CodeB                      CodeA

- sync\_threads
- 

- 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.

•

•        /                      /                      synchronize

```
__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.

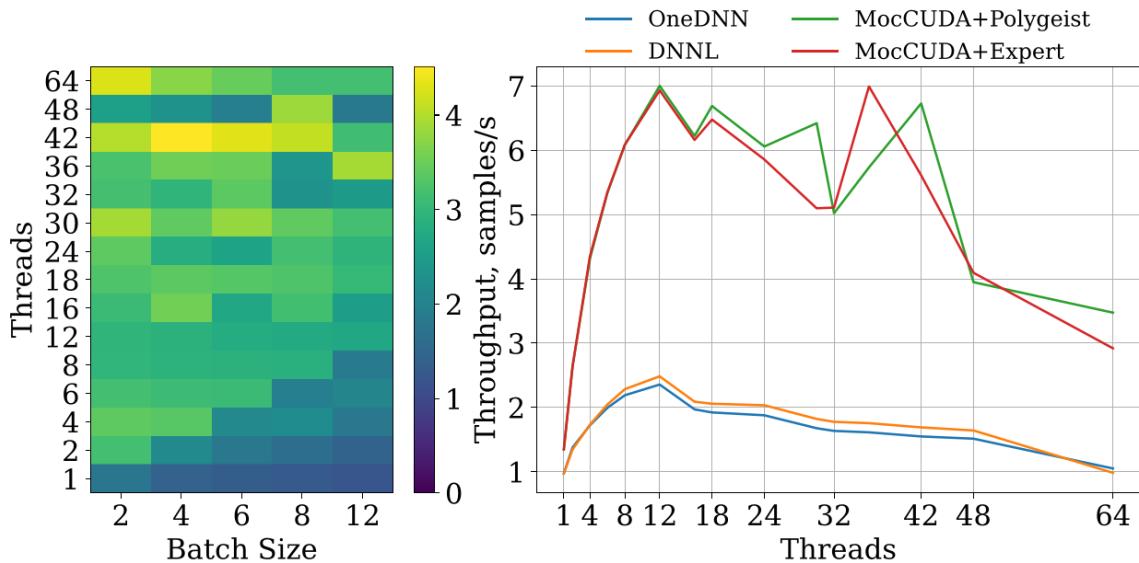
```

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);
 }
}

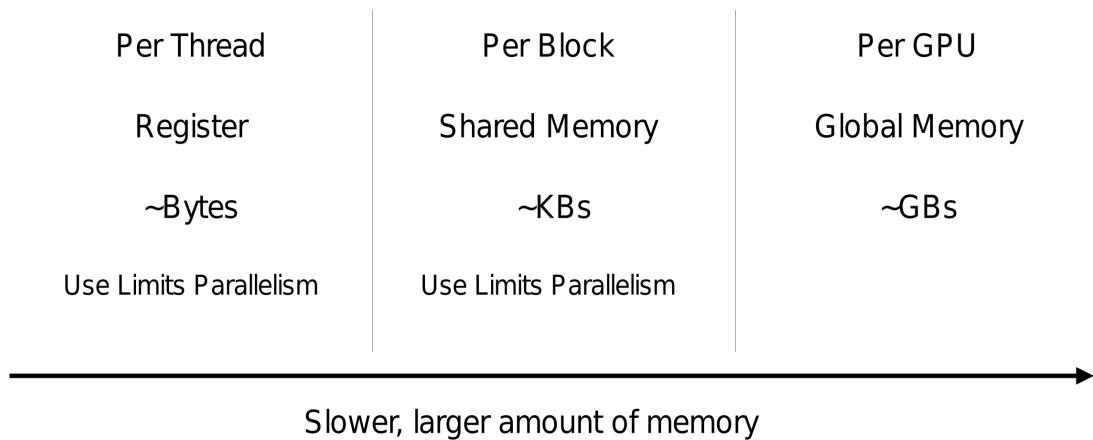
```

## 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.

*affinis “ ”*

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.

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.

X                    X                    X                    X

Examples of affine transformations include translation, scaling, homothety, similarity, reflection, rotation, hyperbolic rotation, shear mapping, and compositions of them in any combination and sequence.

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.

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]

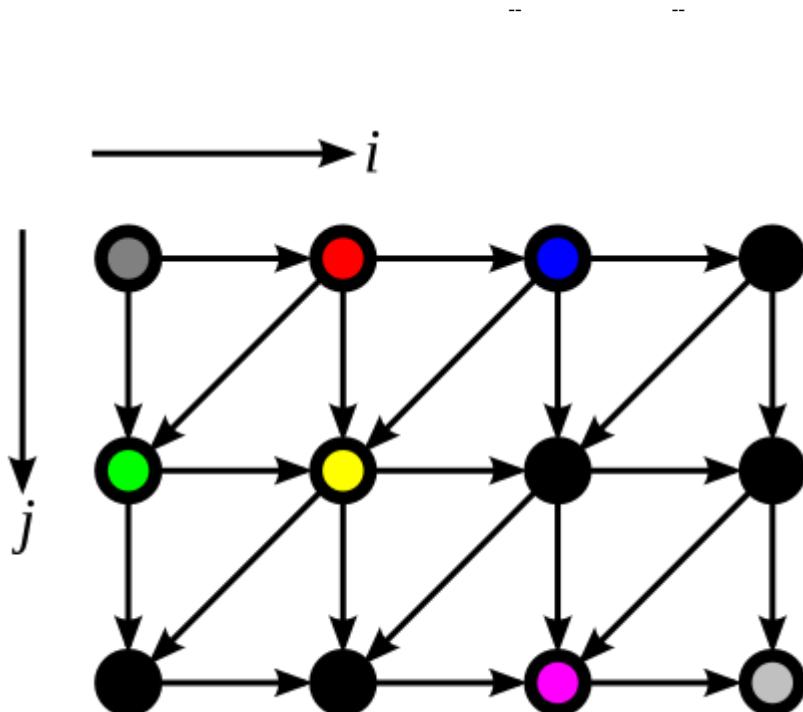
|                                |     |                              |                                         |                                                 |                      |
|--------------------------------|-----|------------------------------|-----------------------------------------|-------------------------------------------------|----------------------|
| $X \quad Z$<br>$\rightarrow Z$ | $k$ | $1$<br>$mf: V \rightarrow W$ | $X \quad V \quad k$<br>$xf \quad x - y$ | $Z \quad W \quad k$<br>$=f \quad x - f \quad y$ | $X.2$<br>$x \quad y$ |
|--------------------------------|-----|------------------------------|-----------------------------------------|-------------------------------------------------|----------------------|

## 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.



## Polyhedral model in programming

[Frameworks supporting the polyhedral model](#)

- 1.
- 2.
- 3.

- 1.
- 2.
3.
  - 
  - 
  - **Tiling**                      tile
  -

- 1.
  - 2.
  3.                                            SIMD
  - 4.
  - 5.
  6.                                            GPU
- 
- - 
  -
- 
- -

```

for (i = 0; i < N; i++) {
 for (j = 0; j < N; j++) {
 A[i][j] = B[i][j] + C[i][j];
 }
}

```

i     j

---

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.

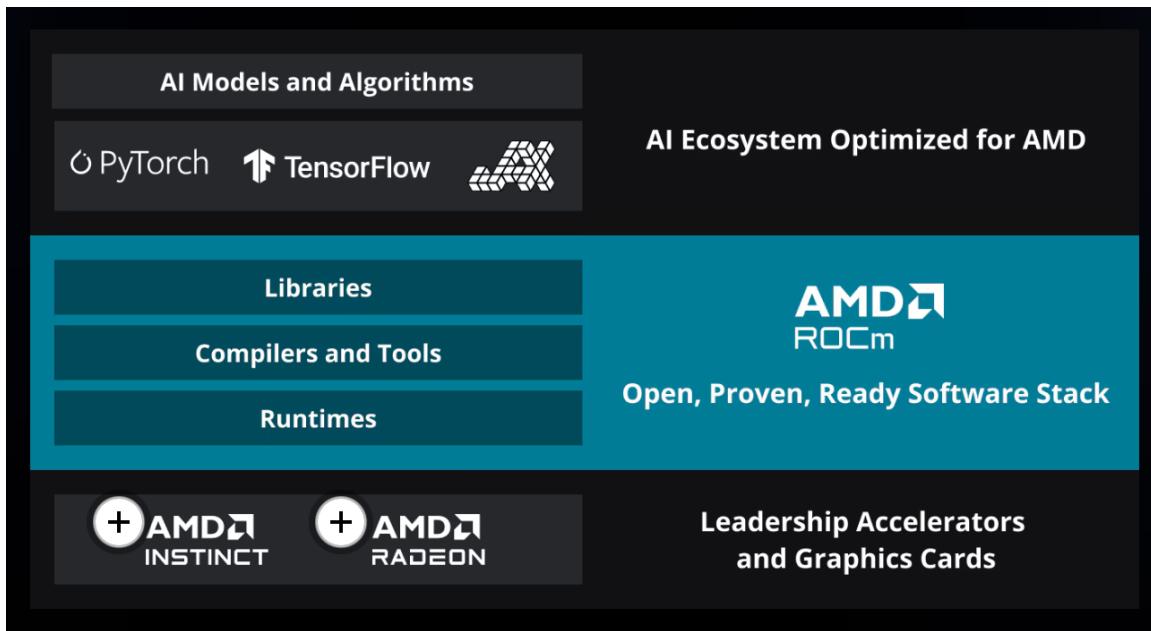
## Conclusion

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      GPU

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™      API      GPU      ROCm      AI      HPC



ROCm 3      (GPU)      Advanced Micro Devices (AMD)      ROCm      (GPGPU)  
 (HPC)      HIP      GPU      OpenMP      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 <a href="#">Jaws</a> 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](#)
- [wiki.ros: cn Introduction](#)
- [github: ros-infrastructure](#)
- [github: ROS core stacks](#)
- [github: ros-dpg](#)
- [rep: learn-ros](#)
- [blog: ros-tutorials](#)

ROS

## ROS Releases/Distributions

- [ros: Distributions](#)
- [ros2: Releases](#)
- [ros - os version match](#)
- 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.
    - 
    - 
    - 
    - 
    - 
    - `/` : 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                                                                                                                                                  | ROS              | ROS | ROS           |
|------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|-----|---------------|
|                                                                                                                                                      | Parameter Server |     | C++    Python |
| <ul style="list-style-type: none"> <li>• <code>roscpp</code></li> <li>• <code>rospy</code></li> <li>• <code>roslisp</code></li> <li>• ...</li> </ul> |                  |     |               |

## 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 | ROS      | rosbuild | catkin | CMake | Python | CMake |
| Catkin | rosbuild |          |        |       | catkin | CMake |
| “ ”    |          |          |        |       |        |       |

```

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] http://
```

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/master/ros.key | sudo tee /usr/share/keyrings/ros-archive-ke
```

## 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-vcstool wget
```

### 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:

```
rosdep install --from-paths src --ignore-src --rostdistro humble -y --skip-keys "libpython3.10 libopencv-core4.5d libpc
```

## 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 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

, 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:                    'input

```
pub enum ExprSymbol<'input> {
 NumSymbol(&'input str),
 Op(Box<ExprSymbol<'input>>, Opcode, Box<ExprSymbol<'input>>),
 Error,
}
```

### GPT

'input                    ExprSymbol

```
1. ExprSymbol 'input
2. NumSymbol(&'input str) ExprSymbol
```

'input

```
fn parse_expr<'a>(input: &'a str) -> ExprSymbol<'a> {
 ExprSymbol::NumSymbol(input) // `input`
}
```

- parse\_expr            ExprSymbol                    &'a str                    input
- parse\_expr        input                    ExprSymbol                    input

Rust                    ExprSymbol

```
1. ExprSymbol Rust
2. &str
```

```
pub enum ExprSymbol {
 NumSymbol(&str), //
 Op(Box<ExprSymbol>, Opcode, Box<ExprSymbol>),
 Error,
}
```

```
error[E0106]: missing lifetime specifier
--> src/main.rs:2:17
 |
2 | NumSymbol(&str),
| ^ expected named lifetime parameter
```

Rust

&amp;str

```
fn example<'a>(input: &'a str) -> ExprSymbol<'a> {
 ExprSymbol::NumSymbol(input)
}

fn main() {
 let input = String::from("123");
 let symbol = example(&input); // input symbol
 println!("{}:{}", symbol);
 // input symbol input symbol
}
```

```
pub enum ExprSymbol {
 NumSymbol(&str), //
 Op(Box<ExprSymbol>, Opcode, Box<ExprSymbol>),
 Error,
}
```

&amp;str

- 1.
2.                   Rust
3.

## **Problems and Solutions**

## Rust Resources

### Rust

- [The Rust Reference](#)
- [Rust Ref: zh](#)
- [~~](#)
- [The Rust Programming Language](#)
- [The Rust Programming Language: Experiment Type](#)
- [Rust \(zh\) - 2022](#)
- [Rust \(zh\) - 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 \(Rust Course\)](#)
- [rust-by-example](#)
- [The Cargo Book](#)
- [The Little Book of Rust Macros](#)
- [~~](#)
- [Rust RFCs - RFC Book](#)
- [~~](#)
- [releases](#)
- [github releases](#)
- [~~](#)
- [Rust](#)
- [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](#)
- [rust-clif](#)
- [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 *owner*
- 2.
- 3.

- 
- 

## Slice

slice slice

```
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` `String` slice
 let word = first_word(&my_string[0..6]);
 let word = first_word(&my_string[...]);
 // `first_word` `String`
 // `String` slice
 let word = first_word(&my_string);

 let my_string_literal = "hello world";

 // `first_word`
 let word = first_word(&my_string_literal[0..6]);
 let word = first_word(&my_string_literal[...]);

 // ** ** slice
 // slice
 let word = first_word(my_string_literal);
}

```

## Struct

### field init shorthand

```

fn build_user(email: String, username: String) -> User {
 User {
 active: true,
 username,
 email,
 sign_in_count: 1,
 }
}

```

### struct update syntax

```

fn main() {
 // --snip--
 let user2 = User {
 email: String::from("another@example.com"),
 ..user1
 };
}

```

|     |          |               |            |          |               |        |               |
|-----|----------|---------------|------------|----------|---------------|--------|---------------|
| 5-7 | user2    |               | email      | user1    | username      | active | sign_in_count |
|     | user1    | ..user1       |            | user1    |               |        |               |
|     |          |               | =          | "        | "             |        |               |
|     | user2    | user1         | user1      | username | String        | user2  | user2         |
|     | username | String        | user1      | active   | sign_in_count | user1  | email         |
|     | active   | sign_in_count | Copy trait | "        | "             |        |               |

## 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);
}
```

## unit-like structs

- 
- trait

```
struct AlwaysEqual;

fn main() {
 let subject = AlwaysEqual;
}
```

sdsd

dsd

## Rust Book

Rust

- `Cargo`
- `Rustfmt`
- `rust-analyzer`
- `rustup`

Rust

IDE  
Rust

# Rust Language Reference

## Ref

### Traits

```
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`
  - - `functions`
    - `types`
    - `constants`
  - `trait`                    `module`    `block`                    `trait`  
        `Self`                “                ”                              `Self`
  - `trait`                    `trait`
  - 
  - 
  - 
  - 
  - `trait function`        `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

traits      type parameters      bounds

## Trait and lifetime bounds

[Trait](#) and lifetime bounds provide a way for [generic items](#) to restrict which types and lifetimes are used as their parameters. Bounds can be provided on any type in a [where clause](#). There are also shorter forms for certain common cases:

where

- Bounds written after declaring a [generic parameter](#):
  - `fn f<A: Copy>() {}` is the same as `fn f<A>() where A: Copy {}`.
  - 
  - `fn f<A: Copy>() {} fn f () where A: Copy {}`
- In trait declarations as [supertraits](#):
  - `trait Circle : Shape {}` is equivalent to `trait Circle where Self : Shape {}`.
  - [supertraits](#)
  - `trait Circle : Shape {} 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; }`.
  - 
  - `trait A { type B: Copy; } 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`

`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.
  - `Ty Trait`
- Associated types from `Trait` can be used.
  - `Trait`
- Generic functions and types with a `T: Trait` bounds can be used with `Ty` being used for `T`.
  - `T: Trait Ty T`

## Trait objects

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](#).

(object safe base trait)

(auto traits)

Trait objects implement the `base trait`, its `auto traits`, and any [supertraits](#) of the base trait.

`Trait`

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`      `dyn`      ?

`?Sized`

For example, given a trait `Trait`, the following are all trait objects:

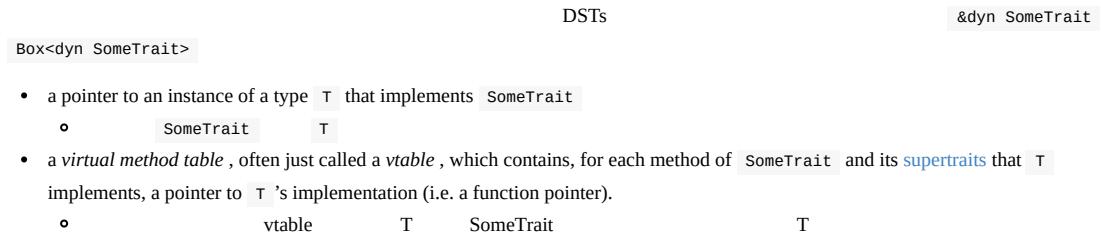
`Trait`

- `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:



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.

“ ” vtable vtable

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`.

| Printable                    | print | main |
|------------------------------|-------|------|
| Trait Object Lifetime Bounds |       |      |

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.

lifetime      `Trait + 'a` .

## Attributes

- [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

- [derivable-traits](#)

| <code>derive</code>                                           | <i>MetaListPaths</i>   | trait                                     |
|---------------------------------------------------------------|------------------------|-------------------------------------------|
|                                                               | <code>Foo</code>       | <code>PartialEq</code> trait              |
|                                                               | <code>PartialEq</code> | <code>Clone</code> trait                  |
|                                                               | <code>^1</code>        | (impl item)                               |
|                                                               |                        | <code>T</code>                            |
|                                                               |                        | (impl)                                    |
| <code>#[test]</code>                                          |                        |                                           |
| <code>#[should_panic(expected = "values don't match")]</code> |                        |                                           |
| <code>fn mytest() {</code>                                    |                        |                                           |
| <code>assert_eq!(1, 2, "values don't match");</code>          |                        |                                           |
| }                                                             |                        |                                           |
| 1                                                             | traits                 | Rust                                      |
| • <code>Debug</code>                                          | Debug trait            | <code>#[derive()]</code>                  |
|                                                               |                        | <code>println!("{}:{}", my_struct)</code> |

- `Clone`      `Clone` trait      `my_struct.clone()`
- `PartialEq`    `Eq`      `PartialEq` trait      `Eq` trait
- `PartialOrd` `Ord`      `PartialOrd` trait      `Ord` trait

```
2 traits traits traits #[derive()]
MyTrait trait
```

```
trait MyTrait {
 // trait
}
```

```
#[derive(MyTrait)]
struct MyStruct {
 //
}
```

traits                    MyTrait trait                    #[derive(MyTrait)]

```
trait MyTrait {
 // trait
}
```

```
struct MyStruct {
 //
}
```

```
impl MyTrait for MyStruct {
 // MyTrait
}
```

```
#[derive(Debug)]
```

```
#[derive(Debug)]
struct Rectangle {
 width: u32,
 height: u32,
}
```

```
fn main() {
 let rect1 = Rectangle {
 width: 30,
 height: 50,
 };
```

```
 println!("rect1 is {rect1:?}");
}
```

```
{} ?: println! Debug Debug trait
```

```
{:#?} println! {:#?} {:#?}
```

```
$ 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,
}
```

| Debug | dbg! | dbg!   | println! |
|-------|------|--------|----------|
| dbg!  |      | stderr | println! |
|       |      |        | 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

```

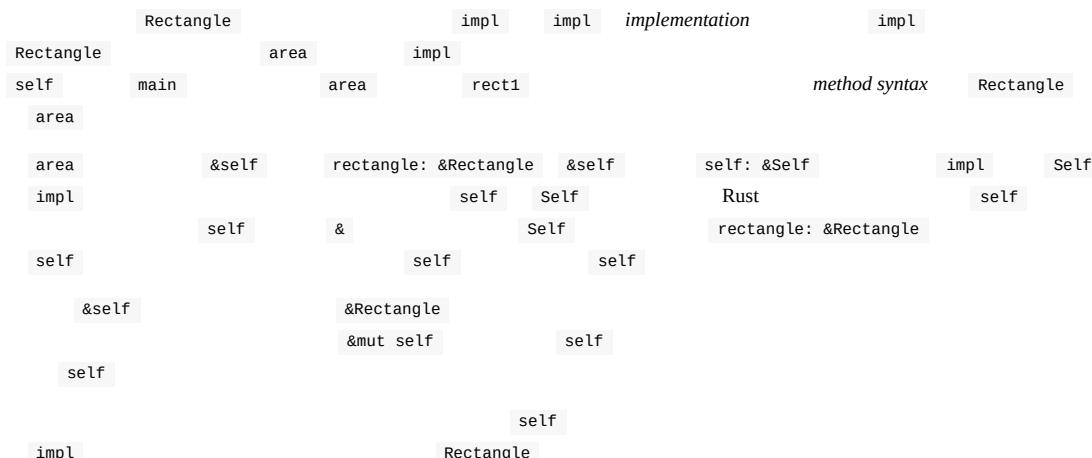
#[derive(Debug)]
struct Rectangle {
 width: u32,
 height: u32,
}

impl Rectangle {
 fn area(&self) -> u32 {
 self.width * self.height
 }
}

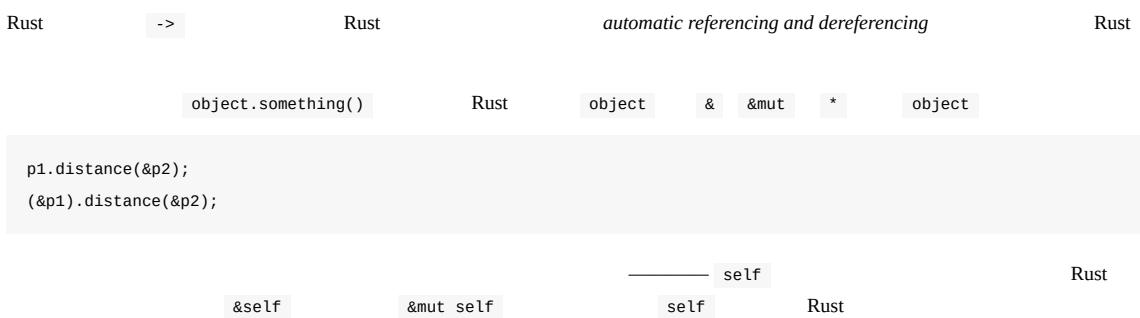
fn main() {
 let rect1 = Rectangle {
 width: 30,
 height: 50,
 };

 println!(
 "The area of the rectangle is {} square pixels.",
 rect1.area()
);
}

```



## automatic referencing and dereferencing



```

new new
square Rectangle

impl Rectangle {
 fn square(size: u32) -> Self {
 Self {
 width: size,
 height: size,
 }
 }
}

Self impl Rectangle
::: let sq = Rectangle::square(3); :::

```

## Option

Tony Hoare null 2009 “Null References: The Billion Dollar Mistake”

I call it my billion-dollar mistake. At that time, I was designing the first comprehensive type system for references

Rust Option<T>

```

enum Option<T> {
 None,
 Some(T),
}

```

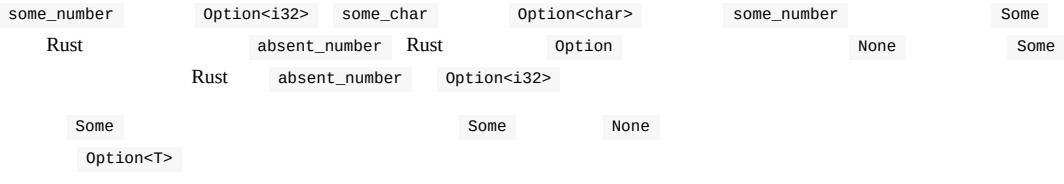
Option<T> prelude Option::  
Some None Option<T> Some(T) None Option<T>  
<T> Option  
Some T Option<T>  
Option

```

let some_number = Some(5);
let some_char = Some('e');

let absent_number: Option<i32> = None;

```



```

let x: i8 = 5;
let y: Option<i8> = Some(5);

let sum = x + y;

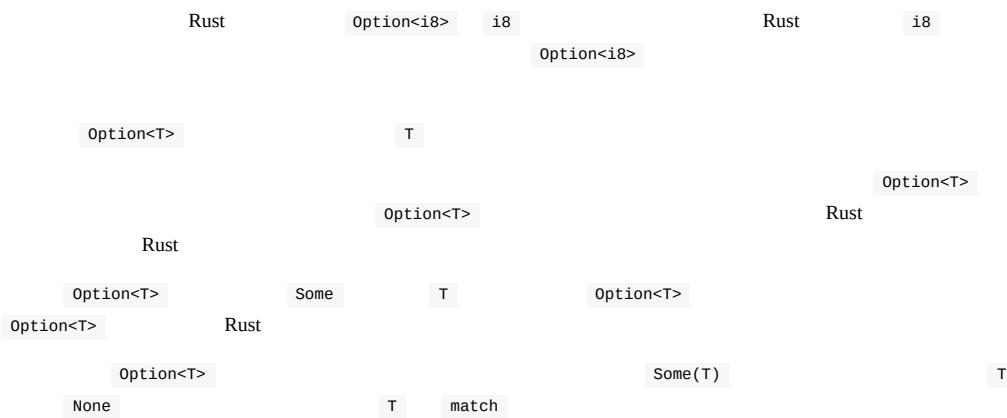
```

```

$ 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

```



## Rust Programming Language

### Ch3: Common Programming Concepts

- Shadowing:

### Ch9:

Rust

- recoverable:
  - `Result`
- unrecoverable: panic
  - `panic!`
  - `backtrace`

#### **Result** (recoverable)

```
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:?}"),
 };
}
```

```
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:?}");
 }
 },
 };
}
```

unwrap\_or\_else

```

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);
 }
 });
}

```

## panic      unwrap    expect

| Result | Ok | unwrap | Ok | Result | Err | unwrap | panic! |
|--------|----|--------|----|--------|-----|--------|--------|
|--------|----|--------|----|--------|-----|--------|--------|

```

use std::fs::File;

fn main() {
 let greeting_file = File::open("hello.txt").unwrap();
}

```

|        |        |        |        |        |        |        |
|--------|--------|--------|--------|--------|--------|--------|
| expect | unwrap | panic! | expect | panic! | expect | unwrap |
|        | panic! |        |        |        |        |        |

```

use std::fs::File;

fn main() {
 let greeting_file = File::open("hello.txt")
 .expect("hello.txt should be included in this project");
}

```

## 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),
 }
}

```

|        |   |     |        |       |        |    |    |
|--------|---|-----|--------|-------|--------|----|----|
| Result | ? | 9-6 | Result | match | Result | Ok | Ok |
|        |   | Err | Err    |       | return |    |    |

```

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)
}

```

|   |      |            |   |      |
|---|------|------------|---|------|
| ? | from | From trait | ? | from |
|   |      |            |   |      |

```

9-7 read_username_from_file OurError impl From for OurError
io::Error OurError read_username_from_file ? from
?
```

9-8

```

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:

|           |      |      |        |      |        |      |        |      |
|-----------|------|------|--------|------|--------|------|--------|------|
| Option<T> | ?    | ?    | ?      | ?    | ?      | ?    | ?      | ?    |
| Result    | None | None | Result | None | Option | Some | Option | None |

```

use std::fs;
use std::io;

fn read_username_from_file() -> Result<String, io::Error> {
 fs::read_to_string("hello.txt")
}

```

|        |      |      |        |      |        |      |        |      |
|--------|------|------|--------|------|--------|------|--------|------|
| Option | ?    | ?    | ?      | ?    | ?      | ?    | ?      | ?    |
| Result | None | None | Result | None | Option | Some | Option | None |

9-11

```

fn last_char_of_first_line(text: &str) -> Option<char> {
 text.lines().next()?.chars().last()
}

```

|        |   |   |   |   |   |   |   |   |
|--------|---|---|---|---|---|---|---|---|
| Option | ? | ? | ? | ? | ? | ? | ? | ? |
|        |   |   |   |   |   |   |   |   |

```

Option
last_char_of_first_line None text next text slice next lines
?
```

```
? slice slice chars last
 Option text "nhi"
Some ? Option ?
match
Result Option Result ? Option ? ?
Result Option Result ok Option ok_or ?
main Result<(), E> 9-12 9-10 main Result<(), Box<dyn Error>>
Ok()
```

```
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> trait trait object trait "
" Box<dyn Error> main Result ? Box<dyn Error> "
main
 Box<dyn Error> main Result ? Err main
 std::io::Error Box<dyn Error> main
 main Err
main Result<(), E> main Ok(())
0 main Err
0 Rust
main std::process::Termination trait ExitCode report
Termination trait
panic! Result
```

```
pub struct Guess {
 value: i32,
}

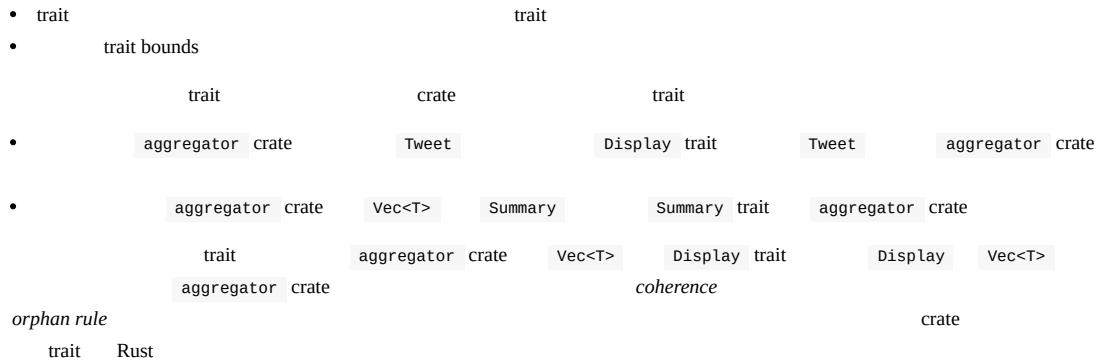
impl Guess {
 pub fn new(value: i32) -> Guess {
 if value < 1 || value > 100 {
 panic!("Guess value must be between 1 and 100, got {value}.");
 }
 Guess { value }
 }

 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



## Validating References with Lifetimes:

- dangling reference
  - borrow checker:
- 
- 
- 
- 
- Lifetime Elision
- 
- 
- trait bounds
- 

```
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 whether it is borrowed
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

|      |        |    |   |
|------|--------|----|---|
| Rust | x    y | if | x |
| else | y      |    |   |

10-21

'a

```
fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
 if x.len() > y.len() {
 x
 } else {
 y
 }
}
```

'a

slice

longest

'a

slice

Rust

longest

x    y

'a

Rust

longest

'a

x

y

'a

'a

x    y

Rust

```
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,
 };
}
```

part

slice

ImportantExcerpt

part

```
main ImportantExcerpt novel String novel
ImportantExcerpt ImportantExcerpt novel ImportantExcerpt
```

## Lifetime Elision

Rust

*input lifetimes*                            *output lifetimes*

| fn   | impl                                   |  |  |      |
|------|----------------------------------------|--|--|------|
| •    |                                        |  |  |      |
| i32) | fn foo<'a, 'b>(x: &'a i32, y: &'b i32) |  |  |      |
| •    | fn foo<'a>(x: &'a i32) -> &'a i32      |  |  |      |
| •    | &self      &mut self      (method)(    |  |  | rust |
| 17 ) | self                                   |  |  |      |

```
impl<'a> ImportantExcerpt<'a> {
 fn announce_and_return_part(&self, announcement: &str) -> &str {
 println!("Attention please: {}", announcement);
 self.part
 }
}
```

'static                                    'static

## trait bounds

```
use std::fmt::Display;

fn longest_with_an_announcement<'a, T>(
 x: &'a str,
 y: &'a str,
 ann: T,
) -> &'a str
where
 T: Display,
```

trait   trait bounds

## Ch11: Writing Automated Tests

- How to write
  - Controlling How Tests Are Run
  - Test Organizations

## How to write tests

- •  
•

Rust 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)] : cargo test
• use super::*;

• #[test] : cargo test
• assert_eq!(result, 4); 4
• cargo test :

• assert!(expression) : true

• assert_ne!(exp1, exp2) : exp1 != exp2
• assert_eq!(exp1, exp2) : exp1 == exp2
• assert_approx_eq!(exp1, exp2, epsilon) : exp1 exp2 epsilon
• assert_ne_precise!(exp1, exp2) : exp1 exp2 f32::EPSILON f64::EPSILON
• assert_xx(abc, def, info) : abc def xx info
```

```

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(200);
    }
}

```

- `#[should_panic] : panic`
- `#[should_panic(expected = "less than or equal to 100")] : panic "less than or equal to 100"`

Result<T, E>

```

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` `Result<(), String>`
- `Ok(())` `Err(String)`

Controlling How Tests Are Run

- `cargo test`
- `cargo test` `--`
- `cargo test --help` `cargo test` `cargo test -- --help`
-
- - `cargo test -- --test-threads=1`
- - `cargo test -- --show-output`
- - `cargo test one_hundred :` `one_hundred`
 - `cargo test add :` `add`
- - `#[ignore]`
 - `cargo test -- --ignored`
 - `cargo test -- --include-ignored`

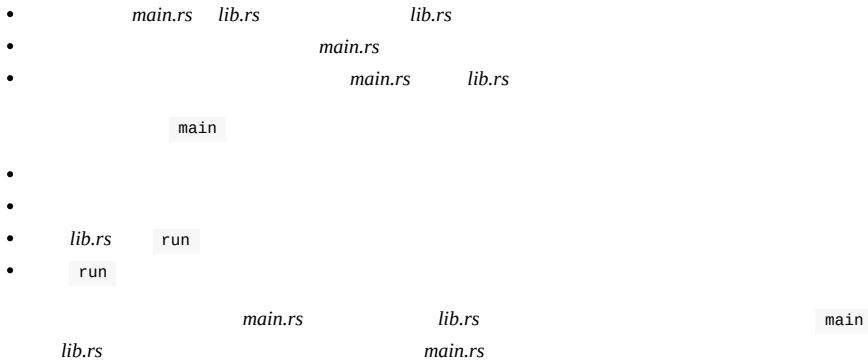
Ch12: An I/O Project

- parse cmd args
- read the file
- reconstruct: dispatch
- tests driver'
- env variables
- stderr

main

Rust

main



Ch13: Functional Language Features: Iterators and Closures

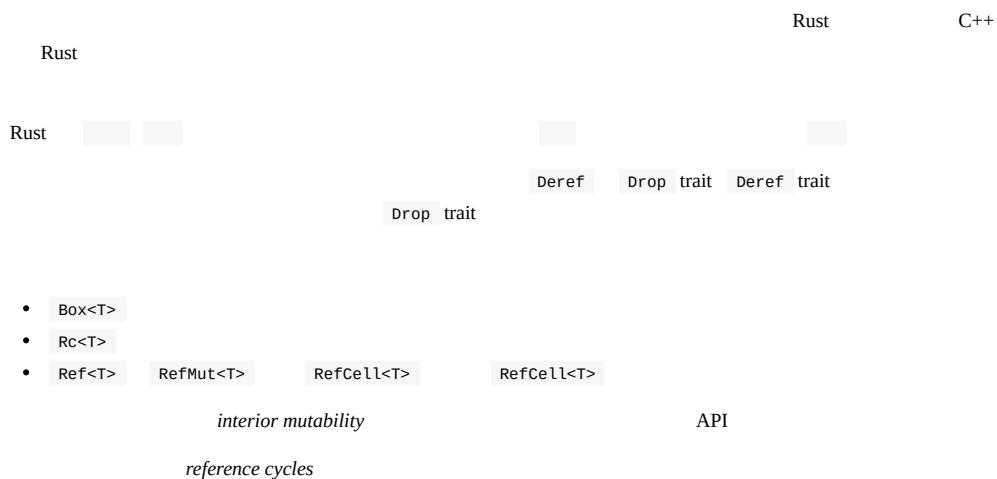
Closures: Anonymous Functions that capture their Environment

Iterators

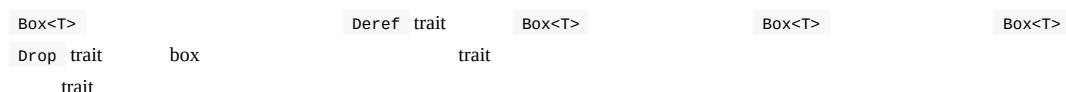
Ch14: More about Cargo and Creates.io

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
- Reference Cycles Can Leak Memory



Box<T>



Deref trait



* * deref *

i32 15-9 assert_eq! 5 *

Deref

Deref	<i>deref coercions</i>	Deref trait	Rust	Deref	<i>&String</i>
&str	String	Deref trait	&str	Deref	deref
Deref trait					
Deref	Rust	& *			

Deref

Deref trait	*	Rust	DerefMut trait	*
Rust	trait	Deref		
<ul style="list-style-type: none"> • T: Deref<Target=U> &T &U • T: DerefMut<Target=U> &mut T &mut U • T: Deref<Target=U> &mut T &U 				
&T T U Deref &U				
Rust	Rust			

Drop Trait

...

Rc

RefCell

Ch16: Fearless Concurrency

Ch17: Object Oriented Programming Features in Rust

-
- trait
-

Ch18: Patterns and Matching

Ch19: Advanced Features

Ch20: Building a Multithreaded WebServer

Jekyll

Transform your plain text into static websites.\

Jekyll is a **static site generator** that runs on the Ruby programming language.

- Simple
 - No more databases, comment moderation, or pesky updates to install—just your content.
 -
 - [How Jekyll works](#)
- Static
 - [Markdown](#), [Liquid](#), HTML & CSS go in. Static sites come out ready for deployment.
 - [Markdown](#) [Liquid](#) HTML CSS
 - [Jekyll template guide](#)
- Blog-aware
 - Permalinks, categories, pages, posts, and custom layouts are all first-class citizens here.
 -
 - [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 specified timeout)

# 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 Nemoe](#)
- [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.



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.



Here are some of the main advantages of static site generators over dynamic sites:

- **Speed:** your website will perform much faster, as the server does not need to parse any content. It only needs to read plain HTML.
 - : HTML
- **Security:** your website will be much less vulnerable to attacks, since there is nothing that can be exploited server side.
 - : HTML
- **Simplicity:** there are no databases or programming languages to deal with. A simple knowledge of HTML and CSS is enough.
 - : HTML CSS
- **Flexibility:** you know exactly how your site works, as you made it from scratch.
 - : CMS

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.

CMS

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.

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 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](#),

Ruby)
Smalltalk Eiffel) Ada BASIC Java) Lisp [10]#citenote-about-10) [[3]]
([https://en.wikipedia.org/wiki/Ruby_\(programming_language\)#cite_note-confreaks-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	Ruby	Ruby	"gem"	gems
Chad Fowler	Jim Weirich	David Alan Black	Paul Brannan	Richard Kilmer 2004 [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 gem gems [3] RubyGems Ruby
gem RubyGems gem

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]

Bundler

Bundler provides a consistent environment for Ruby projects by tracking and installing the exact gems and versions that are needed.

Bundler gem Ruby

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 gem bundle install

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 Ruby Gems gem

Gemfile

- A **Gemfile** is a list of gems used by your site. Every Jekyll site has a Gemfile in the main folder.
- Gemfile gem Jekyll 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

[hyprland](#)

[Github-Hyprland](#)

yprland is a 100% independent, dynamic tiling Wayland compositor that doesn't sacrifice on its looks. Hyprland
Wayland 100%

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... Wayland IPC
QoL

Wayland

[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 X11

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

Wayland “ ” Wayland

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

- Xorg X11 Wayland

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 libwayland XML C

API Wayland Wayland

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

Wayland Linux evdev X11 Wayland
Wayland Web

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 X Wayland Wayland X Wayland

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](#)

```
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

fcitx

1. sudo gedit /etc/apt/sources.list

2.

```
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
```

163 debian

3.

sudo apt-get update

apt-get install fcitx-ui-classic && apt-get install fcitx-ui-light

- 5. :

- 6. fcitx

- 7.

```
sudo apt-get install fcitx-sunpinyin fcitx-googlepinyin fcitx-pinyin
```

fcitx-sunpinyin fcitx-googlepinyin fcitx-pinyin

```
sudo apt-get fcitx-table-wubi fcitx-table-wbpy
```

```
(      fcitx-table-wubi  fcitx-table-wbpy)  
      sudo apt-get fcitx-table-cangjie  
(      )  
• 8.      : fcitx-table*  
      sudo apt-get fcitx-table*  
• 9.  
      sudo apt-get install fcitx-frontend-gtk2 fcitx-frontend-gtk3 fcitx-frontend-qt4
```

(fcitx-frontend-gtk2 fcitx-frontend-gtk3 fcitx-frontend-qt4)

• 10. Ctrl+
--

CC 4.0 BY-SA

<https://blog.csdn.net/ieeso/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

x86-64 GNU/Linux

Debian12 bookworm Release Notes

5.1.5. Fcithx versions no longer co-installable 5.1.5. Fcithx

The packages fcithx and fcithx5 provide version 4 and version 5 of the popular Fcithx 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 Fcithx is to be kept if they had co-installed fcithx and fcithx5 previously.

fcithx	fcithx5	Fcithx	4	5
fcithx	fcithx5		fcithx	

Before the upgrade, users are strongly encouraged to purge all related packages for the unwanted Fcithx version (*fcithx-* for Fcithx 4, and *fcithx5-* for Fcithx 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.

Fcithx	Fcithx 4	fcithx-	Fcithx 5	fcithx5-	im-
config					

[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.asc
```


Useful Links Collection

- [S](#)
 - [CS](#)
 - [linux.do](#)
- [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](#)
 - [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:](#)
 - [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/>

IBM	Python	AI	PDF	DOCX	PPTX	HTML	Markdown
	Markdown	JSON	OCR	PDF			
RAG		AI					

[best-of-ml-python](#)

Python	900	30
	OCR	

[bananas: cross-platform screen sharing](#)

bananas <https://getbananas.net>

<https://github.com/likeflyme/cloudbak>

<https://www.cloudbak.org/>

<https://www.cloudbak.org/use/create-session.html>

<https://github.com/idoottop/MagicMirror/blob/main/docs/cn/readme.md>

: 4ro2 <https://del-wang.lanzout.com/b01qdt5nba>

<https://github.com/Ritr/publicTools>

Traymond <https://github.com/tabris17/traymond>

<https://github.com/ggchivalrous/yiyin>

<https://github.com/yitong2333/Bionic-Reading/blob/main/README-CN.md>

Bochs

Bochs IA-32

Bochs	C++	IA-32(x86)PC	x86 CPU	I/O	BIOS
Bochs		x86 CPU 386 x86-64 AMD			
Bochs		Linux DOS Microsoft Windows	Bochs	Kevin Lawton	
Bochs		bochs " "	x86 PC	x86	
			Unix/X11		Win'95
Bochs	Unix/X11	Win'95	PC		
Bochs	emulator --	virtualization software	X86 ARM MIPS		
CPU					
Bochs		VirtualBox VMWare			
Bochs		/			
•					
•					
•					

Compression

Types of Compression

zip gz tar.gz tar.bz2 tar.xz

1. Zip (.zip)

-
- DEFLATE
-
-
- -
 -
 -
- Unix zip unzip

2. Gzip (.gz)

-
- DEFLATE
- tar /
-
- -
 -
 -
- gzip gunzip

3. Gzip Tar (.tar.gz .tgz)

- tar gzip
- gzip
- Unix
-
- -
 -
 -
- -czvf -xzvf tar

4. Bzip2 Tar (.tar.bz2)

- tar bzip2
- Burrows-Wheeler
-
-
- -
 -
 -
- -cjvf -xjvf tar

5. XZ Tar (.tar.xz)

- tar xz
- LZMA2
-
-
- -
 -

-   

1.
 - Zip DEFLATE
 - Gzip DEFLATE
 - Bzip2 Burrows-Wheeler
 - XZ LZMA2
2.
 - Zip
 - Gzip
 - Bzip2
 - XZ
3.
 - Zip
 - Gzip
 - Bzip2
 - XZ gzip bzip2
4.
 - Zip
 - Gzip tar
 - Bzip2 tar
 - XZ tar
5.
 - Zip
 - Gzip/Tar.gz
 - Bzip2/Tar.bz2
 - XZ/Tar.xz

- Zip
- Gzip tar
- Tar.gz Unix
- Tar.bz2 gzip
- Tar.xz

Ch12

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 && tar -xf -'
```

Explanation:

1. `tar` on the local machine:

- `--exclude='dir_father/dir_child1'` : Skips the `dir_child1` directory during the `tar` process.
- `-cf - dir_father` : Creates a tar archive of `dir_father` but writes it to standard output (`-`).

2. `ssh`:

- Connects to the remote host.

3. `tar` on the remote machine:

- `cd /destination/path` : Ensures the extracted files are placed in the correct destination.
- `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

- [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 # flatpak

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.

```
“           ”
“           ”
”
• 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

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.

```
,afl-fuzz      :      ,      ,      ,  
- ,
```

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](#).

afl.rs	Rust	AFLplusplus	AFLplusplus	AFL
--------	------	-------------	-------------	-----

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

recommand: 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) is the default bootloader for most Linux distributions. It is responsible for loading the kernel and passing control to it. It can be customized to load different operating systems, different kernel versions, and even different boot configurations.

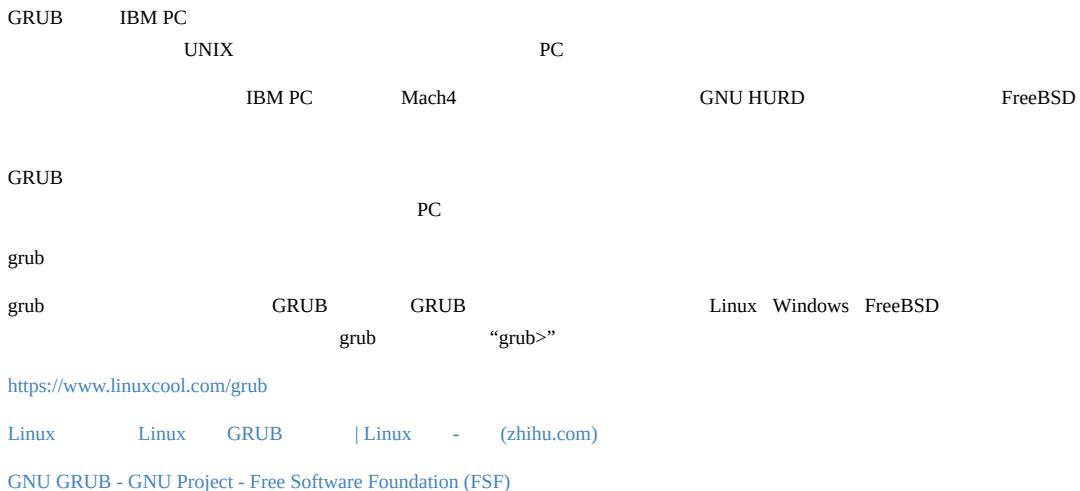
To customize GRUB, you can edit the `/etc/default/grub` file. This file contains a list of configuration options for GRUB. You can add or remove options as needed.

To make changes permanent, you can update the GRUB configuration using the `update-grub` command. This will generate a new `/boot/grub/grub.cfg` file that will be used by GRUB on the next boot.

To see the current GRUB configuration, you can run the `grub-editenv` command. This will open an interactive shell where you can modify the GRUB environment variables.

To add a new operating system to GRUB, you can create a new entry in the `/etc/grub.d` directory. This directory contains a series of shell scripts that are executed by GRUB in order to generate the GRUB configuration. Each script is responsible for adding a new entry to the GRUB menu.

GRand Unified Bootloader



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](#) [Gitbook](#)

GitBook Node.js

GitBook

[gitbook.com](#)

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

#       & build
gitbook install
gitbook build # generate static files under `_book` directory

# start server: localhost:4000
gitbook serve
```

GitBook

- README
- SUMMARY ,
- LANGS
- GLOSSARY

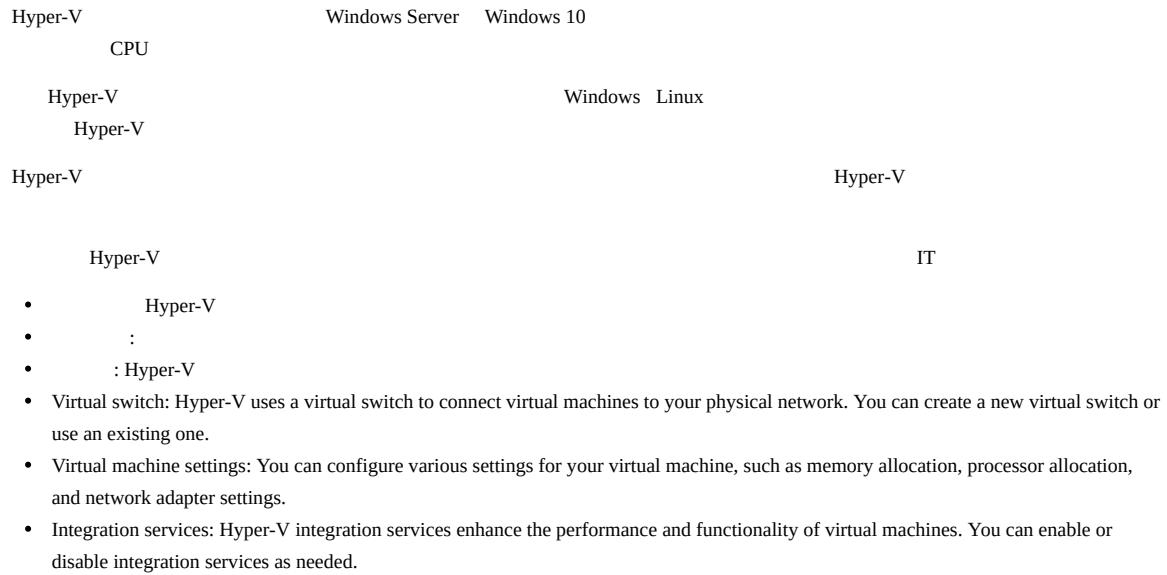
README SUMMARY

Gitbook pdf

```
gitbook pdf <gitbook-folder-location> <pdf-location>.pdf
```

Hyper-V

- [Windows 10 Hyper-V | Microsoft Learn](#)
- [| Microsoft Learn](#)



MartinDavis

Turing, His Machine and Computability Jurg Kohlas RegisterMachinesareTuringMachines ColinB.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

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="" }
```

```
#
Remove-Item * -Include *.json -Recurse
#
# This example deletes all of the files that have names that include a dot (.)
Remove-Item * -Include *.* -Exclude *.md -Recurse
#
#           *bin*      bin
Remove-Item * -Recurse -Include *bin*
```

- \$PSVersionTable.PSVersion
- Get-Host
- Invoke-WebRequest / iwr
- get-command
- get-process
- Set-Alias xxxx0 xxxx1
- cls
- Get-Command -Name xxname

Regular Expressions

- [wikipedia](#)
- [runoob](#)

```
[^a-z]      //  
[^\\^/\\^]    //  (\)(/)(^)  
[^\"\\']     //      ("')  
  
a.*b        // a          b
```

SD eMMC NVMe

SD

- 1.
- 2.
3. SD miniSD microSD
4. SDHC SDXC UHS-I UHS-II
5. SD PCIe SATA
6. GB GB
7. eMMC NVMe

eMMC

- 1.
- 2.
- 3.
4. SD SSD NVMe SSD 200-400 MB/s
5. SD
6. 16 GB 128 GB
- 7.

NVMe

1. SSD
- 2.
3. M.2 U.2 PCIe
4. SD eMMC 3000 MB/s
5. PCIe
6. 256 GB TB
- 7.

- 1.
2. **SD**
3. **eMMC**
4. **NVMe**
- 5.
6. **SD**
7. **eMMC**
8. **NVMe**
- 9.
10. **SD** SD
11. **eMMC** SD
12. **NVMe** PCIe
- 13.
14. **SD**
15. **eMMC**
16. **NVMe**

17.

18. **SD** eMMC NVMe19. **eMMC**20. **NVMe**

SD

eMMC

NVMe

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 Recovery Android Recovery
TWRP Android

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:

$$\text{Lag 1: } \{120, 110, 130, 140, 150, 160, 170, 180, 190, 200, 210\}$$

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:
$$X_t = \phi_{k1}X_{t-1} + \phi_{k2}X_{t-2} + \dots + \phi_{kk}X_{t-k} + \epsilon_t$$

- 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 (ϕ_{22}):** The coefficient (ϕ_{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.

Typst

Typst

show rules

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.

show					
name	name	ArtosFlow	show	logo	
logo	name				

VScode

Shortcuts

- `Ctrl+p`
- `Ctrl+shift+p`
- `Ctrl+w`
- `Ctrl+k+w`
- `Ctrl+f`
- `Ctrl+g`
- `Ctrl+Alt+f` Find in Explorer
- `Ctrl+R` Run Recent Command

Others

`vscode` `Ctrl+shift+p`

VScode server

`server commit_id`

```
# ~/.vscode-server/bin
#     server
#     :${commit_id}      Commit ID
https://update.code.visualstudio.com/commit:\${commit\_id}/server-linux-x64/stable
# vscode-server-linux-x64.tar.gz
#     ~/.vscode-server/bin/${commit_id}/
#
tar -zxvf 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

-
-
-
-
-
-

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:

- Right-click on the Start button and select **Device Manager**.

2. Locate Network Adapters:

- Find **Network adapters** and expand the section. Right-click on your Wi-Fi adapter and select **Properties**.

3. Change MAC Address:

- Go to the **Advanced** tab.
- In the **Property** box, select **Network Address** or **Locally Administered Address**.
- 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:

- Disable and re-enable your network adapter for the changes to take effect.

For Linux

1. Open Terminal.

2. Identify Your Network Interface:

- Run the command:

```
ip link
```

- Note the name of your network interface (e.g., `wlan0`).

3. Bring Down the Interface:

- Replace `wlan0` with your interface name:

```
sudo ip link set wlan0 down
```

4. Change the MAC Address:

- Run:

```
sudo ip link set wlan0 address XX:XX:XX:XX:XX:XX
```

- Replace `XX:XX:XX:XX:XX:XX` with your desired MAC address.

5. Bring Up the Interface:

- 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]
...
[802-11-wireless]
...
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)) $((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` 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.

Debian 12 Wi-Fi

1. SSID	Wi-Fi	SSID
2. BSSID	AP	SSID
3. MAC	MAC	MAC
4.		
◦	MAC	MAC

Attention Is Turing Complete

Not yet complete.

[Attention Is Turing Complete](#)

TLDR:

Attention 1 Encoder 3 Decoder Transformer

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.

Transformer Transformer

Introduction

(x, \dots, x) , $y \in Q$ $Y = (y, \dots, y)$, $d > 0$, seq-to-seq N $x \in Q$ $X =$
 seq-to-seq $N, N(X, s, r) Y = (y, y, \dots, y)$, s , $r \geq 0$, , seq-to-seq , ,

1 seq-to-seq $A = (\Sigma, f, N, F)$, Σ , $f : \Sigma \rightarrow Q$, N seq-to-seq , $s \in Q$, $F \subseteq Q$
 A $w \in \Sigma$, $r \in N$, $N(f(w), s, r) = (y, \dots, y)$ $y \in F$

A $(L(A)) A$

• $f : \Sigma \rightarrow Q$ Σ input :one-hot
 • , F ; f , $F \in F$ f ()

input embedding stopping , embeddings stopping
 criterions

, () $M = (Q, \Sigma, \delta, q, F)$, :
 seq-to-seq N L , N , N

2
 L (), seq-to-seq N

1. The Transformer architecture

, Transformer (Vaswani et al., 2017), Transformer
 , ,
 Transformer $score : Q \times Q \rightarrow Q$ $\rho : Q \rightarrow Q$, $d n > 0$ $q \in Q$, $K =$
 $(k, \dots, k) V = (v, \dots, v) Q$ $Att(q, K, V)$ q-attention over (K, V) $Q \in a$,
 $(s_1, \dots, s_n) = \rho(score(q, k_1), score(q, k_2), \dots, score(q, k_n))$

$$\mathbf{a} = s_1 \mathbf{v}_1 + s_2 \mathbf{v}_2 + \cdots + s_n \mathbf{v}_n$$

, q , K , V ,
 : x = (x, ..., x) ∈ Q, f Q Q, p(x) i p(x),
 j=1 f(x) f(x) , , f(x) e softmax , ,
 , (q, k) ,
 (Bahdanau et al., 2014) q, k , (Vaswani et al., 2017)
 : σ(g(·)) , g , σ (1) S
 , softmax r x, x x , hardmax , x x , f(x) = 1 , f(x) = 0 ,
 attention hardmax(x) = , hardmax(x) = 0 hardmax , hard

$$(Xu \text{ et al., 2015; Elsayed et al., 2019}), \quad , \quad , \\ 5 \quad , \quad F: Q \rightarrow Q \quad X = (x, x, \dots, x), \quad x \in Q, \quad F(X) \quad (F(x), \dots, F(x))$$

Transformer

$$\text{Transformer} \quad \text{Enc}(\theta), \quad \theta \quad , \quad Q \quad X = (x, \dots, x) \quad , \quad \text{Enc}(X; \theta) = (z, \dots, z) Q$$

$$X \quad , \quad \theta \quad , \quad Q(\cdot) \quad K(\cdot) \quad V(\cdot) \quad O(\cdot), \quad Q \quad Q \quad ,$$

$$a = \text{Att}(Q(x), K(X), V(X)) + x \quad (4) \quad z = O(a) + a \quad (5)$$

$$\text{Transformer} \quad (\quad) \quad , \quad K(\cdot) \quad V(\cdot) \quad , \quad L$$

$$\text{Transformer} \quad (1 \leq i \leq L-1 \quad X=X):$$

$$X = \text{Enc}(X; \theta), K = K(X), V = V(X). \quad (6)$$

$$V = V(X) \quad (K, V) = T\text{Enc}(X) \quad (K, V) \leftarrow L \quad X$$

$$Z = (z, \dots, z) \quad , \quad Y = (y, \dots, y), \quad 1 \leq i \leq k \quad , \quad Z = (z, \dots, z) \quad , \quad Y = Q(\cdot) K(\cdot) V$$

$$p = \text{Att}(Q(y), K(Y), V(Y)) + y \quad (7) \quad a = \text{Att}(p, K, V) + p \quad (8) \quad z = O(a) + a \quad (9)$$

$$, \quad (\mathbf{K}(Y), V(Y)) \quad \quad () \quad \quad i \quad \quad Y \quad , \quad \quad pto \quad \quad (K, V) \quad \quad Y \quad (K, V) \\ \text{Dec}((K, V), Y; \theta) \quad \text{Transformer} \quad \quad \quad , \quad \quad F: Q \rightarrow Q \quad \quad \quad , \quad \quad Q \\ \in \quad z \quad , L \quad \text{Transformer}$$

$$Y = \text{Dec}((K, V), Y; \theta), z = F(y) \quad (1 \leq i \leq L - 1 \quad Y = Y). \quad (10)$$

$z = TDec((K, V), Y)$ z L Y (K, V)

$$\text{Transformer} \rightarrow Q, \quad \text{Transformer} F \rightarrow Q, \quad \Gamma \vdash g(\Gamma), \quad \Gamma \vdash g : \Gamma$$

Transformer

$$\text{Transformer } X \quad y \quad r \in \mathbb{N} \quad Y = (y_1, \dots, y_r).$$

$$y = TDec(TEnc(X), (y, y, \dots, y)), \quad 0 \leq t \leq r - 1 \quad (11)$$

$$\boldsymbol{y}_{t+1} = \text{TDec}(\text{TEnc}(\boldsymbol{X}), (\boldsymbol{y}_0, \boldsymbol{y}_1, \dots, \boldsymbol{y}_t)), \quad \text{for } 0 < t < r - 1.$$

$$Y = (y, y, \dots, y) = \text{Trans}(X, y, r)$$

3.1

$$\mathbf{a}_i = \text{Att}(Q(\mathbf{x}_i), K(\mathbf{X}), V(\mathbf{X})) + \mathbf{x}_i$$

$$\mathbf{z}_i = O(\mathbf{a}_i) + \mathbf{a}_i$$

4. Transformer

6

Transformer , , $n \in N$ pos n $n - 1/n$

1/n, Transformer

, : Transformers , ;

, L L TM

6 , , M = Q, Σ , δ , q, F , , # \in

Σ

M Importing :

M q , #

Q qused

0 ,M q,

,qit , #

F

, transformer Tranthat M; ,

,L M = L Trans , , ,

Transwe w = ss...s. sas one-hot , K, V ,

K= k, ..., k V= v, ..., v , v k i , , ,

7

Transwe M w = ss...s. , $i \geq 0$:

q: i M s: i M v: i M m: i M

v = [q1, s1, x1, q2, s2, x2, x3, x4, x5, s3, x6, s4, x7 x8, x9, x10, x11],

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.

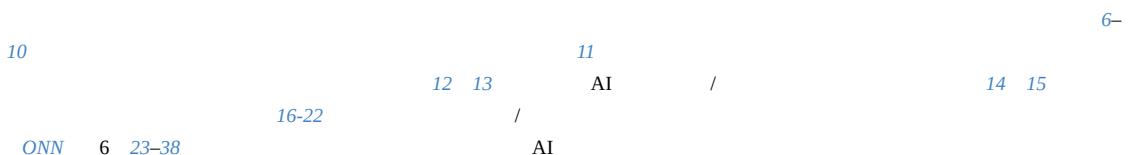
10	72.76%	AlexNet	72.64%	2K	CIFAR-
----	--------	---------	--------	----	--------

INTRODUCTION

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.

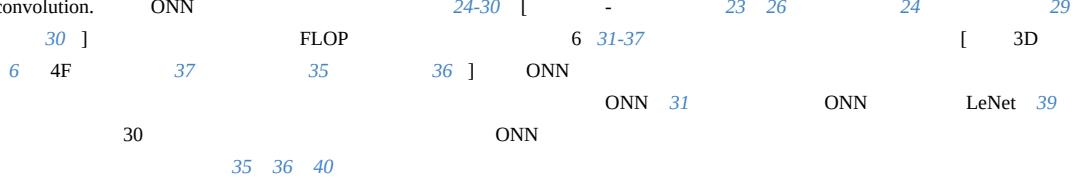


Optical computing has been proposed as a potential avenue to alleviate several inherent limitations of digital electronics, e.g., compute speed, heat dissipation, and power, and could potentially boost computational throughput, processing speed, and energy efficiency by orders of magnitude (6–10). Such 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/photonics 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.

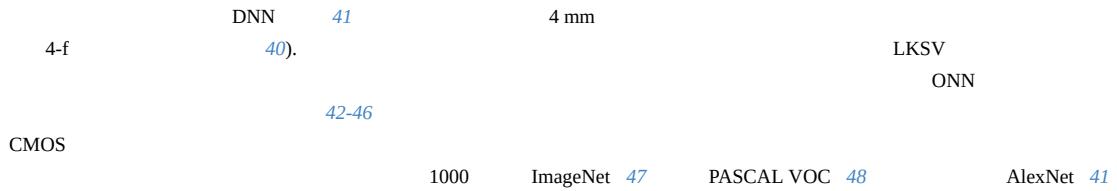


Existing ONNs can be broadly classified into two categories based on either integrated photonics (24–30) [e.g., Mach-Zehnder interferometers (23, 26), phase change materials (24), microring resonators (29), multimode fibers (30)] for physically realizing multiply-adds floating point operations (FLOPs), 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.



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 (LKSV) 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.



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 LKSV convolution instead of uniform convolutions increasing the parameter space by an order of magnitude.

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-fold orders of magnitude, thus bringing onns into deep learning era. [<99% FLOPs <1% FLOPs]

RESULTS

LKSV parameterization LKSV

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.

390 nm 525 nm S1
PSF

CRTP: Compile-Time-Reflection-Programming

[wiki](#)

```
// The Curiously Recurring Template Pattern (CRTP)
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

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.

CRTP

```

// 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().

shapePtr->clone()

ArgParse in Python

unexpected parser.add_argument("--train", type=bool)

```
argparse    parser.add_argument("--train", type=bool)
Python      bool()                         type=bool
                                         True   False
```

- bool("True") True bool("False") True Python True
- type=bool

```
                           action='store_true'    action='store_false'
```

1 **action='store_true'** **action='store_false'**

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("--train", action='store_true', help="")
args = parser.parse_args()

print(f"  : {args.train}")
```

```
python script.py --train
```

```
: True
```

```
--train
```

```
python script.py
```

```
: False
```

2

```
True  False
```

```

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("          'True'      'False'")

parser = argparse.ArgumentParser()
parser.add_argument("--train", type=str_to_bool, help="          ")

args = parser.parse_args()

print(f"  : {args.train}")

```

```
python script.py --train True
```

```
  : True
```

```
python script.py --train False
```

```
  : False
```

```
python script.py --train maybe
```

```

usage: script.py [-h] [--train TRAIN]
script.py: error: argument --train:      'True'      'False'

```

- action='store_true' action='store_false'
- True False
- type=bool

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 C++ C++
 Dlib

```
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

PyQt5

[doc](#)

- **QtCore** – GUI
- **QtGui** –
- **QtMultimedia** –
- **QtNetwork** –
- **QtOpenGL** – OpenGL
- **QtScript** – Qt
- **QtSql** – SQL
- **QtSvg** – SVG
- **QtWebKit** – HTML
- **QtXml** – XML
- **QtWidgets** – UI
- **QtDesigner** – Qt Designer

QWidget **QObject** **QPaintDevice**

QDialog **QFrame** **QWidget**

1	QLabel
2	QLineEdit
3	QTextEdit
4	QPushButton
5	QRadioButton
6	QCheckBox
7	QSpinBox /
8	QScrollBar
9	QSlider
10	QComboBox
11	QMenuBar QMenu
12	QStatusBar QMainWindow
13	QToolBar QMainWindow
14	QListView ListMode IconMode
15	QPixmap QLabel QPushButton
16	QDialog

GUI

QMainWindow



QDialog

Modal

Modeless

PyQt API

Dialog

InputDialog FileDialog FontDialog

QMessageBox

GUI

SDI

MDI

QMdiArea

QMdiArea
subWindowQMainWondow
MDI

QMdiSubWindow

QWidget

PyQt5 -

MIME

QDrag

QMimeData

MIME

QMimeData

MIME

PyQt5

QtSql

SQL

QSqlDatabase

Connection

SQL

QGraphicsScene

- **Create a QGraphicsScene :** This serves as the container for the graphical elements.
 - QGraphicsScene
- **Load the Image with QPixmap :** Use `QPixmap` to load the image.
 - Load the Image with QPixmap QPixmap
- **Add the Image to the Scene :** Create a `QGraphicsPixmapItem` from the `QPixmap` and add it to the scene.
 - QGraphicsPixmapItem QPixmap
- **Display the Scene in a QGraphicsView :** Use `QGraphicsView` to display the contents of the scene.
 - QGraphicsView QGraphicsView

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

QAbstractButton

Qt Style Sheets

- [stylesheet-examples](#)
- [Customizing Qt Widgets Using Style Sheets](#)
- [Qt Style Sheets Reference](#)
- blog: [QSS](#) [PyQt5](#)
- [qt-material](#)

Qt-Meterial:

```
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

- - - 1938
 - - - 1946
 - - - 1948
 - -
 -
 -
-

References

- jyywiki.cn/Letter.md
- -

A Brief History of Humankind

NEW YORK TIMES BESTSELLER

"I would recommend this book to anyone interested in
a fun, engaging look at early human history. . . .

You'll have a hard time putting it down."

—BILL GATES

Yuval Noah Harari



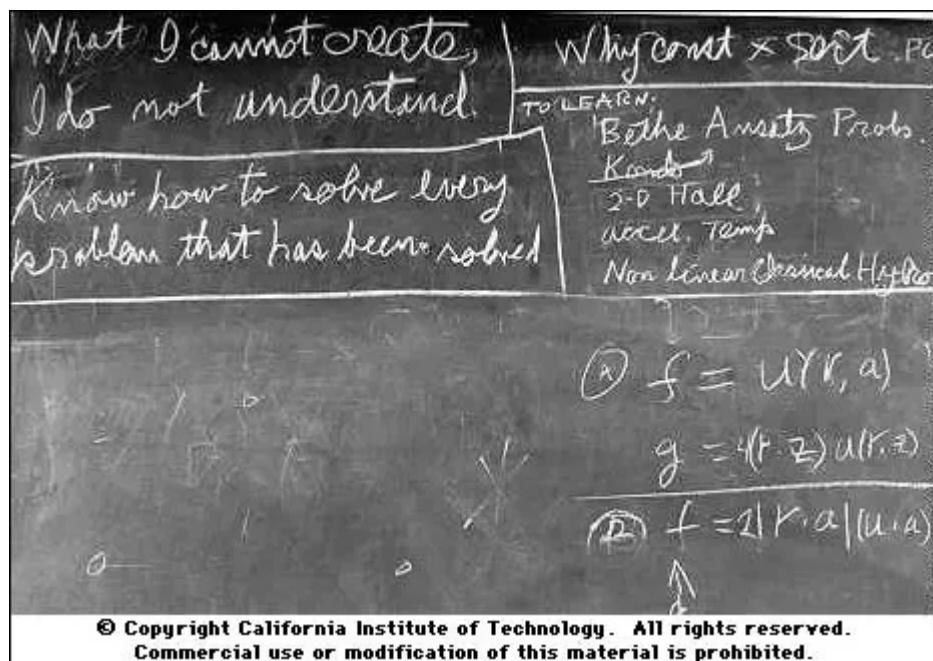
Sapiens

A Brief
History of
Humankind

•		“	”			
•		“	”	“corporation”		“ ”
•	“corpus”	“ ”				
•				DNA		
•					All	Men
•	created		“All men are created equal.”		“are”	
•		equal				

100

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.

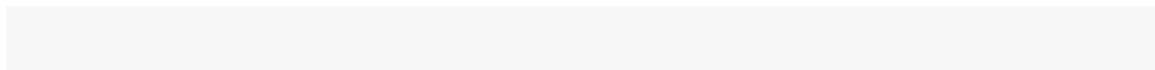
The spinning plate story describes how Feynman felt that curiosity about simple things, and working them out for himself, helped him retain an attitude of play towards his professional work that got him out of a slump.

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.

[1]

foundations of mathematics



-
-
-

“ ” “ ” “ ” “ ”

“ ” “ ”

“ ”



Research
Research
“ ”
Galois

study

Research

Seminary

.....

1948 4 20

2005

—

+

1.

2.

—

CP1:

CP2: CP3:

CP4:

CP5:

CP6: 4

CP7:

CP8: CP9: CP10:

1.

2.

1.

.....

|

~

1+1=2

or " " or

—

" "

-

" "

“ ”

“ ” _____

1.

2.



•

— O

1.1



1.1

...

1.2



—>

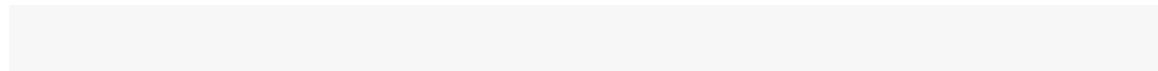
" "



1.

2.

1.2



1.3

1.3

N

N

1. **O(N³)**

• p q p q

```

#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);
    }
    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. $O(N^2)$

-
-

```

#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. O(NlogN)

- S
- 1. [p1, q1]
- 1. [p2, q2]
- 1. [p1, q2]

1 2 3 [p1, q2]

2 [p1,q1] [p2,q2]

1 [p1,q1]
2 [p2,q2]
3 [p1,q2]

mid = (0+8)/2 = 4

:

[-2, 1, -3, 4, -1] [4]	[3,3] p1=q1=3 lmsum = 4
[2, 1, -5, 4] [4]	[8,8] p2=q2=8 rmsum = 4
[p1,q2]	[3,8] 5
[4, -1, 2, 1]	[3, 6] 6
[p1,q2]	

```

#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){
    //
    if(p>=q){
        mp=mq=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;
    //

    maxSubSum(arr, p, mid, lmp, lmq, lmsum);
    maxSubSum(arr, mid+1, q, rmp, rmq, rmsum);

    /*
    midsum      midsum      mp   mq      lmsum rmsum
    */
    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 ;  
}
```

9
-2 1 -3 4 -1 2 1 -5 4

maxSum:5.00
mp:3
mq:8
4.00 -1.00 2.00 1.00 -5.00 4.00

maxSum:6.00
mp:3
mq:6
4.00 -1.00 2.00 1.00

(mid) mid

```

#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){
    //
    if(p>=q){
        mp=mq=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, ttmplsum=0;
    //

    maxSubSum(arr, p, mid, lmp, lmq, lmsum);
    maxSubSum(arr, mid+1, q, rmp, rmq, rmsum);

    /*
    midsum      midsum      mp   mq      lmsum rmsum
    */
    int midp=mid, midq=mid+1;

    for(int j=mid; j>=p; j--){
        ttmplsum+=arr[j];
        if(tmplsum>midlsum){
            midlsum=tmplsum;
            midp=j;
        }
    }
}

```

```
        }
        for(int i=mid+1; i<=q; i++){
            tmprsum+=arr[i];
            if(tmprsum>midrsum){
                midrsum=tmprsum;
                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.

O(N)

```

#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){ // 0
            //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

                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. dp O(N)

```
[p, q]           “ ”
dp[i]      i
/
dp[i-1] > 0   dp[i] = dp[i-1]+arr[i]
dp[i-1]<=0   dp[i]= arr[i]
dp[0]=arr[0];
```

```

#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>//
elemType maxSubSum(vector<elemType> arr, int r, int l){
    int size = arr.size();
    // i dp i
    // dp[i-1] > 0 dp[i] = dp[i-1]+arr[i]
    // dp[i-1]<=0 dp[i]= arr[i]
    // maxdp " i "
    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. O(N)

hisSum maxSum maxSum thisSum	thisSum thisSum thisSum<0
---------------------------------------	-------------------------------------

```

#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;
}

/*
    thisSum
    hisSum maxSum
    maxSum      thisSum<0
    thisSum
*/
//      <0
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;//  thisSum<0      starttmp
        }
        else if(thisSum>maxSum){
            maxSum=thisSum;
            start=starttmp;//  thisSum>maxSum      start
            end=i;
        }
    }
    if(start==size){start=end=0;}
    return maxSum;
}

```

1.3

- Q1 1.3
- Q2

- Q3

Q2

“ ”

```

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])
            //

```

step1: / map list -> q
 $S(p,q) == \text{target}$ $S(1, p-1) = S(1, q) - S(p,q) = S(1, q) - \text{target}$

step2: q $S(1,q)$ map key= $S(1, q) - \text{target}$ $p=\text{map}[key]+1$

map

1.4

1.4

- Q1 GS
- 25
- Q2

N	$[l_1, r_1], [l_2, r_2], \dots, [l_N, r_N]$	$x_i \in [l_i, r_i]$	$i=1, 2, \dots, N$
$[1, 4] \quad [2, 3] \quad [1.5, 2.5]$	$2.1 \quad 2.1 \quad 2.2$	$[2, 3] \quad [1, 4] \quad [1.5, 2.5]$	
$2.1 \quad 2.2 \quad 2.4$	$[1, 2] \quad [2.7, 3.5] \quad [1.5, 2.5]$		
-

• - - O
• - -
• - -

- **1946**

" "

" " "

" "

" " "

1893 1986

1920

1931 1936

1946 4 26 1946 5 4

1 5

- 1938 4 28

!

?

?

“ ”

“ ” ?

“ ” ?

?

“ ”

“ ”

?

“ ” “ ”

“ ”

?

“ ”

?

“ ”

“ ” “ ” “ ” “ ” “ ” “ ”

“ ”

“ ”

“

” “ ”

?

?

?

“ ”

“ ”

?

?

“ ”

“ ”

“ ” “ ” “ ”
“ ” “ ”

?

?

?
;

“ ”
“ ”

“ ”

?

? “ ”
“ ”

? “ ”
“ ”
“ ”
?

“ ”

;

“ ” !

“ ” “ ”

“ ” “ ”

!

“

”

“

”

“

”

“

‘‘，’’

‘‘，’’，’’，’’

……”

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.
- 8.

9.



1835 8