计算机体系结构 作业1

在本次作业中，你将尝试使用**gem5**进行一个**基于Tomasulo算法实现的乱序CPU（O3 CPU**）仿真实验：在本地机器上编译部署gem5，将一份cpp代码编译为RISC-V二进制文件，然后使用gem5 O3 CPU模拟运行该二进制文件，调整O3 CPU几个关键参数并观察分析对系统性能的影响。

背景介绍

1. **gem5**

gem5模拟器是一个用于计算机系统体系结构研究的模块化仿真平台。gem5最初是为学术界的计算机体系结构研究而设计的，但如今它的应用已日益广泛，被学术界和工业界用于研究和计算机系统设计，并应用于教学领域。

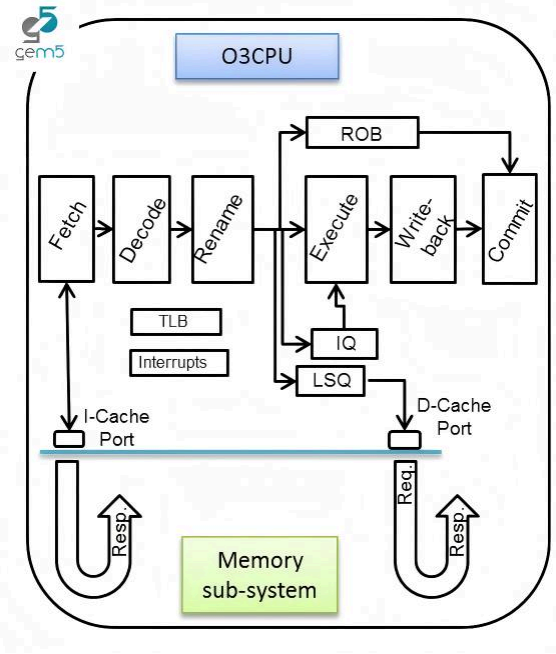
gem5官网：https://www.gem5.org/

2. **O3 CPU**

在理论课中，我们学习了Tomasulo算法，它揭示了现代高性能处理器通过乱序执行（Out-of-Order，OoO）来挖掘指令级并行性的核心思想。本次作业我们尝试使用的是gem5中实现的O3 CPU，参考现实中的Alpha 21264处理器架构（不完全一致），其设计思想同样基于Tomasulo算法——通过寄存器重命名来消除数据伪依赖，并利用一个大的指令窗口来动态地寻找并执行那些操作数已就绪的指令，从而最大限度地挖掘程序中的指令级并行性，不过在实现上与我们学习的经典Tomasulo存在一些差异，例如重命名方式、流水线阶段划分、部件设置等。

注意：这里的 O3 指 Out-of-Order，而不是编译器优化等级。

O3CPU中的关键部件



1. **ROB (Reorder Buffer, 重排序缓冲):** 与经典Tomasulo算法中的ROB功能基本一致，用于实现指令的按序提交和精确异常。
2. **IQ (Instruction Queue, 指令队列):** O3 CPU中的指令在Rename阶段结束后，除了在ROB中创建相应的条目，也会在IQ中创建相应条目。IQ的作用类似经典Tomasulo算法中的保留站：在Tomasulo中每个功能部件拥有一个专属的保留站，而在O3 CPU中所有功能部件共享一个“保留站”，即为IQ（除了load/store指令有专门的Load/Store Queue，LSQ）。
3. **物理寄存器堆**: 这是实现O3 CPU中寄存器重命名的前提。O3 CPU内部拥有远多于ISA逻辑寄存器的物理寄存器，在流水线中的重命名阶段，O3 CPU为指令中的逻辑寄存器分配物理寄存器。如果物理寄存器堆耗尽，O3 CPU流水线中的重命名阶段将阻塞。

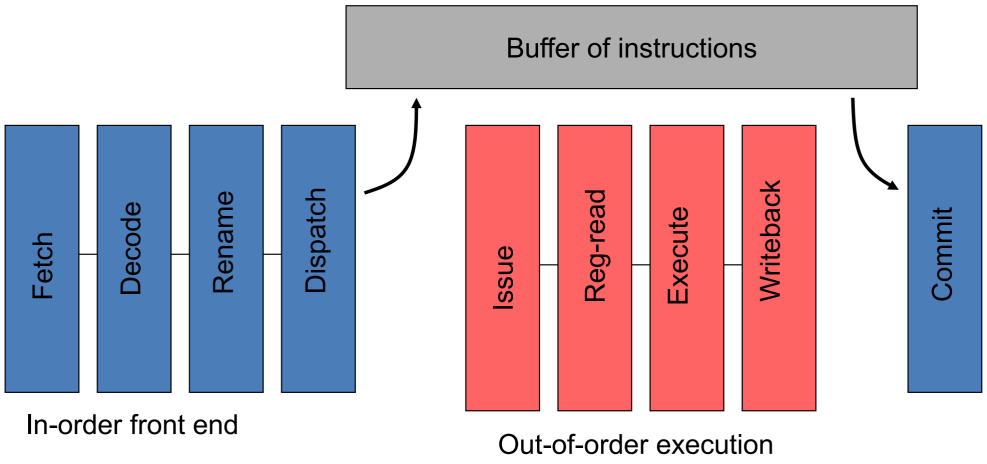
O3 CPU的流水线阶段划分

O3 CPU实现了一个五阶段流水线：

1. **取指 (Fetch)**: 取指阶段在每个周期获取指令，并进行分支预测**（下图中的Fetch阶段）**。
2. **译码 (Decode)**: 译码阶段在每个周期解码指令。并提前处理无条件分支指令**（下图中的Decode阶段）**。
3. **重命名 (Rename)**: 重命名阶段使用**一个带有空闲列表的物理寄存器堆**来为指令中的逻辑寄存器分配物理寄存器（分配算法见resources目录中的DynamicScheduling介绍pdf），从而消除伪依赖。完成重命名后，为指令分配ROB条目和IQ条目，指令进入乱序阶段**（下图中的Rename+Dispatch阶段）**。与经典Tomasulo算法不同的是，O3 CPU使用一个单独的流水线阶段进行寄存器的重命名。
4. **发射/执行/写回 (Issue/Execute/Writeback - IEW)**: 在该阶段中，指令在所有操作数就绪后进行发射，操作数读取，执行以及写回操作**（下图中红色部分）**。该阶段需要的时钟数根据指令操作数准备情况、操作类型等不同，可能从1个到数百个不等。

5. **提交 (Commit)**: 提交阶段在每个周期按序提交已完成的指令，并处理任何遇到的故障和分支预测错误等**（下图中的Commit阶段）**。

除了上述介绍，如果想要对O3 CPU有更彻底的了解，可以查看下面给出的参考资料，里面有对O3 CPU更细致的讲解。



参考资料

- gem5 O3 CPU介绍：https://www.gem5.org/documentation/general_docs/cpu_models/O3CPU
- resources目录下的DynamicScheduling介绍pdf

文件说明

- O3CPU.py: O3 CPU的配置文件，用于gem5仿真配置
- daxpy.cpp: 要仿真运行的程序源代码
- resources: 该目录下存放有帮助理解gem5/O3 CPU的学习资料

实验步骤

1. 部署gem5模拟器

gem5的部署需要Linux环境，你可以在以下3种环境中进行部署（教程在文档底部附录中给出）：

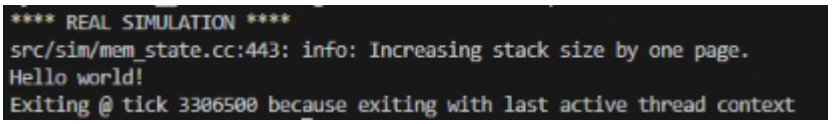
- 虚拟机（VMWare、VirtualBox）
- Docker容器
- WSL

gem5官方部署教程参考：gem5: Building gem5

在gem5根目录运行下列命令，通过启动一个hello world测试程序来验证gem5是否安装成功：

```
build/RISCV/gem5.opt configs/deprecated/example/se.py -c tests/test-
progs/hello/bin/riscv/linux/hello
```

显示如下的“Hello world！”就代表安装成功：



2. 编译提供的daxpy.cpp为RISC-V二进制文件

使用如下命令安装交叉编译工具：

```
sudo apt update
sudo apt install g++-riscv64-linux-gnu
```

编译命令：

```
riscv64-linux-gnu-g++ --static -O2 -o [binary path] [source file path]
# e.g. riscv64-linux-gnu-g++ --static -O2 -o daxpy daxpy.cpp
```

3. 使用提供的O3CPU.py仿真配置文件，模拟RISC-V架构的O3 CPU运行上一步得到的二进制文件

本次仿真实验需要你获取O3 CPU在3个参数不同取值组合条件下的仿真结果，参数说明和取值范围要求如下：

参数名称	说明	取值范围
num-phys-int-regs	物理整数寄存器数目	64, 256, 1024
num-iq-entries	IQ条目数	4, 16, 64, 256
num-rob-entries	ROB条目数	4, 16, 64, 256

仿真运行命令参考如下，你可以通过修改命令中`--num-phys-int-regs`、`--num-iq-entries`、`--num-rob-entries`的值来调整参数：

```
[gem5.opt path] --outdir=[directory to save results] [python config path] \
    -c [daxpy binary path] \
    --num-phys-int-regs=[num of physical integer registers] \
    --num-iq-entries=[num of IQ entries] \
    --num-rob-entries=[num of ROB entries]

# e.g.
gem5/build/RISCV/gem5.opt --outdir=m5out O3CPU.py \
  -c daxpy \
  --num-phys-int-regs=256 \
  --num-iq-entries=64 \
  --num-rob-entries=192
```

上述命令会将仿真结果保存至`--outdir`指定目录下的`stats.txt`中：

```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.006642                       # Number of seconds simulated (Second)
simTicks                                   6642416000                       # Number of ticks simulated (Tick)
finalTick                                  6642416000                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tic
simFreq                                 1000000000000                        # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                     30.90                       # Real time elapsed on the host (Second)
hostTickRate                               214942249                        # The number of ticks simulated per host second (ticks/s) ((Tick/Second))
hostMemory                                   2269528                        # Number of bytes of host memory used (Byte)
```

在上一步得到的`stats.txt`中查找CPU仿真的各项性能指标。建议关注的指标如下：

指标名称	说明
system.cpu.numCycles	CPU模拟运行的总时钟数
system.cpu.rename.ROBFullEvents	由于ROB已满导致的重命名阶段堵塞
system.cpu.rename.IQFullEvents	由于IQ已满导致的重命名阶段堵塞
system.cpu.rename.fullRegistersEvents	由于物理寄存器耗尽导致的重命名阶段堵塞

你至少需要完成以下内容：

- 按照实验步骤3中的要求遍历参数组合进行仿真实验，记录每次仿真得到的**system.cpu.numCycles**数据，以表格或可视化图表方式呈现在实验报告中。

  提示：手动执行所有实验会比较繁琐，可以尝试编写一个自动化脚本来辅助。

- 结合你仿真得到的数据，分别分析**IQ条目数**和**ROB条目数**这两个参数是如何影响CPU模拟运行的总时钟数的？例如，随着这两个参数增大，CPU运行的总时钟数增大还是减小？通过调整这两个参数来减少CPU运行总时钟数是否存在瓶颈？为什么？

  提示：可以参考理论课上学习的Tomasulo算法进行思考和分析。

以下为选做内容（加分）：

- （5分）分析物理整数寄存器数目是如何影响O3 CPU性能的？无限制扩大物理整数寄存器数目对O3 CPU的性能提升是否存在瓶颈？为什么？

  提示：阅读resources目录下的DynamicScheduling pdf中有关重命名的介绍以及daxpy.cpp代码会对本题的解答有所帮助。

- （5分）尝试修改O3CPU.py中的代码，调整O3 CPU的其他设置，分析其他因素对CPU仿真性能的影响。

4. 编写实验报告

实验报告模板不限。你的实验报告需要包含下列内容：

- 对实验过程的介绍，关键步骤（gem5部署成功验证、daxpy.cpp编译、仿真启动/结束等）给出截图，展示你执行的命令和执行结果。如果有编写额外的代码文件来完成任务，也请附上代码文件，在报告内简单介绍代码作用。
- 实验步骤3中要求完成的实验内容。

作业提交要求

1. 请提交pdf版本的实验报告，命名为“[姓名]-[学号]-lab1.pdf”，例如“张三-23000001-lab1.pdf”。如果有要提交的代码文件，请和实验报告一起打包为一个zip压缩包，压缩包命名为“[姓名]-[学号]-lab1.zip”。
2. 作业请提交至超算习堂。
3. **DDL：10.26 23:59**

附录

1 在虚拟机中部署gem5

提示：最终项目文件大小约为 6G，`-j 2`情况下编译过程所需内存约为 7G，请预留足够的空间~

1.1 编译环境准备

这里使用 VMware Workstation，软件安装过程参考 VMware 安装参考教程（用 VirtualBox 等虚拟机软件也可以）

可能会需要用到的镜像网站 Ubuntu 镜像网站，选择你需要的版本，安装完成后可以在命令行窗口通过 `lsb_release -a` 查看版本号。接下来的演示使用 22.04.5 版本进行。

接着拉取仓库代码：

```
sudo apt update
sudo apt install git
git clone https://github.com/gem5/gem5.git
```

1.2 编译项目

项目构建过程可参考官方文档 gem5 构建官方教程

Ubuntu 22.04 使用如下命令安装 gem5 编译和运行所需的系统依赖：

```
sudo apt install build-essential git m4 scons zlib1g zlib1g-dev \
    libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev \
    python3-dev libboost-all-dev pkg-config python3-tk clang-format-15
```

虚拟机分配的内存如果是 4G 建议编译 gem5 时用 `-j 1`；8G 则可用 `-j 2`（实际上爆内存也没事，可以重新用 `-j 1` 接着编译，不会重新开始）

```
cd gem5
# 你可能还需要安装项目的依赖
sudo apt install python3-pip
pip install -r requirements.txt

# 如果有 Cannot find 'pre-commit'. 问题
# export PATH=$HOME/.local/bin:$PATH

scons build/{ISA}/gem5.{variant} -j {cpus}
# e.g. scons build/RISCV/gem5.opt -j 2
# 建议 -j 值设为 1/2 或设置 更大内存，给虚拟机分配 8G 内存不够跑 -j 4
```

编译过程可能花费 2h 左右，请耐心等待。

到此 gem5 部署完成，接下来请自行编译.cpp文件和调整仿真配置文件以完成实验。

你可以还会需要：`sudo apt install gcc-riscv64-linux-gnu g++-riscv64-linux-gnu`

2 使用Docker容器部署gem5

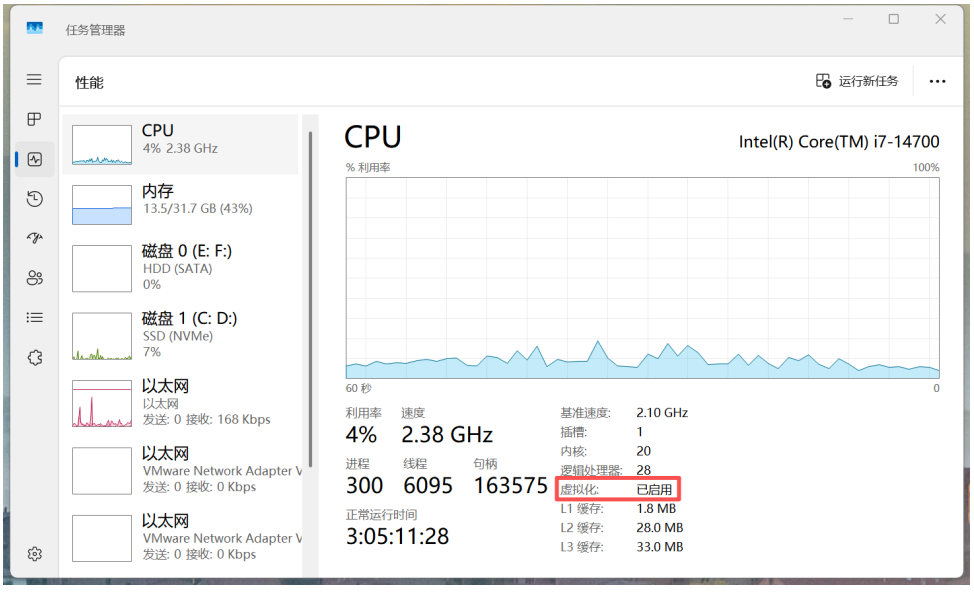
Docker安装参考教程

2.1 确认虚拟化已开启

Docker 依赖虚拟化技术，需要确保 BIOS 中已启用虚拟化

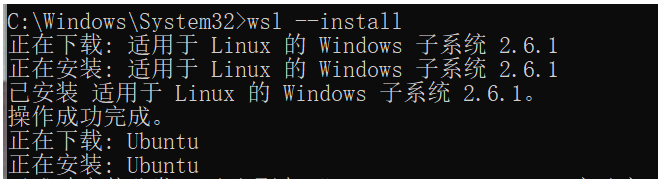
检查方法：

1. 按下 Ctrl + Shift + Esc 打开任务管理器
2. 切换到 性能 选项卡
3. 查看右下角 虚拟化 是否为 已启用



2.2 安装WSL

以管理员的身份启动命令提示符，输入：`wsl --install`



2.3 安装Docker

下载并安装Docker Desktop

2.4 下载Docker镜像

```
docker pull ghcr.io/gem5/ubuntu-24.04_all-dependencies:latest
```

参考链接：https://github.com/gem5/gem5/pkgs/container/ubuntu-24.04_all-dependencies

2.5 在VScode中安装Dev-Containers插件

2.6 启动Docker容器

```
docker run -it -v [lab1 directory]:/lab1 [image]
```

参数解析：

1.`-v [lab1 directory]:/lab1`

这是数据卷挂载 (volume mount)：

- 左边：`[lab1 directory]`->Windows主机上的目录
- 右边：`/lab1`->容器内部的目录

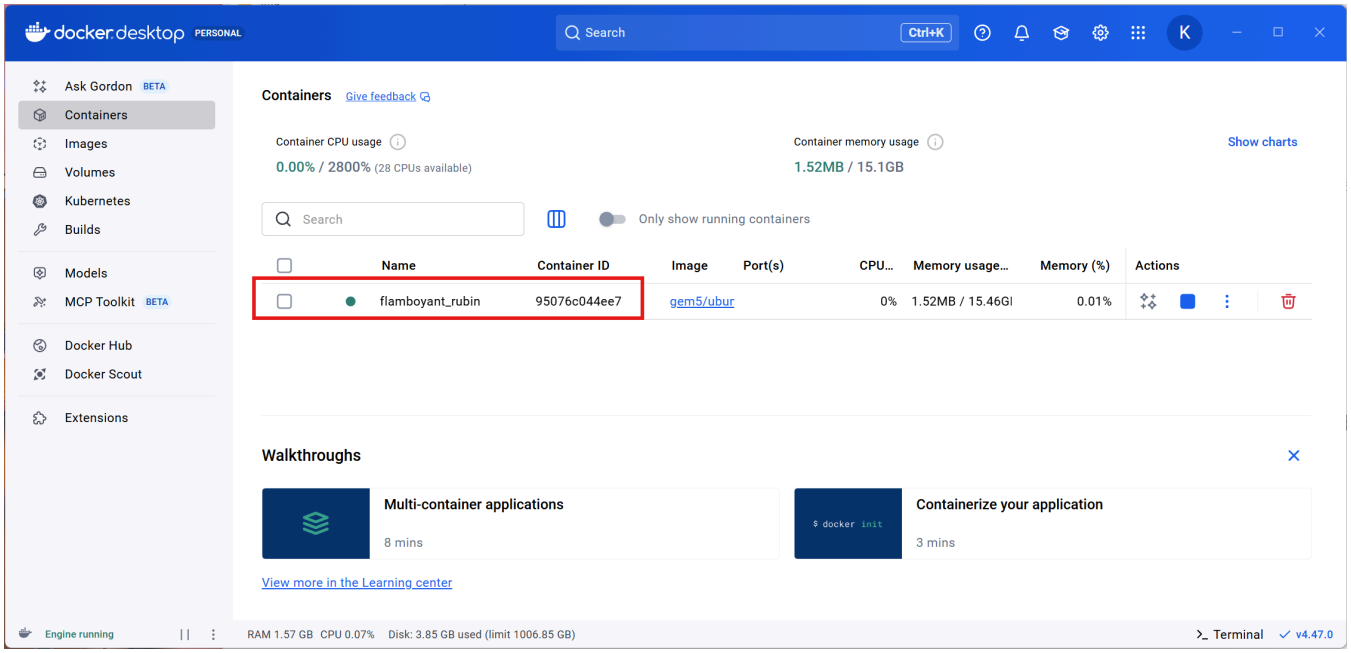
挂载后，你在容器 `/lab1` 里看到的文件，就是 Windows 目录 `[lab1 directory]` 的内容，修改也会实时同步。

2.[image]

指定要运行的 Docker 镜像

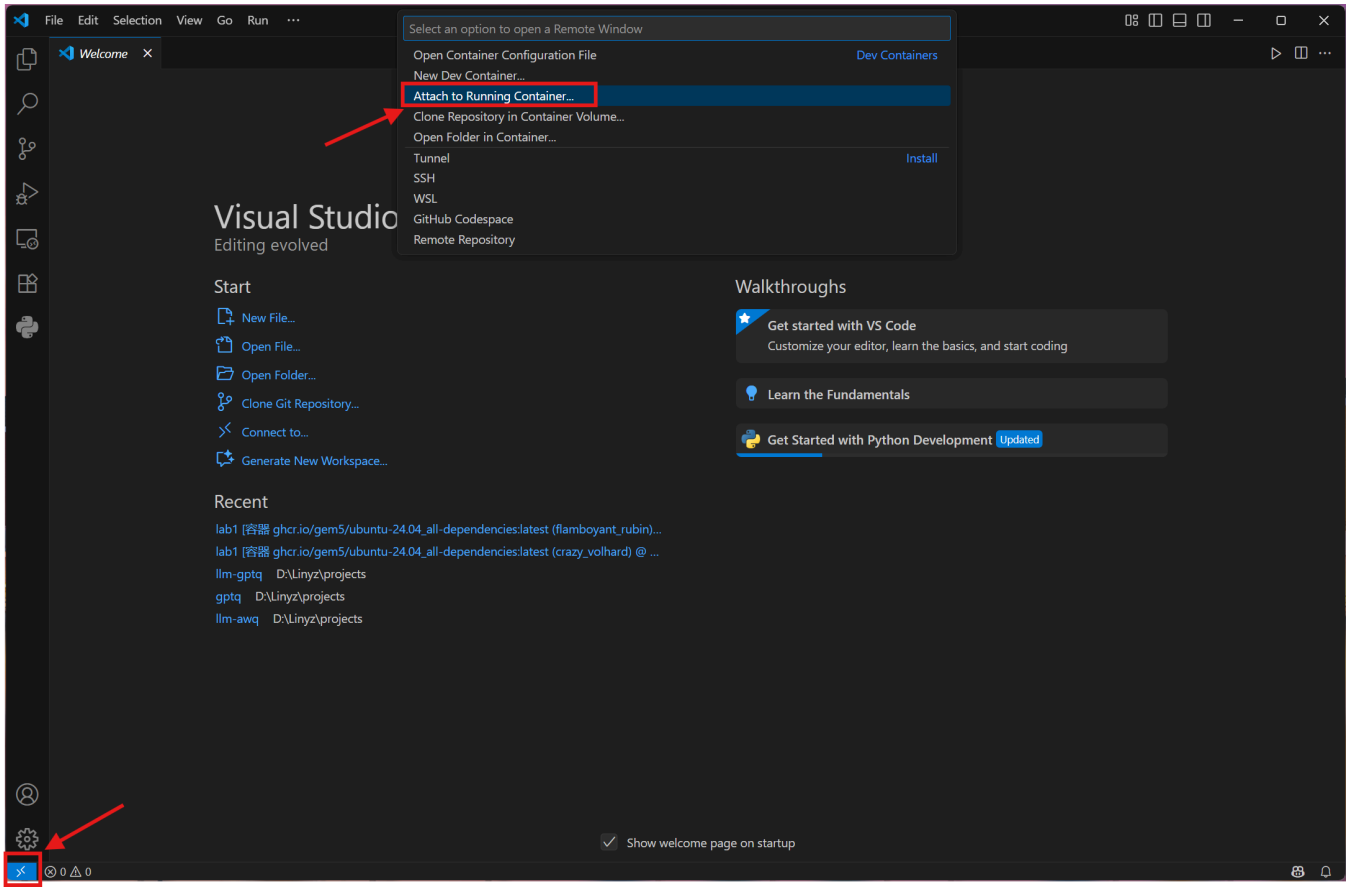
```
# 示例：
docker run -it -v D:\Linyz\projects\CSA\lab1:/lab1 ghcr.io/gem5/ubuntu-24.04_all-dependencies:latest
```

```
D:\Linyz\projects\CSA\lab1>docker run -it -v D:\Linyz\projects\CSA\lab1:/lab1
 ghcr.io/gem5/ubuntu-24.04_all-dependencies:latest
root@95076c044ee7:/#
```



2.7 在Dev Containers中连接容器

通过 Dev Containers 插件连接运行中的容器。



2.8 使用Scons构建gem5

克隆 gem5 仓库：

`git clone https://github.com/gem5/gem5`

注：在windows上clone的代码可能与linux平台存在不兼容，故不建议在windows本地 clone 代码后再挂载到容器中，更安全的做法是在容器里直接 clone 代码。

在 gem5 根目录下执行构建：

```
scons build/{ISA}/gem5.{variant} -j {cpus}

# 示例：
scons build/RISCV/gem5.opt -j 2
```

构建过程可能花费 2h 左右，请耐心等待

3 在WSL中部署gem5

参考教程：

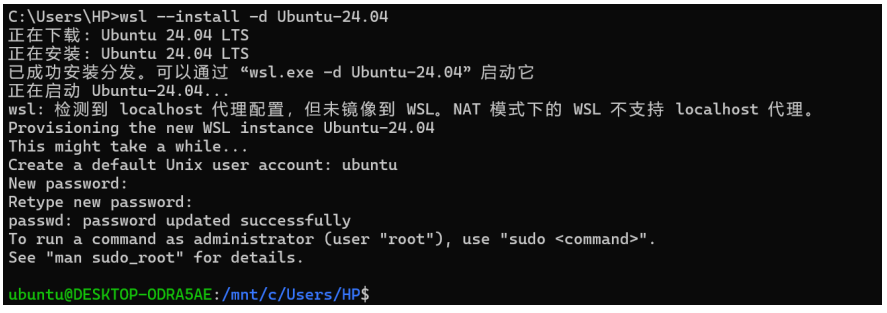
https://learn.microsoft.com/zh-cn/windows/wsl/install

https://blog.csdn.net/x777777x/article/details/141092913

https://learn.microsoft.com/zh-cn/windows/wsl/tutorials/wsl-vscode

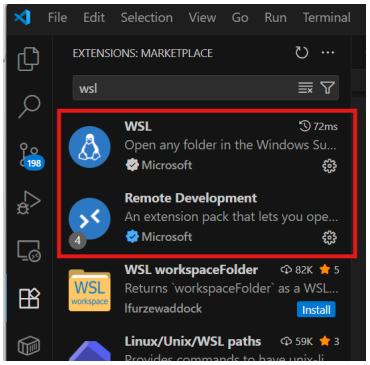
在windows终端中执行下列命令，然后根据提示完成WSL安装：

```
wsl --install -d Ubuntu-24.04
```

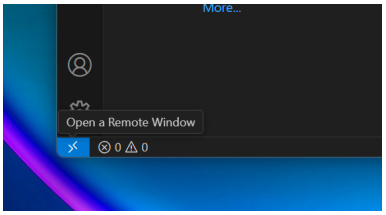


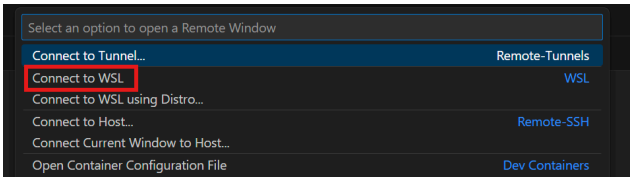
然后，可以通过VSCode远程连接至本地WSL中：

1. 安装VSCode：https://code.visualstudio.com/
2. 在VSCode中安装下列插件：

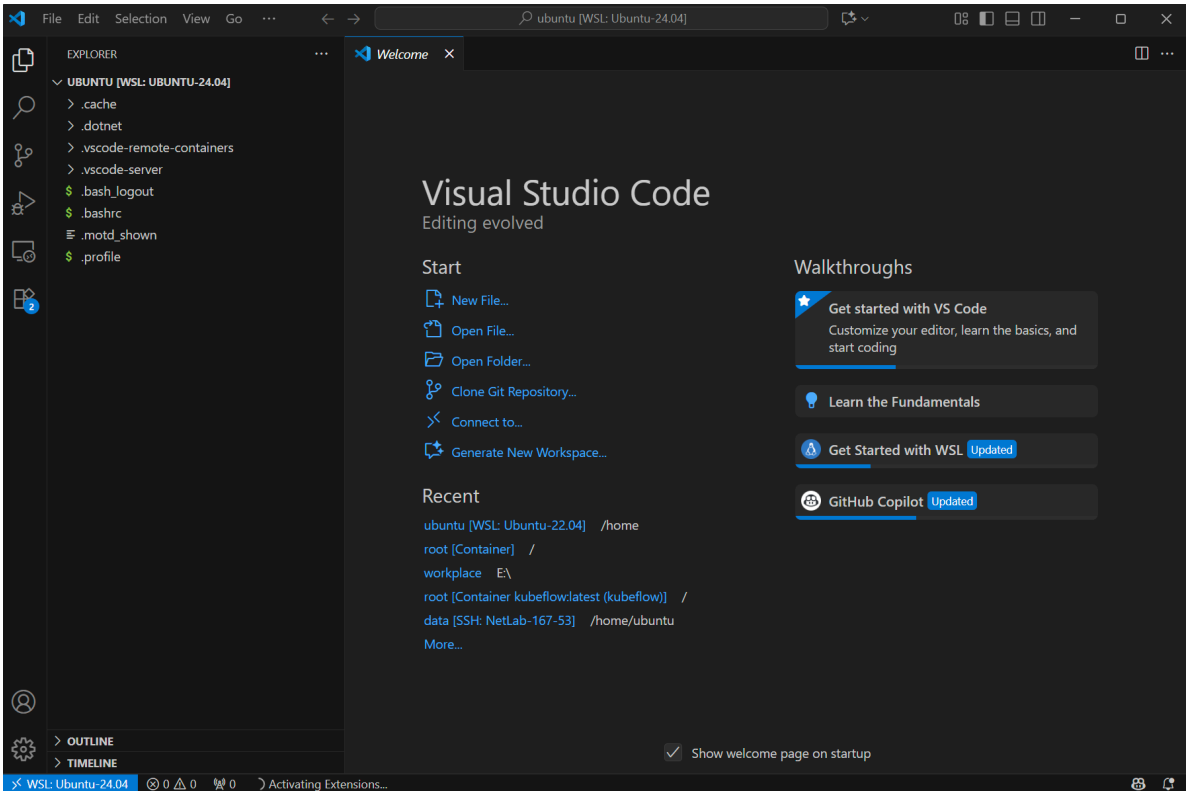


1. 点击VSCode左下角蓝色按钮，然后在弹出的栏目中选择“Connect to WSL”

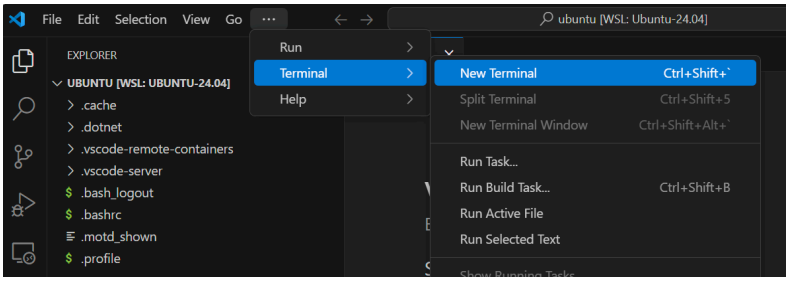


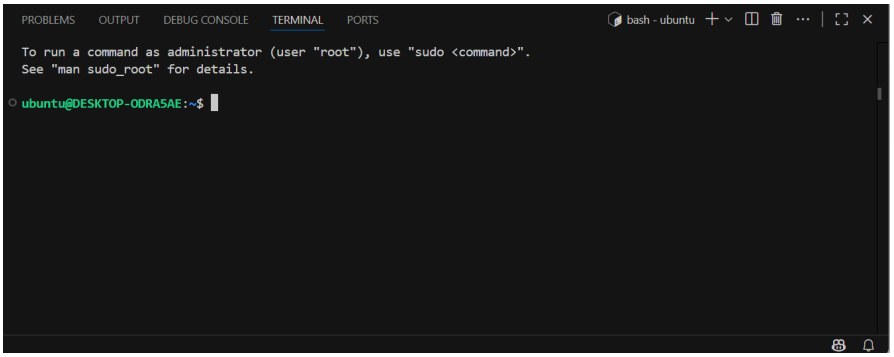


1. 连接至WSL后，你就可以像在本地一样通过VSCode浏览WSL中的各个文件和目录。



WSL中部署gem5与虚拟机/容器方法类似，都需要在命令行中进行操作，在VSCode中可以通过以下方式打开终端：





之后的部署方法就和虚拟机中部署gem5的方法一致。