计算机科学与技术学院
中国科学技术大学
计算机体系结构 2024 春

lab1: running and try to use gem5

助教张亦博

实验目的

- 熟悉 *gem5* 模拟器的运行环境
- 初步理解 *gem5* 进行仿真和模拟执行程序的过程
- 尝试简单的使用 *gem5*

1 *gem5* 项目介绍

以下简单的介绍 *gem5*, 更详细的资料请你参考拓展阅读中给出的内容或者互联网上的资料。

gem5 的发展经历了三个阶段：

1. 初始版本（2003 年）：*gem5* 的前身是两个独立的模拟器：*m5*（由密歇根大学开发）和 *GEMS*（由威斯康星大学麦迪逊分校开发）。这两个模拟器分别关注于不同的模拟需求，*m5* 侧重于处理器和内存系统的模拟，而 *GEMS* 侧重于缓存一致性和内存系统的模拟。
2. 合并至 *gem5*（2011 年）：为了整合资源和优势，*m5* 和 *GEMS* 的开发团队决定合并这两个模拟器，创建了 *gem5*。这一合并使 *gem5* 成为了一个功能更加全面、更加灵活的计算机体系结构模拟平台。
3. 持续发展（2011 年至今）：自从 *gem5* 诞生以来，它一直在不断地发展和完善。开发团队和研究社区持续贡献新的功能、模型和改进，使得 *gem5* 能够支持更广泛的体系结构研究，包括多核处理器、异构系统、能源效率等领域。

gem5 的用户很多，其中应该多为高校用户，但是也不乏开源组织和企业用户。高校用户大多使用 *gem5* 来验证对于体系结构改进的想法，许多论文的实验验证部分采取的就是使用 *gem5* 验证。开源社区的 *gem5* 用户最著名的应该是中科院计算所的香山处理器团队，其使用 *gem5* 来进行微架构的探索，减少芯片设计的开发周期。*gem5* 企业用户应当包括 *Google* 和 *AMD* 等（现在 *gem5* 中的一些代码都是他们贡献的）。

2 前置 0：环境搭建

gem5 只能工作在 linux 环境下编译，因此你需要一台 linux 环境的物理机或者虚拟机，并且这台机子最少有 8 G 内存并至少有 20 G 的空闲存储空间。（助教的操作系统是 LinuxMint，是基于 Ubuntu 20.04 的发行版，因此你们的操作系统可以选择 Ubuntu 20.04）。

根据 *gem5* 官网 提供的教程来安装所需要的包，对于遇到的问题，官网都有相关的解答，此外，项目的构建工具 scons 除了通过 apt 安装之外，也可以通过 pip 安装或者通过官网直接下载二进制安装，这样安装的好处是能够安装自己指定的 scons 版本。助教由于采用了 conda 管理 python 环境，因此通过 pip 安装了 4.4 版本的 scons，pip 安装 scons 的命令如下：

```
pip install scons==4.4
```

实测后续的所有操作在此版本下都能够编译通过，当然，使用从 apt 上安装的 scons 也不会有问题。

至此，你的环境应该已经搭建完成。

3 前置 1：构建 *gem5*

对于 scons 工具，你可以通过 `man scons` 命令调出手册来了解其使用，你也可以调用 `scons --help` 来获得帮助。*gem5* 支持出构建 3 种版本的可执行文件，它们的后缀分别为：

- **debug**：调试版本的二进制文件，包含了所有的调试符号信息，需要调试时构建这个版本。
- **opt**：开启编译优化后得到的二进制文件，仍然包含了调试符号信息，虽然也可以用来调试，但是实测使用这个版本调试非常的难受。
- **fast**：开启了全部优化后得到的二进制文件，删除了调试信息，删除了运行时断言，运行速度最快。

由于本次实验只是简单的使用 *gem5*，而不涉及 *gem5* 的调试，因此我们可以直接编译出 opt 版本：

```
scons build/X86/gem5.opt -j $(($(nproc)-2))
```

这样的编译方式只会空出 cpu 上的两个核，其他全部用于编译，可以加快编译速度。如果你想变更编译器，可以：

```
scons build/X86/gem5.opt -j $(($(nproc)-2)) -compiler="clang"
```

在构建的过程中可能会遇到如下的问题：

链接因内存不够而失败：调高虚拟机使用的内存，或者减小编译时候所用的核数，或者在编译的时候尝试增加选项 `--limit-ld-memory-usage` 。

链接时候找不到符号或者找不到库：根据报错下载相关的库，并将相关的库放到链接器的库搜索路径下。链接器的搜索路径可以通过 `ld --verbose`得到。我在 scons 的编译脚本中已经用注释指示了如何添加库。（请你开启文档搜索，搜索`add your lib here ca2024`）

编译时显示找不到可用的 python 版本：这个问题往往不是 python 导致的，$gem5$ 检查 python 版本的方式比较奇葩，因此遇到这个问题，你需要到 `build/{arch}/gem5.build/scons_config.log` 中查看日志，并根据日志的报错信息进行解决。

在构建完成之后，为了为后续的实验做准备，你可以更改目录下 `.clangd` 文件，将 `path-to-gem5/src` 加入到头文件的包含路径中，这样 clangd 插件就能够正常工作，进行 cpp 的代码跳转了。（前提是你的环境安装了 clangd 和相关插件的情况下）

4 实验 1：运行 $gem5$

第一步直接上手运行 $gem5$，最为直接的体验 $gem5$ 的功能。阅读 Getting started 的 Creating a simple configuration script 章节和 Adding cache to the configuration script 章节，编写配置文件并使用配置文件模拟运行样例程序。再阅读 Getting started 的后续章节，学习和了解 $gem5$ 输出的统计文件内容，如何使用 $gem5$ 自带的配置文件进行模拟。

你需要完成的（4 分）

- 跟随新手教程编写的配置文件 $simple.py, caches.py, two\_level.py$, 代码文件请对你觉得有助于理解的部分加上注释。
- 实验报告第一部分，其中需要有:
  1. 在虚拟机上成功编译
  2. 运行 simple.py 的结果命令行截图
  3. 运行 two_level.py 的结果命令行截图
  4. 流程简述

5 前置 2：$gem5$ 核心概念

5.1 从编程到运行机制

$gem5$ 是一个 cpp/python 混合编程的项目，使用 cpp 编写 $gem5$ 的底层执行引擎，使用 python 编写暴露给用户的接口，同时用户也通过 python 语言编写 $gem5$ 的仿真脚本。

cpp/python 混合编程

现在的很多项目都采取了 cpp/python 混合编程的方式，使用 cpp 去编写项目的内核运行时，使用 python 脚本来调用接口、使用项目。
cpp 是一种高性能的编程语言，适合处理计算密集型任务，python 则在编写简洁、易读的代码方面更有优势。将两者结合起来，可以兼顾性能和开发效率。很多成熟的 cpp 库（如 OpenCV 、TensorFlow 等）都提供了 python 接口，使得 python 开发者可以方便地利用这些强大的库。而 python 提供了多种机制来调用 cpp 代码，如使用 ctypes、PyBind11 等，这些工具使得在 python 和 cpp 之间的互操作变得更加容易。

由上面的描述可以看出在 $gem5$ 中 python 文件是分为两类的，一类是用户接口，另一类是用户仿真脚本：

- 用户接口：随机分布在各个源代码文件夹下，往往和 cc、hh 文件挨在一起，这类就是编程接口文件。阅读其中的代码可以知道 $gem5$ 中的一个组件，有怎么样的 api 能够供你调用。
- 用户仿真脚本：分布在 configs 文件夹下，这类脚本是交由 $gem5$ 去做仿真执行的，也就是`build/X86/gem5.opt xxx.py` 中 gem5.opt 后跟的参数，也就是你们在上一节中编写的仿真文件。

再谈有关 $gem5$ 的仿真执行，$gem5$ 中使用了**事件驱动的仿真机制**。在仿真的命令行命令发出之后，$gem5$ 根据用户传入的配置将各个计算机部件进行连接，连接完成之后，事件队列开始驱动执行，$gem5$ 不断的从事件队列头中取出事件，执行这个事件，利用事件去驱动计算机中的组件工作，记录这个事件产生的影响，将这个事件中产生的新的事件再插回到事件队列中，由此循环往复，直到事件队列为空为止，在此之后 $gem5$ 将记录的统计数据按照用户的要求写入相关位置，整个过程就结束了。以上就是整个 $gem5$ 的运行过程。

5.2 核心概念

SimObject：SimObject 是 $gem5$ 中最核心的概念，SimObject 在 $gem5$ 中代表了对真实世界计算机部件的抽象，典型的 SimObject 包括 cpu 、DRAM 等等，通过观察源码中这些部件的设计，可以发现它们都无一例外的继承自 SimObject。SimObject 能够向全局的事件队列中添加事件（Event），新产生的事件可以通过 SimObject 插入到事件队列中。

赋值连线：现代计算机系统中各个部件之间是会产生联系的，如 cpu 会和 cache 相连，从 cache 中获取数据；又如最后一级的 cache 会和内存相连，在 cache 缺失的时候能够从内存获取数据。通过各个部件的连接，各个计算机的部件得以组成计算机系统。在 $gem5$ 中，这个连线的操作就用等于号完成，如 `system.cpu.icache_port = system.membus.cpu_side_ports` 。$gem5$ 的运行时会在创建出各个组件的对象之后对各个组件进行连接。

Port：$gem5$ 中有许多计算机组件，Port 是组件交互的接口，Port 允许各个组件之间进行交互，相互发送和相互接收数据。组件之间的数据通信是多样的，可能进行的是内存请求与响应，可能进行的是中断的请求与响应，为了适应这些不同的通信类型，Port 封装了统一的通信接口，各个组件只需要将自己需要传递的信息进行包装放到 packet 中，就能实现 Port 的统一接口的通信。

Event：代表了 $gem5$ 中的事件。其定义中定义了纯虚函数 `process` 供使用者实现，在事件从队列中取出并被调度的时候，实际上就是在执行 Event 的 `process` 方法。

6 实验 2：尝试编写 SimObject

在这个部分的实验中，我们尝试对 $gem5$ 简单的进行使用，成为一名开发者去进行 $gem5$ 的开发。在这部分的实验中，我们会简单的体验 $gem5$ 组件中的参数传递，并且简单的尝试 $gem5$ 中的事件机制。本实验这一部分的所有代码均放在 `path-to-gem5/src/lab1` 下，你所要做的就是完成 python 脚本到 SimObject 中的参数传递，尝试使用 gem5 中的日志 debug 机制，再在你写的 SimObject 中完成一次事件的调度，修改构建脚本来编译你写的 SimObject，最后使用编译出的 $gem5$ 来执行你的脚本并观察结果。

让我们开始吧！

6.1 参数传递

$gem5$ 中用 python 配置参数，但是运行时的 cpp 代码却能够使用 python 脚本中传入的参数，参数的传递并不可能凭空完成，这之间必然有一个参数传递的过程。这个过程

是笼罩在初学者头上的第一个大问号（至少对于助教来说是这样的）。

在此简单的讲一下 *gem5* 的参数传递机制，首先 *gem5* 项目中有一个名为 `PyBind11` 的工具作为 python/c 接口，即通过这个工具能够实现 python/c 之间的通信。同时 *gem5* 在构建过程中也会**自动生成**对应于每个 SimObject 的 param 类（前提是你在类定义中调用了`PARAMS`这个宏，你可以去头文件中的类定义下看看），这个 param 类放在 `path-to-gem5/build/src/param` 下，这也是为什么在我们写的头文件中，包含了一个 `"params/FirstObject.hh"` 这样一个诡异的还不存在的文件，编译的时候还不会报错，因为编译的时候这个文件被生成出来了。

到此，参数传递的过程变得简单了，python 脚本中的参数通过 `PyBind11` 传递到了对应的 param 类中，在 cpp 中的 SimObject 初始化的时候，直接通过 param 对象就能获取参数了。

你需要完成的（1 分）

请你补充完整 python 中和 cpp 文件中 SimObject 的参数传递过程，你需要传递的参数是一个字符串，值为你的学号，这个字符串后续会被打印。

6.2 日志跟踪

gem5 已经封装了一系列的跟踪函数，用来在整个执行流程中打印信息，它并不希望用户去调用 `cout` 去打印输出到控制台。*gem5* 一列的日志跟踪函数就封装在 `path-to-gem5/src/base/trace.hh` 下，其中包括了 `DPRINTF` 等宏可供调用。注意，这部分使用的 debug flag 我已经定义好，请你到构建脚本中查看并使用。

在使用这些日志跟踪的宏之后，只需在编译的时候指定开启跟踪，就能够输出一些日志信息。

你需要完成的（1 分）

请你利用 *gem5* 中的日志跟踪宏，打印从 python 中获取到的你的学号，你需要实现的是 `log` 函数。

6.3 事件调度

在前文中我已经阐述了 *gem5* 中的事件驱动仿真机制，为了能够更加形象的理解，我在给出一张图来描述这个过程：

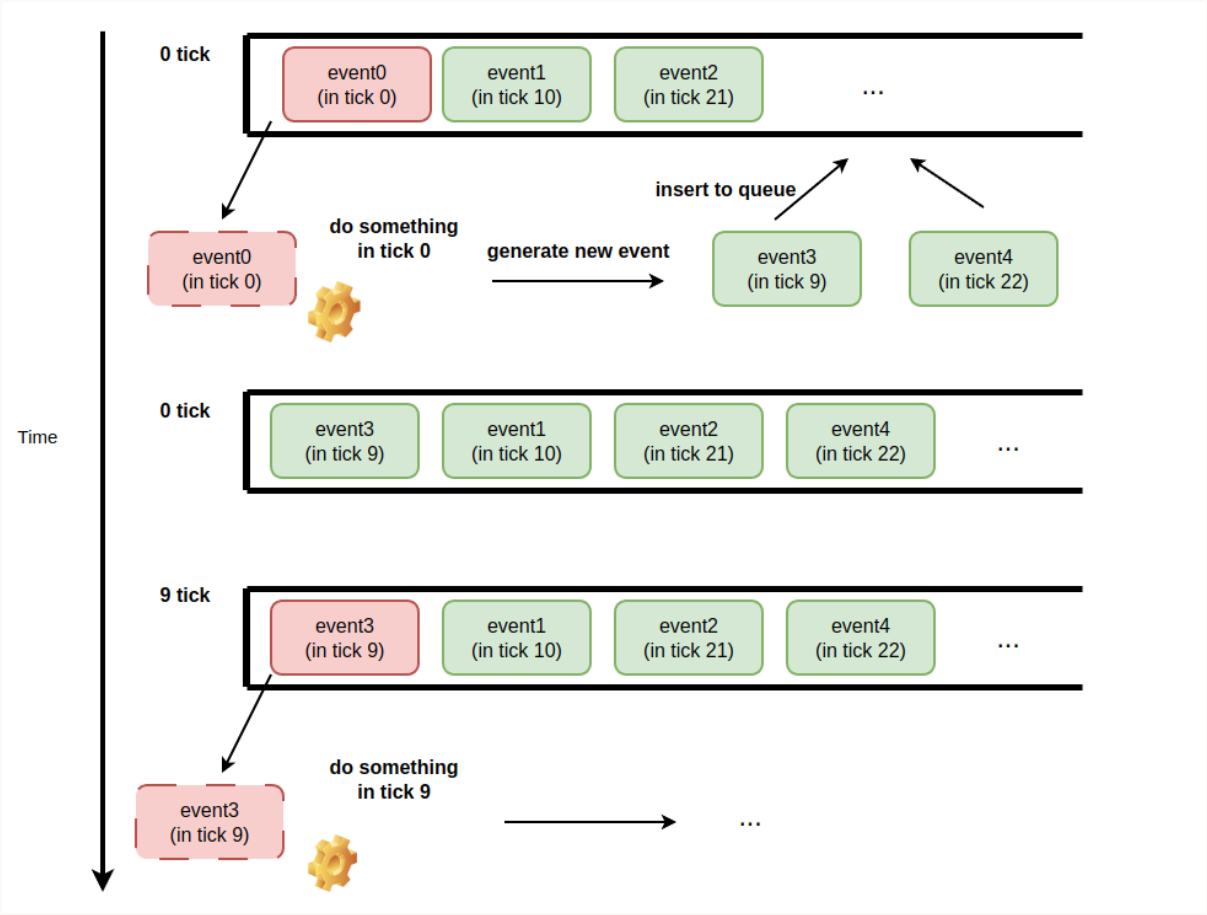

图 1: 事件调度队列

那事件又是如何插入到事件队列中的呢，$gem5$ 定义了一个 `EventManager`来提供将事件插入事件队列的功能，继承了这个类的子类就能够具有事件调度的能力。通过观察doxygen 文档你可以发现 SimObject 类继承了 `EventManager` 这个类，而我们的类继承了 SimObject 类，因此，我们定义的类是能够有调度事件的能力的。

在知道有调度事件的能力之后，调用什么方法进行调度又成为了问题，显然，查看 `EventManager` 的文档就能解决这个问题了。下简单对几个常见的调度方法进行总结:

方法名称	作用
schedule	将一个完全没调度过的事件加入到调度队列中
reschedule	对一个已经在队列中的事件重新设置调度时间
deschedule	销毁一个事件对象，释放其内存

现在你知道了队列的原理，也知道如何调用函数插入队列，所以你只需要在合适的时间调用函数就行了。在本实验中我们要在仿真开始的时候向队列中插入事件，在 $gem5$ 的 SimObject 中有一个 `startup` 虚函数，在仿真开始的时候这个函数会被调用，子类能够通过重写这个虚函数来控制仿真开始的时刻对象的行为，自然在这个时候，是能够往对列中插入事件的。

你需要完成的（1 分）

请你利用 $gem5$ 中的事件调度机制，调度 firstEvent 事件。实现 `process` 和 `startup` 函数。请你将事件插入队列，并将事件的调度事件设置为当前时间的 10 tick 之后。至于当前时间如何获得？请你利用搜索引擎或者查阅资料或者阅读源码，找到获取当前事件的方法并不难。

6.4 修改构建

在你的代码编写完成之后，你需要重新添加构建脚本，让构建系统找到你编写的代码，并将其添加到 $gem5$ 的构建中去。

你需要完成的（1 分）

请你补充完整目录下的构建脚本，将你的代码加入到构建系统中构建。注意在此刻你可能还不知道这些调用的函数是干什么用的，暂时不需要了解这么多，你只需要依葫芦画瓢，看看其他构建脚本是怎么写的，然后再来完成自己的构建脚本即可。

注意，你现在在构建的时候需要将你写的代码添加到构建系统中，因此，现在你的构建命令是：

```
scons build/X86/gem5.opt --part2 -j $(($(nproc)-2))
```

这样才能保证后续的过程不会出错。

6.5 运行脚本

最后一步，启动！

唯一需要注意的是，启动的时候需要开启 debug flag，才能看到打印结果，因此，现在你的启动命令变为：

```
build/X86/gem5.opt --debug-flags=lab1debug src/lab1/test.py
```

你需要完成的（1 分）

请你补充完整目录下 python 脚本，在脚本中创建出你的 SimObject，并对脚本进行仿真。然后，你就可以在控制台看到结果了。

7 其他

你需要完成的（1 分）

- 请你进行头脑风暴，基于本次实验，请你随意构想 *gem5* 这类模拟器在体系结构探索中起到的作用，不限字数。不用长篇大论或者反复修改，想到什么说什么就好。
- 请你对文档质量进行反馈，这对后续文档的质量提升**非常重要**。请你对本次的实验文档给出自己的评价（不错、还行、垃圾），并提一提对文档的改进意见，不限字数。同样是想到什么说什么。
- 请你对本学期的实验难度给出自己的期待。本学期（2024 春）助教想尝试对实验内容进行一些改进，因此想提前调查各位同学对实验内容的难度阈值。你可以给出自己对实验难度的阈值给助教参考，比如：
  - 我可以接受稍微提高难度的课程实验，希望实验内容更加贴近前沿的体系结构研究，通过实验我希望我能成为一名 *gem5* 的使用者 …
  - 我仍希望课程的实验难度同往届相同，虽然我感兴趣于体系结构，但是我并不感兴趣于模拟器，我只是希望别人问起的时候，我可以告诉别人我曾经简单的接触过 *gem5*…
  - 我乐意于接受实验难度提高而出现的挑战，但是这个过程中我希望助教能够引导我去完成 …
  - …

  当然，这也只是先进行调查而已，助教也不能保证在这一学期紧凑的时间之内能够按照课程的执行快速设计出合适难度的实验，但是梦想和期待总归还是要有的。但是话说回来，无关课程实验的难度高低，想要成为一名体系结构研究者，最不应该怕的就是困难，因为你会发现你研究过程中要学的东西就没有简单的，你学的任何一个东西几乎都需要你对操作系统、编译器、指令集架构有所了解，所以用自己对体系结构的热情去应对挑战吧 …

一些拓展的资料

- 根目录下的 ppt 中给出了对 *gem5* 的简单介绍。
- *gem5* 官网中也有许多教程，但是不认为这些教程很细致。
- doxygen 文档很重要，能够查阅类继承关系和 api。
- 我做的 *gem5* 编译过程的笔记，仅供参考。

注意事项

- 整个过程中善用搜索引擎和代码编辑器的搜索功能，当你想知道为什么的时候，尝试用搜索引擎搜索。当你想知道代码怎么写的时候，尝试使用代码编辑器的搜索，看看别人是怎么写的。
- 助教自认为文档写的很详细了，但是助教也是一名体系结构初学者，难免出错，如有问题请邮箱联系助教，联系时请务必表达清楚自己的环境情况。
- 完成实验文档并按照命名规范在规定时间内提交，实验文档务必最后导出为 pdf 格式提交。
- 可以讨论，不许抄袭，判定有抄袭行为的直接零分。