

体系结构实验课——深入浅出地了解仿真器

by 郭天宇

我是谁——简单自我介绍

个人主页 <https://gty111.github.io/>


中山大学 计算机学院 研究生一年级

研究方向：体系结构——GPU相关

对实验部分有问题或有其他问题，可以联系我 [guoty9\[at\]mail2.sysu.edu.cn](mailto:guoty9[at]mail2.sysu.edu.cn)

和仿真器相关的一些概念


功能仿真和时序仿真

- 功能仿真
 - 捕捉和模拟基本的指令语义
 - 更新处理器状态（寄存器，存储，...）  程序的本质就是状态机
 - 生成正确的程序输出
- 时序仿真
 - 一般要实现功能仿真
 - 实现各种微体系架构
 - 通过捕捉事件的时序信息来获得程序的执行时间
- 功能仿真要比时序仿真快

和仿真器相关的一些概念

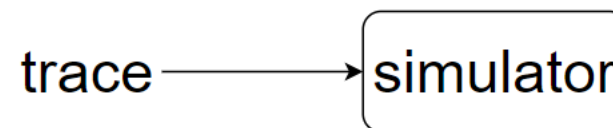
功能仿真和时序仿真

课后思考：GPGPU-Sim中有类似的功能仿真和时序仿真吗？

- 功能仿真
 - 捕捉和模拟基本的指令语义
 - 更新处理器状态（寄存器，存储，...）  程序的本质就是状态机
 - 生成正确的程序输出
- 时序仿真
 - 一般要实现功能仿真
 - 实现各种微体系架构
 - 通过捕捉事件的时序信息来获得程序的执行时间
- 功能仿真要比时序仿真快

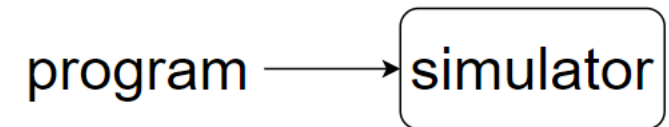
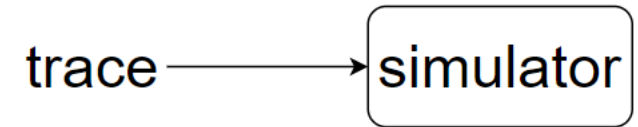
Execution vs. trace-driven

- Trace-driven 仿真
 - 真实执行程序并在执行过程中生成trace信息
 - 仿真器读入trace信息来进行仿真
- Execution-driven 仿真
 - 仿真器真正“执行”程序



Execution vs. trace-driven

- Trace-driven 仿真
 - 真实执行程序并在执行过程中生成trace信息
 - 仿真器读入trace信息来进行仿真
- Execution-driven 仿真
 - 仿真器真正“执行”程序



实验课需要了解的仿真器

- gem5
- Accel-Sim(include GPGPU-Sim)

gem5学术界的地位

谷歌学术搜索gem5有8370条相关结果

Google 学术搜索 gem5 登录

文章 找到约 8,370 条结果 (用时0.03秒) 我的个人学术档案 我的图书馆

时间不限
2022以来
2021以来
2018以来
自定义范围...

按相关性排序
按日期排序

不限语言
中文网页

The gem5 simulator [PDF] wustl.edu
[N Binkert](#), [B Beckmann](#), [G Black](#), [SK Reinhardt](#)... - ACM SIGARCH ..., 2011 - dl.acm.org
... The gem5 community is very active and leverages a number of collaborative technologies to foster gem5 use and development, including mailing lists, a wiki, web-based patch reviews, ...
☆ 保存 引用 被引用次数: 5172 相关文章 所有 18 个版本

gem5-gpu: A heterogeneous cpu-gpu simulator [HTML] escholarship.org
[J Power](#), [J Hestness](#), [MS Orr](#), [MD Hill](#)... - IEEE Computer ..., 2014 - ieeexplore.ieee.org
... are minimal, we will work to more closely integrate with gem5 to ease the use of gem5-gpu. Additionally, as both GPGPU-Sim and gem5 evolve, gem5-gpu will take advantage of new ...
☆ 保存 引用 被引用次数: 283 相关文章 所有 16 个版本

而gem5正式发布的论文“The gem5 simulator”有高达5172的引用次数

gem5工业界的地位和未来发展

- 同时也被许多工业界公司使用，包括ARM、AMD、Google、Micron(美光科技)、HP(惠普)、Samsung等。
- 许多公司也积极为gem5添加了新功能或改进了现有功能。
- 近年来，gem5社区仍在积极更新与开发，以支持未来15年的计算机架构研究。

gem5是什么

- 前身为密歇根大学的m5项目与威尼斯康星大学的GEMS项目。2011年m5与GEMS合并为gem5
- 是计算机系统和体系结构研究的模块化平台，包括系统级体系结构和处理器微体系结构
- 是社区主导并开放管理的(开源)
- 最初是为学术界的计算机体系结构研究而设计的，但它现在已经发展成为学术界、工业界用于研究和教学方面的计算机系统设计



<https://www.gem5.org/>

gem5仿真器是做什么的

- gem5是一个**模块化**离散事件驱动的计算机系统模拟器平台。这意味着，架构研究人员在研究新架构时，只需添加或修改特定于目标的功能模块，而不需要了解模拟器的方方面面。
- 同时，也使得gem5容易与其他模拟器联合仿真，构建联合模拟器平台或搭建自己的模拟系统，目前已有许多工作基于gem5开发了模拟平台。

gem5由什么语言编写？

gem5主要由C++和python编写的。

- 其中C++占绝大多数，主要负责底层架构的具体实现等
- python则负责对象的初始化、配置和模拟控制等。

另外包含了两个领域特定语言DSL

- 其中ISADSL负责统一二进制指令的解码和语义规范
- SLICC用于实现缓存一致性协议。

gem5的仿真模式：

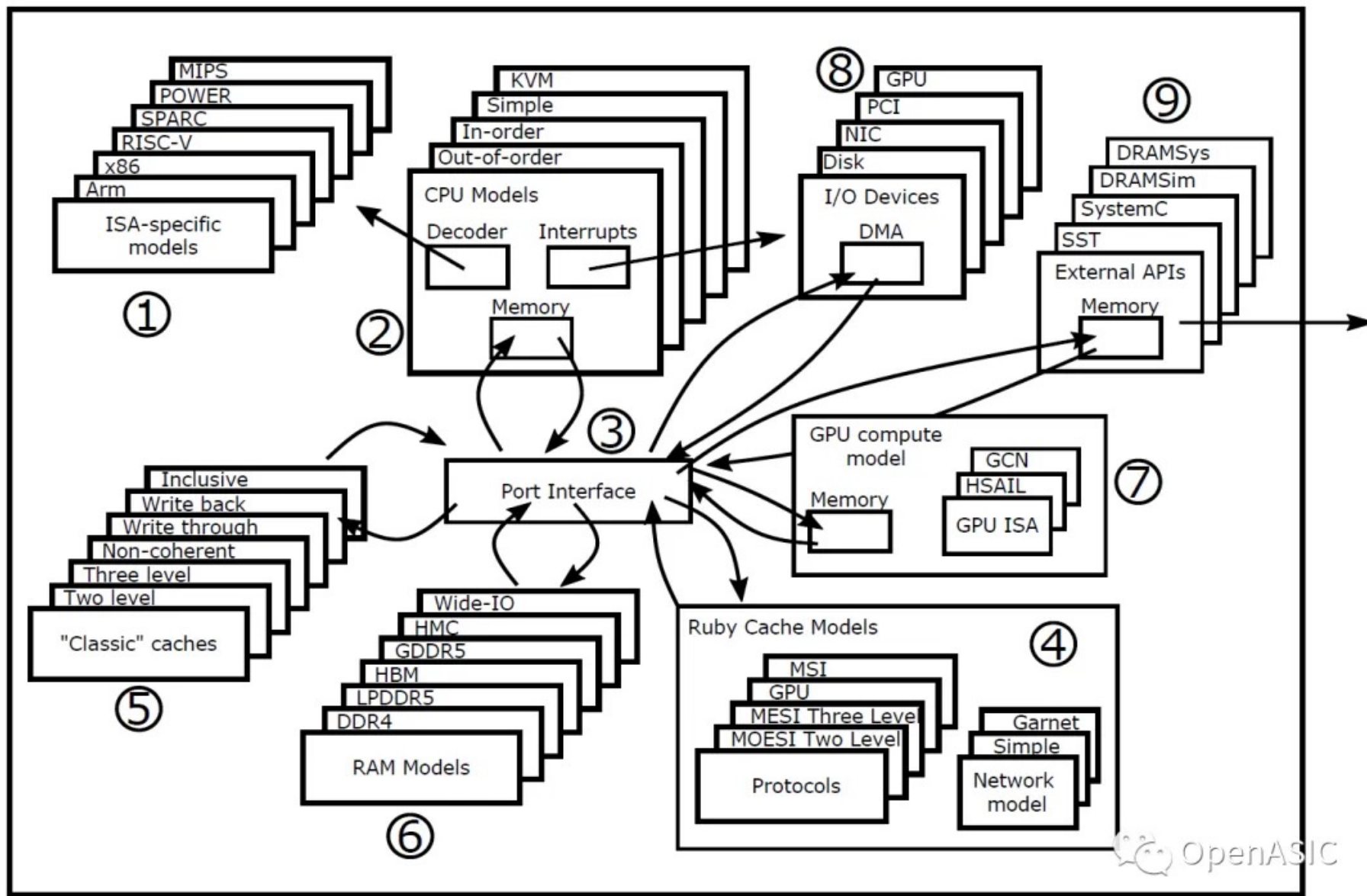
Full system mode (FS)

- 启动完整的基于Linux的操作系统（例如Ubuntu20.04）
- 支持各种IO和其他外设等
- 可以更好地执行多线程基准
- 提高了仿真精度，提供更真实的交互

Syscall emulation mode (SE)

- 不引导操作系统，内部模拟操作系统，模拟系统调用
- 简化了地址翻译模型，是一个无调度的简单模型
- 没有线程调度器，线程必须静态映射，限制了多线程应用
- 具有较快的仿真速度

gem5的主要模块：



OpenASIC

GPGPU-SIM学术界的地位 谷歌学术搜索gpgpu-sim有1650条相关结果

Google 学术搜索 gpgpu-sim 找到约 1,650 条结果 (用时0.03秒)

文章

时间不限
2022以来
2021以来
2018以来
自定义范围...

按相关性排序
按日期排序

不限语言
中文网页
简体中文网页

类型不限
评论性文章

包括专利
 包含引用

[PDF] GPGPU-Sim Overview. [PDF] osti.gov
C Hughes, R Green, GR Voskuilen, M Zhang, T Rogers - 2019 - osti.gov
... GPGPU-Sim simulates kernel Transfer data to GPU memory GPU kernels runs on
GPGPU-Sim: ... GPGPU-Sim with SST: GPGPU-Sim brings promising GPU model to SST ...
☆ 保存 引用 被引用次数: 1 相关文章

Experiment and enabled flow for GPGPU-sim simulators with fixed-point instructions
CL Lee, MY Hsu, BS Lu, MY Hung, JK Lee - Journal of Systems ..., 2020 - Elsevier
... This section provides background information about the GPGPU-Sim architecture and the motivation for this study as well as the fixed-point enabled flow, which demonstrates source ...
☆ 保存 引用 被引用次数: 3 相关文章

Analyzing CUDA workloads using a detailed GPU simulator [PDF] researchgate.net
A Bakhoda, GL Yuan, WWL Fung... - ... analysis of systems ..., 2009 - ieeexplore.ieee.org
... on GPGPU-Sim (as shown in Figure 3(b)). Before the first simulation session, GPGPUSim ...
The GPU binary (cubin.bin) produced by ptxas is not used by GPGPU-Sim. After parsing the ...
☆ 保存 引用 被引用次数: 1893 相关文章 所有 24 个版本

而gpgpu-sim发布的论文“Analyzing CUDA workloads using a detailed GPU simulator”有高达1893的引用次数

GPGPU-SIM History(paper)

2009 ISPASS

Analyzing CUDA Workloads Using a Detailed GPU Simulator

Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong and [Tor M. Aamodt](#)

2020 ISCA

Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling

[Mahmoud Khairy](#), Zhesheng Shen, [Tor M. Aamodt](#), [Timothy G. Rogers](#)

2021 MICRO

AccelWattch: A Power Modeling Framework for Modern GPUs

Vijay Kandiah, Scott Peverelle, [Mahmoud Khairy](#), Junrui Pan, Amogh Manjunath, [Timothy G. Rogers](#), [Tor M. Aamodt](#), Nikos Hardavellas

Main character about GPGPU-SIM

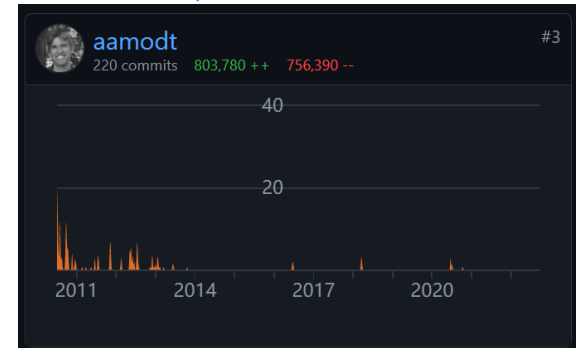
Accel-sim/gpgpu-sim_distribution
Contributors



↓ student

Tor Aamodt

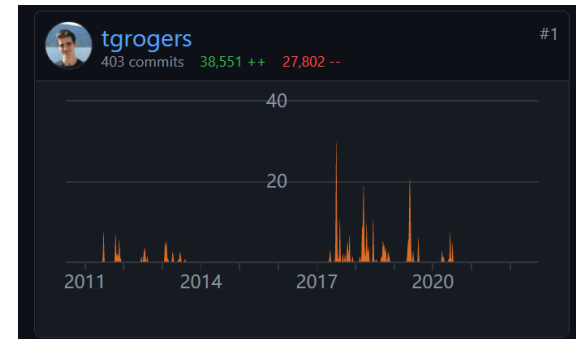
Professor, Electrical and Computer Engineering, University of British Columbia



↓ student

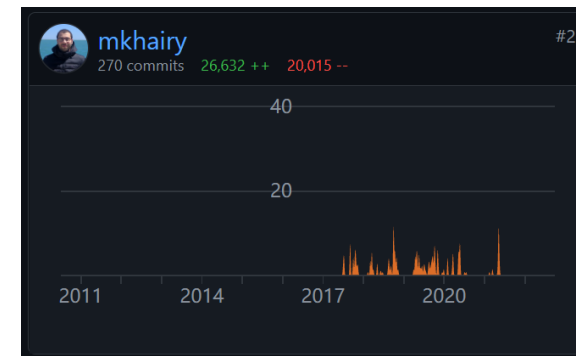
Tim Rogers

Associate Professor, Computer Engineering, University of Purdue University



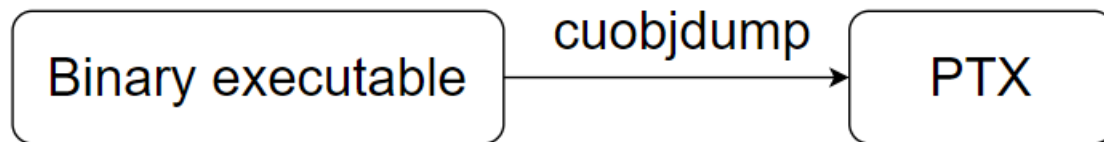
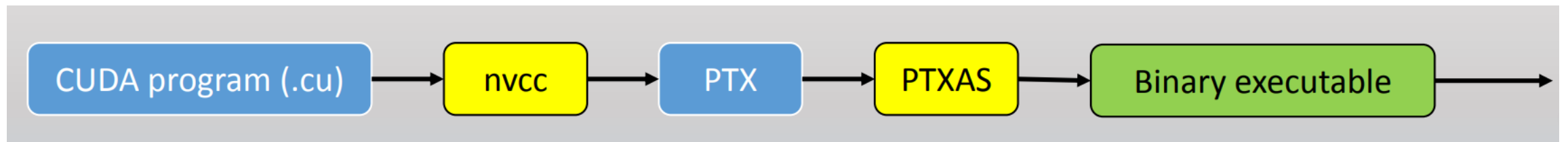
Mahmoud Khairy Abdallah

PhD, Computer Engineering, University of Purdue University



GPGPU-Sim是什么？

- 一个针对NVIDIA GPU微体系结构的性能仿真器
- 仿真执行Parallel Thread eXecution(PTX)指令集--NVIDA CUDA中使用的“伪汇编”语言



Hello World in CUDA

<https://docs.nvidia.cn/cuda/>

Hello World in CUDA

CUDA是C++的扩展，即使你以前从来没了解过CUDA，也可以很快地上手

<https://docs.nvidia.cn/cuda/>

Hello World in CUDA

CUDA是C++的扩展，即使你以前从来没了解过CUDA，也可以很快地上手

CUDA中重要的概念：

- Host端(CPU) 和 Device端(GPU)
- 核函数(kernel function)：__global__ 修饰的函数

Hello World in CUDA

CUDA是C++的扩展，即使你以前从来没了解过CUDA，也可以很快地上手

CUDA中重要的概念：

- Host端(CPU) 和 Device端(GPU)
- 核函数(kernel function)：__global__ 修饰的函数

函数修饰符	device端调用	device端执行	host端调用	host端执行
__device__	√	√		
__global__		√	√	
__host__ (默认)			√	√

Hello World in CUDA

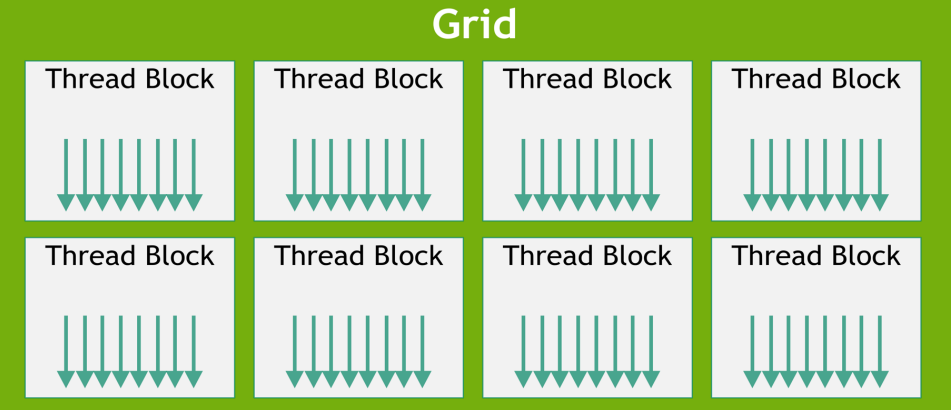
CUDA是C++的扩展，即使你以前从来没了解过CUDA，也可以很快地上手

CUDA中重要的概念：

- Host端(CPU) 和 Device端(GPU)
- 核函数(kernel function)：__global__ 修饰的函数

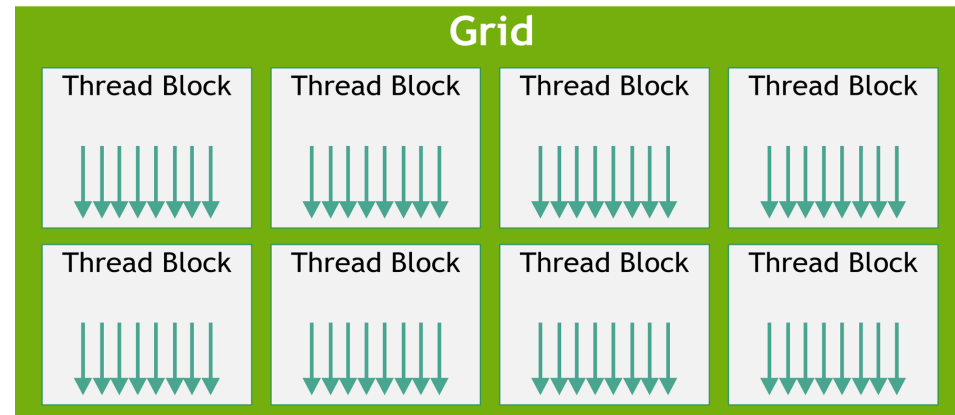
函数修饰符	device端调用	device端执行	host端调用	host端执行
__device__	√	√		
__global__		√	√	
__host__ (默认)			√	√

核函数的调用方法和其他函数略有不同：kernel<<<gridDim,blockDim>>>(args...)



CUDA中的线程集合抽象：

- 每个核函数会启动一个**Grid**
- **Grid**中包含很多**Thread Block**
- **Thread Block**中包含很多个线程

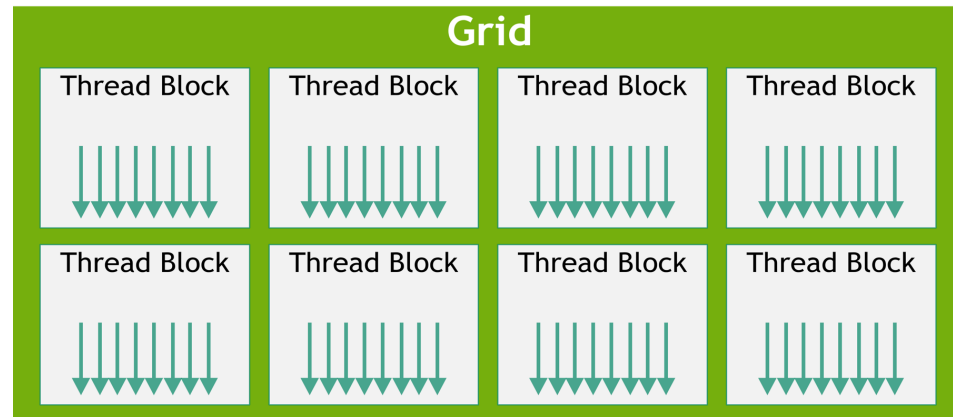


CUDA中的线程集合抽象：

- 每个核函数会启动一个**Grid**
- **Grid**中包含很多**Thread Block**
- **Thread Block**中包含很多个线程

GridDim代表Grid中包含多少个Thread Block

BlockDim代表Thread Block中包含多少个线程



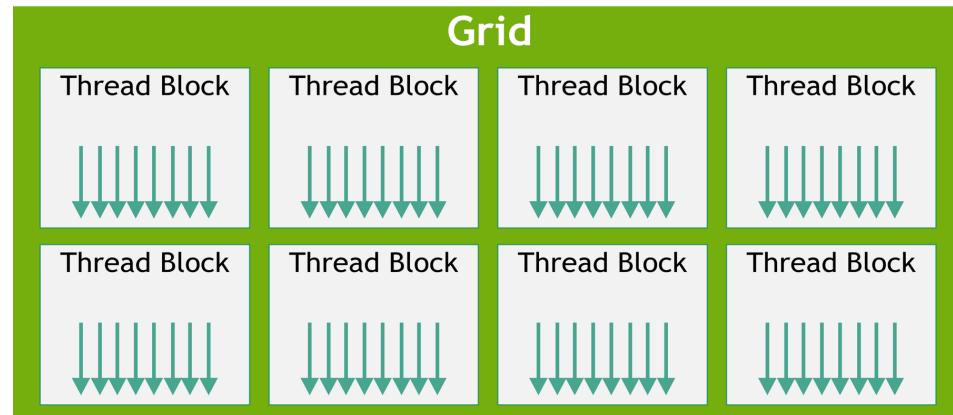
CUDA中的线程集合抽象：

- 每个核函数会启动一个**Grid**
- **Grid**中包含很多**Thread Block**
- **Thread Block**中包含很多个线程

GridDim代表Grid中包含多少个Thread Block

BlockDim代表Thread Block中包含多少个线程

准确地说GridDim和BlockDim是CUDA中dim3变量(代表3维大小)，可以先简化为数字

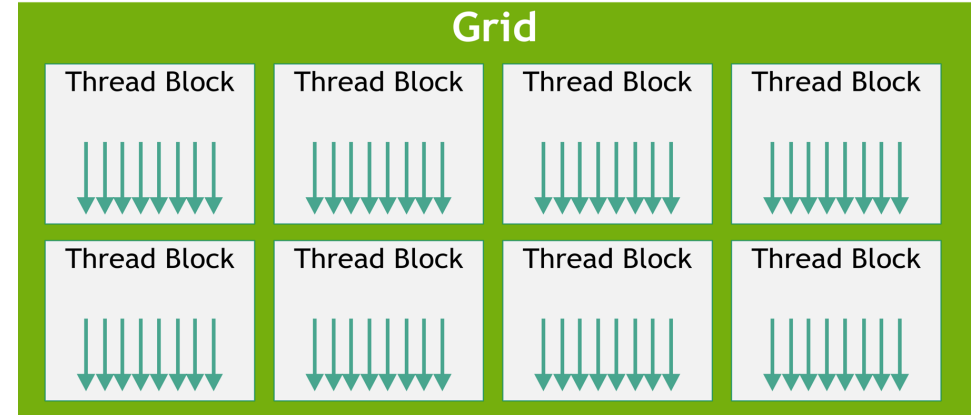


CUDA中的线程集合抽象：

- 每个核函数会启动一个**Grid**
- **Grid**中包含很多**Thread Block**
- **Thread Block**中包含很多个线程

GridDim代表Grid中包含多少个Thread Block

BlockDim代表Thread Block中包含多少个线程



准确地说GridDim和BlockDim是CUDA中dim3变量(代表3维大小)，可以先简化为数字

CUDA核函数编程思想：

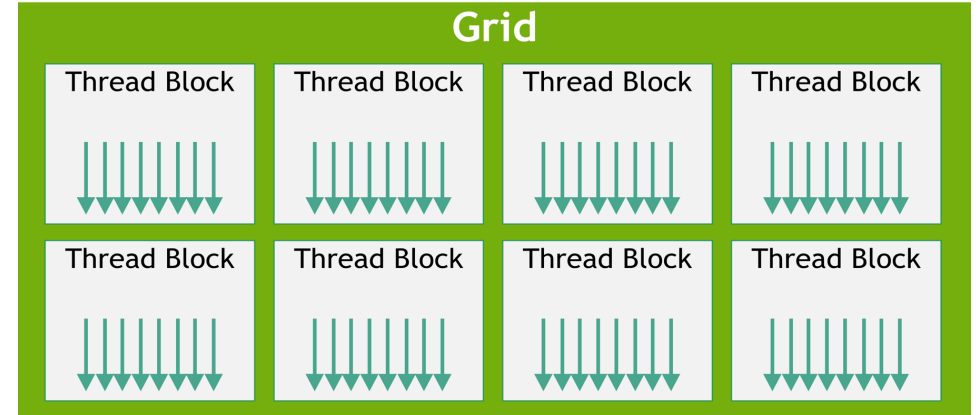
- TLP(Thread Level Parallelism)线程级并行性
- 让每个线程知道自己的任务是什么(把任务划分为多个子任务)

CUDA中的线程集合抽象：

- 每个核函数会启动一个**Grid**
- **Grid**中包含很多**Thread Block**
- **Thread Block**中包含很多个线程

GridDim代表Grid中包含多少个Thread Block

BlockDim代表Thread Block中包含多少个线程



准确地说GridDim和BlockDim是CUDA中dim3变量(代表3维大小)，可以先简化为数字

CUDA核函数编程思想：

- TLP(Thread Level Parallelism)线程级并行性
- 让每个线程知道自己的任务是什么(把任务划分为多个子任务)

核函数是如何在GPU执行的呢？SIMT(Single Instruction Multiple Thread)

核函数启动的所有线程都会执行相同的代码，就是核函数本身

<https://docs.nvidia.cn/cuda/>

那线程怎么知道“我是谁呢?”

核函数中有些特殊的变量，它们是CUDA提供给编程者的了解“我是谁?”的变量

那线程怎么知道“我是谁呢?”

核函数中有些特殊的变量，它们是CUDA提供给编程者的了解“我是谁?”的变量

Thread Block内的线程索引 : threadIdx.x (threadIdx.y threadIdx.z)

Grid 内的Thread Block索引 : blockIdx.x (blockIdx.y blockIdx.z)

那线程怎么知道“我是谁呢?”

核函数中有些特殊的变量，它们是CUDA提供给编程者的了解“我是谁?”的变量

Thread Block内的线程索引 : threadIdx.x (threadIdx.y threadIdx.z)

Grid 内的Thread Block索引 : blockIdx.x (blockIdx.y blockIdx.z)

如果我在CPU端有一部分数据想拷贝到GPU端，或者我想在GPU端分配存储空间怎么办？

那线程怎么知道“我是谁呢?”

核函数中有些特殊的变量，它们是CUDA提供给编程者的了解“我是谁?”的变量

Thread Block内的线程索引 : threadIdx.x (threadIdx.y threadIdx.z)

Grid 内的Thread Block索引 : blockIdx.x (blockIdx.y blockIdx.z)

如果我在CPU端有一部分数据想拷贝到GPU端，或者我想在GPU端分配存储空间怎么办？

CUDA runtime API

那线程怎么知道“我是谁呢?”

核函数中有些特殊的变量，它们是CUDA提供给编程者的了解“我是谁?”的变量

Thread Block内的线程索引 : threadIdx.x (threadIdx.y threadIdx.z)

Grid 内的Thread Block索引 : blockIdx.x (blockIdx.y blockIdx.z)

如果我在CPU端有一部分数据想拷贝到GPU端，或者我想在GPU端分配存储空间怎么办？

CUDA runtime API

```
cudaMalloc(void** devPtr, size_t size);
```

```
cudaMemcpy(void *dst, const void *src, size_t count, cudaMemcpyKind kind)
```

那线程怎么知道“我是谁呢?”

核函数中有些特殊的变量，它们是CUDA提供给编程者的了解“我是谁?”的变量

Thread Block内的线程索引 : threadIdx.x (threadIdx.y threadIdx.z)

Grid 内的Thread Block索引 : blockIdx.x (blockIdx.y blockIdx.z)

如果我在CPU端有一部分数据想拷贝到GPU端，或者我想在GPU端分配存储空间怎么办？

CUDA runtime API

课后思考：为什么cudaMalloc第一个参数需要二级指针?

```
cudaMalloc(void** devPtr, size_t size);
```

```
cudaMemcpy(void *dst, const void *src, size_t count, cudaMemcpyKind kind)
```

那线程怎么知道“我是谁呢?”

核函数中有些特殊的变量，它们是CUDA提供给编程者的了解“我是谁?”的变量

Thread Block内的线程索引 : threadIdx.x (threadIdx.y threadIdx.z)

Grid 内的Thread Block索引 : blockIdx.x (blockIdx.y blockIdx.z)

如果我在CPU端有一部分数据想拷贝到GPU端，或者我想在GPU端分配存储空间怎么办？

CUDA runtime API

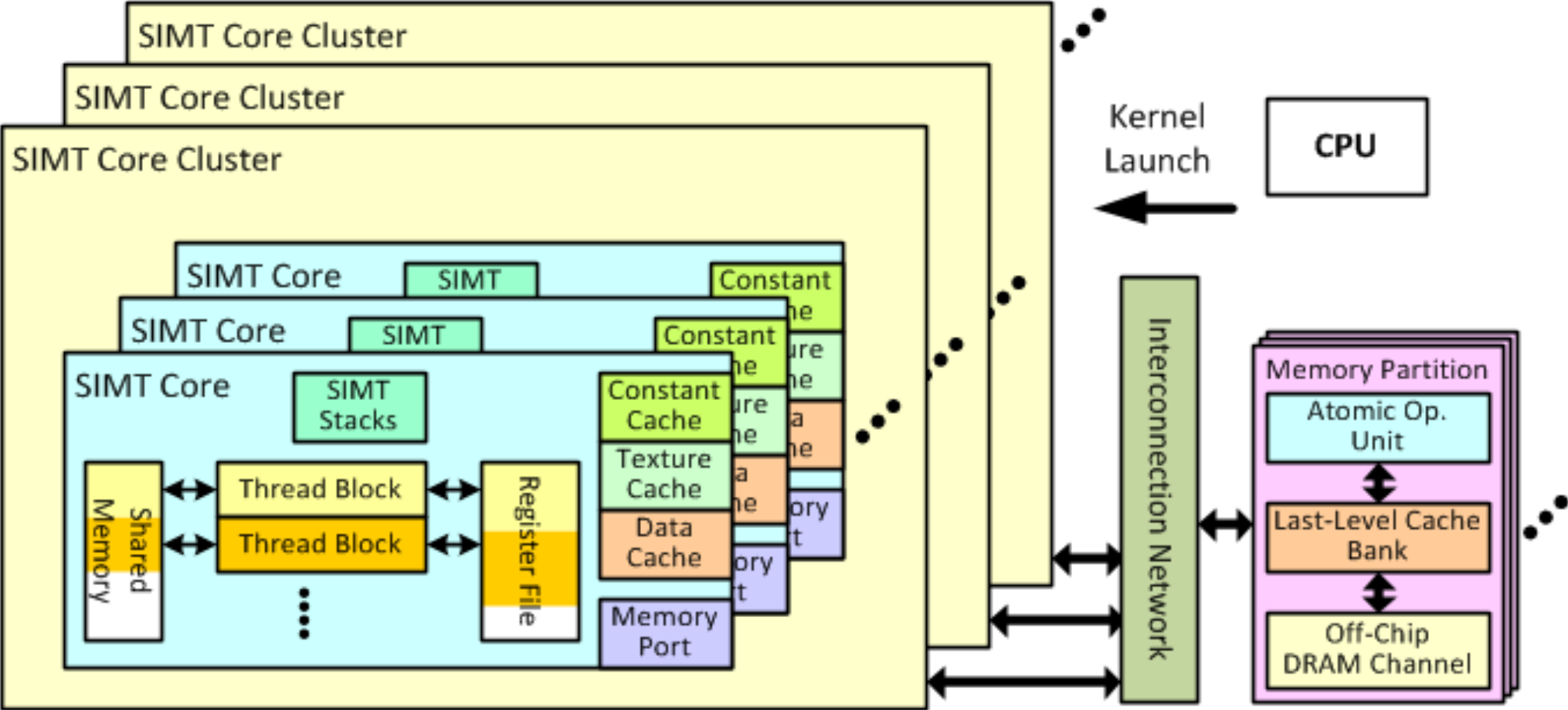
课后思考：为什么cudaMalloc第一个参数需要二级指针?

```
cudaMalloc(void** devPtr, size_t size);
```

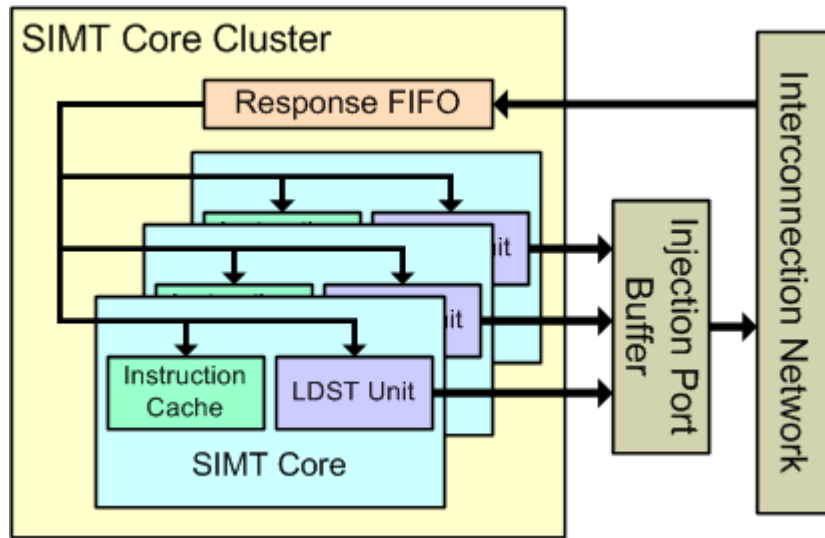
```
cudaMemcpy(void *dst, const void *src, size_t count, cudaMemcpyKind kind)
```

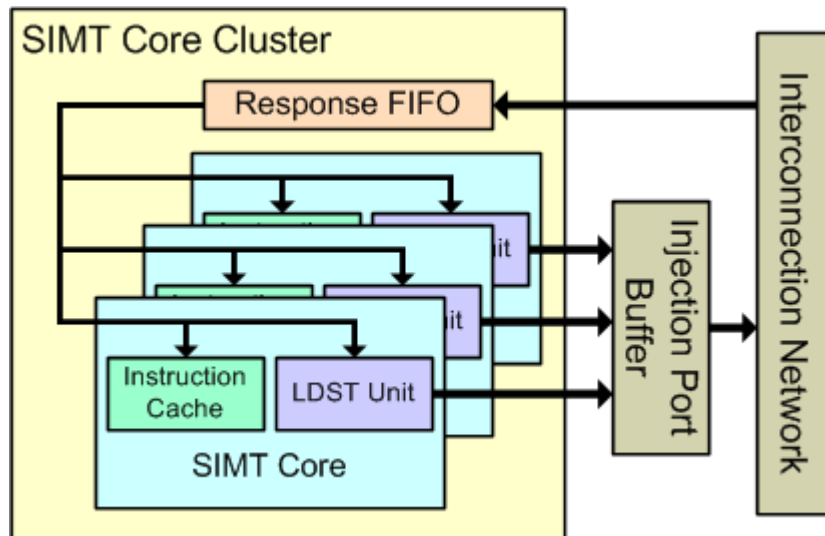
Talk is cheap, show me the code !

GPGPU-Sim架构图

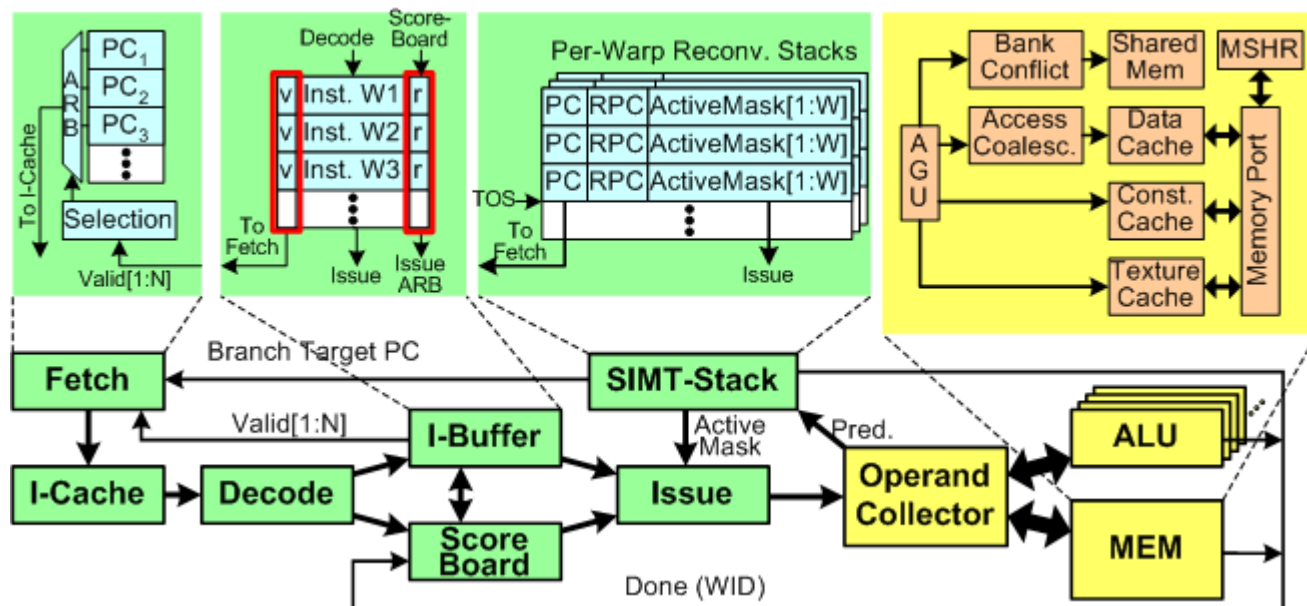


http://gpgpu-sim.org/manual/index.php/Main_Page

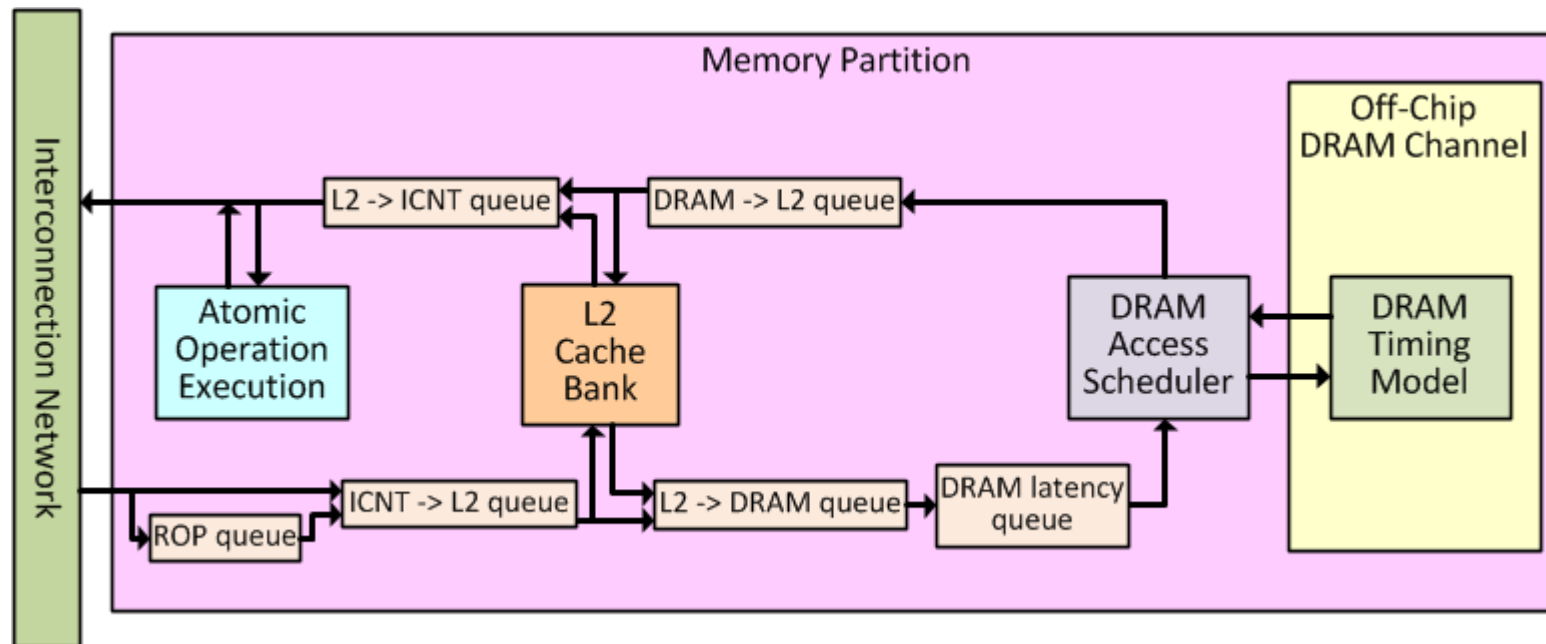
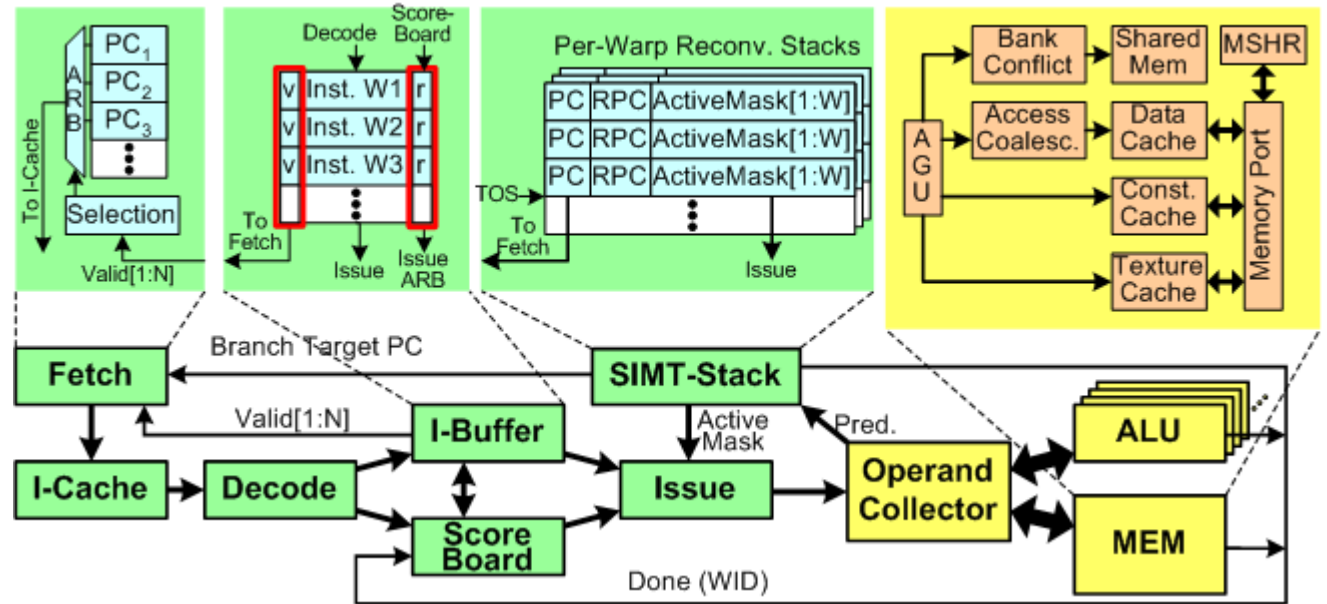
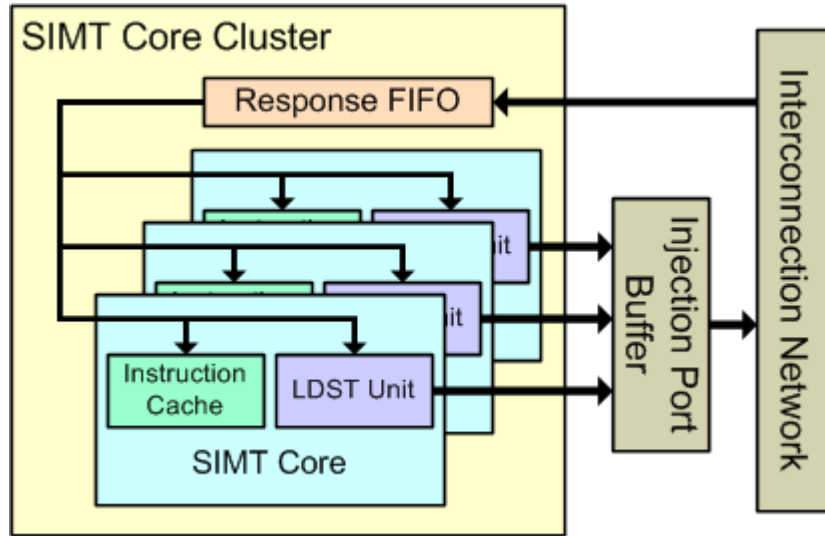




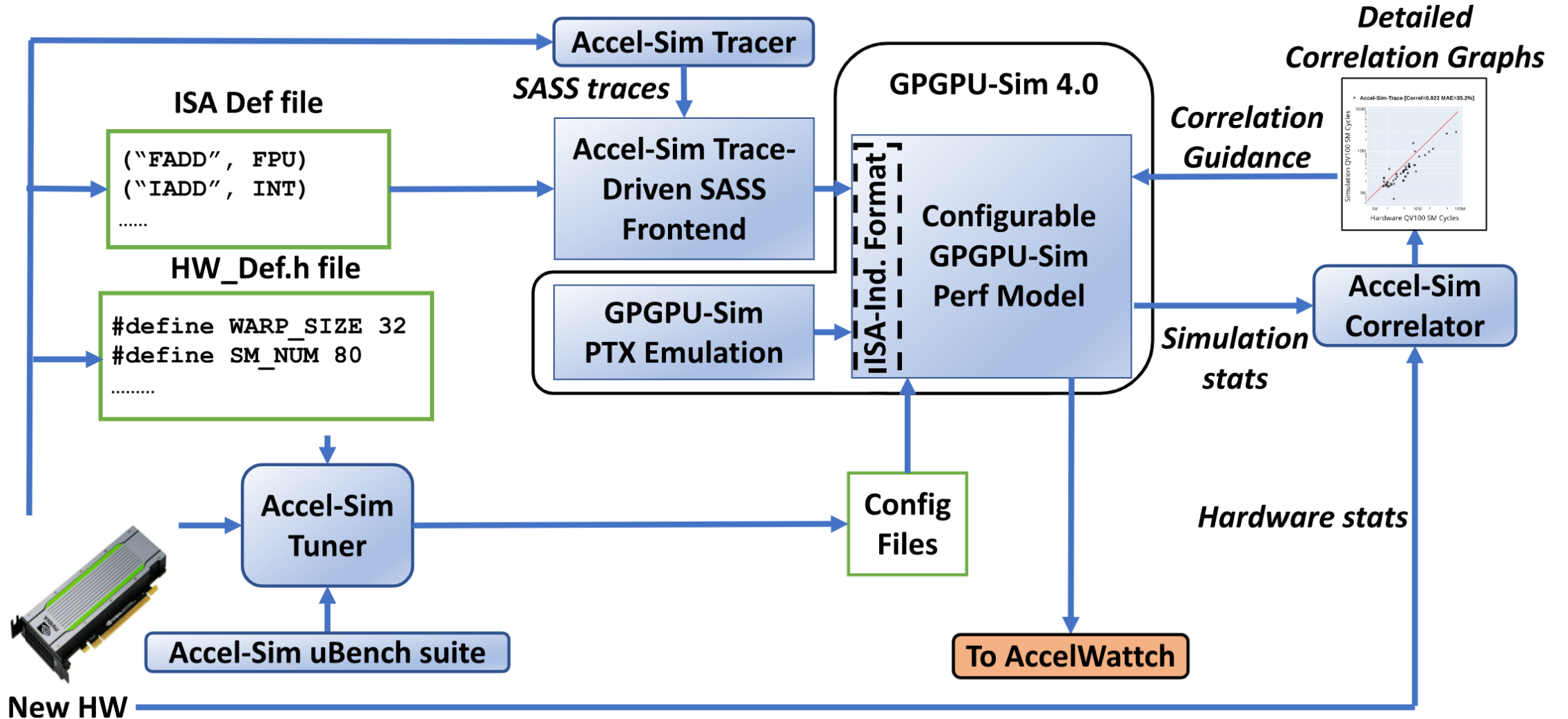
SIMT Core执行阶段



SIMT Core执行阶段



Accel-Sim VS GPGPU-Sim



Hello World in Accel-Sim

准备安装环境(选择以下方法中的一个即可)

- 安装Ubuntu18.04(WSL or VMware or 双系统)

```
sudo apt-get install -y wget build-essential xutils-dev bison zlib1g-dev flex \  
libglu1-mesa-dev git g++ libssl-dev libxml2-dev libboost-all-dev git g++ \  
libxml2-dev vim python-setuptools python-dev build-essential python-pip
```

```
pip3 install pyyaml plotly psutil
```

```
wget
```

```
http://developer.download.nvidia.com/compute/cuda/11.0.1/local\_installers/cuda\_11.0.1\_450.36.06\_linux.run
```

```
sh cuda_11.0.1_450.36.06_linux.run --silent --toolkit
```

```
rm cuda_11.0.1_450.36.06_linux.run
```

- 通过docker镜像(docker pull accel-sim/ubuntu-18.04_cuda-11)

<https://accel-sim.github.io/>

获取Accel-Sim

```
git clone -b dev https://github.com/accel-sim/accel-sim-framework.git
```

开始编译(构建)GPGPU-Sim

在accel-sim-framework目录下执行

```
pip3 install -r requirements.txt  
source ./gpu-simulator/setup_environment.sh  
make -j -C ./gpu-simulator/
```

测试是否安装成功

在accel-sim-framework目录下执行

```
. travis.sh          travis.sh是在做什么 : sim-mode:trace config:QV100 program:rodinia
```

<https://accel-sim.github.io/>

Hello World in GPGPU-Sim

https://github.com/accel-sim/gpgpu-sim_distribution

Hello World in GPGPU-Sim

GPGPU-Sim是Accel-Sim子模块：位于 `accel-sim-framework/gpu-simulator/gpgpu-sim`
可以独立于accel-sim使用

https://github.com/accel-sim/gpgpu-sim_distribution

Hello World in GPGPU-Sim

GPGPU-Sim是Accel-Sim子模块：位于 `accel-sim-framework/gpu-simulator/gpgpu-sim` 可以独立于accel-sim使用

我写了一个CUDA程序test.cu，如何在GPGPU-Sim仿真呢？

https://github.com/accel-sim/gpgpu-sim_distribution

Hello World in GPGPU-Sim

GPGPU-Sim是Accel-Sim子模块：位于 `accel-sim-framework/gpu-simulator/gpgpu-sim` 可以独立于accel-sim使用

我写了一个CUDA程序test.cu，如何在GPGPU-Sim仿真呢？

首先通过nvcc(CUDA程序的编译器)将test.cu编译成可执行文件(需要加“-lcudart”编译参数)

```
nvcc -lcudart test.cu -o test
```

https://github.com/accel-sim/gpgpu-sim_distribution

Hello World in GPGPU-Sim

GPGPU-Sim是Accel-Sim子模块：位于 `accel-sim-framework/gpu-simulator/gpgpu-sim` 可以独立于accel-sim使用

我写了一个CUDA程序`test.cu`，如何在GPGPU-Sim仿真呢？

首先通过`nvcc`(CUDA程序的编译器)将`test.cu`编译成可执行文件(需要加“-lcudart”编译参数)

```
nvcc -lcudart test.cu -o test
```

在`gpgpu-sim`目录下执行
`. setup_environment`

https://github.com/accel-sim/gpgpu-sim_distribution

Hello World in GPGPU-Sim

GPGPU-Sim是Accel-Sim子模块：位于 `accel-sim-framework/gpu-simulator/gpgpu-sim` 可以独立于accel-sim使用

我写了一个CUDA程序test.cu，如何在GPGPU-Sim仿真呢？

首先通过nvcc(CUDA程序的编译器)将test.cu编译成可执行文件(需要加“-lcudart”编译参数)

```
nvcc -lcudart test.cu -o test
```

在gpgpu-sim目录下执行

```
. setup_environment
```

在gpgpu-sim/configs/tested-cfgs目录下选择一个你喜欢的配置，把该配置文件夹目录下的所有文件拷贝到和“test”同级目录，在test所在目录下执行“./test”，如果你看到一大堆看不懂的输出信息，证明你仿真成功了

https://github.com/accel-sim/gpgpu-sim_distribution

Hello World in GPGPU-Sim

课后思考题：如果不加“-lcudart”或不执行
“. setup_environment”，还可以在GPGPU-Sim上仿真吗？为什么？
提示：在编译好“test”后，尝试执行“ldd test”，或者阅读
setup_environment，你有什么发现吗？

GPGPU-Sim是Accel-Sim子模块：位于 accel-sim-framework/gpu-simulator/gpgpu-sim
可以独立于accel-sim使用

我写了一个CUDA程序test.cu，如何在GPGPU-Sim仿真呢？

首先通过nvcc(CUDA程序的编译器)将test.cu编译成可执行文件(需要加“-lcudart”编译参数)

```
nvcc -lcudart test.cu -o test
```

在gpgpu-sim目录下执行
. setup_environment

在gpgpu-sim/configs/tested-cfgs目录下选择一个你喜欢的配置，把该配置文件夹目录下的所有文件拷贝到和“test”同级目录，在test所在目录下执行“./test”，如果你看到一大堆看不懂的输出信息，证明你仿真成功了

https://github.com/accel-sim/gpgpu-sim_distribution

实验介绍 SYSU-ARCH

实验网站 <https://arcsysu.github.io/SYSU-ARCH/>

实验相关的所有信息将会发布到实验网站上

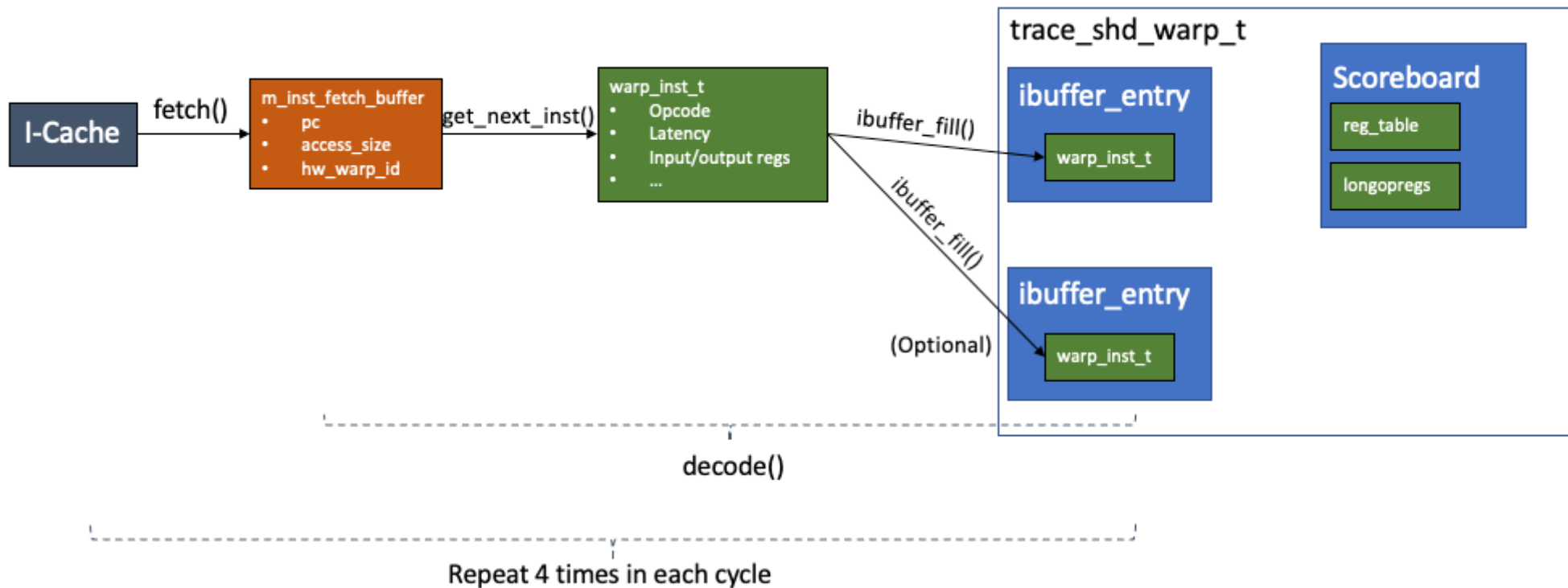
如果大部分同学感觉实验难度过大或者有设置不合理的地方，请及时联系我们，我们会根据情况对实验做出相关调整

GPGPU-Sim 高级使用方法

Reference <https://apuaachen.github.io/Accel-SIM-Code-Study/>

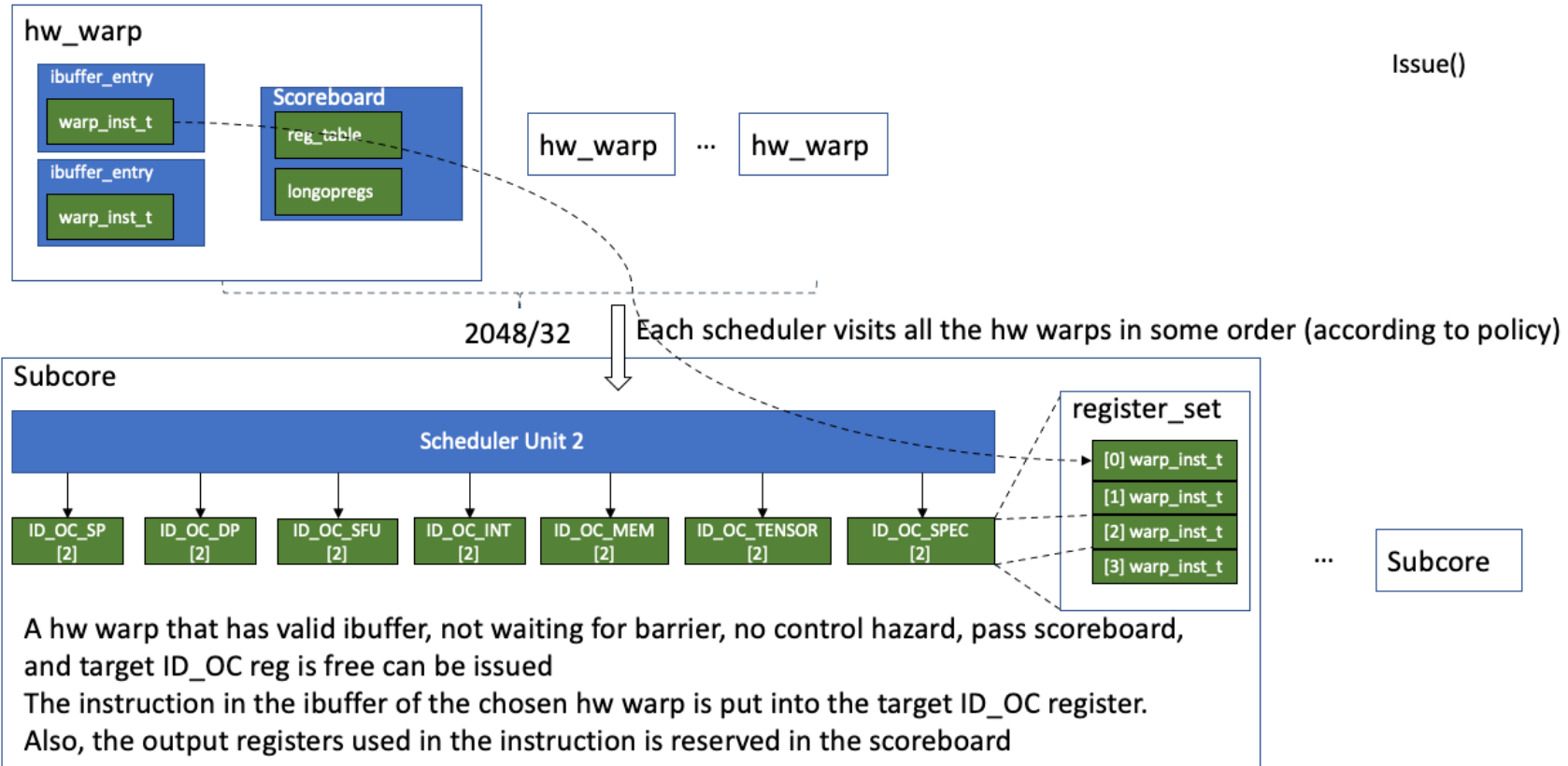
SM ↔ shader_core_ctx

CYCLE() : fetch && decode



SM ↔ shader_core_ctx

CYCLE() : issue



SM ↔ shader_core_ctx

CYCLE() : issue



GTO: greedy then oldest

LRR: loose round robin

TL: two level

Issue ↔ scheduler_unit::cycle()

- iterate warp set at some order (GTO LRR TL)
 1. IF valid (! ibuffer_empty && ! wait_barrier && ! control_hazard)
 2. IF scoreboard check pass (fail eg. wait for memory inst)
 3. IF ID_OC available

Stall at 1 (WO_Idle)=> not enough TLP

Stall at 2 (WO_Scoreboard)=> not enough TLP for latency hiding

Stall at 3 (Stall) PS : content in parenthesis represent metric in gpgpu-sim output

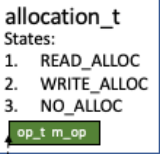
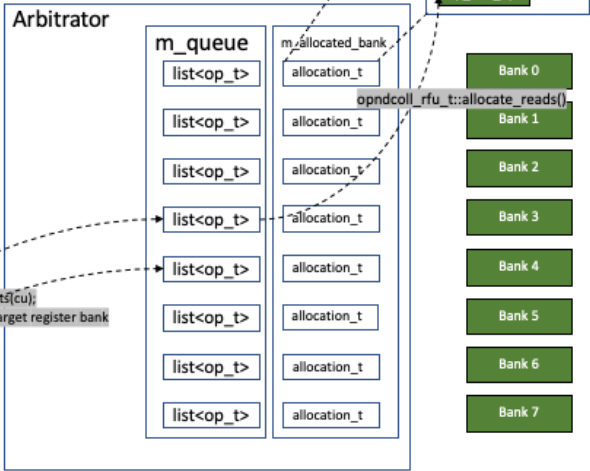
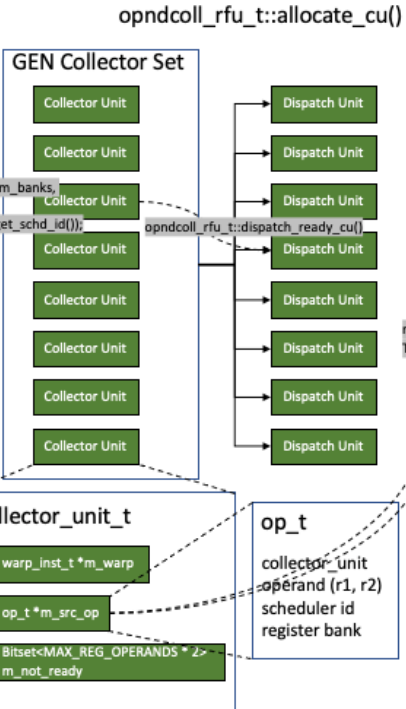
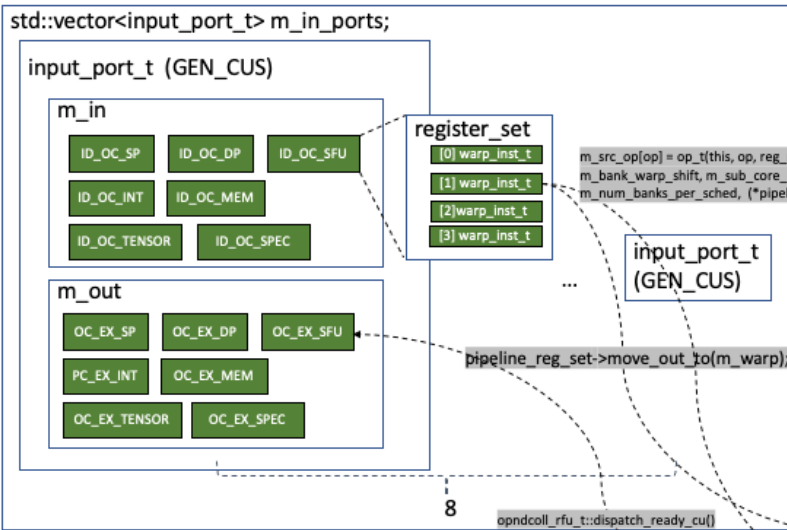
SM ↔ shader_core_ctx

CYCLE() : read_operands



4. Each Dispatch Unit find a ready Collector Unit. The Collector Unit is ready when it has a valid instruction, all this src operands are ready, and its OC_EX pipeline register is available. When dispatching the collector unit, its m_warp is moved to the output pipeline register, and its context is reset.

3. The Arbitrator checks all requests in the head of the m_queue and returns a list of op_t that are in different register banks and the banks are not under state Write. These requests are allocated to the allocation_t of the corresponding bank. The m_not_ready bit in the collector_unit_t is then reset.

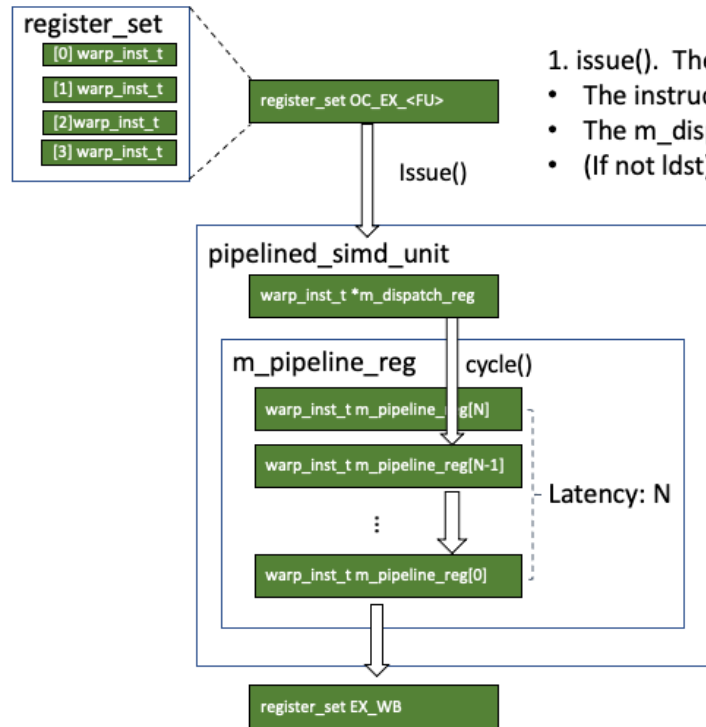


1. Loop over all the input_port_t. For each one, sweep the m_in and see if a register_set in it contains a valid warp_inst_t. After getting the warp_inst_t, find a free collector unit in the GEN collector set, put the warp_inst_t into the collector unit. All the src operands are encoded in op_t and appended to m_src_op. The corresponding bits in m_not_ready are set to 1.

2. The m_queue is a set of FIFO buffers that keeps all the requests to the register file. Each register bank has a corresponding buffer. After the instruction is issued to a collector unit, the src registers are decoded and the target register bank is resolved. The src register request is then pushed into the FIFO buffer of the target register bank.

SM ↔ shader_core_ctx

CYCLE() : execute



1. issue(). The instruction in the OC_EX_<FU> can be issued to the function unit if

- The instruction is ready
- The m_dispatch_reg of the unit is empty
- (If not ldst) The <latency_of_inst> bit of at least of one the result bus is not set

2. cycle(). The context in the m_dispatch_reg can be moved to the Latency-initial_interval m_pipeline_reg if its context is held for initial_interval cycles. The context of each m_pipeline_reg is moved to the next one in each cycle.

SM ↔ shader_core_ctx

CYCLE() : writeback

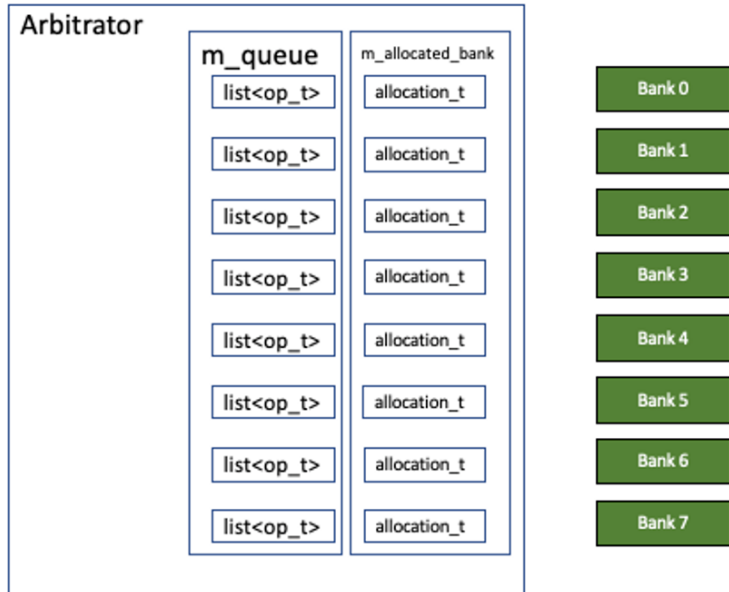


writeback()

register_set EX_WB



3. writeback(). Put the destination registers to the Arbitrator and write them to register file when the bank is available. After that, these reserved registers in the scoreboard are released.



LDST_UNIT (special simd_function_unit) CYCLE() : request



Generate memory request to lower memory level :

shared_cycle

constant_cycle

texture_cycle

memory_cycle

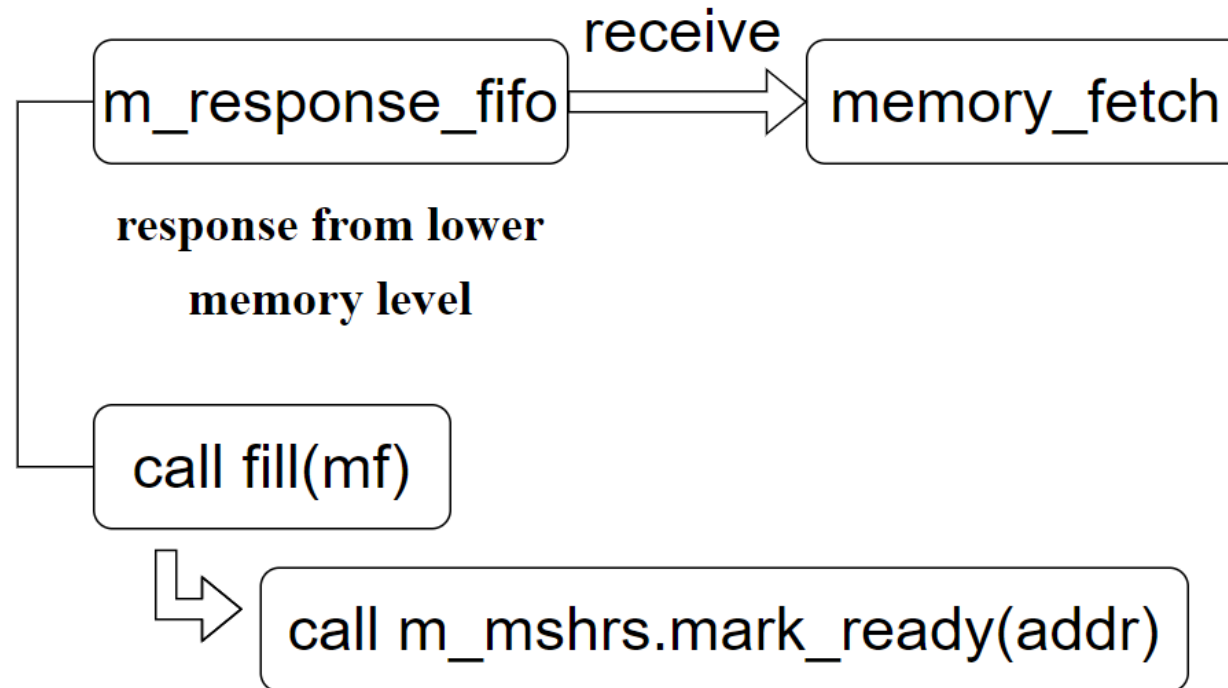


process_memory_access_queue_l1cache



put memory_fetch in l1_latency_queue

LDST_UNIT (special simd_function_unit) `CYCLE() : fill`



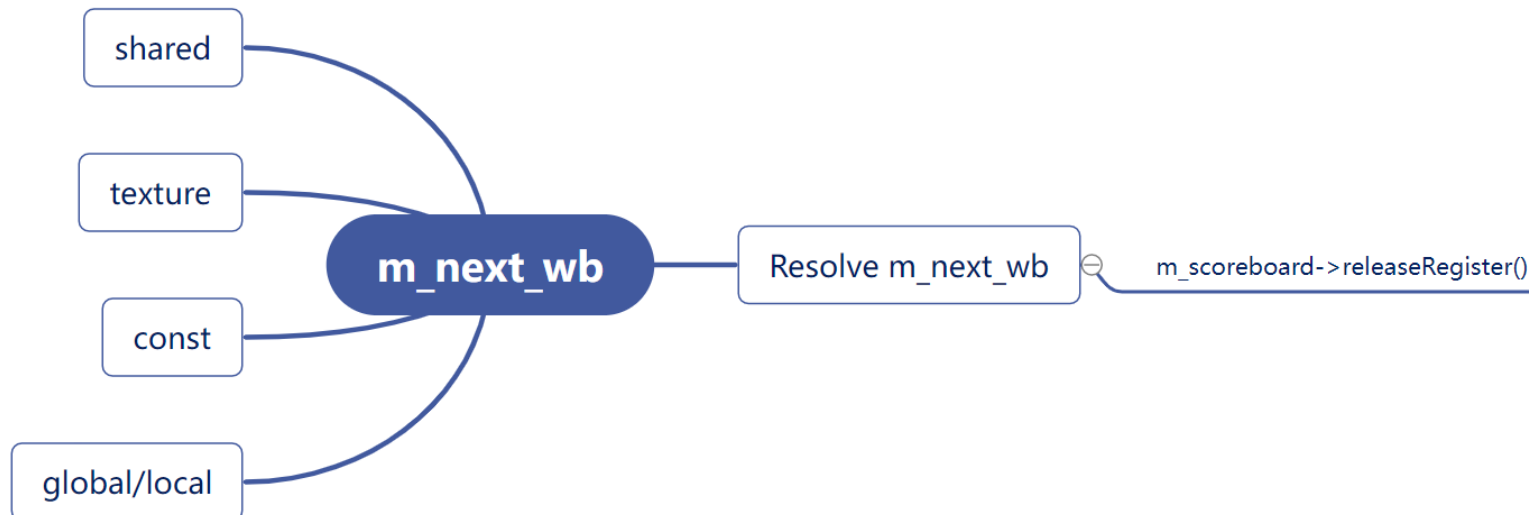
mshrs : missing hold registers

LDST_UNIT (special simd_function_unit) CYCLE() : writeback



Call `mshr.next_access()` to get writeback

put each writeback in `m_next_wb` at round robin order

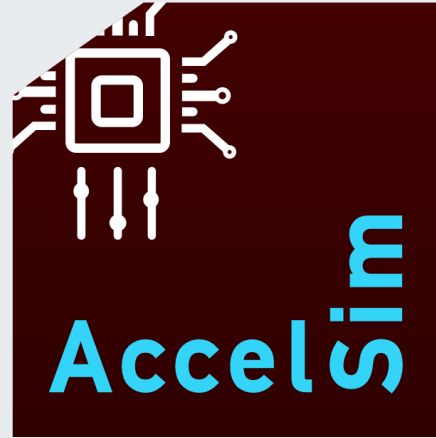


Cache::access()

ldst_unit::cycle() => L1_latency_queue_cycle() => m_L1D->access()

- **m_tag_array->probe()**
 - **return state[HIT MISS ...] and victim cache block idx**
- **process_tag_probe()**
 - **process based on state**
 - **read miss handler**
 - **read hit handler**
 - **write miss handler**
 - **wr_miss_wa_naive**
 - **wr_miss_wa_fetch_on_write**
 - **wr_miss_wa_lazy_fetch_on_read**
 - **wr_miss_wa_write_validate**
 - **wr_miss_no_wa**
 - **write hit handler**
 - **wr_hit_wb**
 - **wr_hit_wt**
 - **wr_hit_we**
 - **wr_hit_global_we_local_wb**

Visit <https://accel-sim.github.io/> to read more about GPGPU-SIM



[View on Github](#) build passing

Accel-Sim v1.2.0 and AccelWattch v1.0 have officially been released!

Accel-Sim is a simulation framework for simulating and validating programmable accelerators like GPUs. For full details, please see our recent [ISCA 2020 paper](#) and download slides from [here](#).

AccelWattch is a power modeling framework that is extensively validated for modern GPUs and enables reliable design space exploration. Please see our recent [MICRO 2021 paper](#), download slides from [here](#), and look at AccelWattch website [here](#).

To keep you up-to-date with the recent news on Accel-Sim and AccelWattch, please join our Google group [here](#)!

If you use GPGPU-Sim 4.x, trace-driven simulation, or any of the Accel-Sim components in your research, please cite:

- Mahmoud Khairy, Jason Shen, Tor M. Aamodt, and Timothy G. Rogers "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling", In The 47th International Symposium on Computer Architecture, May 2020

If you use any component of the AccelWattch power modeling framework in your research, please cite:

- Vijay Kandiah, Scott Peverelle, Mahmoud Khairy, Junrui Pan, Amogh Manjunath, Timothy G. Rogers, Tor M. Aamodt, and Nikos Hardavellas "AccelWattch: A Power Modeling Framework for Modern GPUs", In MICRO- 54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, (MICRO '21), October 18–22, 2021, Virtual Event, Greece.