计算架构性能建模工具及架构对比研究

项目详细设计说明书

文档修订记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 完成人 | 修改内容 | 审核 | 时间 |
| V |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1. 编写目的 4](#_Toc105681795)

[1.1. 适用范围 5](#_Toc105681796)

[1.1.1. 软件名称 5](#_Toc105681797)

[1.1.2. 软件功能 5](#_Toc105681798)

[1.1.3. 软件应用 5](#_Toc105681799)

[1.2. 定义及关键词 5](#_Toc105681800)

[2.配置信息 6](#_Toc105681801)

[2.1 GPU架构信息 6](#_Toc105681802)

[2.1.1 简介 6](#_Toc105681803)

[2.1.2 详细设计 6](#_Toc105681804)

[2.2 GPU指令时延 6](#_Toc105681805)

[2.2.1 简介 6](#_Toc105681806)

[2.2.2 详细设计 6](#_Toc105681807)

[2.2.3 方法 7](#_Toc105681808)

[2.2.4 NVIDIA Turing 架构指令延迟 8](#_Toc105681809)

[3. Trace收集 13](#_Toc105681810)

表目录

表1. 关键词 4

表2. 类XXX的属性 5

表3. 方法描述xxx-function 6

图目录

图1 类图xxx-class 5

图2 流程图xxx-function 6

# 1. 编写目的

GPU可以支撑大量数据的并行计算，拥有强大的浮点运算能力，良好的可编程性和通用性，在数据处理、高性能计算、人工智能等领域得到了广泛应用。以CUDA和 OpenCL为代表的编程模型的产生也使人们能够更好地利用 GPU上大量的计算资源以及存储带宽资源，加速深度学习、高性能计算等任务。

在过去很长的一段时间里，芯片性能的提升都主要依赖于制程工艺的提升，但近些年来摩尔定律逐渐趋近它的发展极限，但GPU依赖于架构的不断迭代，其性能依然保持着较高速度的增长。近年来，随着深度学习和高性能计算领域 GPU应用规模的不断增加，计算需求爆炸性增长，对GPU 性能的要求不断提升，GPU架构快速更新迭代，对新的GPU架构进行探索也成为研究热点。为了促进GPU架构的探索与创新，通常采用高级语言对GPU架构进行建模，评估不同架构下GPU的性能，从而探索设计空间。这种高层次建模方法被广泛应用于CPU、GPU等处理器的体系结构研发过程，例如CPU建模工具gem5，GPU建模工具GPGPU-Sim等。

当前的GPU性能建模主要分为3种：周期精确模型，事件驱动模型，分析模型。当前周期精确模型由于涉及对硬件操作过程的详细仿真，因此仿真速度慢，当在设计过程中对最优设计参数搜索时，仿真时间甚至可以持续上千小时，限制了其性能建模的实用性。事件驱动模型不需要在时间上进行循环，每个周期去更新GPU所有组件的状态，状态更新由事件触发，相比于周期精确模型，避免了不必要的状态更新，建模速度更快。分析模型主要依赖于分析计算架构与性能相关的关键参数，建立数学模型，计算GPU性能指标。相比于周期精确建模，分析模型可以有效地提升设计空间搜索速度。然而分析模型精度误差较大，尤其是在处理存在资源争用的访存行为时，数学模型估计出的时间与真实硬件时间有较大差距，因此对于实际生产应用，参考价值没有周期精确模拟器高。

针对现有的GPU性能模拟器的不足，本文档设计了一种兼顾速度与精度，具有灵活性和可拓展性的GPU性能模拟器， 减少GPU微体系架构探索工作的复杂度和工作量，对GPU架构探索具有促进作用。

## 适用范围

### 软件名称

Flex-GPUsim

### 软件功能

1.兼顾速度与精度

针对现有模拟器难以兼顾速度与精度的问题，本文档设计了一种基于事件驱动的GPU性能模拟器，采用适当的抽象，将GPU中对于性能指标影响大的模块采用更细致的建模，对于性能指标影响小的模块采用简单建模，从而兼顾速度与精度。

2.模拟架构灵活性

采用模块化方法，提升模型灵活性，提供用户自定义接口，用户可根据自身需求对模型进行进一步扩展，如用户可将2级缓存架构修改为3级缓存架构。在NVIDIA架构的基础上，可以灵活地拓展到其他GPU架构上。

3.模拟精度灵活性

针对GPU同一组件，对于不同的需求，可以采用不同的建模方式。例如cache模块，在探索cache架构时，对于cache采用精确建模；在对于其他模块进行建模时，cache可以采用较为简单的分析模型建模。

### 软件应用

本软件应用于对GPU架构进行建模，评估不同架构下GPU的性能，从而探索设计空间。

## 定义及关键词

1. 关键词

| 关键词 | 英文全名 | 中文解释 |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 2.配置信息

## 2.1 GPU架构信息

### 2.1.1 简介

GPU架构信息收集模块，收集GPU的硬件信息，从而构建GPU模型。

### 2.1.2 详细设计

表2 GPU硬件信息

|  |  |  |  |
| --- | --- | --- | --- |
| 属性变量名称 | 属性中文名称 | 类型 | 说明（对属性的简短描述） |
| num\_sms | 流多处理器个数 | 非负整型 | GPU中流多处理器个数 |
| num of hardware units | 流水线硬件单元个数 | 非负整型 | GPU硬件单元，例如INT\_units，DP\_units |
| clock\_speed | GPU时钟频率 | 非负整型 | GPU中核时钟频率 |
| num\_warp\_schedulers | Warp调度器个数 | 非负整型 | SM中warp调度器个数 |
| num\_inst\_dispatch\_units | 指令发射单元个数 | 非负整型 | SM中指令发射单元个数 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

表3 类gpu\_config的属性

|  |  |  |  |
| --- | --- | --- | --- |
| 可见性 | 属性名称 | 类型 | 说明（对属性的简短描述） |
| Public | m\_gpu\_config | map | GPU配置信息 |
| Public | gpu\_isa\_latency | map | GPU指令集延迟 |

表4 read\_config方法描述

|  |  |
| --- | --- |
| 函数原型 | static void read\_config(const std::string& file\_s,std::map <std::string, std::string> &config); |
| 功能描述 | 读取配置信息 |
| 被调函数 | 无 |
| 主调函数 | void read\_m\_config() |
| 输入参数 | file\_s：配置文件路径 &config：读取的配置 |
| 输出参数 | file\_s：配置文件路径 &config：读取的配置 |
| 返回值 | 无 |

表5 config\_split方法描述

|  |  |
| --- | --- |
| 函数原型 | static void config\_split(const std::string &str, const std::string& pattern, std::string &str1, std::string &str2); |
| 功能描述 | 分解配置信息 |
| 被调函数 | 无 |
| 主调函数 | 无 |
| 输入参数 | str：需要分解的字符串 pattern：匹配模式 str1，str2：分解后的配置信息 |
| 输出参数 | str1，str2：分解后的配置信息 |
| 返回值 | 无 |

表6 read\_m\_config方法描述

|  |  |
| --- | --- |
| 函数原型 | static void config\_split(const std::string &str, const std::string& pattern, std::string &str1, std::string &str2); |
| 功能描述 | 分解配置信息 |
| 被调函数 | 无 |
| 主调函数 | 无 |
| 输入参数 | str：需要分解的字符串 pattern：匹配模式 str1，str2：分解后的配置信息 |
| 输出参数 | str1，str2：分解后的配置信息 |
| 返回值 | 无 |

## 2.2 GPU指令时延

### 2.2.1 简介

GPU指令时延是通过micro-benchmark测得的GPU中各个指令的延迟。

### 2.2.2 详细设计

测量原理：如图1所示，NVIDIA指令中包含指令信息和控制代码。其中控制代码有如图所示的6个域。其中Stall cycles表示指令发射器在发射指令前应该等待的周期数。

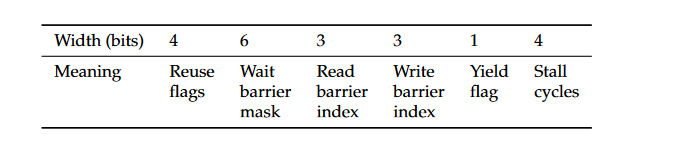


图1 NVIDIA Turing指令中控制指令位数及其含义

如果需要测量指令A的延迟。

如果A的延迟是固定的，则另A的下一条指令B依赖A，并不断减少A的stall cycles，直至B的值为错误的，则最后一次B指令正确执行时的stall cycles即为A指令的时延

如果A的延迟是不固定的，例如访存指令，则另A的下一条指令B依赖A，并设置A指令的Wait barrier mask值，计算指令执行前后的时钟差，再减去B指令时延，即得到A指令的时延。

### 2.2.3 方法

测量代码分为host端和device端，其中host端负责开辟内存，准备参数，加载kernel函数，并将kernel执行完后的数据从显存复制到CPU的内存后，输出。



图2 NVIDIA Turing架构指令时延测量host端代码

Device端代码是由SASS指令编写，然后用TuringAs工具编译为cubin文件，并动态加载到GPU上运行。一个具体的SASS指令示例如图3所示。

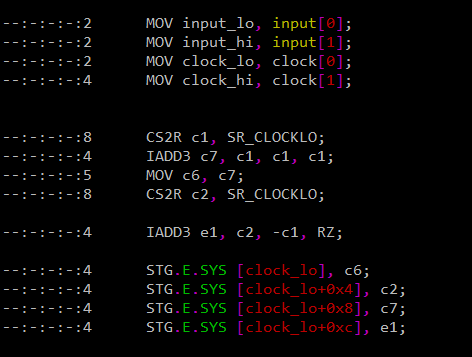


图3 IADD3指令延迟测试device端代码

图3中，IADD3指令执行加法后将结果写入c7寄存器，其后续MOV指令的源寄存器为c7，因此MOV指令依赖于IADD3指令，将IADD3指令的stall cycles改为4，MOV指令执行后的结果符合预期，将IADD3指令的stall cycles改为3后，MOV指令执行时c7寄存器尚没有写入。因此IADD3指令的时延为4cycles

### 2.2.4 NVIDIA Turing 架构指令延迟

本节依据上述测试方法，采用micro-benchmark测得NVIDIA turing架构中SASS指令延迟如表3所示。

表3 NVIDIA Turing架构指令延迟

|  |  |  |
| --- | --- | --- |
| **Opcode** | **Description** | **latency** |
| **Floating Point Instructions** | | |
| FADD | FP32 Add | 5 |
| FADD32I | FP32 Add |  |
| FCHK | Floating-point Range Check |  |
| FFMA32I | FP32 Fused Multiply and Add |  |
| FFMA | FP32 Fused Multiply and Add | 5 |
| FMNMX | FP32 Minimum/Maximum |  |
| FMUL | FP32 Multiply |  |
| FMUL32I | FP32 Multiply |  |
| FSEL | Floating Point Select |  |
| FSET | FP32 Compare And Set |  |
| FSETP | FP32 Compare And Set Predicate |  |
| FSWZADD | FP32 Swizzle Add |  |
| MUFU | FP32 Multi Function Operation |  |
| HADD2 | FP16 Add |  |
| HADD2\_32I | FP16 Add |  |
| HFMA2 | FP16 Fused Mutiply Add |  |
| HFMA2\_32I | FP16 Fused Mutiply Add |  |
| HMMA | Matrix Multiply and Accumulate |  |
| HMUL2 | FP16 Multiply |  |
| HMUL2\_32I | FP16 Multiply |  |
| HSET2 | FP16 Compare And Set |  |
| HSETP2 | FP16 Compare And Set Predicate |  |
| DADD | FP64 Add |  |
| DFMA | FP64 Fused Mutiply Add |  |
| DMUL | FP64 Multiply |  |
| DSETP | FP64 Compare And Set Predicate |  |
| **Integer Instructions** | | |
| BMMA | Bit Matrix Multiply and Accumulate |  |
| BMSK | Bitfield Mask |  |
| BREV | Bit Reverse |  |
| FLO | Find Leading One |  |
| IABS | Integer Absolute Value |  |
| IADD | Integer Addition |  |
| IADD3 | 3-input Integer Addition | 4 |
| IADD32I | Integer Addition |  |
| IDP | Integer Dot Product and Accumulate |  |
| IDP4A | Integer Dot Product and Accumulate |  |
| IMAD | Integer Multiply And Add | 5 |
| IMMA | Integer Matrix Multiply and Accumulate |  |
| IMNMX | Integer Minimum/Maximum |  |
| IMUL | Integer Multiply |  |
| IMUL32I | Integer Multiply |  |
| ISCADD | Scaled Integer Addition |  |
| ISCADD32I | Scaled Integer Addition |  |
| ISETP | Integer Compare And Set Predicate |  |
| LEA | LOAD Effective Address |  |
| LOP | Logic Operation |  |
| LOP3 | Logic Operation |  |
| LOP32I | Logic Operation |  |
| POPC | Population count |  |
| SHF | Funnel Shift |  |
| SHL | Shift Left |  |
| SHR | Shift Right |  |
| VABSDIFF | Absolute Difference |  |
| VABSDIFF4 | Absolute Difference |  |
| **Conversion Instructions** | | |
| F2F | Floating Point To Floating Point Conversion |  |
| F2I | Floating Point To Integer Conversion |  |
| I2F | Integer To Floating Point Conversion |  |
| I2I | Integer To Integer Conversion |  |
| I2IP | Integer To Integer Conversion and Packing |  |
| FRND | Round To Integer |  |
| **Movement Instructions** | | |
| MOV | Move | 4 |
| MOV32I | Move |  |
| MOVM | Move Matrix with Transposition or Expansion |  |
| PRMT | Permute Register Pair |  |
| SEL | Select Source with Predicate |  |
| SGXT | Sign Extend |  |
| SHFL | Warp Wide Register Shuffle |  |
| **Predicate Instructions** | | |
| PLOP3 | Predicate Logic Operation |  |
| PSETP | Combine Predicates and Set Predicate |  |
| P2R | Move Predicate Register To Register |  |
| R2P | Move Register To Predicate Register |  |
| **Load/Store Instructions** | | |
| LD | Load from generic Memory |  |
| LDC | Load Constant |  |
| LDG | Load from Global Memory |  |
| LDL | Load within Local Memory Window |  |
| LDS | Load within Shared Memory Window |  |
| LDSM | Load Matrix from Shared Memory with Element Size Expansion |  |
| ST | Store to Generic Memory |  |
| STG | Store to Global Memory |  |
| STL | Store within Local or Shared Window |  |
| STS | Store within Local or Shared Window |  |
| MATCH | Match Register Values Across Thread Group |  |
| QSPC | Query Space |  |
| ATOM | Atomic Operation on Generic Memory |  |
| ATOMS | Atomic Operation on Shared Memory |  |
| ATOMG | Atomic Operation on Global Memory |  |
| RED | Reduction Operation on Generic Memory |  |
| CCTL | Cache Control |  |
| CCTLL | Cache Control |  |
| ERRBAR | Error Barrier |  |
| MEMBAR | Memory Barrier |  |
| CCTLT | Texture Cache Control |  |
| **Uniform Datapath Instructions** | | |
| R2UR | Move from Vector Register to a Uniform Register |  |
| S2UR | Move Special Register to Uniform Register |  |
| UBMSK | Uniform Bitfield Mask |  |
| UBREV | Uniform Bit Reverse |  |
| UCLEA | Load Effective Address for a Constant |  |
| UFLO | Uniform Find Leading One |  |
| UIADD3 | Uniform Integer Addition |  |
| UIADD3.64 | Uniform Integer Addition |  |
| UIMAD | Uniform Integer Multiplication |  |
| UISETP | Integer Compare and Set Uniform Predicate |  |
| ULDC | Load from Constant Memory into a Uniform Register |  |
| ULEA | Uniform Load Effective Address |  |
| ULOP | Logic Operation |  |
| ULOP3 | Logic Operation |  |
| ULOP32I | Logic Operation |  |
| UMOV | Uniform Move |  |
| UP2UR | Uniform Predicate to Uniform Register |  |
| UPLOP3 | Uniform Predicate Logic Operation |  |
| UPOPC | Uniform Population Count |  |
| UPRMT | Uniform Byte Permute |  |
| UPSETP | Uniform Predicate Logic Operation |  |
| UR2UP | Uniform Register to Uniform Predicate |  |
| USEL | Uniform Select |  |
| USGXT | Uniform Sign Extend |  |
| USHF | Uniform Funnel Shift |  |
| USHL | Uniform Left Shift |  |
| USHR | Uniform Right Shift |  |
| VOTEU | Voting across SIMD Thread Group with Results in Uniform Destination |  |
| **Texture Instructions** | | |
| TEX | Texture Fetch |  |
| TLD | Texture Load |  |
| TLD4 | Texture Load 4 |  |
| TMML | Texture MipMap Level |  |
| TXD | Texture Fetch With Derivatives |  |
| TXQ | Texture Query |  |
| **Surface Instructions** | | |
| SUATOM | Atomic Op on Surface Memory |  |
| SULD | Surface Load |  |
| SURED | Reduction Op on Surface Memory |  |
| SUST | Surface Store |  |
| **Control Instructions** | | |
| BMOV | Move Convergence Barrier State |  |
| BPT | BreakPoint/Trap |  |
| BRA | Relative Branch |  |
| BREAK | Break out of the Specified Convergence Barrier |  |
| BRX | Relative Branch Indirect |  |
| BRXU | Relative Branch with Uniform Register Based Offset |  |
| BSSY | Barrier Set Convergence Synchronization Point |  |
| BSYNC | Synchronize Threads on a Convergence Barrier |  |
| CALL | Call Function |  |
| EXIT | Exit Program |  |
| JMP | Absolute Jump |  |
| JMX | Absolute Jump Indirect |  |
| JMXU | Absolute Jump with Uniform Register Based Offset |  |
| KILL | Kill Thread |  |
| NANOSLEEP | Suspend Execution |  |
| RET | Return From Subroutine |  |
| RPCMOV | PC Register Move |  |
| RTT | Return From Trap |  |
| WARPSYNC | Synchronize Threads in Warp |  |
| YIELD | Yield Control |  |
| **Miscellaneous Instructions** | | |
| B2R | Move Barrier To Register |  |
| BAR | Barrier Synchronization |  |
| CS2R | Move Special Register to Register | 5 |
| DEPBAR | Dependency Barrier |  |
| GETLMEMBASE | Get Local Memory Base Address |  |
| LEPC | Load Effective PC |  |
| NOP | No Operation |  |
| PMTRIG | Performance Monitor Trigger |  |
| R2B | Move Register to Barrier |  |
| S2R | Move Special Register to Register |  |
| SETCTAID | Set CTA ID |  |
| SETLMEMBASE | Set Local Memory Base Address |  |
| VOTE | Vote Across SIMD Thread Group |  |

# 3. Trace收集

本软件是基于trace的GPU性能建模工具。其中trace包含benchmark运行时执行指令trace和访存信息trace。

对于NVIDIA架构，本设计采用动态二进制工具NVBit来收集访存trace和指令trace，NVBit是一个快速、动态、可移植的二进制工具框架，它允许用户使用CUDA/C/C++编写工具，可以有选择性地应用于预先编译好可执行的二进制文件和库中。表4是收集trace的具体信息。

表4 采用NVBit工具收集的信息

|  |  |  |  |
| --- | --- | --- | --- |
| trace种类 | 收集范围 | 字段 | 含义 |
| SASS指令 | 单个Block | warp\_id | warp编号 |
| PC | 程序计数器 |
| opcode | 操作码 |
| reg\_dests | 目的寄存器 |
| reg\_srcs | 源寄存器 |
| 访存指令 | 所有Block | block id | block编号 |
| warp id | warp编号 |
| PC | 程序计数器 |
| opcode | 操作码 |
| mem\_addresses | 访存地址 |
| kernel信息 | 单个kernel | kernel\_id | kernel编号 |
| grid\_size | grid大小 |
| block\_size | block大小 |
| shared\_mem\_bytes | 共享内存 |
| num\_registers | 寄存器个数 |

# 4. Trace解析

## 4.1 SASS trace解析

### 4.1.1 简介

本模块在trace\_reader.cpp中，定义了kernel\_info和trace\_inst两个类，用于trace解析

### 4.1.2 属性与方法

表3 类kernel\_info的属性

|  |  |  |  |
| --- | --- | --- | --- |
| 可见性 | 属性名称 | 类型 | 说明（对属性的简短描述） |
| Public | m\_kernel\_id | unsigned | Kernel id |
| Public | m\_grid\_size | unsigned | Grid大小 |
| Public | m\_block\_size | unsigned | 线程块大小 |
| Public | m\_num\_registers | unsigned | 寄存器数目 |
| Public | m\_shared\_mem\_bytes | unsigned | 共享内存字节数 |

表3 类trace\_inst的属性

|  |  |  |  |
| --- | --- | --- | --- |
| 可见性 | 属性名称 | 类型 | 说明（对属性的简短描述） |
| Public | line\_id | unsigned | 指令在warp中的绝对位置 |
| Public | warp\_id | int | Warp编号 |
| Public | pc | string | 程序计数器 |
| Public | dest\_num | int | 目的寄存器数目 |
| Public | dest\_regs | vector<std::string> | 目的寄存器 |
| Public | opcode | std::string | 操作码 |
| Public | src\_num | int | 源寄存器数目 |
| Public | src\_regs | std::vector<std::string> | 源寄存器 |

表6 trace\_inst方法描述

|  |  |
| --- | --- |
| 函数原型 | explicit trace\_inst(std::string &line,unsigned line\_id); |
| 功能描述 | 构造trace\_inst对象 |
| 被调函数 | 无 |
| 主调函数 | int main() |
| 输入参数 | line：trace需要解析的一行 line\_id:行号 |
| 输出参数 | line：trace需要解析的一行 |
| 返回值 | 无 |

表3 类trace\_reader的属性

|  |  |  |  |
| --- | --- | --- | --- |
| 可见性 | 属性名称 | 类型 | 说明（对属性的简短描述） |
| Private | m\_kernel\_id | int | 内核编号 |

表6 read\_sass方法描述

|  |  |
| --- | --- |
| 函数原型 | void read\_sass(const std::string& benchmark\_path,kernel\_info &kernelInfo,std::vector<trace\_inst> &trace\_insts); |
| 功能描述 | 读取sass trace |
| 被调函数 | 无 |
| 主调函数 | int main() |
| 输入参数 | Benchmark\_path：benchmark路径 kernelInfo：kernel信息 trace\_insts：trace指令 |
| 输出参数 | kernelInfo：kernel信息 trace\_insts：trace指令 |
| 返回值 | 无 |