Parallel Processing

Lecture: **性能评估与模拟**

邵恩
高性能计算机研究中心

Outline

- **性能评估方法**
  - 分析模型
    - » Roofline Model
  - 模拟
    - » 完整系统与用户级别
    - » 跟踪/事件/执行驱动
      - » 事件队列?
    - » 循环精度

Roofline Model

- **目标：将内核函数的并行算法性能、内存带宽和局部性集成到一个易于理解的性能图中**
- **此外，必须以图形方式显示出，再剔除某些软件优化后，性能的下降**

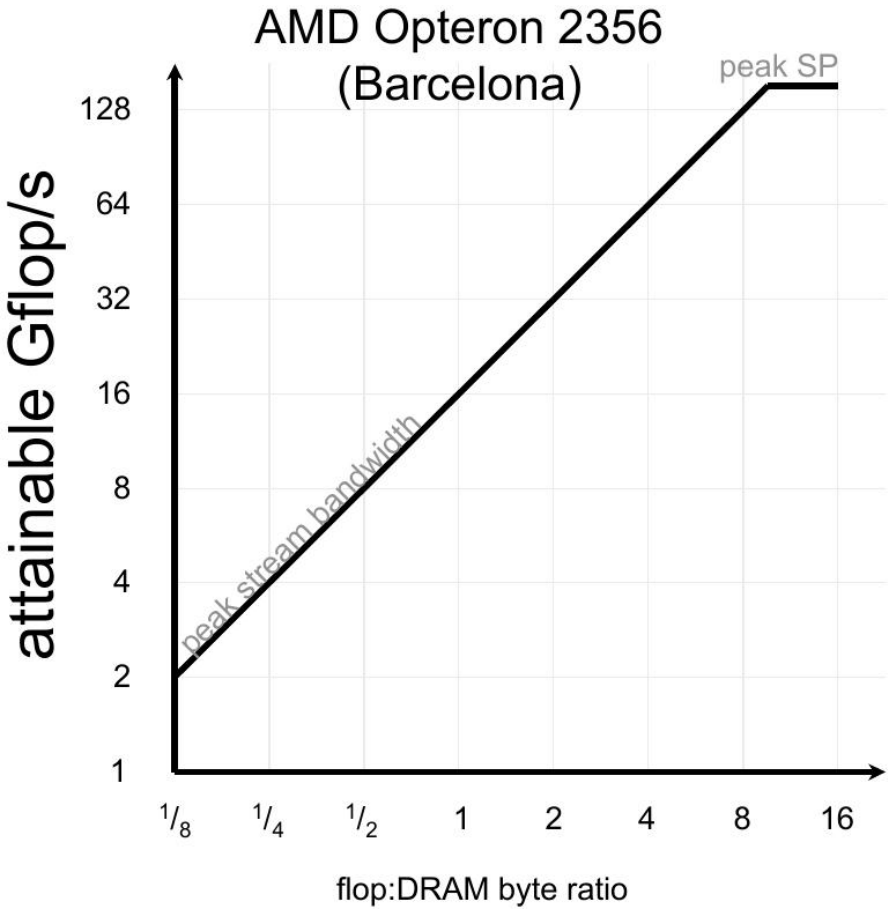
- **Roofline Model对于每种芯片的体系结构，都是独一无二的。**
  - 不同体系结构的Roofline Model是不同的，不可复用
- **对于每种芯片的体系结构，同一个核函数（确定性的并行算子）在Roofline Model图的坐标是唯一的**

Roofline Model

- **核函数的计算性能的受限因素共有三项：**
  - 因素一：芯片的理论计算峰值性能（性能上限）： peak flop rate
    - 指：**一个计算平台倾尽全力每秒钟所能完成的浮点运算数**
    - 单位是FLOPS或FLOP/s
  - 因素二：芯片的访存带宽： streaming bandwidth
    - 指：**一个芯片倾尽全力每秒所能完成的内存交换量**
    - 单位是Byte/s
  - 因素三：核函数的计算强度：actual flop:byte ratio
    - 核函数的计算量除以访存量就可以得到核函数的计算强度
    - 指：**核函数在计算过程中，每Byte内存交换到底用于进行多少次浮点运算**
    - 单位是FLOPs/Byte。

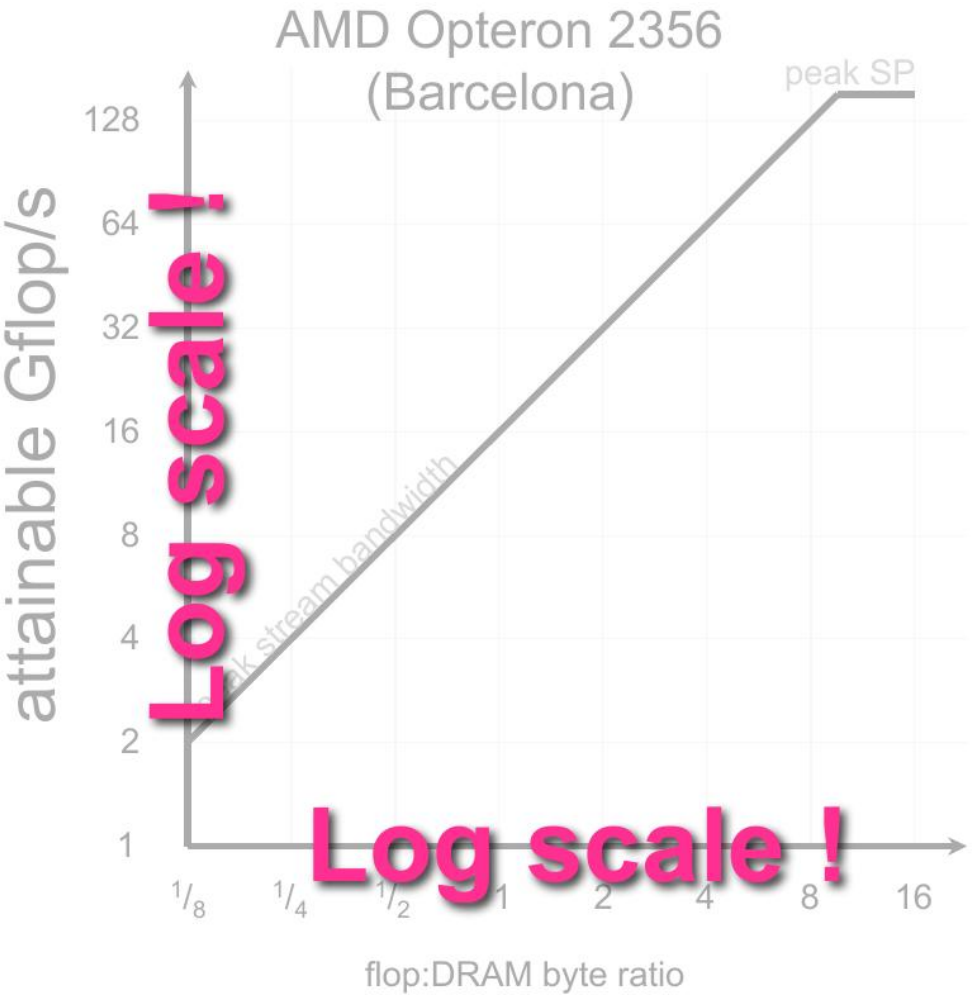
$$\text{Gflop/s} = \min \begin{cases} \text{Peak Gflop/s} \\ \text{Stream BW} * \text{actual flop:byte ratio} \end{cases}$$

Roofline Model



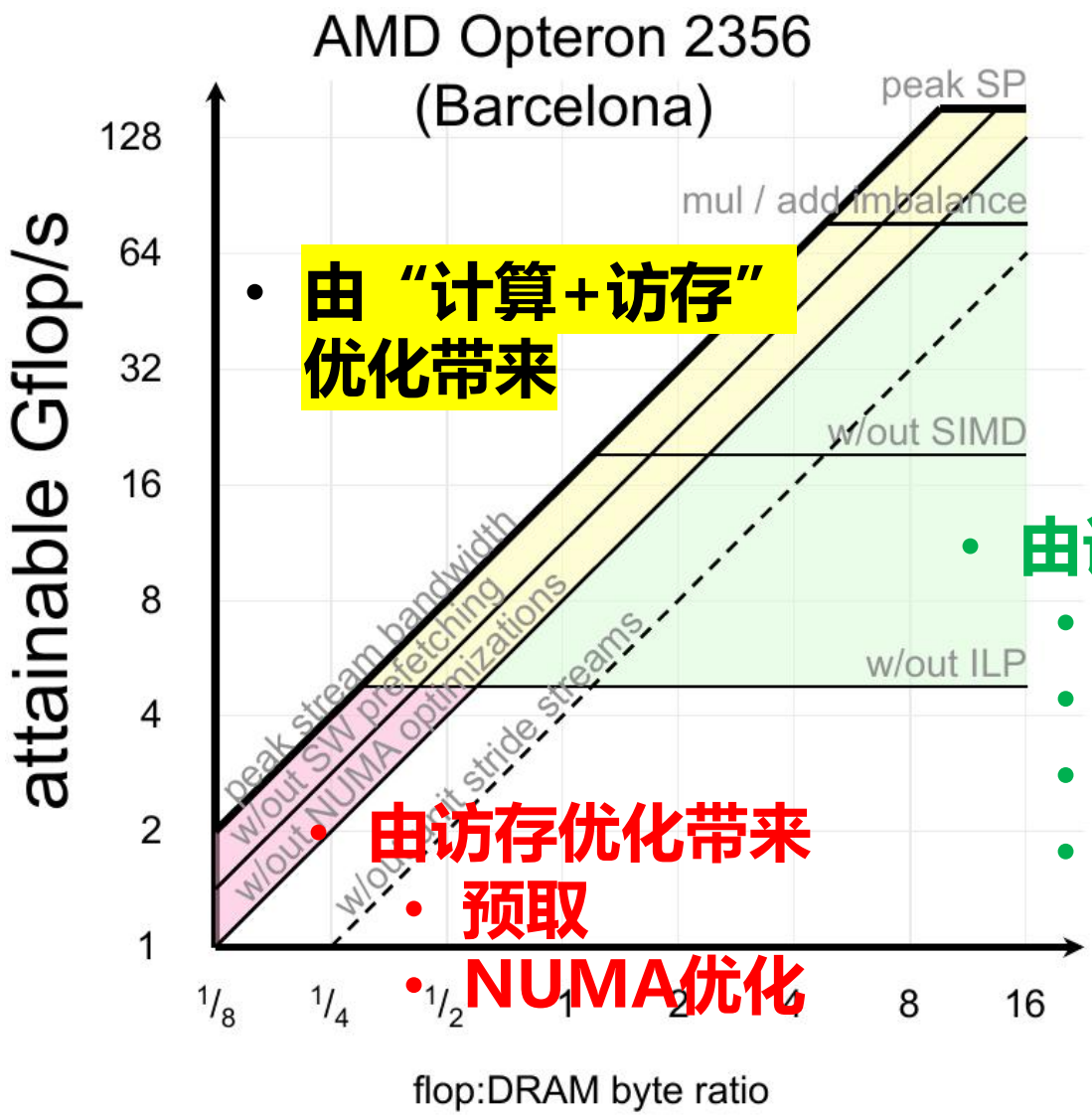
- Roofline模型的峰值性能是基于单精度峰值和流读取带宽计算得出
- 单精度峰值
  - 计算平台每秒钟所能完成的最大浮点运算数
- 流读取带宽
  - 计算平台每秒钟所能完成的最大内存交换量

Roofline Model



- Roofline模型的峰值性能是基于单精度峰值和流读取带宽计算得出
- 单精度峰值
  - 计算平台每秒钟所能完成的最大浮点运算数
- 流读取带宽
  - 计算平台每秒钟所能完成的最大内存交换量

Roofline Model



- 由“计算+访存”优化带来
- 由访存优化带来
  - 预取
  - NUMA优化

- 将预期性能区域划分为三个优化区域：
  - 仅计算
  - 仅内存
  - 计算+内存
- 由计算优化带来
  - ILP
  - SIMD
  - 负载均衡
  - 算力理论上限

模拟（Simulation）

- **模拟器（Simulator） vs 仿真器（Emulator）？**
  - 仿真（Emulation）是模仿外部**可观察到的行为**来匹配现有目标的过程（黑盒仿真）
  - 另一方面，模拟（Simulation）涉及对目标的**内部状态**进行建模。（白盒模拟）
- **模拟器设计类别**
  - 用户级别 vs 全系统
  - 功能 vs 时序
  - 仿真 vs 插桩检测
  - 追踪驱动 vs 执行驱动
  - 循环驱动 vs 事件驱动

模拟器（Simulator）设计类别

- **模拟器类别比较：** 用户级 vs 全系统级
  - 用户级
    - 仅针对单独用户，运行目标应用程序，不关心OS行为
    - 模拟基准的任何系统服务请求都会绕过用户级模拟，并由底层主机操作系统提供服务
  - 全系统级: 更加完善的模拟器建模，包括OS行为：如特权模式和模拟外围设备以支持操作系统
- **模拟器类别比较：** 功能 vs 时序
  - 功能模拟：模拟目标计算机系统的功能
  - 时序模拟：提供模拟系统的真实时序/性能

模拟器类别比较：仿真 vs 插桩检测

- **仿真（Emulation）**
  - 解码每条指令
  - 调用功能和时序模型
  - 当模拟的 ISA 与主机的不同时，自动切换为主机的指令集
  - 软件举例：gem5, Flexus, MARSS
- **插桩检测（Instrumentation）**
  - 将检测调用的插桩函数，添加到模拟程序的二进制代码中
  - 通过主机节点，实施不同的功能模拟
  - 如果时序模型的模拟速度够快，则模拟速度会很快
  - 无需功能模型
    - CISC ISA 的指令集非常复杂
  - 软件举例：CMPSim, Graphite, ZSim

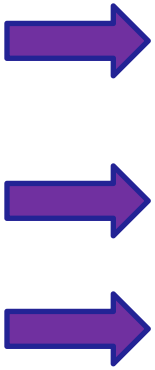
插桩检测(如何来检测?)

- 例子:
  - ZSim 使用动态二进制翻译（Pin）
  - Pin 是用于 IA-32、x86-64 和 MIC 指令集架构的动态二进制检测框架

Basic block

```
mov (%rbp),%rcx
add  %rax,%rbx
mov %rdx,(%rbp)
ja 40530a
```

插桩



Instrumented basic

```
Load(addr=(%rbp))
mov (%rbp),%rcx
add  %rax,%rdx
Store(addr=(%rbp))
mov %rdx,(%rbp)
BasicBlock(BBLDescriptor)
ja 10840530a
```

插桩检测(如何来检测?)

- 第一步：用包含大部分静态信息的BBL描述符，模拟处理器的指令活动

Instrumented basic

```
Load(addr=(%rbp))
mov (%rbp),%rcx
add  %rax,%rdx
Store(addr=(%rbp))
mov %rdx,(%rbp)
BasicBlock(BBLDescriptor)
ja 10840530a
```

将 BBL 解码为 BBL 描述符

BblDescriptor:

指令数	numInstructions = 4
访存量	numBytes = 4
微操作	uop[]

插桩检测(如何来检测?)

- **将 x86 指令解码为微操作（uops）**
  - uops 是 micro-operations 的缩写，也可以写作 μops。它们是用于实现复杂机器指令的详细的**低级指令**。
  - 具有不同的延迟、src/dst 对、功能单元端口

Type	Src1	Src2	Dst1	Dst2	Lat	PortMsk
Load	rbp		rcx			001000
Exec	rbp		rdx		3	110001
Exec	rax	rdx	rdx	rflgs	1	110001
StAddr	rbp		S0		1	000100
StData	rdx	S0				000010
Exec	rax	rip	rip	rflgs	1	000001

插桩检测(如何来检测?)

- 第二步： 用地址访问，模拟内存系统的操作行为

Instrumented basic

```
Load(addr=(%rbp))
mov (%rbp),%rcx
add  %rax,%rdx
Store(addr=(%rbp))
mov %rdx,(%rbp)
BasicBlock(BBLDescriptor)
ja 10840530a
```

```
Load(Address addr) {
    L1D->load(addr);
}
Store(Address addr) {
    L1D->Store(addr);
}
```

插桩检测(如何来检测?)

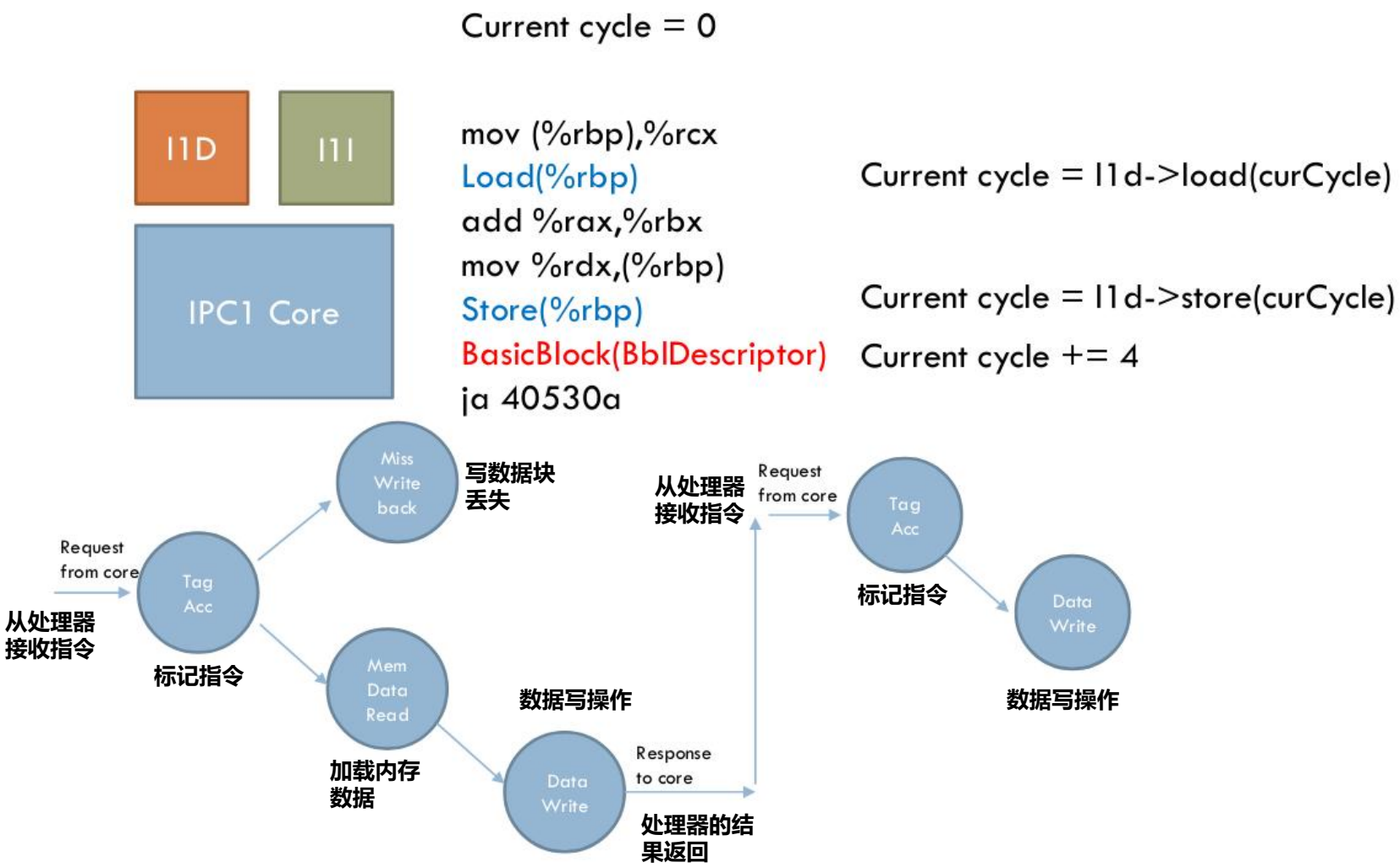
- **指令驱动的核心活动（基本块，basic block) 模拟**
  - 一次模拟单个指令的多个阶段
  - 每个阶段维护一个单独的时钟

```
BasicBlock(BblDescriptor) {
  foreach uop {
    simulateFetch(uop);
    simulateDecode(uop);
    simulateIssue(uop);
    simulateExecute(uop);
    simulateCommit(uop);
  }
}
```

```
simulateIssue(uop) {
  addUopToRob(curRobCycle,uop);
  if(rob.isFull()){
    nextRobAvailCycle = rob.advance();
  }
}
```

插桩检测(如何来检测?)

- 处理器的各个时序模拟



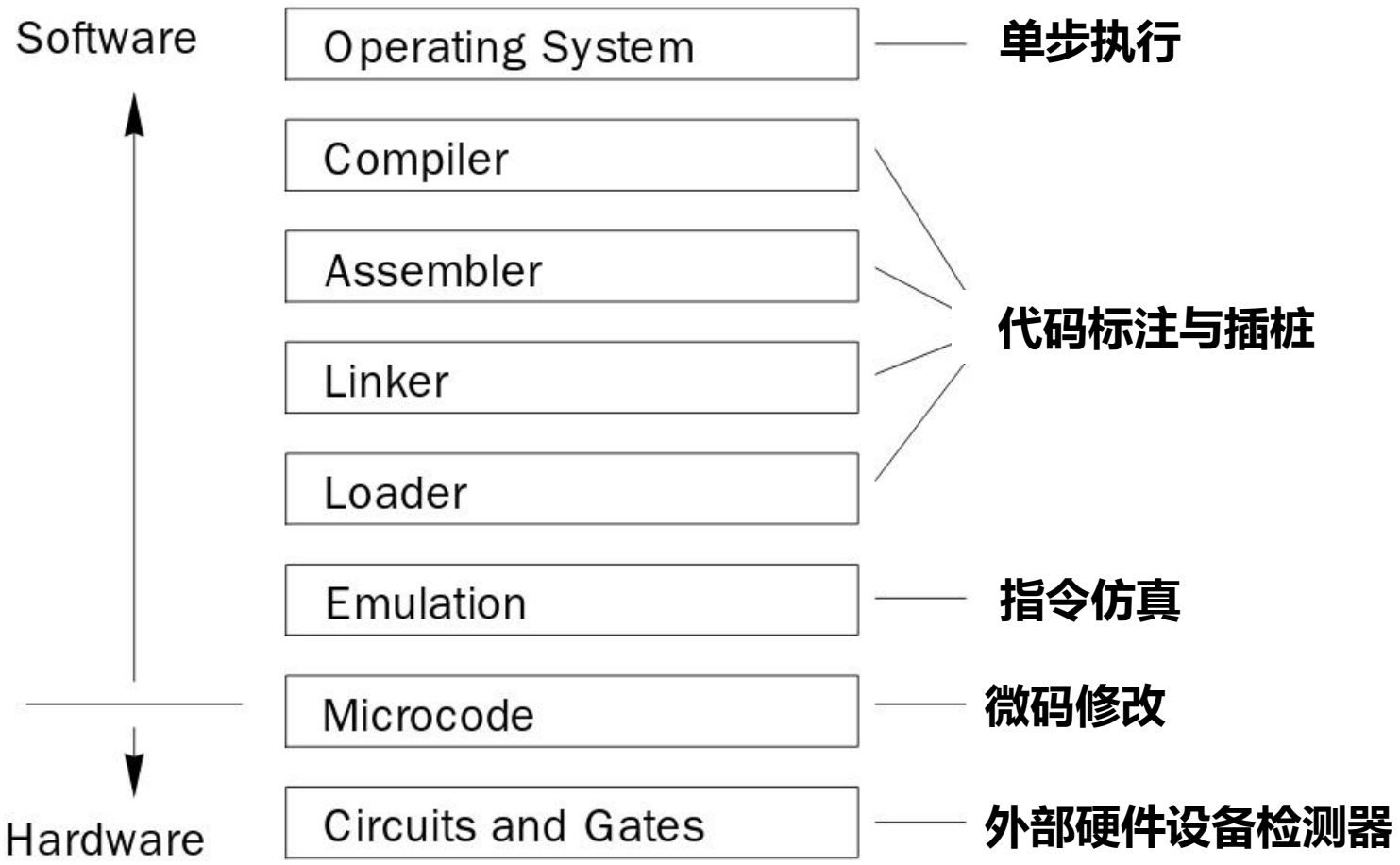
模拟器类别比较：追踪驱动 vs 执行驱动

- **追踪驱动的模拟（Trace-driven）**
  - 基于Trace-driven的模拟通过读取预先记录的指令流来进行模拟。这种方法相对快速，结果具有高度可重复性，但需要大量的存储空间
  - 使用trace 文件作为输入；生成的trace 文件可能非常大
  - 从磁盘读取如此大的文件可能会很慢
  - 不太灵活，取决于完整性、详细程度、trace 失真
  - 模拟多线程工作负载时，无法对线程间排序行为进行建模
  - 示例：CMPSim，...
- **执行驱动的模拟（Execution-driven）**
  - 基于Execution-driven的模拟，则是在模拟过程中动态读取程序并模拟机器指令的执行。这种方法可以更精确地模拟系统，但需要更详细的开发流程，并可能带来性能成本
  - 更灵活、准确、可扩展
  - 设计更复杂
  - 示例：gem5、zsim、...

追踪驱动（Trace-driven）

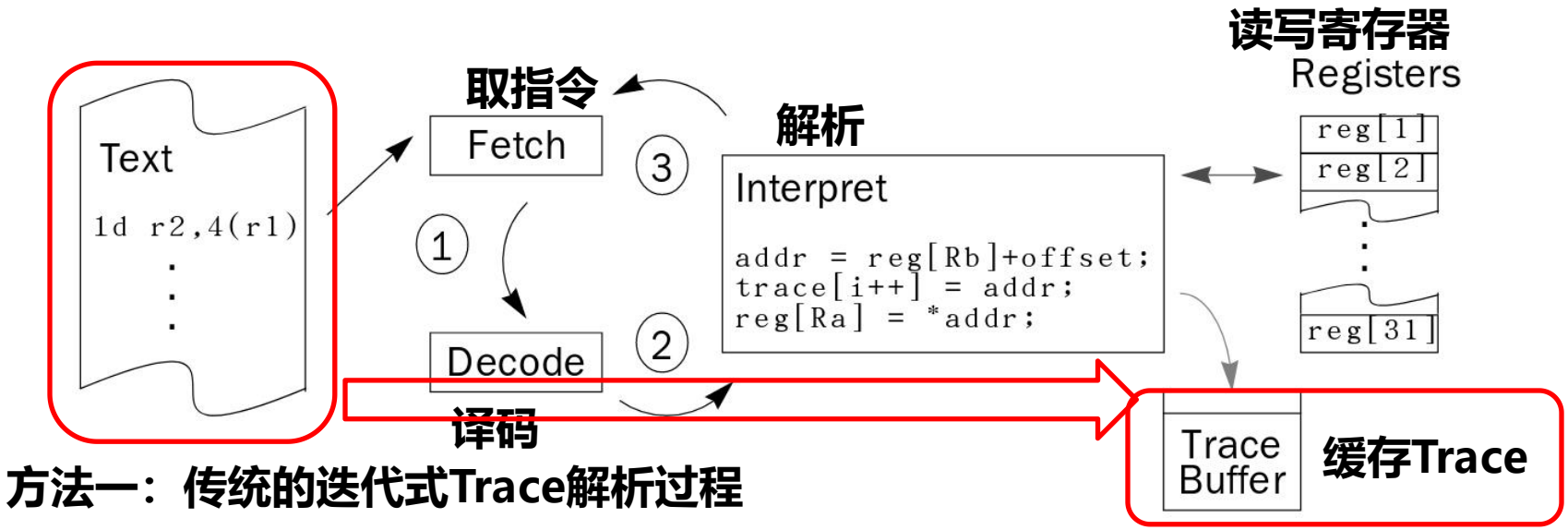
- **3个主要阶段**
  - Trace 收集
  - Trace 减少
  - Trace 处理
- **Trace 收集**
  - 确定由某应用负载所进行的内存引用（或指令），以及其确切执行顺序
- **Trace 减少**
  - 从完整跟踪中删除不需要或冗余的Trace 数据
- **Trace 处理**
  - 将Trace 数据发送给模拟系统行为的模拟程序，并开始处理

Trace 收集



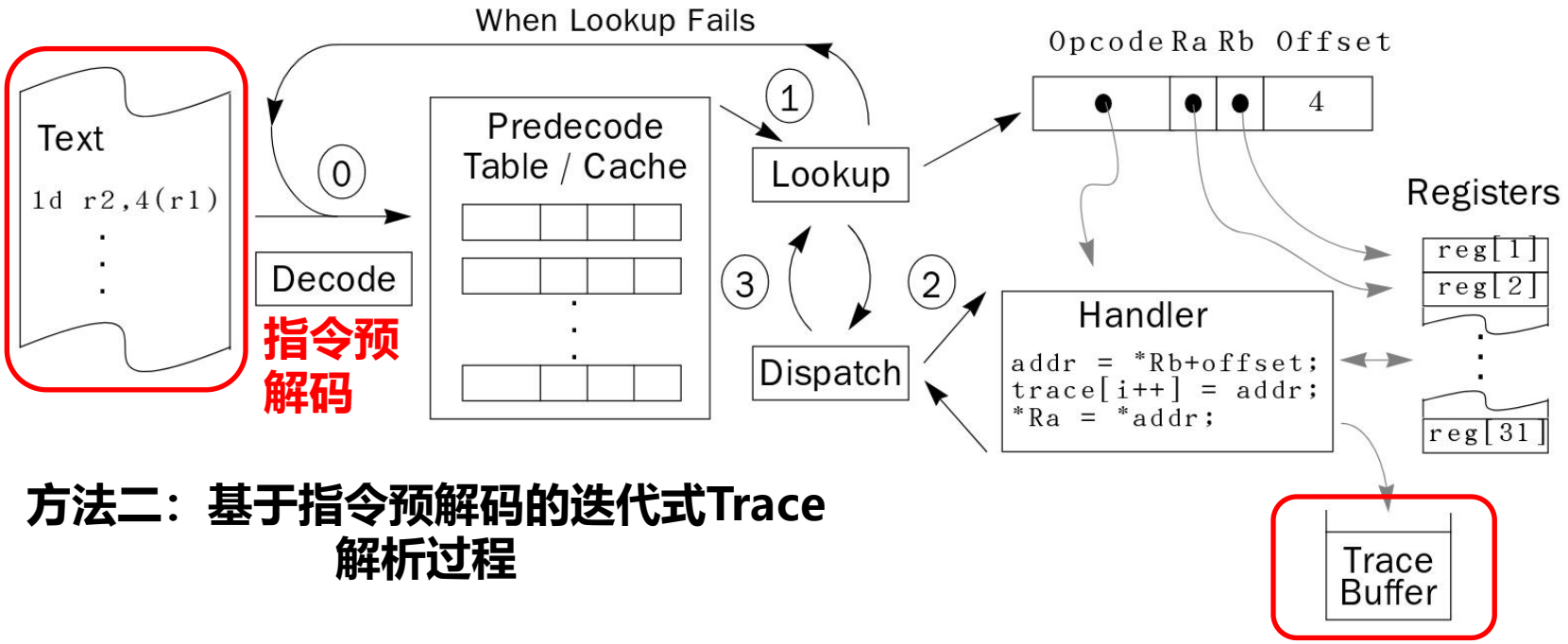
系统抽象层次和Trace 收集方法

Trace 收集——一些仿真方法



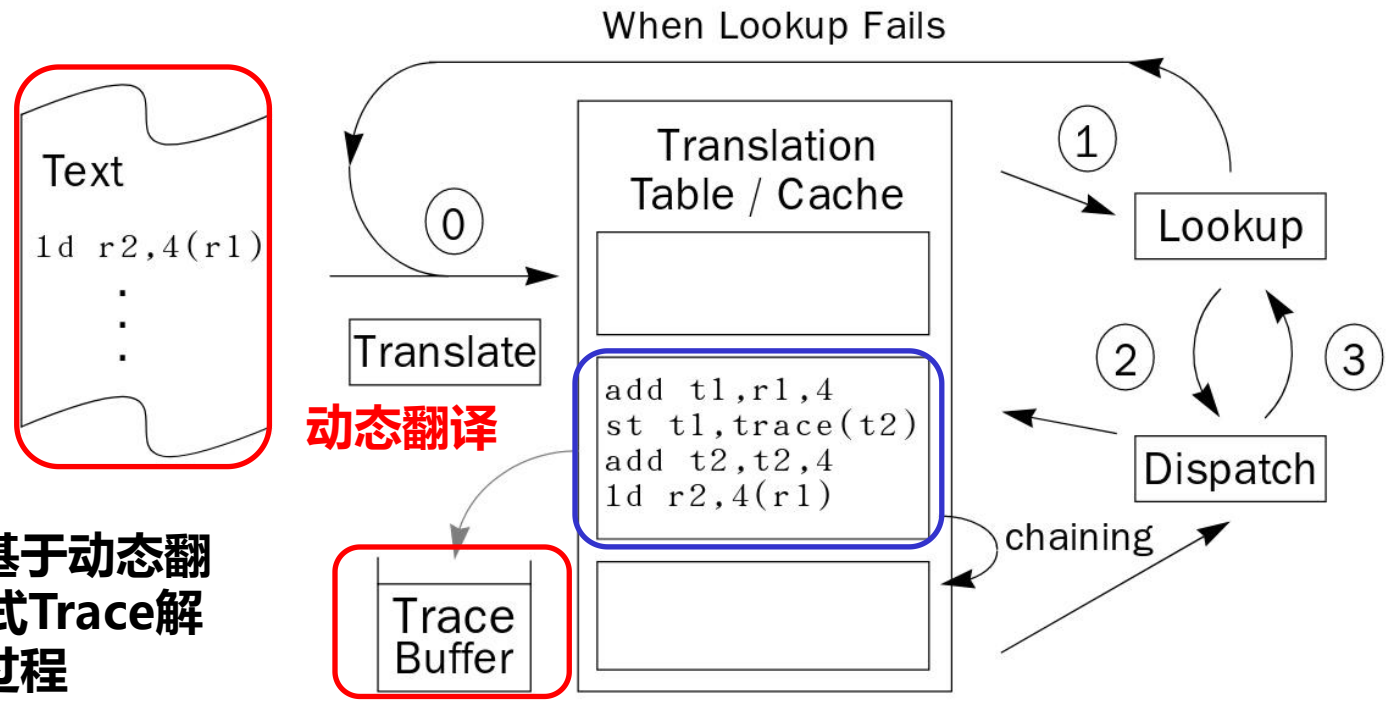
方法一：传统的迭代式Trace解析过程

Trace 收集——一些仿真方法



方法二：基于指令预解码的迭代式Trace解析过程

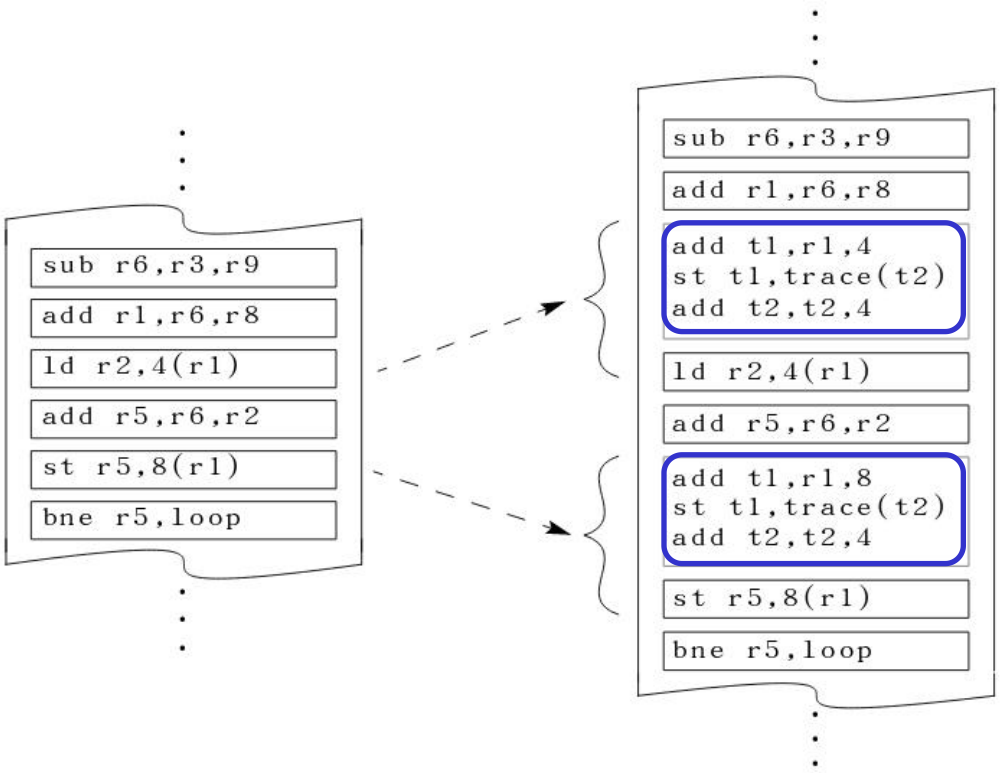
Trace 收集——一些仿真方法



方法三：基于动态翻译的迭代式Trace解析过程

Trace收集——静态代码注释的插桩

原始汇编代码、对象模块或可执行文件



带注释的可执行文件

Trace收集——降低Trace采集数据量

- **Trace的数据量很大**
- **Trace压缩**
  - 使用标准化压缩算法（例如.gz .lzma）
  - 改变trace的数据内容
    - example: 0xA0 0xA5 0xB3 0xB0 -> 0xA0 +5 0xB3 -3
- **关键事件的Traces**
  - 只需将重要事件保存在Traces文件中
  - 例子：只记录内存引用
- **Trace的采样**
  - 仅选择完整Trace数据集合的子集或样本
  - 模拟点的选取（simpoint）
    - 自动发现和利用程序阶段行为
    - 从每个阶段选择有代表性的trace数据

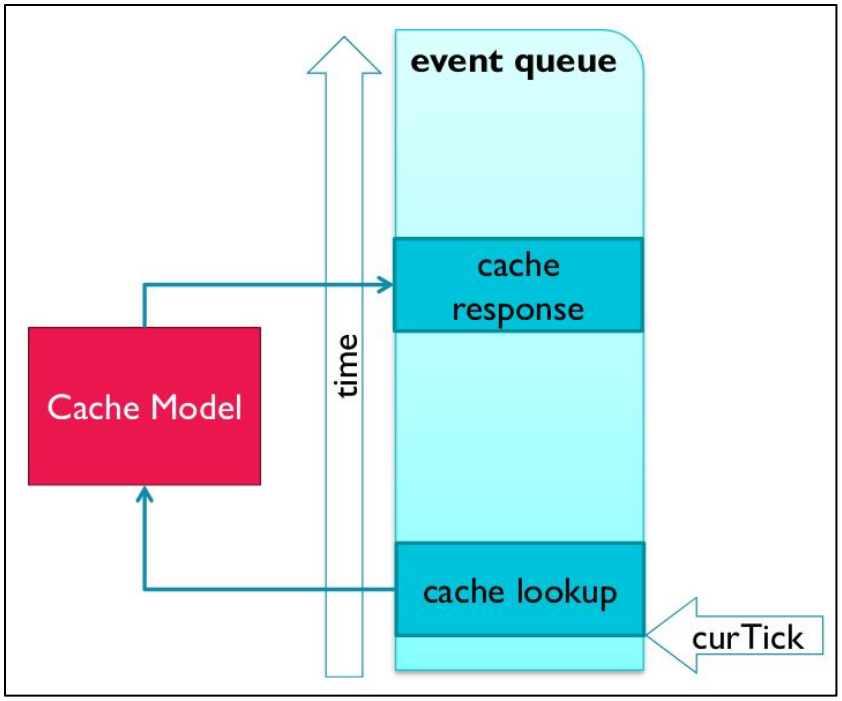
周期性模拟 VS 事件驱动模拟

- **周期性模拟（Cycle-driven）**
  - 模拟整个系统的每个时钟周期
- **事件驱动模拟（Event-driven）**
  - 由触发条件驱动
  - 及时调度事件的发生

周期性模拟（Cycle-driven

```
while (clockCycle++) {
  simulate_cpu(clockCycle);
  simulate_l1(clockCycle);
  simulate_l2(clockCycle);
}
```

事件驱动模拟（Event-driven）



全系统仿真中的误差源

A. Gutierrez et al., "Sources of error in full-system simulation," 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Monterey, CA, 2014, pp. 13-22.

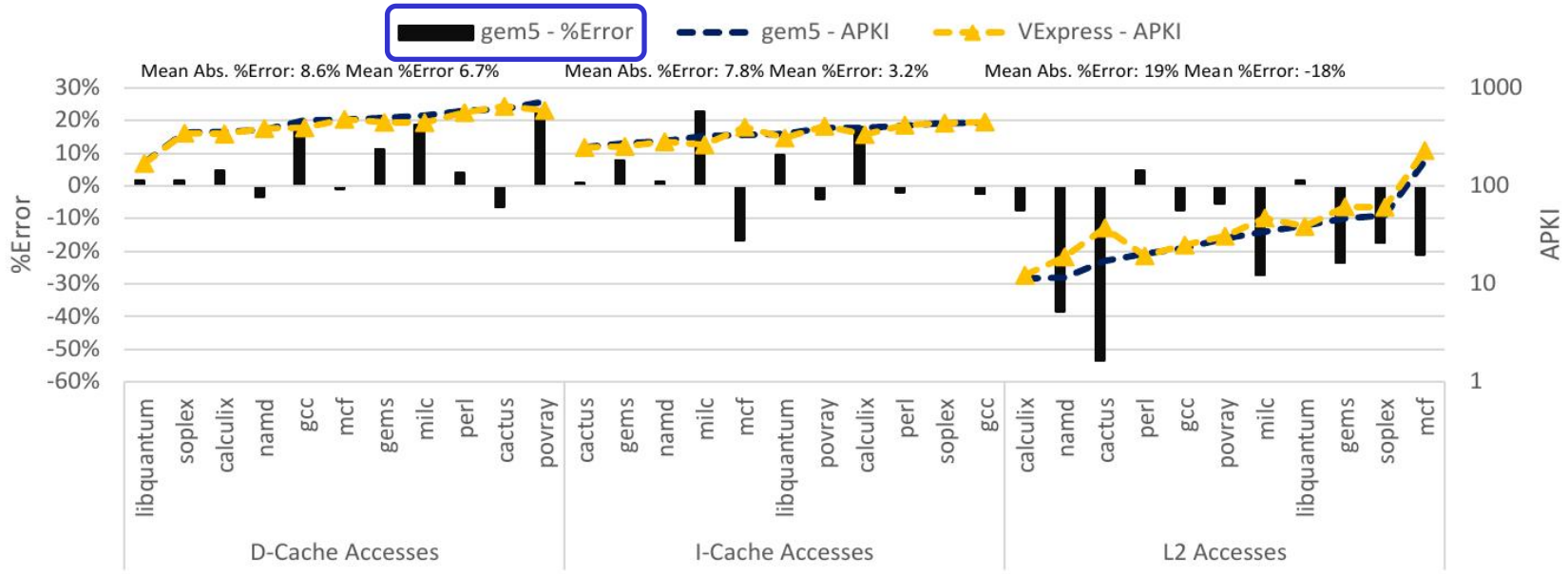


Fig. 6: Cache access stats for SPEC. The cache accesses are accurate to within 10% on average for the D-cache and I-cache. The differences are likely due to a more aggressive fetch stage, and differences in the load-store queue. The error of the L2 cache is around 20% on average, which is due to seeing different accesses coming from the L1 caches and the page table walker.

gem5 引擎只允许对单个I-cache 进行访问，而现代 CPU 是完全流水线化的，允许对指令缓存行进行多个并行访问

全系统仿真中的误差源

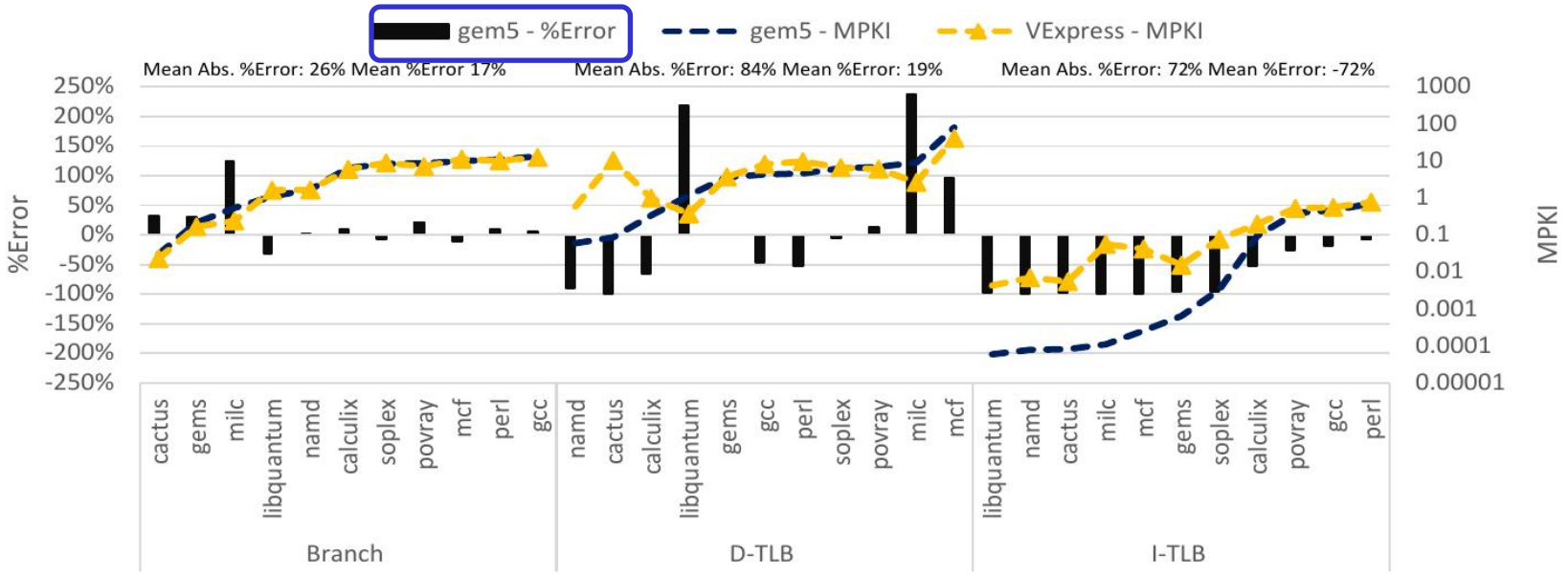
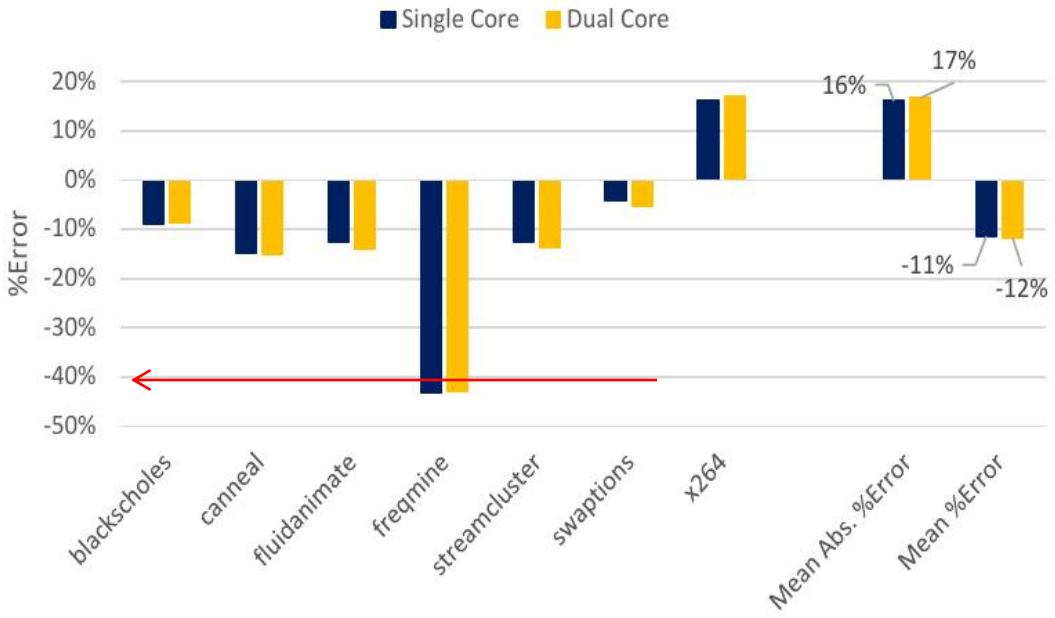


Fig. 7: Branch and TLB miss stats for SPEC. The branch misses are accurate to within 26%, and accuracy improves as the MPKI increases. Because of the extremely low I-TLB miss rates the error is 72% on average, however the accuracy improves as the MPKI increases. The D-TLB misses exhibit large error due to the fact that a single 64-entry TLB is used for loads and stores, which may benefit some workloads, and hurt others.

为了更准确地支持 TLB 建模，需要对独立的 TLB 进行读取和写入，以及二级 TLB。否则会引发误差。

全系统仿真中的误差源

- **针对全芯片体系结构的模拟器并不能作为微架构模拟器**
  - 全系统模拟器（Full-system simulators ）不能以完美的精度对每个微架构组件（microarchitectural component）进行建模
  - 除非经过验证，否则不应基于全系统模拟器来模拟获得的微体系结构统计数据
  - 对运行时间的误差在一些情况下会比较大，甚至会高达40%以上

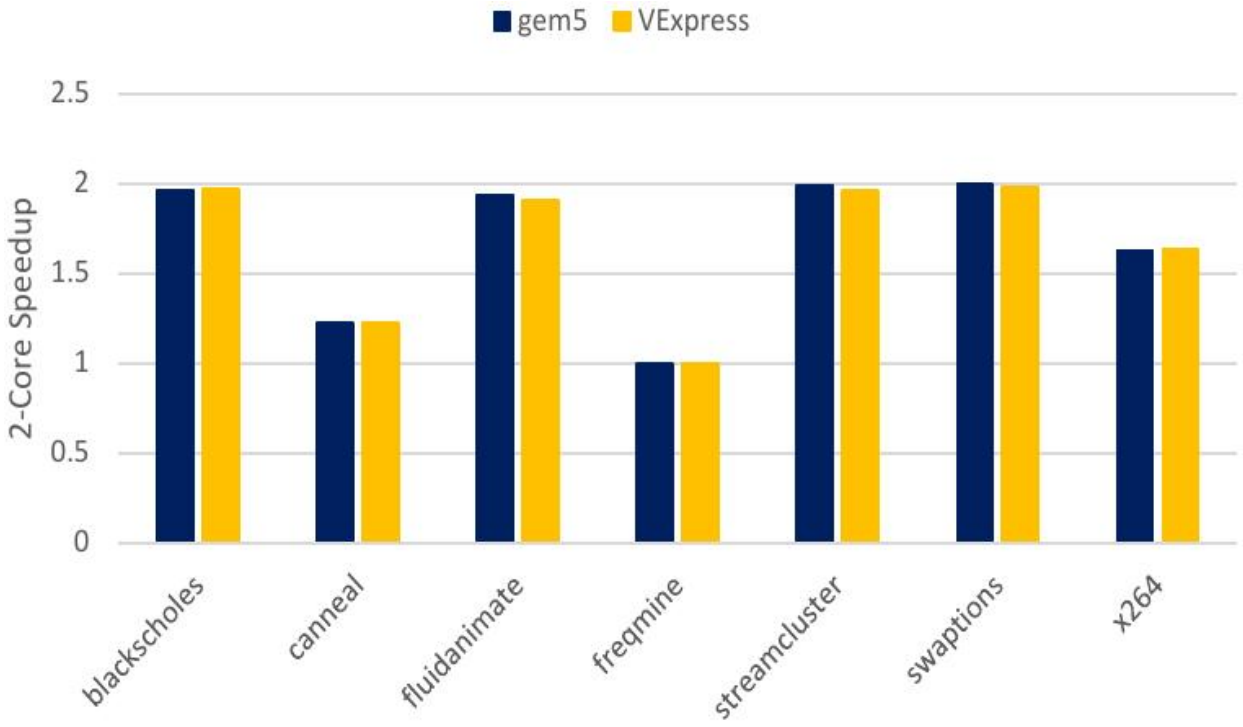


(b) PARSEC runtime accuracy.

普林斯顿共享内存计算机应用程序库（PARSEC）

全系统仿真中的误差源

- **规范和抽象错误并不意味着模拟不准确**
  - 尽管存在规范和抽象错误，仍有可能保持准确性



(a) PARSEC runtime scaling accuracy.

VExpress开发板上运行的结果和和gem5的结果很接近。结合上一页可以知道，使用模拟器需要知道哪些方面是重要的。只要使我们最希望观察到的现象是正确的就可以

全系统仿真中的误差源

- **研究人员必须决定仿真的哪些方面对他们来说很重要**
  - 要在模拟器性能、准确性和灵活性之间的权衡
- **需要努力捕获新出现的工作负载的行为**
  - 交互式工作负载现在正在模拟中使用
  - 但是现代模拟器不会对它们使用的许多硬件设备进行充分的建模

Thanks
Q&A