HSM: A Hybrid Slowdown Model for Multitasking GPUs

ASPLOS'20
Session SMIT
Xia Zhao
Ghent University

➢ 1. Abstract （当前问题+当前解决方式+本文解决方案）

- **第1段，当前GPU多作业的现状与存在的问题。在云环境下，由于负载对GPU不同资源的需求不同，因此导致单应用独占方式导致GPU利用率过低。**

  Current Problem: *However,* ***GPU-compute applications stress the GPU architecture in different ways*** *— leading to suboptimal resource utilization when a single GPU is used to run a single application.*

// 一个直观的解决方式就是让一个GPU同时执行多个作业，但是这通常会导致作业间的相互干扰，影响业务的QoS。

Current Solution: *One solution is to use the GPU in a multitasking fashion to improve utilization. Unfortunately, multitasking leads to destructive interference.*

- **第2段，作者提出的解决方案，包括此方案具体解决了什么问题，设计理念，以及最后达到的效果**

  This Paper Solution: *We propose the* ***Hybrid Slowdown Model (HSM)*** *to dynamically and accurately predict application slowdown due to interference.*

  we use HSM to guide various resource management schemes in multitask in GPUs. HSM significantly improves system throughput by 18.9%.

➢ **本文Introduction包含的内容**：

- 第1段，引出存在的问题
- 第2段，第2段列举当前是如何解决此问题的
- 第3段作者从宏观层面给出了一些相关研究
- 第4段作者仔细深入地分析了GPU多作业系统的相关研究
- 第5段引出作者提出的方法
- 第6段说明这种融合方法中，其中的白盒模型的设计思路
- 第7段说明这种融合方法中，其中的黑盒模型的设计思路
- 第8段简要说明了实验手段与测试指标，并列出测试的数据结果
- 第9段总结文章的贡献

➢ Introduction （引出问题 + 当前解决方式及缺陷 + 相关研究简述 + 本文方法 ）

- **第1段引出问题。GPU已经被广泛使用在各个领域，但是由于负载的多样性，一个单一的任务不能占满整个GPU的各类资源**

For instance, a compute-bound application will utilize the SMs well, but use relatively little memory bandwidth. Conversely, a memory-bound application will have high bandwidth utilization, but use the SMs inefficiently.

- **第2段列举当前是如何解决此问题的，并说明现在的方法有哪些问题。通过同时运行多作业来充分利用GPU的资源，由于多作业之间的作业没有资源隔离所以会产生竞争，影响业务质量。**

Unfortunately, this creates another problem: The co-running applications can interfere unpredictably with each other in the ***shared memory system****. In a GPU-enabled cloud, the main undesirable effects of interference are that it can result in (1) users being billed for resources they were unable to use* ***(fairness）****, and (2) unpredictable Quality-of-Service* ***(QoS)*** *and Service-Level Agreement (SLA) violations.*

➢ Introduction

- **第3段作者从宏观层面给出了一些相关研究，CPU在多作业执行方面的相关研究，GPU在多作业上面的一些研究与困难**

// 参考CPU的相关研究，可以使用slowdown模型来评估干扰程度，具体来说slowdown是利用作业共享的运行指标去估计作业独占的运行指标。

Thus, slowdown prediction schemes use performance models or heuristics that ***take shared mode measurements as input to predict private mode performance.***

// 为了支持计算slowdown，需要在线的profiling应用的指标，因此需要在GPU上更多硬件上的设计。

While slowdown prediction has been studied extensively for CPUs , the problem is less explored for multitasking GPUs. Since slowdown prediction is performed online, architects need to appropriately trade off the overheads of the scheme ***(i.e., the number of counters and logic complexity)*** *against the required accuracy to use* ***chip resources*** *as efficiently as possible.*

➢ Introduction

- 第4段作者仔细分析了GPU多作业系统的相关研究。说明当前研究的缺陷。

// slowdown模型分为白盒和黑盒两种。白盒是根据体系结构的固有特性来建模，但是由于GPU高度的并发与访存，其会存在很多不确定性的overlap，因此单纯的白盒模型准确度较低。

Broadly speaking, slowdown prediction models can be classified as white-box versus black-box.White-box models are derived from fundamental architectural insights which enables them to, in theory, precisely capture key performance-related behavior.
Because of the high degree of overlap effects — we observe an average prediction error of 17.9% and up to 75.3%

// 黑盒模型能学习到一些难以估计的overlap影响，但是像DNN这样的黑盒模型它们的复杂程度很高，如果集成到硬件中需要很大的存储与计算单元来为其服务。
Black-box models circumvent this problem by automatically learning the performance impact of the non-trivial overlap effects. Unfortunately, this ability comes at the cost of non-negligible training and implementation overheads.

➢ Introduction

- 第5段引出作者提出的方法

// 作者提出了黑盒白盒融合的slowdown模型, 这样的好处是能利用白盒模型的低实现复杂度建模确定性slowdown，用黑盒模型来建模不确定性slowdown提高预测准确度

Our key insight is that we can design accurate performance models with low complexity by combining the strengths of ***white-box and black-box*** *approaches. More specifically, the white box approach captures key performance-related behavior — to reduce training and implementation overhead — while the black-box model accounts for non-trivial overlap effects — to achieve high accuracy.*

➢ Introduction

- **第6段说明这种融合方法中白盒模型的设计思路，受到什么架构上的启发来设计这种模型。**

*Hybrid Slowdown Model (HSM) which builds upon **five white-box insights** regarding interference in GPUs :*

Insight 1：GPU的流式执行模型决定了一个kernel是访存密集还是计算密集的

(1） The streaming execution model of GPUs results in kernels being either memory or compute-bound.

Insight 2：对于计算密集型kernel，分配的SM增多性能越高

(2）The performance of compute-bound kernels scale linearly with the number of allocated SMs.

Insight 3： 访存密集型kernel的性能与内存带宽非常相关

(3）The performance of memory-bound kernels is strongly correlated with memory bandwidth.

Insight 4： DRAM的行Buffer命中率决定了一个访存存密集型kernel的潜在带宽利用率

(4）Insight 4： DRAM Row Buffer Hit Rate (RBH) determines a memory-bound kernel’s DRAM bandwidth utilization potential.

➢ Introduction

- **第6段说明这种融合方法中白盒模型的设计思路，受到什么架构上的启发来设计这种模型。**

Insight 5：*RBH在共享与独占GPU的情况下基本上是类似的。*
(5) Memory-bound kernels have similar RBH in the shared and private mode.

PS：从第6段中提出的几个Insight不难看出，对于访存密集型算子的特性是比较复杂的，其涉及到了其中4个insight；而对于计算密集型算子的性能分析是相对简单的其只与SM分配的数量有关系。那么应该如何来区分计算密集型算子与访存密集型算子呢？作者在下文给出了具体的量化模型。

➢ Introduction

- 第7段说明这种融合方法中黑盒模型的设计思路。并在最后一句通过数据说明HSM所取得的效果。

// 黑盒模型解决的问题是，在GPU下，访存密集型算子的不确定性RBH问题
The black-box component of HSM addresses the problem of relating RBH to performance for memory-bound applications，which is ***non-trivial*** *due to the highly parallel execution model of GPUs.*

// 作者观察到由RBH导致此类算子的性能与内存带宽的利用率成正比。通过建模一个线性回归的简单模型来解决RBH的预测精度问题。
We observe that performance is proportional to memory bandwidth utilization,, and we use linear regression to learn the precise relationship for each GPU architecture.

// 最后一句，总结了这种方法能达到的效果。
Overall, HSM has an average slowdown prediction error of 6.8% (30.3% max error); a significant improvement compared to the state-of-the-art.

➢ Introduction

- **第7段说明这种融合方法中黑盒模型的设计思路。并在最后一句通过数据说明HSM所取得的效果。**

// 黑盒模型解决的问题是，在GPU下，访存密集型算子的不确定性RBH问题
*The black-box component of HSM addresses the problem of relating RBH to performance for memory-bound applications，which is **non-trivial** due to the highly parallel execution model of GPUs.*

// 作者观察到由RBH导致此类算子的性能与内存带宽的利用率成正比。通过建模一个线性回归的简单模型来解决RBH的预测精度问题。
We observe that performance is proportional to memory bandwidth utilization,, and we use linear regression to learn the precise relationship for each GPU architecture.

// 最后一句，总结了这种方法能达到的效果。
Overall, HSM has an average slowdown prediction error of 6.8% (30.3% max error); a significant improvement compared to the state-of-the-art.

➢ Introduction

- **第8段简要说明了实验手段与测试指标，如何验证这种HSM的有效性，摆出具体数据说明各个指标的提升。**

// 利用HSM指导的SM分配策略来优化系统的QoS和Fairness（这两个指标是虚拟机关注的指标，**作者的意图是通过优化SM的硬件调度策略来实现类似于虚拟机的效果**）。

***To demonstrate the utility of HSM**, we use its accurate slowdown predictions to guide SM-allocation policies designed to optimize for system-level performance metrics such as QoS and fairness.*

// 进一步解释，HSM通过协调SM与内存带宽来优化Fair，STP，QoS指标
***More specifically**, we find that HSM-based SM allocation is an effective mechanism to manage both the **SM resources** and **shared memory bandwidth** in a coordinated way to improve system-level performance including **fairness** (HSM-Fair) as well as **system throughput** (STP) while maintaining **QoS** for the high-priority application (HSM-QoS)*

➢ Introduction

- **第8段简要说明了实验手段，如何验证这种HSM的有效性，摆出具体数据说明各个指标的提升。**

列举各个性能指标(Fair, QoS, STP)对比多种Baseline（*even partition, Themis ,DASE*）的提升

Our experiments show that HSM-based SM allocation leads to a significant improvement over both ***even partitioning*** *and* ***DASE-based SM allocation****.* ***HSM-Fair improves fairness by 1.59×, 1.36×, and 1.29× compared to even partitioning, Themis and DASE, respectively****.* ***When HSM QoS is used to optimize STP while respecting the QoS target of a high-priority application, it improves STP by 18.9%, 13.0%, and 15.2%*** *in mixed compute/memory-bound workloads compared to proportional SM partitioning, Themis based SM allocation, and DASE-based SM allocation, respectively.*

➢ Introduction

- **第9段总结了文章的贡献**
- *We introduce hybrid GPU slowdown modeling.*
- *We propose HSM*
- *We use HSM to allocate SMs to applications to achieve fairness- and QoS-aware multitasking GPUs*

➢ 对Introduction的总结：

- 第1段，引出存在的问题
- 第2段，第2段列举当前是如何解决此问题的
- 第3段作者从宏观层面给出了一些相关研究
- 第4段作者仔细深入地分析了GPU多作业系统的相关研究
- 第5段引出作者提出的方法
- 第6段说明这种融合方法中，其中的白盒模型的设计思路
- 第7段说明这种融合方法中，其中的黑盒模型的设计思路
- 第8段简要说明了实验手段与测试指标，并列出测试的数据结果
- 第9段总结文章的贡献

➢ 2 The Hybrid Slowdown Model(HSM)

- **第1段对Slowdown进行量化，解释什么是Slowdown**

$$\mathrm{NP}_k = \mathrm{IPC}_k^{\mathrm{Shared}} / \mathrm{IPC}_k^{\mathrm{Private}} \qquad (1)$$

衡量性能的指标IPC，一个时钟周期执行的指令数量

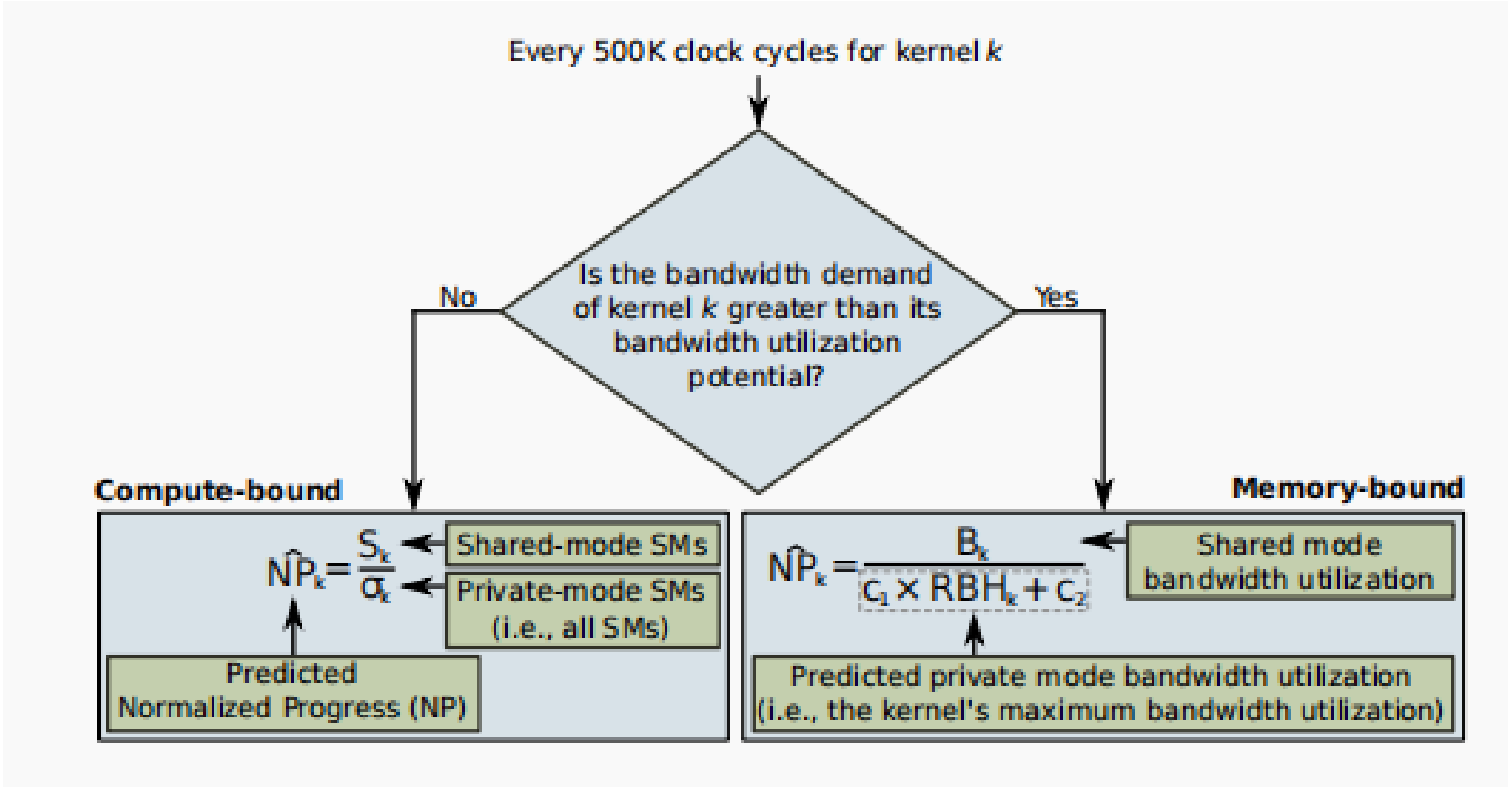
IPC: Instructions per cycle

共享模型下的IPC可以通过硬件计数器获得。难点是需要基于共享状态的信息来估计独占下的IPC

Shared-mode performance(IPC) can be measured straightforwardly with hardware counters. Private-mode performance is much more challenging: It requires predicting isolated performance ***based on shared-mode measurements gathered dynamically while applications are interfering with each other.***

➢ 2.1 HSM Overview

- **第2段宏观上介绍了HSM模型**



Every 500K clock cycles for kernel *k*
Is the bandwidth demand of kernel *k* greater than its bandwidth utilization potential?
No
Yes
Compute-bound
Memory-bound
$\hat{NP}_k = \frac{S_k}{\sigma_k}$
Shared-mode SMs
Private-mode SMs (i.e., all SMs)
Predicted Normalized Progress (NP)
$\hat{NP}_k = \frac{B_k}{C_1 \times RBH_k + C_2}$
Shared mode bandwidth utilization
Predicted private mode bandwidth utilization (i.e., the kernel's maximum bandwidth utilization)

Figure 1. Flow-chart outlining the main operation of HSM.

- 首先对kernel进行分类
- 不同类型的kernel有不同的slowdown量化公式
- 每隔500K clock进行一次评估

➢ 2.1 HSM Overview

- **第2段宏观上介绍了HSM模型**

// 对于计算密集型作业使用白盒模型；对于访存密集型作业使用LR model。
For computebound applications, HSM leverages the white-box insight that performance is proportional to SM allocation to predict NP . If the application is classified as memory-bound, HSM predicts the NP of the application using a linear regression model.

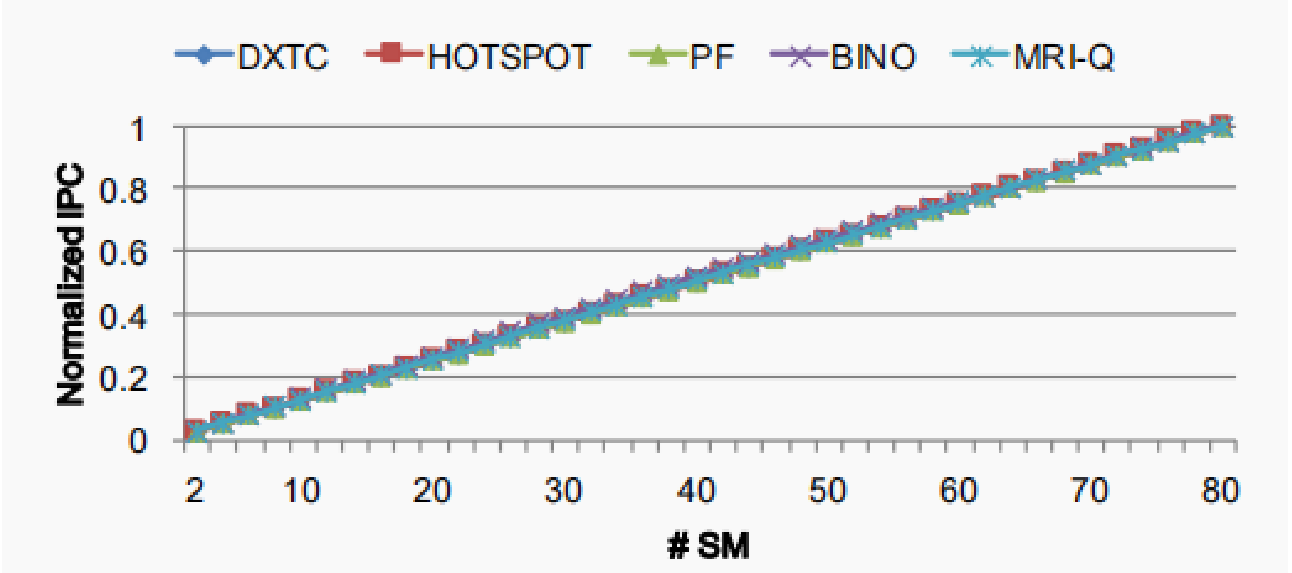
// 一句话总结HSM的特性：HSM足够精简，只会track少量的运行时信息：IPC、SM、BW、RBH
HSM is extremely lean and only tracks the number of executed instructions, allocated SMs, bandwidth utilization, and row buffer hits and accesses for each co-runner in the shared mode

➢ 2.2 HSM Classifier

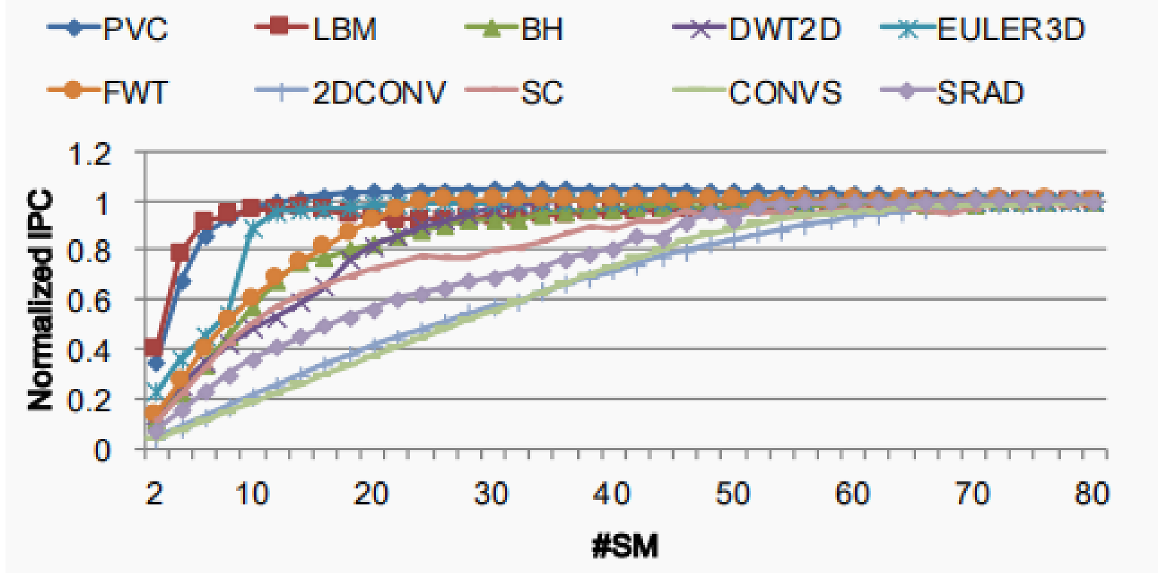
- **第3段：在overview之后，分别来剖析每个部分。第一个是HSM Classifier**

//在段落开头先引出了Insight 1，通过Insight 1可以对kernel进行分类：
Insight #1: The streaming execution model of GPUs results in kernels being either memory-bound or compute-bound.

根据测试的结果图来**直观地**说明计算密集与访存密集算子的不同



(a) Compute-bound kernels



(b) Memory-bound kernels

➢ 2.2 HSM Classifier

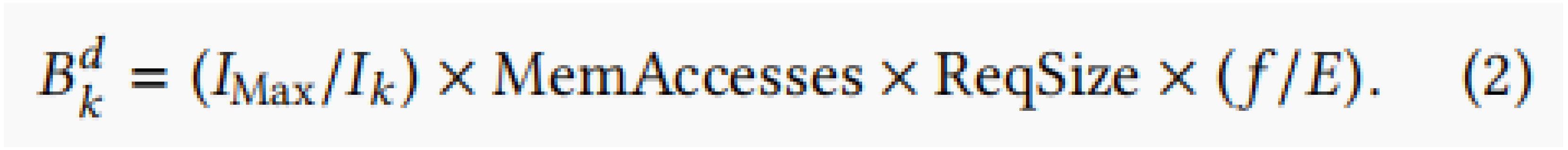
// 作者对Figure 2b的现象做了简要的解释：如果缓存命中率过低就，即使少量的SM就可以填充满内存系统；反之，亦然。
If the kernel misses frequently in the Last-Level Cache (LLC) and has low RBH, few SMs are enough to saturate the memory system. Conversely, a kernel with less frequent LLC misses (or better LLC hit rate) will need many SMs to saturate DRAM bandwidth.

- **在解释完现象之后，作者提出给出了Classifier的公式**

// 作者认为一个应用的带宽需求和其利用DRAM带宽的潜力确定了一个kernel的类型
To classify a kernel, we compute the application's memory bandwidth demand and its potential to utilize the theoretical DRAM bandwidth and compare these.

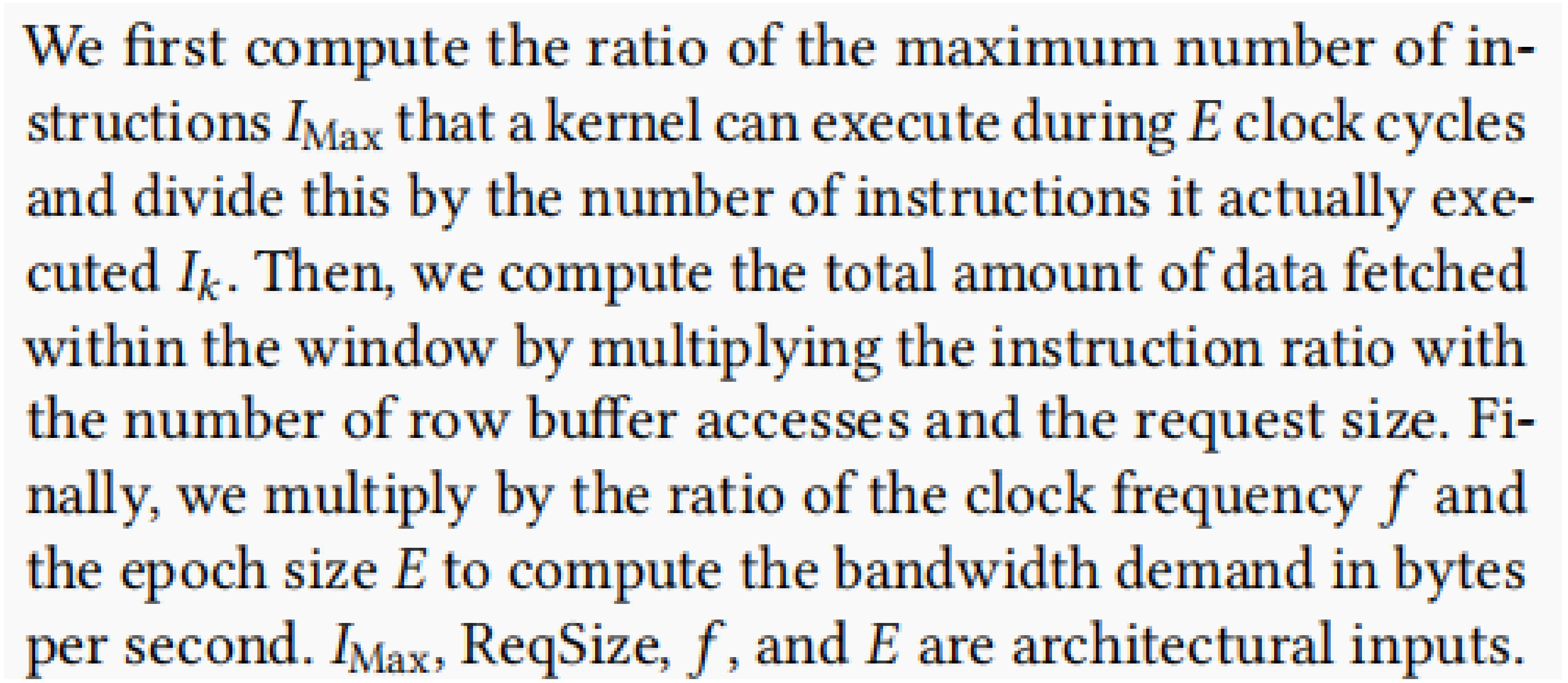
➢ 2.2 HSM Classifier

对于一个应用的最大内存带宽需求可以用公式2来计算



$$B_k^d = (I_{\text{Max}}/I_k) \times \text{MemAccesses} \times \text{ReqSize} \times (f/E). \quad (2)$$

上公式的含义就是：单位时间内多组指令的访存需求，MemAccesses是DRAM中row buffer的访存次数，如果不在row buffer中会进行多次的访问。



We first compute the ratio of the maximum number of instructions I_{Max} that a kernel can execute during E clock cycles and divide this by the number of instructions it actually executed I_k. Then, we compute the total amount of data fetched within the window by multiplying the instruction ratio with the number of row buffer accesses and the request size. Finally, we multiply by the ratio of the clock frequency f and the epoch size E to compute the bandwidth demand in bytes per second. I_{Max}, ReqSize, f, and E are architectural inputs.

➢ 2.2 HSM Classifier

对于一个应用理论上能利用内存带宽的潜力（当前能提供的带宽)

$$B_k^p = B_{\text{Max}} \times (c_1 \times \text{RBH}_k + c_2). \qquad (3)$$

上公式可以看到，一个应用的带宽潜力与内存带宽的最大值成正比，也就是说，因为RBH的存在每次的内存访问不可能达到带宽的最大值

如果一个kernel的带宽需求大于实际能提供的带宽那么就是访存密集型的

The kernel is memory-bound if its bandwidth demand exceeds the effective bandwidth supply.

➢ 2.3 Predicting Normalized Progress (NP)

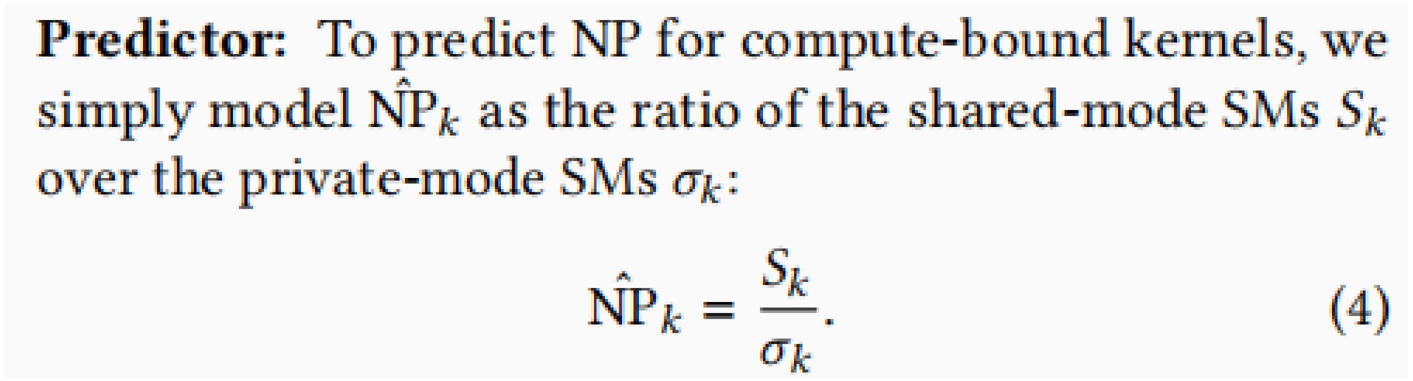
- **下面开始介绍HSM的第二部分，在对kernel分类之后怎么进行slowdown的预测。在2.3.1之前作者将后面用到的符号约定在这里统一声明。**

➢ 2.3.1 Compute-Bound Kernels

再次给出计算密集型算子的特点：计算量足够多，局部性足够好。并且又引出了Insight 2，对于计算密集型算子，其性能和SM分配数量成正比

Compute-bound kernels have sufficiently high arithmetic intensity and locality for the SM's L1 cache, shared memory, or constant cache to hide the latency of accessing the LLC or DRAM： Insight #2: For compute-bound kernels, performance improves linearly with the number of allocated SMs.

对于计算密集型算子的Slowdown预测很容易计算，用共享的性能除以独占的性能。



Predictor: To predict NP for compute-bound kernels, we simply model $\hat{NP}_k$ as the ratio of the shared-mode SMs S_k over the private-mode SMs σ_k:

$$\hat{NP}_k = \frac{S_k}{\sigma_k}. \quad (4)$$

➢ 2.3.1 Memory-Bound Kernels

- **作者首先强调了共享内存系统上干扰的影响的困难，为了克服这个挑战融合了3个白盒insight和黑盒RL模型来建模内存边界的kernel，再次提到Insight3**

Insight #3: The performance of memory-bound kernels is correlated with memory bandwidth, and eventually saturates when increasing the number of allocated SMs.

解释了图2b中，前一段随着SM会线性增长的原因：

The reason is that GPU memory systems are highly parallel, and the effective bandwidth increases when more requests — and in particular row buffer hits — are available within each memory channel. Thus, increasing the number of SMs increases Memory-Level Parallelism (MLP) and results in better memory bus utilization

也解释了最后放缓的原因：

Eventually, all of the potential row buffer hits are available to the memory controller, maximizing the effective memory bandwidth. Increasing the SM allocation beyond this point is useless

➢ 2.3.1 Memory-Bound Kernels

- **对上面的分析进行一个总结，并引出附带的问题**

访存密集的kernel在独占方式下就会填满整个DRAM带宽。因此需要一个架构上的特性来解释填满DRAM性能的原因，以便在共享模式下也可以估计。
A memory-bound kernel will saturate DRAM bandwidth in private mode. Thus, we need to identify an architectural characteristic that is measurable in shared mode and has a clear relationship with saturation performance.

作者引出了Insight 4，说明DRAM带宽的利用率和RBH几乎成正比，并用实验结果来验证这个发现。
Insight #4: RBH determines effective DRAM bandwidth utilization and the relation is approximately linear.
Figure 3 supports Insight #4 by plotting private-mode RBH against bandwidth utilization for our 10 memory-intensive benchmarks as well as the relation we identify with linear regression

➢ 2.3.1 Memory-Bound Kernels

现在知道了在私有模式下RBH是影响带宽的关键，但是我们还需要在共享状态下的RBH是否和私有状态下的RBH二者是否有一定的关系。

We have now determined that RBH is a key parameter for determining a kernel's effective bandwidth utilization in private mode. However, NP prediction is performed using shared-mode counters. Thus, we need establish the relationship between shared-mode and private-mode RBH.

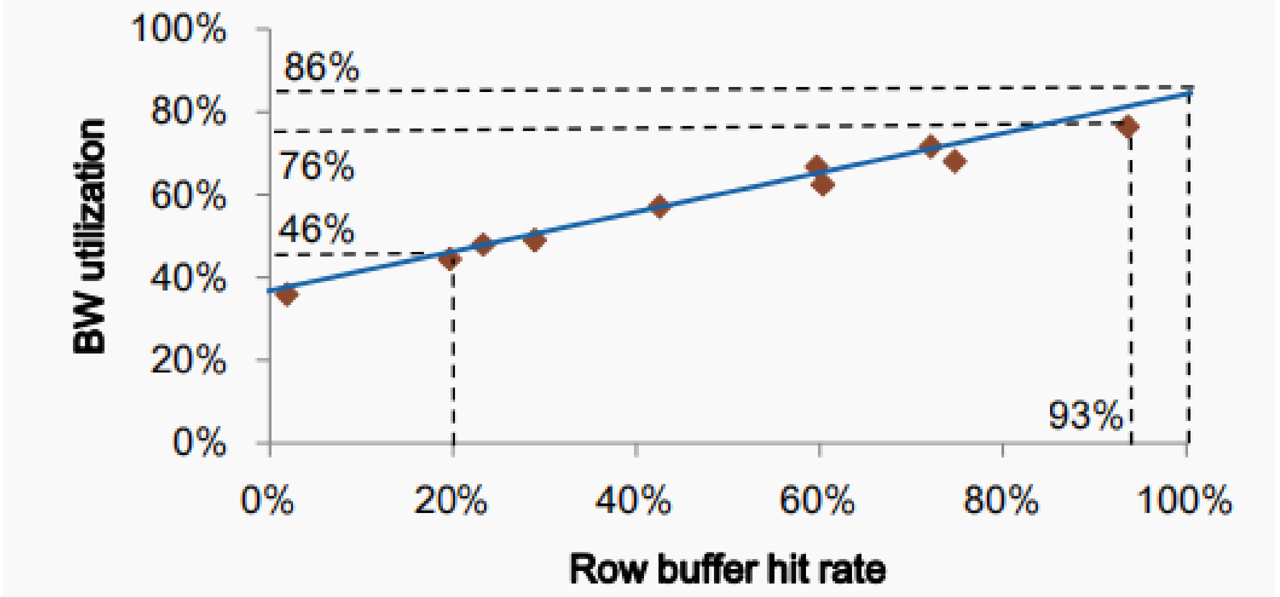


Figure 3. HSM uses linear regression to learn the relationship between RBH and bandwidth utilization.

➢ 2.3.1 Memory-Bound Kernels

共享与私有状态下的关系就是Insight 5，RBH在私有与共享状态下几乎是一致的。

Insight #5: RBH is sufficiently similar in the shared and private mode to enable accurate NP prediction.

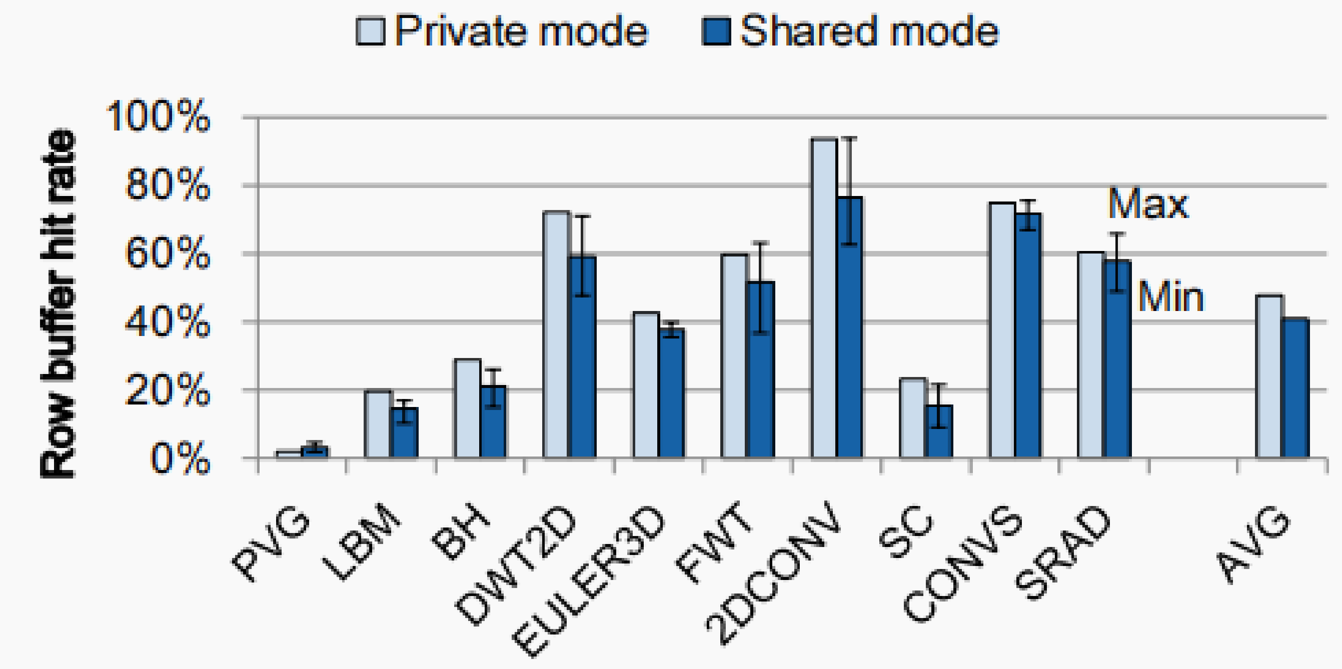
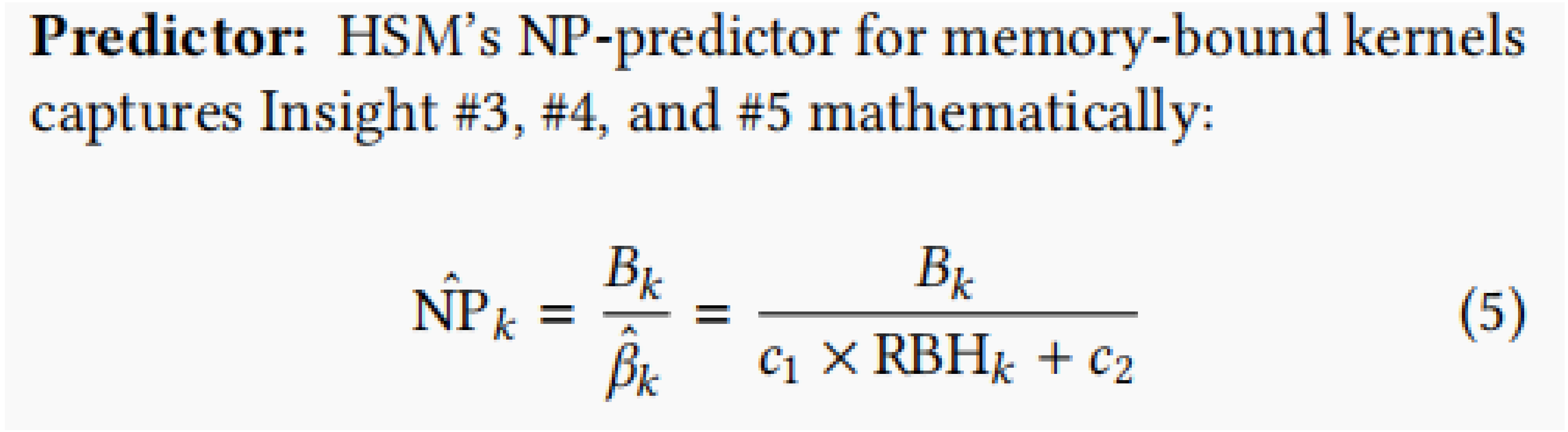


Figure 4. RBH is similar in the shared and private modes.

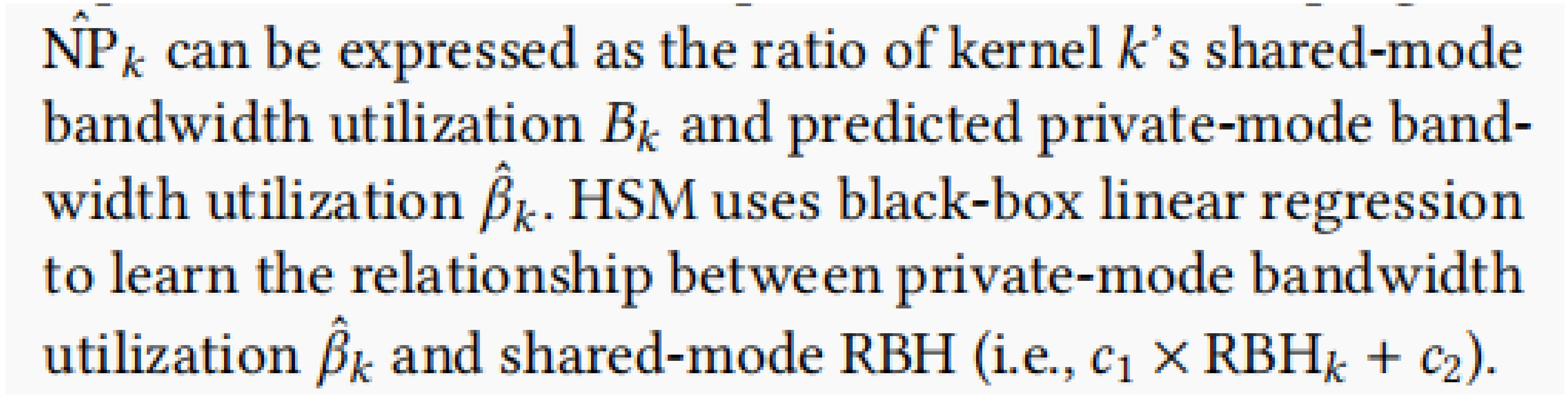
➢ 2.3.1 Memory-Bound Kernels

结合Insight 3,4,5，给出了内存密集型算子的NP预测公式，利用共享状态下的RBH来预测（LR model）私有状态下的带宽



Predictor: HSM's NP-predictor for memory-bound kernels captures Insight #3, #4, and #5 mathematically:

$$\hat{NP}_k = \frac{B_k}{\hat{\beta}_k} = \frac{B_k}{c_1 \times \text{RBH}_k + c_2} \tag{5}$$



$\hat{NP}_k$ can be expressed as the ratio of kernel k's shared-mode bandwidth utilization B_k and predicted private-mode bandwidth utilization $\hat{\beta}_k$. HSM uses black-box linear regression to learn the relationship between private-mode bandwidth utilization $\hat{\beta}_k$ and shared-mode RBH (i.e., $c_1 \times \text{RBH}_k + c_2$).

The architectural meaning of the learned constantsc1 and c2 is that ***c1*** *expresses the expected increase in bandwidth utilization for a* ***corresponding increase in RBH****, while* ***c2*** *is the expected bandwidth utilization* ***when all memory requests are row buffer misses****.* C1 与C2可以在离线模式通过测试获得

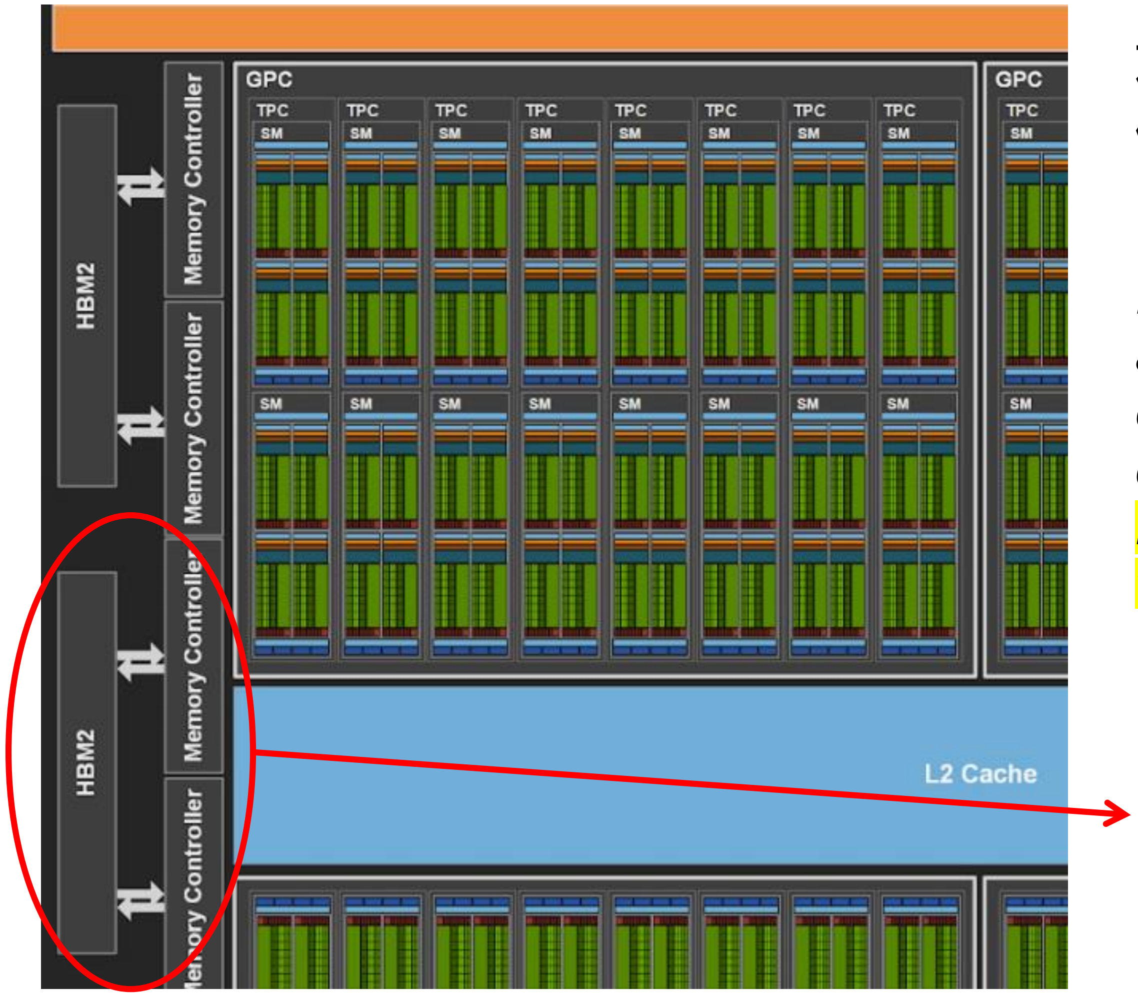
➢ 2 The Hybrid Slowdown Model (HSM) 的总结：

作者以总分的方式展开：首先给出了HSM整体架构；然后依次对架构中的部分展开。在对每部分说明的时候，通过实验数据得到的Insight来说明设计的原则，并进行了公式化的。

➢ 3 Explaining HSM’s Memory System Insights

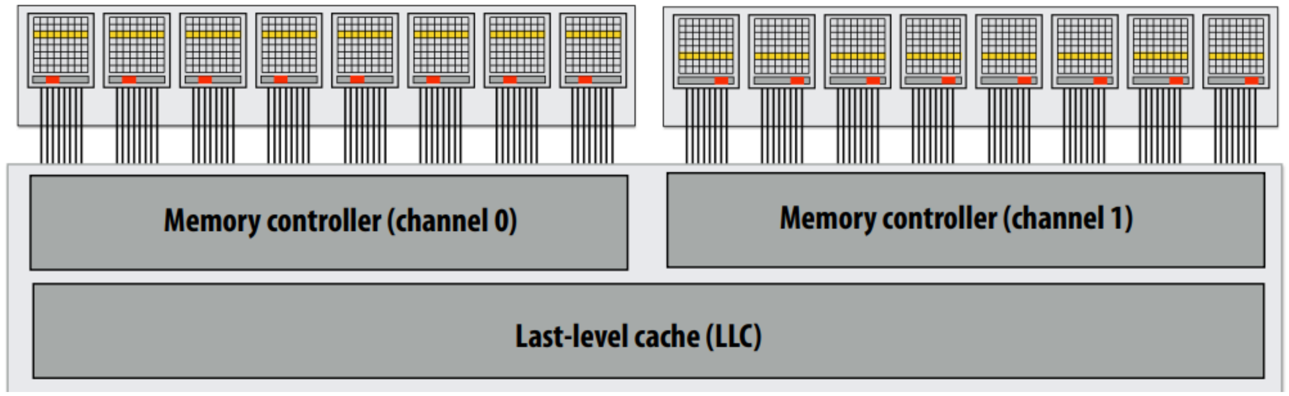
- **这部分作者对Insight4 和 Insight5进行了解释，同样以总分的方式展开。**

PS： 为了方便后面解释，这里我先贴一下GPU的架构和memory controller的图

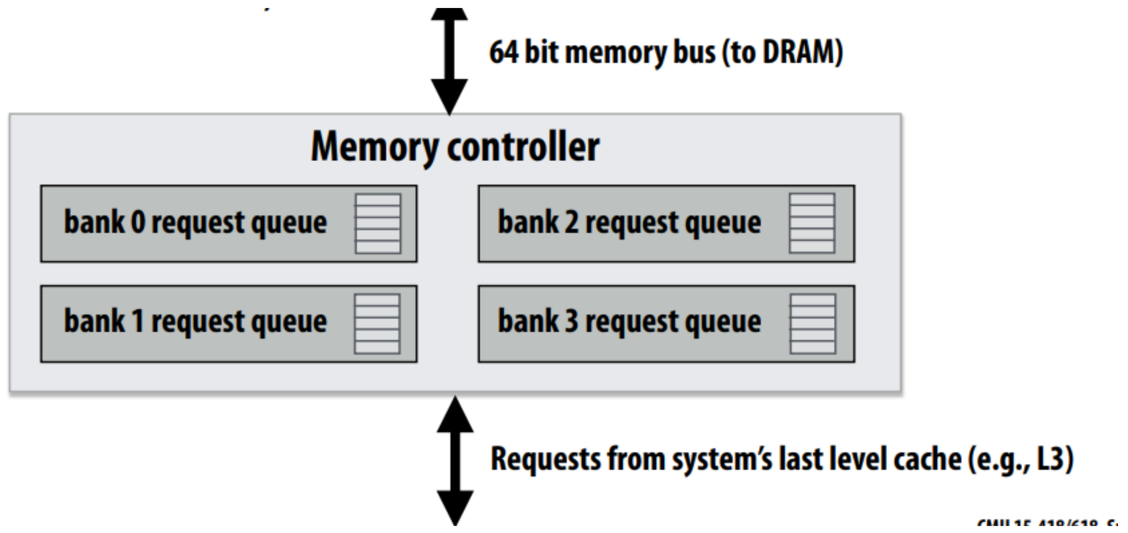


这里贴出了A100的架构图，可以看到访存层次依次为：L1 cache(SM 内共享) -> L2 cache -> HBM2

In HBM, DRAM chips are placed on top of each other and memory accesses go through Through Silicon Vias (TSVs). This organization enables more channels and higher bandwidth per channel than non-stacked organizations.
Each channel still consists of rows, columns, and banks, and it is very efficient to access rows that are already in the row buffers

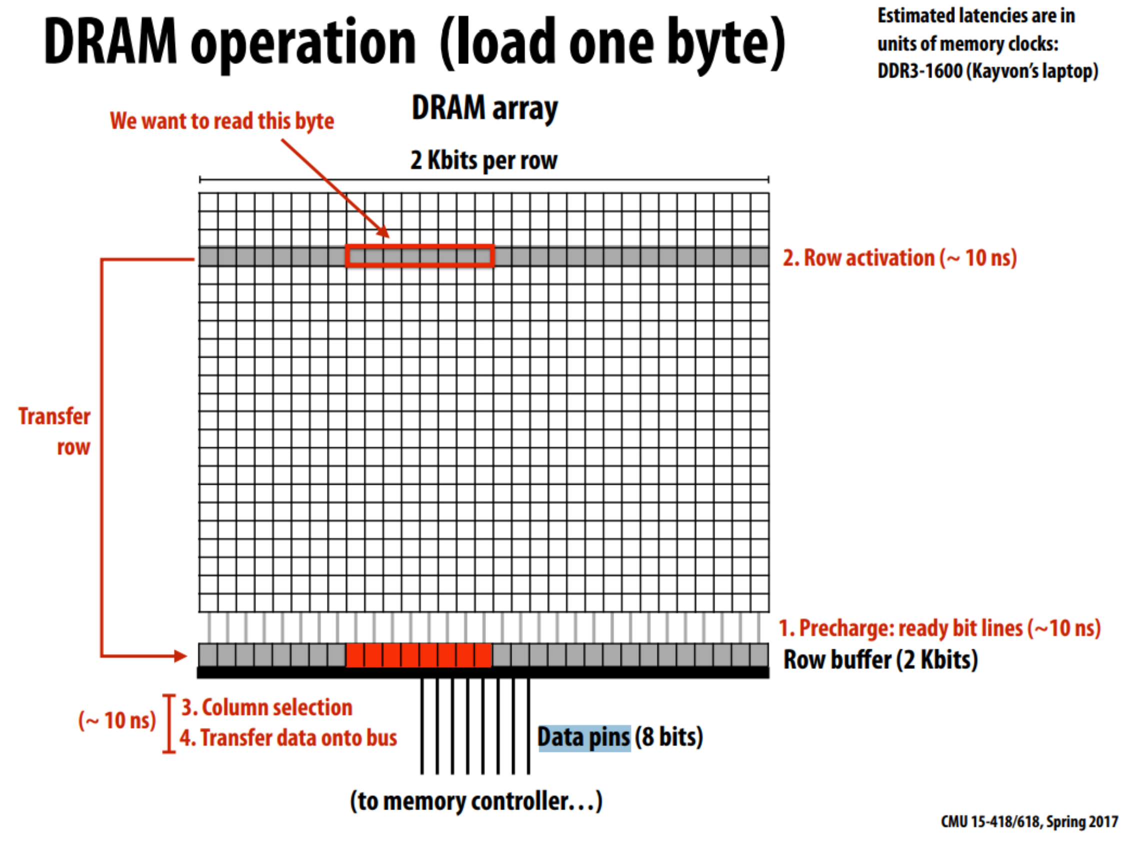


PS:关于Memory Controller的介绍



内存控制器执行过程

Common scheduling policy: FR-FCFS (frst-ready, frst-come-frst-serve)


DRAM operation (load one byte)
Estimated latencies are in units of memory clocks: DDR3-1600 (Kayvon's laptop)
We want to read this byte
DRAM array
2 Kbits per row
2. Row activation (~ 10 ns)
Transfer row
1. Precharge: ready bit lines (~10 ns)
Row buffer (2 Kbits)
(~ 10 ns)
3. Column selection
4. Transfer data onto bus
Data pins (8 bits)
(to memory controller…)
CMU 15-418/618, Spring 2017

一次DRAM的访问过程

- **Best case latency: read from active row**
  - Column access time (CAS)

- **Worst case latency: bit lines not ready, read from new row**
  - Precharge (PRE) + row activate (RAS) + column access (CAS)

➢ 3 Explaining HSM’s Memory System Insights

- 作者首先通过一个例子解释了为什么RBH会决定了带宽的利用潜力
- Explaining Insight #4，*explains why RBH determines bandwidth utilization potential by considering two cases where different memory requests are queued in the memory controller*.

图6的请求访问的过程体现了内存控制器的**FR-FCFS调度特性**，例如caseA实际的访问顺序是：A1 A2 A3 A4

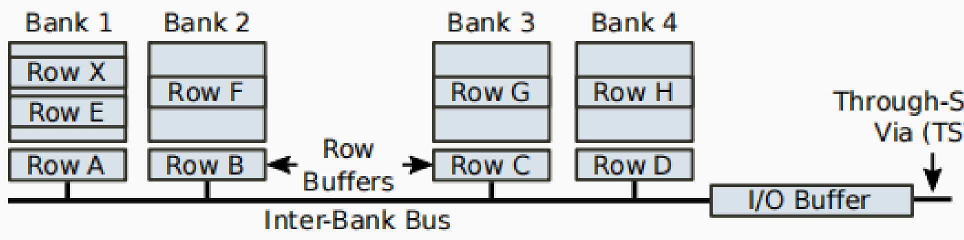
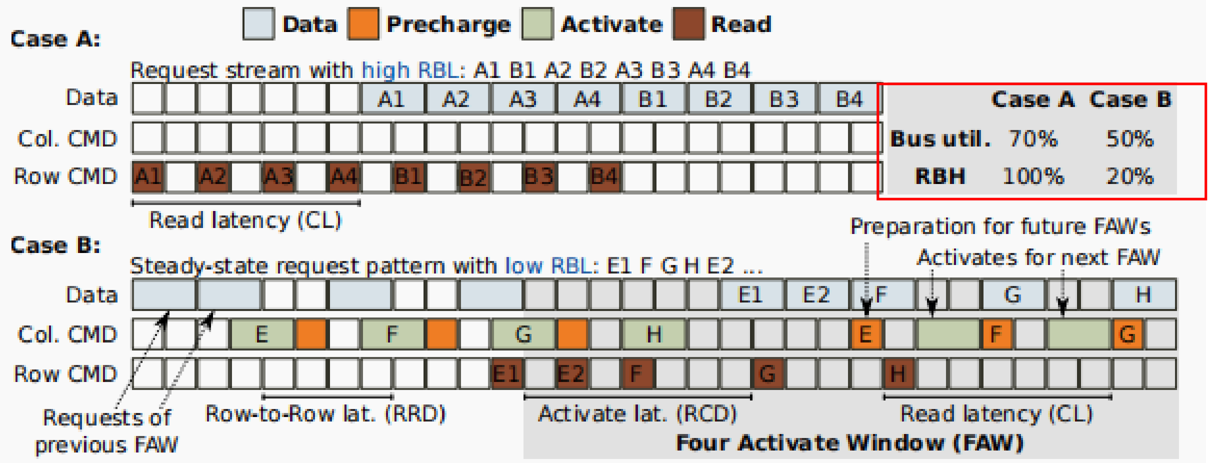


Figure 5. Internal organization of an HBM channel with the initial state for the examples in Figures 6 and 7.

示例中HBM中内存使用的当前状态，A，B，C，D都在Row Buffer中



	Case A	Case B
Bus util.	70%	50%
RBH	100%	20%

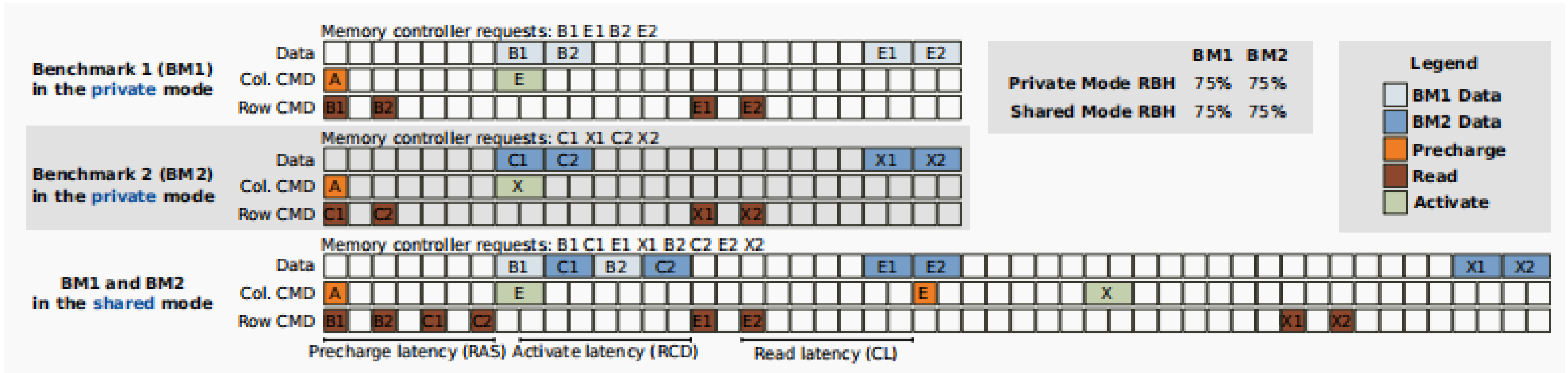
Figure 6. Example explaining why RBH determines effective bandwidth utilization (Insight #4).

Case A 与 Case B在图5的状态下的，RBH与Bus Util

➢ 3 Explaining HSM’s Memory System Insights

- 然后作者解释了Insight#5，

Explaining Insight #5，*why shared and private-mode RBH are similar*
在共享模式下，并不影响row buffer中的数据状态所以RBH保持不变，内存控制器会优先访问在row buffer中的B2和C2；但是对于BM1中的E和BM2中的X他们位于同一行中因此会发生冲突，memory controller根据FIFO的原则会先执行E再激活X，因此这会干扰到BM2对于X的访问，影响BM2的带宽利用率。这也间接驱动了基于HSM的多任务调度策略。
indirectly motivates for HSM-based multitasking schemes



Benchmark 1 (BM1) in the private mode
Memory controller requests: B1 E1 B2 E2
Data
Col. CMD
Row CMD
Benchmark 2 (BM2) in the private mode
Memory controller requests: C1 X1 C2 X2
BM1 and BM2 in the shared mode
Memory controller requests: B1 C1 E1 X1 B2 C2 E2 X2
Precharge latency (RAS)
Activate latency (RCD)
Read latency (CL)
Legend
BM1 Data
BM2 Data
Precharge
Read
Activate

	BM1	BM2
Private Mode RBH	75%	75%
Shared Mode RBH	75%	75%

Figure 7. Example explaining why RBH is similar in shared and private mode (Insight #5).

➢ 4 Fairness/QoS-Aware Multitasking

- **在阐述完HSM之后，作者提出了基于HSM的多任务调度的架构与策略。同样是总分的结构**

- 起始段先说明了HSM评价指标NP与多种指标的关系；然后简单介绍了系统的设计

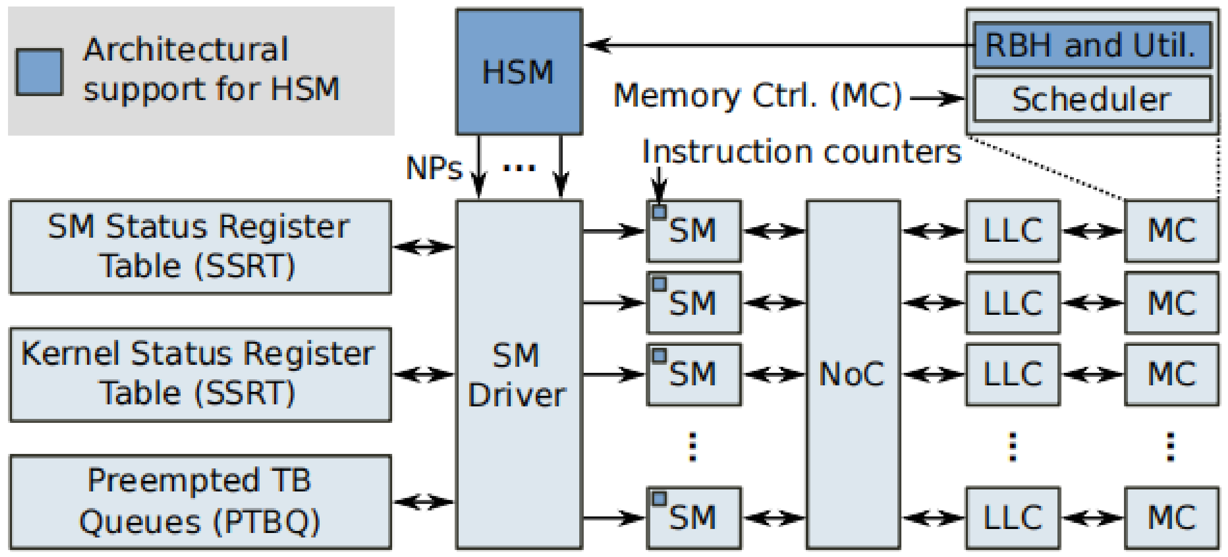
通过NP可以衡量多个指标包括：STP、ANTT、QoS。

More specifically, STP is the sum of NPs, while ANTT（平均周转时间） is the harmonic mean of NPs [24]. Fairness is the ratio of the minimum and maximum NPs [7]. QoS targets can be defined as an NP lower-bound for a high-priority application.

Supporting multitasking requires adding ***two tables*** *— the SM Status Register Table (SSRT) and the Kernel Status Register Table (KSRT) — and* ***queues*** *for storing the handlers of preempted TBs for each kernel. The SM driver schedules TBs onto SMs and uses the SSRT and KSRT to keep track of kernel and TB execution*

➢ 4.1 Architectural Support for HSM

- **这一段说明了基于HSM需要的硬件架构改动**


Architectural support for HSM
HSM
RBH and Util.
Memory Ctrl. (MC)
Scheduler
NPs
...
Instruction counters
SM Status Register Table (SSRT)
Kernel Status Register Table (SSRT)
Preempted TB Queues (PTBQ)
SM Driver
SM
NoC
LLC
MC

igure 8. A block diagram showing the architectural support
r HSM-based SM allocation policies.

- 为了获取共享状态下RBH的值，需要在内存控制器当中添加两个计数器

*HSM measures RBH by adding a **memory request counter** and a **row buffer hit counter** for each co-runner in all memory controllers*

- *同样需要一个计数器来获取*共享模式下的带宽利用率

shared-mode bandwidth utilization can be captured with a 19-bit counter for each kernel and co-runner

➢ 4.2 Implementing HSM-based SM Allocation

- **这一段建立了SM的分配模型**

• SM Driver会在每个epoch获取相应的参数，基于这些参数计算NP值并根据NP值做SM的重分配。如果一个epoch中有任务结束了，那么对于剩余的任务基于NP做不会重分配，因为共享的状态已经发生了改变。
At the end of an epoch (e.g., 500K clock cycles), HSM retrieves the executed instructions, bandwidth utilization, and the number of row hits and accesses for each co-runner, and uses these values to predict the current NP of all co-running kernels

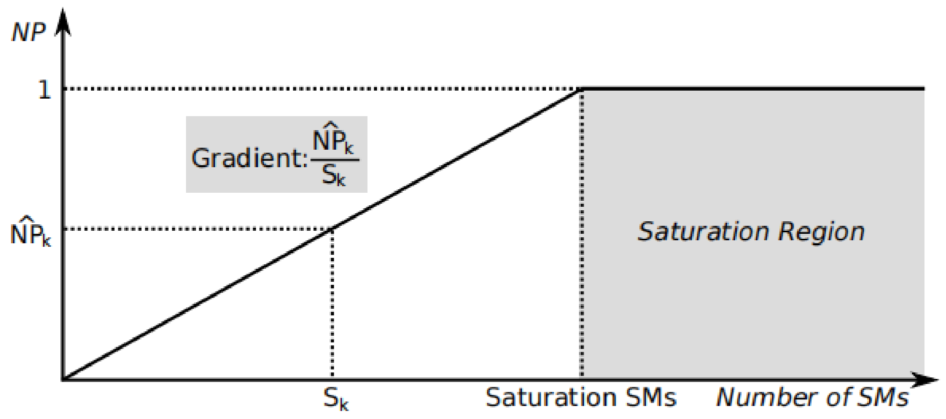


Figure 9. Normalized progress as a linear function of the number of SMs allocated to a kernel.

- 对于SM分配可以建立起一个统一的模型，如图9所示。不论是计算还是访存密集在SM分配少的时候他们的NP都与SM呈线性，只不过有的访存密集的算子会进入饱和区。并且不同的应用图9都会是不一样的。分配的原则就是SM不要进入饱和区

➢ 4.3 HSM-based SM Allocation Policies

- **这一段说明了如何基于HSM做SM的分配策略，有两种策略HSM-Fair 与 HSM-QoS**

• HSM-Fair就是尽量保证所有作业的NP是相同的。
The objective of HSM-Fair is to improve fairness by ensuring that the NPs of co-running kernels are approximately equal.

SM的分配过程是找到最大NP值的kernel，并根据图9找到合适的SM数量，使得所有kernel的NP尽量相同
First, HSM Fair finds the kernel with the maximum NP and the kernel with the minimum NP. Then, it uses the linear NP model to compute how many SMs should be taken from the high-NP kernel and given to the low-NP kernel to result in similar NP for both kernels.

➢ 4.3 HSM-based SM Allocation Policies

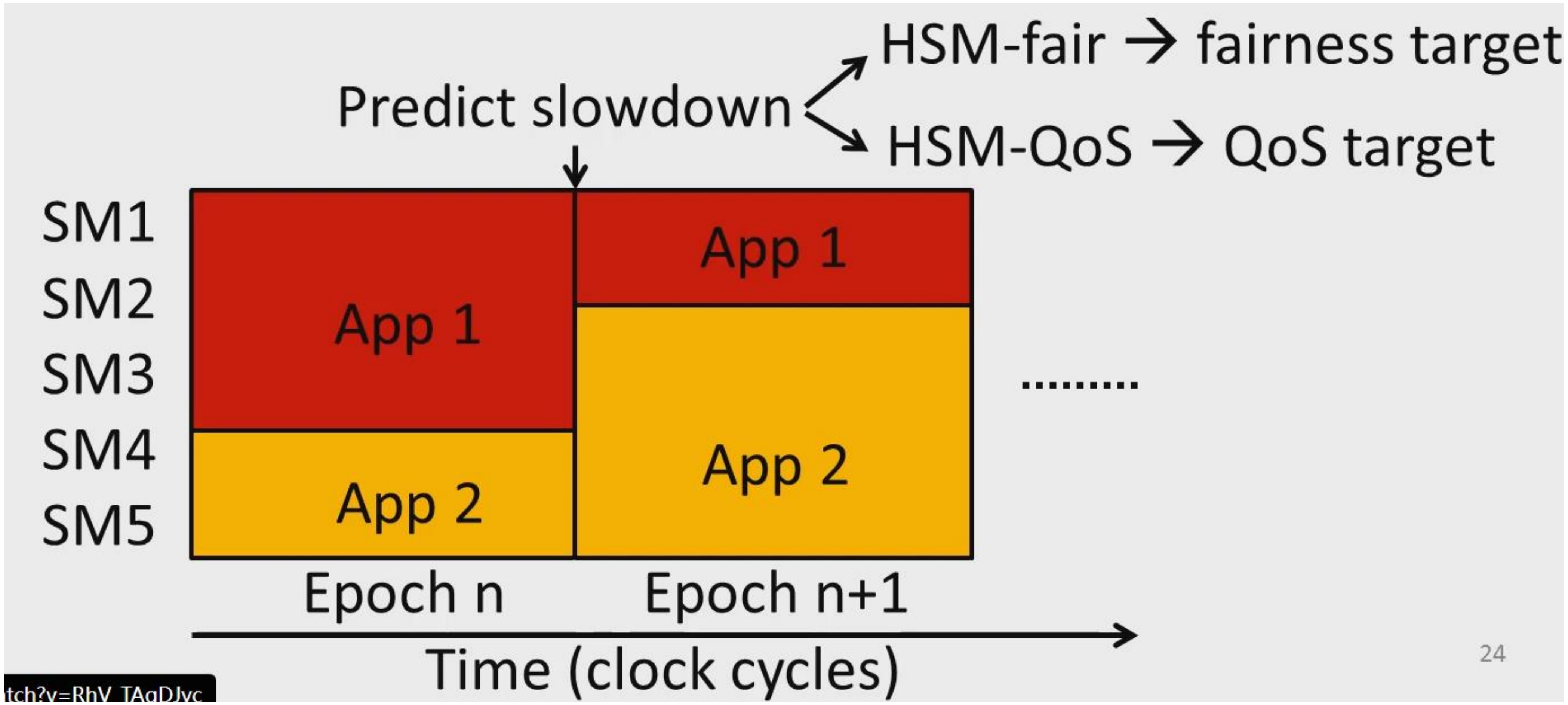
- **这一段说明了如何基于HSM做SM的分配策略，有两种策略HSM-Fair 与 HSM-QoS**

• HSM-QoS就是保证高优先级的kernel的NP高于某个阈值，并且使用剩余的资源来最大化系统吞吐。
This policy maintains the NP of the kernels of a high-priority application at a certain level (e.g., 0.8), and then uses the remaining resources to maximize STP.

这种情况下SM的分配过程是，如果高优先级的作业没有被满足，那么根据图9来分配其SM个数，如果被满足那么会适当减少SM的数量，并分配给低优先级的任务来提高吞吐。
it needs to marginally outperform its performance target. Then, we give the freed SMs to the low-priority kernels such that the sum of their NP values, and thus system throughput, is maximized.

PS: HSM-Based SM 分配的示例

SM对Kernel的分配在运行时是动态的。而AMD GPU的StreamWithCUMask只能在程序运行前来设置，当程序运行时不可被抢占。



➢ 5 Methodology

- **这一节用来告知读者的实现方法与相应的实验参数配置**

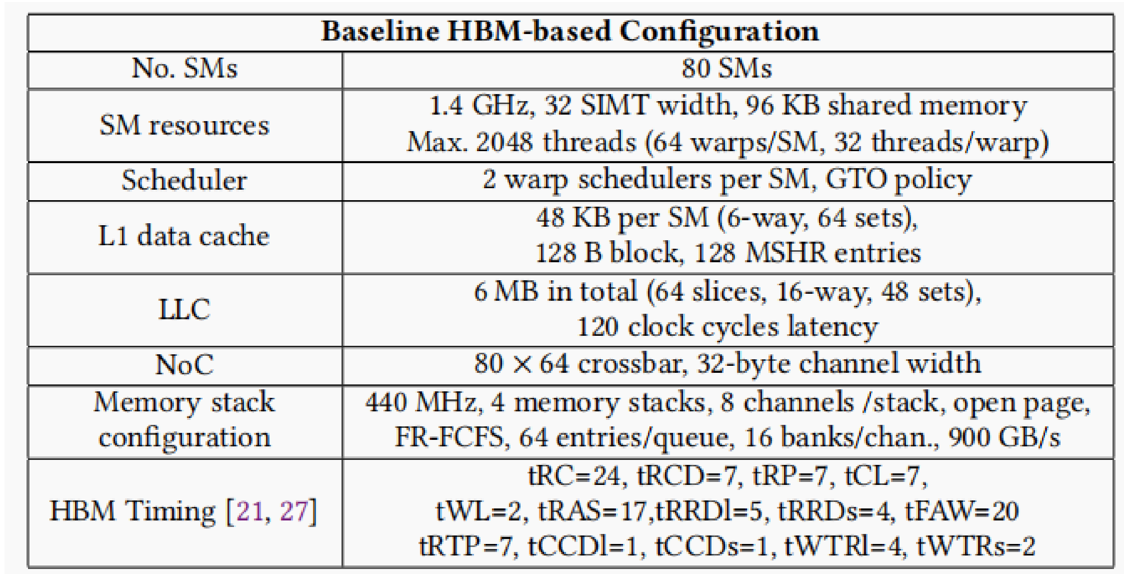
这篇文章的实现是基于**GPGPU-sim** v3.2.2模拟器，并对其扩展内存仿真器。

•*Simulated System: GPGPU-sim is further extended with Ramulator [27], a detailed memory simulator, to model an HBM-based memory system*

模拟的GPU参数：

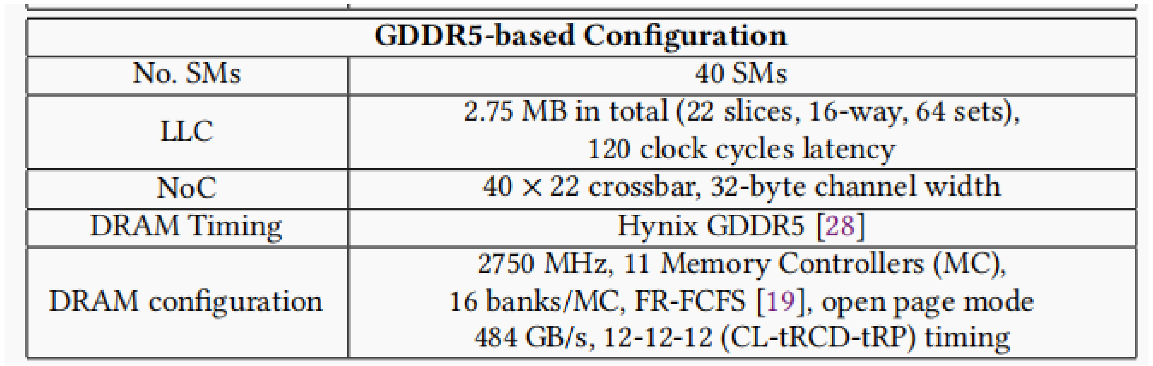
•*We model a GPU with 80 SMs that are connected through a crossbar to 32 memory channels (8 channels per stack) and two 96 KB LLC slices per channel*

- *给出了系统配置参数表格：包含GPU仿真系统配置与内存仿真器配置*



Baseline HBM-based Configuration	
No. SMs	80 SMs
SM resources	1.4 GHz, 32 SIMT width, 96 KB shared memory Max. 2048 threads (64 warps/SM, 32 threads/warp)
Scheduler	2 warp schedulers per SM, GTO policy
L1 data cache	48 KB per SM (6-way, 64 sets), 128 B block, 128 MSHR entries
LLC	6 MB in total (64 slices, 16-way, 48 sets), 120 clock cycles latency
NoC	80 × 64 crossbar, 32-byte channel width
Memory stack configuration	440 MHz, 4 memory stacks, 8 channels /stack, open page, FR-FCFS, 64 entries/queue, 16 banks/chan., 900 GB/s

HBM Timing [21, 27]	tRC=24, tRCD=7, tRP=7, tCL=7,
	tWL=2, tRAS=17,tRRDl=5, tRRDs=4, tFAW=20
	tRTP=7, tCCDl=1, tCCDs=1, tWTRl=4, tWTRs=2

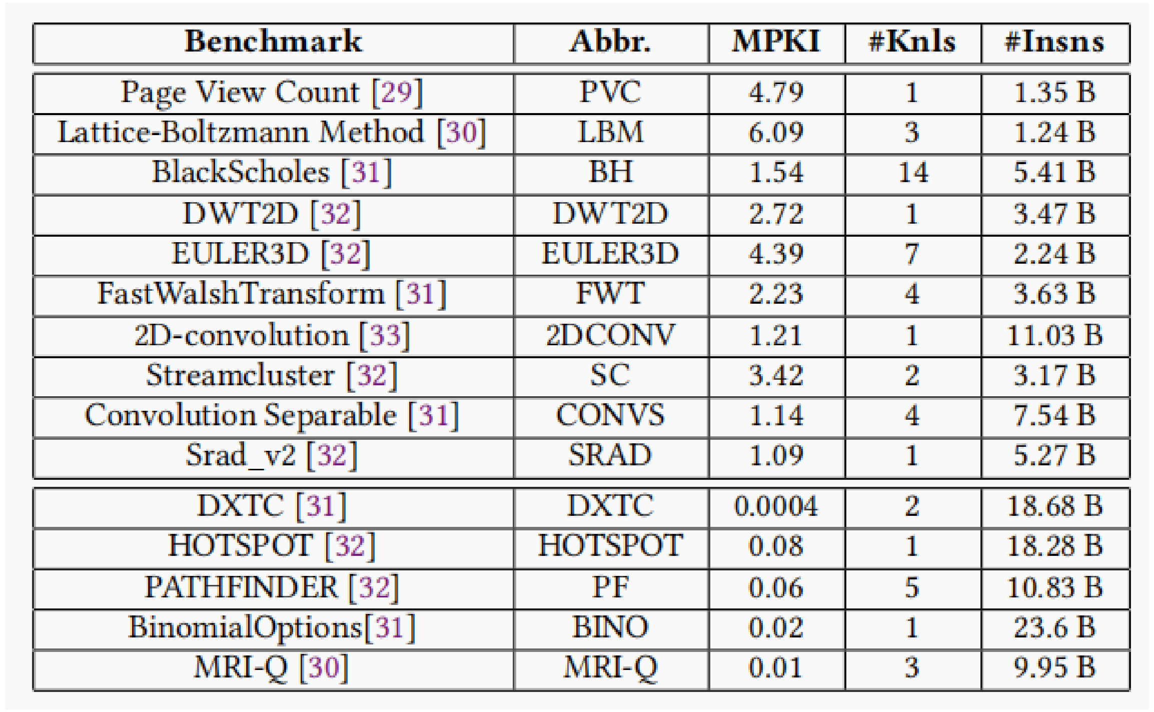


GDDR5-based Configuration	
No. SMs	40 SMs
LLC	2.75 MB in total (22 slices, 16-way, 64 sets), 120 clock cycles latency
NoC	40 × 22 crossbar, 32-byte channel width
DRAM Timing	Hynix GDDR5 [28]
DRAM configuration	2750 MHz, 11 Memory Controllers (MC), 16 banks/MC, FR-FCFS [19], open page mode 484 GB/s, 12-12-12 (CL-tRCD-tRP) timing

➢ 5 Methodology

- *给出了测试的负载，并统计kernel了相关的数据*

Workloads: These benchmarks are selected from Rodinia , Parboil , CUDA SDK , PolyBench , and Mars



Benchmark	Abbr.	MPKI	#Knls	#Insns
Page View Count [29]	PVC	4.79	1	1.35 B
Lattice-Boltzmann Method [30]	LBM	6.09	3	1.24 B
BlackScholes [31]	BH	1.54	14	5.41 B
DWT2D [32]	DWT2D	2.72	1	3.47 B
EULER3D [32]	EULER3D	4.39	7	2.24 B
FastWalshTransform [31]	FWT	2.23	4	3.63 B
2D-convolution [33]	2DCONV	1.21	1	11.03 B
Streamcluster [32]	SC	3.42	2	3.17 B
Convolution Separable [31]	CONVS	1.14	4	7.54 B
Srad_v2 [32]	SRAD	1.09	1	5.27 B
DXTC [31]	DXTC	0.0004	2	18.68 B
HOTSPOT [32]	HOTSPOT	0.08	1	18.28 B
PATHFINDER [32]	PF	0.06	5	10.83 B
BinomialOptions[31]	BINO	0.02	1	23.6 B
MRI-Q [30]	MRI-Q	0.01	3	9.95 B

➢ 5 Methodology

- *给出了系统的评价指标*

Metrics: We use three multi-program metrics: fairness, system throughput (STP) and average normalized turnaround time (ANTT)

➢ 6 Evaluation

- **这一节讲述实验环节：包括Baseline、针对各个指标的测试**

针对实验的评测包含如下几部分：

•*6.1 HSM Accuracy*

•*6.2 Fairness-Aware SM Allocation*

•*6.3 QoS-Aware SM Allocation*

•*6.4 Memory Bandwidth Partitioning*

•*6.5 Sensitivity Analyses*

➢ 6 Evaluation

• 6.1 HSM Accuracy

为了衡量HSM模型的准确度，作者对比了DASE，Themis两个相关工作：

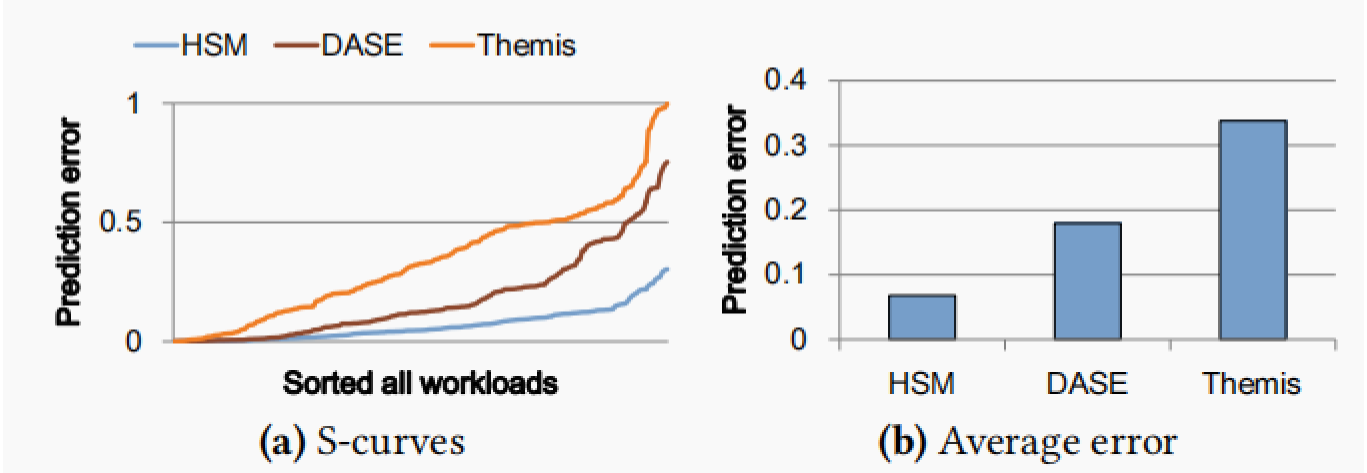


Figure 10. HSM accuracy across all workload mixes. *HSM achieves significantly higher accuracy compared to state-of-the-art DASE (white-box) and Themis (black-box).*

The average prediction error equals 6.8% for HSM (max error of 30.3%),whereas the average prediction errors of DASE and Themis are 17.9% (75.3% max error) and 33.8% (99.8% max error). We sort the prediction errors of each scheme in Figure 10a to illustrate that HSM has a significantly better error distribution than DASE and Themis. In fact, HSM is more accurate than DASE and Themis in nearly all workloads;

➢ 6 Evaluation

• 6.2 Fairness-Aware SM Allocation

图11给出了以HSM-Fair为指导调度下的，三个指标在不同baseline的对比

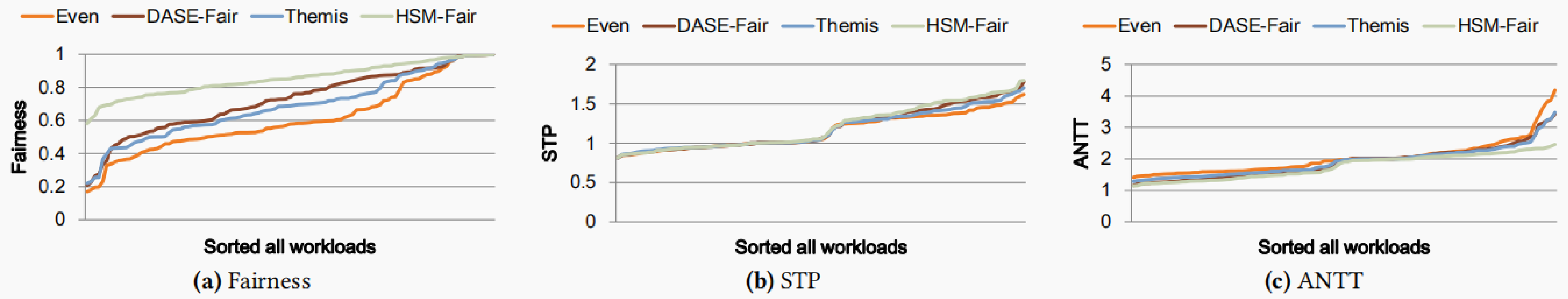


Figure 11. Fairness-aware SM allocation for all workload categories. *HSM-Fair significantly improves fairness compared to even partitioning and DASE for all workload mixes, while improving STP and ANTT for many.*

// 在保证 公平性指标较好的情况下，也提高了STP、ANTT指标。

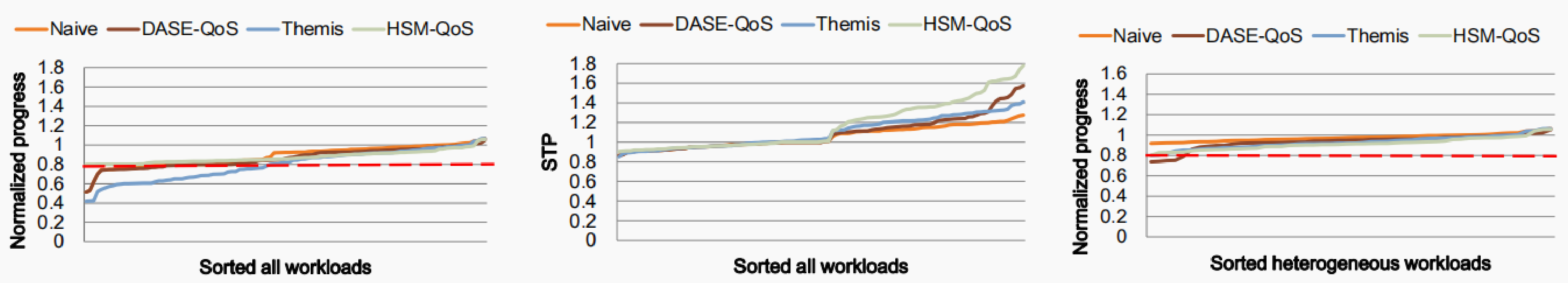
On average, we find that HSM-Fair achieves a fairness of 84.1% compared to 52.7%, 61.9%, and 65.2% for even partitioning, Themis, and DASE, respectively.

we note an average improvement in STP by 7.2% and up to 12.2%; ANTT improves by 15.8% on average and up to 20.4%.

➢ 6 Evaluation

• 6.3 QoS-Aware SM Allocation

图12给出在QoS设定为0.8的情况下，不同baseline的对比，每个子图用一两句话来解释。



(a) QoS (all workloads) **(b)** STP **(c)** QoS (heterogeneous workload mixes)

Figure 12. QoS-aware SM allocation for a QoS target of 0.8. *HSM-QoS meets the 0.8 NP target for all workloads while improving STP; even partitioning and DASE do not always meet the QoS target, and yield suboptimal STP.*

(a) *On HSM-QoS meets the QoS target for all workloads whereas the baseline (proportional partitioning), Themis, and DASE do not.*
(b) Moreover, HSM-QoS improves STP significantly by providing the SMs that are not needed by the high-priority application to meet its QoS target, to the low-priority application. HSM-QoS improves STP by 7.7%, 4.4%, and 5.5% on average across all workloads compared to the baseline, Themis, and DASE, respectively.
(c) *for the more challenging heterogeneous workloads （memory-bound application as the high-priority application）, HSM-QoS improves STP by 18.9%, 13.0%, and 15.2% compared to the baseline, Themis, and DASE, respectively.*

➢ 6 Evaluation

• 6.4 Memory Bandwidth Partitioning

图13给出了以HSM与内存带宽管理的相关工作BW-App的对比，这里只对比了Fairness和STP。

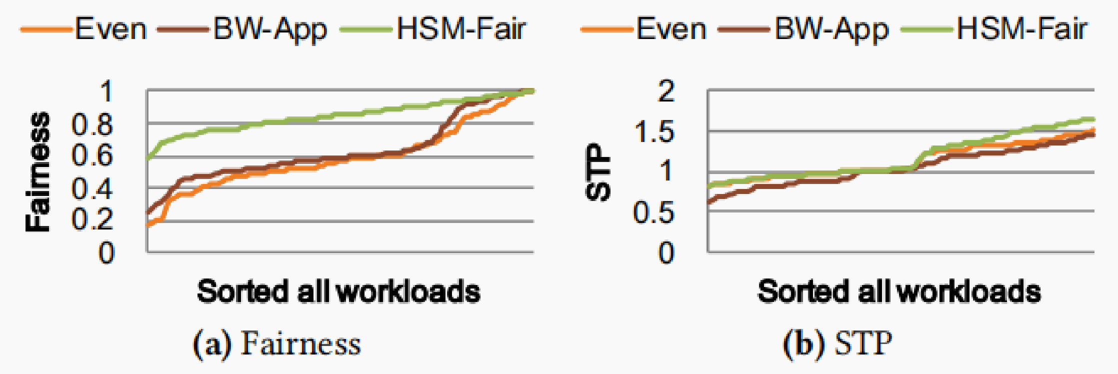


Figure 13. Fairness and STP compared to memory bandwidth partitioning. *HSM-Fair significantly improves fairness compared to memory bandwidth partitioning.*

(a) *For heterogeneous workloads, BW-App is not effective because the compute-bound application needs more SMs to improve fairness, not memory bandwidth. BW-App improves fairness by 5.3% on average, whereas HSM-Fair improves fairness by 55.5%*

(b) 好像忘记写了对b图的描述

➢ 6 Evaluation

• 6.5 Sensitivity Analyses

作者在这一部分想表达的是HSM模型可以在系统建模的不同参数上都有效

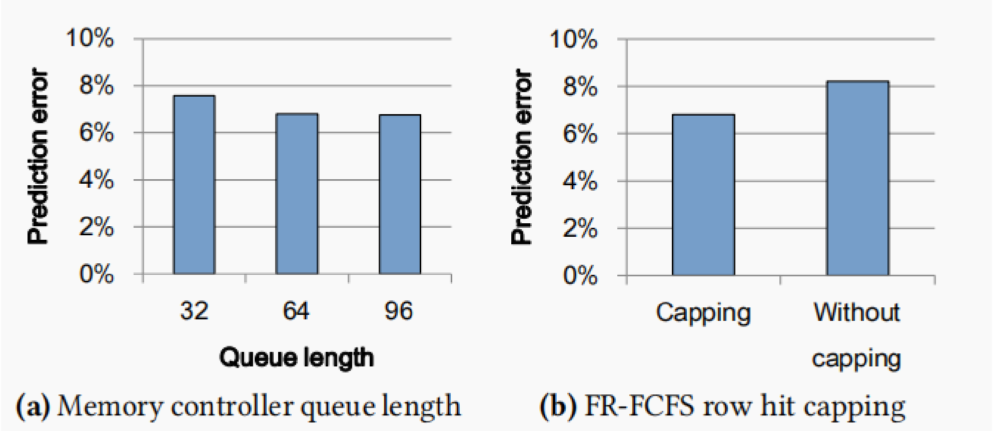


Figure 14. HSM accuracy sensitivity to memory controller parameters. *Overall, HSM's accuracy robust to different memory controller configurations.*

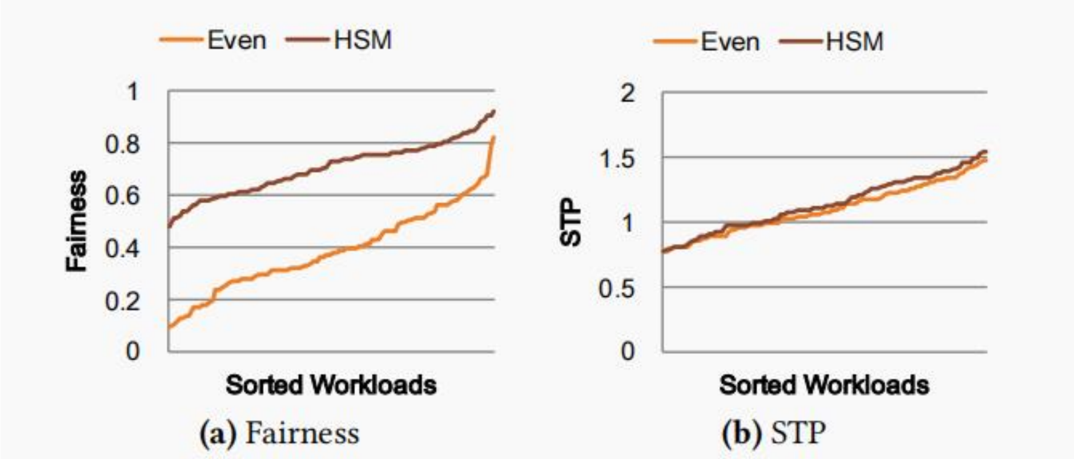


Figure 15. Fairness and STP for 4-program workloads. *Again, HSM-Fair significantly improves fairness.*

- Figure 14 改变Memory controller的参数对HSM预测准确率的影响
- Figure 15 添加新负载后对系统性能的影响

➢ 7 Related Work

- **Related Work作者分成三块来表述：第一块是slowdown模型；第二块是GPU resource management in multitasking GPUs；第三块是Memory bandwidth management**

我们以第二块内容为例子，学习related work的写法：
// 列举之前工作的核心思想
GPU resource management in multitasking GPUs: A number of prior proposals rely on heuristics to manage memory bandwidth in multitasking GPUs. Jog et al. [36, 41] propose application-aware memory schedulers, while Wang et al. [39] scale resources within an SM to manage memory bandwidth.

// 从之前的相关文献当中，我们学习到了什么或者他们的不足是什么
We find that (i) memory bandwidth partitioning does not provide fairness for heterogeneous workloads, and (ii) SM partitioning is a more effective solution managing both compute and memory bandwidth resources.

// 还有一类相关工作是做什么的
Another class of related work uses offline profiling to determine private-mode performance and dynamically allocate resources. Aguilera et al. [4, 42] adjust the number of SMs allocated to applications to improve fairness and QoS. Wang et al. [43] use fine-grained sharing of SM-internal resources to improve QoS. HSM would greatly benefit these approaches as users would no longer be required to retrieve representative private-mode performance numbers for all applications.

➢ 8 Conclusion

- **总结的内容主要由三句话组成**

// 这篇文章提出了什么，主要功能是什么
This paper presented the Hybrid Slowdown Model (HSM) to accurately predict the slowdown of co-running applications (average error of 6.8%) which provides a foundation for interference-aware SM resource allocation policies in multitasking GPUs.

// 列举系统中的核心创新点是什么，达到了什么目标
HSM combines a white-box model derived from fundamental architectural insights — to reduce training and implementation overheads — with a black-box model that accounts for the highly concurrent GPU execution model — to achieve high accuracy.

// 怎么衡量此系统的，指标是什么，实验场景是什么，比现有的baseline好多少
To showcase the capabilities of HSM, we proposed two HSM-based fairness and QoS-aware SM-allocation policies: HSM-Fair and HSM-QoS.
HSM-Fair improves fairness by 1.59× on average compared to even SM partitioning. HSM-QoS maintains the NP target of the high-priority application while improving system throughput by 18.9% on average compared to even SM partitioning for challenging heterogeneous workload mixes.

HSM 带给我们的思考

1. 从软件角度来看能否实现动态地SM分配?
答：目前纯软件是无法做到动态地SM分配。主要原因是目前没有提供给**软件层的抢占接口**，用户无法控制在GPU端的进程。

SM的静态分配现状：对于N卡，目前看到只能在context粒度限制SM资源用量；A卡比N卡在这方面更加细粒度一些，可以以stream为粒度来分配与调度Block到CU。

2. 通过作业切分来模拟作业的抢占过程，与之相关文献：
Asplos’17 FLEP: Enabling Flexible and Efficient Preemption on GPUs