Classification-Driven Search for Effective SM Partitioning in Multitasking GPUs

Xia Zhao
Ghent University, Belgium

Zhiying Wang
National University of Defense Technology, China

Lieven Eeckhout
Ghent University, Belgium

ABSTRACT

Graphics processing units (GPUs) feature an increasing number of streaming multiprocessors (SMs) with each successive generation. At the same time, GPUs are increasingly widely adopted in cloud services and data centers to accelerate general-purpose workloads. Running multiple applications on a GPU in such environments requires effective multitasking support. Spatial multitasking in which independent applications co-execute on different sets of SMs is a promising solution to share GPU resources. Unfortunately, how to effectively partition SMs is an open problem.

In this paper, we observe that compared to widely-used even partitioning, dynamic SM partitioning based on the characteristics of the co-executing applications can significantly improve performance and power efficiency. Unfortunately, finding an effective SM partition is challenging because the number of possible combinations increases exponentially with the number of SMs and co-executing applications. Through offline analysis, we find that first classifying workloads, and then searching an effective SM partition based on the workload characteristics can significantly reduce the search space, making dynamic SM partitioning tractable.

Based on these insights, we propose *Classification-Driven search (CD-search)* for low-overhead dynamic SM partitioning in multitasking GPUs. CD-search first classifies workloads using a novel off-SM bandwidth model, after which it enters the performance mode or power mode depending on the workload's characteristics. Both modes follow a specific search strategy to quickly determine the optimum SM partition. Our evaluation shows that CD-search improves system throughput by 10.4% on average (and up to 62.9%) over even partitioning for workloads that are classified for the performance mode. For workloads classified for the power mode, CD-search reduces power consumption by 25% on average (and up to 41.2%). CD-search incurs limited runtime overhead.

CCS CONCEPTS

• **Computer systems organization → Single instruction, multiple data**;

KEYWORDS

GPU, multitasking, SM partitioning

ICS '18, June 12–15, 2018, Beijing, China

ACM ISBN 978-1-4503-5783-8/18/06…$15.00
https://doi.org/10.1145/3205289.3205311

ACM Reference Format:
Xia Zhao, Zhiying Wang, and Lieven Eeckhout. 2018. Classification-Driven Search for Effective SM Partitioning in Multitasking GPUs. In *ICS '18: International Conference on Supercomputing, June 12–15, 2018, Beijing, China.* , 11 pages. https://doi.org/10.1145/3205289.3205311

1 INTRODUCTION

Graphics Processing Units (GPUs) are increasingly widely used to accelerate general-purpose computation. GPUs achieve high computational power by exploiting thread-level parallelism across a number of streaming multiprocessors (SMs). It is remarkable to note that the number of SMs increases with each successive generation. Whereas Nvidia's Fermi and Kepler architecture implement 14 SMs (Tesla M2050) and 15 SMs (Tesla K40), respectively, the next-generation Maxwell has 24 SMs (Tesla M40), and the latest Pascal and Volta architectures feature as many as 56 SMs (Tesla P100) and 80 SMs (Tesla V100), respectively.

This hardware trend coincides with the application trend towards using GPUs as accelerators in cloud services and data centers to meet the ever-growing demands while maintaining cost efficiency [1–4], as exemplified by Amazon EC2's offering of GPU instances in the cloud [5]. In such systems, multiple independent applications from different users need to be executed as efficiently as possible.

How to share GPU resources among different applications therefore becomes increasingly important. Spatial multitasking and simultaneous multitasking (SMK) are two previously proposed techniques to support multitasking on a GPU. In spatial multitasking, the SMs in a GPU are divided into disjoint subsets to which different applications are assigned to co-run [6, 7]. SMK on the other hand employs fine-grained sharing of SM resources by co-executing two applications on a single SM [8–10]. Commercial GPUs have started to support spatial multitasking by managing SM resources at the chip level [11]. Because spatial multitasking is simpler to implement than SMK, we focus on spatial multitasking in this paper, and compare against SMK in the evaluation section.

Even SM partitioning, in which all applications are given an equal share of the available SMs, is a widely used approach in spatial multitasking [6, 12–15]. The common wisdom is that even partitioning performs well on average [6, 13]. In this paper, we observe that because of wildly varying workload characteristics, uneven SM partitioning can bring substantial performance and power benefits compared to even partitioning. Unfortunately, with an increasing number of SMs and co-executing applications, the number of possible combinations to partition the SMs quickly explodes. Identifying the optimal SM partition in an effective way is a major challenge.

Instead of exhaustively exploring all possible combinations, we find, through detailed offline analysis, that workload classification

along with workload-specific search significantly reduces the overhead for identifying the optimal SM partition. Based on this insight, we propose *Classification-Driven search (CD-search)* to dynamically determine the optimal SM partition in a low-overhead way during run time. In particular, CD-search operates as a two-step process. In the first step, CD-search classifies the co-executing applications as memory-intensive versus compute-intensive based on their performance sensitivity to SM count. In particular, a memory-intensive application sees its performance saturate with an increasing number of assigned SMs; a compute-intensive application on the other hand benefits an almost linear performance increase with the number of SMs. After this initial classification step, the second step steers SM partitioning based on the workload characteristics. For workloads consisting of a mix of memory-intensive and compute-intensive applications, CD-search enters the *performance mode* to find the SM partition that maximizes performance. For workloads consisting of memory-intensive applications, CD-search enters the *power mode* to find the SM partition that assigns the least number of SMs to maintain performance while power-gating the remaining SMs to save power. For workloads consisting of compute-intensive applications, CD-search maintains even partitioning as there is no opportunity to optimize performance nor power.

In the initial classification step, applications are classified as memory-intensive versus compute-intensive based on their sensitivity to the number of assigned SMs. We find that a previously proposed (and widely used) metric, namely memory bandwidth utilization [12, 16, 17], is not an accurate predictor for SM performance sensitivity because it only focuses on main memory bandwidth utilization while not considering NoC and LLC bandwidth demands. To accurately classify workloads, we propose the *off-SM bandwidth model*. The model not only considers the bandwidth demands of the executing applications but it also takes off-SM bandwidth into account, including NoC, LLC and memory bandwidth.

We make the following contributions in this paper:

- We show that compared to widely-used even partitioning, uneven SM partitioning can bring substantial performance and power benefits.
- We propose CD-search to dynamically determine the optimum SM partition at low overhead. CD-search first classifies the applications in the workload mix, after which the optimum SM partition is determined. The optimization target (performance versus power) as well as the search strategy towards the optimum SM partition depends on the workload's characteristics.
- We find that memory bandwidth utilization is an inaccurate predictor for how sensitive an application is to the number of assigned SMs. The newly proposed off-SM bandwidth model takes NoC, LLC and main memory bandwidth into account to more accurately classify workloads.
- We comprehensively evaluate CD-search. Compared to a GPU with even SM partitioning, CD-search improves system throughput by 10.4% on average (and up to 62.9%) for multitasking workloads classified as heterogeneous mixes of compute-intensive and memory-intensive applications. For workloads classified as homogeneous mixes including only memory-intensive applications, we report an average 25% reduction in power consumption (and up to 41.2%).

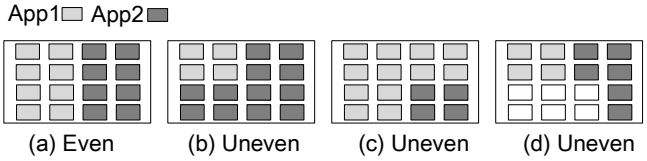


Figure 1: Four examples illustrating how to partition a GPU with 16 SMs across two applications. (a), (b) and (c) use all SMs; (d) only uses a subset of the SMs. *The number of combinations increases exponentially with the number of SMs and co-executing applications.*

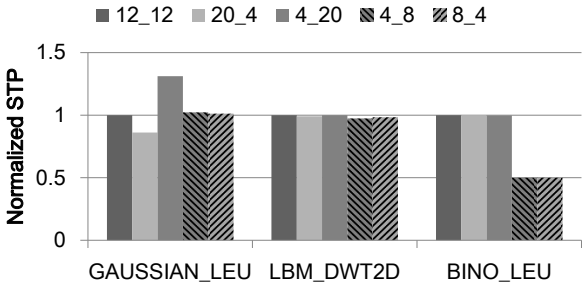


Figure 2: System performance (STP) for different SM partitions normalized to even SM partitioning. 'X_Y' in the legend refers to the number of SMs assigned to the left and right application in the workload mix, respectively. *Uneven SM partitioning affects system performance differently for different workload mixes.*

2 MOTIVATION

As shown in Figure 1, when multiple applications co-execute on a GPU, there are many ways to partition the available SMs. One could evenly partition the SMs across the co-executing applications (Figure 1(a)). When employing an uneven partition, there are many possible combinations (Figure 1(b) and (c)). One could even opt to not use all SMs and leave some SMs idle and power-gate them to save power (Figure 1(d)). Prior work in spatial multitasking assumes even partitioning of the available SMs among the co-executing applications, e.g., for two co-executing applications, each application is assigned half the number of SMs [6, 12–15]. In fact, Adriaens et al. [6] and more recently Park et al. [13] argue that even SM partitioning performs well on average. In this paper, we make the observation that different workloads show different performance sensitivity to uneven SM partitioning, which we exploit to significantly improve performance and reduce power.

Figure 2 reports performance (normalized system throughput) for three workload mixes under different SM partitions on a GPU with a total of 24 SMs. (Details on the experimental setup are provided in Section 6.) 12_12 represents even SM partitioning, i.e., each application gets assigned 12 SMs. Uneven partitioning, i.e., 20_4 and 4_20, assigns 20 versus 4 SMs to the lefthand application, respectively. In addition, we also include 4_8 and 8_4 to represent uneven partitioning while considering only 12 out of 24 SMs; the remaining 12 SMs are left idle. The key message from Figure 2 is that the effect of SM partitioning depends on the workload and leads to three optimization opportunities:

- **Performance opportunity:** For GAUSSIAN_LEU, a workload mix that includes GAUSSIAN and LEU, performance

can be significantly improved through uneven SM partitioning compared to even partitioning, i.e., performance is the highest for the 4_20 configuration. However, an unsuitable uneven SM partition, e.g., 20_4, may degrade performance considerably. This indicates that *for workloads that are amenable to uneven SM partitioning, finding an effective partition is key.*

- **Power opportunity:** For LBM_DWT2D, uneven SM partitioning has no impact on performance. Even reducing the number of assigned SMs by half (see configurations 4_8 and 8_4) does not affect performance. This indicates that *for some workloads, there is no need to use all SMs, hence we can turn off a number of SMs and save power.* The question then is how many SMs to turn off to not degrade performance.
- **No opportunity:** For BINO_LEU, the 20_4 and 4_20 configurations have no impact on overall system performance, however, halving the number of assigned SMs (see 4_8 and 8_4) degrades performance by nearly 50%. This indicates that, *for some workloads, there are no performance nor power optimization opportunities* beyond even partitioning. In such a case, there is no need to incur the overhead for searching an effective SM partition.

In summary, we conclude that uneven SM partitioning creates an opportunity to improve performance and power efficiency compared to even partitioning. How to decide the optimization goal, i.e., optimize for performance or power, and how to find an effective SM partition in a low-overhead way is the key focus of this paper.

3 WORKLOAD CLASSIFICATION

In this section, we show that workloads can be classified into three categories depending on the characteristics of the applications that the workload is composed of. We do so through detailed offline analysis. Applications can be classified as memory-intensive or compute-intensive depending on their performance sensitivity to the number of SMs. When independent applications co-execute on a GPU, this leads to workload mixes with different characteristics. We could have a heterogeneous workload consisting of compute- and memory-intensive applications, or the workload could be homogeneous with only compute-intensive applications or only memory-intensive applications. As we will detail in this section, the opportunity is different for each of these three workload mixes. In addition, we find that determining the optimum SM partition should follow a different search strategy for each of the three mixes, hence the notion of classification-driven search.

3.1 Heterogeneous Workload Mixes

Uneven SM partitioning can significantly improve performance for heterogeneous workloads consisting of a mix of compute-intensive and memory-intensive applications. This is illustrated in Figure 3 which reports how system performance changes as a function of the number of SMs assigned to the memory-intensive versus compute-intensive applications for three example heterogeneous workload mixes. The horizontal axis reports the number of SMs assigned to the memory-intensive application (which is the first application for each application tuple in the legend); the remaining SMs are assigned to the compute-intensive application. We observe that overall system performance, measured using the system throughput

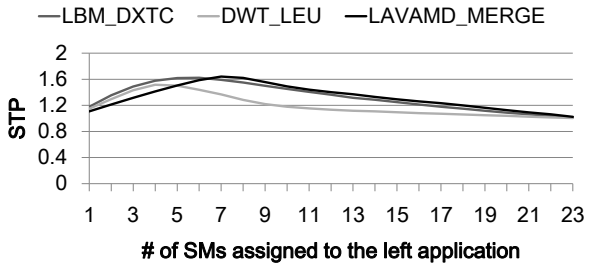


Figure 3: System performance (STP) as a function of the number of SMs assigned to the memory-intensive application in three example heterogeneous workload mixes. *Heterogeneous workload mixes provide opportunity to improve system performance.*

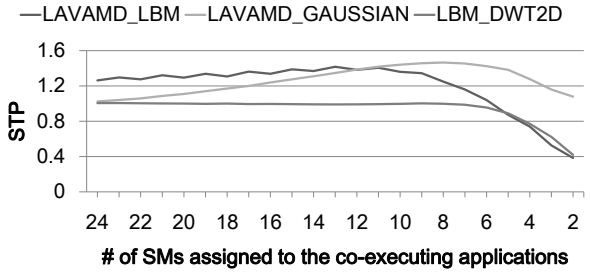


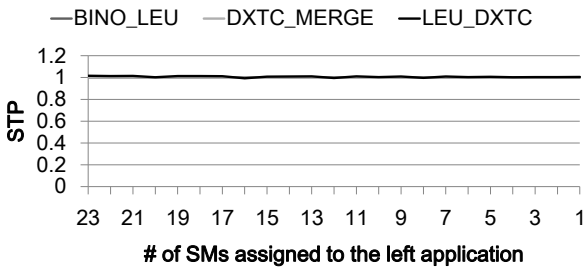
Figure 4: System performance (STP) as a function of the total number of assigned SMs, assuming even partitioning for memory-intensive workload mixes. *Memory-intensive workload mixes provide the opportunity to save power by power-gating unused SMs while not hurting and in some cases even improving overall system performance.*

(STP) metric, increases with the number of SMs assigned to the memory-intensive application until it reaches an optimum around 4 to 7 SMs; this implies that the compute-intensive application gets assigned the remaining 17 to 20 SMs out of a total of 24 SMs. The obtained performance benefit is significant. Compared to executing both applications one after the other, i.e., no spatial multitasking, which leads to an STP of 1, we obtain an improvement in system performance by up to 64% for the LAVAMD_MERGE workload mix. Compared to spatial multitasking with even SM partitioning, i.e., each application gets assigned 12 SMs, we observe a performance improvement up to 38% for the DWT_LEU workload mix.

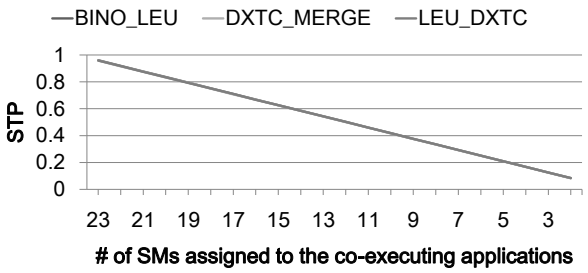
From these illustrative examples, we conclude that (i) overall system performance can be significantly improved through SM partitioning for heterogeneous workload mixes, and (ii) system performance is maximized by assigning a relatively small fraction of the available SMs to the memory-intensive application (i.e., the optimum appears on the left-hand side of the graph in Figure 3) while assigning the remaining SMs to the compute-intensive application. This second observation will help us reduce the search space for finding the optimum SM partition.

3.2 Memory-Intensive Workload Mixes

For workload mixes consisting of memory-intensive applications, uneven SM partitioning provides an opportunity to save power, without affecting performance; and, in some cases, even improving performance. This is illustrated in Figure 4, which reports system



(a) STP as a function of the number of SMs assigned to the left application



(b) STP as a function of the total number of allocated SMs

Figure 5: System performance as a function of the total number of assigned SMs for compute-intensive workload mixes: *There is no opportunity to improve performance and power efficiency through SM partitioning and allocation.*

performance (STP) as a function of the number of SMs allocated for three example memory-intensive workload mixes. We assume even partitioning but we decrease the total number of allocated SMs from left to right. We observe that performance remains stable or even increases with decreasing number of allocated SMs until we reach the saturation point after which performance quickly deteriorates. This point of saturation is reached for a relatively small (7 to 11) total number of SMs. This provides an opportunity to power-gate the remaining (17 to 13, respectively) unallocated SMs to save power. Note that the optimum number of SMs depends on the co-executing applications. When co-executing LBM with DWT2D, the optimum number of SMs per application equals 3; however, when co-executing LBM with LAVAMD, the optimum number of SMs equals 5. For LAVAMD, when co-executed with GAUSSIAN, optimum performance is achieved with 4 SMs rather than 5; this is a result of significant cache contention due to the GAUSSIAN benchmark.

From these illustrative examples, we conclude that (i) when co-executing memory-intensive applications, there is no need to allocate all available SMs in the GPU — optimum performance is achieved when allocating a fraction of the available SMs, which creates an opportunity to save power; (ii) the number of SMs assigned to each memory-intensive application depends on the co-executing application and hence needs to be determined dynamically; and (iii) reducing the number of active SMs sometimes leads to a performance boost due to reduced cache contention.

3.3 Compute-Intensive Workload Mixes

Finally, for workload mixes consisting compute-intensive applications, there is no need to search for an effective SM partition as there is no opportunity to improve performance or save power. Compute-intensive application performance is linear in the number of assigned SMs. De-allocating SMs from one application degrades its performance proportionally, whereas the other application improves its performance proportionally — this leads to a net (system-wide) performance-neutral operation. This is illustrated in Figure 5(a) which shows system performance (STP) as a function of the number of SMs assigned to one of the two compute-intensive applications; the other application gets allocated the remaining set of SMs. We observe a flat performance curve — there is no opportunity to improve system performance. Figure 5(b) shows system performance as a function of the total number of SMs assigned to both applications. Here we observe a linear decrease in performance — there is no opportunity to save power without severely degrading performance.

3.4 Towards Classification-Driven Search

The key take-away message from the discussion so far is that workload classification can help us to decide the optimization goal, to either improve performance or save power. Moreover, the above discussion also provides us a way to steer the search for the optimum SM partition in an efficient way based on the workload's characteristics. Based on these insights, we next propose classification-driven search.

4 CLASSIFICATION-DRIVEN SEARCH

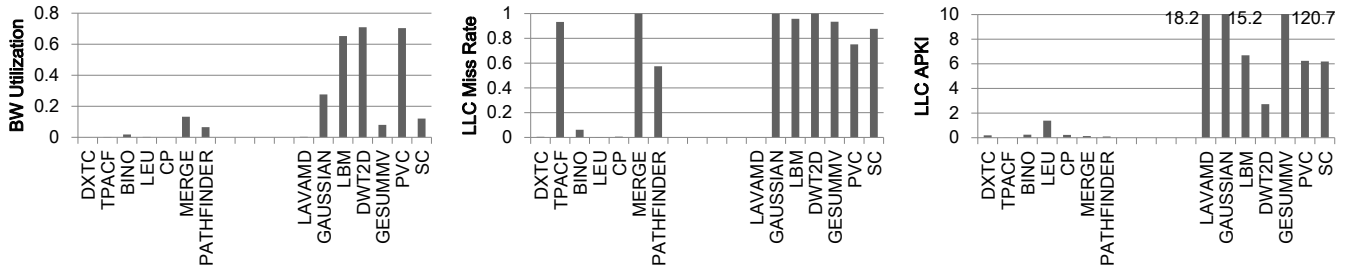
Classification-Driven search (CD-search) dynamically finds an effective SM partition that optimizes either performance or power depending on the characteristics of the workload at hand. CD-search includes an initial phase during which the co-running applications are characterized; next, depending on the workload mix's characteristics, we enter the performance mode (for workload mixes that are classified as heterogeneous) or the power mode (for workload mixes that are classified as memory-intensive); or we employ even partitioning for workload mixes that are classified as compute-intensive. CD-search incurs low runtime overhead.

The initial workload characterization phase works because although GPU applications exhibit phase behavior at the warp level [18, 19], this gets leveled out as several TBs execute concurrently [20]. After the classification phase, CD-search chooses to enter the performance mode or the power mode. After finding the effective SM partition, SMs are preempted; preempted SMs are then re-assigned or power-gated depending on whether CD-search enters the performance or power mode. Upon the arrival of a new application, CD-search restarts the classification phase to profile the newly arrived application and adjust the SM partition accordingly. We now describe the two phases in CD-search in more detail. Note we assume two applications in this description but the algorithm is trivially extended to more than two applications.

4.1 Workload Classification

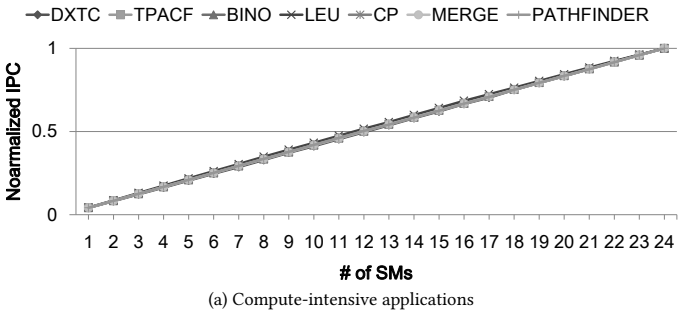
The first phase in CD-search is to classify applications based on their characteristics. This is a critical step because misclassification may lead to a performance penalty and/or excess power consumption.

4.1.1 Need for a novel, low-overhead solution. The main goal of the workload classification phase is to determine which mode to enter in the next phase. One straightforward way to classify

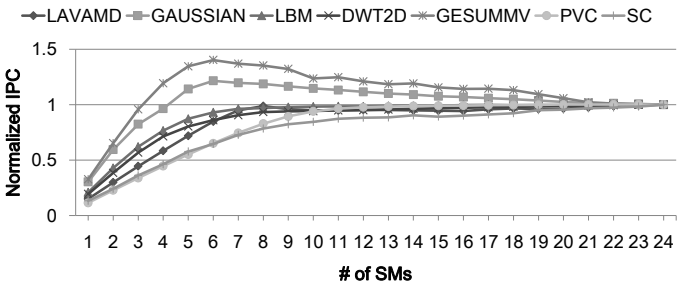


(a) Memory Bandwidth Utilization (b) LLC Miss Rate (c) LLC APKI

Figure 6: Memory-related performance characteristics: (a) memory bandwidth utilization, (b) LLC miss rate, and (c) LLC APKI. *Memory bandwidth utilization by itself is not an accurate application classifier.*



(a) Compute-intensive applications



(b) Memory-intensive applications

Figure 7: Performance as a function of the number of allocated SMs. *Performance improves linearly with increasing SM count for the compute-intensive applications (a) but saturates for the memory-intensive applications (b).*

workloads would be to try out different SM counts, measure performance, and determine its type based on the performance sensitivity to SM count. Unfortunately, this is too high overhead as it may take several iterations to reach the optimum, with each iteration requiring preempting a number of SMs.

An alternative approach would be to use bandwidth utilization, a widely used metric, see for example [12, 16, 17]. Bandwidth utilization is defined as the fraction of DRAM cycles during which a read or write request is serviced. We find that bandwidth utilization is not enough though to accurately classify memory-intensive versus compute-intensive applications. Figure 6 shows three memory-related performance metrics for the different applications considered in this study: (a) memory bandwidth utilization, (b) LLC miss rate, and (c) LLC accesses per kilo instructions (LLC APKI). LAVAMD, a memory-intensive application, has nearly zero memory bandwidth utilization (Figure 6), however, its performance saturates with increasing number of SMs (Figure 7(b)). The reason is that LAVAMD has a very high LLC APKI; its LLC miss rate is close to zero which explains the low memory bandwidth utilization, yet still the available LLC bandwidth is not enough to satisfy the demand when engaging all SMs. Take another example, namely MERGE which is a compute-intensive application: its bandwidth utilization equals 13.3%, which is slightly higher than for some of the memory-intensive applications (Figure 6). However, its performance does not saturate with the number of SMs, in fact, it increases linearly (Figure 7(a)). The reason is that its LLC APKI is low, hence total bandwidth demand from all SMs cannot saturate the memory system. This indicates that focusing on the attained memory bandwidth alone is not accurate; LLC cache utilization and bandwidth demand should also be considered when classifying applications.

4.1.2 Off-SM Bandwidth Model. To accurately classify applications with respect to their sensitivity to the number of assigned SMs, we propose the *off-SM bandwidth model* which builds upon two metrics, namely *SM bandwidth demand* (BW_{Total_SMs}) and *off-SM bandwidth* (BW_{Off_SM}). As shown in Figure 8, in a conventional GPU, many SMs connect to the LLC and DRAM through a network-on-chip (NoC). The NoC, LLC and DRAM make up the 'off-SM' system which supplies the data for the application executing on the SMs. If the off-SM system cannot meet the bandwidth demands of a memory-intensive application, performance will saturate with an increasing number of SMs.

$$\begin{aligned} BW_{Total\_SMs} &= BW_{Per\_SM} \times \#SMs \\ BW_{Per\_SM} &= IPCmax \times APKI_{LLC} \times Size_{CacheLine} \times Freq_{SM} \end{aligned} \tag{1}$$

$$\begin{aligned} BW_{Off\_SM} &= \min\{BW_{NoC}, BW_{LLC\_MEM}\} \\ BW_{LLC\_MEM} &= BW_{LLC} \times HitRate_{LLC} + \\ &\quad BW_{MEM} \times MissRate_{LLC} \times BW_{Util} \end{aligned} \tag{2}$$

In Equation 1, BW_{Total_SMs} represents the total bandwidth demand for a particular number of SMs in the ideal case where the SMs can issue instructions without any stalls. BW_{Per_SM} is the bandwidth demand of a single SM and is a function of $IPCmax$ (max number of instructions issued per cycle without any stalls), $APKI_{LLC}$ (LLC accesses per kilo instructions), $Size_{CacheLine}$ (cache line size) and $Freq_{SM}$ (SM operating frequency). $IPCmax$, $Size_{CacheLine}$ and

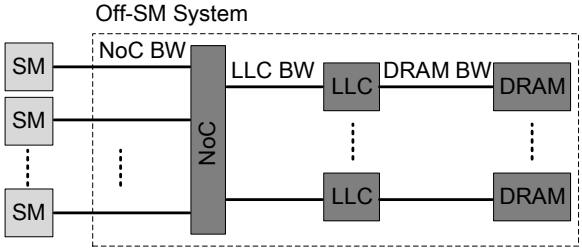


Figure 8: GPU off-SM bandwidth model.

$Freq_{SM}$ are related to the GPU hardware, whereas $APKI_{LLC}$ is a function of both the application and the hardware. Equation 2 represents the bandwidth that the off-SM system can supply. The available off-SM bandwidth is computed as the minimum of the NoC bisection bandwidth and the available LLC/DRAM bandwidth. The latter is computed as the raw LLC bandwidth derated by the LLC cache hit rate (to obtain the effective LLC bandwidth) plus the raw main memory bandwidth derated by the LLC miss rate times the effective main memory bandwidth utilization (which is assumed to be 50% in this work for simplicity). All the values in the above equations are hardware related and are determined ahead of time, except for $APKI_{LLC}$, $HitRate_{LLC}$ and $MissRate_{LLC}$ which are determined through online profiling and can be collected using hardware performance counters. If the application's bandwidth demand is higher than the available off-SM bandwidth, we classify the application as memory-intensive. Otherwise, the application is classified as compute-intensive.

Application classification is done online, while the workload is running on the GPU. Classification is a two-phase process including a warmup phase followed by a profiling phase; both phases take 20K cycles. The warmup phase serves two purposes: it warms up the microarchitecture structures for the profiling phase, and in addition, it is used to determine the preemption policy. If a application can finish a TB's execution during the warmup phase, it is reasonable to assume that other TBs will also finish soon, hence a draining policy will be followed to preempt the SMs occupied by this application. If not, CD-search will employ context switching [7, 21]. We assign half the total number of SMs to each of the co-executing applications during the classification phase and we compute the above two equations for each of the co-executing applications. This enables us to classify the applications as memory-intensive or compute-intensive.

4.2 Performance Mode

CD-search enters the performance mode for heterogeneous workload mixes. The goal is to find an effective SM partition by assigning the minimum number of SMs to the memory-intensive application to maintain its performance, while assigning the remaining SMs to the compute-intensive application to maximize its performance.

We iteratively stall an increasing number of SMs (in steps of two) for the memory-intensive application in subsequent iterations and measure its performance impact in terms of IPC. Each iteration takes 40K cycles (20K cycles for warmup followed by 20K cycles for profiling). This profiling phase ends when the performance loss for the memory-intensive application is no more than a predetermined threshold compared to running with half the total number of SMs. We empirically set the threshold to 5%. After this profiling phase, CD-search determines the optimum SM partition and preempts the stalled SMs from the memory-intensive application and allocates these SMs to the compute-intensive application.

The profiling overhead is greatly reduced by gradually stalling, and not preempting, an increasing number of SMs for the memory-intensive application. Compared to even SM partitioning, this strategy incurs minimal performance loss because the memory-intensive application is insensitive to the number of assigned SMs anyway up until the saturation point. An alternative implementation would be to preempt and re-assign SMs among the co-running applications during each iteration of the profiling phase. Gradually stalling SMs greatly reduces the profiling overhead.

Note that this algorithm is simple, yet effective. The reason why it works is because we accurately classify applications in the workload classification phase. Without the workload classification phase, one would need to explore all possible combinations, or at least a lot more combinations, to identify the optimum SM partition, which would incur significant overhead, up to the point where the overhead offsets the performance benefit. Through workload classification, we are able to dramatically reduce the search space and quickly reach the optimum SM partition.

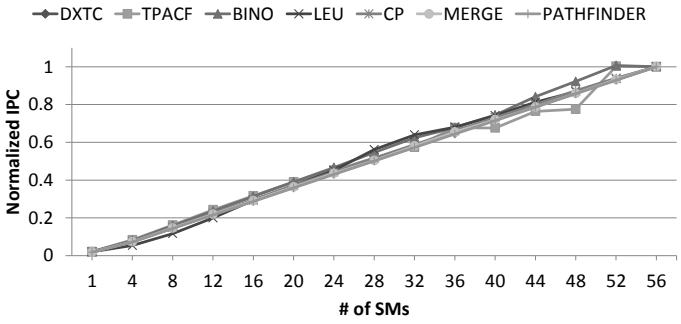
4.3 Power Mode

CD-search engages the power mode for homogeneous workload mixes consisting of only memory-intensive applications. CD-search aims at finding the SM partition that assigns the least number of SMs to both the memory-intensive applications to not degrade their performance compared to even SM partitioning; the unused SMs are then power-gated to save power.

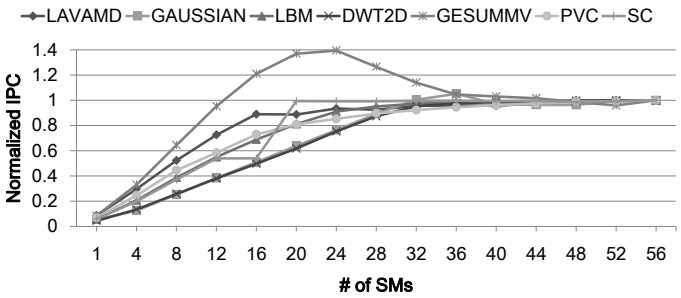
It is important to quickly determine the optimum number of SMs for each memory-intensive application — an inefficient profiling phase incurs excess power consumption for no performance benefit. To do so, we employ a more aggressive strategy than for the performance mode. Instead of gradually increasing the number of stalled SMs, we try to converge more quickly. In particular, we identify two stages. In the first stage, we stall all but one of the SMs for both applications, and measure their respective performance. (This takes 40 K cycles: 20 K cycles for warmup and 20 K cycles for profiling.) Based on the performance ratio for 12 SMs versus 1 SM, we compute a tentative optimum number of SMs for each application, and measure performance (again, this takes 40K cycles). In the second stage, if performance is within 95% of the performance at half the total number of SMs, we preempt the stalled SMs and power-gate them. If not, we iteratively resume one SM at a time until the application's performance is within the 95% range, after which we preempt and power-gate the unused SMs.

5 REAL HARDWARE VALIDATION

It is impossible to evaluate CD-search on current hardware. However, we can verify whether applications exhibit compute-intensive versus memory-intensive execution behavior similar to what we observe in simulation. This is to verify that the observed performance phenomena are not an artifact of our simulation infrastructure. We consider the NVIDIA Tesla P100 (Pascal GPU architecture) [22] which features 56 SMs and 732 GB/s memory bandwidth. Similar to what we do in the simulator, we change the number of allocated SMs and quantify the impact on performance. We consider larger



(a) Compute-intensive applications



(b) Memory-intensive applications

Figure 9: Performance as a function of the number of allocated SMs on the NVIDIA P100 GPU. *Similar performance trends are observed on real hardware compared to simulation, see Figure 7.*

input data sets in the real hardware measurements than what we use in simulation because of the larger the on-chip caches in the real hardware.

Listing 1: Controlling SM count on real GPU hardware.

```
__device__ uint get_SMid(void)
{
  uint ret;
  asm("mov.u32 %0, %SMid;" : "=r"(ret));
  return ret;
}

 __global__ void gpu_uSleep()
 {
  // SM_count decides the number of SMs
  // assigned to the MAIN_KERNEL
  if(get_SMid() < SM_count)
      ;
  else
      Sleep();
}

int main(void)
{
  cudaStream_t cuda0, cuda1;
  cudaStreamCreateWithFlags(&cuda0, cudaStreamNonBlocking);
  cudaStreamCreateWithFlags(&cuda1, cudaStreamNonBlocking);
  // 1024 is the maximum threads per TB in P100
  dim3 block_uSleep(1024);
  // 112 is number of TBs which just occupies 56 SMs in P100
  dim3 grid_uSleep(112);
  gpu_uSleep<<< grid_uSleep, block_uSleep, 0, cuda0>>>;
  MAIN_KERNEL<<< #GRID, #BLOCK, 0, cuda1>>>;
}
```

Table 1: Benchmarks used in this study.

Benchmark	*Grid Dim*	*Type*
DirectX Texture Compressor (DXTC) [25]	(16384,1,1)	Compute
Two Point Angular Correlation Function (TPACF) [26]	(201,1,1)	Compute
BinomialOptions (BINO) [25]	(512,1,1)	Compute
Leukocyte (LEU) [27]	(596,1,1)	Compute
Coulombic Potential (CP) [23]	(32,128,1)	Compute
MergeSort (MERGE) [25]	(4096,1,1)	Compute
PATHFINDER [27]	(9260,1,1)	Compute
LAVAMD [27]	(1000,1,1)	Memory
Gaussian Elimination (GAUSSIAN) [27]	(512,512,1)	Memory
Lattice-Boltzmann Method (LBM) [26]	(120,150,1)	Memory
DWT2D [27]	(97824,1,1)	Memory
GESUMMV [28]	(128,1,1)	Memory
Page View Count (PVC) [29]	(46875,1,1)	Memory
Streamcluster (SC) [27]	(512,1,1)	Memory

Controlling the number of SMs assigned to an application is an interesting problem by itself. We exploit concurrent streaming to do so and create a shadow kernel that occupies a specific number of SMs. Listing 1 shows the (simplified) source code. The shadow kernel, *gpu_uSleep*, assigns a number of SMs to the application of interest while putting the other SMs to sleep. *MAIN_KERNEL* is the application for which we want to measure its performance sensitivity to SM count. The application and the shadow kernel are assigned to two different CUDA streams for execution. In the *gpu_uSleep* kernel, a TB checks the ID of the SM on which it runs, and determines whether to sleep, while occupying the SM, or leave the SM to the *MAIN_KERNEL*. If the SM ID is smaller than a predetermined number of assigned SMs (*SM_count*), the SM is assigned to *MAIN_KERNEL*; otherwise the SM is put to sleep. This mechanism enables us to control the number of SMs assigned to the application of interest and evaluate its sensitivity to SM count.

Figure 9 reports performance as a function of the number of assigned SMs from 1 to 56, in steps of 4. Note that the results reported here on real hardware are quite similar to the results obtained through simulation as shown in Figure 7. This confirms that the performance phenomena observed in simulation also occur on real hardware, i.e., performance increases linearly with increasing SM count for the compute-intensive applications whereas it saturates for the memory-intensive applications. These real hardware validation results further motivate CD-search.

6 EXPERIMENTAL SETUP

Simulated System. We use a modified version of GPGPU-sim v3.2.2 [23] to evaluate CD-search. The modifications allow GPGPU-sim to run multiple applications concurrently through spatial multitasking. We consider a GPU with 24 SMs with a private 16 KB L1 cache connected through a crossbar to 6 memory controllers with two 128 KB LLC slices each. To estimate power consumption, we rely on GPUWattch [24] assuming a 40 nm technology node.

Workloads. We use a wide range of GPU-compute benchmarks implemented in CUDA. These benchmarks are selected from Rodinia [27], Parboil [26], CUDA SDK [25], PolyBench [28], GPGPU-sim [23], and Mars [29], and are listed in Table 1. We classify these benchmarks into two categories based on how performance changes with SM count, see Figure 7. The multi-application workloads are

constructed by pairing all the benchmarks, i.e., we consider all possible combinations; this yields a total of 91 multi-application workload mixes. Among these 91 workload mixes, 49 are heterogeneous, 21 are homogeneous memory-intensive and 21 are homogeneous compute-intensive.

Performance and Power Metrics. During multi-program execution, we simulate two million cycles for each benchmark — this is in line with prior GPU multitasking research [9, 10], and we confirm that this is representative. If a benchmark finishes before others, it is re-launched and re-executed from the beginning. The reported performance results are gathered only for the first run for each of the benchmarks. Throughout the evaluation, we use the metrics named, system throughput (STP) and average normalized turnaround time (ANTT) [30], to measure multi-application performance. STP takes a system's perspective and quantifies overall system performance (higher-is-better). ANTT takes a user's perspective and quantifies average per-application execution time (lower-is-better).

7 EVALUATION

We now evaluate CD-search. We first focus on the performance mode and the power mode. We then quantify profiling overhead, and perform a number of sensitivity studies with respect to memory bandwidth, number of SMs and number of co-executing applications. Finally, we compare against SMK.

Before diving into the performance and power numbers it is important to note that CD-search accurately classifies workloads. For the 91 multi-program workload mixes considered in this work, the off-SM bandwidth model achieves 100% accuracy for classifying applications as either memory-intensive or compute-intensive. Because of the accurate workload classification, we can now confidently focus on the 70 workload mixes that are amenable to the performance and power modes, and exclude the homogeneous workload mixes consisting of only compute-intensive workloads. Note, that memory bandwidth utilization, assuming the best performing threshold for our set of benchmarks, would miss-classify 36 out of 91 workload mixes.

7.1 Performance Mode

Recall that the performance mode is engaged for workloads that are classified as heterogeneous mixes of memory-intensive and compute-intensive applications. Under the performance mode, a smaller number of SMs is assigned to the memory-intensive application, which does not affect its performance. The unused SMs are then assigned to the compute-intensive application, boosting its performance. This leads to an improvement in overall system throughput and per-application performance. Note that the memory-intensive application does not (unfairly) slow down.

Figure 10 reports STP and ANTT delta compared to even SM partitioning. (A positive STP delta and negative ANTT delta is desirable because this means that system throughput and per-application performance is improved, respectively.) The workloads are sorted along the horizontal axis. These workload mixes substantially benefit from SM partitioning; there is a slight performance degradation (because of profiling overhead) for a couple workload mixes that do not benefit from partitioning. Overall, CD-search improves system performance by 10.4% on average (and up to 62.9%) compared to even SM partitioning. For only a few workloads does CD-search lead to a small performance degradation of at most 2.1%. At the same time, CD-search substantially improves per-application performance, i.e., ANTT improves by 22% on average (and up to 199%).

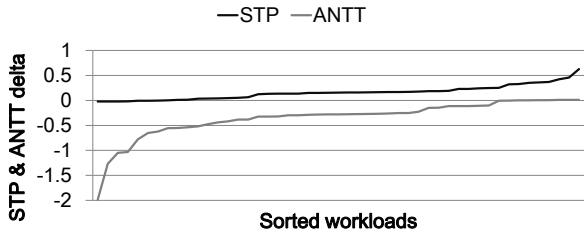


Figure 10: Performance mode: STP and ANTT delta over even SM partitioning. *The performance mode significantly improves STP (positive STP delta) and ANTT (negative ANTT delta) for heterogeneous workload mixes.*

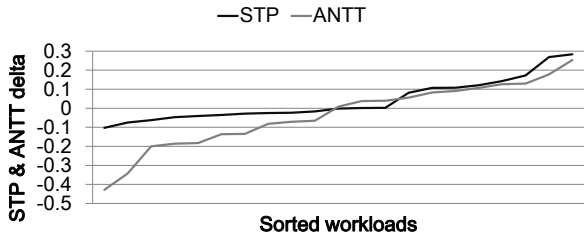


Figure 11: Power mode: STP and ANTT delta over even SM partitioning. *The power mode is performance-neutral on average.*

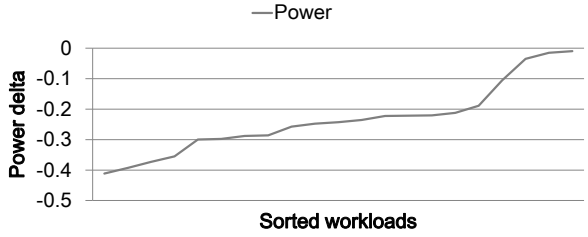
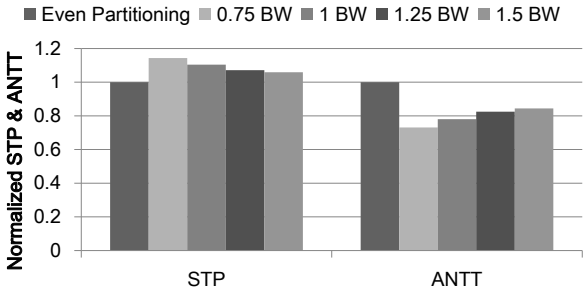


Figure 12: Power mode: power delta over even SM partitioning. *The power mode significantly reduces power consumption.*

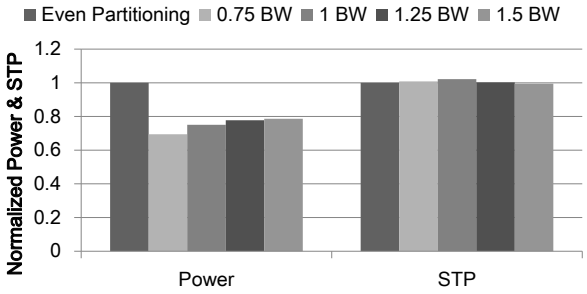
7.2 Power Mode

The power mode is engaged for homogeneous memory-intensive workload mixes. The goal here is to find the effective SM partition and save power by power-gating unused SMs while preserving performance. On average across all memory-intensive workload mixes, CD-search is performance-neutral (and even slightly improves STP and ANTT on average, by 2.2% and 2.8%, respectively), see Figure 11. For some workloads, CD-search degrades performance (by 10.3% at most), and for some workloads, CD-search significantly improves performance (up to 28.3%). The latter is due to cache contention for the GAUSSIAN and GESUMMV benchmarks as previously discussed in Section 3.2, i.e., better performance is achieved with fewer active SMs. Performance degradation is observed for workloads, e.g., LBM, for which the IPC measured during the workload classification phase is a somewhat inaccurate prediction for the subsequent execution because of time-varying execution behavior.

The real purpose of the power mode is to find an effective SM partition that saves power. On average, CD-search power-gates half the total number of SMs, which leads to a significant reduction in power consumption as shown in Figure 12. On average, the SM



(a) Performance mode: Average STP and ANTT



(b) Power mode: Average power and STP

Figure 13: Sensitivity analysis: memory bandwidth. *The opportunity to improve performance and save power increases as memory bandwidth is constrained.*

partition found by CD-search reduces power consumption by 25% on average (and up to 41.2%).

7.3 Profiling overhead

Recall CD-search incurs a runtime profiling phase to classify workload mixes and determine the optimum number of SMs for each application in the mix. Because each of the co-executing applications is making forward progress during profiling, we consider the following procedure to quantify the profiling overhead. We compare CD-search against a setup in which we determine the optimum SM partition through offline analysis. The relative performance difference then is a measure for the profiling overhead. We find the profiling overhead to be small though: 0.92% on average for performance and 2.5% for power.

7.4 Sensitivity Analyses

We now perform a number of sensitivity analyses to better understand CD-search's effectiveness.

7.4.1 Memory bandwidth. To evaluate the impact of memory bandwidth, we vary memory bandwidth in relation to the available memory bandwidth in our baseline architecture, i.e., 75%, 100%, 125% and 150% of the baseline memory bandwidth; the other parts of the system are kept unchanged. Figure 13a reports STP and ANTT for the performance mode; Figure 13b reports STP and power consumption for the power mode. The overall conclusion is that the opportunity is larger for constrained memory bandwidth: the performance gain and power gain decreases with increasing memory bandwidth. At reduced memory bandwidth, the memory-intensive application gets assigned fewer SMs, leaving more SMs for the compute-intensive application, which leads to a higher overall system throughput. The inverse is true when more memory bandwidth is available. At 75% the default memory bandwidth, CD-search improves STP by 14.4% on average (and up to 69%) and improves ANTT by 26.9% on average (and up to 229%). At 150% the default memory bandwidth, STP and ANTT improve by 5.9% and 15.6% on average, respectively. A similar trend is observed for the power mode: power saving decreases from 30.6% (at 75% bandwidth) to 21.3% (at 150% bandwidth), while maintaining system performance.

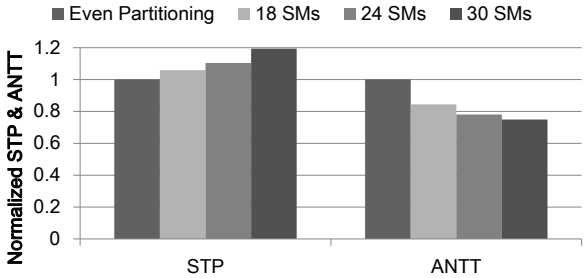
7.4.2 Number of SMs. We now vary the number of SMs. Figure 14a reports average performance under the performance mode. The opportunity for CD-search clearly increases with an increasing number of SMs. The reason is that the memory-intensive application is limited by the available memory bandwidth, hence there are more SMs available for the compute-intensive application to use, further improving overall system performance. On average, compared to even SM partitioning, CD-search improves STP by 19.4% on average for a GPU with 30 SMs.

Figure 14b shows the power and performance implications for the power mode. As expected, power consumption is significantly reduced while not negatively affecting performance. What is interesting here though — and unexpected — is that also system performance significantly improves at 30 SMs (by 19.5% on average), even in power mode. The reason is that as the number of SMs increases, cache contention also increases under even SM partitioning. As a result, reducing the number of active SMs and power-gating the remaining SMs reduces cache contention and leads to significant performance improvements for several memory-intensive workloads. Note the improvement in performance, which leads to higher dynamic power, offsets the power benefit of power-gating — however, we still observe an average 19.7% power saving.

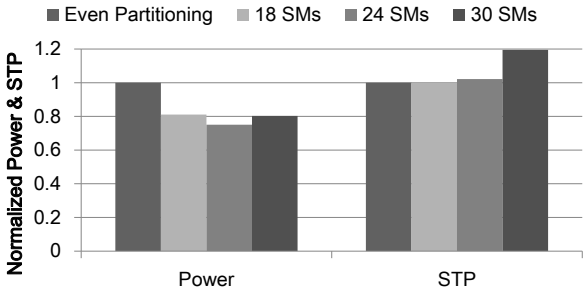
7.4.3 Four co-executing applications. We now consider four co-executing applications. The CD-search algorithm employed here for 4 co-running applications is a straightforward extension upon the one described in Section 4: we first find the least number of SMs to be assigned to the memory-intensive applications to maintain performance and then assign the remaining SMs through even partitioning to the compute-intensive application(s) in performance mode or power-gate them in power mode. Figure 15a summarizes average normalized STP and ANNT results for the heterogeneous workloads. Co-executing four aplications with different SM demands improves the overall resource utilization of even SM partitioning. As a result, the opportunity of CD-search decreases. Yet, CD-search still improves STP by 6.2% on average (and up to 42.3%), while at the same time reducing ANTT by 14.2% for the heterogeneous workload mixes in performance mode. In power mode, CD-search reduces power consumption by 16.4% on average.

7.5 Comparison against SMK

As mentioned in the introduction, simultaneous multikernel (SMK) execution [9, 10] is another approach to improve resource utilization in a fine-grained way within an SM. Although previous work showed that SMK works well for mixes of applications with different execution characteristics [9, 10, 13], Hongwen et al. [31, 32] more recently pointed out that even under a state-of-art intra-SM sharing scheme, performance still suffers due to interference among concurrent applications. Here we implement SMK following [10]

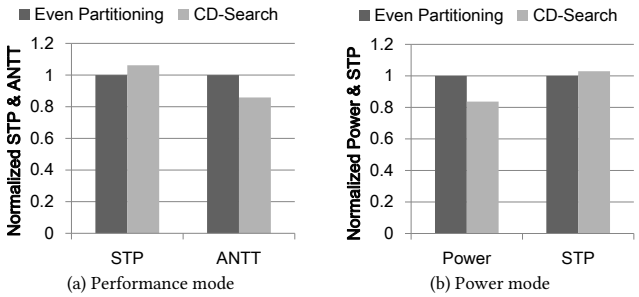


(a) Performance mode: Average STP and ANTT



(b) Power mode: Average power and STP

Figure 14: Sensitivity analysis: number of SMs. *The opportunity to improve performance and reduce power increases with increasing SM count.*



(a) Performance mode (b) Power mode

Figure 15: Sensitivity analysis: four concurrent applications. *CD-search improves performance and reduces power consumption even with four applications.*

which improves performance by dynamically partitioning SM resources. We also exploit a loose round-robin warp scheduler, which first selects kernels in a round-robin way and then selects warps within a kernel using the GTO policy. This guarantees fairness and achieves high STP [13].

SMK outperforms even SM partitioning for many workloads, however, SMK degrades performance severely for nearly one fourth of the workloads, see Figure 16. Ideally, SMK enables the compute-intensive application to execute instructions while the memory-intensive application is stalled. However, when co-executing two applications with different characteristics on an SM, if the memory-intensive application stalls the load/store unit due to serious cache or memory contention, the compute-intensive application cannot make forward progress even it is highly optimized for the shared memory or local cache. Park et al. [13] exploit this observation in Maestro by combining SMK with spatial multitasking (assuming

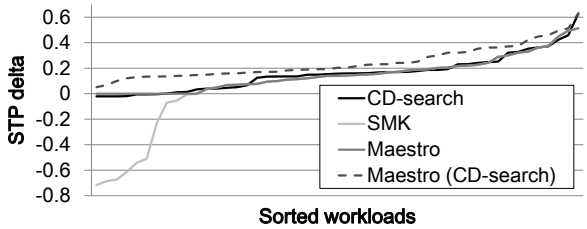


Figure 16: Comparing CD-search against SMK for heterogeneous workload mixes. *SMK leads to severe performance degradation for one fourth of the workloads due to intra-SM resource contention. Employing CD-search on top of Maestro achieves the highest performance.*

even SM partitioning) to achieve the best of both worlds. CD-search achieves similar performance as Maestro, see Figure 16, however it does not require SMK support. Moreover, Maestro assumes even SM partitioning. Adding CD-search (uneven SM partitioning) on top of Maestro further improves performance by 8.9% and outperforms all other designs as shown in Figure 16.

8 RELATED WORK

Spatial Multitasking. Commercial GPUs support spatial multitasking to increase GPU resource utilization. Adriaens et al. [6] argue that uneven SM partitioning does not provide a significant performance improvement over even partitioning on average. However, in this paper, we show that different optimization opportunities (performance versus power) can be exploited based on the characteristics of the multitasking workload. To guarantee fairness, Aguilera et al. [33] adjust SM allocation between co-executing applications to balance individual per-application performance. To improve throughput, Jog et al. [12] design a novel memory scheduler driven by an analytical performance model. To support preemption, Tanasic et al. [7] propose context switching and draining mechanisms and demonstrate their importance to improve system responsiveness and fairness. Chimera [21] combines different preemption approaches to reduce overall preemption overhead. Ausavarungnirun et al. [14] propose application-transparent multiple page size support to solve the address translation and demand page challenges in GPU spatial multitasking. Wang et al. [15] propose application-aware TLP management to improve fairness among co-executing applications. None of these prior works discuss nor propose a dynamic mechanism to determine an effective SM partition to optimize performance and/or power.

Simultaneous Multitasking (SMK). Several prior works explore the notion of multitasking within an SM. Wang et al. [9] propose Simultaneous Multikernel (SMK) execution, which exploits a TB dispatch mechanism and a warp scheduling algorithm to ensure fair resource allocation among applications within an SM. Xu et al. [10] propose Warp-Slicer, a run-time method to partition SM resources among different applications to maximize performance. To better share the resources in a single SM, Li et al. [34] combine TLP adjustment and cache bypassing. Dai et al. [32] balance memory accesses by limiting the number of in-flight memory requests issued from different applications. These SMK techniques are used to partition resources within an SM. Nevertheless, applications may still interfere with each other, especially in the L1 cache and load/store units.

Maestro overcomes this limitation by reverting to spatial multitasking for such cases [13]. However, Maestro is limited to even SM partitioning. As shown in this paper, CD-search outperforms SMK and is orthogonal to Maestro by further improving performance through uneven SM partitioning.

9 CONCLUSION

How to effectively multitask independent applications on a GPU begins to attract wide attention. In this paper, we show that uneven SM partitioning can bring substantial performance and power benefits. However, with an increasing number of SMs and co-running applications, a key challenge is how to determine an effective SM partition. This paper presents CD-search which dynamically classifies workloads based on a novel off-SM bandwidth model and then chooses the performance mode in case of heterogeneous workload mixes or the power mode in case of homogeneous memory-intensive workload mixes. Then in each mode, CD-search determines the optimum SM partition following a pre-determined search strategy, making dynamic SM partitioning tractable and low-overhead. Experimental results show that CD-search improves system throughput by 10.4% on average (and up to 62.9%) compared to even partitioning for heterogeneous workload mixes. For homogeneous memory-intensive workload mixes, CD-search reduces power consumption by 25% on average (and up to 41.2%).

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable feedback. We thank CalcUA for letting us use the NVIDIA P100 GPU. This work is supported by the European Research Council (ERC) Advanced Grant agreement No. 741097, FWO projects G.0434.16N and G.0144.17N, NSFC under Grant No. 61572508 and 61672526, NUDT Research Project No. ZK17-03-06. Xia Zhao is supported through a CSC scholarship and UGent-BOF co-funding.

REFERENCES

[1] Q. Chen, H. Yang, J. Mars, and L. Tang, "Baymax: QoS Awareness and Increased Utilization for Non-Preemptive Accelerators in Warehouse Scale Computers," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 681–696, April 2016.

[2] V. T. Ravi, M. Becchi, G. Agrawal, and S. Chakradhar, "Supporting GPU Sharing in Cloud Environments with a Transparent Runtime Consolidation Framework," in *Proceedings of the International Symposium on High Performance Distributed Computing (HPDC)*, pp. 217–228, June 2011.

[3] C. Margiolas and M. F. P. O'Boyle, "Portable and Transparent Software Managed Scheduling on Accelerators for Fair Resource Sharing," in *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, pp. 82–93, March 2016.

[4] Y. Suzuki, S. Kato, H. Yamada, and K. Kono, "GPUvm: Why Not Virtualizing GPUs at the Hypervisor?," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, pp. 109–120, June 2014.

[5] Amazon, "Amazon web services." https://aws.amazon.com/cn/ec2/.

[6] J. T. Adriaens, K. Compton, N. S. Kim, and M. J. Schulte, "The Case for GPGPU Spatial Multitasking," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 1–12, February 2012.

[7] I. Tanasic, I. Gelado, J. Cabezas, A. Ramirez, N. Navarro, and M. Valero, "Enabling Preemptive Multiprogramming on GPUs," in *Proceeding of the International Symposium on Computer Architecture (ISCA)*, pp. 193–204, June 2014.

[8] M. Awatramani, J. Zambreno, and D. Rover, "Increasing GPU Throughput using Kernel Interleaved Thread Block Scheduling," in *Proceedings of the International Conference on Computer Design (ICCD)*, pp. 503–506, October 2013.

[9] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo, "Simultaneous Multikernel GPU: Multi-tasking Throughput Processors via Fine-Grained Sharing," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pp. 358–369, March 2016.

[10] Q. Xu, H. Jeon, K. Kim, W. W. Ro, and M. Annavaram, "Warped-Slicer: Efficient Intra-SM Slicing through Dynamic Resource Partitioning for GPU Multiprogramming," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 230–242, June 2016.

[11] "NVIDIA Tesla V100 Volta Architecture."

[12] A. Jog, O. Kayiran, T. Kesten, A. Pattnaik, E. Bolotin, N. Chatterjee, S. W. Keckler, M. T. Kandemir, and C. R. Das, "Anatomy of GPU Memory System for Multi-Application Execution," in *Proceedings of the International Symposium on Memory Systems (MEMSYS)*, pp. 223–234, October 2015.

[13] J. J. K. Park, Y. Park, and S. Mahlke, "Dynamic Resource Management for Efficient Utilization of Multitasking GPUs," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 593–606, April 2017.

[14] R. Ausavarungnirun, J. Landgraf, V. Miller, S. Ghose, J. Gandhi, C. J. Rossbach, and O. Mutlu, "Mosaic: A GPU Memory Manager with Application-transparent Support for Multiple Page Sizes," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 136–150, October 2017.

[15] H. Wang, F. Luo, M. Ibrahim, O. Kayiran, and A. Jog, "Efficient and Fair Multiprogramming in GPUs via Effective Bandwidth Management," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, March 2018.

[16] A. Jadidi, M. Arjomand, M. T. Kandemir, and C. R. Das, "Optimizing Energy Consumption in GPUS Through Feedback-driven CTA Scheduling," in *Proceedings of the High Performance Computing Symposium (HPC)*, pp. 12:1–12:12, April 2017.

[17] A. Jadidi, "Kernel-Based Energy Optimization In GPUs," Master's thesis, The Pennsylvania State University, December 2015.

[18] N. Vijaykumar, K. Hsieh, G. Pekhimenko, S. Khan, A. Shrestha, S. Ghose, A. Jog, P. B. Gibbons, and O. Mutlu, "Zorua: A Holistic Approach to Resource Virtualization in GPUs," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 1–14, October 2016.

[19] H. Jeon, G. S. Ravi, N. S. Kim, and M. Annavaram, "GPU Register File Virtualization," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 420–432, December 2015.

[20] J. Lee and H. Kim, "TAP: A TLP-Aware Cache Management Policy For a CPU-GPU Heterogeneous Architecture," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–12, February 2012.

[21] J. J. K. Park, Y. Park, and S. Mahlke, "Chimera: Collaborative Preemption for Multitasking on a Shared GPU," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 593–606, March 2015.

[22] Nvidia, "NVIDIA TESLA P100 GPU ACCELERATOR." https://images.nvidia.com/content/tesla/pdf/nvidia-tesla-p100-PCIe-datasheet.pdf, 2016.

[23] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *Proceeding of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 163–174, April 2009.

[24] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling Energy Optimizations in GPGPUs," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 487–498, June 2013.

[25] "NVIDIA CUDA SDK Code Samples." https://developer.nvidia.com/cuda-downloads.

[26] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing," tech. rep., March 2012.

[27] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, pp. 44–54, October 2009.

[28] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a High-Level Language Targeted to GPU Codes," in *Proceedings of Innovative Parallel Computing (InPar)*, pp. 1–10, May 2012.

[29] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: A MapReduce Framework on Graphics Processors," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 260–269, October 2008.

[30] S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads," *IEEE Micro*, vol. 28, no. 3, pp. 42–53, 2008.

[31] H. Dai, Z. Lin, C. Li, C. Zhao, F. Wang, N. Zheng, and H. Zhou, "POSTER:Accelerate GPU Concurrent Kernel Execution by Mitigating Memory Pipeline Stalls," in *Proceedings of the International Conference on Parallel Architectures and Compilation (PACT)*, pp. 144–145, September 2017.

[32] H. Dai, Z. Lin, C. Li, C. Zhao, F. Wang, N. Zheng, and H. Zhou, "Accelerate GPU Concurrent Kernel Execution by Mitigating Memory Pipeline Stalls," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, March 2018.

[33] P. Aguilera, K. Morrow, and N. S. Kim, "Fair Share: Allocation of GPU Resources for Both Performance and Fairness ," in *Proceedings of the International Conference on Computer Design (ICCD)*, pp. 440–447, October 2014.

[34] X. Li and Y. Liang, "Efficient Kernel Management on GPUs," in *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 115:1–115:24, March 2016.