Process Variation-Aware Workload Partitioning Algorithms for GPUs Supporting Spatial-Multitasking

Paula Aguilera[1], Jungseob Lee[1], Amin Farmahini-Farahani[1],
Katherine Morrow[1], Michael Schulte[2], Nam Sung Kim[1]
[1]University of Wisconsin – Madison, [2]Advanced Micro Devices
{paguilera, jslee9, farmahinifar, klmorrow, nskim3}@wisc.edu, michael.schulte@amd.com

Abstract— **High-level programming languages have transformed graphics processing units (GPUs) from domain-restricted devices into powerful compute platforms. Yet many "general-purpose GPU" (GPGPU) applications fail to fully utilize the GPU resources. Executing multiple applications simultaneously on different regions of the GPU (spatial multitasking) thus improves system performance. However, within-die process variations lead to significantly different maximum operating frequencies (F_{max}) of the streaming multiprocessors (SMs) within a GPU. As the chip size and number of SMs per chip increase, the frequency variation is also expected to increase, exacerbating the problem. The increased number of SMs also provides a unique opportunity: we can allocate resources to concurrently-executing applications based on how those applications are affected by the different available F_{max} values. In this paper, we study the effects of per-SM clocking on spatial multitasking-capable GPUs. We demonstrate two factors that affect the performance of simultaneously-running applications: (i) the SM partitioning algorithm that decides how many resources to assign to each application, and (ii) the assignment of SMs to applications based on the operating frequencies of those SMs and the applications characteristics. Our experimental results show that spatial multitasking that partitions SMs based on application characteristics, when combined with per-SM clocking, can greatly improve application performance by up to 46% on average compared to cooperative multitasking with global clocking.**

I. INTRODUCTION

Initially developed for compute-intensive graphic applications, GPUs have evolved considerably over the last few years. Increased programmability in the graphic pipeline and higher precision computations also allow GPUs to accelerate general purpose applications. With the development of high-level programming languages such as CUDA [1] and OpenCL [2], general-purpose computing on GPUs [3] (GPGPU) has become widely used for a variety of applications from scientific computations to multimedia processing [4].

It has been shown that GPGPU spatial multitasking, which allows multiple applications to simultaneously execute (each on a subset of the GPU resources), provides increased resource utilization, performance, and quality of service over cooperatively multitasking the GPU resources [5]. NVIDIA's *Fermi* [6] and *Kepler* [7] architectures support basic spatial multitasking, and future generations of GPUs are expected to implement more flexible forms of spatial multitasking [8, 9] since spatial multitasking will be even more useful with more cores to increase throughput.

As process technology scales below 65-nm, considerable transistor delay variations occur both within-die (WID) and die-to-die (D2D) [10]. Specifically, WID process variations result in a large difference among the maximum operating frequencies (F_{max}) of each core on the same die. Using a single global clock, as is the case in most CPUs and GPUs, requires all cores to run at the same frequency—the minimum of the cores' F_{max} values. Because GPUs have a much larger number of cores than CPUs, this issue is particularly problematic. Clocking each core (called a streaming multiprocessor, or SM) at its own F_{max}, removes the artificial restriction of global clocking (GC), and thus improves throughput. This technique is called per-SM clocking (PSMC) [3]. When multiple GPGPU applications with different characteristics run on a spatially-multitasking GPU with PSMC, the assignment of SMs to applications can significantly impact performance. Some applications benefit from higher SM frequencies, while others do not suffer when executing on SMs with lower frequencies.

In this paper, we exploit (i) WID process variations and (ii) characteristics of GPGPU applications to maximize the overall performance of spatial-multitasking GPUs. More specifically, we make the following key contributions:

- We characterize GPGPU the sensitivity of applications' performance to SM's operating frequency in the context of spatial-multitasking GPUs.
- We propose WID process variation-aware SM-to-application assignment techniques for spatial-multitasking GPUs supporting PSMC.
- We demonstrate that assigning faster SMs to compute-bound and slower SMs to memory-bound applications can considerably improve overall performance of spatial-multitasking GPUs supporting PSMC.
- We evaluate the efficacy of the proposed application assignment technique for various SM partitioning algorithms for spatial-multitasking GPUs.

This paper is organized as follows: Section II gives some background, Section III introduces the related work, Section IV describes our methodology, Section V characterizes our applications, Section VI shows our results and Section VII concludes the paper.

II. BACKGROUND

A. GPU Multitasking

Because GPUs are highly-parallel devices with an ever growing number of cores, they can use *spatial* multitasking (dividing resources amongst co-executing applications) in addition to *temporal* multitasking (dividing resource time amongst applications so they appear to execute simultaneously) [5]. Current GPUs only have primitive multitasking support.

For example, The NVIDIA Fermi architecture only supports spatially-multitasking kernels from the same application [6]. NVIDIA's newer Kepler architecture supports co-executing multiple GPGPU applications [6], as long as there are resources available on the GPU, but the implementation details have not been disclosed and it is not clear how the resources are distributed among the different kernels and what the performance implications are. Generally, however, GPU applications must wait to execute until the application currently occupying the GPU voluntarily yields control—a form of cooperative multitasking.

In contrast, an operating system (OS) typically preemptively multitasks a CPU, suspending and later resuming applications to time-share the CPU without the applications' intervention or control. Both cooperative and preemptive multitasking are forms of *temporal* multitasking. Multi-core CPUs also use *spatial* multitasking—multiple applications execute simultaneously on different cores. As the GPU moves onto the same chip as the CPU [11], the potential advantage of using GPUs as parallel accelerators with multitasking capabilities will grow. Fully exploiting GPUs will require new temporal and spatial GPU multitasking techniques.

B. *SM Partitioning Algorithms*

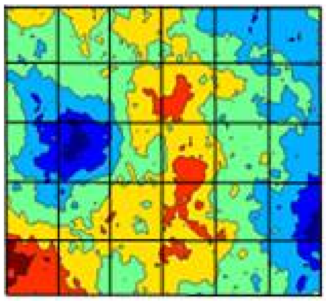
We use the following partitioning algorithms [5] to assign SMs to GPGPU applications and evaluate PSMC: *Even-Split* assigns SMs evenly among running applications, regardless of their computational needs. *Profile* uses isolated kernel profiling information to choose the best SM partitioning for each application. In our experiments, we profile applications in advance, and the results of profiling are used by the allocator. However, the profiling could be performed at runtime [12] if data-dependent application behavior is a concern.

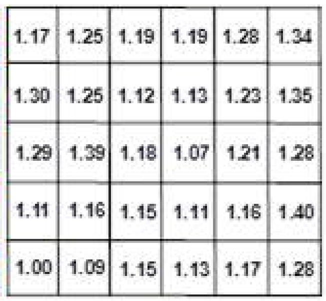
We use *Even-Split* and *Profile* to examine SM partitioning methods that have different objectives: *Even-Split* aims to provide applications with equal access to GPU resources. *Profile*, on the other hand, tries to assign the number of resources to each application that maximizes system throughput. The *best* partitioning method would depend on the system's/user's goals.

C. *SM-to-SM Frequency Variation and GPUs Supporting PSMC*

As technology scaling increases WID, but also encourages integrating more SMs per die, the SM-to-SM frequency (and leakage power) variations also increase, and significantly so. According to a model applied to a GPU comprised of 30 SMs in 32nm technology, the fastest SM can be up to 40% faster than the slowest SM [13]. In a globally-clocked GPU this leads to the "worst of both worlds": all SMs are limited to the F_{max} of the slowest SM, but cores otherwise capable of higher frequencies consume much more leakage power, degrading power efficiency considerably [10]. In contrast, clocking each SM individually (Per-SM Clocking, or PSMC) allows us to exploit SM-to-SM F_{max} variation to improve overall performance [3].

Fig. 1(a) shows a WID threshold voltage (V_{th}) variation map for a region of a die in a GPU with 30 SMs. To generate spatially-correlated V_{th} and effective channel length (L_{eff}) maps for a GPU die, we used an analytical WID variation model and the parameters presented in [13]. In Fig. 1(b), each rectangle represents an SM and each number corresponds to the F_{max} of


(a)



1.17	1.25	1.19	1.19	1.28	1.34
1.30	1.25	1.12	1.13	1.23	1.35
1.29	1.39	1.18	1.07	1.21	1.28
1.11	1.16	1.15	1.11	1.16	1.40
1.00	1.09	1.15	1.13	1.17	1.28

(b)

Fig. 1: (a) A WID Vth variation map for a 30-SM GPU and (b) the corresponding normalized F_{max}

that SM normalized to the F_{max} of the slowest SM. The F_{max} of the fastest SM is 1.4 times the F_{max} of the slowest SM in the GPU die shown in Fig. 1. As more SMs are integrated in a die (e.g., the die size becomes larger), the relative SM-to-SM F_{max} variations across the die increase.

We can exploit this WID F_{max} variation to improve performance by implementing PSMC on a GPU. Per-core PLLs are already commonly used even for globally-clocked multiprocessors to optimize clock skew and reduce power dissipation of clock trees [14]. Thus, we assume there would be little to no extra cost to support PSMC using per-SM PLLs on a GPU. If the number of SMs is such that it makes the cost of having a PLL per SM too high, we can provide a PLL per cluster of neighboring SMs to lower the overhead. This approach would still provide benefit, since WID process variations are typically dominated by spatially correlated ones.

III. Related Work

K. Bowman *et al.* [15] and Humenay *et al.* [16] examined the impact of WID C2C frequency variations on the performance of multi-core processors. The performance improvement of a *frequency islands* (FIs) scheme exploiting WID C2C frequency variation across a range of multicore processor designs is examined using an analytical throughput model of the FIs multi-core processors [13]. Lee *et al.* [3] showed that PSMC on a GPU having WID process variation can significantly improve performance because there is rare or no inter-SM communications in most GPGPU kernels. The independence of the thread blocks within the same GPGPU kernel allows any available SM in a GPU to execute multiple blocks independently. To the best of our knowledge this paper is the first time PSMC and application sensitivity to frequency scaling are used to improve GPGPU resource partitioning for spatial multitasking.

IV. Methodology

In this work, we use an assortment of applications with different characteristics to study potential opportunities for PSMC to improve overall performance. These applications are a subset of the ones used by Adriaens *et al.* [5]: AES-E, DVC, ID, JPG-D, RAY, SAD and SHA1. We also use two new applications: Neural Networks (NN) [17] and Back Propagation (BPR) [18]. We simulate CUDA applications with GPGPU-Sim v2.1.1, a detailed cycle-level execution-driven GPU simulator capable of simulating CUDA and OpenCL applications [19]. We use the default GPGPU-Sim parameters that approximate the NVIDIA Quadro FX 5800 GPU [20], as

TABLE I. KEY GPU HARDWARE CONFIGURATION PARAMETERS

# SMs	30
SM Frequency	1300 MHz
Warp Size	32 Threads
SIMD Width	8
# Threads per SM	1024
Registers / Shared Memory per SM	64 KB / 16 KB
# Memory Controllers	8
Memory Frequency / Bandwidth	800 MHz / 102 GB/s
Bandwidth per Memory Module	8 Bytes per Cycle
Interconnect Topology / Frequency	Crossbar / 650 MHz

shown in TABLE I. We further modified GPGPU-Sim Spatial Multitasking [5] to include WID process variation by supporting separate clock frequency domains for each of the main GPU components: the SMs, the interconnect network, and the off-chip memory (DRAM). To model the clock domain crossing, we implement send/receive buffers at the clock boundaries. The buffers are filled and drained at different clock rates (e.g., the buffer from the interconnect to the memory controller is filled at the interconnect's clock rate, but drained at the DRAM's clock rate). This modification allows each SM to execute threads at its own F_{max}.

We use the methodology presented in [5] to compare performance of multitasking GPGPU applications. We run all spatial multitasking simulations for 5M GPU cycles and measure the number of instructions simulated for each application. If an application completes in less than 5M cycles, it is restarted so that we are able to gather data for a full 5M cycles. We measure the number of instructions that each application completes, which represents a measure of work accomplished by the application during the 5M cycle timeframe. We then run the applications for the same number of instructions in cooperative multitasking, and determine the change in time required for those instructions. This ensures that we are comparing the same amount of steady-state work for each application between spatial and cooperative multitasking. The source code for our modified version of GPGPU-Sim Spatial Multitasking that models WID and supports PSMC is available at: http://mesa.ece.wisc.edu/gpgpu.

V. APPLICATION CHARACTERIZATION

WID process variation provides an opportunity to assign SMs with different clock frequencies to concurrently-running applications. While some applications show considerable speedup at higher frequencies, others do not benefit as much. To determine how sensitive applications are to frequency, we run each application in isolation using the simulator. For each run we increase the frequency of all the SMs on the GPU. We study four different simulated architectures related to the NVIDIA Quadro FX 5800 GPU. That GPU has 30 SMs and 8 memory controllers (MCs), so to estimate spatial multitasking performance from performance in isolation, we test performance for 15 and 10 SMs and 2, 3, and 8 memory controllers to model a reduced resource allocation as would occur in spatial multitasking. In particular, we examine one architecture with 15 SMs and 4 memory controllers, one with 10 SMs and 3 memory controllers, and finally both 15 and 10 SMs with 8 memory controllers. The latter two architectures help identify if the performance of each application is more affected by raw SM frequency, memory bandwidth, or a combination of the two. We use five different SM frequencies, ranging from 1.3GHz to 1.82GHz with 130MHz steps. Note that all SMs are clocked at the same frequency in each run (i.e., GC) and the highest tested frequency is 1.4 times the lowest frequency. This is a realistic frequency range considering WID process variations in a large die for 32nm technology, as discussed in Section II.

Fig. 2 shows, for each application in isolation, speedups for different clock frequencies normalized to the baseline 1.3GHz clock. Based on these results, we classify applications in two groups: *compute-bound* and *memory-bound*. This classification is performed depending on how an application performance scales with increasing SM frequency.

AES-E, ID, RAY and SAD show almost linear speedup in all cases, demonstrating compute-bound behavior. For example, AES-E has a speedup of 1.39 times for a clock frequency increase of 1.4 times in all cases. On the other hand, NN has little speedup at high frequencies.

Applications such as DVC and JPG-D show moderate performance improvement with frequency increase, though by a greater degree when more memory bandwidth is also available. For instance, when there are 10 SMs and three memory controllers, the performance of DVC increases by 23% when the frequency increases by 40%. By increasing the number of memory controllers to eight, the performance of

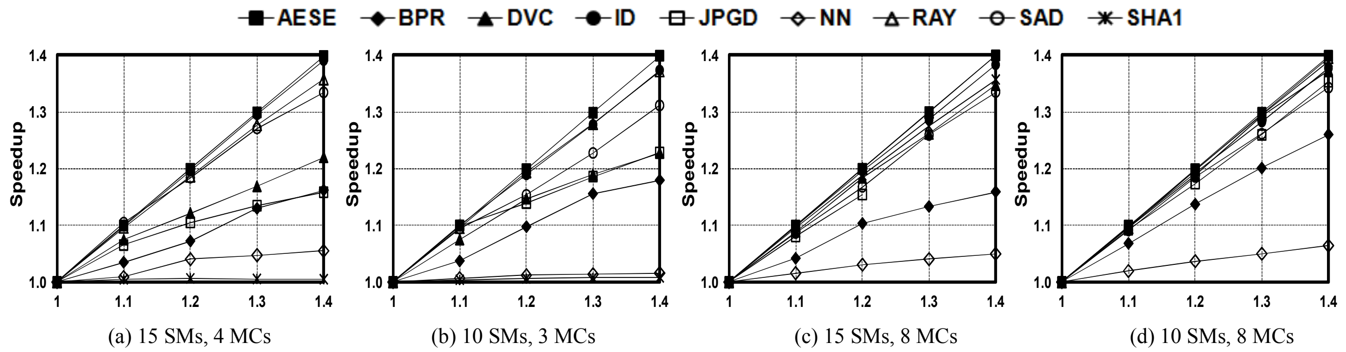


Fig. 2: Speedup vs. SM frequency for 15 and 10 SMs and 8, 4 and 3, memory controllers. Results are normalized to a GPU with SMs operating at the base frequency (1.3GHz). The X-axes is the normalized SM frequency.

DVC increases by 37% (close to the linear performance improvement of 40%). SHA1's performance trend is also dependent on the number of memory controllers; at 1.82GHz, when SHA1 is allocated 10 SMs, it shows at most 1% performance improvement when there are three memory controllers available, yet up to 39% performance improvement when there are eight memory controllers On the other hand, although BPR also shows moderate performance improvement with increasing frequency, its speedup is almost independent of the number of available memory controllers.

We classify RAY, AES-E, ID, and SAD as compute-bound because they show nearly linear speedup with frequency. On the other hand, we classify SHA1, JPG-D, NN, BPR, and DVC as memory-bound since they do not benefit significantly from increased SM frequency. This classification can be done either in advance or at runtime by using a sampling method [12].

VI. Variation-Aware Application Assignments for Spatial-Multitasking GPUs Supporting PSMC

This section presents experiments that combine PSMC with our SM partitioning algorithms to improve overall system performance in a spatial-multitasking GPU.

A. *Performance Impact of Application Assignment Methods for Spatial-Multitasking GPUs Supporting PSMC*

In a spatial-multitasking GPU supporting PSMC, different applications respond differently to being assigned *fast* vs. *slow* SMs. We hypothesize that assigning more compute-bound applications to fast SMs and less compute-bound applications to slow SMs will result in higher overall performance. We compare this approach to one where we perform the opposite assignment (compute-bound applications to slow SMs and memory-bound applications to fast SMs). In both methods, we use the *Profile* SM partitioning algorithm (Section II) and we run several combinations of two applications for a time corresponding to 5M GPU cycles of the slowest SM. We use *Profile* because it provides better overall system performance.

TABLE II compares the performance of the two application assignment methods: (i) compute-bound applications to faster SMs and memory-bound applications to slower SMs (CFMS) and (ii) compute-bound applications to slower SMs and memory-bound applications to faster SMs (CSMF). The results, normalized to cooperative multitasking with GC, verify that CFMS is better and never worse than CSMF. The difference (Δ in TABLE II) is greatest when a compute-bound kernel executes with a memory-bound kernel, and negligible for applications where kernels have similar characteristics. The average difference between these two methods is 5%. For the remainder of the paper, we will use the CFMS method to gather all PSMC data.

B. *Impact of SM Partitioning Algorithms on Performance of Spatial-Multitasking GPUs Using CFMS*

First, we run combinations of two applications on the GPU to study the performance improvement of using PSMC with CFSM assignment of SMs with our partitioning algorithms over GC. In Fig. 3 each graph shows the speed up of a compute-bound application running with the memory-bound ones. Assigning faster SMs to compute-bound application(s) and slower SMs to memory-bound application(s) leverages the PSMC benefits provided by WID. In Fig. 3 we observe that

TABLE II. Speedup Of two-Application Workload Using CSMF and CFMS Assignments (*Profile* Partitioning, Normalized to Global Clocking Cooperative)

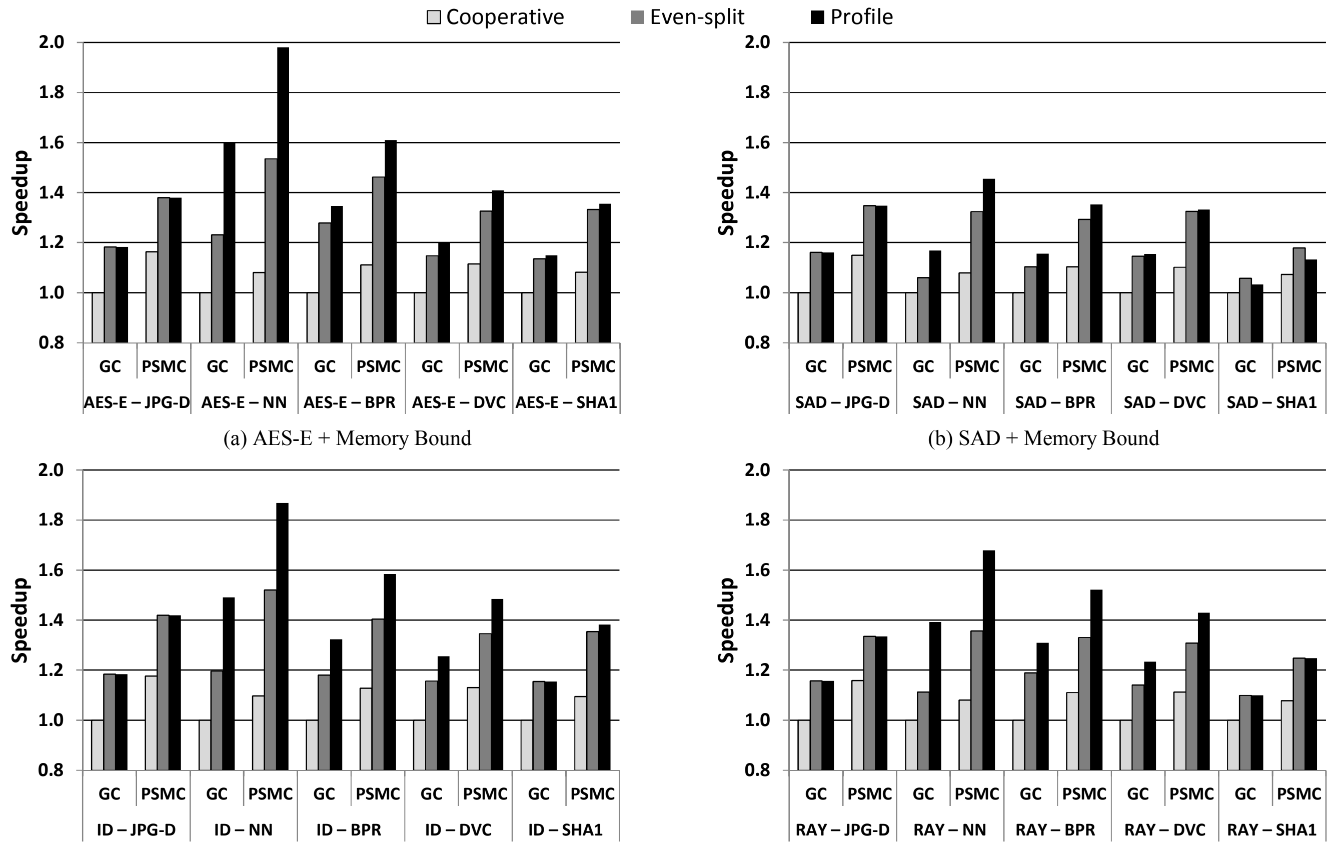
Workload	CSMF	CFMS	Δ	Workload	CSMF	CFMS	Δ
AES-E / JPG-D	1.34	1.38	**0.04**	**ID / DVC**	1.44	1.48	**0.04**
AES-E / NN	1.82	1.98	**0.16**	**ID / SHA1**	1.30	1.38	**0.08**
AES-E / BPR	1.57	1.60	**0.03**	**RAY / JPG-D**	1.29	1.33	**0.04**
AES-E / DVC	1.39	1.40	**0.01**	**RAY / NN**	1.56	1.67	**0.12**
AES-E / SHA1	1.28	1.35	**0.07**	**RAY / BPR**	1.48	1.52	**0.04**
SAD / JPG-D	1.32	1.34	**0.02**	**RAY / DVC**	1.40	1.42	**0.02**
SAD / NN	1.32	1.45	**0.13**	**RAY / SHA1**	1.19	1.24	**0.05**
SAD / BPR	1.30	1.35	**0.05**	**BPR / NN**	1.12	1.13	**0.01**
SAD / DVC	1.31	1.33	**0.02**	**AES-E /ID**	1.24	1.24	**0.00**
SAD / SHA1	1.08	1.13	**0.05**	**JPG-D / DVC**	1.36	1.36	**0.00**
ID / JPG-D	1.36	1.42	**0.06**	**RAY / SAD**	1.27	1.27	**0.00**
ID / NN	1.70	1.86	**0.16**	**DVC / SHA1**	1.17	1.18	**0.01**
ID / BPR	1.51	1.58	**0.07**	**Geo Mean**	**1.36**	**1.41**	**0.05**

performance improvements of PSMC using CFSM to be higher when the compute-bound application is executed with NN or BPR. As shown in Fig. 2, NN and BPR do not benefit from high frequency SMs and thus are not penalized when assigned slower SMs. Furthermore, Fig. 2 showed that AES-E and ID scale better with frequency than SAD or RAY. Hence, combinations of either AES-E or ID with memory-bound applications give higher speedups than when those memory-bound applications are combined with SAD or RAY. *Even-Split* partitioning outperforms cooperative multitasking, and *Profile* generally outperform *Even-Split*.

Fig. 4 shows PSMC performance when two compute-bound applications or two memory-bound applications run simultaneously on the GPU. For brevity, we only present results for a few combinations likely to occur in real-world scenarios, but other combinations would have similar results. As expected, PSMC only performs slightly better than GC. When two memory-bound applications co-execute, neither benefits much from higher frequency SMs. When two compute-bound applications co-execute, the benefit achieved by one application using fast SMs is largely offset by the penalty of the other using slow SMs.

Fig. 5 shows the geometric means of the speedups of the application combinations shown previously (in Fig. 3 and Fig. 4). The first four pairs of columns are the geometric means of the compute-bound applications AES-E, SAD, ID, and RAY running simultaneously with all the memory-bound applications. The fifth column is the geometric means of the speedup of the combinations shown in Fig. 4 (including tested combinations not individually shown in the figure), and the sixth column is the geometric mean of all the combinations.

As before, the most compute-intensive applications gain the most from PSMC when executed with memory-bound applications, and the overall speedup using PSMC for the different partitioning algorithms ranges from 18%-20% over their counterparts using GC. Compared to cooperative multitasking with GC, these speedups increase to 33%-41% across all tested combinations, and 36%-46% for cases that combine compute- with memory-bound applications.



(a) AES-E + Memory Bound

(b) SAD + Memory Bound

(c) ID + Memory Bound

(d) RAY + Memory Bound

Fig. 3: Speedup of compute- and memory-bound application combinations for spatial multitasking partitioning algorithms using both GC and PSMC. The results are normalized to cooperative multitasking with GC.

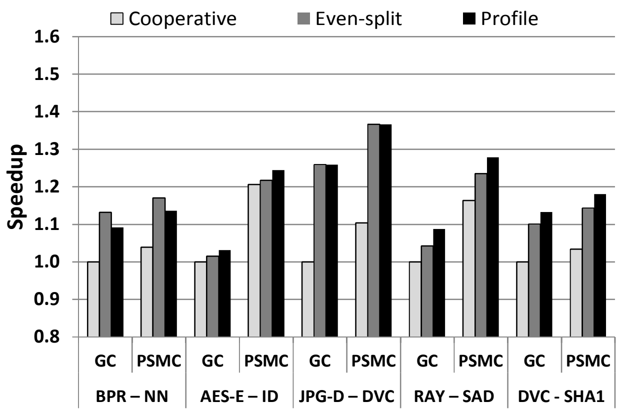


Fig. 4: Speedup of combinations of two compute- and two memory-bound apps for different partitioning algorithms with GC and with Per-SMC. The results are normalized to cooperative multitasking with GC.

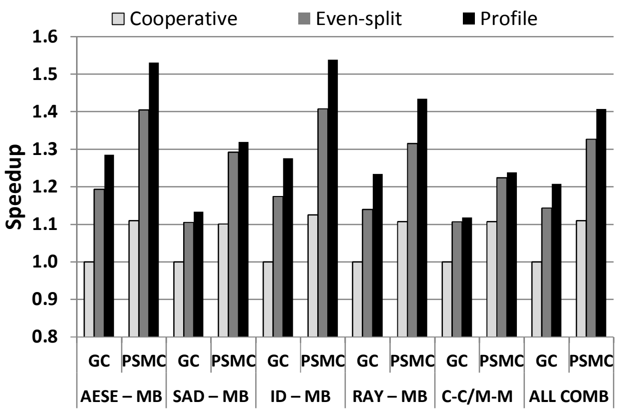


Fig. 5: Geometric mean speedup of combinations of compute- and memory-bound applications, and the geometric mean of speedups for all combinations. The results are normalized to cooperative multitasking with GC.

C. *Combinations of three applications*

We extended the experiments from the previous section to fourteen combinations of three applications that might run together by an user on our target platform. For example, ID, JPG-D, and DVC together represent a user decoding and editing video and images. Combining AES-E, DVC and NN represents encrypting and decoding a high-definition video while searching for objects of interest in that video.

Fig. 6 shows the speedups achieved using the different partitioning algorithms over cooperative multitasking with global clocking. As expected, speedups are best when combining a mix of compute- and memory-bound applications. Also, similar to combinations of two applications, *Profile* results in better speedups than *Even-Split* for most combinations.

For some combinations that contain DVC (e.g., AES-E/DVC/BPR, DVC/DVC/NN, and DVC/BPR/SHA1), Fig. 6 shows that *Profile* partitioning algorithm actually performs worse than *Even-Split*, which was initially an unexpected result. However, *Profile* takes into account performance information from running the applications in isolation on the GPU;

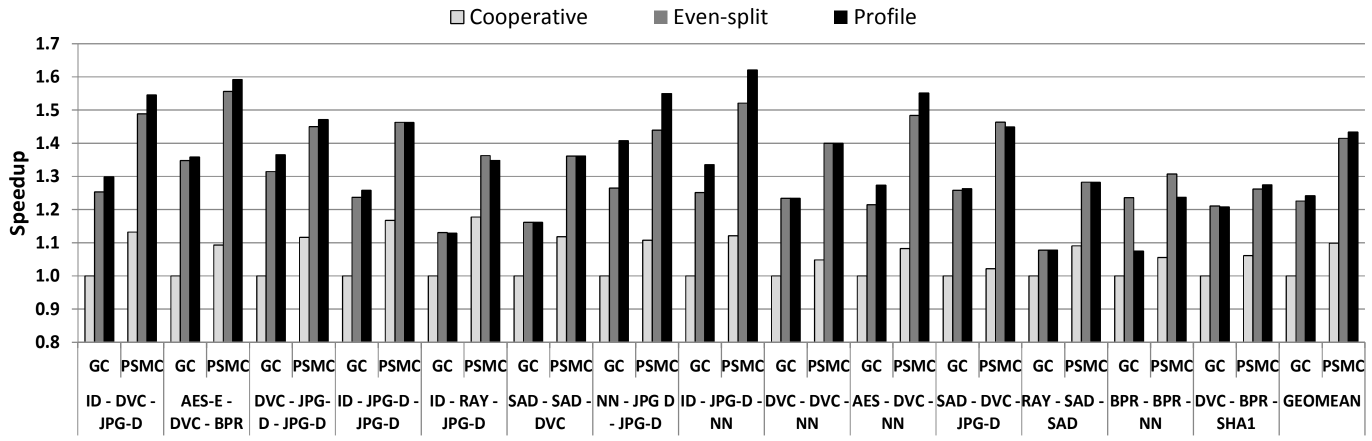


Fig. 6: Speedups of combinations of three applications using different multitasking partitioning algorithms with GC and PSMC. The results are normalized to cooperative multitasking with GC.

sometimes when multiple applications run simultaneously on the GPU, the contention between different applications in the shared resources (such as interconnect and memory) can significantly affect performance.

When we combine resource partitioning algorithms with PSMC, we achieve average increased speedups of 19% for both *Even-Split* and *Profile*, respectively, compared to the performance achieved by the same partitioning algorithms with GC. Compared to cooperative multitasking with GC, these speedups increase up to 42% and 43% when executing a combination of three applications simultaneously on the GPU using spatial multitasking with PSMC.

VII. CONCLUSION

We propose a variation-aware assignment of SMs to applications in a GPU that supports spatial multitasking and PSMC. We characterize GPGPU applications as compute-bound or memory-bound based on their change in performance as the frequency of SMs is increased. Next, we apply a variation-aware assignment of SMs to applications that co-execute with spatial multitasking. Our proposed technique that assigns faster SMs to compute-bound applications and slower SMs to memory-bound applications demonstrates an improvement in performance. We also evaluate different SM partitioning algorithms that take advantage of PSMC in a multitasking GPU. When executing two or three applications concurrently, we achieve speedups of 18%-20% using *Even-Split* and *Profile* partitioning algorithms, respectively, compared to the same partitioning algorithms with global clocking. These speedups increase to 36%-46% when compared to the performance of cooperative multitasking with global clocking.

ACKNOWLEDGEMENT

This work is supported in part by an SRC grant (Task ID: 2080.001), NSF grants (CNS-0952425, CCF-095360, and CNS-1217102), and generous gift grants from AMD.

REFERENCES

[1] "NVIDIA CUDA Programming Guide," 2013. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[2] Khronos Group, 2011. [Online]. Available: http://www.khronos.org/opencl/.

[3] J. Lee, P. Ajgaonkar and N. S. Kim, "Analyzing throughput of GPGPUs exploiting within-die core-to-core frequency variation," in *ISPASS*, 2011.

[4] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer and K. Skadron, "A performance study of general-purpose applications on graphics processors using CUDA," *Parallel and Distributed Computing,* 2008.

[5] J. T. Adriaens, K. Compton, N. S. Kim and M. J. Schulte, "The case for GPGPU spatial multitasking," in *HPCA*, 2012.

[6] "NVIDIA's Next Generation Computer Architecture: Fermi".

[7] NVIDIA, "NVIDIA's Next Generation CUDA Compute Architecture: KeplerGK110".

[8] S. Vaughn-Nichols, "Vendors draw up a new graphics-hardware approach," *Computer,* 2009.

[9] P. Rogers, "The Programmer's Guide to the APU Galaxy," 2011.

[10] S. Borkar, T. Karnik and V. De, "Design and reliability challenges in nanometer technologies," in *DAC*, 2004.

[11] S. Vaughn-Nichols, "Vendors draw up a new graphics-hardware approach," *Computer,* vol. 42, no. 5, p. 11–13, 2009.

[12] J. Lee, V. Sathisha, M. Schulte, K. Compton and N. S. Kim, "Improving Throughput of Power-Constrained GPUs Using Dynamic Voltage/Frequency and Core Scaling," in *PACT*, 2011.

[13] S. Herbert and D. Marculescu, "Characterizing Chip-Multiprocessor Variability-Tolerance," in *DAC*, 2008.

[14] N. Kurd, J. Douglas, P. Mosalikanti and R. Kumar, "Next generation Intel® micro-architecture (Nehalem) clocking architecture," in *IEEE Symp. on VLSI Circuits*, June 2008.

[15] K. Bowman, A. Alameldeen, S. Srinivasan and C. Wilkerson, "Impact of die-to-die and within-die parameter variations on the throughput distribution of multi-core processors," in *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2007.

[16] E. Humenay, D. Tarjan and K. Skadron, "Impact of Process Variations on Multicore Performance Symmetry," in *DATE*, 2007.

[17] B. Kavinguy et al, "A Neural Network on GPU," [Online]. Available: http://www.codeproject.com/KB/graphics/GPUNN.aspx.

[18] "The Rodinia Benchmark Suite, version 2.0," [Online]. Available: https://www.cs.virginia.edu/~skadron/wiki/rodinia.

[19] A. Bakhoda, G. Yuan, W. Fung, H. Wong and T. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *ISPASS*, 2009.

[20] NVIDIA, "NVIDIA Quadro FX 5800," [Online]. Available: http://www.nvidia.com/object/product_quadro_fx_5800_us.html.