Research Statement

Adwait Jog, Ph.D. Candidate, Penn State

The ideology of "think parallel" has ushered in an abundance of parallelism in real-world applications emerging from a wide range of areas such as medicine, science, finance, engineering, social media, and gaming. In order to provide high-performance, energy-efficient, and cost-effective architectural support for these applications, computer architecture is undergoing a paradigm shift towards employing *many-core* data-parallel accelerators along with *multiple* processors in the same heterogeneous computing platform. These platforms are making headway in almost every kind of computing system ranging from warehouse-scale to wearable computers and my research interests lie in improving their performance and energy efficiency. I am also interested in rethinking the hardware/software interface for heterogeneous computing systems and leveraging emerging memory technologies to design memory and storage systems for them.

1 Research Thrusts and Approach

Graphics Processing Units (GPUs) are becoming an inevitable part of every computing system because of their ability to accelerate applications consisting of abundant data-level parallelism. They are not only used to accelerate big data analytics in cloud data centers or high-performance computing (HPC) systems, but are also employed in mobile and wearable devices for efficient execution of multimedia rich applications and smooth rendering of display. My research shows that, in spite of the highly parallel structure of GPUs and their ability to execute multiple threads concurrently, they are far from achieving their theoretically achievable peak performance. This is attributed to several reasons such as contention for limited shared resources (e.g., caches and memory), high control-flow divergence, and limited off-chip memory bandwidth. Another reason for the low utilization and subpar performance is that the current GPUs are simply not well-equipped to efficiently and fairly execute multiple applications concurrently, potentially originating from different users. These problems pose several forward looking research questions: a) How should data-parallel GPU architectures be designed in order to improve their efficiency in terms of performance, energy, and cost?; b) How can we maximize fairness and throughput when concurrent threads belonging to a single application or concurrent multiple applications contend for the same shared resources?; c) How can we effectively leverage emerging memory technologies for architecting the memory hierarchy?; and d) How should we design the hardware/software interface for heterogeneous systems consisting of different form-factor GPUs, CPUs, and IPs, in addition to addressing the aforementioned questions?
Research Approach. My research is tackling these problems by taking a holistic approach that is: 1) *application-driven*, which considers application-level characteristics and requirements; 2) *understanding-driven*, which provides mathematical, qualitative, and quantitative understanding of the performance, energy-efficiency, locality, and parallelism trade-offs; 3) *idea-driven*, which proposes novel ideas to design better computing systems; 4) *hardware/software interface-driven*, which inspires appropriate abstractions for enabling several optimizations; and 5) *technology-driven*, which leverages emerging memory technologies to design memory and storage systems.

2 Dissertation Research Contributions

My dissertation research is focused on managing contention in GPUs for shared cache and memory resources caused by concurrently executing threads. This contention causes severe loss in performance, fairness, locality, and parallelism. To manage this contention, I proposed techniques that are employed at two different places: *core* and *memory*. First, I showed that by intelligently scheduling the threads at the *core*, the generated memory request patterns can be more amenable for existing resource management techniques such as cache replacement and memory scheduling as well as performance enhancement techniques such as data prefetching. Second, I showed that considering criticality and other application characteristics to schedule memory requests at the memory controller is an effective way to manage contention at the *memory*.

2.1 Managing Contention from Cores via Warp Scheduling

GPU cores schedule threads at the granularity of a *warp*, which is essentially a collection of individual threads that execute a single instruction on the functional units in lock step. My research showed that the existing warp

scheduling techniques employed at the core are oblivious to the underlying shared resource management policies, and therefore, the warp scheduling decisions taken at the core might not always be in harmony with the shared resource management scheduling policies. In this context, I proposed two warp scheduling techniques: 1) cache and memory-aware warp scheduling; and 2) data prefetching-aware warp scheduling. Both of these schedulers exploit an important property of GPUs that there is no ordering restriction among the execution of warps and the warp scheduler can efficiently choose the desired warps without incurring significant overhead.

Cache and Memory-Aware Warp Scheduling [1]. The key problem with the traditional round-robin warp scheduling policy is that it allows a large number of warps to concurrently access the cache. This makes it harder for the underlying cache management policies to leverage the locality present in many CUDA applications. In order to manage the contention in caches, I proposed a *cache-aware* warp scheduler that essentially reduces the number of warps that can benefit from caches in a given time interval. Although this scheduler improves the cache hit rates significantly, it turns out that it is not aware of the warp scheduling decisions taken at the *other* cores. This unawareness causes the schedulers at different cores to schedule warps such that they happen to concurrently access a limited set of global memory banks, consequently leading to inefficient utilization of the available memory bandwidth. To this end, I proposed a *memory-aware* warp scheduler called OWL by extending the cache-aware warp scheduler such that it can facilitate better coordination between warp scheduling decisions taken at different cores. OWL enabled the warps across different cores to collectively access a larger number of memory banks concurrently, thereby improving the memory-level parallelism and easing the job of the memory scheduler in managing contention at the banks. Since the publication of OWL in ASPLOS 2013, warp scheduling has become an active area of research.

Prefetch-Aware Warp Scheduling [2]. Effectiveness of data prefetching is dependent on the prefetching accuracy as well as timeliness of prefetches. My analyses of the existing warp schedulers show that they do not coordinate well with the prefetching mechanisms because they happen to schedule consecutive warps accessing nearby cache blocks in close succession. Therefore, a simple prefetcher that could have prefetched nearby cache blocks with high accuracy will not contribute significantly to the performance because many of them will be tagged as late prefetches. To this end, I proposed a *prefetch-aware* warp scheduling policy that can coordinate with prefetching decisions in GPUs to better tolerate long memory latencies. This scheduler separates the scheduling of consecutive warps in time, and by not executing them in close succession, it enables effective incorporation of simple prefetching techniques for improving the overall GPU performance.

2.2 Managing Contention at Memory via Memory Scheduling

Memory access schedulers employed in GPUs implicitly assume that all requests from different cores are equally important. I showed that this assumption does not necessarily help in achieving: 1) the best performance when GPU cores concurrently execute threads belonging to a *single* application; and 2) the best system throughput and fairness when GPU cores concurrently execute threads belonging to *multiple* applications. To address these two scenarios, I proposed criticality-aware and application-aware memory scheduling techniques, respectively.

Criticality-Aware Memory Scheduling [3]. Shared resource contention causes significant variation in average memory latencies experienced by individual GPU cores. Due to this variation, the number of stalling warps belonging to the cores that suffer from higher memory access latencies is typically higher than that of other cores, making the former type of cores less latency tolerant, i.e., more *critical*. This implies that because different GPU cores have varying degrees of tolerance to latency during the execution of an application, their corresponding memory requests have varying degrees of criticality. Based on this observation, I proposed a criticality-aware scheduler that takes into account the criticality of memory requests, i.e., the latency-tolerance of the cores that generate memory requests, thereby improving the overall GPU performance over existing schedulers.

Application-Aware Memory Scheduling [4, 5, 6]. I find that an uncoordinated allocation of GPU resources among concurrently executing multiple applications can lead to significant degradation in system throughput and fairness. Therefore, it is imperative to make the GPU memory system aware of the application characteristics. To this end, I proposed two different application-aware memory scheduling policies. The first scheduler is developed with the aim of sharing the memory bandwidth across concurrent applications in a fair and efficient manner. The second scheduler is more sophisticated and is based on an analytical performance model for GPUs. I showed that the common use of misses-per-instruction (MPI) as a proxy for performance is not accurate for

many-threaded architectures and memory scheduling decisions based on *both* MPI and attained DRAM bandwidth are more effective in enhancing system throughput and fairness.

Research Funding. I gained good insight into the funding process by helping in writing a successful NSF grant proposal with Professors Chita Das, Mahmut Kandemir, and Onur Mutlu titled *Enabling GPUs as First-Class Computing Engines*. This grant funded some of my research on warp and memory scheduling in GPUs.

3 Other Research Contributions

Leveraging Emerging Memory Technologies [7]. High density, low leakage, and non-volatility are the attractive features of Spin Transfer Torque RAM (STT-RAM), which has made it a strong competitor against SRAM. However, high write latency and energy has impeded widespread adoption of STT-RAM. With the aid of detailed characterization of many multi-programmed and multi-threaded applications, I showed that trading-off STT-RAM's non-volatility (or data-retention time) can reduce write latency and energy overheads.
Performance, Energy, and Power Optimization Techniques for GPUs [8, 9, 10, 11, 12]. We showed that it is not always a good idea to allow the maximum level of concurrency (multi-threading), because it can cause severe contention in caches and memory. To this end, we devised many throttling mechanisms [8] that limit the number of concurrently executing warps on a core. We also extended these mechanisms for CPU-GPU heterogeneous architectures [9]. Furthermore, I contributed to efforts towards: 1) designing a configurable and application-aware GPU architecture [11]; and 2) enabling efficient data-compression in GPUs [12].
Leveraging Compiler Technology [13]. We looked at developing a compiler-runtime cooperative data layout optimization approach that takes as input an irregular program that has already been optimized for cache locality and generates an output code with the same cache performance but better row-buffer locality. We then investigated a more aggressive strategy that sacrifices some cache performance in order to further improve row-buffer performance. The ultimate goal of this strategy was to find the right trade-off point between cache and row-buffer performance for improving the overall application performance.

4 Future Research Directions

I believe in the concept of ubiquitous computing and envision that all types of computing systems will take advantage of GPUs by considering them as an important class of computing citizens instead of mere co-processors. To realize this vision, my future research is focused on inventing novel architectures, scheduling mechanisms, hardware/software interfaces as well as investigating system and security issues related to GPU-based systems. Specifically, I am planning to pursue the following research directions in a synergistic fashion.
(I) Futuristic GPU Computing. In order to facilitate efficient GPU computing, the community is exploring features such as 1) dynamic parallelism, where a GPU can generate new work for itself; 2) Hyper-Q, where multiple CPUs can launch computations on a GPU; and 3) hybrid architectures consisting of CPUs and GPUs on the same die. However, many of these concepts are still in their infancy, and I am interested in an in-depth understanding of the design space for each of these avenues and their combinations. I am also interested in developing new execution models and architectures involving multiple GPUs and other kinds of accelerators. I envision these architectures will also concurrently execute multiple applications, potentially originating from different users. In this context of multi-application execution, I plan to research on 1) designing an application-aware on-chip network fabric and memory hierarchy; 2) developing shared resource management, scheduling and concurrency management techniques; and 3) designing light-weight and efficient hardware and software support that deals with virtualization, security, and other system issues. As a part of my long-term research, I am also interested in extending these techniques to situations where applications with different latency and bandwidth demands are executed concurrently on mobile and wearable devices under stricter energy and power constraints.
(II) Near-Data Computing in GPU-based systems. Near-data computing architectures are built on the notion that moving computation near the data is far more beneficial in terms of performance and energy efficiency than moving data near the computation. In the context of data-intensive computing, such architectures can be useful as they can significantly cut down the movement of data between memory/storage and computation units, and therefore, can conserve precious bandwidth. However, there are many questions that need to be answered

before realizing such architectures. Some research questions that I plan to investigate are: 1) Which parts of the CUDA/OpenCL application code triggers significant amounts of data movement?; 2) Between which components (e.g., between CPUs and GPUs, or between multiple GPUs, or between different levels of the CPU/GPU memory hierarchy) is the data movement the most expensive in terms of performance and energy; 3) Given the cost and the amount of data movement, between which components should the movement of data be minimized?; 4) Which parts of the computation can be computed near memory/storage; 5) What architectural enhancements are required to perform the computation near memory/storage?; and 6) How do I leverage my past insights from my dissertation work to design efficient techniques to co-schedule data and computation for minimizing the data movement? I am also planning to explore the opportunities for employing approximate computing and leveraging different emerging memory/storage technologies in conjunction with near-data computing.

(III) Commonality-Aware GPU Computing. There is an increasing trend to co-host multiple applications or games from different users on the same GPU cloud platform. For example, many service providers have started to use cloud gaming technology (e.g., NVIDIA GRID) as the foundation for their on-demand Gaming as a Service (GaaS) solution. In this context, a multi-player gaming environment might require rendering of similar scenes/objects separately for every player. Such redundant rendering computations are not only costly from performance and energy perspective but also can consume significant GPU memory space. To this end, I am interested to pursue at least two major aspects towards designing commonality-aware GPU systems. The first aspect is to design mechanisms for detecting computation and data commonality across concurrent applications. The second aspect is to design scheduling mechanisms that can effectively co-schedule applications possessing high-commonality on the same GPU hardware. I believe both aspects should be pursued synergistically for enabling inter-application optimizations for improving the overall performance and energy efficiency. As a part of my long-term research, I am also interested in addressing the security vulnerabilities that might arise because of commonality-aware GPU computing.

While working on the above future research directions and my ongoing projects, I intend to develop collaborations with other researchers working in computer architecture, emerging memory/storage technologies, compilers, programming languages, systems, and security. I also intend to strengthen my existing collaborations with NVIDIA Research, Intel Labs, AMD Research, and CMU. Finally, I hope to leverage the strengths of the new working environment to identify prospective research topics and develop more collaborations.

References

[1] A. Jog, O. Kayiran, N. C. Nachiappan, A. K. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," in *ASPLOS*, 2013.

[2] A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "Orchestrated Scheduling and Prefetching for GPGPUs," in *ISCA*, 2013.

[3] A. Jog, O. Kayiran, A. Pattnaik, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "Exploiting Core-Criticality for Enhanced Performance in GPUs," in *Submission*, 2015.

[4] A. Jog, E. Bolotin, Z. Guz, M. Parker, S. W. Keckler, M. T. Kandemir, and C. R. Das, "Application-aware Memory System for Fair and Efficient Execution of Concurrent GPGPU Applications," in *GPGPU*, 2014.

[5] A. Jog, O. Kayiran, T. Kesten, A. Pattnaik, E. Bolotin, N. Chatterjee, S. Keckler, M. T. Kandemir, and C. R. Das, "Anatomy of GPU Memory System for Multi-Application Execution," in *Submission*, 2015.

[6] E. Bolotin, Z. Guz, A. Jog, S. W. Keckler, and M. Parker, "A US patent related to concurrent execution of multiple applications on GPUs is filed and currently pending (assigned to NVIDIA Corp.)," 2014.

[7] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache Revive: Architecting Volatile STT-RAM Caches for Enhanced Performance in CMPs," in *DAC*, 2012.

[8] O. Kayiran, A. Jog, M. T. Kandemir, and C. R. Das, "Neither More Nor Less: Optimizing Thread-level Parallelism for GPGPUs," in *PACT*, 2013.

[9] O. Kayiran, N. C. Nachiappan, A. Jog, R. Ausavarungnirun, M. T. Kandemir, G. H. Loh, O. Mutlu, and C. R. Das, "Managing GPU Concurrency in Heterogeneous Architectures," in *MICRO*, 2014.

[10] O. Kayiran, N. C. Nachiappan, A. Jog, R. Ausavarungnirun, M. T. Kandemir, G. H. Loh, O. Mutlu, and C. R. Das, "Concurrency Management in Heterogeneous Architectures," in *Submission*, 2015.

[11] O. Kayiran, A. Jog, A. Pattnaik, R. Ausavarungnirun, M. T. Kandemir, G. H. Loh, O. Mutlu, and C. R. Das, "Characterization and Identification of Data-path Under-utilization for GPGPU Power Management.," in *Submission*, 2015.

[12] N. Vijaykumar, G. Pekhimenko, A. Jog, A. Bhowmick, O. Mutlu, C. Das, M. T. Kandemir, T. Mowry, and R. Ausavarungnirun, "Enabling Efficient Data Compression in GPUs," in *Submission*, 2015.

[13] W. Ding, M. Kandemir, D. Guttman, A. Jog, C. R. Das, and P. Yedlapalli, "Trading Cache Hit Rate for Memory Performance," in *PACT*, 2014.