Connor Smith

Final Project Proposal: GPUs

GPUs were originally designed to accelerate the creation of images for displays, since general-purpose CPUs lack the performance necessary to render enough frames per second to create a satisfactory experience. GPUs contain hundreds or even thousands of "cores" that can operate on blocks of data in parallel, exploiting the great amount of parallelism implicit in processing visual data. Within the past decade or so, GPUs have begun to be used frequently on more general-purpose (GPGPU) applications that profit from a high number processing units.

In order to fully take advantage of a GPU architecture, the processing of data must be highly parallelizable, and the load on each unit of execution should be balanced. Traditional or naive implementations use warps (threads of SIMD instructions) that are all the same size. Breaking up the work to be done in this way performs well for inherently balanced loads, such as processing the next frame for a display. However, many applications that could benefit from the parallelism of GPUs cannot be divided so evenly. Heterogeneous workloads can cause imbalance in resource utilization and negatively impact performance. A number of different methods of managing warp size and scheduling have been proposed to mitigate this problem [1-7].

Modern GPUs consume a significant amount of power between the "cores" themselves and memory. In the past the majority of the power was spent on the computations themselves, but as GPUs have shifted to new applications, the power spent in memory and data movement has grown to significant levels. Better allocation of resources can better balance the power distribution and lower the overall dissipation, and there are several strategies to accomplish this in recent literature [8-10].

Limitations on the memory bandwidth of GPUs results in a bottleneck that restricts performance. Prioritizing different memory requests and managing CPU/GPU memory interaction can lead to much higher efficiency. Dynamic scheduling methods can detect memory saturation and provide better caching, while different page placement policies can improve performance on heterogeneous systems. [11-13]

In addition to the microarchitectures and runtimes of GPU systems, the code written by the programmer and generated by the compiler also has a significant impact on performance. There are several compiler-generated mappings from code patterns to GPU architectures, each of which are optimal under different circumstances [14]. Assumptions made by the programmer can also have a major impact, especially when the assumptions are inaccurate. [15]

[1] Nandita Vijaykumar et al. "A Case for Core-Assisted Bottleneck Acceleration in GPUs: Enabling Flexible Data Compression with Assist Warps" in the 42nd International Symposium on Computer Architecture, Portland, OR, 2015, pp. 41-53.

[2] Shin-Ying Lee et al. "CAWA: Coordinated Warp Scheduling and Cache Prioritization for Critical Warp Acceleration of GPGPU Workloads" in the 42nd International Symposium on Computer Architecture, Portland, OR, 2015, pp. 515-527.

[3] Jin Wang et al. "Dynamic Thread Block Launch: A Lightweight Execution Mechanism to Support Irregular Applications on GPUs" in the 42nd International Symposium on Computer Architecture, Portland, OR, 2015, pp. 528-540.

[4] Sethia Ankit and Scott Mahlke. "Equalizer: Dynamic Tuning of GPU Resources for Efficient Execution" in the 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK, 2014, pp. 647-658.

[5] Onur Kayiran et al. "Managing GPU Concurrency in Heterogeneous Architectures" in the 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK, 2014, pp. 114-126.

[6] Ji Kim and Christopher Batten. "Accelerating Irregular Algorithms on GPGPUs Using Fine-Grain Hardware Worklists" in the 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK, 2014, pp. 75-87.

[7] Timothy G. Rogers et al. "A Variable Warp Size Architecture" in the 42nd International Symposium on Computer Architecture, Portland, OR, 2015, pp. 489-501.

[8] Indrani Paul et al. "Harmonia: Balancing Compute and Memory Power in High-Performance GPUs" in the 42nd International Symposium on Computer Architecture, Portland, OR, 2015, pp. 54-65.

[9] Gene Wu et al. "GPGPU Performance and Power Estimation Using Machine Learning" in the IEEE 21st International Symposium on High Performance Computer Architecture, Burlingame, CA, 2015, pp. 564-576.

[10] Sangpil Lee et al. "Warped-Compression: Enabling Power Efficient GPUs through Register Compression" in the 42nd International Symposium on Computer Architecture, Portland, OR, 2015, pp. 502-514.

[11] Ankit Sethia et al. "Mascar: Speeding up GPU Warps by Reducing Memory Pitstops" in the IEEE 21st International Symposium on High Performance Computer Architecture, Burlingame, CA, 2015, pp. 174-185.

[12] Neha Agarwal et al. "Page Placement Strategies for GPUs within Heterogeneous Memory Systems" in the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, New York, NY, 2015, pp. 607-618.

[13] Neha Agarwal et al. "Unlocking Bandwidth for GPUs in CC-NUMA Systems" in in the IEEE 21st International Symposium on High Performance Computer Architecture, Burlingame, CA, 2015, pp. 354-365.

[14] HyoukJoong Lee et al. "Locality-Aware Mapping of Nested Parallel Patterns on GPUs" in the 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK, 2014, pp. 63-74.

[15] Jade Alglave et al. "GPU concurrency Weak behaviours and programming assumptions" in the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, New York, NY, 2015, pp. 577-592.