Preemptive Thread Block Scheduling with Online Structural Runtime Prediction for Concurrent GPGPU Kernels

Sreepathi Pai
The University of Texas at Austin
Austin, Texas
sreepai@ices.utexas.edu

R. Govindarajan
Indian Institute of Science
Bangalore, India
govind@serc.iisc.in

Matthew J. Thazhuthaveetil
Indian Institute of Science
Bangalore, India
mjt@serc.iisc.in

ABSTRACT

Recent NVIDIA Graphics Processing Units (GPUs) can execute multiple kernels concurrently. On these GPUs, the thread block scheduler (TBS) currently uses the FIFO policy to schedule thread blocks of concurrent kernels. We show that the FIFO policy leaves performance to chance, resulting in significant loss of performance and fairness. To improve performance and fairness, we propose use of the preemptive Shortest Remaining Time First (SRTF) policy instead. Although SRTF requires an estimate of runtime of GPU kernels, we show that such an estimate of the runtime can be easily obtained using online profiling and exploiting a simple observation on GPU kernels' grid structure. Specifically, we propose a novel Structural Runtime Predictor. Using a simple Staircase model of GPU kernel execution, we show that the runtime of a kernel can be predicted by profiling only the first few thread blocks. We evaluate an online predictor based on this model on benchmarks from ERCBench, and find that it can estimate the actual runtime reasonably well after the execution of only a single thread block. Next, we design a thread block scheduler that is both concurrent kernel-aware and uses this predictor. We implement the Shortest Remaining Time First (SRTF) policy and evaluate it on two-program workloads from ERCBench. SRTF improves STP by 1.18x and ANTT by 2.25x over FIFO. When compared to MPMax, a state-of-the-art resource allocation policy for concurrent kernels, SRTF improves STP by 1.16x and ANTT by 1.3x. To improve fairness, we also propose SRTF/Adaptive which controls resource usage of concurrently executing kernels to maximize fairness. SRTF/Adaptive improves STP by 1.12x, ANTT by 2.23x and Fairness by 2.95x compared to FIFO. Overall, our implementation of SRTF achieves system throughput to within 12.64% of Shortest Job First (SJF, an oracle optimal scheduling policy), bridging 49% of the gap between FIFO and SJF.

PACT'14, August 24–27, 2014, Edmonton, AB, Canada.
ACM 978-1-4503-2809-8/14/08.
http://dx.doi.org/10.1145/2628071.2628117.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features; C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)

1. INTRODUCTION

Concurrent kernel execution in Graphics Processing Units (GPUs), such as the NVIDIA Fermi and Kepler families, exploits task-level parallelism among independent GPU kernels primarily by *space-sharing*. Each GPU kernel exclusively occupies the resources (i.e. registers, thread contexts, shared memory) it needs and concurrent execution is only achieved if there are enough resources left over to accommodate any concurrent kernel. Therefore, NVIDIA positions concurrent kernel execution as allowing "programs that execute a number of small kernels to utilize the whole GPU".

Programs whose kernels already utilize the whole GPU ("large kernels") – most kernels in Rodinia, Parboil2, etc. – do not benefit from concurrent kernel execution and execute serially. However, since every GPU kernel is organized as a hierarchy, a *grid* of *thread blocks* and executes at thread block granularity, recent works on concurrent GPU kernel execution have been able to achieve concurrent execution for large kernels. However, on current GPU hardware, concurrent kernels are executed in arrival order (i.e. FIFO). We show that FIFO is a poor choice and that preemptive scheduling policies such as those presented herein improve throughput and turnaround time.

This work focuses on the *Thread Block Scheduler* (TBS), the first-level hardware scheduler in GPUs. In particular, we propose two thread block scheduling policies – Shortest Remaining Time First (SRTF) and SRTF/Adaptive – for concurrent GPGPU workloads. These policies use estimates of kernel runtime to determine their scheduling decisions when executing concurrent workloads, we propose a novel online runtime predictor for GPU grids to provide these estimates of runtime. Our predictor is based on an observation that the GPU kernel execution time is approximated by a simple linear function of the durations of its thread blocks, and therefore online profiling of the first few thread blocks suffices to estimate the kernel execution time for scheduling.

2. STRUCTURAL RUNTIME PREDICTION

Our principle of *Structural Prediction* on which our online predictor is based essentially views the execution of a grid's N thread blocks (all of which have the same code) as N repeated executions of the same program. So profiling the first few thread blocks of a grid lets us predict the behaviour of the remaining thread blocks. In this work, the runtime of a thread block is used to predict the runtime of the whole kernel. We call this *Structural Runtime Prediction*.

Assuming that all thread blocks take the same time to complete, our *Staircase model* approximates runtime using the following equation for runtime $T = (\lceil N/R \rceil)\, t \quad (1)$.

Here, $N = B/N_{SM}$ where the original B thread blocks are assumed to be evenly distributed across N_{SM} streaming multiprocessors (SMs). The grid's maximum residency, R, can be determined at launch time. The value of t can be obtained by sampling, possibly as soon as a single thread block finishes execution. However, for timely predictions, the grid must execute more than R blocks per SM.

To evaluate the Staircase model, we instrumented major kernels in the Parboil2 and the ERCBench suites to record the start and end time of each thread block and the SM it executed on and ran them on a Fermi-based NVIDIA Tesla C2070, with a quad-core Intel Xeon W3550 CPU, 16GB RAM and running Debian Linux 6.0 (64-bit) with CUDA driver 295.41 and CUDA runtime 4.2.

Using these recorded end times, linear regression resulted in normalized predictions between 0.99x to 1.11x of actual runtime for ERCBench and 0.87x to 1.13x for Parboil2. This strongly supports our hypothesis that GPU kernel runtime is a linear function. For the Staircase model, predictions normalized to actual runtime lie between 0.54x to 1.18x for ERCBench and 0.39x to 1.49x for Parboil2. If we exclude outliers, normalized predictions are between 0.66x and 1.18x for ERCBench and 0.6x and 1.2x for Parboil2.

The major causes for inaccuracy for the Staircase model are (i) non-Staircase model behaviour precipitated by the the first R blocks each ending at different times leading to staggering in the starting times of subsequent blocks; (ii) differing work per thread block because thread block durations depended on *value* of their inputs, e.g. `render` in RayTrace; (iii) poor static load-balancing across SMs. Concurrent execution affected throughput due to changes in residencies and hence changed t and total runtime. Co-running kernels and the exact resource split between them also affected t.

3. SCHEDULER AND POLICIES

The Simple Slicing (SS) runtime predictor is an online, concurrent-kernel aware predictor based on Equation 1 that tries to correct for systematic inaccuracies. Its prediction of runtime is an estimate of how much time a kernel would take to complete if it was running from now (i.e. the time at which the prediction is made) to completion, under the current conditions (t, residency and co-runners).

Since t can change as the kernel executes, we split the execution into multiple *slices*. Currently, each slice is demarcated by kernel launches and kernel endings. We assume that t is constant within a slice when predicting timings for blocks in that slice. Finally, as each SM can vary in behaviour, our predictor predicts runtimes for each kernel on a per-SM basis.

$$\begin{aligned} Pred\_Cycles = &Active\_Kernel\_Cycles + \\ &\frac{(Total\_Blocks - Done\_Blocks) \times t}{Resident\_Blocks} \end{aligned} \quad (2)$$

$Total\_Blocks$ and $Done\_Blocks$ count the number of thread blocks assigned to and completed on a SM respectively. The actual runtime of a kernel so far, $Active\_Kernel\_Cycles$, is used to correct predictor drift. For ERCBench, the SS predictor is accurate to between 0.48x to 1.08x of actual runtime after observing t for only one thread block.

Scheduler	STP	ANTT	Fairness
FIFO	1.35	3.66	0.19
MPMax	1.37	2.15	0.36
SRTF	1.59	1.63	0.52
SRTF/Adaptive	1.51	1.64	0.56
SJF	1.82	1.13	0.80

Table 1: Geomean STP, ANTT and Fairness for various scheduling policies. Note that ANTT is a lower-is-better metric.

We evaluate four policies: *Shortest Remaining Time First (SRTF)* which executes a single kernel at a time pre-empting the running kernel if newly arrived kernels are deemed to finish faster (using a sampling run); *SRTF/Adaptive* which statically shares resources between running kernels if it deems SRTF unfair; *Just-in-time MPMax*, a resource-allocation policy; and *FIFO* based on the NVIDIA Fermi Thread Block Scheduler. SRTF and SRTF/Adaptive use estimates of runtime provided by the Simple Slicing predictor to guide their scheduling decisions.

4. EVALUATION

The primary metrics evaluated are are system throughput (STP), average normalized turnaround time (ANTT)and the StrictF metric for fairness. StrictF is defined as the ratio of minimum slowdown to maximum slowdown, with a value of 1 indicating high fairness.

We use all possible 28 2-program workloads from the ERCBench suite running them on GPGPU-Sim simulator (3.2.0) modified to support execution of concurrent kernels. A zero sampling (ZS) experiment provides accurate runtimes to the SRTF to evaluate sensitivity to predictor accuracy. The results are summarized in Table 1 with full discussion in [1].

SRTF is very tolerant of errors in the simple slicing predictor. Zero-sampling improves STP by only 3% to 1.64 and ANTT by 22% to 1.33, thus we find that the inability to preempt running thread blocks is the major performance limiter for GPU scheduling.

5. CONCLUSION

We presented a novel online runtime predictor for GPU kernels that exploited the structure of kernel grids to obtain runtime predictions by observing thread block durations. We used it to build a thread block scheduler with runtime aware policies, SRTF and SRTF/Adaptive, that were found superior to the other realizable policies in terms of system throughput, turnaround time and fairness. Compared to FIFO, SRTF improved STP by 1.18x and ANTT by 2.25x. SRTF also outperformed MPMax with improvements of 1.16x in STP and 1.3x in ANTT. Our SRTF/Adaptive policy achieved the highest fairness among all the realizable policies, 2.95x better than FIFO. Finally, SRTF bridged 49% of the gap between FIFO and SJF, approaching to within 12.64% of SJF's throughput.

6. REFERENCES

[1] S. Pai, R. Govindarajan, and M. J. Thazhuthaveetil, "Preemptive Thread Block Scheduling with Online Structural Runtime Prediction for Concurrent GPGPU Kernels," in *CoRR* abs/1406.6037, 2014.