POSTER: A Microbenchmark to Study GPU Performance Models

Vasily Volkov
NVIDIA Corporation
vvolkov@nvidia.com

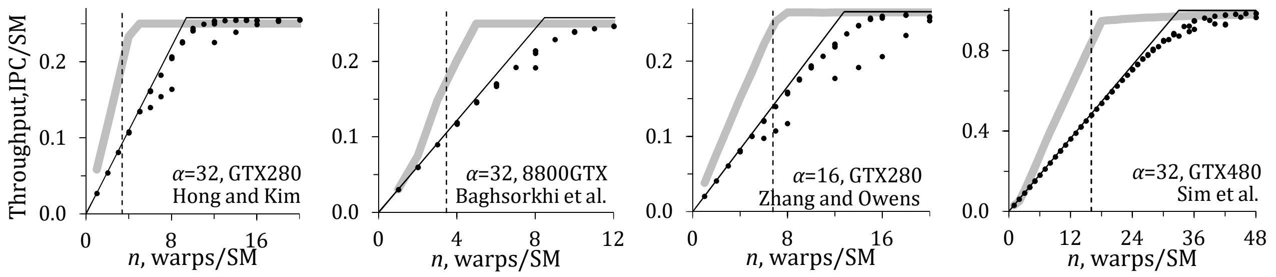

Figure 1. We use a simple kernel to examine how various GPU performance models model key hardware functions. Grey lines show the indicated models. Dots show the experimental data. Dashed lines show the occupancy estimates according to the vendor's guide.

Abstract

Basic microarchitectural features of NVIDIA GPUs have been stable for a decade, and many analytic solutions were proposed to model their performance. We present a way to review, systematize, and evaluate these approaches by using a microbenchmark. In this manner, we produce a brief algebraic summary of key elements of selected performance models, identify patterns in their design, and highlight their previously unknown limitations. Also, we identify a potentially superior method for estimating performance based on classical work.

1 Introduction

Analytical performance models have many applications, including understanding inefficiencies in both software and hardware. It is therefore important to have a mature and reliable performance modeling methodology, which was not only shown to be accurate on a limited set of realistic workloads, but is also based on carefully reviewed and widely accepted principles.

Reviewing the principles that underlie GPU performance models poses a challenge because of the complexity and diversity of the solutions. The approaches share little in common, they often require several pages to explain, and some include more than 20 equations. In this work, we suggest a way to improve our understanding of prior work by considering it in application to a simpler problem: estimating performance of a synthetic kernel that uses only a few critical processor functions. This allows producing a concise algebraic (as opposed to qualitative) survey of performance models and gaining a new insight into their accuracy.

We limit our attention to the following few basic features of the GPU architecture: fine-grained multithreading, CUDA cores, and fully coalesced memory access with no caching. These features are present in all CUDA-capable NVIDIA GPUs released over the last decade and will likely persist in the future.

PPoPP '18, February 24–28, 2018, Vienna, Austria
© 2018 Copyright is held by the owner/author.
ACM ISBN 978-1-4503-4982-6/18/02…$15.00
https://doi.org/10.1145/3178487.3178536

The numerous features that are excluded from this workload include control and memory divergence, synchronization, double-precision and transcendental arithmetic operations, atomic operations, and others. We also ensure that experimental results are not substantially polluted by kernel launch overhead, thread block retirement overhead, TLB misses, register file bank conflicts, and dynamic frequency scaling. Finally, we leave out of scope instruction-level parallelism and memory writes as some of the reviewed performance models do not consider them.

What remains is a synthetic mix of arithmetic and memory instructions executed in a highly controlled setup. It has only two parameters: occupancy equal n warps per SM, and arithmetic intensity equal α arithmetic instructions per memory instruction.

2 Survey of Performance Models

The given workload reduces even the more complex performance models to only a few equations. This helps to both differentiate the models and identify common patterns in their design.

One family of the approaches is based on latency and throughput bounds. If arithmetic latency is A and memory latency is L, instruction throughput cannot exceed $n \cdot (\alpha+1) / (\alpha A + L)$. If the peak rate of issuing instructions is I, and peak rates of executing arithmetic and memory instructions are T and B, respectively, then instruction throughput cannot exceed $\min(I, T \cdot (1+1/\alpha), B \cdot (1+\alpha))$. This approach dates back to the 1970s [1, p. 240]. Below, we denote $b = 1/B$ and $t = 1/T$.

The Hong and Kim model [2] reduces to the following few equations in our case (we omit "Case 5," which is inconsistent with the rest of the model):

$$CWP = \min\left(n,\ 1 + \frac{L \cdot T}{\alpha+1}\right),\quad MWP = \min(n,\ L \cdot B),$$

$$\begin{aligned} &Throughput = n \cdot (\alpha+1) / ((\alpha+1) \cdot t + L) && \text{if } CWP = MWP = n, \\ &Throughput = B \cdot (\alpha+1) && \text{if } CWP > MWP, \\ &Throughput = T && \text{if } CWP < MWP. \end{aligned}$$

We may recognize one latency- and two throughput-bound cases. It can be shown that the definitions of *CWP* and *MWP* are related to Little's law (*concurrency* = *latency* × *throughput*).

Chen and Aamodt [3] suggest four models for fine-grained multithreaded processors. If λ is the single-threaded throughput, and $\lambda_k(n)$ is the throughput when n threads are executed at the same time, then their first three models can be written as:

$$\lambda_1(n) = \lambda, \qquad \lambda_2(n) = n \cdot \lambda, \qquad \lambda_3(n) = 1 - (1-\lambda)^n.$$

The second model is similar to the latency bound. The third model assumes a similar near-linear growth at small n, which gradually saturates at a throughput bound equal to 1 IPC as n increases.

The Huang et al. model [4] is formulated for two warp scheduling policies: round-robin and greedy-then-oldest. Assuming the round-robin scheduling policy, the model reduces to

$$Throughput = n \cdot (\alpha + 1) / (\alpha A + L),$$

which is identical to the latency bound.

Another family of approaches is to separately compute the time to execute all arithmetic instructions and the time to execute all memory instructions and then take the maximum of them. This includes **the Sim et al. model** [5] and **the Zhang and Owens model** [6]. The latter can be reduced to the following equations:

$$Time = \max(\, T_{comp}, T_{mem} \,), \qquad T_{comp} = n \cdot \alpha \cdot \tau, \qquad T_{mem} = n \cdot \mu.$$

Here, μ and τ are the times it takes to execute a memory and an arithmetic instruction, respectively; they are found by using microbenchmarking. For many GPUs, we find $\tau \approx \min(\, A\,/\,n,\, t\,)$. The overall throughput in IPC/SM is found as $n \cdot (\alpha+1) / Time$.

The Baghsorkhi et al. model [7] similarly assumes that each arithmetic instruction takes $\tau = \min(\, A\,/\,n,\, t\,)$ cycles to execute. The total execution time per warp is found as the length of the longest path in the following graph (where we denote $alu = \alpha \cdot \tau$):

$$\begin{tikzcd} {} \arrow[r] \arrow[r, dashed, bend left] & C \arrow[r, "alu"] & M \arrow[r, "\max(t{,}\,b-alu)"] \arrow[r, dashed, bend left=40, "L-(n-1)\cdot(alu+t)"] & C \arrow[r, "alu"] & M \arrow[r] \arrow[r, dashed, bend left] & {} \end{tikzcd}$$

Finally, **the vendor's programming guide** [8] suggests the following estimate for the occupancy needed to attain maximum throughput; note the similarity with the definition of *CWP* above:

$$warps\ needed \approx L \cdot T / \alpha.$$

For comparison, the solutions for coarse-grained multithreaded processors [9] suggest that if each thread executes for R cycles until stalling on a memory access and the context switch overhead is C cycles, then maximum throughput is attained at the number of threads equal to $1 + L/(R+C)$. If $C = 0$ as on GPUs, and $R = \alpha \cdot t$, this solution becomes $1 + L \cdot T/\alpha$, which is similar to the above.

3 Evaluation and Pitfalls

The proposed workload exposes several inherent limitations in well-known performance models. Fig. 1 shows that many performance models may substantially overestimate throughput in the latency-bound case and, as a result, underestimate the occupancy needed to attain maximum throughput. Fig. 2 shows that the Baghsorkhi et al. model may severely underestimate throughput in the memory-intensive case and that the Huang et al. model may violate hardware limits for issue throughput (here, 4 IPC/SM). The Huang et al. model also may produce negative throughput estimates when memory accesses are diverging.

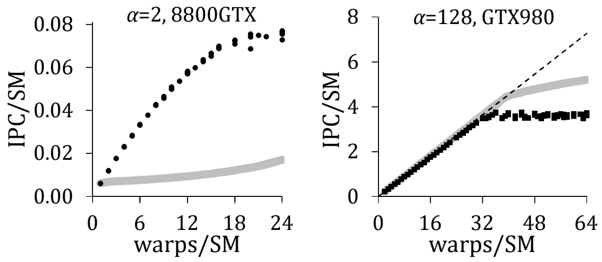

Figure 2. The Baghsorkhi et al. model (left) may substantially underestimate throughput. The Huang et al. model (right) may overestimate throughput if assuming round-robin (dashed line) or greedy-then-oldest (grey line) warp scheduling policy.

We tracked errors to the following causes: (a) ignoring arithmetic latencies, which are small but may accumulate to large numbers; (b) ignoring memory bandwidth, which may limit throughput even if all accesses are fully coalesced; (c) assuming that NVIDIA GPUs can switch to a different warp only on long latency events, not on each cycle; and (d) using wrong concurrency in Little's law: warp concurrency (occupancy), instruction concurrency, arithmetic instruction concurrency, and memory instruction concurrency are all different quantities.

We found that for this workload, using the classical asymptotic bounds for concurrent systems [1] as throughput estimates produces a higher accuracy than when using the other reviewed models. These estimates are shown in Fig. 1 as a thin black line.

4 Conclusion

We used a microbenchmark to classify approaches to modeling GPU performance and exposed previously unknown limitations. Future work includes investigating the practical implications of our findings, such as whether the found limitations impair predictions for realistic workloads, and whether the asymptotic bounds may produce accurate throughput estimates in such cases.

References

[1] Denning, P. J., and Buzen, J. P. 1978. The Operational Analysis of Queuing Network Models. *ACM Computing Surveys 10*, 3, 225–261.
[2] Hong, S., and Kim, H. 2009. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. In *International Symposium on Computer Architecture* (ISCA '09), 152–163.
[3] Chen, X. E., and Aamodt, T. M. 2009. A first-order fine-grained multithreaded throughput model. In *International Symposium on High Performance Computer Architecture* (HPCA '09), 329–340.
[4] Huang, J.-C., Lee, J. H., Kim, H., and Lee, H.-H. S. 2014. GPUMech: GPU performance modeling technique based on interval analysis. In *International Symposium on Microarchitecture* (MICRO-47), 268–279.
[5] Sim, J., Dasgupta, A., Kim, H., and Vuduc, R. 2012. A performance analysis framework for identifying potential benefits in GPGPU applications. In *Symposium on Principles and Practice of Parallel Programming* (PPoPP '12), 11–22.
[6] Zhang, Y., and Owens, J. D. 2011. A quantitative performance analysis model for GPU architectures. In *International Symposium on High Performance Computer Architecture* (HPCA '11), 382–393.
[7] Baghsorkhi, S. S., Delahaye, M., Patel, S. J., Gropp, W. D., and Hwu, W. W. 2010. An adaptive performance modeling tool for GPU architectures. In *Symposium on Principles and Practice of Parallel Programming* (PPoPP '10), 105–114.
[8] NVIDIA. 2017. CUDA C Programming Guide v9.1. November 2017.
[9] Saavedra-Barrera, R., Culler, D., and von Eicken, T. 1990. Analysis of multithreaded architectures for parallel computing. In *Symposium on Parallel Algorithms and Architectures* (SPAA '90), 169–178.