A SIMT Analyzer for Multi-Threaded CPU Applications

Ahmad Alawneh
ECE department
Purdue University
aalawneh@purdue.edu

Mahmoud Khairy
ECE department
Purdue University
abdallm@purdue.edu

Timothy G. Rogers
ECE department
Purdue University
timrogers@purdue.edu

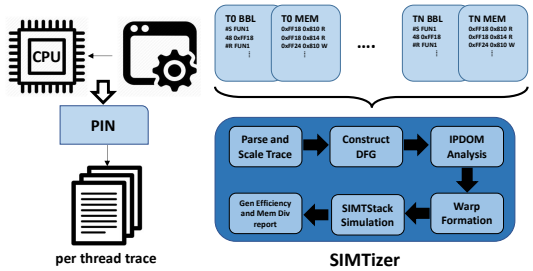
***Abstract*—The use of GPUs for general purpose applications has drastically increased. However, the performance gain from porting multithreaded CPU workloads to massively parallel SIMT-based accelerators, like GPUs, is often unpredictable. Even with enough parallelism, programmers do not know if their CPU code will run well on a GPU without first investing the effort to refactor it into a GPGPU programming language. Most of this unpredictability stems from two key side-effects of the GPU's energy-efficient SIMT hardware: control-flow and memory divergence.**

To alleviate this issue, we propose SIMTec, an analysis tool that computes the control-flow and memory divergence of arbitrary pre-compiled CPU binaries. The tool constructs and analyzes a dynamic control flow graph of the application, batches threads into warps and emulates the operation of a SIMT stack for each warp to compute the projected SIMT efficiency. Given each warp's execution mask, memory coalescing is computed using the addresses accessed by memory instructions from parallel threads. The tool reports the SIMT efficiency and memory divergence characteristics.

We validate SIMTec using a suite of 11 applications with both x86 CPU and CUDA GPU implementations on an NVIDIA Volta V100, demonstrating that SIMTec has a correlation factor of 1.00 and 0.98 for SIMT efficiency and memory divergence, respectively. To demonstrate the predictive power of SIMTec, we explore another 16 CPU workloads for which there is no 1:1 GPU implementation. We perform case studies on these applications that range from compute-intensive thread-parallel workloads to cloud-based request-parallel microservices. Using SIMTec, we demonstrate that many of these CPU-only workloads are amenable to SIMT acceleration as-is.

I. Introduction

Single Instruction Multiple Thread (SIMT) hardware, like Graphics Processing Units (GPUs), has been widely adopted in many areas, including graphics, High Performance Computing (HPC) and machine learning. The Single Program Multiple Data (SPMD) pattern available in these workloads make them amenable to lock-step execution on SIMT hardware, as threads execute the same program code and exhibit similar control flow. Moreover, these workloads show regular memory behavior which increases memory coalescing opportunities. These program characteristics have led to significant performance and energy efficiency gains when these workloads are ported to SIMT-based GPU hardware [1]. The inherent efficiency in SIMT hardware comes from: (i) amortizing the pipeline front-end overhead by fetching and decoding each instruction only once for all the threads in the same warp, and (ii) generating less memory traffic to the memory system by coalescing accesses from threads in the same warp.



(a) Step 1: Per-thread PIN trace generation (b) Step 2: Control flow and memory divergence analysis

Fig. 1: SIMTec architecture.

However, these efficiency gains cannot be captured by all parallel programs. The grouping of threads into warps for lock-step execution creates significant penalties for workloads where parallel threads traverse different control-flow paths or access dissimilar data. Often, the greatest barrier to the adoption of the GPU as an accelerator for arbitrary parallel workloads is that the effects of these inefficiencies cannot be evaluated without porting the code to a General Purpose Graphics Processing Unit (GPGPU) language and evaluating the results on real hardware. The porting process takes significant effort and often results in programs that are less efficient than their parallel CPU counterparts. As a result, the exercise is often only attempted on codes with obviously regular control-flow and access patterns, leaving the acceleration opportunities in many highly-parallel applications unexplored.

II. System Overview

To remove the high initial evaluation barrier and uncover new multithreaded CPU workloads that might benefit from SIMT hardware, we propose SIMTec. SIMTec traces the control-flow and memory access pattern of unaltered CPU binaries to generate a per-function breakdown of a parallel application's SIMT efficiency and memory divergence level. Figure 1 depicts the end-to-end flow of our SIMTec tool. First, we use Intel's x86 PIN tool [2] to generate the dynamic control flow graph and memory accesses pattern for each CPU thread. Second, these traces are then fed up to our backend that

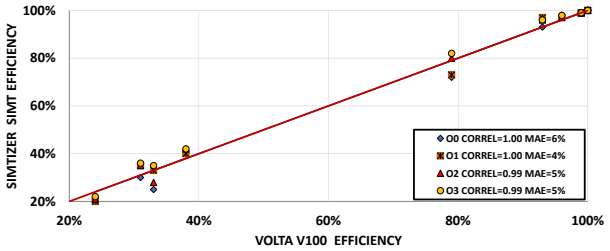


Fig. 2: SIMTec SIMT control flow efficiency correlation.

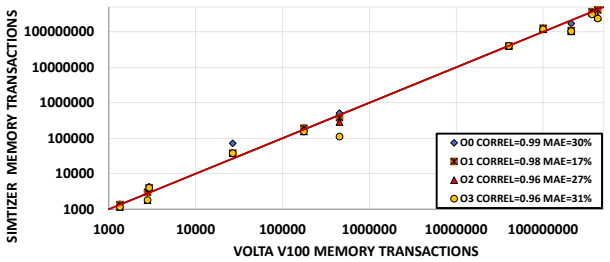


Fig. 3: SIMTec memory divergnce correlation.

groups the threads into warps based on a configurable batching algorithm. Then, SIMTec runs a stack-based Immediate Post-Dominator (IPDOM) reconvergence analysis for the grouped threads [1] to calculate the SIMT control-flow efficiency. The tool also reports the application's predicted memory divergence by determining the number of memory transactions for each x86 instruction that generates memory accesses after employing a memory coalescing algorithm similar to the ones used in GPU hardware [1]. The tool reports the SIMT efficiency and memory divergence characteristics for each function in the application, which gives the programmer the ability to identify SIMT-specific bottlenecks. It also explores the effect of warp size on their efficiency and identify GPU-specific bottlenecks in the source code.

III. Correlation Results

To demonstrate the validity of our tool, we correlate the reported metrics with real SIMT hardware using 11 multi-threaded CPU workloads that have OpenMP/Pthread implementations and CUDA implementations that mirror each other. On these applications, SIMTec uses the CPU execution of the programs to predict the GPU's SIMT efficiency and memory divergence characteristics with a mean absolute error of 4% and 17% respectively, as depicted in Figures 2 and 3.

The CUDA workloads are compiled via *nvcc* with *-O3* optimization level. We did not find significant differences in the results when compiled with different optimization options. On the other hand, the CPU workloads are compiled via *gcc* with different optimization options, i.e., *O0*, *O1*, *O2*, *O3*. SIMTec has a 1.0 correlation with hardware when *O0*, and *O1* optimizations are used. *O1* has a 5% mean absolute error making it the closest to the real SIMT control flow efficiency. In general, when *O3* is used, the analyzer tends to slightly overestimate the SIMT efficiency, in view of the fact that

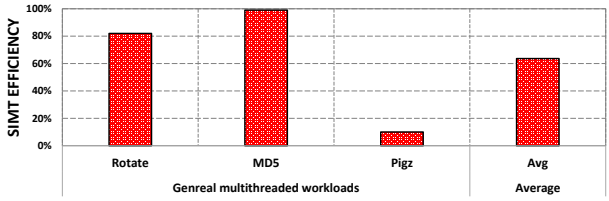


Fig. 4: Warp efficiency of general-purpose multi-threaded workloads (Warp size=32).

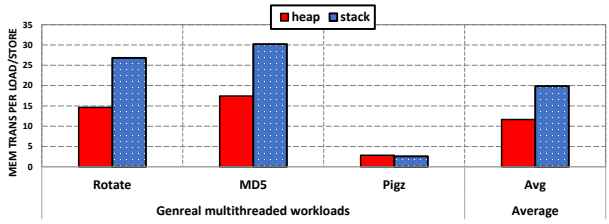


Fig. 5: Memroy transactions per load/store instruction for general-purpose multi-threaded workloads (Warp size=32).

the compiler applies more aggressive optimizations to deliver higher performance.

IV. Case Study

Using our tool, we perform a case study to demonstrate both its potential use cases and the opportunities present in contemporary multithreaded CPU applications that have not been ported to the GPU. In our case study, we examine the feasibility of porting general-purpose multi-threaded Linux utilities to SIMT hardware [3], [4]. In these compute-intensive applications, the program breaks down the workload into multiple chunks and each thread computes the assigned chunk in parallel. We used the original workloads as-is without any alterations.

Figure 4 shows that some of these workloads exhibit promisingly convergent control-flow with up to 99% SIMT efficiency, suggesting that a straight-port of the application code to SIMT hardware would result in significant efficiency gains. However, the data compression benchmark, *pigz*, exhibits limited efficiency as its control flow is inherently data-dependent. In Figure 5, we plot the memory divergence degree. The workloads demonstrate high memory divergence, as each thread has its private stack and the memory manager allocates scattered data chunks in the heap segment, decreasing the data coalescing opportunity at run-time. Data reformatting -like changing data representation from array-of structure (AoS) to structure-of-array (SoA)- can improve the memory efficiency of these workloads [1].

V. Conclusion

Application developers can use SIMTec to quickly evaluate the SIMT-friendliness of any multithreaded CPU application without changing the source code. We believe that SIMTec can uncover new classes of applications that can benefit from GPU acceleration. With access to the source code, we demonstrate

how our per-function analysis quickly identifies problem areas in the code, allowing the developer to understand the scope of code changes necessary to make the application more amenable to the SIMT-based hardware, like GPU.

From an architect and system designers standpoint, we believe SIMTec provides interesting insights into new software classes that can help drive future GPU and general SIMT-accelerator design. It allows architects to ask: What should a SIMT-accelerator look like for general-purpose multithreaded CPU applications? SIMTec can also be easily modified to generate warp-based instruction traces from CPU programs that can be input into contemporary trace-based GPU simulators like Accel-Sim [5], enabling more detailed microarchitectural design studies.

VI. Acknowledgments

This work was supported, in part, by NSF CCF #1943379 (CAREER) and the Applications Driving Architectures (ADA) Research Center, a JUMP Center cosponsored by SRC and DARPA.

References

[1] "NVIDIA CUDA C Programming Guide," https://docs.nvidia.com/cu-da/cuda-c-programming-guide/index.html, NVIDIA Corp., 2020, accessed August 6, 2020.

[2] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2005.

[3] M. Andersch, B. Juurlink, and C. C. Chi, "A benchmark suite for evaluating parallel programming models," in *Proceedings 24th Workshop on Parallel Systems and Algorithms*, 2011. [Online]. Available: http://www.aes.tu-berlin.de/fileadmin/fg196/publication/andersch01.pdf

[4] "A parallel implementation of gzip for modern multi-processor, multi-core machines," https://zlib.net/pigz/.

[5] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*. ACM, 2020.