Paper Review 2

Ayush Kumar (170195)

5 December 2020

I pledge on my honor that I have not given or received any unauthorized assistance.

1 CUDAAdvisor: LLVM-Based Runtime Profiling for Modern GPUs (Shen et al., 2018)

1.1 Summary

The paper first briefly discusses the need for GPU code profilers and debuggers in order to aid programmers and developers in writing efficient, correct and architecture independent code that runs parallelly on GPU cores. It presents the limitations inherent in existing state of the art coarse-grained GPU profilers such as NVProf, TAU and G-HPCToolkit. It states that even the fine-grained CUDA code profiler SASSI developed by NVIDIA is neither portable (in particular, it doesn't work for CUDA runtime 8.0) nor expansible and is quite complex requiring developers an extensive knowledge of PTX assembly code in order to extend/modify it for future architectures. The paper then presents a new fine-grained CUDA code profiler CUDAAdvisor which the authors claim is not only portable across different CUDA runtime versions due to its LLVM backend but also, it instruments both CPU and GPU code to analyze their interaction. To demonstrate portability, the authors test their implementation on two modern NVIDIA GPUs, the Tesla K40c (Kepler architecture) and the Tesla P100 (Pascal architecture) with CUDA runtimes 7.0 and 8.0 respectively.

CUDAAdvisor derives its portability from the open-source and scalable nature of LLVM on top of which the instrumentation engine is built. LLVM's fontend - Clang supports translation of both the host and device source code into the intermediate representation. The instrumentation engine then inserts mandatory instrumentation for function calls and returns, malloc and memcopy (on both device and host) which enables reconstruction of call paths and data flow paths during profiling. It also allows optional instrumentation which can be used to capture all memory read/writes and arithmetic operations. For code-centric profiling, CUDAAdvisor maintains a shadow stack for each CPU/GPU thread and connects them using the CPU call to the device code. For data-centric profiling, it correlates GPU and CPU memory allocation using the instrumentation for *memcpy*(). Finally, the authors evaluate CUDAAdvisor on 10 commonly used benchmark GPU applications and demonstrate how it provides feedback to users based on 3 metrics, **1. GPU Reuse Distance**, **2. Memory Divergence Frequency and Degree** and **3. Branch Divergence**, while also suggesting code optimizations such as **Horizontal Cache Bypassing** to users.

1.2 Key Contributions

Some of the key contributions of the paper are as follows:

- CUDAAdvisor is one of the first GPU profiling tools that uses the LLVM backend. This is advantageous in several ways. First of all, LLVM is open-source and hence has a large community of developers actively working to improve/optimize the current version. Secondly, LLVM's intermediate respresentation is highly scalable and has been used for developing efficient compilers for other languages. Thirdly, it supports compilation of both host and device code which allows complete profiling for a CUDA program and not just the device specific code.

- The evaluation performed by the paper is quite comprehensive and full-proof. In the case studies presented, the authors have walked through the code and illustrated how instrumentation is performed and code optimization suggestions are provided. For example they have performed profiling on the standard benchmarks and shown how programmers could target applications that require branch divergence or cache bypassing optimizations.

1.3 Weaknesses

- To instrument the memory maps created on the device and host and to correlate them, CUDAAdvisor overloads calls to *malloc*(), *cudaMalloc*(), *cudaMemcpy*() and family functions for instrumenting data allocation and transfer. While this seems to do the job in the short run, future GPU architectures or compiler versions could introduce new ways of allocating or transferring data onto the GPU, hence this might not work. Perhaps a more scalable technique for instrumenting the memory map could be inserting code at the assembly level and instrumenting system calls such as **mmap**, **brk** or **sbrk** as this would not be function dependent.

- For code-centric profiling, CUDAAdvisor maintains a shadow stack for each thread. Due to the large number of concurrent threads (this number might increase in future GPU architectures), this might quickly blow up the overheads of data-collection for profiling. This could prove to be a bottleneck in future architectures.

- CUDAAdvisor is built on top of LLVM. Even though this has a positive note as discussed earlier, this requires the availability of the source code for recompilation. For many programs, only the binaries are published and the source code is not always available.

1.4 Future Work

- The overheads of profiling with CUDAAdvisor, even though lesser than most simulators, is still 10x-120x which is significant. Currently, CUDAAdvisor insterts a function call at each instrumentation site. In the future, to reduce call overheads, the authors could explore other techniques such as ringbuffers or nanologgers which could prove to be more efficient.

- CUDAAdvisor could extend the analyses and potential optimizations to include analysis of register pressure and stack memory. The authors have not discussed optimizations related to optimal register and stack memory usage. Such analyses on faster memory access through registers and local memory could help programmers speed up their code even further.

- Currently, CUDAAdvisor allows optional instrumentation for programmers to capture all read/writes, arithmetic operations and control flow operations. In the future, this could be extended to include instrumentation for test coverage for testing a CUDA program.

References

Shen, D., Song, S. L., Li, A., and Liu, X. (2018). Cudaadvisor: Llvm-based runtime profiling for modern gpus. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, CGO 2018, page 214–227, New York, NY, USA. Association for Computing Machinery.