# CS610 - Programming for Performance
# 2020-2021-I: Paper Reading 2

Name: Sarthak Singhal Roll No.: 170635

## CUDAAdvisor

### Paper Summary

General purpose GPU computing is widely used in scientific applications, deep learning, etc, but designing efficient GPU applications are difficult using low level programming models like CUDA and hence performance bottlenecks are very common which are very difficult to analyze manually. Existing performance profilers perform coarse grained analysis which lacks insights into kernel's instructions, loops or functions and also lack support across different GPU architectures and runtime versions. For fine-grained analysis, NVIDIA released a prototype SASSI. But it has several issues like it not portable across CUDA runtimes and architectures, it is close-source, its implementation is too low level for developers and it does not take into consideration the interaction of CPU and GPU. This paper introduces first-fine grained profiler CUDAAdvisor that is portable across CUDA runtimes and architectures. It is built upon LLVM so it is open source. It also instruments CPU code with GPU code. CUDAAdvisor combines code and data centric profiling results to associate performance bottlenecks with their root causes.
CUDAAdvisor consists of 3 components: instrumentation engine, profiler and analyzer. The instrumentation engine which is built on top of LLVM framework, inserts mandatory and optional instrumentation. The mandatory instrumentation includes calls to CPU functions, GPU kernels, memory allocation, memory tranfer between CPU and GPU. CUDAAdvisor also provides an interface to add optional instrumentation for memory, control flow and arithmetic operations. The profiler performs both code and data centric analysis by collecting data during kernel execution and attributing data at end of kernel instance. The code-centric profiling which is done using a shadow stack enables to get call path for each instruction. The data-centric profiling which uses a map from names/call path to allocated memory ranges helps to get the complete data flow. Finally, the analyzer performs analysis on data collected from profiler and it is also very customizable.
Analysis on critical bottlenecks like memory accesses and control flow, was done. Like, analysis of reuse distance was done for chosen benchmarks by appropriate instrumentation. This analysis gives a clear picture of cache locality for the given application and provides some insights for some optimizations. Similarly, analysis of memory and branch divergence was performed to get insights that which applications are in need of potential optimizations. An optimal horizontal cache bypassing was implemented using reuse distance and memory divergence. Also, the code and data centric profiling helps in debugging the GPU applications which can be cumbersome for large applications.

### Key insights/Strengths

- CUDAAdvisor is portable unlike prior tools i.e., it supports profiling across different CUDA versions and architectures which is due to that it uses LLVM which works across different GPU architectures and CUDA versions.

- CUDAAdvisor is built upon LLVM which is open source. So, the developers using CUDAAdvisor can extend it by making contributions unlike previous fine-grained profilers like SASSI.

- The idea of instrumenting both CPU and GPU codes is a good idea as sometimes the code present in CPU like memory copying from CPU to GPU can also be a bottleneck for extracting performance.

- The evaluation is performed very well as the evaluations are not only done on Kepler architecture but also on Pascal architecture with CUDA 8.0 which was most recent at the time when evaluations were performed. This idea of evaluating on different architectures demonstrates the portability of CUDAAdvisor.

- CUDAAdvisor is much faster than simulators like GPGPU-Sim. Generally, overhead is 10X to 120X for CUDAAdvisor while it goes upto 1-10 millions for GPGPU-Sim.

- The analyzer is designed in such a way that it is highly customizable for different analysis. E.g. CUDAAdvisor was used for reuse distance, memory divergence and branch divergence analysis.

- For horizontal cache bypassing an exhaustive search of all options for number of warps was required to identify the best warp number. But, the idea of combining different analyses to derive metric to guide optimization of cache bypassing is very useful for improving the performance and it gives upto 2X speedup.

**Weaknesses**

- CUDAAdvisor has high overhead similar to other profilers due to heavy weight instrumentation to both CPU and GPU codes.

- Currently, it cannot profile register related stats as it is implemented at the bitcode level. But it is also not possible as NVIDIA has not made the assembly layer public.

- The data obtained by instrumentation of GPU code first resides on GPU's global memory. This can compete with the shared resources of GPU that will not give very accurate per-

formance measures.

- CUDAAdvisor requires the source code for its analysis. It also requires to recompile the code to get insights. But it can be possible that only the binary is available or compilation takes huge amount of time which can make it very difficult to use.

**Unsolved problems/potential future work**

- Currently, CUDAAdvisor inserts function calls for instrumentation which leads to huge overhead. A more efficient way to insert the instructions should be devised so that the runtime overhead can be reduced.

- In the evaluation, the authors have only shown the horizontal cache bypassing. But as the vertical bypassing is more fine-grained it can also be tried to implement to check its performance relative to horizontal bypassing.

- Some methods should be developed so that the source code is not recompiled again after the changes from insights provided by the previous iterations. As doing analysis for the programs taking huge amount of time for compiling can be tedious.