INDIAN INSTITUTE OF TECHNOLOGY KANPUR

CS 610 Semester 2020–2021-I

Paper Reading 2

Professor:

Dr. Swarnendu Biswas
Department of Computer Science

Student

Hirak Mondal
Roll-20111022
hirakm20@iitk.ac.in

1 Paper Summary

There is a wide application of General Purpose GPU's to accelerate parallel computation. But producing efficient GPU code is quite a hard work when it comes to relatively complex programming model and given how fast modern day architecture is evolving. In this paper, "CUDAAdvisor: LLVM-Based Runtime Profiling for Modern GPUs"by Du Shen et.al. the authors propose CUDAAdvisor, a profiling framework to guide code optimization in modern NVIDIA GPUs. CUDAAdvisor focuses on fine grained analysis based on profiling result from GPU kernels, like memory-level analysis, control flow analysis and code-/data-centric debugging. Presently, we need to heavily rely on GPU simulators to perform fine grained analysis inside a GPU kernel. But there is high overhead and complexity associated with it. Along with it there may arise compatibility issues and since they do not simulate or emulate every feature of these GPU architectures, optimization guidance may not always be up to the mark. So a demand for fine grained profiler can be seen. To help with the cause, NVIDIA released SASSI, a GPU code instrumentation tool for fine grained analysis. But SASSI being close source has several limitations related to portability, expansibility, complexity and coverage. To overcome all these challenges, Du Shen et.al. has presented CUDAAdvisor. CUDAAdvisor takes the help from LLVM to instrument CUDA code for both its CPU and GPU. CUDAAdvisor consists of 3 components: instrumentation engine, profiler and analyzer. The souce code passes through instrumentation engine and then produces binary, which is then passed to the profiler which then produces the profiling data. The profiling data serves as a input to the analyzer which then generates optimization advice, which the programmers can follow to optimize the source code and begin another round of analysis if necessary. The CUDAAdvisor's instrumentation engine is built on top of LLVM framework because LLVM is a general compiler infrastructure that works on both CPU and GPU codes, it works across different GPU architectures and CUDA versions, adding to it LLVM is robust even for complex HPC programs. The engine provides two kinds of instrumentation, one is Mandatory and the other one is Optional. Now, coming to CUDAAdvisor's profiler, it operates in 2 stages, (1) Data collection during CUDA kernel execution (2) The data attribution at the end of each CUDA kernel instance. With the help of these CUDAAdvisor performs both code and data centric analyses on the go, providing basic infrastructure for code optmization guidance. Other than these, one of the most important component of CUDAAdvisor is CUDAAdvisor's Analyzer. It has both online and offline components. The former performs analysis on the profiling data, while the later merges the analysis results of the kernel instance in the same cell path

providing an aggregated statistical view such as mean, min, max, standard deviation accross all instances. The use cases of CUDAAdvisor is also spread to detailed memory analysis and control flow analysis, both of which serves as a critical performance bottleneck factor for modern GPUs. Moreover, conditional control flow can lead to branch divergence, to which the CUDAAdvisor can also provide meaningful insights, such as how many threads execute this branch and how often a certain branch cause a warp to diverge. With these analysis, CUDAAdvisor is able to further device new metric to guide GPU optimization of cache bypassing which achieves up to 2x speedup along with code-centric and data-centric debugging capability.

2 Key insights/Strengths

- **Portability**

Unlike prior tools, CUDAAdvisor supports GPU profiling across different CUDA versions and architectures, including CUDA 8.0 and Pascal architecture.

- **Code-centric and Data-centric profiling**

CUDAAdvisor combines the code- and data-centric profiling results from both CPU and GPU and associates performance bottlenecks with their root causes.

- **Good Analysis and better optimization guidance**

CUDAAdvisor is able to combine different analyses and derive useful metrics and insights to guide optimizations (e.g., cache bypassing)

- **Expandability**

Unlike SASSI which is close source, CUDAAdvisor's instrumentation engine is based on LLVM, which is open source. Tool developers are able to extend CUDAAdvisor.

• **Coverage**

CUDAAdvisor instruments both CPU and GPU code to analyze its interactions unlike SASSI which only instruments GPU code and overlooks interaction between CPU and GPU.

• **Robust Instrumentation Engine**

CUDAAdvisor's instrumentation engine on top of LLVM framework and LLVM is robust, even for complex HPC programs.

• **Faster and Convinient Runtime profiling of Reuse distance**

CUDAAdvisor enables much faster and convenient run time profiling of reuse distance across generations of NVIDIA GPU architectures.

• **Memory Level analysis and Control flow analysis**

CUDAAdvisor has both memory level analysis and control flow level analysis capability. In present day GPU's we allow developers to add control flow to increase flexibility but conditional control flow can significantly affect warp efficiency and overall performance leading to branch divergence. CUDAAdvisor can provide insights of the kernel's divergence, such as how many times a branch is executed, how many threads execute this branch and how often a certain branch causes a warp to diverge.

• **Improved Analysis**

If a programmer is interested to know which memory accesses suffer from memory divergence, CUDAAdvisor can show not only the source code location, but also the calling context. CUDAAdvisor concatenates the call path from both host and device to show the calling context starting from main function on host all the way to the suspicious site on device, to better guide programmers to understand the program's behavior. Note that CUDAAdvisor is able to capture and display function calls in CUDA kernel as well.

• **Fast Performance**

The run time overhead mostly ranges from 10X to 120X. It is much faster than simulators such as GPGPU-Sim that usually incurs 1-10 millions of slow down to the native execution

• **Enhanced Speed-Up**

Based on analysis. CUDAAdvisor is able to further devise a new metric to guide GPU optimization of cache bypassing which achieves up to 2x speedup.

3 Weakness

• **Relatively High Overhead**

Like SASSI and other runtime profilers it incurs relatively high overhead, (though much lower than simulators) which is caused by heavy weight fine grained instrumentation to CPU and GPU instructions. The high overhead can disturb a program exectution.

• **Requirement of Source Code**

CUDAAdvisor is based on LLVM, which requires the availability and recompilation of source code of monitored program. Without the availability of source code CUDAAdvisor will not be be able to perform its operation.

• **Dependent on LLVM**

The performance analysis of CUDAAdvisor is based on the code generation of LLVM, not the other GPU compilers such as nvcc

• **Cannot Profile Register Related Stats**

CUDAAdvisor is implemented at the bitcode level, It cannot profile register-related stats since NVIDIA has not released its assembly layer to the public.

4 Unsolved problems/Potential future work

• **Decreasing The Overhead**

One of the main concern with CUDAAdvisor is that its high overhead which is resulting from heavy weight fine grained instrumentation to CPU and GPU instructions. This is a point of concern as this overhead may disturb the program execution and also consume system resources. One of the approaches that I think will help is to take the help of Machine Learning. Say we have a data-set of GPU codes as input and their detailed analysis as output. We can learn from them which code structure yields what kind of drawbacks and group them in clusters based on their drawbacks in specific regions of code. Now what we can do is that, instead of doing fine grained analysis on the entire GPU code we can first run the ML algorithm on that GPU code and see in which cluster the code belongs to. The cluster will give us the information in which part of the code, it is more likely to have drawbacks and after that we can use our CUDAAdvisor to only perform fine grained instrumentation on that part of the code. One may argue that it is not possible to get large datasets of GPU codes and perform CUDAAnalysis on them to form the dataset, but emerging Machine Learning fields like few-shot learning(It learns to build classifiers from a few labeled examples of each class), one shot learning (When only one example with supervision is available) and zero shot learning (When NO example with supervision is available) can take care of that.

With this approach, since we will already have an idea which part of the code is likely to contain the drawbacks after running the ML algorithm on it, we can heavily focus on that region only and can perform extensive instrumentation using CUDAAdvisor to obtain the detailed analysis at full extent rather than worrying about the remaining parts of the code. This will help to achieve a great analysis and on the other hand it will help to minimize the overhead which was our main concern at the outset.

• **Profiling Register-related stats**

If after sometimes, NVIDIA makes its assembly layer public then it will be a high time to profile register related stats as it will give us more insights on register allocation and de-allocation and thus we will have a more deep understanding on what is going at the register level and take proper measures to boost the performance.