Paper Reading 2

Aniket Sanghi (170110)

8th December 2020

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

1 Paper 1 - Critique

Summary

The paper presents a fine-grained profiling framework - **CUDAAdvisor** that guides code optimisations in modern NVIDIA GPUs with various CUDA versions and architectures including CUDA 8.0 and Pascal architecture. It has been built over an open source compiler infrastructure - **LLVM** making it wide ranged and easily accessible to all.

CUDAAdvisor's workflow can be divided into 3 major components - instrumentation engine, profiler and analyser. The instrumentation engine inserts functions at instrumentation site to perform mandatory and some optional instrumentation to the code based on the analysis requirement. The profiler performs code-centric and data-centric profiling based on the data obtained from instrumentation engine. The analyser helps in the analysis of reuse distance, memory divergence, branch divergence etc and give suggestions on how to improve code performance.

CUDAAdvisor has less over-head as compared to the then state-of-the-art fine-grained analysers which were mostly simulators/emulators. It was tested on benchmarks and obtained reliable measures such as GPU reuse distance, memory divergence frequency and degree which offered important insights about the bottlenecks.

Key Insights & Strengths

- **Run-time** It is a very major lead over the other fine-grained analyser - simulators and emulators, which use to take a lot of time to give performance statistics and even weren't general enough to be applicable to all CUDA evolutions easily.
- **Open Sourced** One of the major plus points about it is that it is Open-sourced, making it accessible and also help other contribute to its evolution at a faster pace.
- **Coverage** It not only help analyse the performance for the GPU but also CPU-GPU combined. This offers potential for important insights that analysis of only GPU couldn't have offered.
- **Well-structured** The paper is very well structured and easy to understand. Offers all the major highlights needed to understand the approach.

Weaknesses

- **High Overhead** It has very high overhead majorly due to the function calling in the instrumentation engine and the use of atomic operations at various places of the code.
- **Poor Memory Management** It stores data in the global memory of the GPU and hence compete with the code data in the GPU. This can impact the performance severely particularly in applications with high memory usage.

- **Restricted** Since it uses the LLVM framework the performance analysis it gives work best with them. Though the performance reports still help with other compilers but the performance improvement may not be the optimal one for them.
- **Missing future insights** The paper doesn't talk about how the paper can be improved further. It talks about problems but no plausible cures.

Unsolved Problems & Potential future work

- **Time Overhead Improvement** Using alternate choices than the function calling in the instrumentation stage could help relieve a lot of time overhead that we get (since we know functions calls are really expensive)
- **Memory Management** Instead of keeping all of the buffer in the GPU memory we can optimally send the buffer data in chunks to the CPU to effectively manage GPU memory and save it from competing with the programs memory.
- **Suggestive approach** The paper doesn't mentions if CUDAAdvisor also suggests plausible changes to the code. If not, we can add feature to also explicitly guide programmer with some of the plausible changes in easy scenarios that can help optimise code.