ECE 584 Class Project Proposal

Arjun Vedantham

1. Introduction

GPUs have become a critical infrastructure component of any high performance computing cluster. Much of the code targeting GPUs leverages CUDA, ROCm, and other GPU-specific programming frameworks that have unique behaviors that may lead to common bugs or performance inefficiencies. For instance, programmers can easily miscalculate the memory address required by each thread to access its working set. Given these common classes of bugs, we would like to examine using verification tools to validate the correctness of GPU kernel code. Specifically, we would like to target kernel implementations generated by large language models, since these have increased the expressivity of synthesis tools to beyond what was previously capable with constraint solver guided program synthesis. (Solar-Lezama)

2. Problem statement

Given generated GPU kernels, we want to verify both the correctness and performance aspects of the generated code. On the correctness end, we would like to verify safety constraints such as ”there are no out of bounds memory accesses”. Additionally, we plan to consider liveness properties could include constraints that we would like our generated kernel implementation to meet, like maximizing the number of accesses made to shared memory, while minimizing more expensive global memory accesses.

3. Related work

There are numerous papers on LLM-guided program synthesis, including with respect to GPU kernels. Traditional GPU verification tools include GPUVerify (Betts et al.), which checks for data races and other types of unsafe memory behavior. More recent work has examined the role of LLMs in kernel code generation - for instance, a team at SJTU compared LLM-generated code with human written versions to verify their correctness, using a heuristic to ensure that LLM generated kernel implementations did not produce results that deviated far from the ground-truth human written examples (Chen et al.). In addition, it incorporated some runtime performance feedback to guide the LLM's synthesis process towards more efficient kernel implementations. KernelBench (Ouyang et al.) is another framework for automated performance optimization of GPU kernels, however, it primarily relies on wall-clock time as its main feedback, and does not incorporate any ground-truth kernel implementations - in fact, the authors specifically mention leveraging formal verification to provide stronger correctness guarantees. Finally, a project named SHARD (Zhao and Hua) focused on formalizing security guarantees for GPU kernels, using Dafny and Verus to actually handle the correctness checks.

4. Methodology

We would like to start by taking LLM-generated GPU kernels for simple tasks (e.g. tiled matrix multiplication), and adding formal safety guarantees in a verification language like Dafny, similar to the approach described in the SHARD paper. Dafny has plenty of features for stating loop invariants, pre and post-conditions, etc. Due to the extensive scope of potential bugs in the CUDA ecosystem, we would like to initially focus on verifying the absence of memory indexing bugs (the ”memory model” category as described in table 1 of the SHARD paper). Modeling our approach after the SHARD paper, we would formulate this as a Hoare triple, which we would then discharge in Dafny. Once these safety guarantees are implemented, we would then explore verifying specific performance-centric behavior. For instance, we would like to use inductive invariants to formally prove a bound on the number of global memory accesses for a specific kernel.

Since capturing the behavior of the GPU kernel within Dafny may be difficult, an alternative approach could be to simply add constraints about how each thread calculates its memory address, and pass this to an SMT solver (like Z3) to verify that the block or thread-specific memory bounds are not violated. However, since Z3 is less expressive than Dafny, it may be more difficult to leverage the Z3 abstractions to prove runtime characteristics, like bounding the number of global memory accesses.

. Correspondence to: Arjun Vedantham <arjunsv2@illinois.edu>.

ECE 584 Class Project, Fall 2025. Instructor: Prof. Sayan Mitra

5. Results

In the end, what we hope to achieve is:

- A framework for extracting kernel launch parameters and moving them into Dafny (2025-10-16)
- Leveraging this framework to state and prove safety invariants about how the kernel accesses memory - e.g. verifying that there are no out-of-bounds memory accesses based on the launch parameters (2025-11-01)
- Stating and proving general performance or behavior constraints - e.g. bounding the number of global memory accesses. This expressivity improvement will be the main focus of our project.

Although fine-tuning the LLM output by providing feedback is not a primary goal of this project, we would also be interested in seeing if any violations of safety or liveness constraints could be added as synthesis constraints in the form of a new prompt.

References

Adam Betts, Nathan Chong, Alastair Donaldson, Shaz Qadeer, and Paul Thomson. GPUVerify: a verifier for GPU kernels. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*, pages 113–132. ACM. ISBN 978-1-4503-1561-6. doi: 10.1145/2384616.2384625. URL https://dl.acm.org/doi/10.1145/2384616.2384625.

Wentao Chen, Jiace Zhu, Qi Fan, Yehan Ma, and An Zou. CUDA-LLM: LLMs can write efficient CUDA kernels. URL http://arxiv.org/abs/2506.09092.

Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. KernelBench: Can LLMs write efficient GPU kernels? URL http://arxiv.org/abs/2502.10517.

Armando Solar-Lezama. Introduction to program synthesis. URL https://people.csail.mit.edu/asolar/SynthesisCourse/Lecture1.htm.

Jiacheng Zhao and Baojian Hua. SHARD: Securing GPU kernels with lightweight formal methods. URL https://csslab-ustc.github.io/publications/2025/gpu-security-full.pdf.