CS6101-09

Automatic Accelerator Code Generation via LLM Agentic Workflow

Takanori Aoki

Abstract

Code optimization for a specific accelerator requires deep expertise in parallel programming, memory hierarchies, and low-level performance tuning techniques specific to hardware architecture. This project is to employ LLM for automatic accelerator code generation and optimization by providing algorithm specification and optimization strategies from various sources, and then leverage LLMs to generate CUDA C++ implementations, targeting for a popular algorithm for Deep Learning.

Approach

VibeCodeHPC

The project leverages VibeCodeHPC [1] to build an auto code generation system. VibeCodeHPC is an automatic tuning system for High Performance Computing (HPC) programs based on agentic LLM, generating and optimizing HPC code through agents' interaction and iterative prompt refinement.

Agent roles

The auto code optimization system consists of following LLM-powered agents:

- Project Manager (PM)
- Programmer (PG)
- System Engineer (SE)

Code optimization flow

1. Provide following code and data as context
   - Unoptimized code written in C++
   - Specification of Accelerator to run the code
   - Login info to a server to run the benchmark
2. Trigger agentic flow
   - PM conducts requirements definition, resource allocation, budget management, and write the final report
   - PG runs experiment including server login, code optimization, compilation, and benchmarking
   - SE monitors the latest achievement, analyze result, and then determine next step

Experimentation

Run benchmark on a NVIDIA A100 SXM4 40GB GPU server to compare performance of code generated by the agentic flow against the one using existing library (cuBLAS). To evaluate LLM's hardware specific code tuning capability, LLM agent is not allowed to use the existing library to implement the code.

Target application

GEMM (Mixed Precision): It's widely used for computing the large matrix multiplications especially in layers like fully connected or convolutional layers.

Attention with KV-cache: In transformer models like GPT, attention computes token relationships using Q, K, and V. The KV cache stores K and V to avoid recalculating them for new queries.

Result and Future Work

Performance gaps were observed between cuBLAS and LLM's custom kernels—12× in GEMM and 4.3× in KV-cache attention—arising from differences in kernel optimization depth and workload characteristics. Because the KV-cache benchmark involves non-GEMM operations (e.g., gather and softmax) that show similar performance across implementations, the overall performance gap appears smaller. Introducing a Performance Profiling Agent using tools such as NVIDIA Nsight Compute could improve performance analysis and code optimization feedback loop.

Application	With cuBLAS	Without cuBLAS (LLM Agentic Workflow)
GEMM (MP)	40.268 ms 218.44 TFLOPs	482.917 ms 18.21 TFLOPs
Attention KV-Cache	0.900 ms 19.08 TFLOPs	3.898 ms 4.41 TFLOPs

References

[1] [SKD25] VibeCodeHPC: An Agent-Based Iterative Prompting Auto-Tuner for HPC Code Generation Using LLMs, arXiv:2510.00031, 2025.