LLM for Performance

Working Group Outbrief

Scalable Tools Workshop 2025

Performance Optimization Research Overview

- Creating evolution benchmarks to understand architectural optimization and GPU performance gaps in LLMs
- Collected dataset of inefficient vs optimized code from GitHub repositories
- Two categories analyzed:
  - Sequential optimizations (e.g., loop invariant code motion)
  - Parallelization optimizations (using OpenMP)
- Testing methodology includes:
  - Correctness tests
  - Runtime optimization measurements
  - Compiler output validation (-O3 flags)

Benchmarks

- Probir has begun collecting benchmarks
  - OpenMP parallelism
  - Loop-invariant code motion
- Compare
  - Original
  - Human optimized
  - Best LLM-generated code
- Use prompts to direct code improvements
  - Two approaches
    - Prompts just tell the LLM to use its knowledge to improve the program
      - Multi-step process (guess and check) to see what helps
    - Detailed prompts that provide a lot of optimization suggestions. Use the LLM to perform the complex changes according to the advice
- Evaluate LLM code generation
  - Examples came from leetcode
    - Mostly sequential code
  - Need to understand why LLM is performing poorly in certain cases

Training Strategies

- Examples from AMD programming contest
- Neutrino paper shows tuning of flash attention model by improving its memory access patterns
- Idea
  - Train a model to use a profiler so that it can collect data to understand a bottleneck
  - Use PC sampling to identify where bottlenecks are to provide guidance to LLM about what needs to be improved
- Need to assess what problems are the most important at particular scales
- Can prune performance information (e.g. HPCToolkit data) to what is most relevant to simplify training
- How to make experiments less costly
  - Try to get scaling predictions and reduce the number and scale of experiments needed

LLM Code Optimization Experiments

- Used multi-step prompting approach for code optimization
- Process includes:
  - Initial code analysis
  - Performance reasoning
  - Code generation
  - Testing across multiple iterations
- Results show LLMs can sometimes outperform human optimizations
- Key challenge: LLMs need context-specific information to make optimal decisions
- Example success: Generated better OpenMP code with specific scheduling guidance

Performance Analysis Tools Integration

- Discussion of Neutrino paper (March 2025)
- Analysis of Flash Attention versions showing memory access patterns:
  - Flash Attention v1
  - Flash Attention v2
  - Naive implementation
- Integration of profilers with LLMs to:
  - Identify bottlenecks
  - Analyze memory access patterns
  - Guide optimization decisions

Project Updates

- Development of ChatHPCtoolkit
- Pedro Valero Lara’s work on ChatKokkos
  - Translating Fortran kernels to Kokkos C++
  - Using codalama

Key Challenges Identified

- Scale-dependent performance issues
  - Small vs large-scale optimization differences
  - MPI communication becoming critical at 1000+ GPUs
- Measurement challenges:
  - Performance measurement noise
  - Trade-off between granularity and perturbation
  - Need for better metrics beyond wall clock time
- Compiler limitations:
  - NVIDIA compiler weaknesses in code optimization
  - Manual intervention often required for optimal performance

Action Plan

- Scraping github for commits from leetcode that tune performance is a great way for identifying code for training
- Using an LLM to synthesize unoptimized and optimized code examples based on a commit
- Use examples from AMD contest to create training examples: code with various features paired with performance information about the code
- What kinds of performance features do we need to consider
  - Memory access patterns
  - Memory footprint
  - Wall clock time
- Curate database of performance problem patterns

Resources

- **Marco**: A multi-agent system for optimizing HPC code generation using large language models.
  - Rahman, A., Cvetkovic, V., Reece, K., Walters, A., Hassan, Y., Tummeti, A., Torres, B., Cooney, D., Ellis, M., & Nikolopoulos, D. S. (2025). *arXiv preprint arXiv:2505.03906.*
  - https://arxiv.org/pdf/2505.03906
- **Neutrino**: Fine-grained GPU Kernel Profiling via Programmable Probing.
  - Huang, S., & Wu, C. (2025). *OSDI 2025.*
  - https://open-neutrino.github.io
- Hugging Face Kernelbot Data
- Surprisingly Fast AI-Generated Kernels We Didn’t Mean to Publish (Yet)
- hpcresear.ch

Participants

Jonathan Madsen

Terry Jones

Probir Roy

John Mellor-Crummey

Yumeng Liu

Yuning Xia

Gustavo Morais

Hsuan-Heng Wu

Edgar Leon

Please add your name