Can Large Language Models Predict Parallel Code Performance?

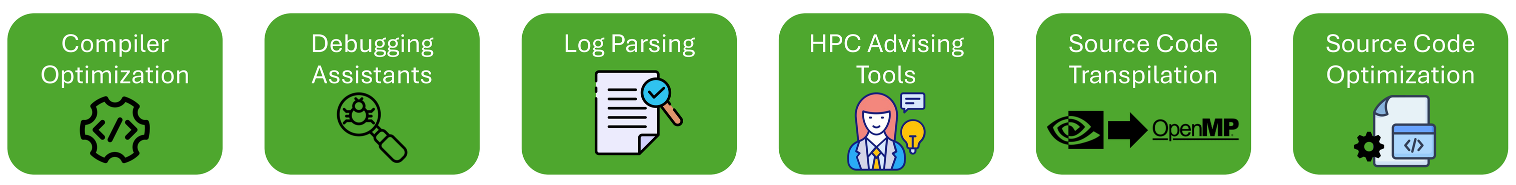
Gregory Bolet[1], Giorgis Georgakoudis[2], Harshitha Menon[2], Konstantinos Parasyris[2], Niranjan Hasabnis[3], Hayden Estes[1], Kirk Cameron[1], Gal Oren[4]

[1]Virginia Tech (VT), [2]Lawrence Livermore National Laboratory (LLNL), [3]Code Metal AI, [4]Technion & Stanford University

Motivation

Trend 1: Large Language Models (LLMs) are becoming ubiquitous in Software Development.

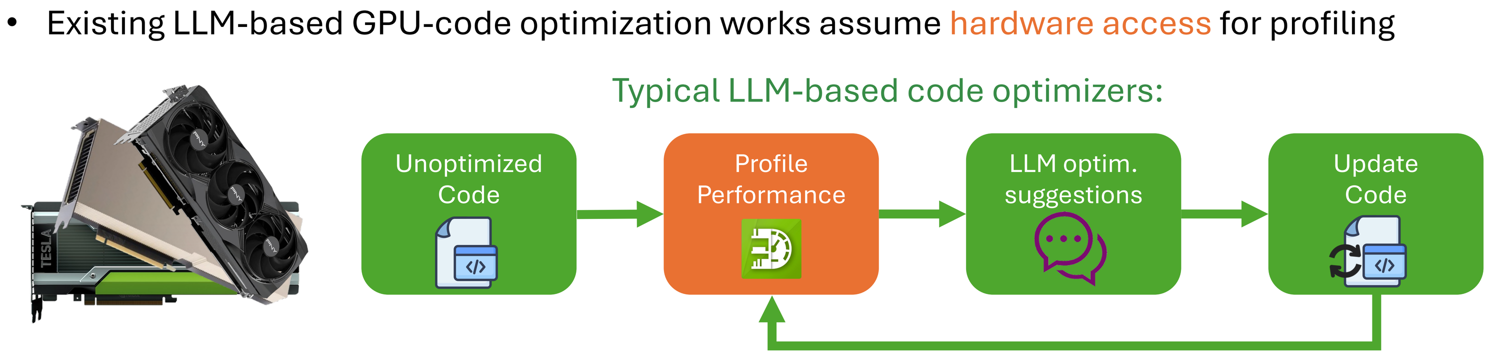
Trend 2: Not many Performance Analysis sub-fields using LLMs for GPU execution profiling/analysis



Trend 3:

- New GPU hardware is becoming increasingly inaccessible (due to datacenter demand)
- Existing LLM-based GPU-code optimization works assume hardware access for profiling

Typical LLM-based code optimizers:



Trend 1
Trend 2
Trend 3

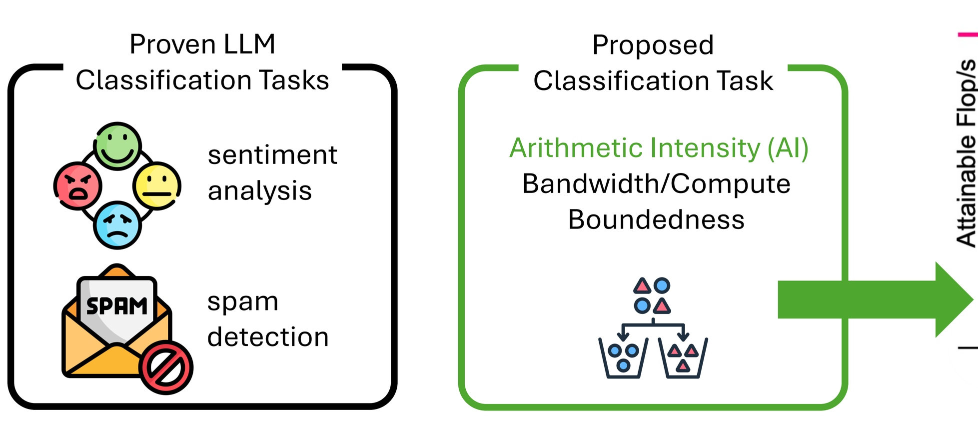
Idea: Can LLMs predict GPU code performance *without* the need for profiling?

Possible Metrics:

- Execution Time
- FLOP/s
- Cache Misses
- Bytes Read/Written
- FLOP/Byte
- Instructions/Cycle

Problem: LLMs are traditionally *BAD* at regression tasks

Solution: Focus on a simple *classification* task instead!



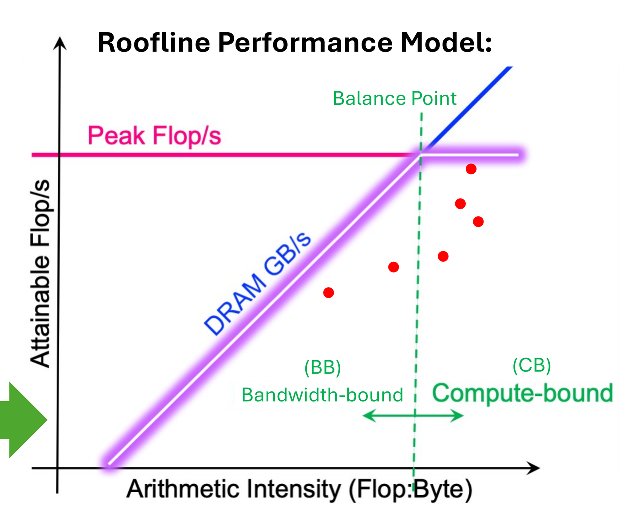


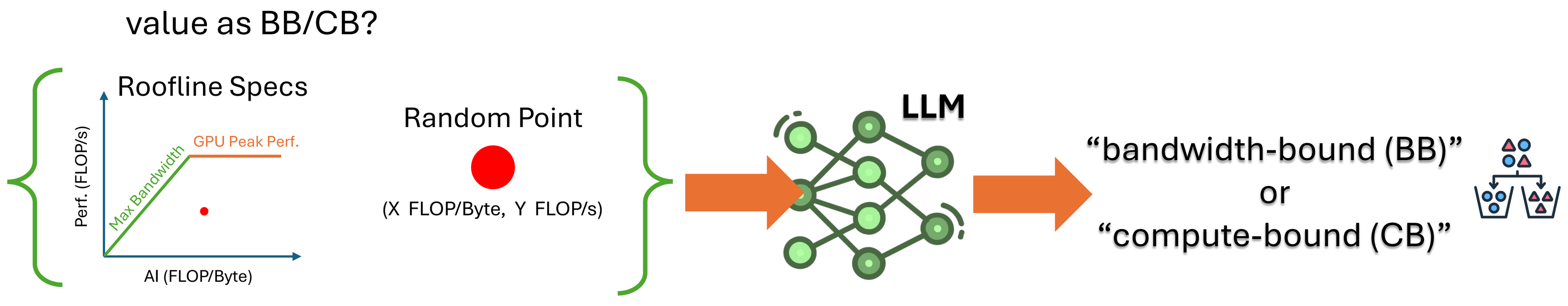
Figure credit: SAM WILLIAMS Roofline Lecture

Research Questions

How well can LLMs classify the Arithmetic Intensity (AI) of GPU codes?

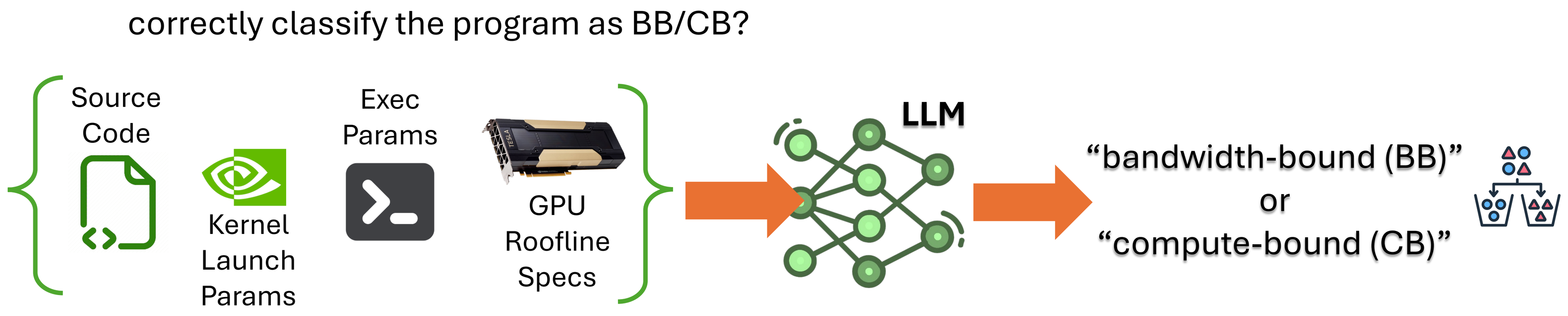
RQ1 (Baseline Roofline Classification)

- Given the GPU Roofline specs and an explicit AI value, can an LLM correctly classify the value as BB/CB?



RQ2 (Source Code Classification)

- Given the source code, necessary execution specs, and minimal instructions, can an LLM correctly classify the program as BB/CB?



Dataset Design Decisions

https://github.com/zjin-lcf/HeCBench

Built + Profiled:
170 CUDA + 170 OpenMP HeCBench Codes

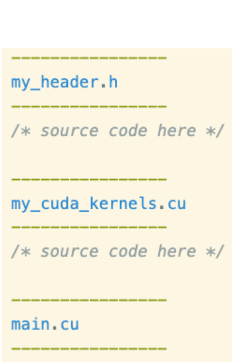
zjin-lcf / HeCBench

NVIDIA RTX 3080 GPU

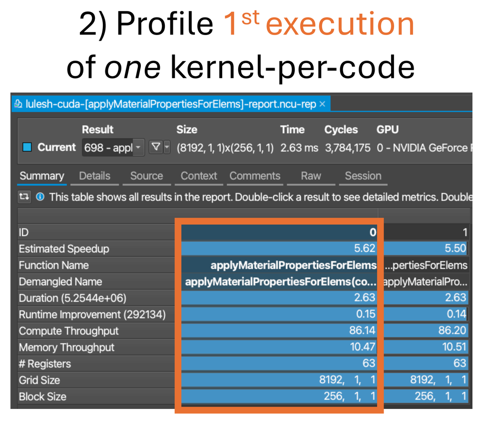
Perf. Metrics

- SPFLOP (FP32),
- DPFLOP (FP64),
- INTOP,
- AI for each kernel

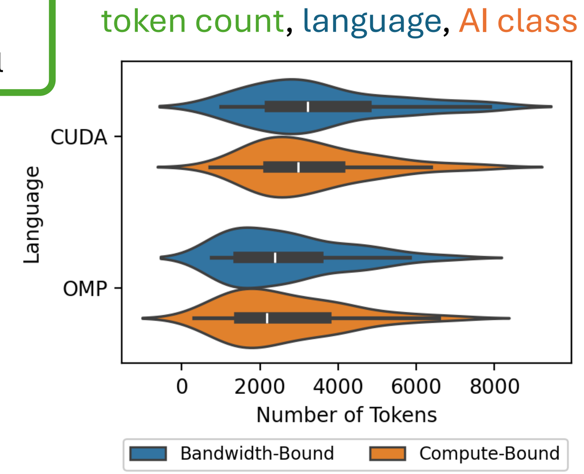
1) Concatenate each kernels' source files for prompting



2) Profile 1st execution of *one* kernel-per-code



3) Balance dataset w.r.t: token count, language, AI class



RQ1: Roofline Understanding

Experimental Setup

- 120 CB + 120 BB prompts
- Random Rooflines + AI values
- 2, 4, and 8-shot examples
- Fixed temp = 0.1, top_p = 0.2
- Evaluation metric: accuracy

RQ1 Prompting Template (w/ CoT)

Few-shot Examples:

CoT example 1 (shown below):

Question: **Question:** Given a GPU having a global memory with a max bandwidth of 45.9 GB/s and a peak performance of 52.22 GFLOP/s, if a program executed with an Arithmetic Intensity of 0.6 FLOP/Byte and a performance of 19.4 GFLOP/s, does the roofline model consider the program as compute-bound or bandwidth-bound?

Optional CoT: **Thought:** The max bandwidth is 45.9 GB/s, and peak performance is 52.22 GFLOP/s. The balance point is at 52.22 / 45.9= 1.14 FLOP/Byte. The program's Arithmetic Intensity is 0.6 FLOP/Byte. Because 0.6 < 1.14, it is before the balance point, putting the program in the bandwidth-bound region. The roofline model would consider the program as bandwidth-bound.

Response: **Answer:** Bandwidth

CoT examples 2-8 [redacted]

Target Query: **Question:** Given a GPU having a global memory with a max bandwidth of 99.9 GB/s and a peak performance of 73.45 GFLOP/s, if a program executed with an Arithmetic Intensity of 1.55 FLOP/Byte and a performance of 32.8 GFLOP/s, does the roofline model consider the program as compute-bound or bandwidth-bound?

Results:

Model Name	Reasoning	RQ1 Acc.	RQ1 CoT Acc.
o3-mini-high	✓	**100**	**100**
o1	✓	–	–
o3-mini	✓	**100**	**100**
gpt-4.5-preview		–	–
o1-mini-2024-09-12	✓	**100**	**100**
gemini-2.0-flash-001		91.25	92.50
gpt-4o-2024-11-20		91.25	96.25
gpt-4o-mini		90.00	**100**
gpt-4o-mini-2024-07-18		90.00	**100**

Findings

- All models have a reasonably-good understanding of AI
- Reasoning models have good prediction accuracy w/ and w/out CoT
- 2 examples was often sufficient

RQ2: Source Code Classification

Experimental Setup

- 170 CB + 170 BB CUDA/OMP codes
- 2-shot examples
- Fixed temp = 0.1, top_p = 0.2
- Evaluation metric: accuracy

Results:

Model Name	Reasoning	Input/Output Cost (1M tokens)	RQ2 Acc.
o3-mini-high	✓	$1.1 / $4.4	**64.12**
o1	✓	$15 / $60	64.12
o3-mini	✓	$1.1 / $4.4	62.06
gpt-4.5-preview		$75 / $150	59.71
o1-mini-2024-09-12	✓	$1.1 / $4.4	59.64
gemini-2.0-flash-001		$0.1 / $0.4	55.59
gpt-4o-2024-11-20		$2.5 / $10	52.06
gpt-4o-mini		$0.15 / $0.6	50.59
gpt-4o-mini-2024-07-18		$0.15 / $0.6	50.29

RQ2 Prompting Template (see paper for full prompt)

[omitted context-setting beginning of prompt]

Pseudo-code examples:

Provide **only one word** as your response, chosen from the set: ['Compute', 'Bandwidth'].

Examples:

Example 1:

```
Kernel Source Code (simplified):
for i = 0 to 1000000 {
 a[i] = a[i] + b[i];
}
```

Response: **Compute**

Example 2:

```
Kernel Source Code (simplified):
for i = 0 to 10 {
 load_data(large_array);
 process_data(large_array);
 store_data(large_array);
}
```

Response: **Bandwidth**

Now, analyze the following source codes for the requested kernel of the specified hardware.

GPU specs:

Classify the [language] kernel called [kernel name] as **Bandwidth** or **Compute** bound. The system it will execute on is a [GPU model] with:

- peak single-precision performance of [X] GFLOP/s
- peak double-precision performance of [X] GFLOP/s
- peak integer performance of [X] GINTOP/s
- max bandwidth of [X] GB/s

Exec specs:

The block and grid sizes of the invoked kernel are (X,Y,Z) and (X,Y,Z), respectively. The executable running this kernel is launched with the following command-line arguments: [arg1 arg2 arg3].
Below is the source code of the requested [language] kernel:
[concatenated source code files]

Findings

- Non-reasoning models are akin to a coinflip
- Similar CUDA/OMP prediction accuracy
- Room for improvement with o3-mini-high achieving highest accuracy of 64%

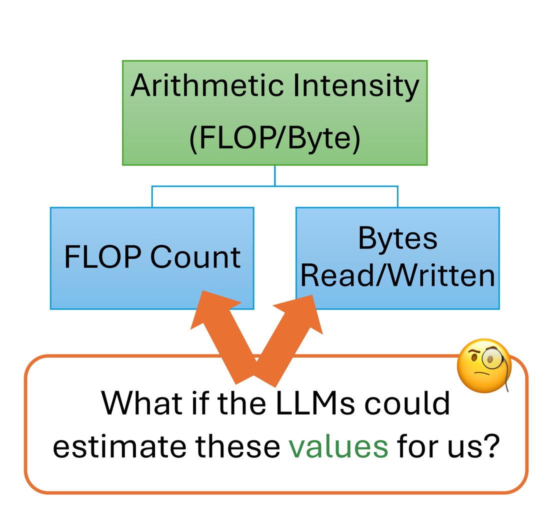
Conclusions

- SoTA LLMs *do* understand the Roofline Model for GPU performance analysis
- SoTA LLMs *can* predict parallel code performance – when limited to classifying Arithmetic Intensity (AI) of CUDA/OpenMP programs
- Reasoning-equipped LLMs (e.g.: o3-mini-high) offer significantly better classification accuracy when compared to non-reasoning LLMs

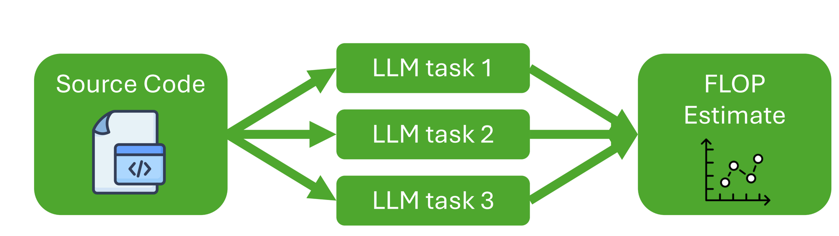
Next Steps

Major Shortcomings:

- Binary classification
- Single-prompting approach



We currently have *some* success in applying Question Decomposition to estimate FLOPs



Target Name	Empirical FLOP Count	LLM-Estimated FLOP Count	% Diff
resize-cuda	16779307	16777216	0.012 %
zerocopy-cuda	1050389	1048576	0.17 %
iso2dfd-cuda	54419825	53196468	2.24 %
nlll-cuda	6006	6273	4.44 %
backprop-cuda	3080240	3080192	0.001 %

Acknowledgements

This work was funded in part by NSF awards:
#1838271 VarSys: Managing Variability in High-Performance Computing Systems
#1939076 iLORE: Computer Systems Performance Integrated Lineage Repository

Lawrence Livermore National Laboratory | COMPUTING | VIRGINIA TECH

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-POST-

NNSA National Nuclear Security Administration

CUT CUT CUT CUT CUT CUT CUT CUT CUT CUT