Nolan Anderson | Week 4 Assignment 2 | CPE 613 | February 4 2023

1. What is arithmetic intensity?

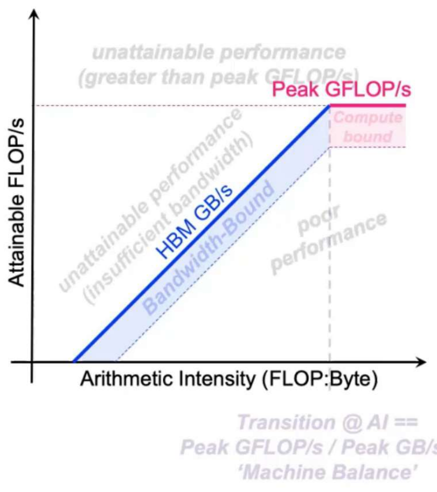
Arithmetic intensity is a measure of data locality. It is defined by the ratio of total flops performed to total bytes moved. It is the relationship between the amount of computation is being performed to the amount of time spent moving data around in an architecture. The formula is as follows: AI(Arithmetic Intensity) = FLOPs / Bytes (moved to/from DRAM)

2. What usually takes longer--data movement or compute?

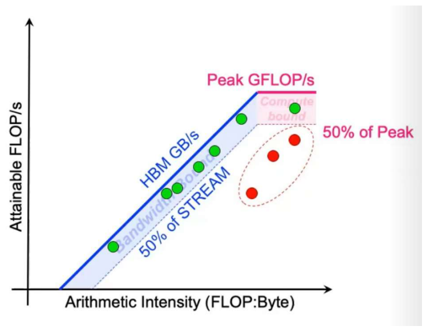
Data movement certainly takes longer. Once the device / cpu has the data, it's not going to take very long to perform the instruction. If you look at a general 5 stage ISA, you are only going to spend 1 stage in the ALU. The rest is going to memory management and instruction fetching. The time to manage data is going be much greater than computing the data.

3. Explain, with pictures and examples, why the roofline model is helpful.

The roofline model is the "roof" of performance, and shows how different performance metrics stack up the the peak performance of a device. Above the roofline is unattainable performance since it is above the maximum performance of the device. Below the roofline, and above the HBM GB/s is unattainable performance due to insufficient bandwidth for your data. Directly below the line is the bandwidth bound region where you have sufficient bandwidth but not enough data locality to keep up. Directly under the roofline, you have the compute bound region where you can get close to the peak performance of the machine. Elsewhere is poor performance. The graph to the right shows the roofline model on the relationship between Arithmetic Intensity and Attainable FLOP/s. This roofline model is very helpful in understanding how well your software is performing compared to the peak of the system itself. If your program or compiler is performing well below the compute-bound / bandwidth-bound regions, you may want to take a look at how to improve your architecture.

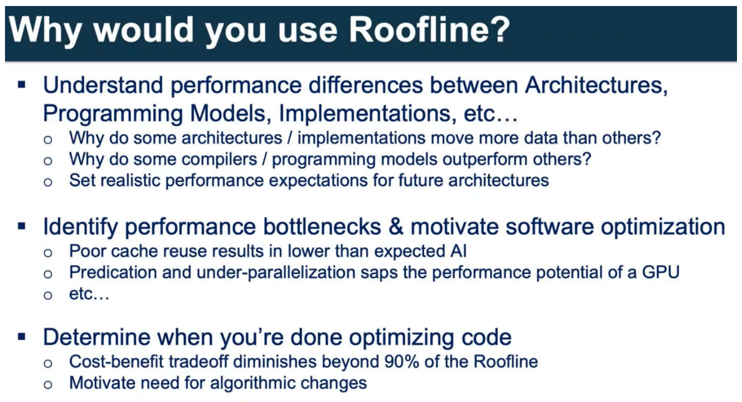


In this example to the right, the kernels in red show poor performance and would benefit the most from



optimization, while the kernels in green would not see much benefit. Focusing on the kernels in red will give us the most value when performing optimization.

Overall, the roofline model is helpful to figure out exactly where you are leaving performance on the table. Some kernels may be performing well, but aren't making great use of the hardware available. Some may be performing well, but are still not taking good use of the hardware. The graph tells you a lot about how well your program works against the peak performance of the hardware. The following screenshot specifically identifies reasons to use the roofline model:



Why would you use Roofline?

- Understand performance differences between Architectures, Programming Models, Implementations, etc…
  - Why do some architectures / implementations move more data than others?
  - Why do some compilers / programming models outperform others?
  - Set realistic performance expectations for future architectures
- Identify performance bottlenecks & motivate software optimization
  - Poor cache reuse results in lower than expected AI
  - Predication and under-parallelization saps the performance potential of a GPU
  - etc…
- Determine when you're done optimizing code
  - Cost-benefit tradeoff diminishes beyond 90% of the Roofline
  - Motivate need for algorithmic changes

4. What part of this video was most interesting to you?

The memory chart on the Nsight program was really cool. Showing the utilization and how the memory is mapped given a program is very helpful to figure out where you need to optimize your program better. He even shows where the program had bottlenecks in memory, even giving recommendations for your code. It takes you right to the place in the code that could be improved.

5. What part of this video would you like to hear more about?

I would like to spend some time with Nsignt, and do some analysis on given code. I think that would be super helpful to see how poor code can impact performance. The example they gave on the SOL program was pretty interesting.

6. What part of this video was most confusing, if any?

The optimizations near the latter half of the video were a little confusing. That mostly comes from not understanding the implementation / problem, though I think this was just an example of how different optimizations impact performance metrics. I think spending more time with the actual optimizations they made would be helpful. In general, seeing how specific optimizations will change different performance metrics was a little confusing.