Part 2: Write Up

A graph with red and blue lines

Description automatically generated

A graph with numbers and lines

Description automatically generated

**3. Peak performances, bandwidths, and ridgeline:**

The above Roofline plots illustrate the relationship between operational intensity and the performance of the computing architecture – GFLOPS/s / FLOPS/byte. Value labels of each red line correspond to the maximum memory bandwidth available at cache levels L1, L2, and DRAM – this being the rate data can be read or written to memory from the CPU. The different cache levels have a theoretical maximum memory bound bandwidth of 56.4, 37.4 and 20.5 GB/s for L1, L2 and DRAM, respectively. The peak theoretical performance for FP64 – the double-precision operations is 12.3 GLOPS/sec. Intersections of L1, L2, and DRAM and the horizontal peak performance line are the ridge-line points that highlight the moment when arithmetic/operational intensity is constrained by the machine’s computing ability, rather than its memory bandwidth. Hence, by attempting to keep more of a task’s workload in the higher-level caches, like L1, we can approach the peak performance and, thus, efficiency for that machine.

Hence,

These results indicate that workloads relying on lower parts of the memory hierarchy for data access must have a substantially larger arithmetic intensity before overcoming memory bandwidth limitations and approach peak computing performance. (Note – arithmetic intensity levels on x-axis prior to reaching the peak perf line are memory bound, rather than compute bound).

**4. Consider the four FP kernels in "Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures" (see their Table 2). Assuming the high end of operational (i.e., "arithmetic") intensity, how would these kernels perform on the platforms you are testing? What optimization strategy would you recommend to increase performance of these kernels?**

**5. Address the same questions in (4) for the four kernels given in the Warm-up above.**

(be sure to address the fact that we’re assuming ‘’the high end of operational intensity’)

Among the four kernels in the warm-up, each have different characteristics regarding temporal and spatial locality, so the optimization methods might change for each. For example, the first kernel:

, we see that this likely has low temporal locality due to the two indices j and i. Thus, j is the outer loop and it must complete all of the computations for one j before moving to the ith index. To improve computing efficiency, SIMD, stencil, or blocking – referenced in the paper, are methods that could improve a kernel like this one.

**6. Compare your results for the roofline model to what you obtained for the matrix-matrix multiplication operation from Part 1. How are the rooflines of memory bandwidth related to the features in the algorithmic performance as a function of matrix size?**

By comparing the results of the roofline model to those of the matrix-matrix multiplication in part 1, there are several noteworthy observations and explanations. For example, we see that the L1 cache ridge-point is and over the range of matrix size N, matrix-matrix multiplication peak around , which occurs when N is quite small - likely between 10 and 20. When N is small, the entire matrices fit into the CPU’s cache (perhaps caches L1 and L2) and efficiency is at its peak because there are no memory access delays. As N grows, there are clear points in the plot where the matrices no longer fit into the upper memory caches and cache misses begin to occur, thus showing decreases in GFLOPs/sec followed by plateaus. When N gets very large and the upper levels caches become fully utilized, then requiring DRAM access, resulting in slower computational efficiency. Eventually, we reach a point, around where N=2000, that DRAM is also being heavily utilized and the bottleneck becomes the memory bandwidth.

To maximize the efficiency in computing matrix- matrix multiplication, making the cache size correspond closely to the row/column size of the matrix improves the spatial locality of the data within memory. If the dimensions of the rows/columns fit within the first cache, for example, the number of cache misses will decrease – this is specifically relevant since much of the matrix rows/cols are reused in matrix-matrix multiplication. The strategy to accomplish this, as the paper, Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures‘ mentions, is called blocking. This strategy could improve the efficiency of matrix multiplication by making more efficient use of the CPU cache by dividing the matrices into smaller pieces that fit in the cache and thus do not need to be repeatedly accessed in lower levels of memory due to improved spatial and temporal locality.