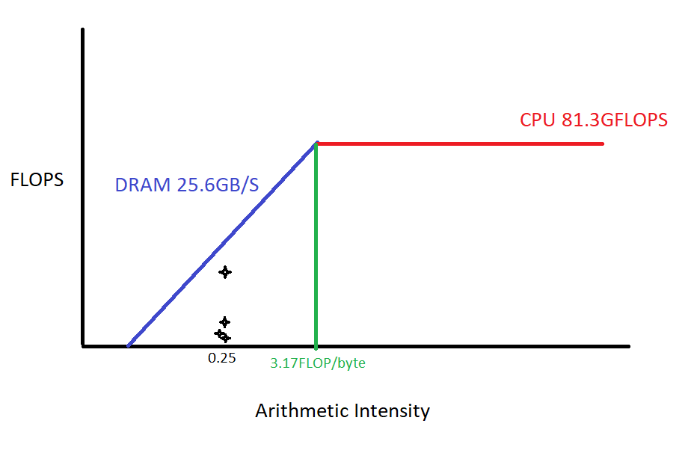
HPC William Jiang

Prof. Finckler 5/17/2020

**Extra Credit Assignment**

The code for the assignment is run using NYU CIMS’s crackle1 server which has the Intel Xeon E5630 CPU.



**Figure: roofline model for Q1, Q2, Q3**

Q1. The result for the unrolled loop dot product is:

**time:1.262464secs bandwidth:6.339985GB/s flops:1.584204GFLOP/s arithmetic\_intensity:0.249875FLOP/byte**

Q2. The L1 and L2 caches are 128KB and 1024KB, respectively (found from [this link](https://www.cpu-world.com/CPUs/Xeon/Intel-Xeon%20E5630%20-%20AT80614005463AA%20(BX80614E5630).html)). To get our blocks to fit within the caches we must first find the appropriate size of the blocks. Since we will be using single-point precision floats (4 bytes) and are storing 2 submatrices in “shared” memory (i.e. cache in this case), we have 100x100 Block dimension for L1-cache. Similarly, for L2-cache, we get 250x250. The result for the tiling matric multiplication routine for BlockDim = 100x100 is:

**time:2.042972secs bandwidth:3.956981GB/s flops:0.983861GFLOP/s arithmetic\_intensity:0.248639FLOP/byte**

For 250x250:

**time:1.932407secs bandwidth:4.158543GB/s flops:1.037048GFLOP/s arithmetic\_intensity:0.249378FLOP/byte**

Q3. I wasn’t able to find the actual implementation of cblas\_sgemm other than that it performs . Therefore, I assume that it uses the standard matrix multiply procedure but with hardware optimizations. The result for the MKL matrix multiply routine is:

**time:0.154507secs bandwidth:51.803576GB/s flops:12.944422GFLOP/s arithmetic\_intensity:0.249875FLOP/byte**

**Notes:**

It is interesting to note that our problem size is 1000x1000x4bytes = 4MB, which happens to fit inside the L3 cache of 12MB. So, the roofline model (which I’ve drawn from the lecture notes since I am using the same CPU) might be inaccurate for our example. L3 cache is faster than DRAM, the scale of which I have not been able to determine from Intel’s spec sheet or anywhere else.