Profiling Report

Group 1

June 6, 2022

Setup

To profile the code we utilised the programs `vtune`[1] and `perf`[2]. We ran the profiling on an Intel Whiskey Lake Processor.

1 First optimizations, ILP

The first optimizations were limited by a low level of ILP. Despite some improvements in the cache locality, performance was still capped at around 0.6 flop/cycle across all matrix sizes because the innermost computation of the MMM consisted of a multiplication and an addition accumulating the result in the same variable. Since the latency of such instruction is 4, the expected performance for the MMM is 0.5 flop/cycle and is coherent with the measured overall NMF performance. Vtune reports a low microarchitecture usage possibly due to lack of ILP.

2 Loop unrolling, hitting the memory bound

To overcome the issue with ILP, the computation in the MMM and the norm was unrolled by a factor of 8 in order to have 8 lines of independent computation. The unrolling factor for the block by block multiplication was chosen to be equal to the number of ports times the latency of the instructions. With a read bandwidth of 7 bytes/cycle and an operational intensity of 0.31 were hit the bound imposed by the memory bandwidth.

Since the rank of the factorization is significantly smaller than the other two dimensions and with a block size expanding to the rank size, increasing the block size would mean adopting a rectangular blocking. This however would not change the memory access pattern and we decided to improve the cache locality by reusing the matrices as much as possible.

3 Matrix reuse

Considering a LLC of size 6MB, a squared matrix V mxm and the rank of the factorization being 16, the working set stops fitting in L3 for m greater 900

$$m^2 + 2rm - cache\_size/8 > 0$$

The performance plot show a drop in performance at 800, which can be explained by the fact that some intermediate results had to be stored in temporary matrices. As we can see from profiling

[1] `https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html`
[2] `https://man7.org/linux/man-pages/man1/perf.1.html`

op47, opt53, opt60 and opt61 for m = 1256 (this way V is double the size of L3), we significantly reduced the number of cache misses. Each optimization was run for 100 iterations.

Opt	LLC_miss
47	19.6M
53	14.5M
60	6.2M
61	2.1M

Computing W_{n+1} blockwise and reusing it for the computation of H_{n+1} reduces the number of LLC by 26%, likewise not computing the entire approximation matrix reduces the LLC by 65% and the two optimization combined reduce the LLC miss by 88%.

At each multiplicative update in opt60, the error function avoids reading and storing the entire computation matrix.