Programming Assignment #4

Due May 1, 2020 by 11:59pm **Points** 4 **Submitting** a text entry box or a file upload
Available until May 6, 2020 at 11:59pm

This assignment was locked May 6, 2020 at 11:59pm.

***Note**: In light of the circumstances surrounding the Covid-19 pandemic, this assignment will be designated as optional, in the sense that: (a) If you do not turn it in, your overall programming assignment grade will be determined solely by programming assignments #1 through #3, and (b) If your grade for assignment #4 is less than the average for assignments #1-#3, then only that earlier assignment grade average will count towards your overall programming assignment grade.*

In this assignment, you will survey the dependence of GEMM performance (General Matrix-Matrix multiply) operations on parameters such as (a) Matrix Size, (b) Block size (for algorithms relying on "blocking" practices), and (c) Thread count in an OpenMP implementation.

The recommended starting point for this exploration is the blocked GEMM implementation in subdirectory ***DenseAlgebra/GEMM_Test_0_10*** in our code repository, which should compile/run with the most ease across all platforms, but you are welcome to use some of the later, assembly-optimized blocked implementations if you can successfully build them, e.g. ***DenseAlgebra/GEMM_Test_1_2_avx2*** or ***DenseAlgebra/GEMM_Test_1_2_avx2.***

Specifically, you are asked to benchmark the following combinations of parameters:

i. Test 3 different matrix resolutions. For example, you could use matrix sizes 1024x1024, 2048x2048, and 4096x4096 (assuming the computer you are running the tests on is fast enough to complete the test for the highest resolution at a reasonable time). You are free to venture into smaller or larger matrix sizes, if you prefer, among the 3 resolutions you choose. Note that smaller matrices might make cache effects be more evident (matrices might start fitting in local caches), while larger resolutions might make parallelism easier (more work to do, easier to distribute on threads).

ii. The implementation you were pointed to as a starting point uses a "blocking" concept to accelerate the matrix-matrix multiply operation. The block size (which has been 32x32 or 64x64 in most of our examples) is also a parameter you can modify. Experiment with 3 different block sizes (it is recommended that you try powers of 2, i.e. 16x16, 32x32, 64x64, 128x128 ...) to examine how this choice affects parallel performance.

iii. Examine how the number of active threads used (which can be controlled via the environmental variable OMP_NUM_THREADS) influences the scaling performance across different matrix sizes and different block sizes. At the very minimum, experiment with using just 1 thread, and with using all

available threads on your computer (which is OpenMP's default behavior). A third option of using some intermediate number of threads would be desirable, but not mandatory.

Create a table with the parallel performance in your experiments. This table may include, for example, 18 different timings (3 resolutions, 3 block sizes, one/all threads) or 27 timings if you choose to use another intermediate core count in your experiment. Contrast these timings with the MKL-optimized version (***DenseAlgebra/GEMM_Test_0_1****)* for the corresponding matrix resolutions and number of threads (note: the MKL version does not allow customization of "block sizes"; just use it as-is). Provide some commentary on notable observations resulting from the numbers you collect, any any thoughts on possible explanations. For example:

- Does operating at certain (large or small) matrix sizes seem to make it less important what the size of blocks is (for performance)?
- Are you seeing the same changes in parameters such as block size *helping* performance in some resolutions and *hindering* performance in others?
- For some of the resolutions/block sizes in your experiment, are you witnessing speed-up that is not quite proportional with the number of cores used when using all vs. one thread?

Deliverable: Only a write-up of your timing experiments and commentary is required. You are welcome to submit the code you used in collecting these experiments, if you made non-trivial changes.

Grading:

- You will be graded on a 0-3 scale (0 = very inadequate, 1 = quite problematic, 2 = buggy, or other minor issues, 3 = correct).
- A grade of 4 ("above and beyond") will be very rarely awarded for amazing work.