CS 378 – Project #1

Due Thursday, Sep 25, Midnight

You may, but don’t have to, work in groups up to 2. Turn in a single project for all members of the group with all names listed.

1 Miss Ratio Measurement on Real Hardware

Write the 6 variants of matrix-matrix multiply you can generate by interchanging loops. The data-type in the matrix should be doubles.

- Instrument the implementation using PAPI to measure:
  - PAPI_LST_INS (Total Load Store Instructions)
  - PAPI_FP_INS (Total Floating Point Instructions)
  - PAPI_L1_DCM (Data L1 miss)
  - PAPI_L2_TCM (Total L2 miss)

- Using stampeded at tacc, collect these measurements for each matrix size ranging from 1 to 400. Since the values you obtain will depend a lot on the machine you use, you must use stampede for the numbers you report. To ensure no interference with other processes, submit your runs to the job scheduler.

- Plot the miss rates for L1 and L2 for each of the matrix sizes, as well as the total load and store instructions and floating point instructions for each matrix size.

1.1 Notes

Flush the cache (L1 and L2) before the matrix multiply.

1.2 Deliverables

Submit your code and a writeup (pdf) showing the plots.

2 Fast Matrix Multiply

Write a very fast matrix multiply. Your code should work for any size square matrix. You may use any of the techniques presented in class or from the literature (except, of course, calling an existing library).

This will be graded on performance. For 3000x3000 using Eigen with vectorization disabled, I see about 7.5 GFLOPS on stampede. Using ijk I see 205 MFLOPS. Blocking for L1 I get 1.1 GFLOPS. I am taking $2 \times N^3/time$. The target performance for your code is 3 GFLOPS using this metric. Don’t explicitly vectorize your code, vectorization will be a future project. Use gcc (I suggest the flags -ffast-math -O3 -fno-tree-vectorize). This means a 1000x1000 matrix multiply should take no more than 0.67 sec and a 3000x3000 18 seconds.
2.1 Notes

This isn’t easy. Don’t expect you can finish this in a couple hours.

   Since even old GCCs will vectorize your code, compile with -fno-tree-vectorize. Performance tests will be compiled with -ffast-math -O3 -fno-tree-vectorize.

2.2 Deliverables

Submit your code and include in your writeup performance (both FLOPS and time) as a function of matrix size for up to 3000x3000 matrix.

3 Turning in

Measurements should be taken on stampede at tacc. Do your development on the login node, but submit a job to run the measurements (to ensure your code runs on a dedicated machine).

   Turn in by email an archive (.zip or .tgz) to the TA.

4 Bonus

First bonus is if your code beats my code. The web page gives a link to a sample executable. I’ve complied with the same flags listed above.

   Second bonus goes to the fastest, correct two implementations.