CPS 5401 Introduction to Computational Science Name _____
Shirley Moore, Instructor
Due September 29, 2014

Lab 2: Programming for Performance

Please upload your finished programs (source code only) and Makefile to a directory called Lab2 in your SVN repository for this course. You may upload your answers to the questions (in a PDF file) to this directory as well or turn in a printed copy in class.

1. Download and install the PAPI library in your account on the lab machines.

2. Construct a Roofline model for our lab machines. Use the STREAM benchmark to find the effective memory bandwidth.

3. Write pseudo-code for matrix-vector multiplication y = Ax. Does this algorithm have reuse – if so, where? Analyze the arithmetic intensity of the code, assuming the vectors fit in cache. According to the Roofline model, what performance would you expect to get on our lab machines?

4. Change the matmult.c code from Linux Lab 1 to use double precision real data. Instrument the code with PAPI_flops(). Note that you can change the dimensions of the arrays to stress different parts of the memory hierarchy.

a. Assuming all three arrays fit in cache (don't worry about levels of cache for this question), analyze the arithmetic intensity of the code. According to our Roofline model, what performance would you expect in this case? Dimension the arrays so that they all fit in cache and measure the performance. Do you get the performance you expect? Try to explain why or why not.

b. Put the arrays back to the original dimensions. Compare performance for the original ordering of the loops with the performance after switching the order of the j and k loops. Explain why the performance is different. Note that arrays are stored in row-major order in C. Measure the L1 and L2 cache misses for each version using PAPI and compare the results. Analyze the arithmetic intensity of the better version of the code and place it on the Roofline model.

c. See http://www.netlib.org/utk/papers/autoblock/node2.html for an explanation of blocked matrix multiplication. Assuming blocks of all three matrices fit in cache, analyze the arithmetic intensity of the code. Implement blocking in matmult-blocked.c and measure the performance and the cache misses. Do you get the performance you expect? Try to explain why or why not.

d. See https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html for an explanation of GNU compiler options that control performance optimization. Try different compiler optimization levels –O0 through –O3 with the original and blocked matrix multiplication code and observe the results. Which optimization level obtains the biggest performance gain (compared to the previous level) and what sorts of optimizations is it doing?

e. Construct a Makefile with a separate target for each program you wrote. Use macros to define the compiler, compiler flags, linker flags, and PAPI library.