# CS594 – Scientific Computing for Engineers

Homework #5
15 February 2012
Due: 29 February 2012

This assignment involves analyzing performance of matrix multiplication code using PAPI and various compiler optimizations.

Please run your experiments on one of the machines in the hydra lab. You can log in remotely to these machines via `ssh` to `hydra`*NUM*`.eecs.utk.edu` (where *NUM* is a number from 1 to 30).

Please provide your homework results as a .pdf file, e-mailed to the TA.

To begin, download the `hw5.tar.gz` file from the class website. Uncompress it on the hydra machine with `tar -xzvf hw5.tar.gz`. Enter the subdirectory with `cd hw5` and run `make` to compile the `mmm_papi` executable. Run it with `./mmm_papi`. You should see some results with time and floating point events.

The `mmm_papi` program takes two command line parameters: the first is a size of the matrix to use in the matrix-matrix multiply, the second is the name of the event to measure (as found with the `papi_avail` utility). For example: `./mmm_papi 480 PAPI_TOT_CYC`

The questions will ask you to change the compiler optimization level. To do this, edit the Makefile and uncomment/comment the proper `CFLAGS` line so only the one you are interested in is selected. (In a Makefile the comment specifier is a # character at the beginning of the line). Once the proper `CFLAGS` line is set, re-build `mmm_papi` by running `make clean` followed by `make`.

Please answer the following questions:

1. Why does the provided PAPI code use the `PAPI_get_virt_usecs()` call rather than `PAPI_get_real_usecs()`?

2. Plot the results for GFLOP/s (this is PAPI_FP_OPS/s divided by 1.0e9) with matrix sizes 100, 200, 400, 800 and 1000. Run the tests with `papi_mmm` compiled with no optimization, `-O3`, and `-O3 -msse4 -march=native` (by setting the CFLAGS and recompiling, as described

previously). On the same graph, plot the results of the `mmm_atlas` program.

Does changing the optimization level (by varying the options to the compiler) increase the GFLOP/s rate? Can the compiler optimize the naive matrix multiply to be as good as the tuned ATLAS version?

Explain any interesting features you see in the graph, especially any difference between the ATLAS plots versus the naive `mmm_papi` ones.

3. Plot the results for `PAPI_TOT_INS` with matrix sizes 100, 200, 400, 800 and 1000. (Note, make sure you plot total instructions, *not* instructions/s). Make the y-axis logarithmic to make the results clearer.

   Run the tests with `papi_mmm` compiled with no optimization, `-O3`, and `-O3 -msse4 -march=native`. On the same graph, plot the results of the `mmm_atlas` program.

   Does executing fewer instructions correspond with better performance?

   Describe a case when running fewer instructions *does not* provide better performance.

4. Plot the results for L2 Total Cache Misses (use `papi_avail` to find the name of this event) with matrix sizes 100, 200, 400, 800 and 1000. Make the y-axis logarithmic to make the results clearer.

   Run the tests with `papi_mmm` compiled with no optimization, `-O3`, and `-O3 -msse4 -march=native`. On the same graph, plot the results of the `mmm_atlas` program.

   Can you explain what you see in this graph? Remember that unlike the previous events we measured, Cache behavior can be non-deterministic and depend on outside effects. Do any of the results you see seem odd and hard to explain?

   Does this graph indicate to you that the naive matrix-matrix multiply is memory bound or CPU bound?

   What optimizations might ATLAS be doing that reduce its cache miss rate?

5. Which other event available in `papi_avail` do you think might tell you interesting behavior about the benchmarks? Why?