Hardware provides performance counters which allow us to measure micro-architectural events such as clock ticks, floating-point multiplies, pipeline flushes to do to 4k-modulo aliasing in the load-store queue, etc. We will use these to measure the behavior of a simple matrix-matrix multiply.

Recall the code for the ijk-ordered matrix multiply. Implement this version of matrix multiply. Instrument the code using PAPI to measure total instructions, total cycles, total floating point instructions, data L1 misses, and total L2 misses (see note about making multiple measurements). Measure these values for a matrix multiply as a function of problem size. Reason from your source code what the values should be as a function of problem size.

Make the measurements with a matrix initialized with finite random numbers which will not result in inf, nan, or denormal numbers in the result. These drastically slow down the hardware.

Make the measurements with a cold cache. There isn’t an architectural way to flush a cache from userspace, but if you reflect on how a cache works, you should see that you can write code that will effectively do so. You only need to flush L1 and L2.

You cannot do more than 1 matrix matrix multiply for a given allocation or you will not get valid numbers. It is fine to take multiple measurements with one execution of your code, but you must reallocate the matrixes each time.

Turning in

Measurements may be taken on any Linux machine with PAPI and working permissions for you to use PAPI. If possible, stampede at TACC is recommended, but we may not have access yet. When using stampede, Do your development on the login node, but submit a job to run the measurements (to ensure your code runs on a dedicated machine).

Please write up your experiment and show the graph of the performance counters v.s. problem size (N x N matrix) up to $N = 300$. You should be able to justify the number of loads and stores and floating point instructions based on the matrix multiply problem size. You don’t need to collect numbers for every possible problem size up to 300, but you should do all the small values.

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

Useful Notes

Tacc uses a module system to load software. module avail will list available software, module load papi will add papi to your path, module help papi will show you what you have to do to compile and link with papi. papi_avail will list available hardware performance counters.

You will need to measure many performance counters and many loop variants next week, so structure your code accordingly.

The hardware can only collect a few performance counters at once, so you may have to run your code more than once to collect all the numbers. Which subsets of performance counters can be measured simultaneously varies.

The loop may be vectorized, you might try turning off automatic vectorization in the compiler if your numbers seem off by a constant factor.
Bonus Questions (Extra Credit)

Why reallocate the matrixes between measurements? This randomized the starting addresses. Since you explicitly flushed the cache, this provision of the homework doesn’t seem like it should be addressing a cache related effect. Are there performance counters you could use to test this hypothesis?

The compiler may be able to vectorize the code or reorder the loops. How would you tell from just the performance counter data? Does either gcc or icc reorder the loops to a more efficient matrix multiply form?