B649: Parallel Architectures and Programming
Assignment 3: Memory Hierarchy Optimizations

Announced: 2011-11-21 (Released: 2011-11-23) Due: **2011-12-02, 11:59 pm**

1 Motivation

This assignment will help you understand how memory-hierarchy optimizations work in practice, on real machines and real code examples. We will look at two optimization techniques: *blocking* (also called *tiling*) for temporal locality and adjusting the data layout for spatial locality.

2 Deliverables

You have been given access to a mercurial repository called `A3` that is accessible only to you and the instructor, which you can access as follows:

```
$ hg clone ssh://username@sharks.cs.indiana.edu//u/achauhan/HG/Teaching/B649/2011-Fall/username/A3
```

Replace *username* by your own CS username. All your code, examples, and results should be committed within this repository. Make sure that you commit **only** the essential files. In other words, **do not commit** editor backup files, object (`.o`) files, executables, and any other temporary files. You are strongly encouraged to use `make` or similar utility to manage your code, testing, and report generation. Since we are concerned about controlling the behavior of the code in a predictable manner on the hardware, the programs must be written in C or C++. You will use PAPI to access the hardware counters provided on most modern processors in order to measure cache misses. Note that there is only a limited number of hardware counters on a processor, which restricts the number of distinct quantities (such as cache misses) that can be measured in any given program run. PAPI is already installed on `timon` and `pumbaa`.

In order to ensure repeatability and eliminate noise, run each test several times (at least, 10 times) and take averages.

Part 1 (Blocking): 35 points Modify the simple matrix-multiply that you wrote for Assignment 1 to a blocked version, as discussed in class. Use PAPI to measure L1 and L2 cache misses for a range of matrix sizes and block sizes. Plot two graphs, one each for L1 and L2 misses. Each graph will consist of multiple lines, representing different block sizes. The horizontal axis will represent matrix size (one side of a square matrix) and the vertical axis will represent the number of misses. Run your tests for as many matrix sizes as you can and go up to the largest matrix size that would fit in the physical memory of the machine. Run your tests for block sizes 1×1 (which is equivalent to simple, unblocked matrix multiply), 4×4, 16×16, 64×64, 256×256, 1024×1024, and 4096×4096.

Part 2 (Data layout): 15 points In class we considered an example of matching traversal through the elements of a multi-dimensional array with its layout in memory to improve spatial locality. That was achieved by changing the code (loop-interchange). An alternative is to change the data layout in memory. An example where this makes sense is an array of `struct` types.

Construct an example where looping through a field of an array-of-structs results in poor spatial locality. Then, change the array-of-structs into struct-of-arrays (or, equivalently, multiple arrays). Measure cache misses for a range of array sizes, as in Part 1, to demonstrate that doing this transformation improves cache miss rate. Explain your results.