ECE571: Advanced Microprocessor Design – Homework 3

Due: Friday 1 March 2013, 5:00PM

1. Background

   - For this assignment, use the ARM pandaboard machine just as in HW1 and HW2
     `ssh username@vincent-weaver-1.umelst.maine.edu`
   - Create a document that contains the data described in the Analysis sections below. A .pdf or .txt file is preferred but I can accept MS Office format if necessary.
   - Download the HW sourceode to your directory on the pandaboard:
     `wget http://www.eece.maine.edu/~vweaver/classes/ece571_2013s/hw3_code.tar.gz`
     Uncompress with `tar -xzvf hw3_code.tar.gz`.
     Enter the directory and build the code with `make`

2. Cache behavior

   - Part 1
     - Run `./matrix_multiply N` to do an NxN matrix multiply.
       Use `perf stat ./matrix_multiply N`
       and gather the "seconds time elapsed" values for each of matrix sizes 16 24 32 40 48 56 64 72 80 88 96 112 128 160 192 256 320 384 448 512.
     - Plot the above values (runtime in seconds vs array size) with a log/log plot using a graphing program of your choice.
   - Part 2
     - This time run `./matrix_random N` to do an NxN matrix multiply in random order. This will not give the same results, it just randomly iterates over the matrices in order to access them in a random manner. Use `perf stat ./matrix_random N` and gather the "seconds time elapsed" values for matrix sizes 16 24 32 40 48 56 64 72 80 88 96 112 128 160 192 256 320 384 448 512.
     - Plot the above values (runtime (in seconds) vs array size) with a log/log plot using a graphing program of your choice. Plot the values on the same plot as Part 1.
   - Analysis
     (a) Include the graph described above.
     (b) In theory performance should plateau at some level and then ramp up when our memory size grows larger than an on-chip memory structure. For example, L1 dcache (which is 32kB 4-way), L2 dcache (which is 1MB 16-way) or micro-TLB coverage (with is 32 entries times 4kb pages, or 128k).

         The predominant code in the benchmarks accesses two NxN arrays of 8-byte floating point values, so the size of the arrays is 2*8*N*N bytes. To translate back from kB to N you would divide by 16 and take the square root. So the value of N that might indicate L1 is 45, L2 is 256, and TLB is 90.

Do you see any changes in the graph at these values?
Do you see changes in the graph at other values?
What hardware features might explain the graphs you see?
Do the graphs correspond with expected results?

3. Branches

   - The `./branch_test` example just reads in a file from standard input and counts the types of ASCII characters it finds: total, uppercase, lowercase, and numbers.
   - Run `perf stat --log-fd=2 -- ./branch_test < random.input`
     Note the branch miss rate and the cycle count
   - Run `perf stat --log-fd=2 -- ./branch_test < sorted.input`
     Note the branch miss rate and the cycle count
   - Analysis
     (a) What were the values for cycle count and branch miss rate for the random and soted cases?
     (b) By what percent did the cycle count improve when using the sorted input?
     (c) Why did the branch miss rate go down with the sorted input?

4. Submitting your work.

   - Create the file as described in the Background and Analysis sections.
   - Please make sure your name appears in the document.
   - e-mail the file to me by the homework deadline.