LSU EE 7700-1 **Homework 2** **Due: 28 March 2007**

The dataset files for this assignment, the same as those used for Homework 1, are located in directory /home/faculty/koppel/pub/ds/2007/batch.sm.0098 which should be accessible from ECE Linux machines. The dataset files are named using *run ids*, view the web page `index.html` to see the benchmark and machine configuration. If your system is set up properly you can view the dataset by clicking the run id.

The simulated machines have the following characteristics:

- Dynamically scheduled. (Out of order execution.)
- Superscalar. Issue (fetch/decode/commit) width (IW) is either 2, 4, 8, or 16 instructions per cycle (depending on configuration).
- Reorder buffer (ROB) size is either 64, 128, 256, or 512 entries.
- Can predict either 1 or 3 blocks per cycle.
- Uses YAGS predictor with 8-branch GHR and 2^{16}-entry PHTs.

For the problems below **consider only systems** *that are 8-way superscalar (*`IW = 8`*) and that can predict three blocks per cycle.*

For the problems below it will be helpful to find all occurrences of a particular instruction in PSE. To do so enter the address of the instruction into the search box displayed below the main toolbar in both the overview and segment plots. Enter the address in hexadecimal using C syntax (e.g., `0x1da4c`*). In the overview plot segments in which the instruction appears are highlighted by blue bars. The arrows on either side of the search box can be used to move to the first, previous, next, or last occurrence.*

Problem 1: Among the systems with 256-entry ROBs (and 8-way superscalar with 3 blocks per cycle predicted) find a load with the following characteristics:

- Occurs frequently (in at least 1/3 of the segments) and experiences some L2 cache misses.
- Is a good candidate for hardware prefetch (see next problem).
- Is a good candidate for pre-execution (see Problem 3).

(*a*) Provide the following information: Name of benchmark, run id, address of the load, and a segment number in which that load suffers an L2-cache miss.

(*b*) Using PSE determine the miss time (the time from when the address is resolved (yellow) until the data has arrived [light blue, check dependent instructions if you're not sure]).

Consider a *Lucky Load [tm]* system which is just like the one chosen for this problem except your load (the one chosen for this problem) always hits the L1 cache (yes, it's the lucky load!). The hit ratios for other loads are identical (so they are not lucky) on the two systems, as are branch predictions, etc. Assume that on both systems your load is always on the critical path. Compute the speedup of the Lucky Load system over the original one. (The speedup is execution time of the original system divided by the execution time of the new one.)

For benchmarks which spend the bulk of their time in a small kernel of code a single load instruction can have a significant impact on overall execution time, on others, like gcc, even the worst-behaved load can have only a trivial impact. For this problem, find a benchmark in which a single load does have a significant benchmark.

Problem 2: Determine some details of hardware prefetch for the load from the previous problem. See Vanderwiel 2000 ACM CS, `http://www.ece.lsu.edu/tca/papers/vanderwiel-00.pdf`, for some background.

(*a*) Describe the address reference pattern emitted by the load. If it's not sequential or stride, explain how it would be predicted in hardware. (To make life easy, limit yourselves to sequential and stride.)

(*b*) An important design question for hardware prefetch is triggering, when to initiate the prefetch of the next address in the sequence. (See Section 4.1 of Vanderwiel 2000 ACM CS.) For your load describe an

effective triggering. The triggering may be something described in Vanderwiel, something used elsewhere, or something you invent. The triggering must be implementable and should be timed such that the prefetched line arrives soon before the load that needs it executes.

Problem 3: Find and evaluate a good p-thread for the load. A good p-thread maximizes aggregate advantage (latency tolerance minus overhead). See Roth 02 Micro (`http://www.ece.lsu.edu/tca/s/roth-02.pdf`) for terminology and background, but before reading it note that the aggregate advantage computation for this problem uses some simplifying assumptions.

(*a*) Find a good p-thread for the load. Show the instructions (including addresses). Show the original instructions in the backward slice, and if any optimization was done show the final p-thread (if no optimization is done the backward slice is the p-thread).

(*b*) Estimate the overhead as follows. Assume that the trigger (first backward slice instruction used to construct the p-thread) and target load execute the same number of times and that the path from the trigger to the load is always the same. Further, assume that p-thread instructions can be injected at a rate of 8 insn per cycle and when doing so fetch/decode of main-program instructions stops. Assuming pre-execution never works, what is the speedup given this overhead? (The speedup should be less than one.)

(*c*) Find a place where the overhead is on the critical path. If possible, find a place where overhead is not on the critical path. Otherwise, sketch an execution diagram in which the overhead would not be on the critical path. (Think about "fetch limited.")

(*d*) Estimate the latency tolerance given that the load misses. The easiest way is to work out a pipeline execution diagram of the p-thread (using timings based on those observed using PSE) and compare the time that the p-thread load starts to the time the main thread load starts. (Note that the formula for latency tolerance in Roth 02, Equation 5 in Table 1, describes the same sort of calculation one would do with a pipeline execution diagram.)

(*e*) Estimate how much of that latency tolerance will result in improved performance. Do this using an execution diagram of a sample miss.

Problem 4: Find another load which misses the L1 or L2 cache, is a good candidate for pre-execution, but is NOT a good candidate for prefetch.

(*a*) Provide the following information: Name of benchmark, run id, address of the load, and a segment number in which that load suffers a cache miss.

(*b*) Show a p-thread.

(*c*) Indicate why the load is not a good candidate for hardware prefetch.

Problem 5: Find a load which misses the L1 or L2 cache but is NOT a good candidate for pre-execution or for hardware prefetch.

(*a*) Provide the following information: Name of benchmark, run id, address of the load, and a segment number in which that load suffers a cache miss.

(*b*) Explain why it is not a good candidate for either mechanism.