CSCE 614 Fall 2016 Project 2

The second project for our class will be a cache replacement contest. It will again be an individual projects, i.e., no group project. The project is tentatively due on December 10, 2015.

The Project

You will implement a last-level cache replacement and bypass policy. You will use a simulation infrastructure that will be distributed to implement your policy. The basic idea is this: a 16-way set-associative cache simulator will call your code to decide which among 16 blocks in a set should be evicted when a replacement is needed. Your code will return a number from 0 to 15 to indicate the block number (way) of the victim block, or -1 to indicate that no block should be replaced, i.e. that the incoming block should bypass the cache. Your code will also be consulted when a block is referenced so that it may update its state. The simulator can be downloaded from here. The traces are here (tar file) and here (individual files). Note that the traces are quite large so the tar file is about 10GB. If you would rather not download them, you can access them on the CS network through ~djimenez/traces. Maybe. If don't know if that works. Let me know. Unpack the file and read the README file for more information about how to use the infrastructure.

How To Choose A Cache Replacement Policy

You have two options: choose one of the following cache replacement policies, or propose another one. You may choose from one of the following papers:

1. Qureshi *et al*., ISCA 2007, Adaptive Insertion Policies for High Performance Caching.
2. Xie and Loh, ISCA 2009, PIPP: Promotion/Insertion Pseudo-Partitioning of Multi-Core Shared Caches
3. Petoumenos *et al*., SAMOS 2009, Instruction-Based Reuse-Distance Prediction for Effective Cache Management
4. Jaleel *et al*., ISCA 2010, High Performance Cache Replacement Using Re-Reference Interval Prediction
5. Khan *et al*., MICRO 2010, Sampling Dead Block Prediction for Last-Level Caches
6. Wu *et al*., MICRO 2011, SHiP: Signature-Based Hit Predictor for High Performance Caching
7. Khan *et al*., HPCA 2014, Improving Cache Performance by Exploiting Read-Write Disparity

Alternately, you may propose another paper, or if you like, your own novel replacement policy. In this case you will need to submit an informal proposal by email to the professor or discuss it in person.

Optimizing Your Policy

You are responsible for implementing the technique in the paper you choose, but beyond that you may feel free to implement additional optimizations. For instance, some of the papers do not do bypassing; you may augment the technique to decide when to do bypassing.

What Is Due for the Project

You will turn in a project writeup giving an introduction, related work, main idea of the replacement policy, methodology, results with bar charts, and conclusion. You will also turn in your source code: the files `replacement_state.cpp` and `replacement_state.h`. As with the branch prediction project, your code must compile and run cleanly under the unmodified simulation infrastructure. It must not make any output.

Hardware Budget

Unlike the branch prediction contest, we will not place a hardware budget restriction on the cache replacement policy. Use a reasonable amount of storage (i.e., the cache is 4MB so if you find yourself using several megabytes, that is unreasonable).

How Is The Contest Judged?

The contest will be won by the entry that has the best geometric mean speedup over the LRU policy. That is, for each trace we will divide the IPC given by your technique by the IPC given by LRU, then find the geometric mean of all of these speedups. The highest one wins.

Due Date

The project is tentatively due on December 10, 2015.