Project 1: Cache Simulator

1 Overview

This lab will help you better understand cache memories. You will extend the functionality of your cache simulator from ICS to support new replacement techniques and storage policies.

2 Logistics

This is a partner project. You may use C or C++ for this assignment, and you should only rely on standard libraries. The fifth lecture of the course gave an overview of the simulation framework that will be provided in a private repository for each group.

- Create a copy of the cache subdirectory
- Modify the CMakeLists.txt to add that subdirectory
- Modify the CMakeLists.txt in your subdirectory to name the project
- Run `$ cmake .` in the root of the repository

3 Cache Simulator

You will be implementing a cache simulator, an extension of the functionality you implemented in ICS. The cache will support 64-bit addresses and is byte-addressable.

Your cache will need to support the flags from cachelab:

-s : Number of set index bits ($S = 2^s$ is the number of sets)

-E : Associativity (number of lines per set)

-b : Number of block bits ($B = 2^b$ is the block size), with a range of [4,10]

The cache can have two replacement policies:

- LRU (default)
- RRIP (-R <k>) The static Re-Reference Interval Prediction (RRIP) algorithm learns a re-reference prediction value (RRPV) for each cache block to estimate how near in the future the block will be accessed again. The goal of RRIP is to prevent blocks estimated as having a distance re-reference interval from polluting the cache.
  To implement an M-bit RRIP policy, each cache block is assigned a RRPV of k-bits. On a cache hit, the RRPV is reset to 0 for the cache block. On a cache miss, the eviction algorithm finds the first cache block predicted to be accessed in the distant future, i.e.

RRPV of $2^k - 1$, and evicts that block. If none is found, it increments all blocks by one and continues until we find a block to evict. When inserting a block, the RRPV for the block should be set to some non-zero value. We recommend the use of $2^k - 2$ for this lab. For more information on this policy and its implementation details, refer to the paper: High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP).

The cache may also have a victim cache (-i <v>), which contains v entries (max 8, default 0). The victim cache is a small, fully associative cache. The victim cache is always LRU, so inserting an entry is the most recently accessed. Entries moving from victim cache to rest of cache are also always placed as the most recently accessed. This feature is *not* required, nor will it be evaluated, in Fall 2022.

Caches are expected to implement subblocking, via (-u <U>), such that each line then contains 2^U blocks. For practical purposes, U should be between 0 (default) and 10. Each subblock of the line will maintain a separate valid state, but otherwise share replacement and dirty information.

To help resolve confusion, if an access spans two cache lines, the lower address is accessed first.

Later in the semester, your cache simulator will need to be usable in a multiprocessor simulation. There is also the following "future" features:

The global variable, processorCount, will be set by the simulation engine.

Simulators will also rely on a separate coherence and interconnection component

At present and for simplicity, you can assume that a hit takes 1 cycle. Cache misses take 100 cycles per block, with an additional 50 cycles per evicted dirty block. The miss times will be variable once the interconnection component is used.

4 Interface

You should only modify files within your cache's subdirectory, along with the CMakeLists.txt. Modifying other files may result in simulator instability or merge conflicts at later points.

The following functions may need further explanation:

```
cache* init(cache_sim_args* csa)
```

Uses the arguments in csa to initialize the cache, including any allocation of structures. On success, returns a pointer to the allocated cache, including the interface functions. On failure, returns NULL.

```
void memoryRequest(trace_op* op, int processorNum, int64_t tag,
void(*callback)(int64_t))
```

op - Memory operation being accessed by the processor

processorNum - currently 0

tag - identifier of this request, will matter if / when processors have multiple memory accesses. This is different from the tag bits for an address.

callback(tag) - function in the processor component to be called when the memory request of that tag completes

5 Running the Simulator

Recall from lecture 5 that most of the configuration for the simulator is in the configuration file. Two sample files are provided for this assignment: ex_sub.config and ex_rrip.config.

```
/usr/bin/time ./cadss-engine -s ex_rrip.config -c refCache -t astar.trace
Ticks - 13757067
{"real":0.26, "user":0.24, "sys":0.00, "mem":1984 }
```

To test other configurations, you will need to make edits or create new configuration files.

The small trace files from 213 cachelab have been provided for you; however, a set of five trace files have been placed into AFS. Later labs will rely on increasingly longer and more complex traces.

6 Experiments

For each trace in the trace directory*, design a cache subject to the following goals:

1. You have a total budget of 54KB for the entire cache system's storage (including all tag storage, valid bits, data storage and overhead storage)

2. For simplicity, RRIP uses k-bits per line. LRU uses e-bits per line.

3. The cache should have the lowest possible AAT, there may be multiple solutions. We do not have a target value. But you should propose one configuration for all of the experiment traces.

7 What to hand in

1. A short report (PDF) on the design and experiments for your simulator, to be submitted to Gradescope. About 2 pages in “technical paper” formatting (suggested but not required - https://www.acm.org/publications/proceedings-template).

2. The commented source code for the simulator program itself. Typing “cmake .; make” should be sufficient to build your simulator. We recommend committing your work regularly and downloading the zip from github as your submission file.

3. **Autolab will use grace days for late submissions. After the deadline, Prof Railing will review submissions and provide feedback. Any simulator that is at the “experiment stage”, will have the opportunity to resubmit.**

8 Grading (approximate)

0%	You do not hand in anything by the deadline
+40%	Your simulator doesn't run, does not work, but you hand in significant commented code
+22%	Your cache simulator runs (6%), Your simulator supports RRIP and Subblocking (6%), Variable access sizes (10%)
+8%	Your simulator’s output matches the reference
+10%	You ran all experiments and found a cache design
+20%	The report demonstrating key experimental results, simulation design, etc.

9 Reminder

This is a new course, so there may be bugs in the labs, assignments, etc. If the results look wrong, try to come up with a MRE (minimal reproducible example) and post ot Piazza.