Lab 2: Cache

> Due: 2023/12/03 (Sun) 23:59:59

Please note lab 2 takes 12% of the overall score for CS 211, Fall 2023.

In this lab, you would dive deeply into CPU cache, including cache hierarchy, cache replacement policies and victim cache.

Along with the `Simulator` we used before, you can use the standalone cache simulators `CacheSim` and `CacheOptimized` in the `build/` folder to help you verify your implementation. You shall see `MainCache.cpp` and `MainCacheOptimization.cpp` for more information. These cache simulators receive a trace of memory accesses and perform cache simulation as if there was a processor that actually performs these memory operations.

A trace of memory access is a series of read/write operations to memory addresses. In order to build your own test case(s) in the format of trace(s) of memory accesses, take a look at files in the `cache-trace/` folder.

You can use trace(s) of memory accesses to test in section 1, 2 and 3. In that case, you may use the single cache simulator `CacheOptimized` and `CacheSim` to complete the tests. For experiments with applications mentioned in section 4, you should actually run each application on `Simulator` to
evaluate your implementations on one or more real programs.

1. Multi-level Caches

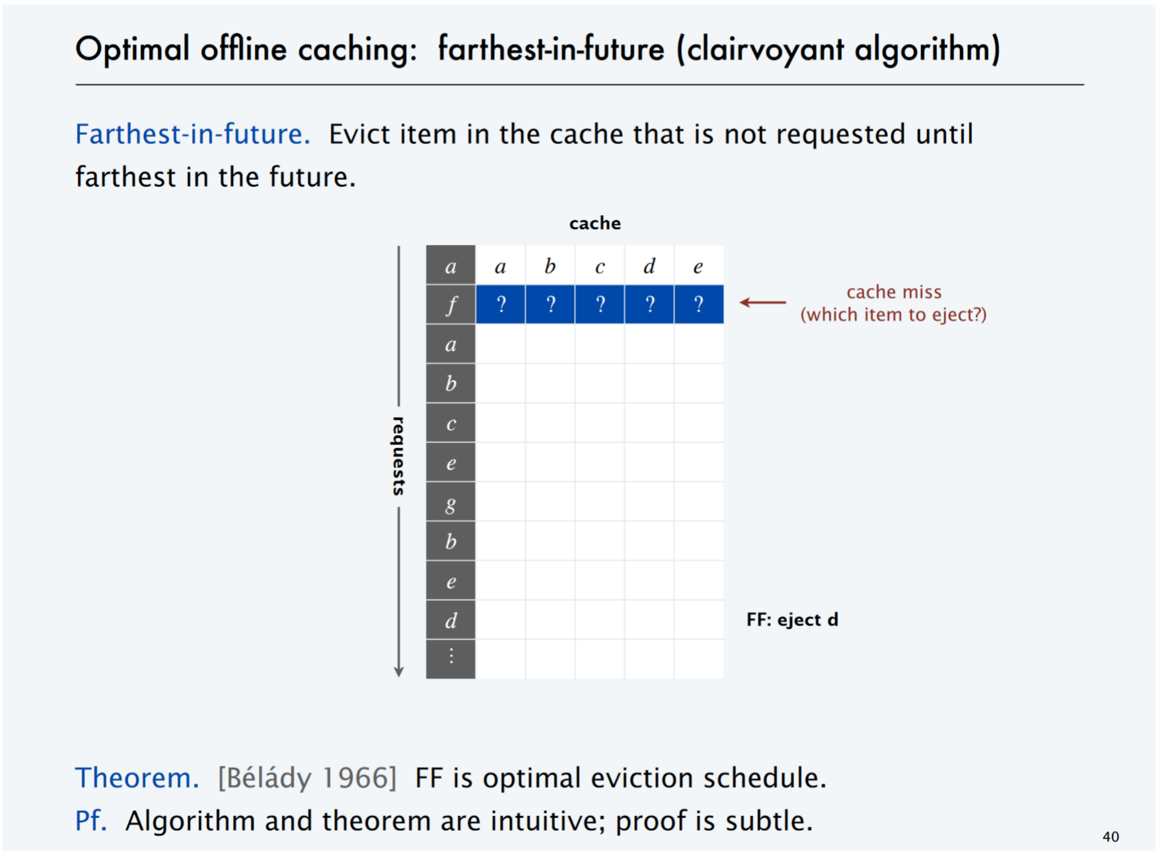
Read the code in `Cache.h` and `Cache.cpp` , etc. Answer which inclusion policy is at this point between the multi-level caches? Is it inclusive, exclusive or non-inclusive?

Implement the **two** other inclusion policies. Your implementation should be conducted on a **2-level cache hierarchy**.

To validate your implementations, you shall design trace(s) of memory accesses as your test case(s).

2. Cache Replacement

Implement **Re-Reference Interval Prediction (RRIP)** and **Optimal (Belady's Algorithm)**, and design experiments to validate your implementations. Here is a description for optimal caching.



Optimal offline caching: farthest-in-future (clairvoyant algorithm)

Farthest-in-future. Evict item in the cache that is not requested until farthest in the future.

Theorem. [Bélády 1966] FF is optimal eviction schedule.
Pf. Algorithm and theorem are intuitive; proof is subtle.

40

You shall provide at least 1 test case(s) to show the performance difference among these replacement policies. The expected performance difference should be: Optimal >= {RRIP, LRU}.

It is suggested that you can justify your implementations with a single-level cache hierarchy. A 2-level cache hierarchy is also encouraged. Note that your cache associativity cannot be lower than 4.

You shall implement RRIP and Optimal by following the paper listed in the references below (1 and 2).

> **References**
>
> 1. RRIP: High performance cache replacement using re-reference interval prediction (RRIP)
> 2. Belady's Algorithm: A study of replacement algorithms for a virtual-storage computer (See: Optimal replacement algorithm start from page 86)
> 3. Optimal offline caching: Slides

To validate your implementations, you shall design **trace(s) of memory accesses** as your test case(s).

3. Victim Cache

Iimplement your victim cache according to the slides of L08 (Page 12), and design experiments to validate your implementations. Your victim cache shall be a small, **fully associative** cache and **First-In First-Out** (FIFO) replacement policy. In your implementation, L1 cache shall be direct-mapped.

Victim cache shall complete its access in 1 cycle. You can vary the size of victim cache to observe the consequence of this factor on performance. You shall implement with an **inclusive** cache hierarchy. However, you only need to consider L1 and L2, without victim cache, for the inclusive hierarchy.

> Bonus will be credited to those who explore victim cache design with exclusive/non-inclusive cache hierarchy. Still, you only need to consider L1 and L2 caches, without victim cache, for either inclusion policy.

To validate your implementations, you shall design trace(s) of memory accesses as your test case(s).

4. Evaluation with Application(s)

In this section, you shall use real-world application(s) to test your cache implementations. Implement algorithm(s) of your choice (sorting, graph search, etc.) as your test case(s). Complete the compilation using the previous cross-compilation process.

After trace tests, run the CPU `Simulator` to experiment the composition of **RRIP** and **LRU** with a two-level cache hierarchy, and no fewer than one **application** (**NOT traces of memory accesses**) as your test case(s). Compare the performances between different combinations of replacement policies on each cache level and three inclusion policies.

Your experiments shall be conducted on a 2-level cache hierarchy. You can consider varying the associativity and sizes for both cache levels as well as the victim cache. However, the set associativity of L1/L2 cache cannot be lower than 4. Also keep the size of L2 size larger than that of L1 and the size of victim cache shall be realistically small.

For your convenience, here are the experiment variations among different settings you shall cover.

Experiment Variation	L1 Replacement Policy	L2 Replacement Policy	Inclusion Policy
1	RRIP	RRIP	Inclusive
2	RRIP	LRU	Inclusive
3	RRIP	RRIP	Exclusive
4	RRIP	LRU	Exclusive
5	RRIP	RRIP	Non-Inclusive
6	RRIP	LRU	Non-Inclusive
7	LRU	RRIP	Inclusive
8	LRU	LRU	Inclusive
9	LRU	RRIP	Exclusive
10	LRU	LRU	Exclusive
11	LRU	RRIP	Non-Inclusive
12	LRU	LRU	Non-Inclusive

5. Report

You need to prepare a report for this lab. The lab report must include at least the following sections:

1. Answers to the question in section 1;
2. Description of your designs and implementations;
3. Evaluation and analysis on your results, test case(s), etc.

6. Hand In

The submission shall include the following three parts:

1. Source code that can be built and executed;
2. Your test case(s);
3. Readable lab report in PDF format.

You shall pack the source code, test cases and lab report into a `zip`.

You are recommended to arrange your submission like：

```
2023114514wangdch12023.zip
├── report.pdf # Your report
├── source_code # Directory of your source codes
|    └─── ... # Make sure to not include .git/ and build/
|
└── test_program
     └─── ...
```

Hand in your zip at Gradescope.

All hand-in requirements enforced to previous lab(s) are applicable to this lab. Still, for each student, we will evaluate only the latest submission before the deadline. We will not evaluate any submission that passes the deadline.