Project 2 -- Cache Prefetch Simulator

This is an *individual* project. You may only collaborate with your classmates according to the CS Collaboration Policy. Plagiarism will be punished severely.

Learning Objectives

- Expand the set-associative cache system from Project 1 to include prefetching functionality.
- Solidify your understanding of cache prefetch strategies.
- Implement a custom prefetcher using techniques found in academic literature.

Overview

In this project, you expand upon the cache simulator from Project 1 by adding three prefetchers to improve cache performance. Two of the prefetchers have predefined functionality, but you will also implement a custom best-effort prefetcher which implements a prefetch strategy of your choosing.

All the requirements from Project 1 still apply (although, only the LRU replacement policy will be used).

It is ***highly recommended*** that you develop this project on a Linux system. If you choose to use a different type of system (such as macOS or Windows), the instructors and TAs will not be able to assist you with any platform-specific issues.

Rubric

This project is worth **100** points, distributed as follows:

#	Item	Points
1.1	Adjacent Prefetch*	15
1.2	Sequential Prefetch*	25
2.1	Custom Prefetch	20
2.2	Custom Prefetch Documentation	25
2.3	Custom Prefetch Compared to Other Students' Implementations	15
	TOTAL	**100**

* indicates that the rubric item is auto-graded

NOTE: Since your submission will be partially auto-graded on Gradescope, you will be only be allowed a total of three submissions. Be sure to test your code before submitting!

Prefetcher Functionality

All the requirements from Project 1 still apply. However, after updating the replacement policy state, the prefetcher will be called. The prefetcher should prefetch cache lines according to the specified strategy (see below).

Note that for all of the test cases, **only the LRU policy will be used**.

Prefetch Strategies

You must implement three prefetch strategies.

- `NULL` --- this is already implemented for you in the starter code but included here for clarity. For this strategy, no prefetching will be done.
- `ADJACENT` (item 1.1 in the rubric) --- For this strategy, whenever a cache line is accessed, the next cache line should be prefetched.
- `SEQUENTIAL` (item 1.2 in the rubric) --- For this strategy, whenever a cache line is accessed, the next `N` lines should be prefetched. (`N` will be passed as a command line parameter. See the Full Requirements below for details about the command line arguments.)
- `CUSTOM` (item 2 in the rubric) --- For this strategy, you must implement your best attempt at a prefetcher.

Custom Prefetcher

Your custom prefetcher can do anything that you want as long as it could conceivably be implemented in hardware. An example of a prefetcher that could *not* be implemented in hardware would be a "god mode" prefetcher which looks ahead to future memory accesses to perfectly prefetch the next needed cache lines. (Rubric item 2.1)

Before implementing your custom prefetcher, please read the following papers and articles for ideas of how to implement your prefetcher:

- https://www.cs.utah.edu/~rajeev/cs7810/papers/chen95.pdf
- http://www.cc.gatech.edu/~hyesoon/lee_taco12.pdf
- https://software.intel.com/content/www/us/en/develop/articles/disclosure-of-hw-prefetcher-control-on-some-intel-processors.html

In addition to implementing your prefetcher, you must also provide documentation for your custom prefetch strategy (rubric item 2.2). Your documentation should be in a file in the root of the submission TAR file called `PREFETCHER.md`. The documentation must:

1. Describe how your prefetcher works.
2. Explain how you chose that prefetch strategy.
3. Discuss the pros and cons of your prefetch strategy.
4. Demonstrate that the prefetcher could be implemented in hardware (this can be as simple as pointing to an existing hardware prefetcher using the strategy or a paper describing a hypothetical hardware prefetcher which implements your strategy).
5. Cite any additional sources that you used to develop your prefetcher.

Your documentation should be the minimal documentation required to fully communicate each the above points. **Documentation with superfluous information will be penalized at the graders discretion.**

Your prefetcher will be tested using the LRU replacement policy. The grader will use a variety of input files and LRU parameters to determine the efficacy of your prefetcher. Your grade for rubric item 2.3 will be assigned according to how well your prefetcher does compared to other students in the class. The metrics

that will be used to compare your submission to other students in the class include: hit rate, compulsory misses, conflict misses, and prefetches.

Starter Code Overview

See the Project1-Introduction.

Full Requirements

As with the starter code for Project 1, the following requirements are automatically fulfilled by the starter code (assuming correct usage). They are included so that if you choose to write your simulator without using the starter code, your submission will be able to be graded.

Only things which have changed from Project 1 are mentioned here.

Submission Format

The `PREFETCHER.md` file must be in the root of the TAR archive.

Inputs

Your program must accept two new positional command line arguments after the existing command line arguments:

- **Prefetch strategy**: this will be one of the following: `NULL`, `ADJACENT`, `SEQUENTIAL`, or `CUSTOM` representing the prefetch strategy.
- **Prefetch amount**: this will be an integer representing `N`, the number of additional cache lines to prefetch (this parameter is only used for the `SEQUENTIAL` strategy and the `CUSTOM` strategy if you choose to make your strategy depend on `N`).

Example execution with LRU replacement policy, cache size of 32 KiB, 2048 cache lines, 4-way associativity, the SEQUENTIAL prefetcher, a prefetch amount of 2, and passing the contents of the `./inputs/trace1` file via `stdin`.

```
$ ./cachesim LRU 32768 2048 4 SEQUENTIAL 2 < ./inputs/trace1
```

Output

An additional three lines of statistics output are now required right after the `MISSES` line.

- Prefetches (int): the number of cache lines which were prefetched
- Compulsory Misses (int): the number of misses which were compulsory.
- Conflict Misses (int): the number of misses which were conflict misses.

Example output with the new lines:

```
OUTPUT ACCESSES 496611
OUTPUT HITS 494428
```

```
OUTPUT MISSES 2183
OUTPUT PREFETCHES 993222
OUTPUT COMPULSORY MISSES 1788
OUTPUT CONFLICT MISSES 395
OUTPUT DIRTY EVICTIONS 1919
OUTPUT HIT RATIO 0.99560421
```

Other Instructors

If you have any questions regarding this project, please contact the instructor or the teaching assistants.