Project 1. Cache Design

Due 23:59, Thursday, March 28th, 2024 (KST)

TA: cs510_ta@casys.kaist.ac.kr

1. Introduction

For the first project of CS510, your task is to develop a simulator of 2-level caching system with write/replacement policies and prefetching. This simulator should read traces of memory access instructions from various applications, perform the simulation, and produce statistics regarding caching behaviors.

Please read the descriptions below carefully and complete the missing components of the given framework. If you have any questions, feel free to ask TA via email.

2. Explanation of Provided Framework

We are providing you with a framework to build the cache simulator. You must fill in the following functions in the framework, defined in `cachesim.cpp` :

```
void setup_cache(uint64_t c1, uint64_t b1, uint64_t s1, uint64_t c2, uint64_t b2, uint64_t s2, uint32_t k);
```

Subroutine for initializing the cache. You may add and initialize any global or heap variables as needed. See [4. Specification of Simulator] for explanation of arguments.

```
void cache_access(char rw, uint64_t address, cache_stats_t* p_stats);
```

Subroutine that simulates the cache one trace event at a time. Type can be either READ or WRITE, which is each defined in `cachesim.hpp` . A READ event is a memory load operation of 1 byte to the address specified in arg. A WRITE event is a memory store operation of 1 byte to the address specified in arg.

```
void complete_cache(cache_stats_t *p_stats);
```

Subroutine for cleaning up memory and calculating overall system statistics such as miss rate or average access time.

You are only allowed to modify `cachesim.cpp` and `cachesim.hpp` .

3. Simulator Options

Build

```
$ make
```

Basic command

```
$ ./cachesim [OPTIONS] < traces/[file.trace]
```

Options

- -c : Total size of L1 (2^{C1} bytes)
- -b : Size of each block in L1 (2^{B1} bytes)
- -s : Number of blocks per set in L1 (2^{S1})
- -C : Total size of L2 (2^{C2} bytes)
- -B : Size of each block in L2 (2^{B2} bytes)
- -S : Number of blocks per set in L2 (2^{S2})
- -k : Number of prefetch blocks
- -h : Print help message

Tips

We provide output and log files that would be helpful for your development. The output files show the simulator configurations and the cache statistics result that your simulator should generate. If your simulator produces identical results to the output files, it is implemented perfectly. The log files list how cache system behaves for each memory instruction. It would be of great help in debugging your simulator.

4. Specification of Simulator

1. The simulator should model a 2-level caching system with 2^{C1} and 2^{C2} bytes of data storage, having 2^{B1}-byte and 2^{B2}-byte blocks, and with sets of 2^{S1} blocks per set and 2^{S2} blocks per set, in level 1 and level 2 respectively (note that $S = 0$ is a direct-mapped cache, and $S = C - B$ is a fully associative cache). Also note that the values are restricted to $C2 \geq C1$, $B2 \geq B1$ and $S2 \geq S1$.
2. The memory addresses are 64-bit addresses.
3. Caches are byte-addressable.
4. Each cache implements the write-back, write-allocate (WBWA) policy. There is an additional *dirty bit* for each tag in the tag store.
5. Each cache implements least-recently-used (LRU) as the replacement policy. The LRU cache chooses the least recently used block for replacement.
   - If there exist multiple LRU blocks, you can evict the first oldest block when traversing the cache entries from the beginning.
6. There is 1 valid bit per block of storage overhead required. The valid bits are set to 0 when the simulation begins.
7. The following are used to calculate AAT, hit time, and miss penalty for L1 and L2 (HT means Hit Time, MR means Miss Rate, and MP means Miss Penalty):
   - $AAT = HT1 + MR1 \times MP1$
     - $HT1 = 2 + 0.2 \times S1$
     - $MP1 = AAT$ for $L2 = HT2 + MR2 \times MP2$
     - $HT2 = 4 + 0.4 \times S2$
     - $MP2 = 500$
8. L2 cache performs stride prefetching to reduce compulsory misses. The prefetching works as the following when L2 misses:
   - Define the function `Block_Addr(X) = address of X with offsets bits of X to zero` .
   - On a L2 miss to address `X` , calculate `d = Block_Addr(x) - Last_Miss_Block_Addr` . Then set `Last_Miss_Block_Addr = Block_Addr(x)` .
   - If `d == Pending_Stride` then prefetching the next K blocks. This means that we bring blocks with block addresses $X + i \times \text{Pending\_Stride}$, for $i = 1 \cdots (K)$ from memory to L2 cache. K is given as a simulation parameter.
   - Note that prefetch does not contribute to AAT. Any prefetch is not included in the calculation of the miss rate of L2.
   - A prefetched block becomes an LRU (least recently used) block in its set. *(Hint: you can make the timestamp of the prefetched block equal to `min(timestamps of all other blocks in the set) - 1` .)*
   - Finally, regardless of whether `d` matches `Pending_Stride` or not, set `Pending_Stride = d` .
   - Vary K in the range [0, 4].
9. The entire cache system uses a partially-inclusive cache policy.
   - When a cache block is read from the main memory, it should be loaded in both the L1 and L2 caches.
   - Evicting a block from the L2 cache does not require evicting a corresponding block from the L1 cache.
   - When evicting an L2 block, you can check whether the block is dirty or not and write back the block if it is dirty. When evicting an L1 block, you may write back L1 block if it is dirty and there's no L2 block that includes the L1 block. If there's an L2 block that includes the L1 block, you just need to set the dirty bit of the L2 block and not write back the L1 block.
10. In general, $(C1, B1, B2, C2, B2, S2, K)$ completely specifies the caching system.

5. Cache Statistics (output)

The simulator must output the following statistics after completion of the run:

- Total number of accesses
- Number of accesses to L1 (either hit or miss)
- Number of accesses to L2 (either hit or miss)
- Number of reads
- Number of read misses to L1
- Number of read misses to L2
- Number of writes
- Number of write misses to L1
- Number of write misses to L2
- Number of write backs to main memory
- Number of prefetched blocks
- Number of cache misses reduced by prefetching (count cache hits on prefetched blocks that are not accessed yet)
- The average access time (AAT)

The output of these variables should be handled by the driver code (`cachesim_driver.cpp`), and you only need to fill in the structure `cache_statistics_t` , defined in `cachesim.hpp` .

6. Grading Policy

Your code will be evaluated using four memory traces provided to you, and will be graded based on its performance with both the default configurations defined in `cachesim.hpp` , as well as a hidden configuration that has not been announced.

Your grade will be determined based on the following criteria:

- **100%**: Produces the same result as the correct answer for all configs.
- **95%**: Produces the same result as the correct answer for default config, but not for the hidden config.
- **85%**: Produces a result different from the correct answer, but with an average error of under **5%**.
- **50%**: Produces a result different from the correct answer, with an average error of over **5%**.
- **0%**: Code is not compilable or does not produces any result.

7. Submission

Zip your project directory and name it with **your student id**.

```
$ tar -cvf [student id].tar cachesim.cpp cachesim.hpp
```

or

```
$ make submit
```

Submit the tar file to KLMS.

Please make sure that your code compiles and runs well. We will give 0 points to the code that does not compile without any exception.

gcc / g++ version = 7.5.0 / c++11

8. Late Policy & Plagiarism Penalty

You will lose **30%** of your score on the **first day** (March 29th 0:00 ~ 23:59). We will **not accept** any works that are submitted after then.

Be aware of plagiarism! Although it is encouraged to discuss with others and refer to extra materials, **copying other students' or opened code is strictly banned.**

TA will compare your source code with some open-source codes and other students' code. If you are caught, you will receive a serious penalty for plagiarism as announced in the first lecture of this course.

If you have any requests or questions regarding administrative issues (such as late submission due to an unfortunate accident), please send an e-mail to TA.