Cache Lab report

StudentID = 5140309320, name = "Bill Cai", Email = caiwanxin2009@hotmail.com

Task 1 : Simulate a Cache

observation

- The address is a 8 byte word, so in order to save it, we need a long long.
- The length of a memory access wouldn't exceed a block, so there's no need to do an extra check for another block.

  Actually it's necessary for most of the caches in the computer, for saving a valid data in multiple blocks may increase chance of being evicted, thus the conflict miss will increase.
- the data is not available in the trace file, so our simulation needn't create a space for data.

The structure of my solution goes like this

- initialize (read the argument of a cache and the trace file)

- simulate (check and fill the cache step by step)
- print (call the print function to summarize the performance of a cache)
- clean up (delete the allocated data)

As the header file is not permitted in the task, I can only put all the structures and functions in the main functions. But I've add some comment in the code in order to separate the main functions and the data structure.

The data structure cache goes like this :

attributes

- hit times,miss times,evict times
- cache line(include the information of validation, replacing condition, saved tags)
- way , set Bits, block Bits (to illustrate the type of a cache)

methods

- new (construct a specific cache simulator)
- delete (destruct the cache simulator)

- pseudo LRU,pseudo update (the pseudo LRU algorithm which allocates o(n) space)
- true LRU, true update (the real LRU algorithm which allocates o (n log n) space)
- try access (simulate accessing an address)

And the main function is the implement of the solution structure.

If you are interested, you can check my pseudo LRU policy functions, for its performance is acceptable compared to the true LRU policy.

Pseudo LRU

```
+             Your simulator         Reference simulator
(s,E,b)    Hits  Misses  Evicts    Hits  Misses  Evicts
(1,1,1)       9       8       6       9       8       6  traces/yi2.trace
(4,2,4)       4       5       2       4       5       2  traces/yi.trace
(2,1,4)       2       3       1       2       3       1  traces/dave.trace
(2,1,3)     167      71      67     167      71      67  traces/trans.trace
(2,2,3)     189      49      41     201      37      29  traces/trans.trace
(2,4,3)     205      33      19     212      26      10  traces/trans.trace
(5,1,5)     231       7       0     231       7       0  traces/trans.trace
(5,1,5)  265189   21775   21743  265189   21775   21743  traces/long.trace
```

True LRU

```
+                       Your simulator       Reference simulator
Points  (s,E,b)    Hits  Misses  Evicts    Hits  Misses  Evicts
3       (1,1,1)       9       8       6       9       8       6  traces/yi2.trace
3       (4,2,4)       4       5       2       4       5       2  traces/yi.trace
3       (2,1,4)       2       3       1       2       3       1  traces/dave.trace
3       (2,1,3)     167      71      67     167      71      67  traces/trans.trace
3       (2,2,3)     201      37      29     201      37      29  traces/trans.trace
3       (2,4,3)     212      26      10     212      26      10  traces/trans.trace
3       (5,1,5)     231       7       0     231       7       0  traces/trans.trace
6       (5,1,5)  265189   21775   21743  265189   21775   21743  traces/long.trace
27
```

Task 2 : Optimizing loop iteration

Observation

- separating the matrix in to smaller k X k matrix can decrease the miss rate and is optimized for this task.
- the block size is 32 bytes, that is to say we can at most carry eight integers into the memory. So the compulsory miss rate is 1/8.

  In this task 32 X 32 matrix will cause at least 256 misses and 64 X 64 will cause at least 1024 misses. Which means the conflict misses should be enormously eliminated.

- the cache has 32 lines but an optimized 8 X 8 matrix only fills 8 lines, you can fill 4 matrices like this type, which means you can take advantage of this property to eliminate the conflict miss.
- The best way to solve this problem is to divide and conquer, our promising matrix size is 8 X 8,so we should focus on transposing the 8 X 8 matrices.

  However, in 64 X 64 matrices, the **a[i][j]** will conflict to **a[i + 4][j]** in the cache. it's a tough question needs to be solved, transposing smaller matrices like 4 X 4 may not be a good idea, for the compulsory miss rate will increase to 1/4, so we won't try such method.

Idea

notice that there are so many empty block in the cache which can be used. Thus, to transpose a 8 X 8 matrix in **a** we can temporarily move(copy) the upper 4 X 8 blocks and lower 4 X 8 blocks from **a** to two unused 4 X 8 blocks in **b** and these two blocks should satisfy three properties

- they will not conflict to the source 8 X 8 block in **a** .
- they will not conflict to the target 8 X 8 block in **b** .
- they will not conflict to each other.

(conflict means they have the same index in cache)

After that, we can transpose the two blocks to the target 8 X 8 block to **b**, as these two blocks are already in the cache, we can do any arbitrary type transpose, which won't cause misses. so we can transpose the head 4 X 4 of these 2 blocks to the upper 4 X 8 blocks of the target block, and then transpose the tail 4 X 4 to the lower 4 X 8 blocks of the target block.

The conflict miss will decrease but it will provide 1/2 extra movements. which means the miss rate was about 3/8, still not meet the needs.

Improvement

As we are dividing and conquering the problem. it's possible that target block in **b** may have a sequential move, which means each target block we are doing will not conflict to the next target block. then we can choose the temporary unused data in the next target block and the one after it. It will not be evicted, because we will immediately work on it after we solve the current problem.So we can consider this temporary movements as the prefetch of the next target block. So, it will decrease the miss rate to almost 1/8, except for last few blocks which we are not able to find the unused block in b.

Result

```
+                          Points   Max pts      Misses
Trans perf 32x32            8.0         8          292
Trans perf 64x64            8.0         8         1165
Trans perf 61x67           10.0        10         1992
```

notice that 32 X 32 may have slightly different optimization, but the idea is similar, so I didn't do it. And 61 X 67 can be passed by a pretty simple optimization, I am also not seeking for a better solution.

Final Result

```
Cache Lab summary:        Points   Max pts      Misses
Csim correctness           27.0        27
Trans perf 32x32            8.0         8          292
Trans perf 64x64            8.0         8         1165
Trans perf 61x67           10.0        10         1992
Total points               53.0        53
```