Computer Organization

0716214 江岳勳

0716222 黃偉傑

Detailed description of the implementation:

- direct_mapped_cache

  程式在啟動的時候會先將 LRU.txt 中的資料先讀取進來，然後按照各種不同的 cache size/associativity 組合來執行測試。

  1. 用 ifstream 讀檔案，經過 std::hex() 之後輸入會從字串被轉成整數，再用 vector<unsigned int> 作為 container 儲存
  2. "memory_access()" 這個 function 可以輸入 cache type, cache size, and block size ，所以在 main function 裡面可以一次運行所有 required combinations
  3. 在"memory_access()" 這個 function 裡面，用一個 defined structure 的 vector 來模擬 direct mapped 中的 table，每一筆資料記錄 valid bit, tag, and index，並且以預先計算好的 number of blocks 來設定大小。提出每一筆輸入時，先把 offset 去掉，再比對 index，若相符的資料為 invalid (即 compulsory miss)的情形，就直接更新資料進去並且設為 valid；如果是 valid 的資料，就比對 tag 來確定有沒有 miss。 Miss handling 在這裡因為只有儲存 memory address，所以只做了更新 tag
  4. 每做完一種(cache type, cache size, block size)的組合後，會呼叫 print function，把記錄的 hit, miss 次數算出需求並輸出

- set_assoiciative_cache

  程式在啟動的時候會先將 LRU.txt 中的資料先讀取進來，然後按照各種不同的 cache size/associativity 組合來執行測試。

  測試的一開始會先根據 cache size 和 associativity 算出 block 和 set 的數量，使用一個 vector<vector<unsigned int>>來模擬 set associative cache，內層的 vector 代表一個一個的 set，新 touch 過的位址會被 push 到尾端，這會讓 LRU 永遠在 vector 的頭部。接著便從預先讀好的 data 中拿記憶體位址來做處理。

讀出來的記憶體位址要先除以 block size，轉換成 block address，然後再用 set_count 對 block address 分別取餘和商，得到該 block address 的 set 編號及 tag。接下來分為三種情況(從上至下 if、else if、else)：

1. cache[set_idx]包含 tag，這樣的話代表該 block 已經在 cache 中了，將 tag 從 vector erase 掉，重新 push_back 到尾端。**(HIT + 1)**
2. cache[set_idx]大小未滿 associativity，代表該 block 不在 cache 中，且該 set 仍有 invalid 的空白區域，直接將 tag push_back 進去。**(MISS + 1)**
3. 以上皆非的話意味著要將 LRU 給替換成要寫入的 tag，將 vector 的頭部元素 erase 掉，再將 tag push_back 進去。**(MISS + 1)**

模擬完後只要將 hit rate 和 miss rate 算出輸出便結束了。

Implementation results:

➢ direct_mapped_cache (**Output Result 在報告最後面**)

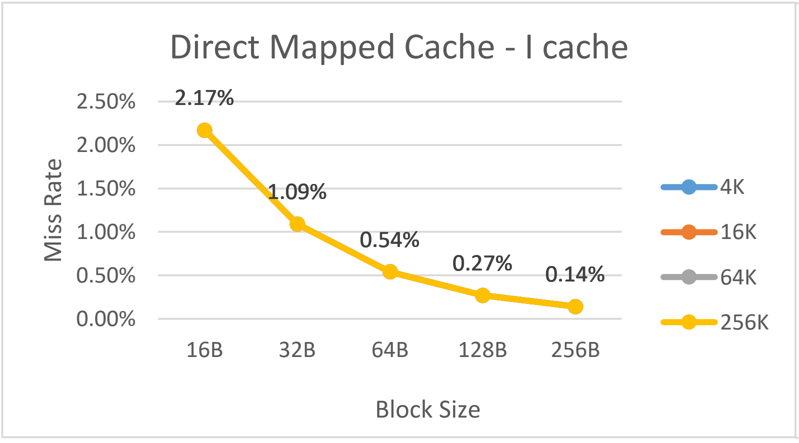
以下分別是 I-cache 與 D-cache 下，miss rate 在不同 Cache Size 和 Block Size 組合下的數值：

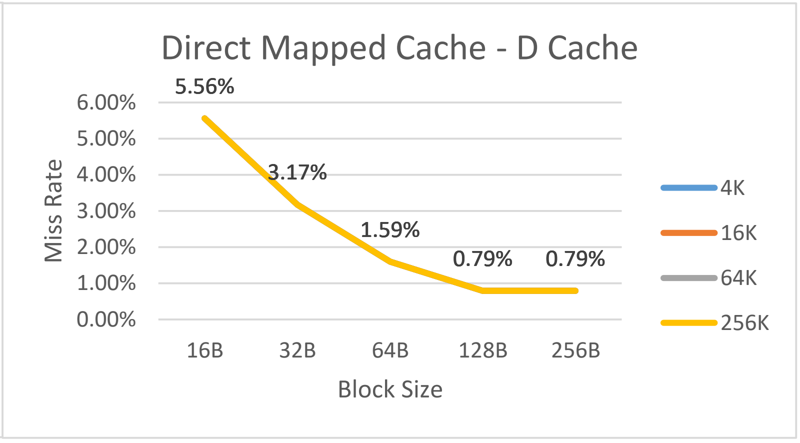
✓ I-cache

	16B	32B	64B	128B	256B
4K	2.17%	1.09%	0.54%	0.27%	0.14%
16K	2.17%	1.09%	0.54%	0.27%	0.14%
64K	2.17%	1.09%	0.54%	0.27%	0.14%
256K	2.17%	1.09%	0.54%	0.27%	0.14%

✓ D-Cache

	16B	32B	64B	128B	256B
4K	5.56%	3.17%	1.59%	0.79%	0.79%
16K	5.56%	3.17%	1.59%	0.79%	0.79%
64K	5.56%	3.17%	1.59%	0.79%	0.79%
256K	5.56%	3.17%	1.59%	0.79%	0.79%

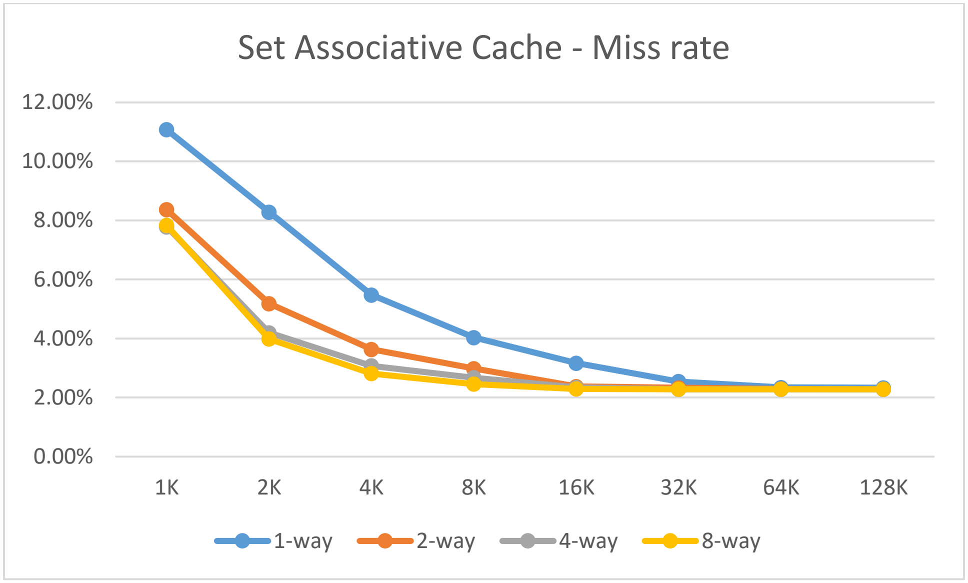




➢ set_assoiciative_cache (Output Result 在報告最後面)

以下是 miss rate 在不同 cache size 和 associativity 組合下的數值：

	1K	2K	4K	8K	16K	32K	64K	128K
1-way	11.07%	8.28%	5.47%	4.03%	3.16%	2.54%	2.34%	2.33%
2-way	8.36%	5.18%	3.63%	2.98%	2.37%	2.33%	2.29%	2.28%
4-way	7.78%	4.19%	3.07%	2.67%	2.34%	2.28%	2.28%	2.28%
8-way	7.83%	3.98%	2.81%	2.45%	2.29%	2.28%	2.28%	2.28%



從圖表中可以發現，miss rate 隨著 cache size 和 associativity 的變大而降低，而且有出現趨近一個定值的情況。其中比較有趣的是 1K-8way 的 miss rate 反而比 1K-4way 的 miss rate 高了一些，我推測應該是由於測資的量不夠大，剛好有一些巧合讓 8way 反而比 4way 的 miss rate 還大。

Problems encountered and solutions:

➢ direct_mapped_cache

一開始最花心思處理的是輸入，原本的做法是將輸入的 Hex number 轉成 binary，然後再根據上課教的做法炮製，用 Log2()算 tag, index 需要幾個 digits，然後再用 substring 的函式去擷取，但是後來覺得自己造輪子實在太傻了，發現了 std::hex()之後就輕易搞定了。

後來發生過一個 bug 是忘記做轉型(cast)，結果 miss rate 怎麼算都是 0，後來加上 cast 就解決了。

- set_assoiciative_cache

  在一開始讀檔的時候，我使用 ifstream 搭配 std::hex 來處理十六進位的資料，卻發現一直失敗，打開 LRU.txt 手動模擬了一下後才發現裡面有會讓 signed int overflow 的數字，將 code 內的 int 換成 unsigned int 後便解決了。

  模擬 set associative cache 的部分則沒有什麼問題。

Comment:

0716214 江岳勳：

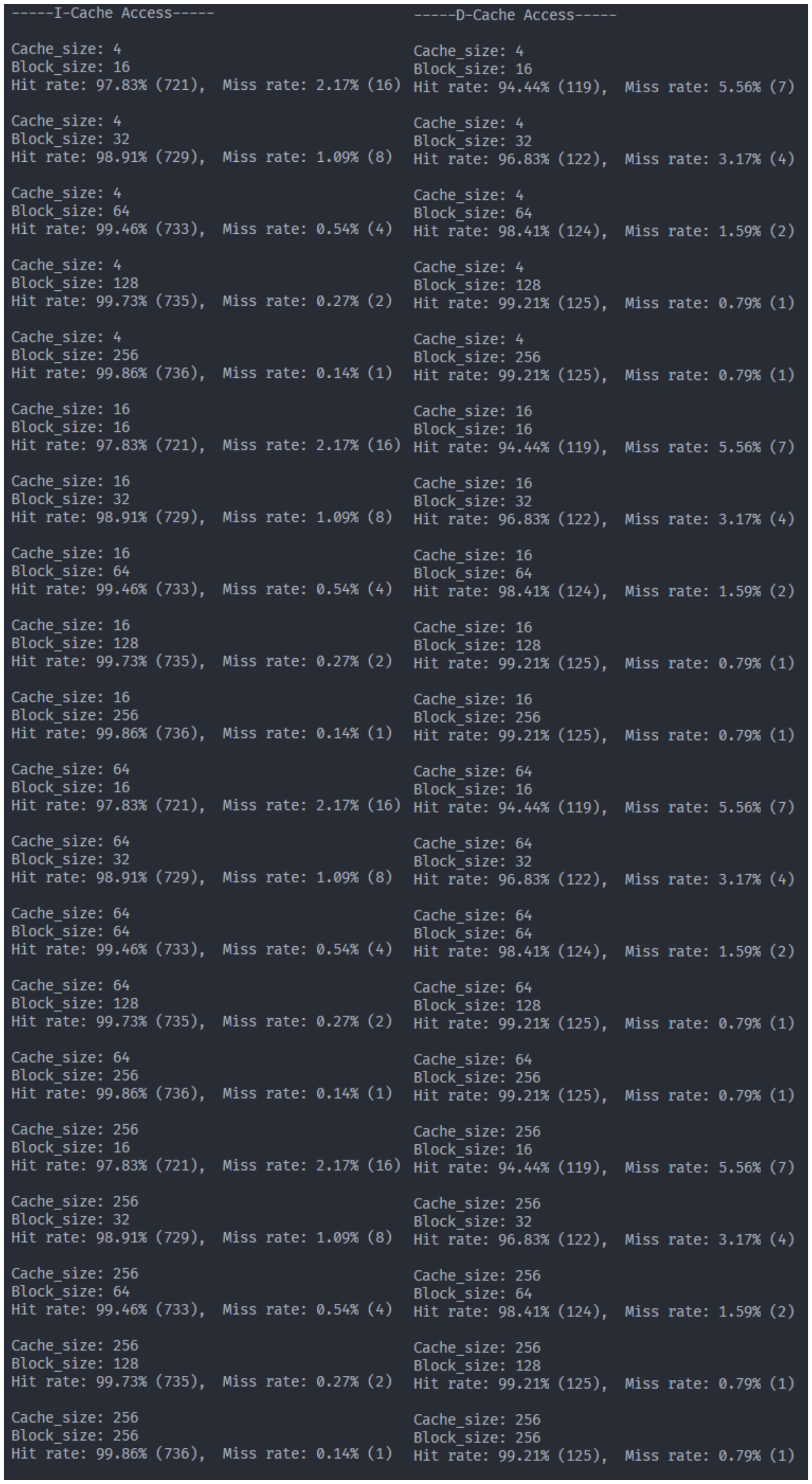
本次我負責 set associative 的部分，實作過程中剛好在複習了一次整個第五章有關 cache 的部分，期末考前在複習時就有發現我常常會把 byte address、block address、tag、index 之類的名詞搞混在一起，經過這次作業有確實感覺到自己整個搞懂這個部份了。

0716222 黃偉傑：

寫 C++真的是比 Verilog 快樂多了......同樣的東西如果用 Verilog 來刻可能會很崩潰，感謝助教。不過這次作業帶給我最大的影響應該是實際實驗過 Direct Mapped 的 Miss Rate 有多麼差勁，如果存取都像給定的 Data 一樣一直讀小範圍的 block 的話，硬體加多大真的都沒什麼用。

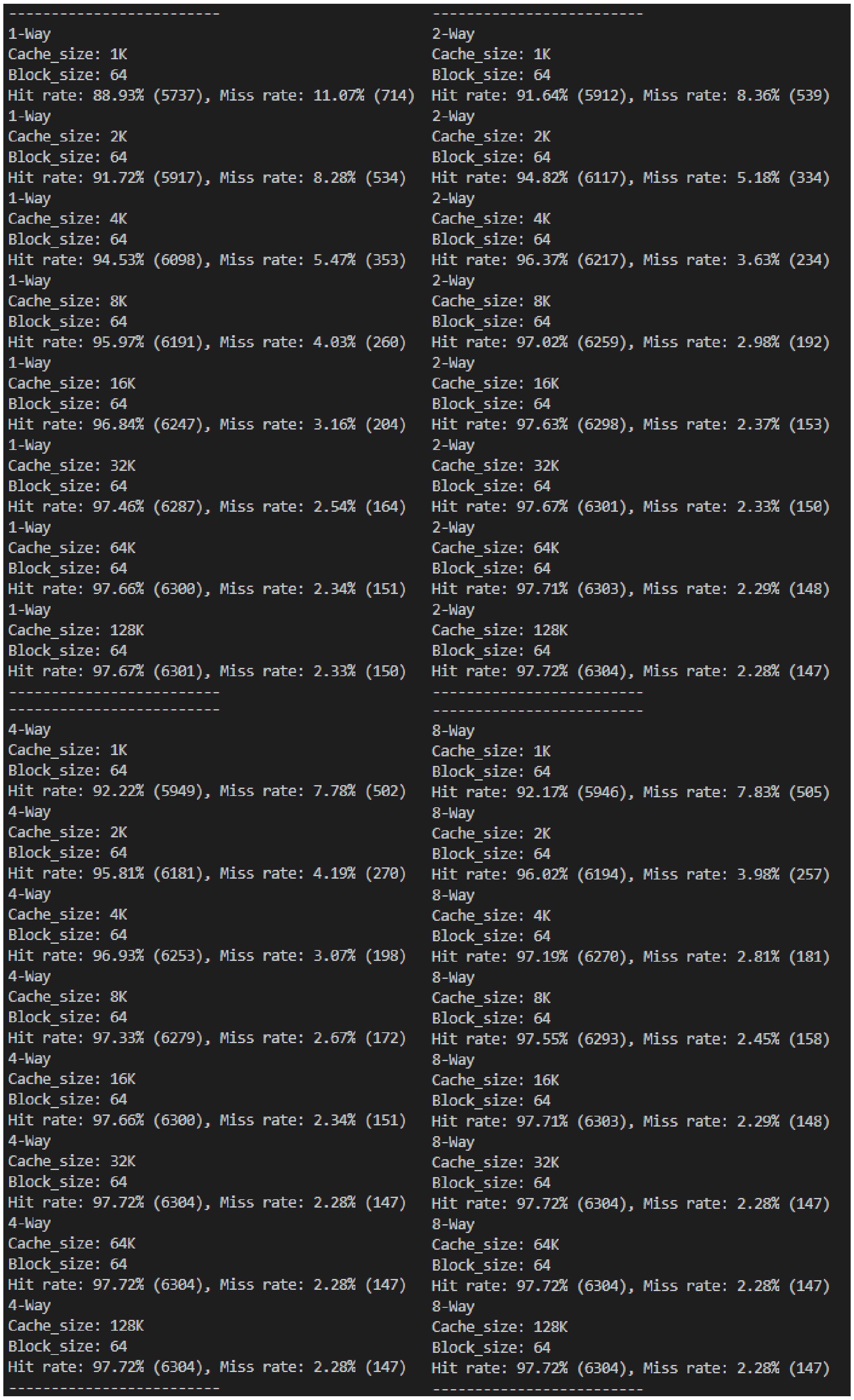
Implementation Result (Output Screenshots):

- direct_mapped_cache



```
-----I-Cache Access-----                         -----D-Cache Access-----

Cache_size: 4                                    Cache_size: 4
Block_size: 16                                   Block_size: 16
Hit rate: 97.83% (721),  Miss rate: 2.17% (16)   Hit rate: 94.44% (119),  Miss rate: 5.56% (7)

Cache_size: 4                                    Cache_size: 4
Block_size: 32                                   Block_size: 32
Hit rate: 98.91% (729),  Miss rate: 1.09% (8)    Hit rate: 96.83% (122),  Miss rate: 3.17% (4)

Cache_size: 4                                    Cache_size: 4
Block_size: 64                                   Block_size: 64
Hit rate: 99.46% (733),  Miss rate: 0.54% (4)    Hit rate: 98.41% (124),  Miss rate: 1.59% (2)

Cache_size: 4                                    Cache_size: 4
Block_size: 128                                  Block_size: 128
Hit rate: 99.73% (735),  Miss rate: 0.27% (2)    Hit rate: 99.21% (125),  Miss rate: 0.79% (1)

Cache_size: 4                                    Cache_size: 4
Block_size: 256                                  Block_size: 256
Hit rate: 99.86% (736),  Miss rate: 0.14% (1)    Hit rate: 99.21% (125),  Miss rate: 0.79% (1)

Cache_size: 16                                   Cache_size: 16
Block_size: 16                                   Block_size: 16
Hit rate: 97.83% (721),  Miss rate: 2.17% (16)   Hit rate: 94.44% (119),  Miss rate: 5.56% (7)

Cache_size: 16                                   Cache_size: 16
Block_size: 32                                   Block_size: 32
Hit rate: 98.91% (729),  Miss rate: 1.09% (8)    Hit rate: 96.83% (122),  Miss rate: 3.17% (4)

Cache_size: 16                                   Cache_size: 16
Block_size: 64                                   Block_size: 64
Hit rate: 99.46% (733),  Miss rate: 0.54% (4)    Hit rate: 98.41% (124),  Miss rate: 1.59% (2)

Cache_size: 16                                   Cache_size: 16
Block_size: 128                                  Block_size: 128
Hit rate: 99.73% (735),  Miss rate: 0.27% (2)    Hit rate: 99.21% (125),  Miss rate: 0.79% (1)

Cache_size: 16                                   Cache_size: 16
Block_size: 256                                  Block_size: 256
Hit rate: 99.86% (736),  Miss rate: 0.14% (1)    Hit rate: 99.21% (125),  Miss rate: 0.79% (1)

Cache_size: 64                                   Cache_size: 64
Block_size: 16                                   Block_size: 16
Hit rate: 97.83% (721),  Miss rate: 2.17% (16)   Hit rate: 94.44% (119),  Miss rate: 5.56% (7)

Cache_size: 64                                   Cache_size: 64
Block_size: 32                                   Block_size: 32
Hit rate: 98.91% (729),  Miss rate: 1.09% (8)    Hit rate: 96.83% (122),  Miss rate: 3.17% (4)

Cache_size: 64                                   Cache_size: 64
Block_size: 64                                   Block_size: 64
Hit rate: 99.46% (733),  Miss rate: 0.54% (4)    Hit rate: 98.41% (124),  Miss rate: 1.59% (2)

Cache_size: 64                                   Cache_size: 64
Block_size: 128                                  Block_size: 128
Hit rate: 99.73% (735),  Miss rate: 0.27% (2)    Hit rate: 99.21% (125),  Miss rate: 0.79% (1)

Cache_size: 64                                   Cache_size: 64
Block_size: 256                                  Block_size: 256
Hit rate: 99.86% (736),  Miss rate: 0.14% (1)    Hit rate: 99.21% (125),  Miss rate: 0.79% (1)

Cache_size: 256                                  Cache_size: 256
Block_size: 16                                   Block_size: 16
Hit rate: 97.83% (721),  Miss rate: 2.17% (16)   Hit rate: 94.44% (119),  Miss rate: 5.56% (7)

Cache_size: 256                                  Cache_size: 256
Block_size: 32                                   Block_size: 32
Hit rate: 98.91% (729),  Miss rate: 1.09% (8)    Hit rate: 96.83% (122),  Miss rate: 3.17% (4)

Cache_size: 256                                  Cache_size: 256
Block_size: 64                                   Block_size: 64
Hit rate: 99.46% (733),  Miss rate: 0.54% (4)    Hit rate: 98.41% (124),  Miss rate: 1.59% (2)

Cache_size: 256                                  Cache_size: 256
Block_size: 128                                  Block_size: 128
Hit rate: 99.73% (735),  Miss rate: 0.27% (2)    Hit rate: 99.21% (125),  Miss rate: 0.79% (1)

Cache_size: 256                                  Cache_size: 256
Block_size: 256                                  Block_size: 256
Hit rate: 99.86% (736),  Miss rate: 0.14% (1)    Hit rate: 99.21% (125),  Miss rate: 0.79% (1)
```

- set_associative_cache



```
------------------------                          ------------------------
1-Way                                             2-Way
Cache_size: 1K                                    Cache_size: 1K
Block_size: 64                                    Block_size: 64
Hit rate: 88.93% (5737), Miss rate: 11.07% (714)  Hit rate: 91.64% (5912), Miss rate: 8.36% (539)
1-Way                                             2-Way
Cache_size: 2K                                    Cache_size: 2K
Block_size: 64                                    Block_size: 64
Hit rate: 91.72% (5917), Miss rate: 8.28% (534)   Hit rate: 94.82% (6117), Miss rate: 5.18% (334)
1-Way                                             2-Way
Cache_size: 4K                                    Cache_size: 4K
Block_size: 64                                    Block_size: 64
Hit rate: 94.53% (6098), Miss rate: 5.47% (353)   Hit rate: 96.37% (6217), Miss rate: 3.63% (234)
1-Way                                             2-Way
Cache_size: 8K                                    Cache_size: 8K
Block_size: 64                                    Block_size: 64
Hit rate: 95.97% (6191), Miss rate: 4.03% (260)   Hit rate: 97.02% (6259), Miss rate: 2.98% (192)
1-Way                                             2-Way
Cache_size: 16K                                   Cache_size: 16K
Block_size: 64                                    Block_size: 64
Hit rate: 96.84% (6247), Miss rate: 3.16% (204)   Hit rate: 97.63% (6298), Miss rate: 2.37% (153)
1-Way                                             2-Way
Cache_size: 32K                                   Cache_size: 32K
Block_size: 64                                    Block_size: 64
Hit rate: 97.46% (6287), Miss rate: 2.54% (164)   Hit rate: 97.67% (6301), Miss rate: 2.33% (150)
1-Way                                             2-Way
Cache_size: 64K                                   Cache_size: 64K
Block_size: 64                                    Block_size: 64
Hit rate: 97.66% (6300), Miss rate: 2.34% (151)   Hit rate: 97.71% (6303), Miss rate: 2.29% (148)
1-Way                                             2-Way
Cache_size: 128K                                  Cache_size: 128K
Block_size: 64                                    Block_size: 64
Hit rate: 97.67% (6301), Miss rate: 2.33% (150)   Hit rate: 97.72% (6304), Miss rate: 2.28% (147)
------------------------                          ------------------------
------------------------                          ------------------------
4-Way                                             8-Way
Cache_size: 1K                                    Cache_size: 1K
Block_size: 64                                    Block_size: 64
Hit rate: 92.22% (5949), Miss rate: 7.78% (502)   Hit rate: 92.17% (5946), Miss rate: 7.83% (505)
4-Way                                             8-Way
Cache_size: 2K                                    Cache_size: 2K
Block_size: 64                                    Block_size: 64
Hit rate: 95.81% (6181), Miss rate: 4.19% (270)   Hit rate: 96.02% (6194), Miss rate: 3.98% (257)
4-Way                                             8-Way
Cache_size: 4K                                    Cache_size: 4K
Block_size: 64                                    Block_size: 64
Hit rate: 96.93% (6253), Miss rate: 3.07% (198)   Hit rate: 97.19% (6270), Miss rate: 2.81% (181)
4-Way                                             8-Way
Cache_size: 8K                                    Cache_size: 8K
Block_size: 64                                    Block_size: 64
Hit rate: 97.33% (6279), Miss rate: 2.67% (172)   Hit rate: 97.55% (6293), Miss rate: 2.45% (158)
4-Way                                             8-Way
Cache_size: 16K                                   Cache_size: 16K
Block_size: 64                                    Block_size: 64
Hit rate: 97.66% (6300), Miss rate: 2.34% (151)   Hit rate: 97.71% (6303), Miss rate: 2.29% (148)
4-Way                                             8-Way
Cache_size: 32K                                   Cache_size: 32K
Block_size: 64                                    Block_size: 64
Hit rate: 97.72% (6304), Miss rate: 2.28% (147)   Hit rate: 97.72% (6304), Miss rate: 2.28% (147)
4-Way                                             8-Way
Cache_size: 64K                                   Cache_size: 64K
Block_size: 64                                    Block_size: 64
Hit rate: 97.72% (6304), Miss rate: 2.28% (147)   Hit rate: 97.72% (6304), Miss rate: 2.28% (147)
4-Way                                             8-Way
Cache_size: 128K                                  Cache_size: 128K
Block_size: 64                                    Block_size: 64
Hit rate: 97.72% (6304), Miss rate: 2.28% (147)   Hit rate: 97.72% (6304), Miss rate: 2.28% (147)
------------------------                          ------------------------
```