COL216 BEST CACHE REPORT

Jahnabi Roy
2022CS11094

18 April 2024

1 BEST CACHE PERFORMANCE

1.1 Cache Simulator was run on the following combination of parameters.

1. Write-Allocate, Write-Back, LRU
2. Write-Allocate, Write-Back, FIFO
3. Write-No-Allocate, Write-Back, LRU
4. Write-No-Allocate, Write-Back, FIFO
5. Write-Allocate, Write-Through, LRU
6. Write-Allocate, Write-Through, FIFO
7. Write-No-Allocate, Write-Through, LRU
8. Write-No-Allocate, Write-Through, FIFO

With regards to permutations and combinations, all possible combinations of parameters were thus chosen for testing.

1.2 Test Results

Testing was primarily done for the following test case -

`./cacheSim 256 4 16 <writeMissPolicy> <writeHitPolicy> <replacementPolicy> < gcc.trace`

For interpreting the combination results, with constant cache size of 16,384 bytes, we use hit rates and miss penalties.

`Hit Rate = Total Hits/ Total Operations`
`Miss Penalty = Total Cycles/ Total Misses`

1.2.1 write-allocate write-back lru

```
Hit Rate = 0.9747
Miss Penalty = 728.97
```

1.2.2 write-allocate write-back fifo

```
Hit Rate = 0.9387
Miss Penalty = 642.28
```

1.2.3 write-no-allocate write-back lru

```
Hit Rate = 0.9255
Miss Penalty = 2329.36
```

1.2.4 write-no-allocate write-back fifo

```
Hit Rate = 0.883
Miss Penalty = 1272.89
```

1.2.5 write-allocate write-through lru

```
Hit Rate = 0.9747
Miss Penalty = 6496.931
```

1.2.6 write-allocate write-through fifo

```
Hit Rate = 0.9386
Miss Penalty = 2914.709
```

1.2.7 write-no-allocate write-through lru

```
Hit Rate = 0.9255
Miss Penalty = 11,133.262
```

1.2.8 write-no-allocate write-through fifo

```
Hit Rate = 0.883
Miss Penalty = 297.627
```

1.3 Conclusion of the test results :

As per the results, write-allocate combined with write-back and LRU replacement policy, seems to perform the best since this combination exploits spatial locality (with allocating entire block from memory into cache on miss), and temporal locality (with least recently used replacement policy to deallocate indices which have not accessed very recently). Also the write-back combination reduces the overall cycles due to the delayed memory access. Hit Rate is the

Highest for this strategy and the miss penalty is not very significantly large. The last case of `write-no-allocate write-through fifo` is also a good performing case in terms of miss penalty and hit rate. But due to write-through policy along with write-no-allocate, the number of cycles is much much higher; thus rendering it to not be a very good choice for the case.

Thus, as per theory and tested results, the best performing cache combination is `write-allocate write-back LRU`.

A check was also done to consider the variable parameter of cache size. We consider the case of `256 1 16` and `1 256 4` for the `write-allocate write-back lru strategy`.

1.3.1 For case : 256 1 16 write-allocate write-back lru

```
Hit Rate = 0.9387
Miss Penalty = 642.28
```

1.3.2 For case : 1 256 4 write-allocate write-back lru

```
Hit Rate = 0.914
Miss Penalty = 184.82
```

According to the above data, fully associative cache with lesser cache size seem to promise a low miss penalty and low cycles with a comparable hit rate; while a direct mapped cache showed comparable hit rates and miss penalties, though with lessr cache size, the miss penalties did reduce but cycles increased.

It can be said with regards to cache size that `lower cache size` lead to better miss penalty rates, with comparable hit rates.

Also, a fully associative cache shows a much better miss penalty than set-associative and direct-mapped caches. But this generalisation might fail with comparable cache sizes.

1.3.3 For case : 256 1 64 write-allocate write-back lru

```
Hit Rate = 0.988
Miss Penalty = 2540.443
```

As proved above, fully associative cache with same cache size has a very high Hit Rate but a very Miss Penalty to pay too. So, m-way n-set associative achieves better results than direct mapped cache and fully associative cache.

So, the best performing cache is thus `m-way n-set associative cache with write-allocate write-back lru policy.`