For each microbenchmark, we have simulated cache performance with following configurations:

Simulation configurations

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Random | | NMRU | | LIP | |
| Configuration # | Frequency | Assoc | Frequency | Assoc | Frequency | Assoc |
| 1 | 1.0 GHz | 2 | 1.0 GHz | 2 | 1.0 GHz | 2 |
| 2 | 1.5 GHz | 2 | 1.5 GHz | 2 | 1.5 GHz | 2 |
| 3 | 2.0 GHz | 2 | 2.0 GHz | 2 | - | - |
| 4 | 1.0 GHz | 8 | 1.0 GHz | 8 | 1.0 GHz | 8 |
| 5 | 1.5 GHz | 8 | 1.5 GHz | 8 | 1.5 GHz | 8 |
| 6 | 2.0 GHz | 8 | 2.0 GHz | 8 | - | - |
| 7 | 1.0 GHz | 16 | - | - | - | - |
| 8 | 1.5 GHz | 16 | - | - | - | - |
| 9 | 2.0 GHz | 16 | - | - | - | - |

Note that due to different limits in max associativity and lookup time, some results cannot be obtained. For example, the associativity for NMRU and LIP is 8, and the frequency limits are 2.0 GHz and 1.5 GHz respectively. Selected results are as follows:

Comparison of associativity

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | mm.c | | lfsr.c | | merge.c | | sieve.c | |
| Assoc | miss% | seconds | miss% | seconds | miss% | seconds | miss% | seconds |
| Random (1.0 GHz) | | | | | | | | |
| 2 | 1.3278% | 0.00699 | 92.883% | 0.00411 | 1.0427% | 0.00267 | 34.953% | 0.02557 |
| 8 | 1.1467% | 0.00729 | 92.892% | 0.00411 | 1.2221% | 0.00273 | 34.939% | 0.02557 |
| 16 | 1.1225% | 0.00741 | 92.876% | 0.00410 | 1.2590% | 0.00275 | 34.945% | 0.02557 |
| NMRU (1.0 GHz) | | | | | | | | |
| 2 | 3.9844 % | 0.00966 | 93.607% | 0.00422 | 12.841 % | 0.00311 | 35.129% | 0.02568 |
| 8 | 1.4403% | 0.00715 | 93.070% | 0.00411 | 1.3152% | 0.00271 | 34.995% | 0.02561 |
| LIP (1.0 GHz) | | | | | | | | |
| 2 | 3.9844% | 0.00966 | 93.607% | 0.00422 | 12.841% | 0.00311 | 35.129% | 0.02568 |
| 8 | 7.2513% | 0.00932 | 94.183% | 0.00410 | 15.122 % | 0.00319 | 35.154% | 0.02570 |

Comparison of replacement/insertion policy:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | mm.c | | lfsr.c | | merge.c | | sieve.c | |
|  | miss% | seconds | miss% | seconds | miss% | seconds | miss% | seconds |
| Freq=1.0GHz, Assoc=2 (when assoc=2, NMRU and LIP are identical) | | | | | | | | |
| Random | 1.3278% | 0.00699 | 92.883% | 0.00411 | 1.0427% | 0.00267 | 34.953% | 0.02557 |
| NMRU | 3.9844% | 0.00966 | 93.607% | 0.00422 | 1.2841% | 0.00311 | 35.129% | 0.02568 |
| LIP | 3.9844% | 0.00966 | 93.607% | 0.00422 | 1.2841% | 0.00311 | 35.129% | 0.02568 |
| Freq=1.0GHz, Assoc=8 | | | | | | | | |
| Random | 1.1467% | 0.00729 | 92.892% | 0.00411 | 1.2221% | 0.00273 | 34.939% | 0.02557 |
| NMRU | 1.4403% | 0.00715 | 93.070% | 0.00411 | 1.3152% | 0.00271 | 34.995% | 0.02561 |
| LIP | 7.2513% | 0.00932 | 94.183% | 0.00410 | 15.122% | 0.00319 | 35.154% | 0.02569 |

As we can see from the results, cache design affects performance only slightly. For the given micro benchmarks, we have performance of Random > NMRU > LIP most of the time.

However, we do not know other design requirements for the CEO’s CPU. If the CPU is designed for specific purposes, we should carry out simulations on these programs and choose the best configuration. Moreover, if the frequency of the CPU exceeds 2GHz, Random may be the only option for the CEO.

Q1: Why does the 16-way set-associative cache perform better/worse/similar to the 8-way set-associative cache?

A1:

We compare 8-way and 16-way cache under Random replacement policy, because it is the only one available.

By comparing the results with 1GHz frequency, we can see that 16-way cache performs slightly outperforms 8-way cache on all 4 micro-benchmarks. This is probably because 16-way cache is more flexible to place a cache block, and costs for different associativity are similar for Random replacement policy.

However, this may not be the case for NMRU/LIP. Because 16-way configuration is not available for NMRU/LIP, we compare 8-way and 2-way configuration. We can see that 8-way configuration may not necessarily outperform 2-way configuration.

Q2: Why does Random/NMRU/LIP/None perform better than the other replacement policies?

A2:

When other configurations are the same, we observe NMRU > LIP > Random in performance, but just slightly. However, there are some exceptions.

First, NMRU and LIP are identical when associativity is 2, which can be easily verified. When associativity is 8, the simulation results show that LIP slightly outperforms NMRU. This is probably because LIP prevents thrashing by prioritizing the blocks that are accessed repeatedly. For example, if we access blocks sequentially with working set larger than cache size, we will see a very lot hit rate with NMRU (even 0% if LRU). With LIP, we are able to maintain a set of blocks of size (cache\_size-1) in cache consistently.

Intuitively, Random doesn’t outperform LIP/NMRU because it doesn’t prioritize which block gets evicted. At least for the access pattern of the 4 given micro benchmarks, Random slightly deteriorates performance.

However, since Random has lower overhead, it supports larger associativity and frequency, so it may be the only option on some machines.

Q3: Is the cache replacement/associativity important for this workload, or are you only getting benefits from clock cycle? Explain why the cache architecture is important/unimportant.

A3:

It depends on the micro benchmark. For lfsr.c and sieve.c, cache replacement/associativity seems to affect overall performance only slightly. For mm.c and merge.c, performance varies across different cache policies. It really depends on the access pattern.

However, clock cycle does affect performance in all cases.