OV Assignment 2 Report

Bogdan Foca
Student number - 4419936

1 CACHE ARCHITECTURE

We have 3 cache layers followed by the DRAM memory. Each cache layer is larger in size than the previous one. All of them have the same line width. More details are presented in other chapters.

2 CACHE PERFORMANCE

To evaluate cache performance, I have looked at the number of cache hits, both when reading and when writing. Due to the nature of the simulator, I have not measured the time performance of the cache.

It is more beneficial to have more hits in the lower levels of the cache, but this happens naturally anyway, so it was not something I have taken in consideration in the rest of the report.

3 WORK DIVISION AND RESOURCES

I worked on this assignment alone. As resources, I used this Wikipedia article: https://en.wikipedia.org/wiki/Cache_inclusion_policy. For the rest of the tasks I used the OV lectures + lectures in my bachelor's related to this.

4 IMPLEMENTATION

4.1 N-Way Set Association

The cache line array is split into sets of the same size. The set of which a line is part of is determined based on its address. From my research I found that the set is usually computed using a modulo operation, so adjacent lines correspond to adjacent sets, not the same one.

Eviction operations only operate on the set that the line is a part of.

4.2 Third Testing Algorithm

The third algorithm reads from memory the addresses from 0 to N, and then writes at N + 1. N is increased, and the operation is repeated. The loop has been split across frames, so that the program does not become unresponsive. I am expecting to see an improvement in cache hits when using LFU eviction, since the earliest addresses are always used and should stay in the cache.

4.3 Per-Level Parameters

Each cache level has its own size, line width, set size, eviction policy, and can be set to be inclusive or exclusive.

To implement dynamic sizing and line width, the line arrays are dynamically allocated and deallocated in the constructor/destructor. In order not to have to change the base code too much, I added copy constructors and overloaded the assignment operator, so that cache line assignments still work.

Generally when choosing the best parameters, I followed this logic:

- For cache size: Try to keep it as low as possible, as larger cache is more expensive and slower.
- For line width: A larger size benefits spatial locality, but might be detrimental for temporal locality.
- For set size: A lower set size makes the cache faster, but increases the chances of an eviction.

4.3.1 Spiral Algorithm. Temporal locality is basically non existant, but as we get closer to the center of the spiral, spatial locality starts to be important, so I am proposing a large line width. We are hoping that the next accesses will already be present in cahce. Because of that, we do not need a lot of lines stored in cache, so set size can be on the lower side. For the same reason cache size can also be smaller.

4.3.2 Buddhabrot Algorithm. Random accesses mean very low spatial locality, so I recommend a low line width, to not waste cache space. Cache size should be large to be able to store many lines. A high set size will also reduce the number of evictions.

4.3.3 General Use. For general use, I read about what real life cpus use. Usually the set size is 4 or 8, with a larger value for the last level. In my case I used 8 for the first 2 levels and 16 for the last one.

Line width is usually 32 or 64 bytes long, but more recent cpus usually use 64.

For the cache size, I started with 4096 for the first layer and doubled it for each one after. The usual increase is way larger than that, the last layer being in the order of MBs, but the low size of the DRAM in our simulation does not allow for such high increases.

4.4 Eviction Policies

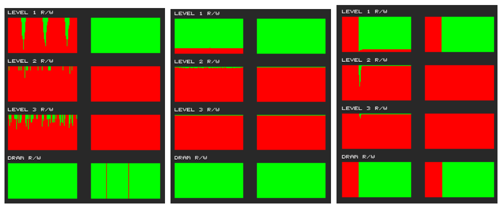
To implement LFU and LRU eviction policies, each cache level has a total access counter.

All tests have been performed with the following architecture:

	size	line width	set size
L1	4096	64	8
L2	16384	64	8
L3	32768	64	16
DRAM	3276800	64	-

All levels are exclusive.

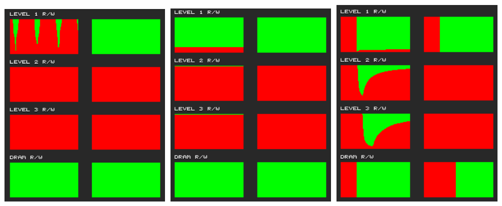
4.4.1 Random. A random line in the set is evicted.



(a) (b) (c)

Figure 1: (a) Alg. 1 (b) Alg. 2 (c) Alg. 3

4.4.2 LFU. Every time a line is accessed, its access counter is updated. The line with the least accesses in the set is evicted.



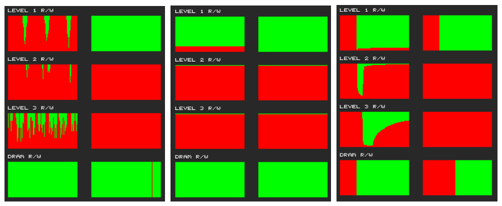
(a) (b) (c)

Figure 2: (a) Alg. 1 (b) Alg. 2 (c) Alg. 3

For the first 2 algorithms, the figures look similar. This is expected since algorithm 1 does not reaccess previously accessed line before completing an entire spiral. The second algorithm has pretty random accesses, so again no change is expected.

For the third algorithm, we can see a big increase in cache hits, as I assumed when adding it.

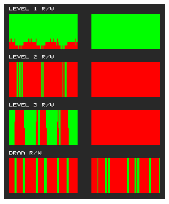
4.4.3 LRU. Every time a line is accesses, it's last access time is set to the cache's current access counter. The line with the lowest last access time is evicted.



(a) (b) (c)

Figure 3: (a) Alg. 1 (b) Alg. 2 (c) Alg. 3

As above, algorithm 2 does not show any changes. Algorithm 3 has more cache hits, but less than with LFU. This is caused by the fact that the last elements in the array are also accessed again in a new iteration, but they might getting evicted until then.



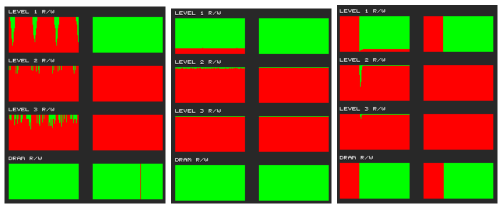
When the spiral approaches the center we get a lot more hits. This is observable with any of the policies, but even more with LRU.

4.5 Inclusiveness and Exclusiveness

The template that was provided had an inclusive cache scheme, so no changes needed there.

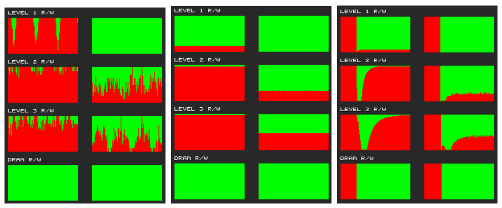
When a cache level is set to be exclusive, whenever a line is brought from a lower cache level, the replaced line from the upper one is sent to the position in the lower level from where we executed the retrieval. That way, the retrieved line does not exist in both levels.

For testing, the same architecture was used as in the previous subchapter, frst with all layers set to inclusive, and then with all layers set to exclusive.



(a) (b) (c)

Figure 4: Inclusive (a) Alg. 1 (b) Alg. 2 (c) Alg. 3



(a) (b) (c)

Figure 5: Exclusive (a) Alg. 1 (b) Alg. 2 (c) Alg. 3

We can see a big increase in cache hits when the cache is set to exclusive. More distinct lines are stored in cache at a specific time, causing those hits.