1 Pin Assignment: Victim Cache [100 Points]

1.2 Inclusion Property [10 Points]

A. *Explain the concept of cache inclusion property.*

In multi-level caches cache inclusion dictates how data is stored between two levels of a cache hierarchy. If copies of all cache blocks stored in the lower level cache are present in the upper level cache then the lower level cache supports the multi-level inclusion (MLI) property [1]. For example, in a L2 cache that is inclusive of a L1 cache, a miss in both would load the cache line into both L1 and L2 from memory. Although MLI is less complex than the alternatives, a disadvantage is that space for unique cache entries is wasted. Cache area that could have been used for holding unique data across all caches is now used to store copies of data that is already present at other levels.

B. *The paper also mentions that the line size of the second-level cache is 8 to [16] times larger than the first-level cache line size, which also violates the inclusion property. Explain why this is true with a simple example.*

Consider Jouppi's baseline cache design [2]:

- L1 Cache: direct-mapped, 4KB size with 16B lines, number of sets 500
- L2 Cache: direct-mapped, 1MB size with 128B lines (which is 8 times larger than that of L1), number of sets around 7800

To show that the L2 cache is not inclusive of the L1 cache we need to show that the L1 cache can contain entries that the L2 cache does not contain. Equationally we can show that inclusion is violated through a counter-example using *Lemma 1* of Baer et al. [1] for associativity of the caches, A, and cache line sizes, B: **if MLI holds then** $A_{L2} \geqslant A_{L1} \times \frac{B_{L2}}{B_{L1}}$. For the baseline cache design given above:

$$1 \geqslant 1 \times \frac{128\,\textit{(L2 line size)}}{16\,\textit{(L1 line size)}}$$

Thus the design obviously cannot satisfy MLI. To satisfy inclusion for direct-mapped caches as in Jouppi's design the cache line sizes have to be equal. The intuition for this is that with the parameters set as is, blocks at addresses that are close together but in different sets in L1 map to the same set in L2. Since L2 holds a single line per set, multiple lines would fail to be propagated to L2 thus violating inclusion.

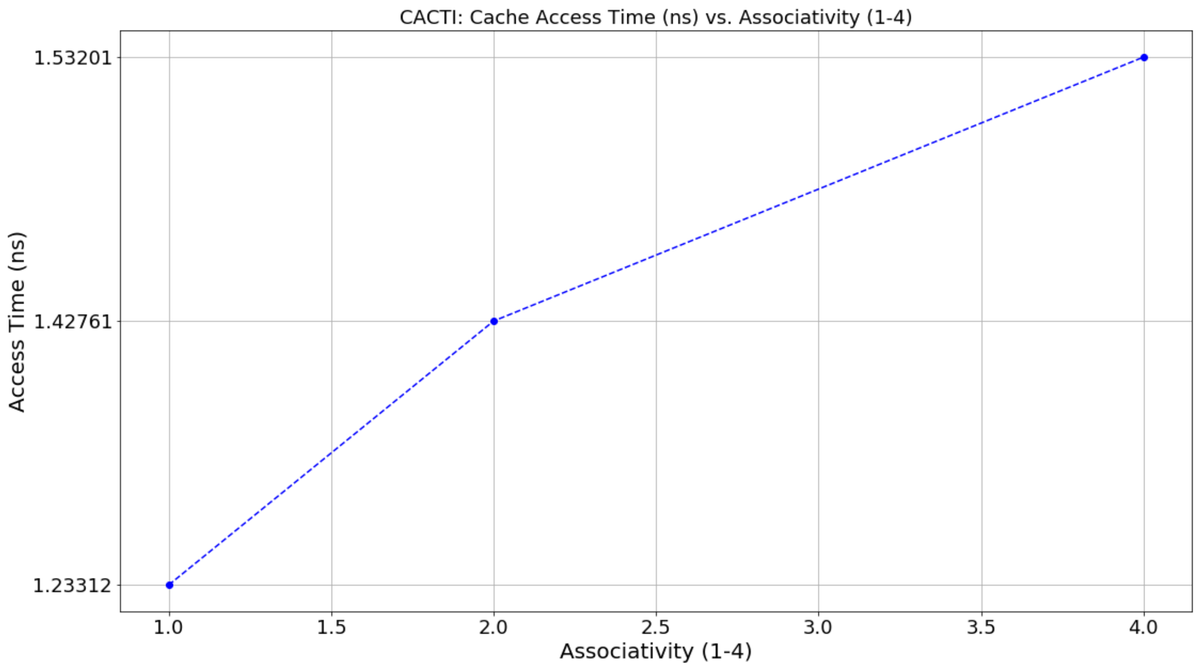

Figure 1: Cache simulation in CACTI while sweeping associativity from 1 to 4 for a cache with 32B line size, 8KB total cache size, 1 bank and using 40nm technology.

1.3 CACTI [10 Points]

Associativity (A) in caches dictates the number of cache lines that each cache set can hold. Figure 1 shows our CACTI simulation. We record the access time (ns) for three associativities: 1, 2, 4. Our results show a clear near-linear increase in cache access time when increasing associativity. One reason for this is the increased effort to find a line in the cache with higher associativitiy. In a direct-mapped cache ($A = 1$) a cache line is found immediately by its set field. On the other spectrum, in a fully-associative cache (all cache lines are part of the same set) finding a cache line requires comparisons with each cache line's tag field. Thus as associativity increases the work required to find and place a cache increases; this is reflected in our CACTI simulation.

1.4 Stream Buffers [10 Points]

A. *Explain how a stream buffer works with a reference stream that skips to every third word*

Stream buffers are a hardware prefetching technique to reduce cache misses, especially due to compulsory and capacity conflicts, in a multi-level cache. On a cache miss a stream is allocated that starts fetching some number of consecutive cache lines into a buffer (not cache in Jouppi's proposal [2]). On cache access the stream buffer(s) are checked as well and if the desired line is present, no expensive accesses to upper-level memory are needed. On a stream buffer miss the buffer has to be flushed (since it works in a FIFO manner).

A data reference stream stride (the memory address gap between consecutive accesses) that is non-sequential (e.g., 3-unit stride) could cause the buffer to be flushed more often depending on the cache line size. If the cache line size is not capable of holding 3 words and the reference stream skips to every 3rd word, the head of the buffer will never yield a hit and thus the buffer will always be flushed. On the other hand, if the lines that the buffer fetches are, for instance, four words long, the stream buffer will provide benefit for a strided reference stream of 3 words.

B. *How would you design stream buffers differently to handle non-unit-stride?*

The main issue with non-unit-stride accesses in the original stream buffer design is that we fetch sequential lines into the buffer immediately after the miss address but we only ever are allowed to compare to the head of the buffer. One solution proposed by Jouppi in his extended technical report [1] is that of quasi-sequential stream buffers. There each entry in the buffer has a comparator (instead of only the head) which allows non-sequential lines to be fetched from the buffer. An alternative that would work for larger and dynamic strides is to predict the reference stream's stride using the difference between the current and the previous address and prefetch according to the predicted value [3].

[1]https://www.hpl.hp.com/techreports/Compaq-DEC/WRL-TN-14.pdf

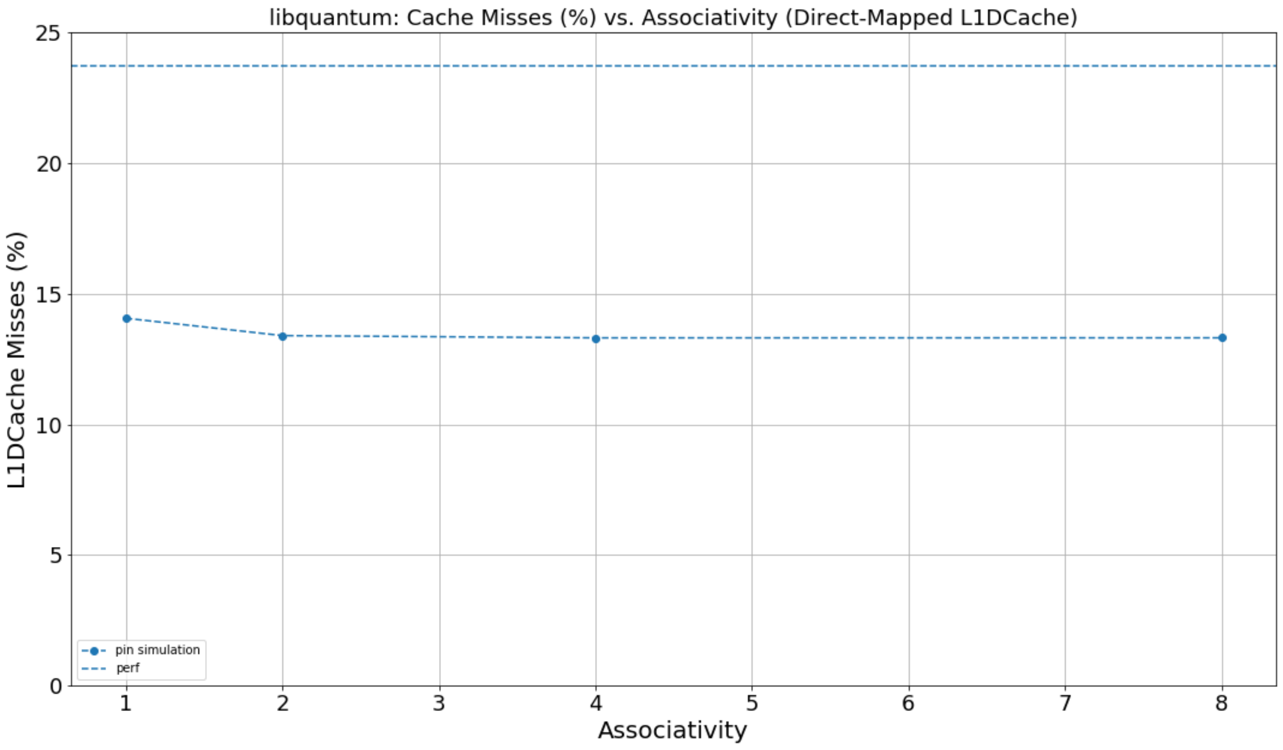

Figure 2: Simulate (using Pin) an direct-mapped L1 data cache without victim cache when running the libquantum_O3 benchmark. We sweep associativity from 1 to 8 and record the percentage of L1DCache misses. We also plot the *perf* measured cache misses.

1.5 Victim Cache Implementation [70 Points]

A: Sweep L1 DCache associativity without victim caching and compare to perf

Figures 2 and 3 show the percentage of L1 data-cache misses for the libquantum_O3 and hmmer_O3 benchmarks respectively while sweeping associativity (of L1DCache) from 1 to 8. As a reference we also record the miss percentage detected by *perf-stat(1)* through the same benchmarks (**note:** we measure all perf-stat events in user-space). Through the following command we found that the **L1 cache associativity on our test server is 8**:

```
$> dmidecode -t cache
Cache Information
        Socket Designation: L1-Cache
        Configuration: Enabled, Not Socketed, Level 1
```

```
Operational Mode: Write Back
...
Associativity: 8-way Set-associative
```

For the hmmer benchmark (figure 3) varying associativity has the expected effect of reducing the number of cache conflicts by a significant 30%. This is because higher number of cache lines per set reduce the number of cache conflicts that occur when multiple lines map to the same set. Interestingly the rate at which cache conflicts are removed slows down at an associativity of 8 (despite the fully-associative cache case only occurring at associativity of 64). As expected perf shows lower cache miss results, though our simulated miss-rate tends towards it (at an associativity our miss-rate is within 4% of perf's).

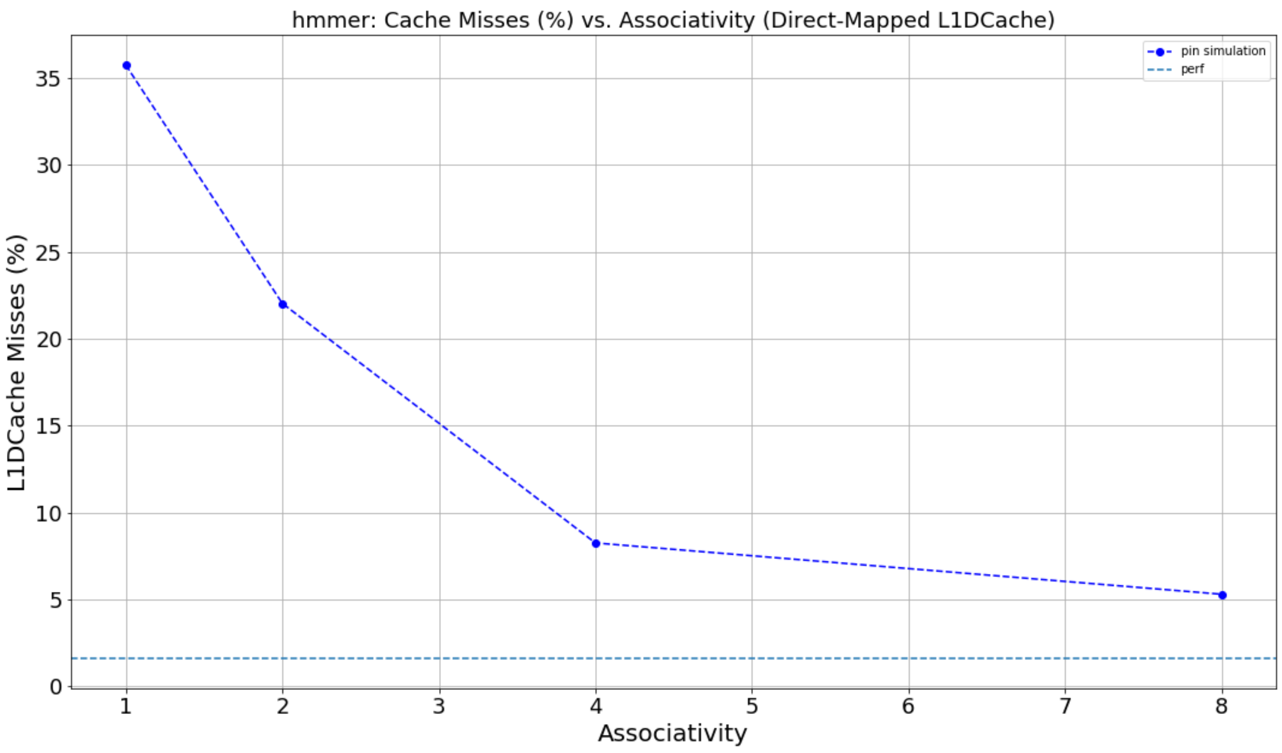

Figure 3: Simulate (using Pin) an direct-mapped L1 data cache without victim cache when running the hmmer_O3 benchmark. We sweep associativity from 1 to 8 and record the percentage of L1DCache misses. We also plot the *perf* measured cache misses.

Important to note is that perf versus simulation measurements cannot be accurately compared since both operate on different cache hierarchies (ours being a highly simplistic model).

In the libquantum_O3 case our results show that sweeping associativity has a negligible effect on cache miss rate (the absolute number of L1DCache misses is shown in figure 6; there we see the actual number of misses decrease by around 5%). This implies that the conflicts that arise from the workload are predominantly not conflict misses. Example workload patterns that would trigger such behaviour are large number of random/new address accesses (compulsory misses) or concurrent operations on large data structures that do not fit into the cache conveniently (capacity misses).

B,C,D: sweep number of entries in victim cache for a direct-mapped and 8-way set-associative L1 Dcache

The hmmer_O3 benchmarks in figure 4 show an approximately linear decrease in cache miss percentage as we increase the number of victim cache (VC) entries when the L1 cache is direct-mapped. When the cache is 8-way set-associative the cache miss rate negligibly decreases. As VC entries increase, more L1 data cache conflicts are removed as expected. Victim caches, however, are not capable of removing compulsory or capacity misses [2] and this possibly explains the stark difference in the direct-mapped vs. set-associative L1 benchmarks. When running hmmer_O3 with set-associative cache the conflict misses are naturally reduced due to associativity. Then if the remaining misses are due to cache capacity or first-seen data, increasing VC size would not affect the miss rates. The purpose of victim caches is to retain the benefit of direct-mapped cache access time while reducing the number of conflict misses due to their low associativity [2]; this intent is reflected in our results. Both of our simulations are outperformed by the actual cache of the server.

For the libquantum_O3 benchmark we see little improvement (1% decrease) in cache miss rates when introducing a victim cache. The miss-rate improvement plateaus at 2 VC entries. For a 8-way set-associative L1 cache our simulation shows that miss rate is not affected by a victim cache whatsoever. These findings mean there are not enough conflict misses that occur for the VC to remove which in addition to the previously described example reference patterns (Q1.7A) could be caused by data accesses that are not closely spaced and thus cause less conflicts.

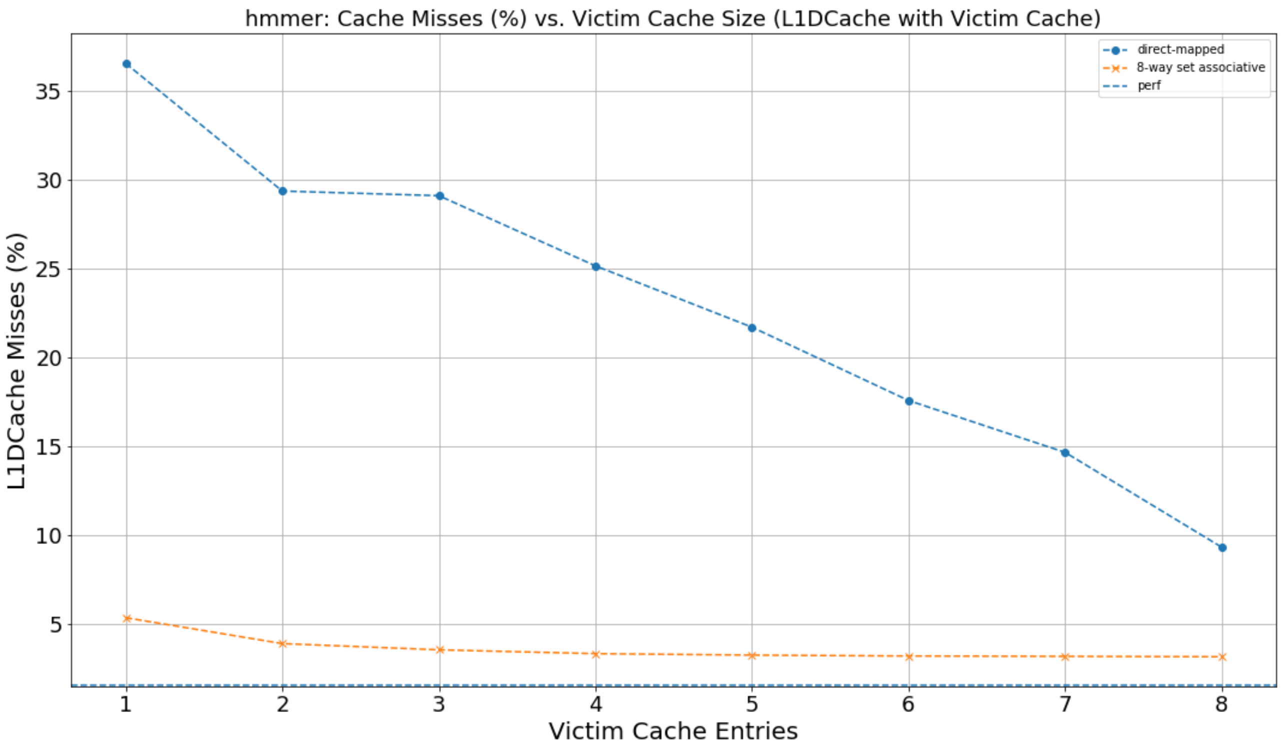

Figure 4: Simulate (using Pin) an direct-mapped/8-way set-associative L1 data cache with a victim cache when running the hmmer_O3 benchmark. We sweep the number of victim cache entries from 1 to 8 and record the percentage of L1DCache misses. We also plot the *perf* measured cache misses.

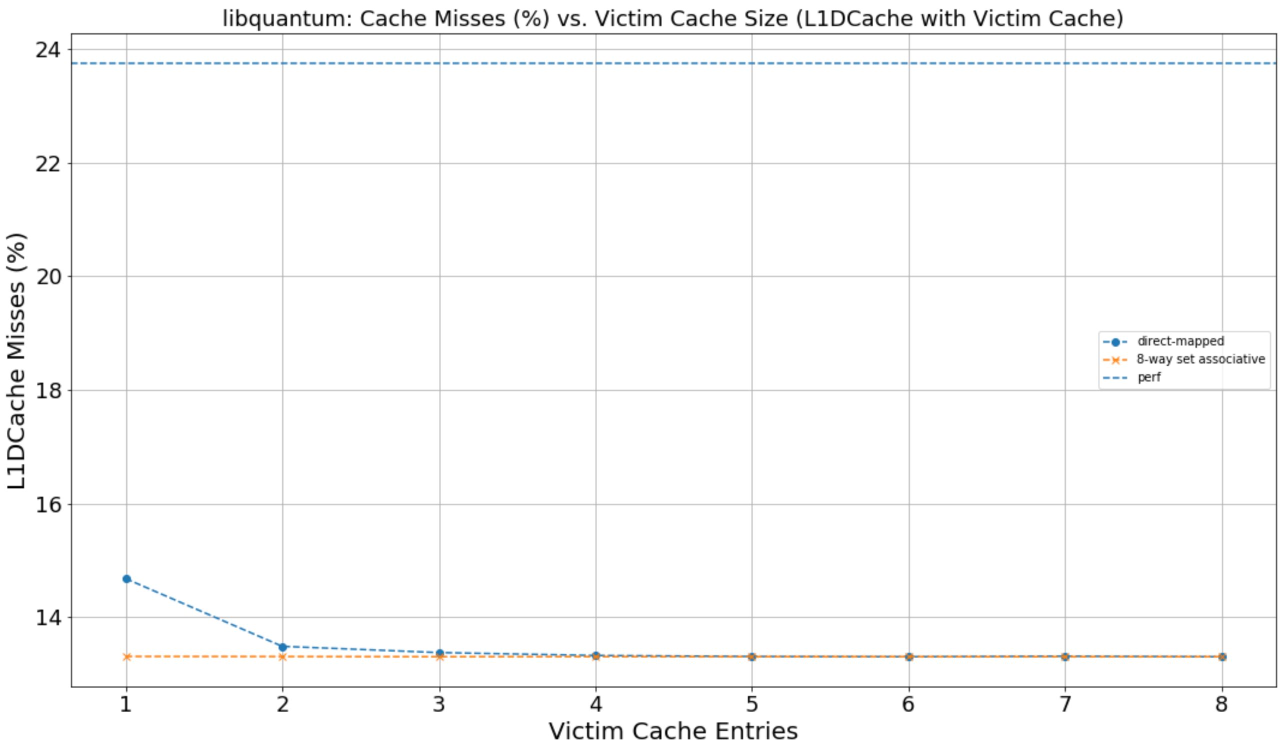

Figure 5: Simulate (using Pin) an direct-mapped/8-way set-associative L1 data cache with a victim cache when running the libquantum_O3 benchmark. We sweep the number of victim cache entries from 1 to 8 and record the percentage of L1DCache misses. We also plot the *perf* measured cache misses.

References

[1] J-L Baer and W-H Wang. On the inclusion properties for multi-level cache hierarchies. In *[1988] The 15th Annual International Symposium on Computer Architecture. Conference Proceedings*, pages 73–80. IEEE, 1988.

[2] Norman P Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *ACM SIGARCH Computer Architecture News*, pages 364–373. ACM, 1990.

[3] Subbarao Palacharla and Richard E Kessler. Evaluating stream buffers as a secondary cache replacement. In *ACM SIGARCH Computer Architecture News*, pages 24–33. IEEE Computer Society Press, 1994.

Appendices

A Additional Figures

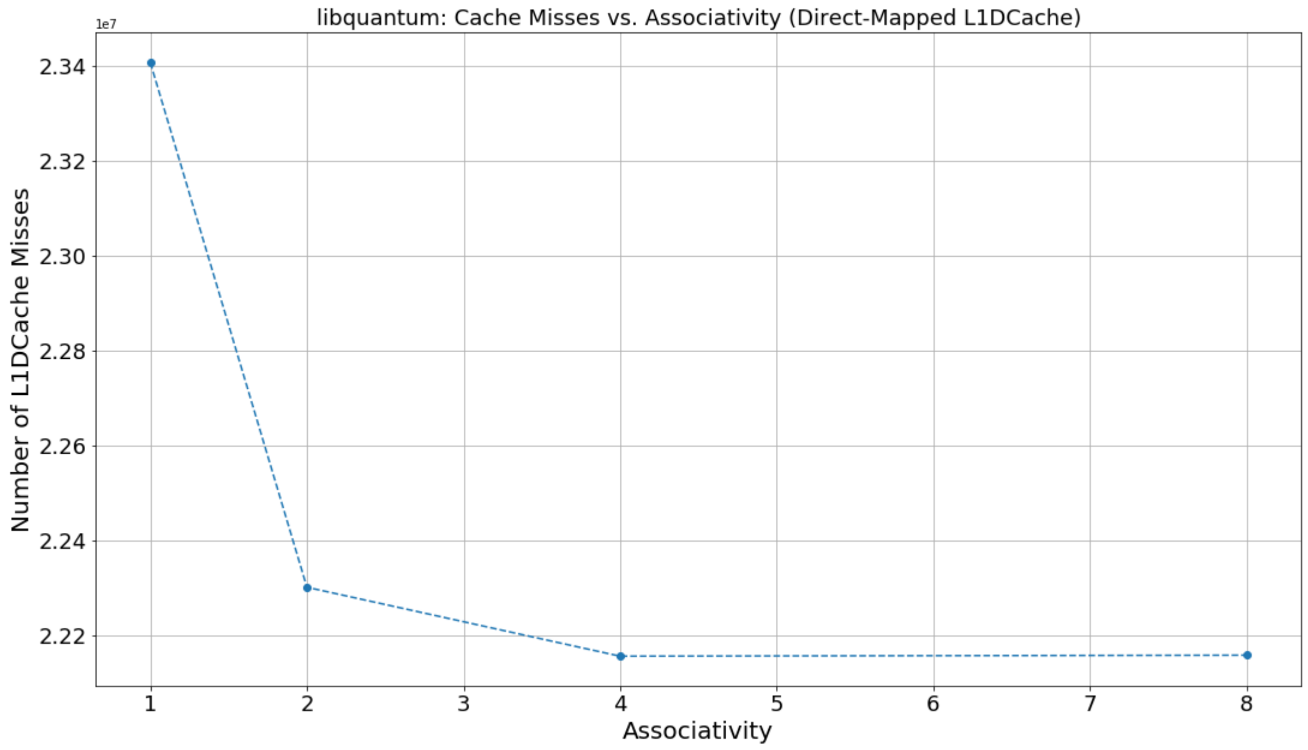

Figure 6: As in figure 2 but we plot the actual number of L1DCache misses

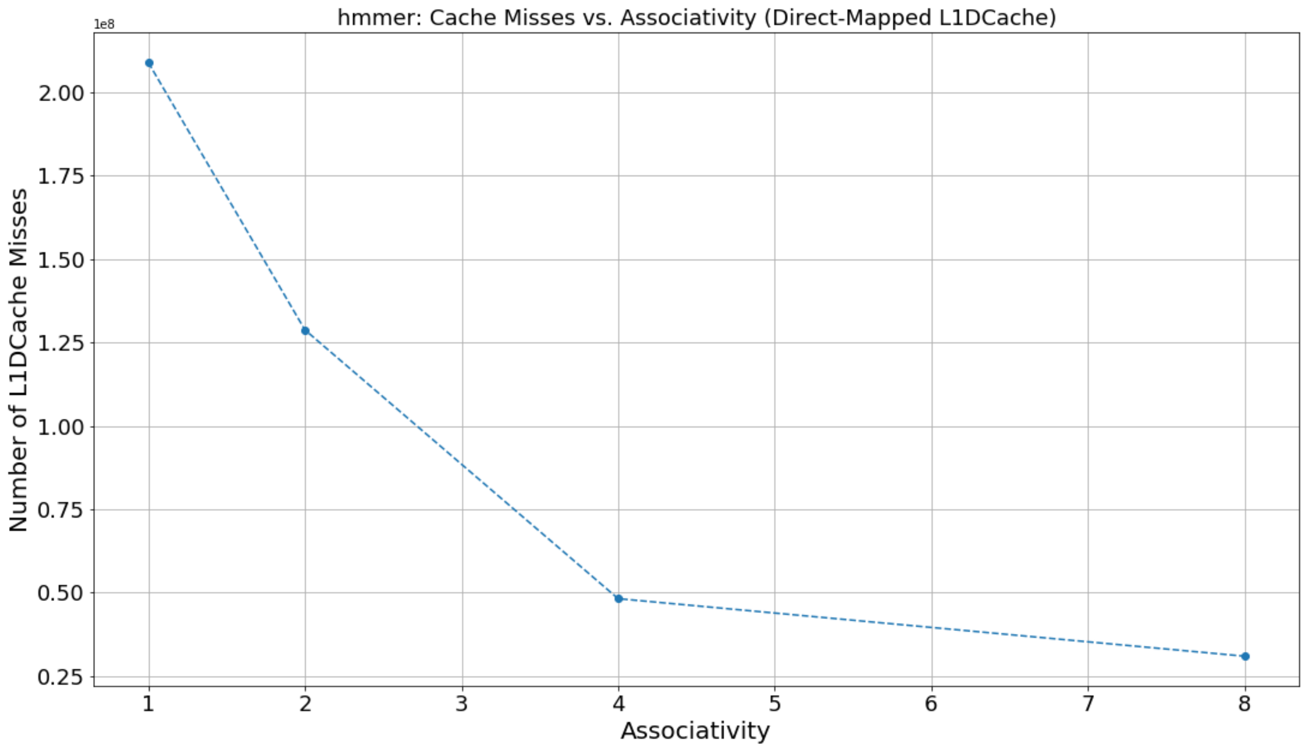

Figure 7: As in figure 3 but we plot the actual number of L1DCache misses

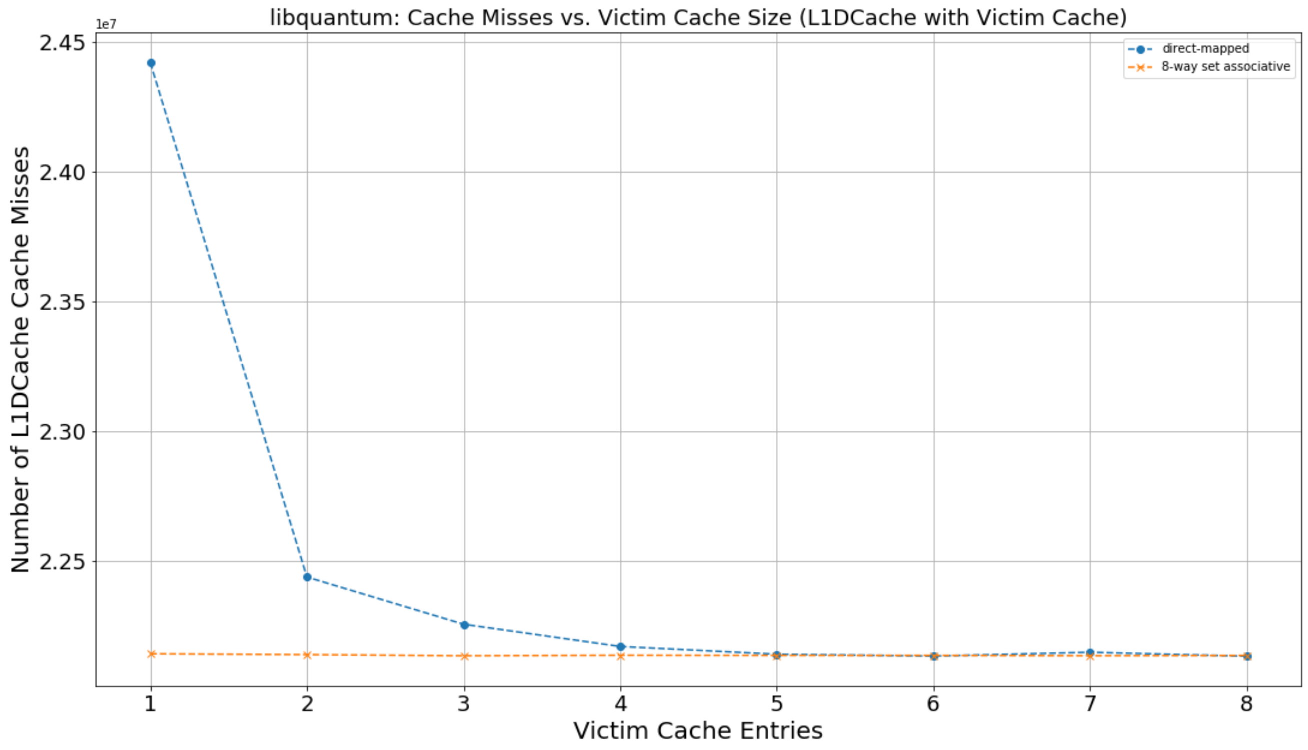

Figure 8: As in figure 8 but we plot the actual number of L1DCache misses

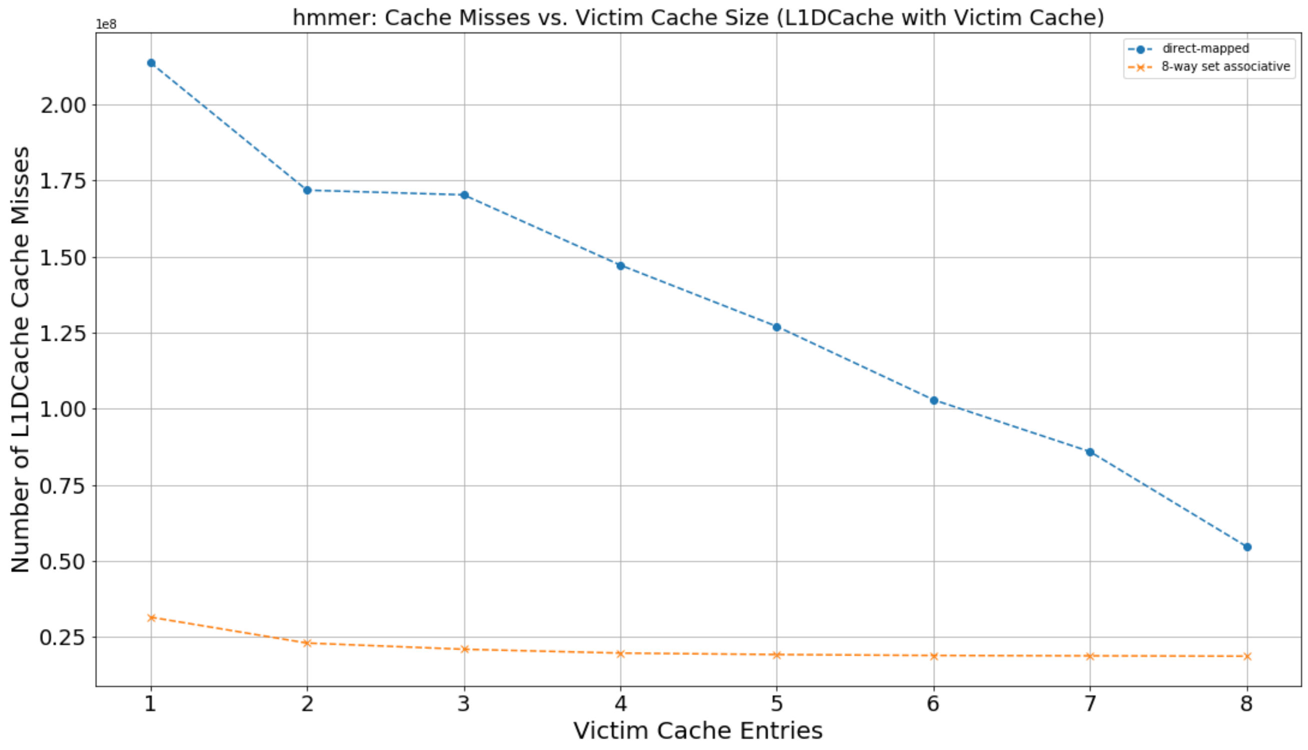

Figure 9: As in figure 9 but we plot the actual number of L1DCache misses