Computer Architecture (CS422) Review

Critical Paper Review: The V-Way Cache
Reviewer: Manish (190477)

Summary

Generally, caches have static associativity which can result in a negative performance hit by conflict misses due to the non-uniform distribution of memory accesses across different sets. To reduce conflict misses we can use fully associative caches but fully associative caches consume more power and have more hit latency than set associative. The V-way Cache allows changing associativity on a per-set basis depending on the non-uniformity of access using more tag stores than the data lines. Replacement can be done locally on a set basis or globally depending on the different misses. For global replacement Reuse Replacement policy is used based on frequency information. A 256kb, 8-way L2 outperforms traditional L2 implementation by 13% and this results in an 8% IPC improvement.

Details

The V way Cache Implementation:

- For every data line, there are 2 tags (TDR = 2). Every Tag store entry contains a valid bit, dirty bit and Forward Pointer(FPTR). Every datastore entry contains a data line, a valid bit, Reverse Pointer(RPTR).
- Tag store uses LRU replacement policy per set basis and Reuse Replacement for global replacement scheme.
- If cache access is a hit then using FPTR data will be accessed and replacement information will be updated.
- If it is a cache miss then there are 2 cases to consider. The first case is when there exists at least one invalid tag store entry in the target set. The second case is when all the tag store entries are valid in the target set.
- In the First case, a global replacement policy is used to choose the data victim. The data victim will be removed and the RPTR tag of the data victim will be invalidated. Victim tag bits, FPTR, valid bit and RTPR will be set properly. Replacement info will be updated in both the tag and data store.
- In the Second case LRU replacement policy will be applied and a tag victim will be chosen. The data line pointed by the tag victim will be evicted. FPTR and RPTR will not be changed. Replacement info will be updated in both the tag and data store.

The Reuse Replacement policy:

- 2 bits for every data line are used to store 4 different reuse counters (0,1,2,3+). Reuse Counter Table(RCT) is a physically separate structure to not cause any delay in reading and writing during RCT updates.

- PTR register is used to point to the start of the search to the data victim. For the newly came cache line, the reuse counter will be zero.
- On a cache hit reuse counter of the data line is incremented by 1. On a cache miss, start searching from PTR. The replacement engine test and decrement every non-zero reuse counter. The test continues till the data victim is found(wrap around if the bottom of RCT is reached).
- After the data victim is found, Increment the PTR to point to the data line next to the data victim.

Positives

- Benchmarks are chosen wisely based on the scope of improvement. A good analysis of performance gain or hit is provided for every benchmark. How variable set demand, global replacement policy, cache size and TDR affect performance for every benchmark.
- Cost analysis for implementing V-way cache is done properly covering hardware, latency and energy. Different variables are taken in account such are line size, the technology of the processor and TDR which provide a good understanding of the cost associated with its implementation.
- Good global replacement policy which uses fewer hardware resources and has similar performance to perfect LRU. Analysis of the number of bits used for the reuse counter is also done beautifully.

Negatives

- In conclusion author claimed 8% IPC improvement. But this improvement is when we don't count the latency added due to the extra tag store and FPTR which add an extra 1 cycle to hit latency. Due to that we only got a 6.8% improvement. The authors didn't give any accurate method to reduce this extra hit latency.
- SPEC CPU2000 benchmark is used to analyse all the results. SPEC CPU2000 is not good enough to represent all real-world applications. In the RRIP paper we see that on SPEC we are not getting much benefit but on other benchmarks like PC games, multimedia etc we got the benefit. In this case, maybe it is the reverse of that. More real-world application benchmarks should be used to make results more robust for different use cases.
- Only one configuration of L1 is used to analyse most of the results. L1 size can affect the number of misses which comes to L2 and can change the distribution of RCT which can affect the number of cycles to detect data victim. This change in the eviction of the blocks can change the performance.
- Analysis of how V-way cache performance changes with changing the L2 associativity is not done which makes results more specific to the configuration used in results.

Possible Extensions

Having at least one invalid tag in its target set invokes global replacement, so we call them global misses. Misses which invoke local replacement are local misses.

- Currently 2 bits are used per data line to store the reuse counter. Data shows that the replacement latency for finding data victim is 1 cycle for 91.3% of the global misses. After 5 cycles we terminate by returning the current data line even if we found the victim or not. The spatial locality for L2 is very less. This motivates us to use only one bit per data line instead of 2 which can serve our purpose for storing whether this data line is used or not. One bit is enough because on avg after 1.2 cycles we have data victim. So for traversing the complete RCT we need around 853(1024/1.2) global misses. We evict this data line only after there are 854 global misses and there is no hit on this line. This seems a reasonable tradeoff. This will save energy and cost for extra hardware.
  We can further extend this idea to local replacement. In local replacement, we use 3 bits per tag for an 8-way associative cache to implement the LRU replacement policy. In the traditional cache, we expect to use all 8 tags as for every tag we have a data line. But in the V-way associative cache, we double the tag store by doubling the number of sets and keeping the associativity the same to reduce power consumption and the data lines are also constant. On avg. we only expect to use 4 tags per set so 2 bits suffice our purpose. Even when we have all 8 tags valid per set(rare cases), 2 bits can store a good amount of info to choose the victim out of them. In most of the cases(data shown in the paper) we invoke global replacement, not a local replacement. This can save energy and reduces hardware cost further.

- More Energy consumption and extra latency due to additional tag store entry and FPTR to each individual entry is a problem in the V-way cache. For some of the benchmarks, TDR value of 1.7 is enough and for some of the benchmarks, more TDR means more performance. What we can do is hardware can support up to TDR 3. Depending on the application we can implement to vary TDR from 1 to 3. This saves us energy and latency on applications with uniform access and provides performance on highly skewed applications.

- For smaller caches like L1, victim caches can provide variable associativity. For LLC we can also do an analysis of how the v-way cache can affect the performance. We know the LLC is of a much bigger size and has more associativity. But it is also shared between processors which can have a non-uniform distribution of memory accesses across different sets. LLC have the potential to get benefit from V-way cache.