CS 529 Project 1 Report

Jou-Chi Huang

Step1, 2:

Assoc. Level	L1I accesses	L1I misses	L1D accesses	L1D misses	L1I hit rate	L1D hit rate	AMAT
1	757341	83386	242661	117514	88.99%	51.57%	12.01
2	757341	79575	242661	114936	89.49%	52.64%	11.51
4	757341	79454	242661	115859	89.51%	52.25%	11.49
8	757341	79130	242661	116316	89.55%	52.07%	11.45
16	757341	79165	242661	115858	89.55%	52.26%	11.45
32	757341	79272	242661	116275	89.53%	52.08%	11.47

Step 3, 4:

Assoc. Level	L1I accesses	L1I misses	L1D accesses	L1D misses	L1I hit rate	L1D hit rate	AMAT
1	757341	83386	242661	1581	88.99%	65.13%	12.01
2	757341	80185	242661	1412	89.41%	65.20%	11.59
4	757341	79985	242661	1361	89.44%	65.22%	11.56
8	757341	79699	242661	1319	89.48%	65.24%	11.52
16	757341	79738	242661	1265	89.47%	65.26%	11.53
32	757341	79916	242661	1256	89.45%	65.27%	11.55

Step 5:

Assoc. Level	L1I accesses	L1I misses	L1I hit rate	L1D accesses	L1D misses	L1D hit rate	L2 accesses	L2 misses	L2 hit rate	AMAT
1	757341	80185	89.41%	242661	1412	65.20%	82668	13754	83.36%	11.59
4	757341	80185	89.41%	242661	1412	65.20%	82668	10835	86.89%	11.59
16	757341	80185	89.41%	242661	1412	65.20%	82668	9851	88.08%	11.59
64	757341	80185	89.41%	242661	1412	65.20%	82668	10031	87.87%	11.59
128	757341	80185	89.41%	242661	1412	65.20%	82668	10117	87.76%	11.59

The Assoc. Level columns are for L2 cache setting. After deploying two-level caches, I consistently observed the same hit rate for both L1 instruction and data caches. This could potentially be attributed to misconfigurations in the settings of the dirty bit. Consequently, when L2 uses the

return value (cache hit/cache miss) from L1 to determine whether the data is present in L2 or not, it results in inaccuracies in the collected statistics. I am currently working to diagnose and resolve this issue.

Step 6:

- By increasing both the cache size and block size of L1 and L2, as well as the associativity of L1, I observed a higher L1 hit rate. This is primarily due to the increased cache size, which increases the likelihood of most data being stored in the caches, resulting in more hits. However, the L2 hit rate did not show significant growth, possibly because the increased number of hits in L1 reduces the need for the system to access L2 for data retrieval. Additionally, it is possible that the cache size exceeds the workload's requirements, resulting in a considerable portion of the cache remaining empty without data.

```
two_caches = TwoLevelCache(l1_size = 2056,l1_block_size = 64, l1_associativity=4, l2_size = 18000,
l2_block_size = 256, l2_associativity = l2_associativity)
```

Assoc. Level	L1I accesses	L1I misses	L1I hit rate	L1D accesses	L1D misses	L1D hit rate	L2 accesses	L2 misses	L2 hit rate	AMAT
1	757341	45521	93.99%	242661	888	65.42%	46979	9579	79.61%	7.01
2	757341	45521	93.99%	242661	888	65.42%	46979	8174	82.60%	7.01
4	757341	45521	93.99%	242661	888	65.42%	46979	8590	81.72%	7.01
8	757341	45521	93.99%	242661	888	65.42%	46979	8699	81.48%	7.01
16	757341	45521	93.99%	242661	888	65.42%	46979	8693	81.50%	7.01
32	757341	45521	93.99%	242661	888	65.42%	46979	8926	81.00%	7.01

- After changing the cache block size from 32 to 128 and increasing the associativity of both L1 instruction and data caches from 2 to 4, the program now takes much longer to execute. I believe this is because it now requires more time to access each larger block. The hit rate of the L1 cache has increased since more cache lines can now be accommodated in each set. Additionally, the hit rate of the L1 instruction cache has shown a more significant increase compared to the L1 data cache. I attribute this to the fact that there are typically more instruction commands, so the impact of the increased block size may be more obvious in the instruction cache

```
l1_instruction_cache = WriteBackCache(total_size=1024, block_size=128, set_associativity=4)
l1_data_cache =  WriteBackCache(total_size=1024, block_size=128, set_associativity=4)
l2_cache = WriteBackCache(total_size=16384, block_size=128, set_associativity=l2_associativity)
```

Assoc. Level	L1I accesses	L1I misses	L1I hit rate	L1D accesses	L1D misses	L1D hit rate	L2 accesses	L2 misses	L2 hit rate	AMAT
1	757341	38731	94.89%	242661	570	65.55%	39615	13065	67.02%	6.11
4	757341	38731	94.89%	242661	570	65.55%	39615	10213	74.22%	6.11
16	757341	38731	94.89%	242661	570	65.55%	39615	9585	75.80%	6.11
64	757341	38731	94.89%	242661	570	65.55%	39615	9698	75.52%	6.11
128	757341	38731	94.89%	242661	570	65.55%	39615	9402	76.27%	6.11

- After reducing the cache block size to 32 and increasing the associativity of the L1 cache, we observed a lower hit rate for the L2 cache compared to Step 5. This could be attributed to the fact that only the block size argument for the L2 cache was modified, potentially resulting in multiple addresses being mapped to the same index and leading to competition between data blocks. Additionally, smaller blocks may result in wasted locality as they may not fully store complete data within one cache line. As for the increased associativity of the L1 cache, there was no significant growth in hit rate. This might be because the number of blocks per set also increases with higher associativity, which can result in more conflicts within the set and frequent replacements.

```
l1_instruction_cache = WriteBackCache(total_size=1024, block_size=32, set_associativity=16)
l1_data_cache =  WriteBackCache(total_size=1024, block_size=32, set_associativity=16)
l2_cache = WriteBackCache(total_size=16384, block_size=32, set_associativity=l2_associativity)
```

Assoc. Level	L1I accesses	L1I misses	L1I hit rate	L1D accesses	L1D misses	L1D hit rate	L2 accesses	L2 misses	L2 hit rate	AMAT
1	757341	79738	89.47%	242661	1265	65.26%	82063	25118	69.39%	11.53
4	757341	79738	89.47%	242661	1265	65.26%	82063	18434	77.54%	11.53
16	757341	79738	89.47%	242661	1265	65.26%	82063	17620	78.53%	11.53
64	757341	79738	89.47%	242661	1265	65.26%	82063	17544	78.62%	11.53
128	757341	79738	89.47%	242661	1265	65.26%	82063	17407	78.79%	11.53

Step 7:

I implemented a victim cache based on the Least Recently Used (LRU) technique. When either the L1 or L2 cache becomes full, the least recently used data needs to be evicted. The program checks if the victim cache is full; if not, the victim cache stores the evicted data from L1 or L2. If the victim cache is also full, it evicts its least used data to make room for the data abandoned by L1 or L2. The order in which the program checks if the data is in a specific cache is L1 -> Victim -> L2. It's possible that the victim cache, due to its large total size and block size settings, ends up storing every evicted data, thereby resulting in 0 accesses to L2.