CS423A

MULTI CORE AND MULTIPROCESSOR ARCHITECTURE

Assignment 1

Multi Level Cache Simulator

GROUP 11

Divyansh Garg
190309

Harshit Bansal
200428

Instructor:
Dr. Mainak Chaudhury

February 17, 2023

1 Introduction

In this assignment we have simulated L1 cache misses for L2, L3 cache hierarchy for various applications. The simulator is programmed for three different inclusion policies, i.e. *Inclusive*, *Exclusive* *Non-Inclusive Non-Exclusive* (*NINE*).

The three configurations for the assignment are:

- Part 1: 8-Way L2 cache (LRU), 16-Way L3 cache (LRU)
- Part 2: 8-Way L2 cache (LRU), Fully-Associative L3 cache (LRU)
- Part 3: 8-Way L2 cache (LRU), Fully-Associative L3 cache (Belady-MIN)

We have to report the cache misses for six SPEC CPU 2006 applications: bzip2, gcc, gromacs, h264ref, hmmer, sphinx3.

2 Simulation Results

2.1 PART 1: Set Associative, LRU eviction policy

L2 Cache misses:

Table 1: L2 cache misses for inclusive, NINE, exclusive policy

Trace Files	Inclusive	NINE	Exclusive
bzip2	5398166	5397576	5397576
gcc	3036461	3029809	3029809
gromacs	336851	336724	336724
h264ref	969678	965624	965624
hmmer	1743421	1735322	1735322
sphinx3	8820349	8815130	8815130

L3 Cache misses:

Table 2: L3 cache misses for inclusive, NINE, exclusive policy

Trace Files	Inclusive	NINE	Exclusive
bzip2	1446388	1445846	889221
gcc	1373402	1366248	1242824
gromacs	170531	170459	159302
h264ref	342146	333583	143681
hmmer	391226	376344	300046
sphinx3	8207362	8205144	7220776

2.2 PART 2: Fully Associative L3 Cache

L3 Cache misses classified into cold, capacity and conflict misses:

Table 3: L3 Cache misses for inclusive (LRU)

Trace Files	**L2 miss**	**L3 miss**	**Cold**	**Capacity**	**Conflict**
bzip2	5397991	1361401	119753	1241648	84987
gcc	3035483	1369924	773053	596871	3478
gromacs	336940	169368	107962	61406	1163
h264ref	969235	335880	63703	272177	6266
hmmer	1743222	377024	75884	301140	14202
sphinx3	8818925	8387248	122069	8265179	-179886

Table 4: L3 Cache misses for inclusive (Belady)

Trace Files	**L2 miss**	**L3 miss**	**L3 Cold**	**L3 Capacity**
bzip2	5351953	536836	119753	417083
gcc	2946879	939289	773053	166236
gromacs	321640	143254	107962	35292
h264ref	960590	111605	63703	47902
hmmer	1732342	153447	75884	77563
sphinx3	8680310	3068580	122069	2946511

3 Simulation Analysis

3.1 Part 1

3.1.1 L2, L3 misses for inclusive, NINE and exclusive policies.

The results of L2 cache misses and L3 cache misses are shown in Table 1 and Table 4 respectively.
The first observation that can be made is that for any of the trace files, the number of L3 cache misses are significantly less than the L2 cache misses. This can be attributed to large L3 cache capacity and associativity. And since

$$\text{L3 cache hits} = \text{L2 cache misses - L3 cache misses}$$

We can easily interpret that there were considerable number of L3 cache hits.

3.1.2 L2 misses: NINE = Exclusive, policies

This observation can be proved if we show that at all time L2 cache holds same addresses' data for both NINE and exclusive policies. To do this we analyze all possibilities:

- **L2 Hit:** Nothing changes here.
- **L3 Hit:** The block gets added to L2 Cache. Although the block is invalidated from L3 cache in exclusive policy but it doesn't change anything about L2 misses.
- **L3 Miss:** The block is brought from memory and added to L2 cache in both policies. If there is a L3 cache block eviction, the corresponding block need not be invalidated in the L2 cache, for both policies.

Hence , in all the cases, if the configuration is same after k^{th} step, it will be same after $(k+1)^{th}$ step. And configuration is same at the start i.e $k = 0$.

3.1.3 L2 misses: Inclusive > NINE = Exclusive, policies.

As stated in the problem statement, an issue with Inclusive policy is that if a block repeatedly receives hits in the L2 cache, it will have a high rank in the LRU indexing and likely won't be removed from the L2 cache. However,

its age in the L3 cache will decrease and it may eventually be evicted from there. As per the Inclusive policy, the block will also need to be removed from L2 cache, which could result in unnecessary L2 cache misses.

3.1.4 L3 misses: Inclusive $>$ ($\approx$) NINE $>>$ Exclusive, policies.

Inclusive policy has a slightly more L3 misses than NINE policy, that is because of replication of L2 cache data in L3. In Inclusive When a block is evicted from L3 cache, it is also evicted from L2 cache if it is present there. This does not happen in NINE. Thus when that block is accessed again, it causes L3 miss in Inclusive policy, but is a L2 hit in NINE.

Exclusive policy has far less number of L3 misses than both Inclusive and NINE policies, that is because in Exclusive policy, L2 and L3 caches combined store far more number of distinct cache blocks than those in Inclusive and NINE. For example, in case of L3 Hit, that block is evicted from L3 and copied in L2 for exclusive case, whereas in both Inclusive and NINE cases, block in L3 remains. Thus Exclusive allows more space for new blocks.

3.2 Part 2

3.2.1 Miss Categorization in L3 Cache for LRU

Misses in L3 cache are categorized into Cold, Capacity and conflict misses. Cold misses represent the number of distinct addresses accessed by the application. Thus more the cold misses, larger the working set. Capacity misses might occur due to premature eviction of block from L3 and L2 due to LRU eviction policy since the age of blocks accessed in L2 is not updated in L3. Another reason might be due to large access gap for a address in the trace file.

- $bzip2, gromacs$, and $sphinx3$ have similar number of cold misses, indicating similar working set size, but their capacity misses are varying by a considerable factor with $gromacs$ having least and $sphinx3$ having the largest. This is because in $gromacs$, access gap is considerably less for a address in working set. While $sphinx3$ has a large access gap, which might be exceeding the L3 cache capacity.

- $sphinx3$ has negative conflict misses, which shows that fully associative L3 cache is performing poorly than set associative L3 cache. This

can be explained by considering an address A, whose two consecutive accesses are very far apart. Now when the L3 cache becomes full, A will be evicted from L3 and L2 in fully associative cache. Thus next access to A will give a miss. Same pattern would repeat. Now set associative cache is avoiding this kind of misses by mapping A to a set which is not so hot as others. Thus A is a lot less likely to be replaced now and thus would not miss. Also, the block evicted by set associative cache might not be accessed in future.

3.2.2 LRU and Belady Comparison for fully associative L3

- Obviously, cold misses will be same in both the cases.
- The capacity misses in case of Belady are very less as compared those in LRU. The reason being, LRU evicts those cache blocks which were accessed long ago but will be accessed soon after the eviction, thus leading to cache miss. Whereas Belady avoid these type of misses by evicting only those blocks which will be accessed farthest from current access or will never be accessed at all.

4 Execution Time Optimization

We have tried to optimize various subsystems in the code including LRU, Belady, cache lookup and memory allocation. After all this, we were able to achieve maximum execution time of 2.2 seconds for part 1 and part 2 LRU. For Optimal algorithm, *sphinx*3 takes 10 secs to run.