CS622A

Advanced Computer Architecture

ASSIGNMENT 1

GROUP 10

Mohit Kumar
20111034

Rohit Raj
20111051

Instructor:
Dr. Mainak Chaudhuri

October 6, 2020

1 How To Run The Code

- Suppose you are in the folder containing the code. Our code expect that the trace file is present in : '../traces' . For example the 'File'(name of the folder) is the folder containing two sub folder i.e traces and assignment1.

- To compile the code for question 1 :
  run the following code $ g++ ques1.cpp -o ques1

- To execute the code for question 1 :
  for example just write gcc(as the first argument, if the trace file named as $gcc.log_l1misstrace$) and 2(as the second argument indicating the number of part of that trace file)
  run the following code $ ./ques1 gcc 2

- To compile the code for question 2 part a :
  run the following code $ g++ lru.cpp -o lru

- To execute the code for question 2 part a :
  for example just write gcc(as the first argument, if the trace file named as $gcc.log_l1misstrace$) and 2(as the second argument indicating the number of part of that trace file)
  run the following code $ ./lru gcc 2

- To compile the code for question 2 part b :
  run the following code $ g++ belady.cpp -o belady

- To execute the code for question 2 part b :
  for example just write gcc(as the first argument, if the trace file named as $gcc.log_l1misstrace$) and 2(as the second argument indicating the number of part of that trace file)
  run the following code $ ./belady gcc 2

2 Cache Performance Results

2.1 Set Associative, LRU Replacement policy

2.1.1 Inclusive Policy

INCLUSIVE	L2			L3	
	Access	Hits	Misses	Hits	Misses
h264ref	2348573	1378895	969678	627532	342146
hmmer	3509765	1766344	1743421	1352195	391226
gromacs	3431511	3094660	336851	166320	170531
sphinx3	10753447	1933098	8820349	612987	8207362
gcc	14610811	11574350	3036461	1663059	1373402
bzip2	10657627	5259461	5398166	3951778	1446388

2.1.2 Exclusive Policy

EXCLUSIVE	L2			L3	
	Access	Hits	Misses	Hits	Misses
h264ref	2348573	1382949	965624	821939	143685
hmmer	3509765	1774443	1735322	1435276	300046
gromacs	3431511	3094787	336724	177418	159306
sphinx3	10753447	1938317	8815130	1594353	7220777
gcc	14610811	11581002	3029809	1786983	1242826
bzip2	10657627	5260051	5397576	4508354	889222

2.1.3 NINE Policy

NINE	L2			L3	
	Access	Hits	Misses	Hits	Misses
h264ref	2348573	1382949	965624	632041	333583
hmmer	3509765	1774443	1735322	1358978	376344
gromacs	3431511	3094787	336724	166265	170459
sphinx3	10753447	1938317	8815130	609986	8205144
gcc	14610811	11581002	3029809	1663561	1366248
bzip2	10657627	5260051	5397576	3951730	1445846

2.2

2.2.1 Fully Associative L3 Cache,LRU Replacement Policy

INCLUSIVE	LRU			
L3	Total Misses	Cold	Capacity	Conflict
h264ref	342146	63703	272177	6266
hmmer	391226	75884	301140	14202
gromacs	170531	107962	61406	1163
sphinx3	8207362	122069	8265179	-179886
gcc	1373402	773053	596871	3478
bzip2	1446388	119753	1241648	84987

2.2.2 Fully Associative L3 Cache,Belady Min Replacement Policy

INCLUSIVE	LRU			
L3	Total Misses	Cold	Capacity	Conflict
h264ref	342146	63703	272333	6110
hmmer	391226	75884	86995	228347
gromacs	170531	107962	161096	-98527
sphinx3	8207362	122069	3576891	4508402
gcc	1373402	773053	22731	577618
bzip2	1446388	119753	642037	684598

3 Cache Performance Analysis

3.1 Set Associative, LRU Replacement Policy

3.1.1 Number of hit/miss in L2 cache is equal in both Exclusive and NINE policy

- Initially in both policy L2 cache is empty.
- So, if the there is miss in L1 cache , it is checked in L2 cache. When there is miss in L2 cache, the cache block is brought either from L3 or memory in both policy.
- If the block is brought from memory to L2 cache , in case of Exclusive ,that block is fill only in L2 cache to maintain the exclusion property.
- If the block is brought from memory to L2 cache , in case of NINE ,that block is fill in both L2 cache and L3 cache.
- If the block is brought from L3 cache to L2 cache , in case of Exclusive ,that block is evicted from L3 cache.
- If the block is brought from L3 cache to L2 cache , in case of NINE ,that block is still present in L3 cache.
- In case of eviction of cache block in L3 cache, it is not back-invalidated in NINE policy and in case of Exclusive, it was not present initially to maintain the exclusion policy.
- So,in L2 cache, both policy have same block addresses.
- Hence,the number of hit/miss in L2 cache is equal in both Exclusive and NINE policy.

3.1.2 Number of misses in L2 cache is higher in Inclusive policy

- In case of Inclusive policy,if the cache block is invalidated from L3 cache ,it is back-invalidated from L2 cache to maintain inclusion property.

- This is a serious problem with Inclusive Policy . Suppose there is block which is constantly accessed in L2 cache . But the LRU replacement policy ,does not update the hit of L2 cache in L3 cache replacement buffer . So, that block will have old age in L3 cache. If due to any reason,that block is selected as victim for replacement in L3 cache. To maintain the inclusion property, it will also invalidated from L2 cache. And again when the running program will try to access the evicted block from L2 cache ,it will cause a miss.

- Due to above reasons , the number of misses in L2 cache is higher in case of Inclusive policy.

3.1.3 Number of Misses in L3 cache is lower in Exclusive Policy

- At any point of time , the cache following Exclusive Policy will have more data as compare to NINE and Inclusive Policy.As the there is no common data between L2 and L3 cache.

- In particular the number of distict block address is greater in Exclusive policy.

- Therefore, the number of misses in L3 cache(i.e the amount of data needed to bring from memory)is lower.

3.2 Fully Associative L3 cache

3.2.1 Reason for negative amount of conflict misses

- The method of calculating the number of conflict misses for L3 cache by looking the whole access trace of L1 misses is not fully correct. But this is the best we got . Because we can't calculate the actual access trace of L3 cache anyhow.

3.2.2 Number of cold misses

- The number of cold misses is equal to distinct number of block addresses accessed.

- Whenever ,our program want to access a block address for the first time ,it is brought from memory to our cache . This is responsible for cold misses.