CS622A

Advanced Computer Architecture

Assignment 1

L2-L3 Cache Simulator

GROUP 24

Aman Aryan
20111009

Mani Kant Kumar
20111030

Instructor:
Dr. Mainak Chaudhuri

1 Problem

A program to simulate a two-level cache hierarchy which counts the number of misses and hits when L1 cache miss trace is passed through them. Let us call the two levels of the hierarchy L2 and L3. The L2 cache should be 512 KB in size, 8-way set associative, and have 64 bytes block size. The L3 cache should be 2 MB in size, 16-way set associative, and have 64 bytes block size. The aim is to report the number of L2 and L3 cache misses for each of the six applications for the following three cases. i.e *Inclusive*, *Exclusive* & *Non-Inclusive Non-Exclusive* (*NINE*).

1.1 Instructions to Run the Simulator

The commands to compile and run the code are explained below. First put all the trace files in **traces** folder.

1. `To run code individually`:

   ```
   g++ cache_sim_incl.cpp -o inclusive
   ./inclusive bzip2.log_l1misstrace 2

   g++ cache_sim_exlcl.cpp -o exclusive
   ./exclusive bzip2.log_l1misstrace 2

   g++ cache_sim_nine.cpp -o nine
   ./nine bzip2.log_l1misstrace 2
   ```

   Here we need to pass the name of trace file and number of trace files as argument. Similarly we can simulate for all the applications.

2. `To run entire code`:

   ```
   g++ driver.cpp -o driver
   ./driver > report.txt
   ```

   Reads the traces, compiles and runs the cache simulator for this problem. L2, L3 cache hits and misses are printed in report.txt

Note: The miss traces MUST be present in a directory named *traces* in the root folder to run entire code and may need to create object file for each policy. It may be needed to build essential if required : sudo apt-get install build-essential

1.2 Simulation Results

INCLUSIVE	**L2**		**L3**	
	Hits	Misses	Hits	Misses
bzip2	5249186	5408441	3996823	1411618
gcc	11565075	3045736	1576313	1469423
gromacs	3088119	343392	148798	194594
h264ref	1376960	971613	544936	426677
hmmer	1765829	1743936	1362859	381077
sphinx3	1912164	8841283	528871	8312412

EXCLUSIVE	**L2**		**L3**	
	Hits	Misses	Hits	Misses
bzip2	5260051	5397576	4493598	903978
gcc	11581002	3029809	1785780	1244029
gromacs	3094787	336724	163925	172799
h264ref	1382949	965624	814663	150961
hmmer	1774443	1735322	1427112	308210
sphinx3	1938317	8815130	1582358	7232772

NINE	**L2**		**L3**	
	Hits	Misses	Hits	Misses
bzip2	5260051	5397576	4005402	1392174
gcc	11581002	3029809	1576803	1453006
gromacs	3094787	336724	149035	187689
h264ref	1382949	965624	551833	413791
hmmer	1774443	1735322	1371571	363751
sphinx3	1938317	8815130	524910	8290220

1.3 Simulation Analysis

Following are some of the observations on the simulation results.

1.3.1 L2 Miss is more in Inclusion compared to Exclusion, NINE

When an access hits in the L2 cache, the LRU order of the block is updated only in the target L2 cache set. The LRU order of the block remains unchanged in the L3 cache. As a result, a block that is receiving a lot of hits in the L2 cache may become LRU in its L3 cache set and may get evicted. In an inclusive L3 cache, this will invalidate the block from the L2 cache which would unnecessarily increase L2 misses. This is a big drawback of an inclusive cache hierarchy as also mentioned in problem statement.

1.3.2 L2 Hits/Misses (Exclusive) == L2 Hits/Misses (NINE)

This can be analyzed considering all the possibilities to verify this observation:

- **L2 Hit:** Nothing changes here.
- **L2 Miss & L3 Hit:** The block gets added to L2 Cache in both policy. However the block is invalidated from L3 in exclusive case so there is nothing to cause difference in L2 cache.
- **L3 Miss:** The block is brought from memory and added to both L2 and L3 cache in NINE policy and only added in L2 cache in case of exclusion.

Hence, it will cause the same number of misses in L2 cache.

1.3.3 L3 Misses: NINE > Exclusive

In exclusion policy the intersection of L2 and L3 cache blocks is NULL which may not be NULL in case of NINE. So exclusion policy covers a large range of blocks which is not present in L2 that causes more hits and significantly lesser L3 misses in Exclusive case.

2 Problem

In this problem misses in the case of an inclusive L3 (case A) is classified into into cold, conflict, and capacity misses assuming LRU replacement policy. For the fully associative cache, both LRU and Belady's optimal policies are implemented.

2.1 Simulation Results

2.1.1 Set Associative L3 Cache

INCLUSIVE	**LRU**		
L3	**Cold**	**Capacity**	**Conflict**
bzip2	119753	1271376	20489
gcc	773053	694274	2096
gromacs	107962	84161	2471
h264ref	63703	193205	169769
hmmer	75884	293384	11809
sphinx3	122069	7882266	308077

2.1.2 Fully Associative L3 Cache

INCLUSIVE	**LRU**		
L3	**Cold**	**Capacity**	**Conflict**
bzip2	119753	1236957	0
gcc	773053	651433	0
gromacs	107962	77993	0
h264ref	63703	456298	0
hmmer	75884	297356	0

2.2 Instructions to Run the Simulator

The commands to compile and run the code are explained below.

1. `To run code individually`:
   g++ 2_cache_sim_sa_lru.cpp -o 2_inclusive_sa
   ./2_inclusive_sa bzip2.log_l1misstrace 2

   g++ 2_cache_sim_fa_lru.cpp -o 2_inclusive_fa
   ./2_inclusive_fa bzip2.log_l1misstrace 2

   Here we need to pass the name of trace file and number of trace files as argument that we want to execute. Similarly we can simulate for all the applications.

Reads the traces, compiles and runs the cache simulator for this problem. Type of L3 cache misses are printed in stdout.

Note: To run code individually for specific applications, code and trace files should be in the same folder .
Note: Since this problem also generates the result of previous problem so it will take a reasonable time to execute the entire program. In case of Belady the code is taking unreasonable amount of time in my case so couldn't report the result.

2.3 Simulation Analysis

Following are some of the observations on the simulation results.

Cold: miss occur when the first access to a block happens. It is the count of unique block references. So compulsory misses should be same in all types of direct mapped, set associative and fully-associative cache.

Conflict: miss occurs when still there are empty lines in the cache, block of main memory is conflicting with the already filled line of cache, ie., even when empty place is available, block is trying to occupy already filled line. So conflict misses should be zero in fully-associative mapped cache as no block of main memory tries to occupy already filled line.

Capacity: miss occurs when all lines of cache are filled. In fully-associative, the number of capacity misses can also be verified by excluding cold misses from total L3 misses since conflict misses are zero.