Assignment 2- Cache Simulator
CS4110 -Computer System Design Lab

Aritra Ghosh(CS11B062),Sasidhar Sanapala(CS11B024)

14 September,2014

1 Introduction

The main aim of the assignment was to design a Cache Simulator in C++. The sequence of memory accesses to be used in the simulator are obtained from Intel Pin tool which runs the Matrix multiplication program for different values of n. This included varying the different parameters like size of cache,block size and associativity and making statistics of them. The Cache Replacement Policy used can be any one of LRU(Least Recently used), Least Frequently Used(LFU) and RR(Random Replacement).Also the writing policy used is write back and the caches are inclusive.The correctness of the code was ensured by checking this property after every read and write simulation.Another important assumption is the inclusion of both instructions and data in a single cache.

1.1 Hit and Miss

A cache miss is said to take place when something is looked up in the cache and is not found. The cache did not contain the entry being looked up. A cache hit is said to take place when something is looked up in cache and it was found in the memory. Also by this definition, the total number of memory accesses for a particular level of cache may not be equal to the sum of the hits and misses.

2 Statistics

All the below are run for the cache Configuration given with the assignment.

Table 1: LRU Policy

Dimension	Miss Ratio		Cache Hits		Memory Accesses	
	L1	L2	L1	L2	L1	L2
8	0.0226	0.467	21089	19141	926485	40906
16	0.018	0.468	21107	19178	1148936	40939
32	0.0077	0.469	21456	19591	2776009	41722
64	0.00158	0.454	24133	20800	15233071	45806
128	0.0027	0.368	308302	181505	113018066	492199
256	0.0027	0.4826	2415139	2296908	886821037	4723415
512	-	-	-	-	-	-
1024	-	-	-	-	-	-

Table 2: LFU Policy

Dimension	Miss Ratio		Cache Hits		Memory Accesses	
	L1	L2	L1	L2	L1	L2
8	0.0394541	0.453185	39299	35716	996069	78811
16	0.0325041	0.466382	39702	38234	1221446	81980
32	0.0174759	0.461974	50105	46622	2867085	100919
64	0.0200852	0.486949	318517	315012	15858295	646910
128	0.0251986	0.493333	2990990	2987440	118696501	6055631
256	0.028674	0.492054	26899198	26895667	938102710	54660022
512	-	-	-	-	-	-
1024	-	-	-	-	-	-

Table 3: RR Policy

Dimension	Miss Ratio		Cache Hits		Memory Accesses	
	L1	L2	L1	L2	L1	L2
8	0.024436	0.439196	22874	19741	936077	44948
16	0.020289	0.435305	23521	20051	1159296	46062
32	0.00840618	0.439427	23420	20320	2786046	46242
64	0.00176109	0.424721	26847	22232	15244570	52345
128	0.00338736	0.198869	383284	96991	113151338	487714
256	0.00381314	0.428988	3389811	2565140	888981272	5979517
512	-	-	-	-	-	-
1024	-	-	-	-	-	-

3 Observation

All the analysis assumes constant size of cache.We see that as n increases the miss ratio in L1 first decreases and then increases. Initially, the size of the cache is much larger than that of matrices(for small n), so there are constant(approximately) compulsory misses and very few conflict misses as compared to when n is large.Once the whole matrix is fetched into the cache, most of the queries result in hits and the number of hits($O(n^3)$) are much higher than the number of misses.With increasing size of matrix,the conflict misses increase. Eventually,the rate of increase of conflict misses outnumbers the rate at which hits increase and thus miss ratio begins to increase. After a certain threshold, the matrix does not fit into the cache anymore and thus there will be a lot of misses.From here capacity misses comes into picture. In L2 we see that the miss ratio remains almost constant except for a decrease in the middle. This is because the inital compulsory misses in L1 are also misses in L2. Also after the compulsory misses, the matrix almost always resides in the cache(size of cache is much larger than size of all three matrices combined) and thus there are very few queries in L2.This can be clearly seen in the table where

the number of memory accesses in L2 is less than that in L1. For every miss in L2 there are two memory accesses(since we include the fetch from memory as a memory access). Thus miss ratio is always less than 0.5

We also see that LRU works better than RR which works better than LFU.This is because in case of LFU, once a row is brought, it is use it many times(since we multiply each row by different column).Once the row is multiplied, that specific row is never used again.It becomes very difficult to remove it once it is brought into the cache and does not serve any purpose. Thus it tends to perform even worse than RR.