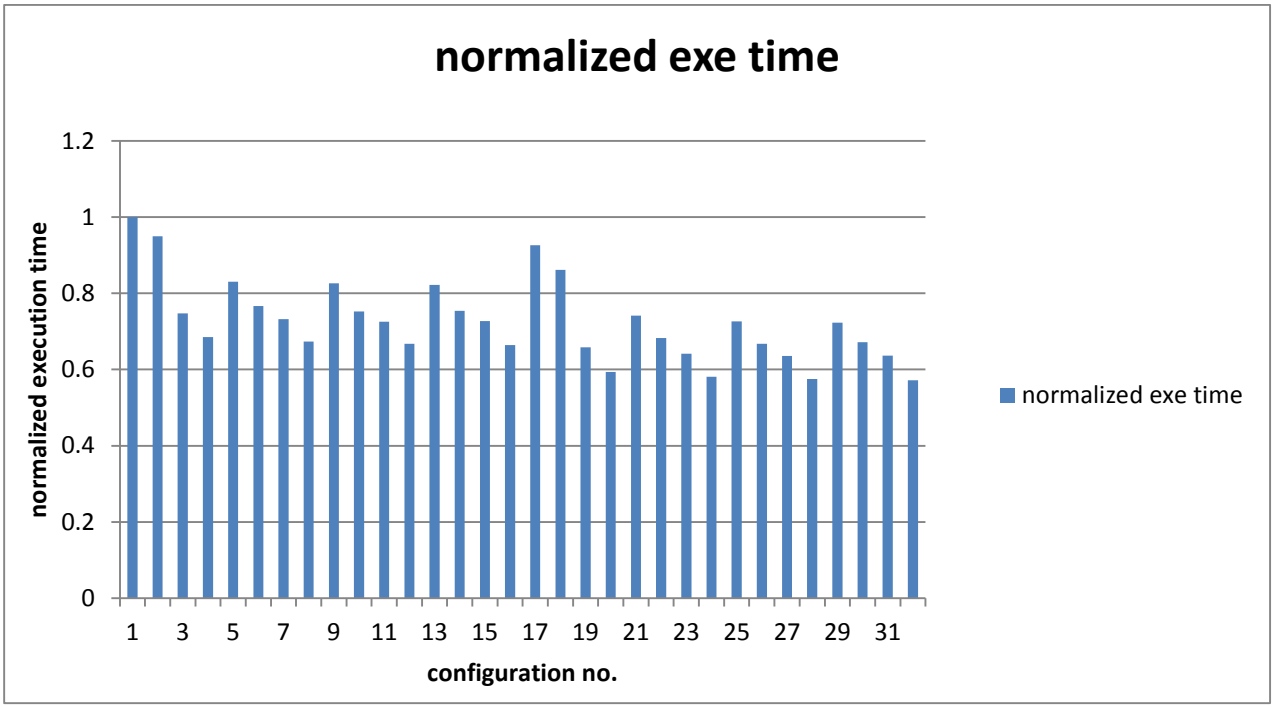
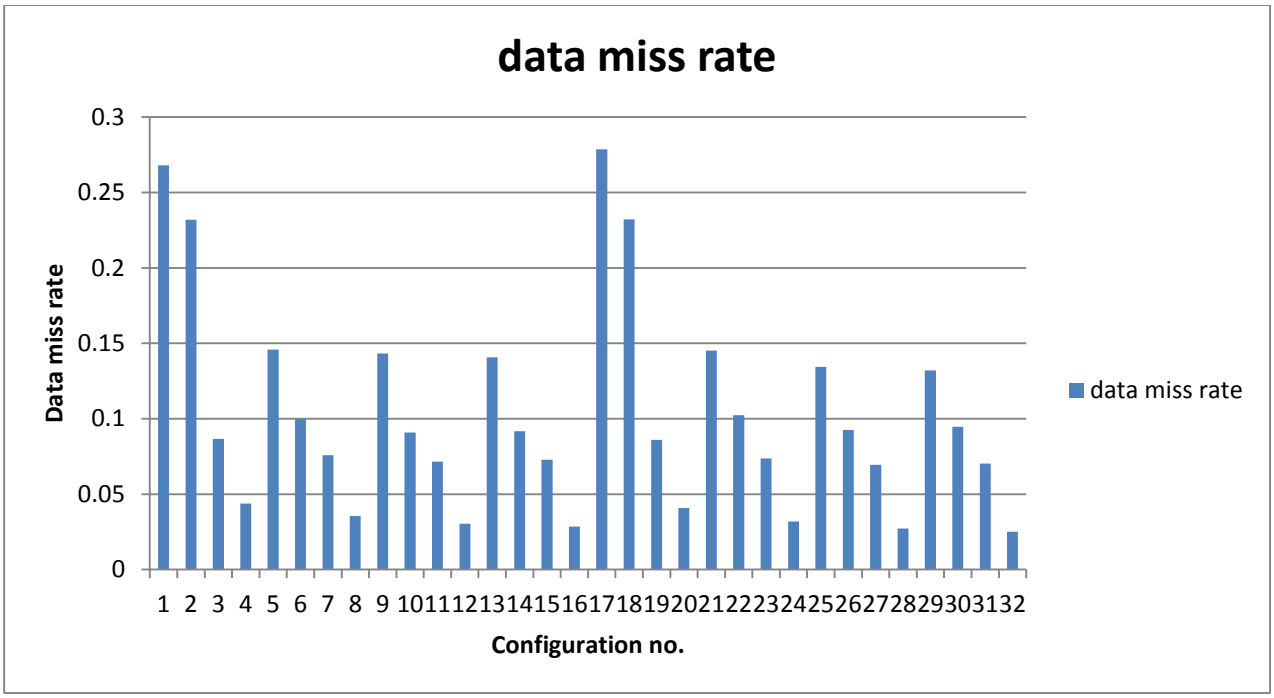
CSE-420 : Computer Architecture I (Fall 2015)
Project 3 Report

Problem-1:

a. Question-1

i. Graphs



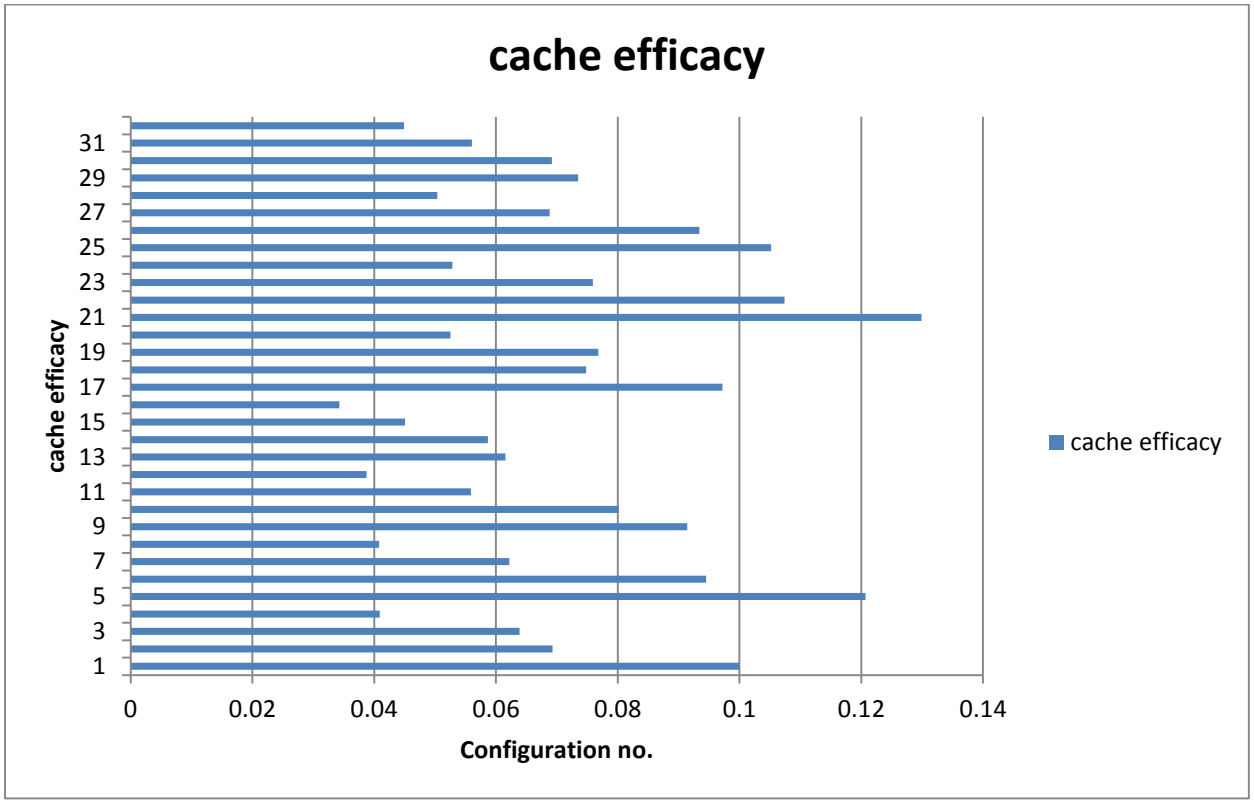


ii. Explanation:

L1 data cache miss rate is directly proportional to execution time. Because as miss rate increases the access time to the memory increases which will result in increase of execution time.

b. Question-2

(a) Cache Efficacy graph



(b) Most effective cache configuration:

We have observed that configuration-21 has highest cache efficacy but at the same time has comparatively higher miss rate. Hence, we tried to obtain a tradeoff between miss rate and cache efficacy. To obtain tradeoff we compared cache efficacy and data miss rate for various configurations in order of decreasing cache efficacy as conf-21, conf-5, conf-17, conf-22, conf-25, conf-1, conf-6, conf-9 and conf-26. Configuration-26 has been found as optimal configuration because it has lower miss rate and comparatively higher cache efficacy.

Conf-26> L1d_size=2kB, L1d_assoc=4, cache line size=32 bit.

Problem-2:

a. Question-1

i. Working lru.cc file is been included along with report which indicates appropriate changes for the implementation of SRRIP policy.

ii. **SRRIP IMPLEMENTATION IN GEM 5:**

(a) Initializations Required to Implement SRRIP:

A base class 'CacheBlk' can be found in the header blk.hh, which holds all the variables like size, set etc., associated with each block. Here we initialize another variable called RRPV, so that it becomes a variable in each of the objects created.

In lru.cc, initially a big chunk of memory is allocated for the cache and is divided into respective blocks using a simple technique based on the number of sets and the associativity. Here the variables in each of the object blk are initialized. Here we initialize RRPV as 3.

(b) Changes required to implement replacement policy:

- We need to get rid of the MovetoTail and MovetoHead methods that are the integral parts of the LRU implemented in GEM5. These methods are not necessary for SRRIP. Hence, commented.
- In the method **Access block,** we can observe an analogy to a cache hit. When originally in LRU, we move the block to the head of the set, here we just make the RRPV variable of the block as 0.
- In method **Insert block,** we observe an analogy to a cache miss and insertion of a new block, in case of an LRU we simply perform the operation and move it to the head of the set. Whereas we perform the operation and set the RRPV as 2.
- In the method **FindVictim,** we can observe an analogy to a cache miss and no empty space for insertion. In case of an LRU, the default victim would be the block at the tail which is set.blk[assoc-1]. In case of SRRIP we write a small code to make sure the victim is passed on right.
- In the method **FindVictim,** We check for the block with RRPV 3 from 0 to assoc-1 and return the blk. If 3 is not found, we increment the RRPV of all the blocks in the set and repeat the check for RRPV==3. RRPV==3 is obtained by set flag is_3 (is_3=1). We continue this process till the flag is_3 is set and we return a specific block with RRPV as 3. [Actual code change]

(c) Difference between LRU and SRRIP:

The LRU algorithm cannot differentiate an insert (while cache miss) and a hit. In that way it just takes into account the period of reoccurrence and not the frequency. And because of the need to hold history, the implementation is complex.
SRRIP can differentiate an insert and a hit. This way it gives higher priority to recurring blocks and holds them for a longer time based on the precision of RRPV bit.

Effect on data patterns:
SRRIP is advantageous in cases where the Data Pattern is such that a Data block is re-referenced consecutively in near future and distant future. In this case LRU fails to understand the frequency in the repetition of the block and missed the block out when it is re-referenced for the third time.
Ex: a1,a2,a3,a1,b1,b2,b3,b4,b5,a1(LRU fails to Hit)

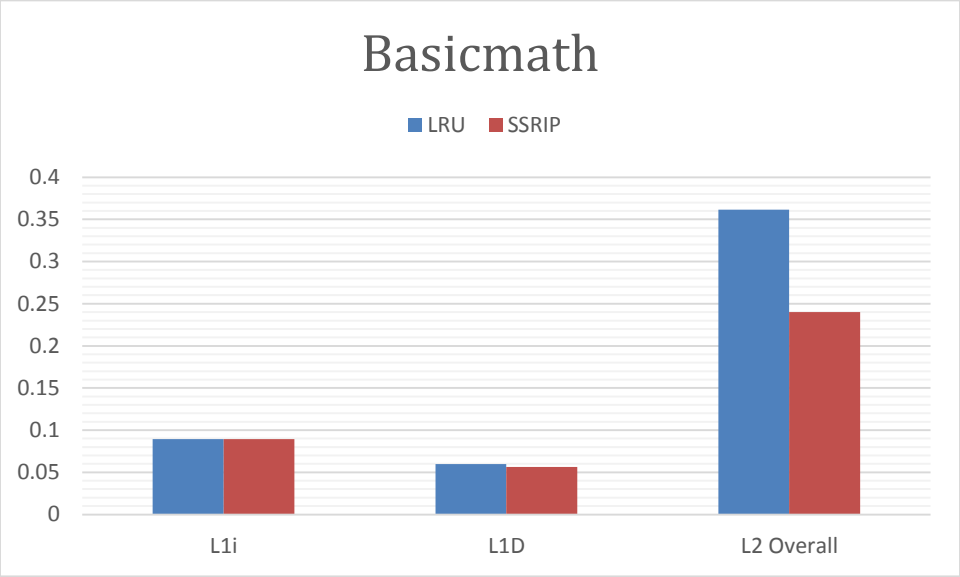
SRRIP can also disadvantages over LRU. One such disadvantage may be holding an unnecessary block for a very long time. In case we want to analyse a data pattern failure we have the following data pattern where continuous hits leave the four blocks with the following values and RRPVs.
[a1]1 [a2]1 [b1]0 [b2]0
In case we have a data pattern of a3, a1, a2, a3 (SRRIP fails)

We can come to an abstract conclusion saying, for data patterns with importance to frequency of occurrence can perform better in SRRIP and the ones that have importance to the recent occurrence of blocks can perform better in LRU.

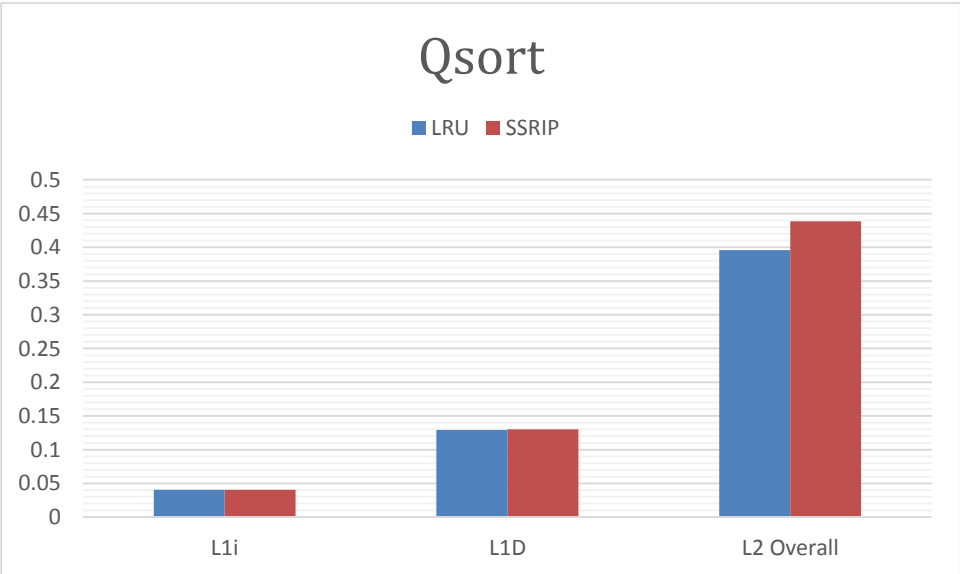
Problem-3:

Question-1

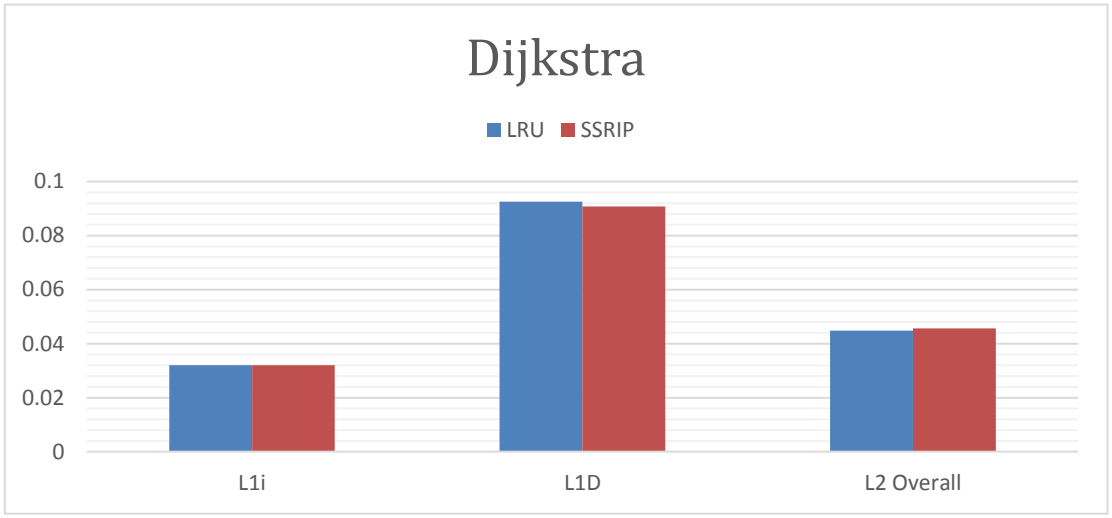
Basicmath	LRU	SRRIP
L1i	0.089333	0.089333
L1D	0.059819	0.056315
L2 Overall	0.361381	0.240133



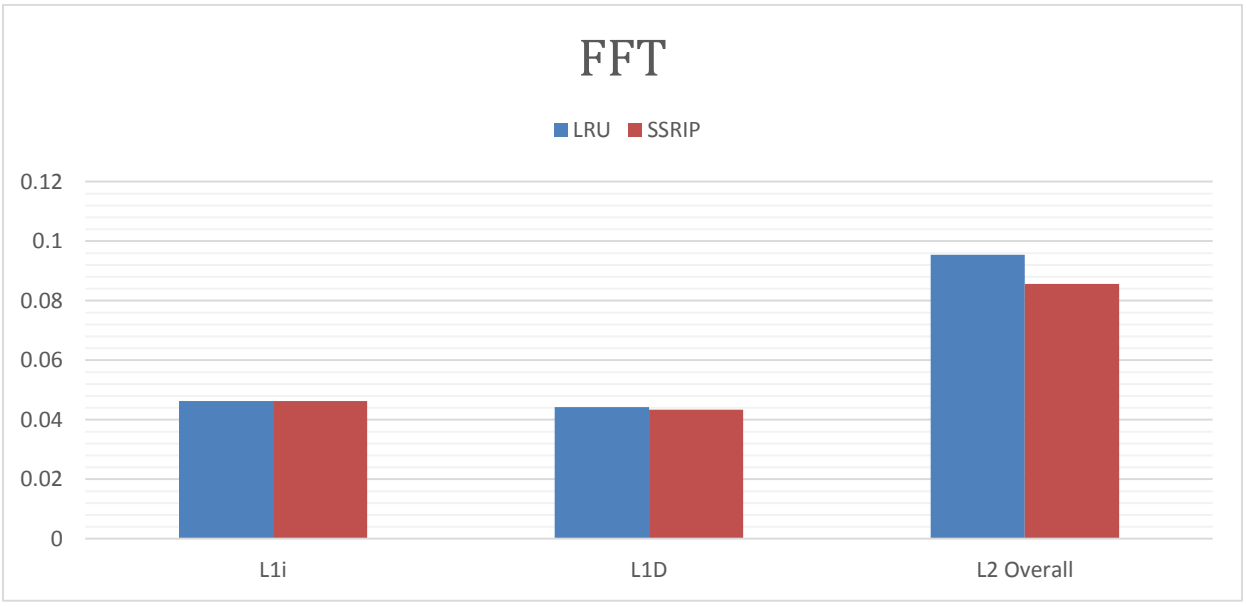
Qsort	LRU	SRRIP
L1i	0.040371	0.040371
L1D	0.129263	0.130025
L2 Overall	0.395675	0.438584



Dijkstra	LRU	SRRIP
L1i	0.032045	0.032045
L1D	0.092527	0.090729
L2 Overall	0.044761	0.045629



FFT	LRU	SRRIP
L1i	0.046255	0.04625
L1D	0.044288	0.043357
L2 Overall	0.095409	0.08561



These observations showed expected decrease in the Cache miss rate on L2, L1i and L1d. Here we have an exception in Qsort, cache miss rate of the L1d and L2 overall cache miss rate seem to be slightly lesser in LRU than the implemented SRRIP. The reason behind this may be because the instruction set of Qsort calculation may have the data pattern in such a way favored better by LRU. The loops In the Qsort program may have initial continuous hits and no recurrence in even a slightly longer run for many blocks. In this case the LRU may seem more advantageous as it gives priority to the MRU block.