**Name: Bhargava Kalla** **ASU ID: 1209323137**

**Declaration:** All the code changes and simulations for this assignment are done by my own but I have discussed and got clarifications regarding the concepts for the 4th and 5th questions from the TA and my fellow classmates.

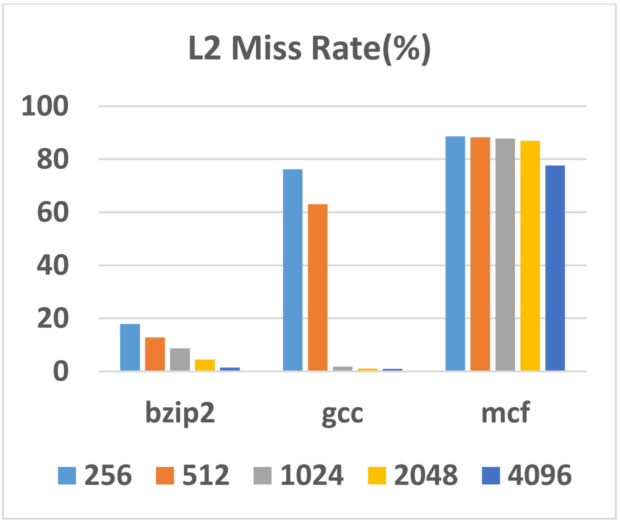
**Problem 1:**

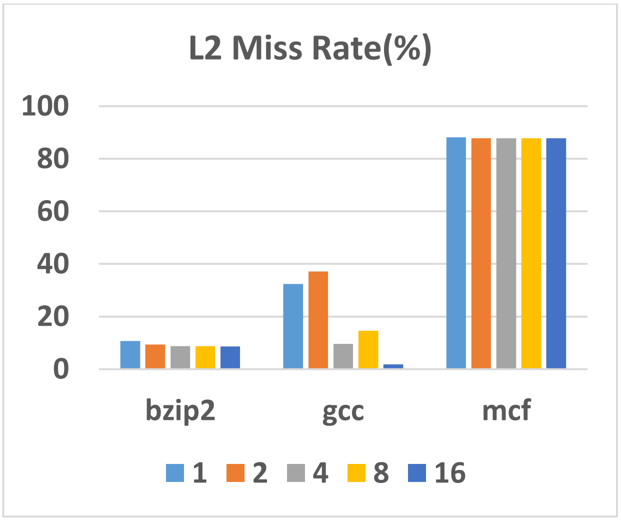
**1)** L2 Miss Rate (%) results for various sizes for the L2 cache having the block size as 64B and associativity fixed as 16. Size is varied from 256KB to 4MB in the steps of 256KB.

| **L2 miss rate (%)** | **256** | **512** | **1024** | **2048** | **4096** |
|---|---|---|---|---|---|
| **bzip2** | 17.81 | 12.77 | 8.67 | 4.47 | 1.42 |
| **gcc** | 76.21 | 62.97 | 1.76 | 1.01 | 0.98 |
| **mcf** | 88.63 | 88.27 | 87.78 | 86.89 | 77.67 |

**2)** L2 Miss Rate (%) results for various sizes for the L2 cache having the block size as 64B and size fixed at 16 and the associativity is varied from 1 to 16 by doubling the associativity at each step.

| **L2 miss rate (%)** | **1** | **2** | **4** | **8** | **16** |
|---|---|---|---|---|---|
| **bzip2** | 10.67 | 9.39 | 8.79 | 8.71 | 8.67 |
| **gcc** | 32.43 | 37.15 | 9.54 | 14.57 | 1.76 |
| **mcf** | 88.19 | 87.82 | 87.78 | 87.78 | 87.78 |





**3)** From the above plots, we can say that the miss rate decreases with increasing cache size. This is because it helps in reducing the capacity misses, incase if cache cannot hold all the blocks needed by a particular program. We can also say that miss rate decreases with increasing associativity. This is because it reduces the conflict misses, incase if two different instructions map to the same data record in the cache.

**4)** Optimal cache configuration derived from the above results can be as follows:

| | **Size** | **Associativity** |
|---|---|---|
| **bzip2** | 4MB | 16MB |
| **gcc** | 4MB | 16MB |
| **mcf** | 4MB | 16MB |

**Problem 2: Dynamic Insertion Policy(DIP)**

| | **Insertion** | **Hit (Re-reference)** |
|---|---|---|
| **LRU** | MRU position | MRU position |
| **LIP** | LRU position | MRU position |
| **BIP** | Bimodal throttle parameter, ε = 1/32. It means that for 1 out of 32 misses insert at LRU position and for remaining 31 misses insert at MRU position. | MRU position |
| **DIP** | If the dedicated set type is LRU then MRU position. If the dedicated set type is BIP then bimodal throttle parameter, ε = 1/32 that is in 1 out of 32 misses insert at LRU position and for remaining 31 misses insert at MRU position. If dedicated set type is a follower then if MSB of policy selection counter is 0 then follow the same as LRU policy, else follow the same as BIP policy. | MRU position |

**Problem 3: Dynamic Re-Reference Interval Prediction Policy(DIP)**

| | **Insertion** | **Hit (Re-reference)** |
|---|---|---|
| **SRRIP** | In place of replaced block, whose RRPV counter value = 2. | Accessed cache line's RRPV counter value = 0. |
| **BRRIP** | ε = 1/32, which means that for 1 out of 32 misses insert with RRPV counter value = 2 and for remaining 31 misses insert with RRPV counter value = 3. | Accessed cache line's RRPV counter value = 0. |
| **DRRIP** | If the dedicated set type is SRRIP then insert with RRPV counter value = 2. If the dedicated set type is BRRIP then bimodal throttle parameter, ε = 1/32 that is in 1 out of 32 misses with RRPV counter value = 2 and for remaining 31 misses with RRPV counter value = 3. If dedicated set type is a follower set then if MSB of policy selection counter is 0 then follow the same as SRRIP policy, else follow the same as BRRIP policy. | Accessed cache line's RRPV counter value = 0. |

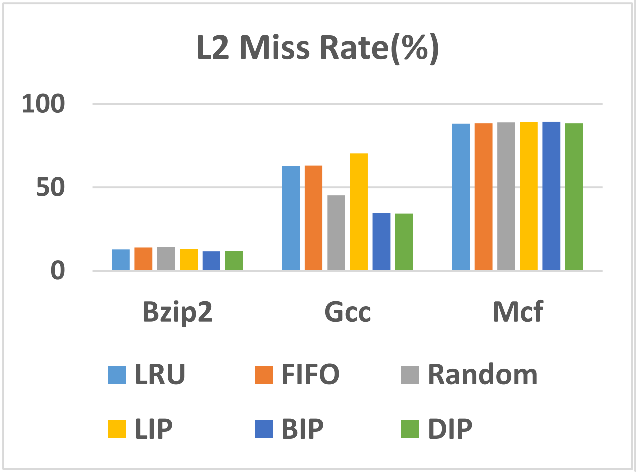
**Problem 4:**

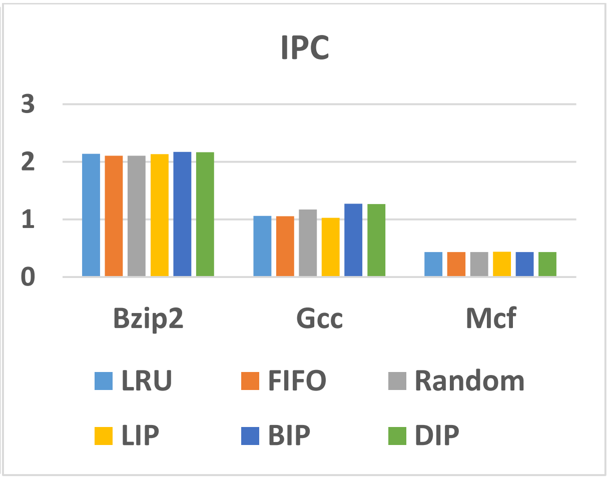
**Implementation:**

For LIP, BIP and DIP, I have added separate cases in the cache.c in the cache miss section and wrote conditions for hits for each policy. I have defined policy selection counter and BIP counter variables. I have provided the appropriate comments explaining the operations performed. I have used the set dueling mechanism as mentioned in the assignment and divided the total number of sets into three categories namely lru, bip and follower. Depending on the set type, I implemented the respective policies as described according to the papers.

1)

| | | LRU | FIFO | Random | LIP | BIP | DIP |
|---|---|---|---|---|---|---|---|
| **L2 miss rate (%)** | **Bzip2** | 12.77 | 13.99 | 14.08 | 12.95 | 11.62 | 11.52 |
| | **Gcc** | 62.97 | 63.14 | 45.17 | 70.37 | 34.43 | 34.25 |
| | **Mcf** | 88.28 | 88.42 | 88.99 | 89.25 | 89.43 | 88.31 |
| **IPC** | **Bzip2** | 2.1387 | 2.1075 | 2.1051 | 2.1315 | 2.1705 | 2.1672 |
| | **Gcc** | 1.060 | 1.0589 | 1.1739 | 1.0296 | 1.2720 | 1.2694 |
| | **Mcf** | 0.4361 | 0.4357 | 0.4352 | 0.4384 | 0.4371 | 0.4361 |





2) For the Bzip2 benchmark, the DIP policy reduces the miss rate by 9.78% and increases the IPC by 1.33%. For Gcc benchmark, it reduces the miss rate by 45.6% and increases the IPC by 19.75%. For Mcf benchmark, it increases the miss rate by 0.033% and there is no increase in IPC. Thus, DIP works better than LRU for Bzip2 and Gcc benchmarks but does not work well in case of Mcf. Thus, we can say that the new replacement policy (DIP) does not always work better than LRU. If the working set size is less than cache size, LRU performs better but if it is more, then thrashing of cache takes place. To protect cache from thrashing, we use LIP that preserves a part of the working set making it available for future hits and reduces the miss rate. If there is a change in the working set, then all the retained blocks in the cache does not contribute for the further hits or misses. To solve this issue, we use BIP that infrequently inserts the block in the MRU position and adapts to the changes in the working set while protecting the cache against thrashing. DIP is implemented to select between LRU and BIP based on the policy selection counter and takes the advantage of both the policies dynamically depending on the type and size of the working set. Hence, we can say that miss rate of a policy depends on the size of the cache and also on the size and type of the benchmark. From the above results, we can say that the Mcf benchmark has working set that is LRU friendly and hence, LRU performs better here. Bzip2 and Gcc benchmarks does not have working sets favorable to LRU and thus, DIP performs better.

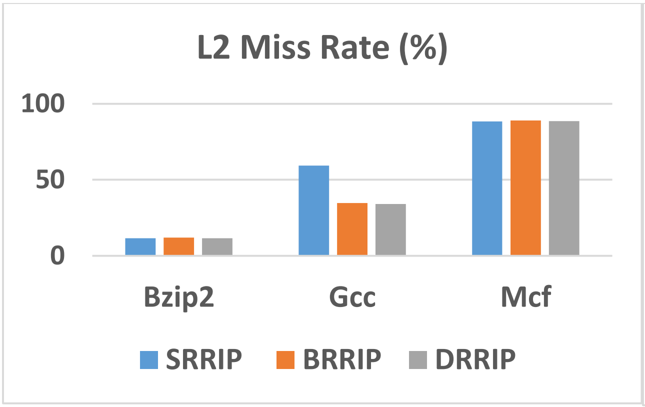
**Problem 5:**

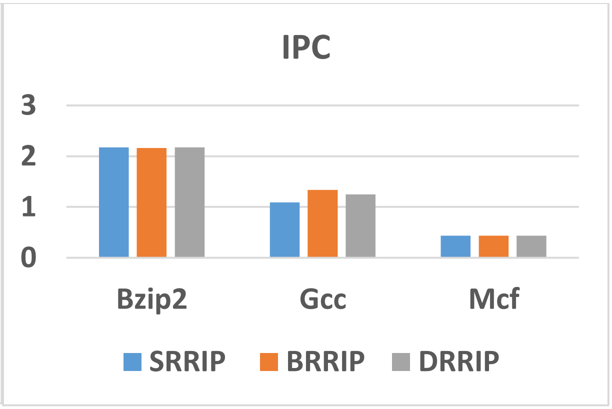
**Implementation:**

For SRRIP, BRRIP and DRRIP, I have added separate cases in the cache.c in the cache miss section and wrote conditions for hits for each policy. I have defined RRPV counter, policy selection counter and BRRIP counter variables. I have provided the appropriate comments explaining the operations performed in cache.c and cahe.c files. I have used the set dueling mechanism as mentioned in the assignment and divided the total number of sets into three categories namely srrip, brrip and followerset. Depending on the set type, I implemented the respective policies as described according to the papers.

1)

| | | SRRIP | BRRIP | DRRIP |
|---|---|---|---|---|
| L2 miss rate (%) | Bzip2 | 11.53 | 11.9 | 11.54 |
| | Gcc | 59.36 | 34.7 | 34.15 |
| | Mcf | 88.48 | 89.02 | 88.5 |
| IPC | Bzip2 | 2.1714 | 2.1640 | 2.1711 |
| | Gcc | 1.0897 | 1.3362 | 1.2462 |
| | Mcf | 0.4361 | 0.4379 | 0.4361 |





2)

| | | LRU | FIFO | Random | LIP | BIP | DIP | DRRIP |
|---|---|---|---|---|---|---|---|---|
| L2 miss rate (%) | Bzip2 | 12.77 | 13.99 | 14.08 | 12.95 | 11.62 | 11.52 | 11.54 |
| | Gcc | 62.97 | 63.14 | 45.17 | 70.37 | 34.43 | 34.25 | 34.15 |
| | Mcf | 88.28 | 88.42 | 88.99 | 89.25 | 89.43 | 88.31 | 88.5 |
| IPC | Bzip2 | 2.1387 | 2.1075 | 2.1051 | 2.1315 | 2.1705 | 2.1672 | 2.1711 |
| | Gcc | 1.060 | 1.0589 | 1.1739 | 1.0296 | 1.2720 | 1.2694 | 1.2462 |
| | Mcf | 0.4361 | 0.4357 | 0.4352 | 0.4384 | 0.4371 | 0.4361 | 0.4361 |

DRRIP does not always perform better than the other replacement policies. The performance of a replacement policy depends on the size of the cache, size and type of the working set. From the above results, we can say that for Gcc benchmark DRRIP performs better than other replacement policies. But for Bzip2 and Mcf it does not perform better because those working sets appear to be larger in size than the size of the cache and may be the type of the working sets these benchmarks possess may also effect the performance of these policies. For benchmarks that are LRU friendly, LRU performs better. The other advanced policies solve the cache thrashing problem. To still improve the granularity in dealing with misses, we use SRRIP, BRRIP and ultimately DRRIP in order to reduce the miss rate and improve the existing policies.