# ECE 506 – Project 2 Report

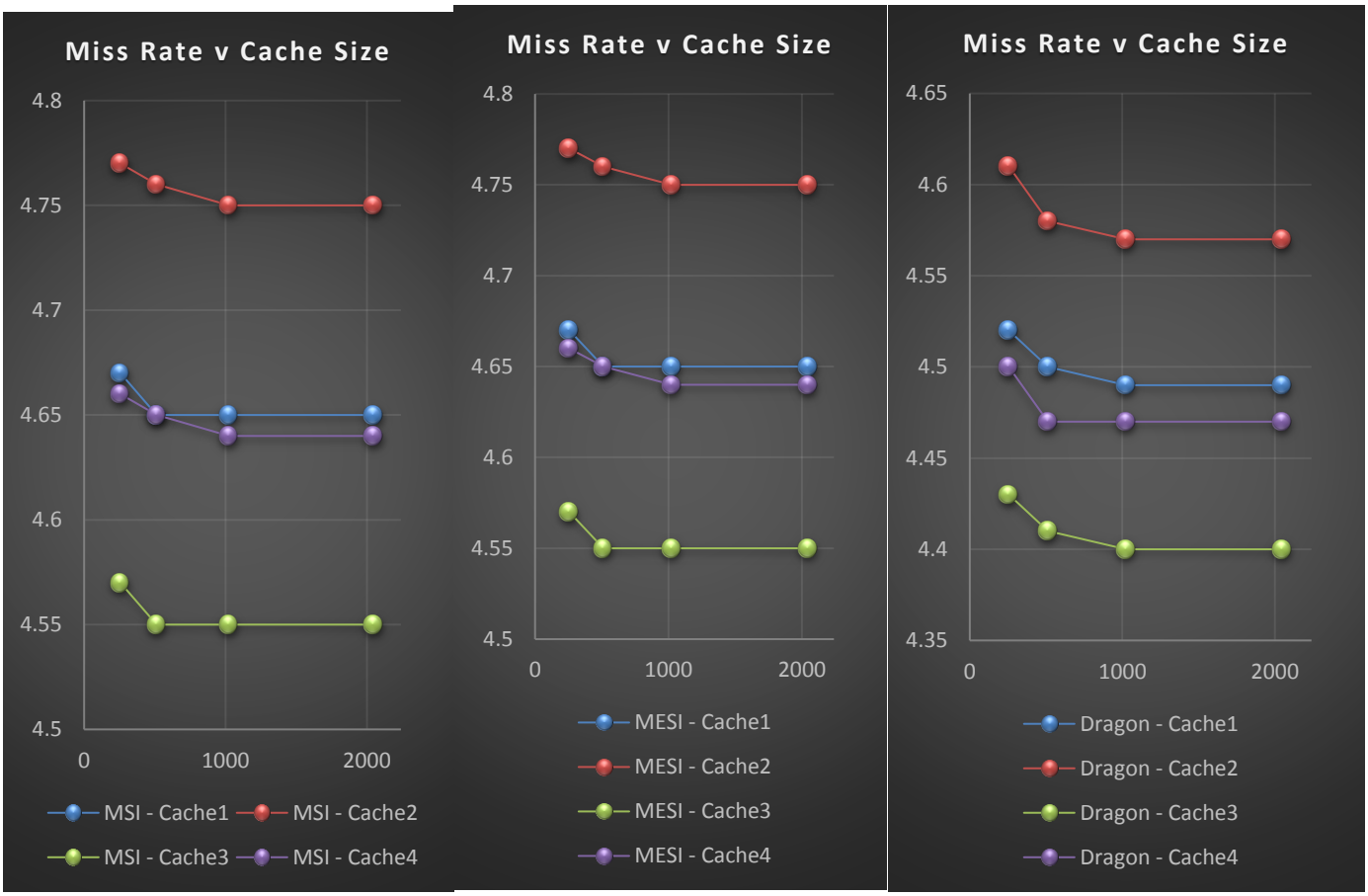
**Varun Varadarajan – vvarada2@ncsu.edu**

***NOTE:*** *0,1,2,3 processor digit of trace file corresponds to processors 1,2,3,4 in this report*

**Miss Rate v Cache Size Analysis (Caches 1 to 4 as columns)**

Miss Rate in **%** and Cache Size in **kiloBytes (row headers) –** N=8, BlkSize=64

| **MSI** | MR1 | MR2 | MR3 | MR4 |
|---|---|---|---|---|
| 256 | 4.67 | 4.77 | 4.57 | 4.66 |
| 512 | 4.65 | 4.76 | 4.55 | 4.65 |
| 1024 | 4.65 | 4.75 | 4.55 | 4.64 |
| 2048 | 4.65 | 4.75 | 4.55 | 4.64 |
| **MESI** | MR1 | MR2 | MR3 | MR4 |
| 256 | 4.67 | 4.77 | 4.57 | 4.66 |
| 512 | 4.65 | 4.76 | 4.55 | 4.65 |
| 1024 | 4.65 | 4.75 | 4.55 | 4.64 |
| 2048 | 4.65 | 4.75 | 4.55 | 4.64 |
| **DRAGON** | MR1 | MR2 | MR3 | MR4 |
| 256 | 4.52 | 4.61 | 4.43 | 4.5 |
| 512 | 4.5 | 4.58 | 4.41 | 4.47 |
| 1024 | 4.49 | 4.57 | 4.4 | 4.47 |
| 2048 | 4.49 | 4.57 | 4.4 | 4.47 |

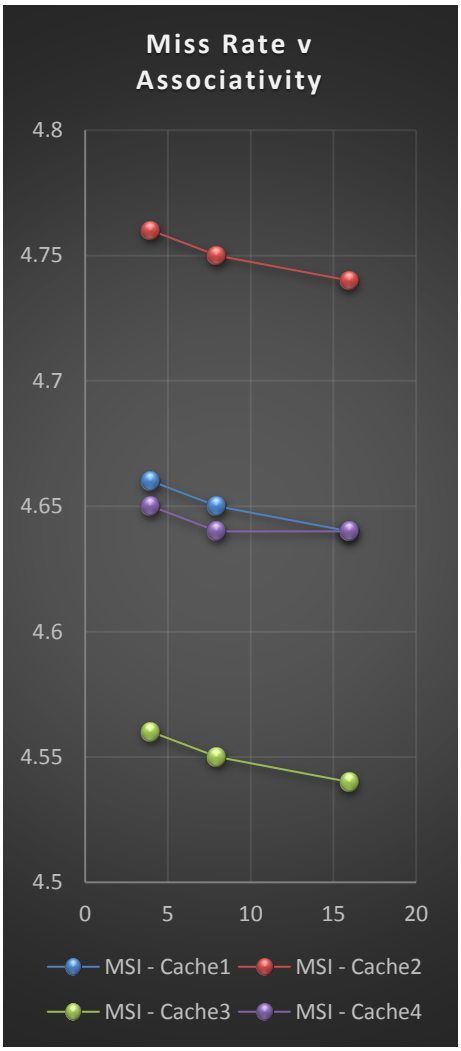


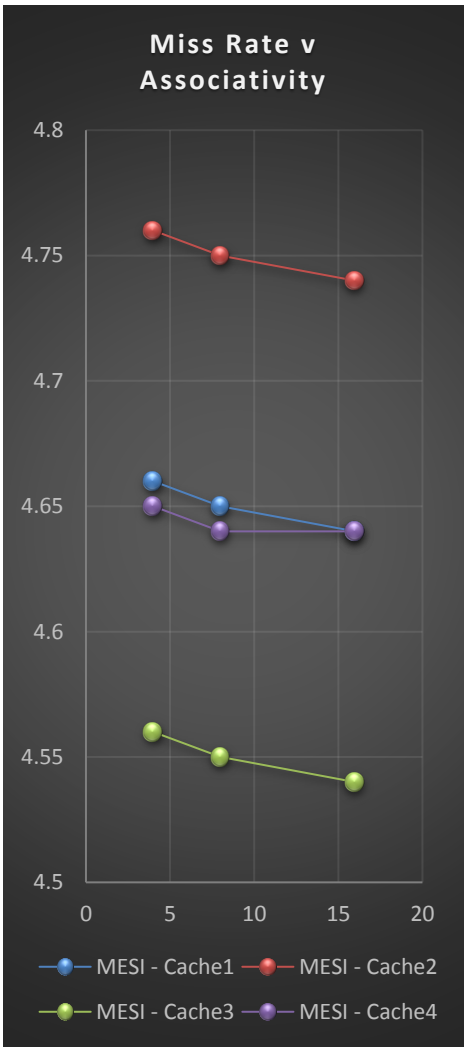

Miss rate reduces with cache size in all 3 cases but when size is too high, MR saturates because a big cache is already good enough to provide same amount of hit references as even if we have many indexes now, the tags still reference some of them only mostly due to temporal/special locality. An even bigger cache is going to accommodate the roughly same amount of hit references… so, we will have only diminishing returns and so, our development cost will only become costlier, also requiring bigger hardware for index multiplexing, etc. This also reduces capacity misses.

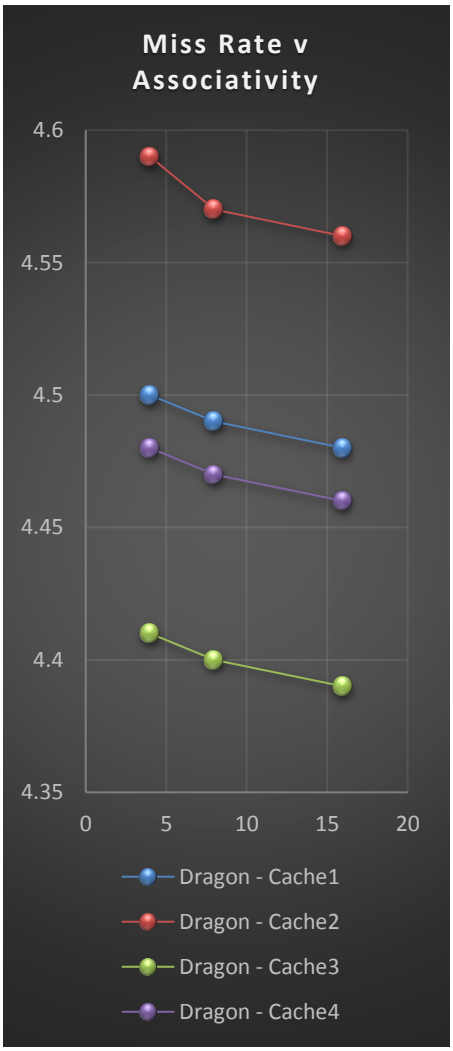
**Miss Rate v Cache Associativity Analysis (Caches 1 to 4 as columns)**

Miss Rate in **%** and Cache Associativity **(row headers) -** CacheSize-1MB, BlkSize 64B

| **MSI** | MR1 | MR2 | MR3 | MR4 |
|---|---|---|---|---|
| 4 | 4.66 | 4.76 | 4.56 | 4.65 |
| 8 | 4.65 | 4.75 | 4.55 | 4.64 |
| 16 | 4.64 | 4.74 | 4.54 | 4.64 |
| **MESI** | MR1 | MR2 | MR3 | MR4 |
| 4 | 4.66 | 4.76 | 4.56 | 4.65 |
| 8 | 4.65 | 4.75 | 4.55 | 4.64 |
| 16 | 4.64 | 4.74 | 4.54 | 4.64 |
| **DRAGON** | MR1 | MR2 | MR3 | MR4 |
| 4 | 4.5 | 4.59 | 4.41 | 4.48 |
| 8 | 4.49 | 4.57 | 4.4 | 4.47 |
| 16 | 4.48 | 4.56 | 4.39 | 4.46 |



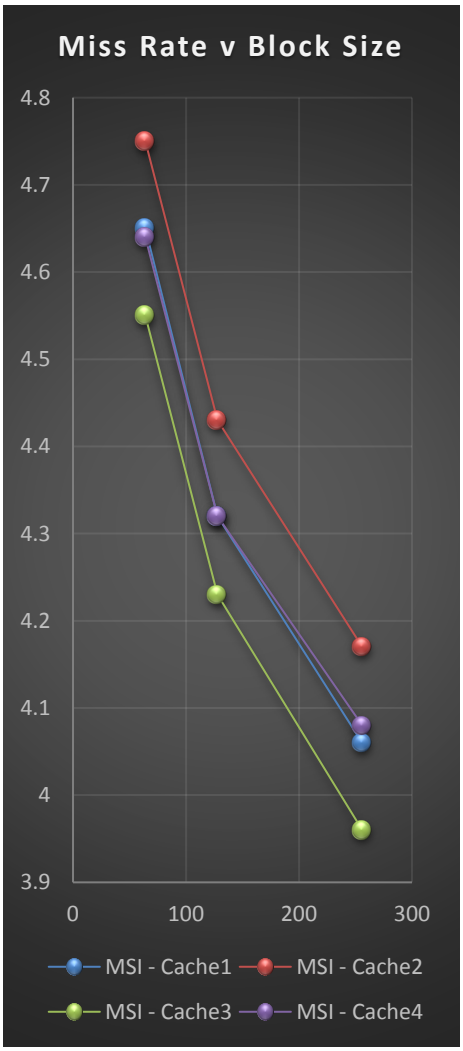



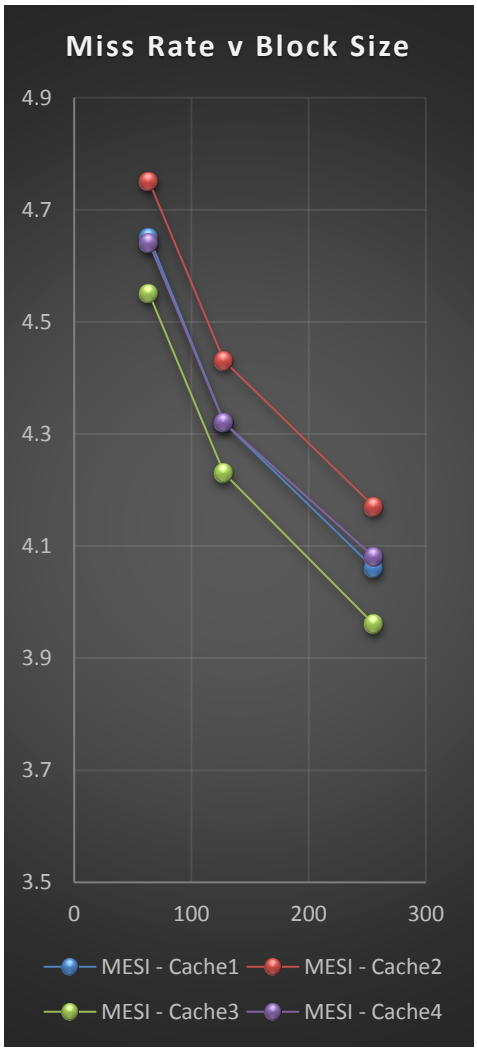

Miss rate reduces with associativity in all 3 cases but when cache size is too higher, it can begin to saturate because a big associativity already accommodates more tags in a given way/index. Beyond a point, there won't be more competition for same index and so, some cache blocks are not used. We will just waste memory space and incur higher costs if we expand associativity too much. Diminishing returns is the result. It can reduce conflict misses.

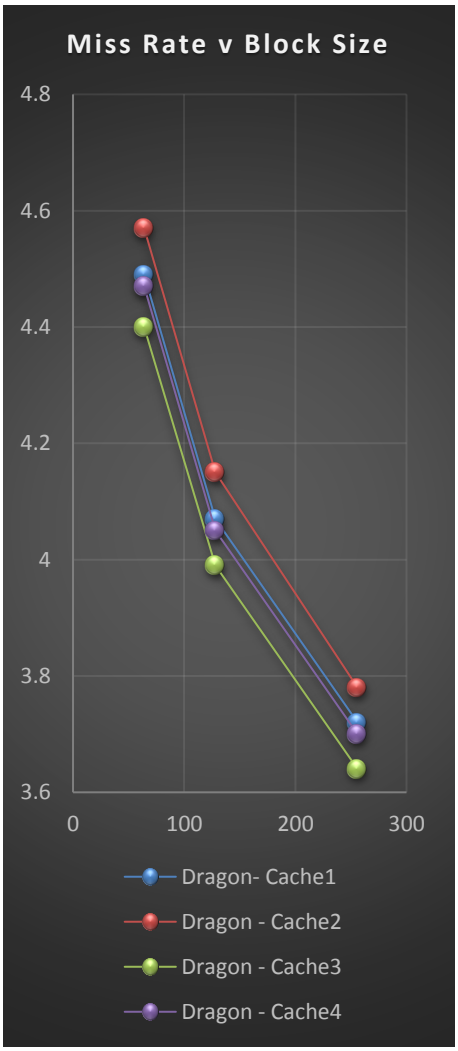
**Miss Rate v Block Size Analysis (Caches 1 to 4 as columns)**

Miss Rate in **%** and Cache Block Size in **Bytes (row headers) – N=8, Cache Size 1MB**

| **MSI** | MR1 | MR2 | MR3 | MR4 |
|---|---|---|---|---|
| 64 | 4.65 | 4.75 | 4.55 | 4.64 |
| 128 | 4.32 | 4.43 | 4.23 | 4.32 |
| 256 | 4.06 | 4.17 | 3.96 | 4.08 |
| **MESI** | MR1 | MR2 | MR3 | MR4 |
| 64 | 4.65 | 4.75 | 4.55 | 4.64 |
| 128 | 4.32 | 4.43 | 4.23 | 4.32 |
| 256 | 4.06 | 4.17 | 3.96 | 4.08 |
| **DRG** | MR1 | MR2 | MR3 | MR4 |
| 64 | 4.49 | 4.57 | 4.4 | 4.47 |
| 128 | 4.07 | 4.15 | 3.99 | 4.05 |
| 256 | 3.72 | 3.78 | 3.64 | 3.7 |



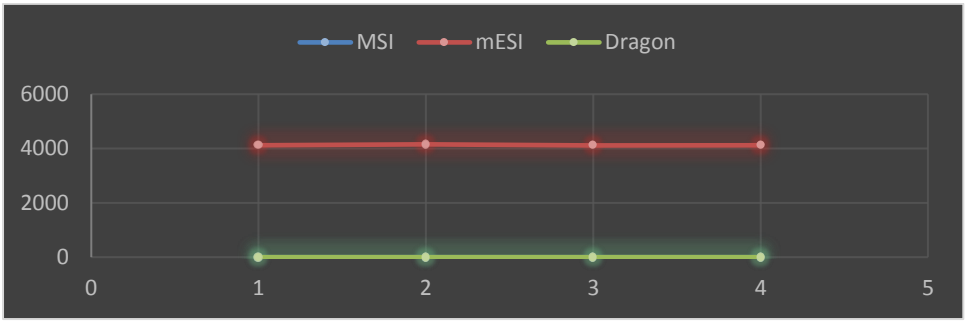

Miss rate decreases steadily as block size increases for all three cases because with higher block size, a single tag can address more data blocks and hence provide higher hits. However, this also increases latency of operations as one fetch will need to bring a bigger block of data every single time.

**Miscellaneous Parameters Analysis** – *CacheSize 1MB, N=8, BlkSize 128B*

From above, we noticed DRAGON protocol has generally lower miss rate compared to MSI and MESI. This is because in MSI and MESI, write invalidate forces all other caches to suffer coherence misses each time a new write modification happens. However, in dragon, since only write update happens, the SHARED MODIFIED states helps enables successful hits in all involved caches, so that coherence misses are reduced.

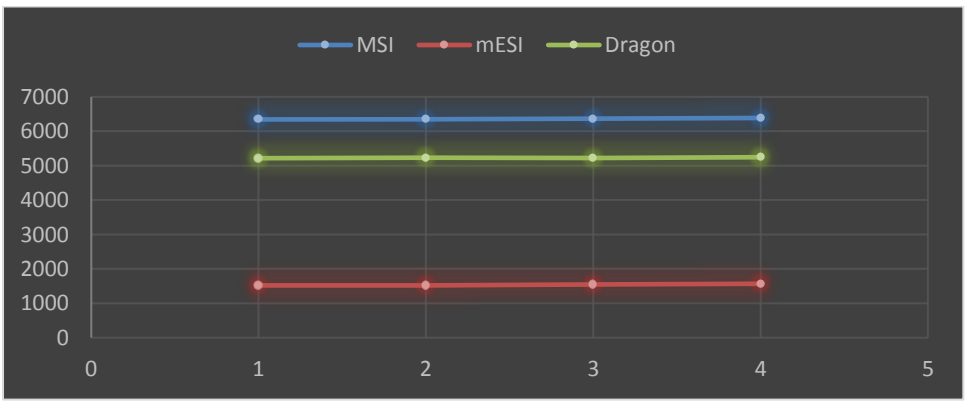
| | CacheNum | Cache-Cache Transfer | MemTrasfer | Inteventions | Invaliudations | Flushes | BusRdX |
|---|---|---|---|---|---|---|---|
| MSI | 1 | 0 | 6344 | 119 | 2066 | 164 | 729 |
| | 2 | 0 | 6347 | 84 | 2089 | 129 | 711 |
| | 3 | 0 | 6369 | 127 | 2053 | 166 | 745 |
| | 4 | 0 | 6386 | 110 | 2068 | 135 | 733 |
| MESI | 1 | 4124 | 1520 | 1367 | 2066 | 164 | 39 |
| | 2 | 4155 | 1521 | 1337 | 2089 | 129 | 40 |
| | 3 | 4115 | 1551 | 1377 | 2053 | 166 | 42 |
| | 4 | 4125 | 1567 | 1385 | 2068 | 135 | 39 |
| DRG | 1 | 0 | 5217 | 1256 | 0 | 3 | 0 |
| | 2 | 0 | 5230 | 1266 | 0 | 9 | 0 |
| | 3 | 0 | 5227 | 1257 | 0 | 6 | 0 |
| | 4 | 0 | 5246 | 1287 | 0 | 9 | 0 |

**Cache-Cache Transfers Analysis (for processors 1-4)**



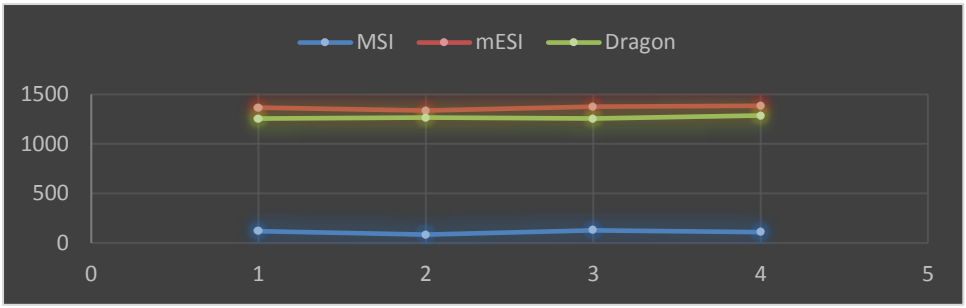

MSI and MESI have similar CCT numbers as the reads make the cache's state into SHARED no matter what. So, a new value from a MODIFIED cache can be transferred to another cache which requests it, either with/without flushing. Dragon in our assumption (for validation runs) has no CCT because it derives blocks from memory during misses

**Memory Transfers (for processors 1-4)**



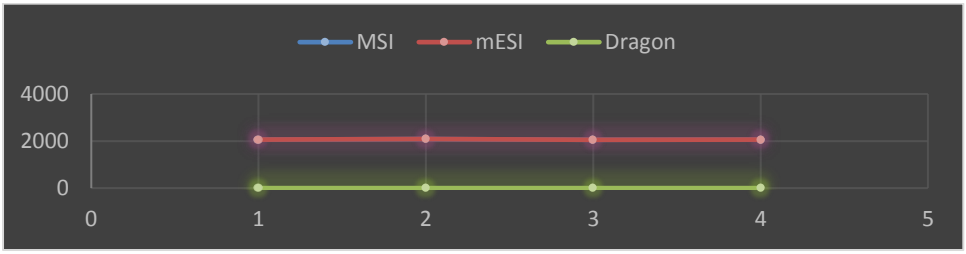

MSI has more memory-transfers than MESI because it doesn't have the EXCLUSIVE state. So, for each invalidation and then read, the cache has to read value from the memory instead of transferring from another cache that has the MODIFIED state for that line. Dragon has more memory transfers because of our assumption for validation runs. (In reality, it's an update protocol that involves updating memory only when optimally required)

**Interventions (for processors 1-4)**



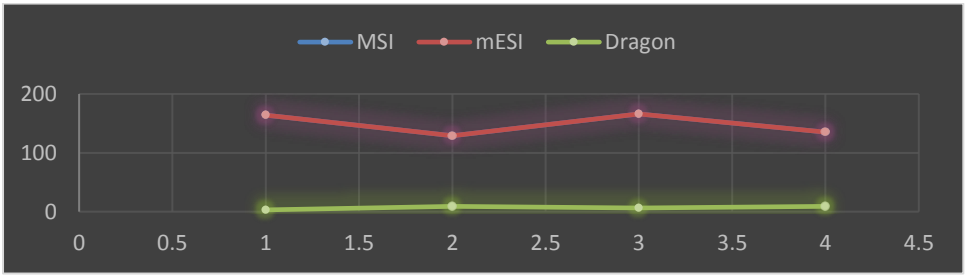

MSI has low interventions because a lot of M-state lines become invalid before they go change to Shared state. MESI has more interventions because E-states exist. Both E and M are changed to S on reads. Dragon protocol also has more interventions than MSI because it has an extra SHARED MODIFIED state compared to MSI.

**Invalidations (for processors 1—4)**



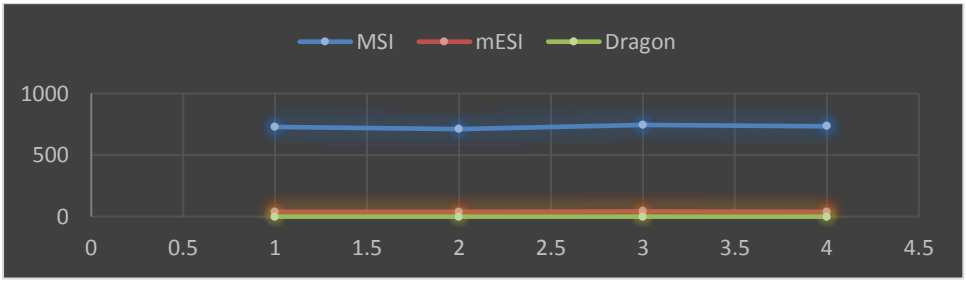

MESI and MSI (overlapped) have same no. of invalidations as each write sends an invalidate request that is picked up by the snoopy protocols. However, Dragon has no invalidations because it is an update protocol in nature.

**Flushes (for processors 1—4)**




MSI and MESI have similar flushes because new reads that read from a modified state of another cache with change it to shared state and flush its value to memory if required to update it (MSI/MESI difference decides it). However, Dragon protocol has low flushes because it does so only during BusRd to change MODIFIED to SHARED-MODIFIED.

**Bus Read Exclusive (for processors 1—4)**




MESI has very low busRdX compared to MSI because it uses bus-upgrade instead of busRdX for all involved transactions apart from the when NULL state (unused cache line) is written on, to become MODIFIED.
Dragon has no busRdX as no invalidations occur (it uses only busUgr for updating)