Assignment - 1

Kuchi Sai Teja (21317, kuchisai@iisc.ac.in)
Utkrisht Patesaria (20892, utkrishtp@iisc.ac.in)

Question-1:

- **perf_event_open()** is used to profile the program using various Performance Monitoring Counters.

- Matrix initialization has been excluded from the performance measurements and all the results shown below are that of the Matrix multiplication part.

- Two different Matrix sizes :- 2048x2048 & 8192x8192 are used.

- For part 1a) only tile/block size of 32 is considered. While for part 1b) tile/block sizes are varied in multiples of 2 starting from (8,..,2048) & (8,…,8192) for matrix of size 2048 and 8192 respectively.

- Since **perf_event_open()** is being used, there are no plots for L2 cache references or misses.

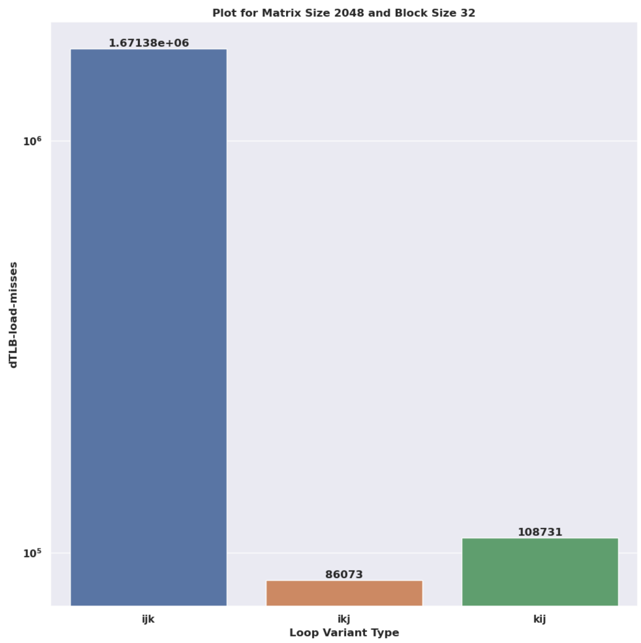
- Miss percent for different events (dTLB/LLC etc) are calculated as misses for that event/total references for that event *100.

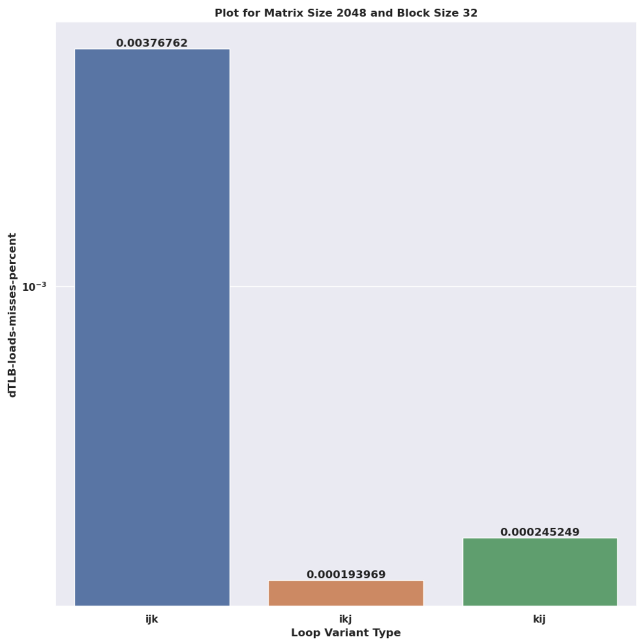
- Average miss percent is nothing but the average of the above miss percent which was calculated for different events

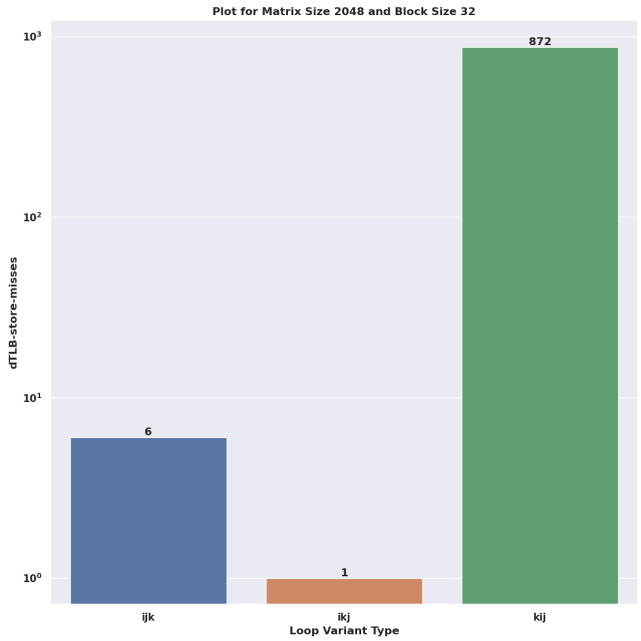
All the graph below have been averaged over 5 runs for matrix size 2048 and 1 run for matrix size 8192

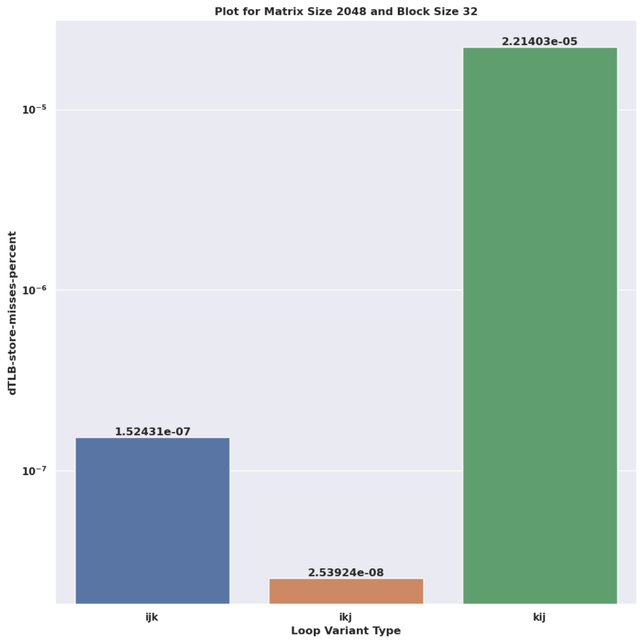
1a) Finding the loop variant which performs better for 2 different matrix sizes with block size of 32.

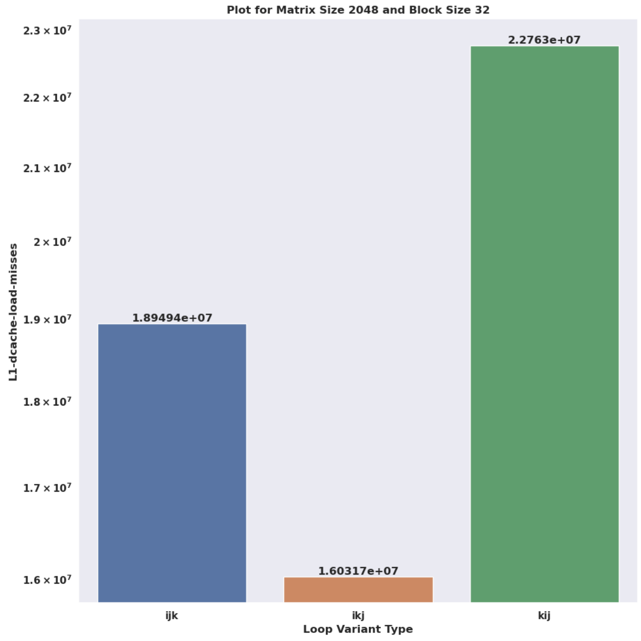
1. **For Matrix Size 2048 x 2048 and Block Size 32, below are the plots for various parameters/events like LLC misses, cycles, time taken to execute etc. listed on y-axis, while the loop variant is listed on x-axis.**

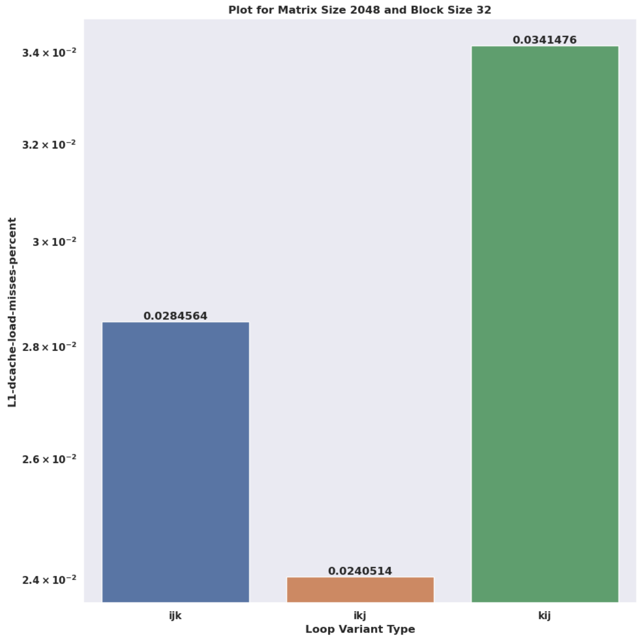

Plot for Matrix Size 2048 and Block Size 32
dTLB-load-misses
10^6
10^5
1.67138e+06
86073
108731
ijk
ikj
kij
Loop Variant Type

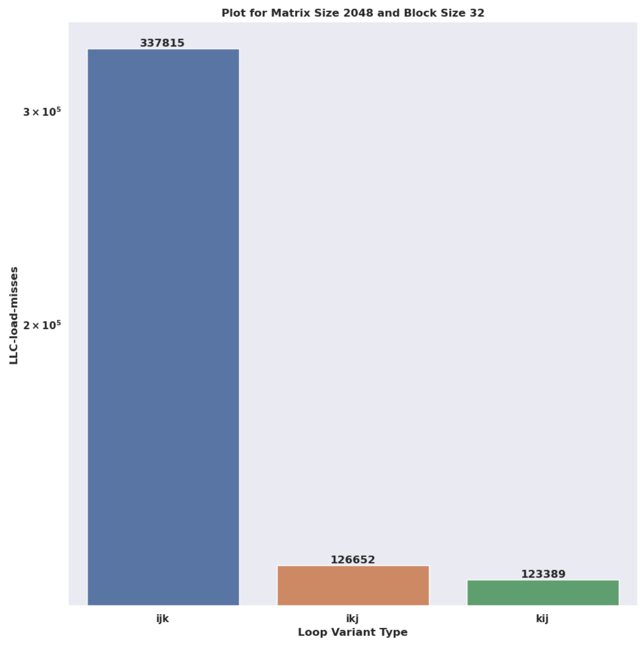

Plot for Matrix Size 2048 and Block Size 32
dTLB-loads-misses-percent
10^{-3}
0.00376762
0.000193969
0.000245249
ijk
ikj
kij
Loop Variant Type


Plot for Matrix Size 2048 and Block Size 32
dTLB-store-misses
10^3
10^2
10^1
10^0
6
1
872
ijk
ikj
kij
Loop Variant Type

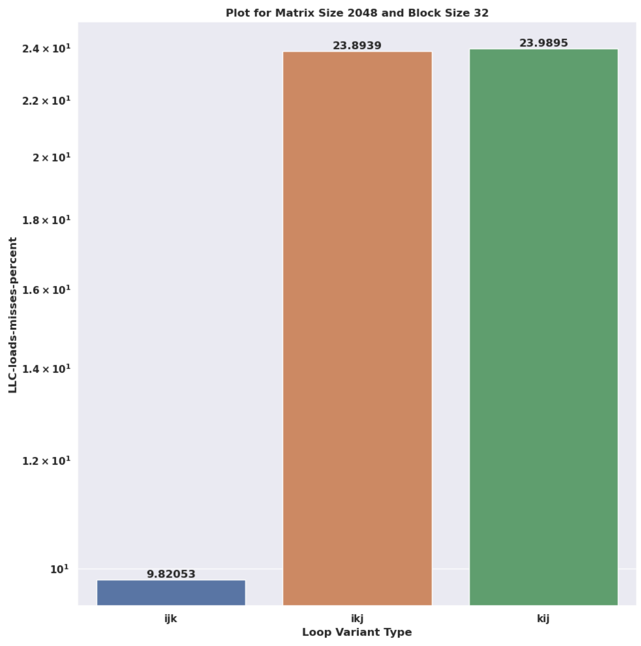

Plot for Matrix Size 2048 and Block Size 32
dTLB-store-misses-percent
10^{-5}
10^{-6}
10^{-7}
1.52431e-07
2.53924e-08
2.21403e-05
ijk
ikj
kij
Loop Variant Type


Plot for Matrix Size 2048 and Block Size 32
L1-dcache-load-misses
2.3×10^7
2.2×10^7
2.1×10^7
2×10^7
1.9×10^7
1.8×10^7
1.7×10^7
1.6×10^7
1.89494e+07
1.60317e+07
2.2763e+07
ijk
ikj
kij
Loop Variant Type

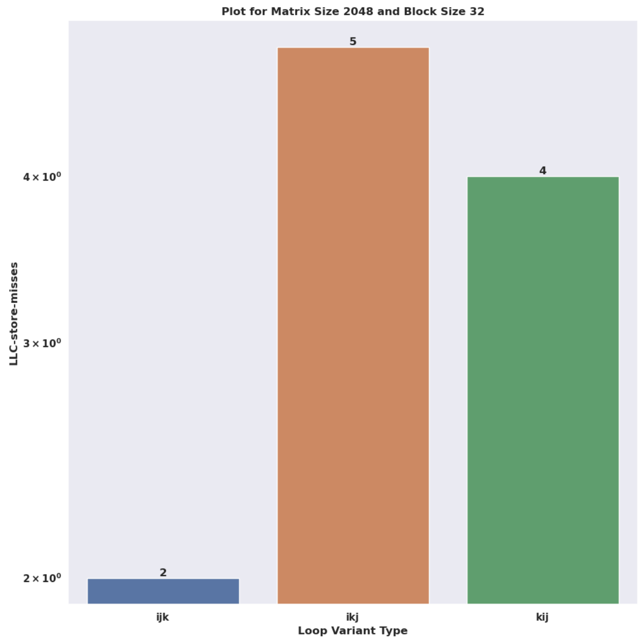

Plot for Matrix Size 2048 and Block Size 32
L1-dcache-load-misses-percent
3.4×10^{-2}
3.2×10^{-2}
3×10^{-2}
2.8×10^{-2}
2.6×10^{-2}
2.4×10^{-2}
0.0284564
0.0240514
0.0341476
ijk
ikj
kij
Loop Variant Type



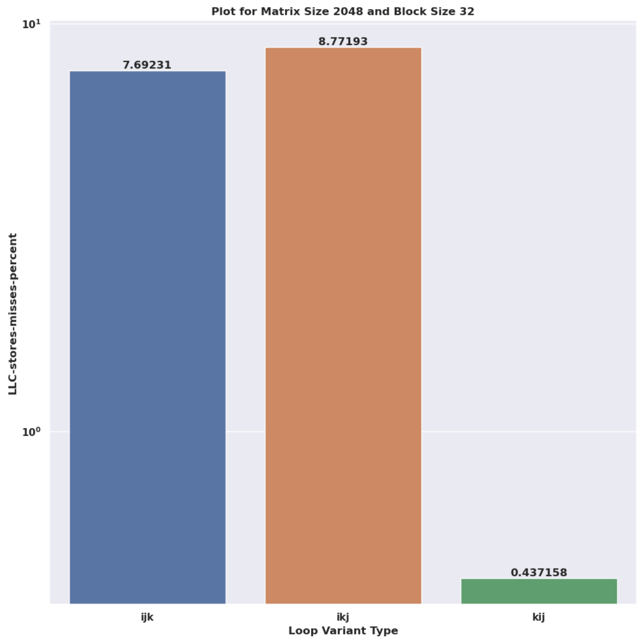
Plot for Matrix Size 2048 and Block Size 32



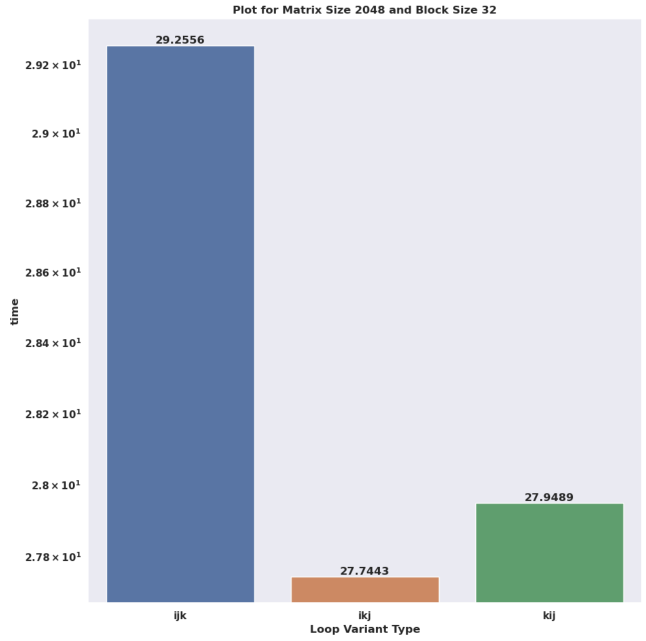
Plot for Matrix Size 2048 and Block Size 32



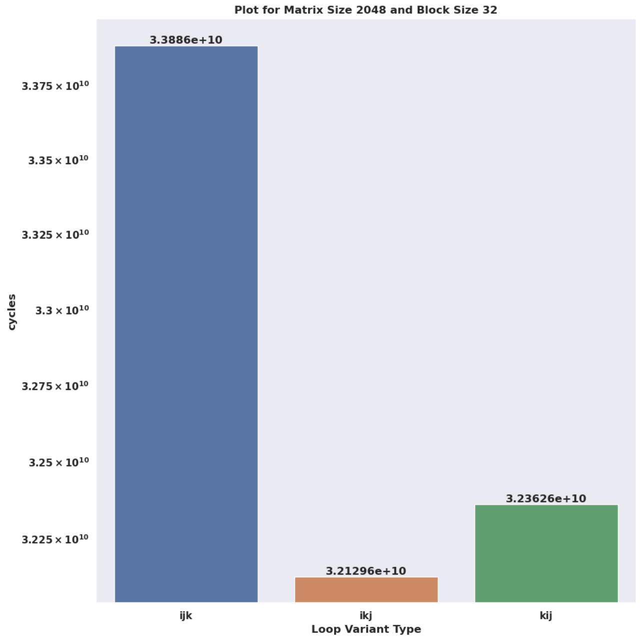
Plot for Matrix Size 2048 and Block Size 32



Plot for Matrix Size 2048 and Block Size 32



Plot for Matrix Size 2048 and Block Size 32



Plot for Matrix Size 2048 and Block Size 32

- The plots from left to right are event-misses and event-misses-percent. From top to bottom are different events that the perf_event_open captured.

- To justify that even when there are higher number of misses of a loop variant compared to others, miss percentage for that parameter is calculated using the formula that is mentioned previously, so that a fair comparison among percentages can be used when an loop invariant might have higher number of references when compared to its counterparts .

- From the above plots, we can observe that most of the time the ikj variant is either having lower or close to lower number of misses in most of the events and execution time along with cycles is the lowest compared to the other variants.

- Only the LLC load and store parameters for the ikj variant is higher compared to all other parameters, we are assuming this is due to other background processes evicting current cache blocks.

- Few events like page faults are having either 0 or 1 as its values, thus overall not affecting the final decision.

- Thus, we can conclude from overall above plots that the **ikj variant** performs better than other variants when matrix size is 2048 x 2048 and block size is 32.

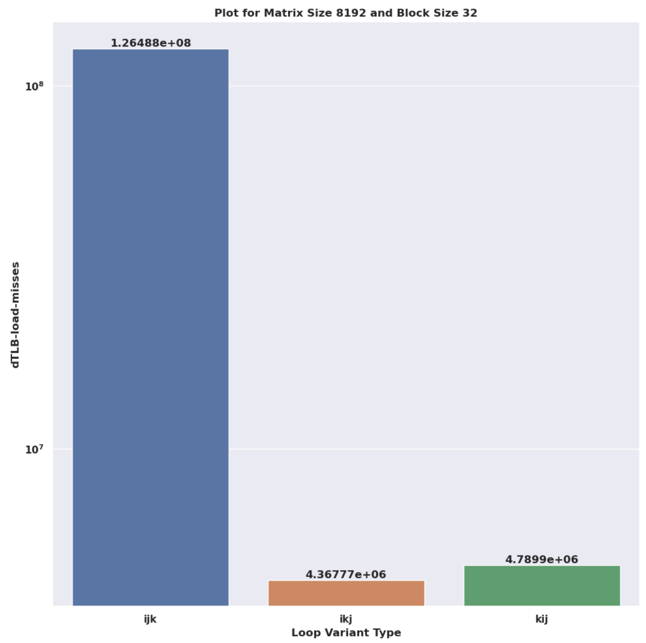
- Below is the table which shows the actual values obtained for different events and its corresponding variants ordered in ijk, ikj and kij respectively for matrix size 2048 x 2048 and block size of 32. For full table with other events data please contact us in person.

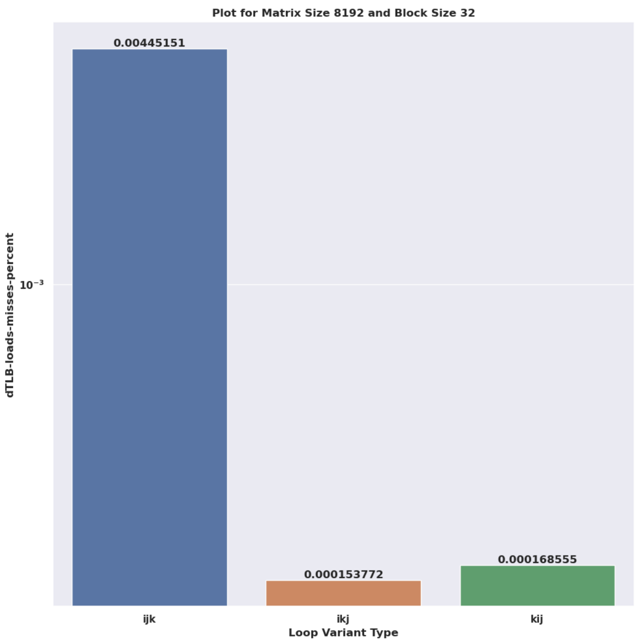
LLC-loads	***LLC-load-misses***	***LLC-stores***	***LLC-store-misses***	***dTLB-loads***	***dTLB-load-misses***	***dTLB-stores***	***dTLB-store-misses***
3439884	337815	26	2	44361784429	1671382	3936207183	6
530060	126652	57	5	44374613136	86073	3938181218	1
514346	123389	915	4	44335026044	108731	3938510894	872

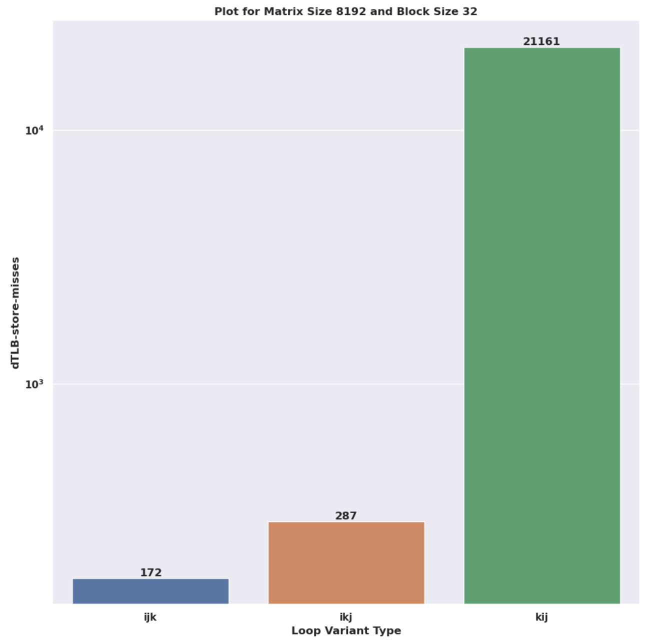
cycles	***page-faults***	***L1-dcache-loads***	***L1-dcache-load-misses***
33885960950	1	66590873602	18949352
32129616778	0	66656039649	16031722
32362603549	1	66660792609	22763029

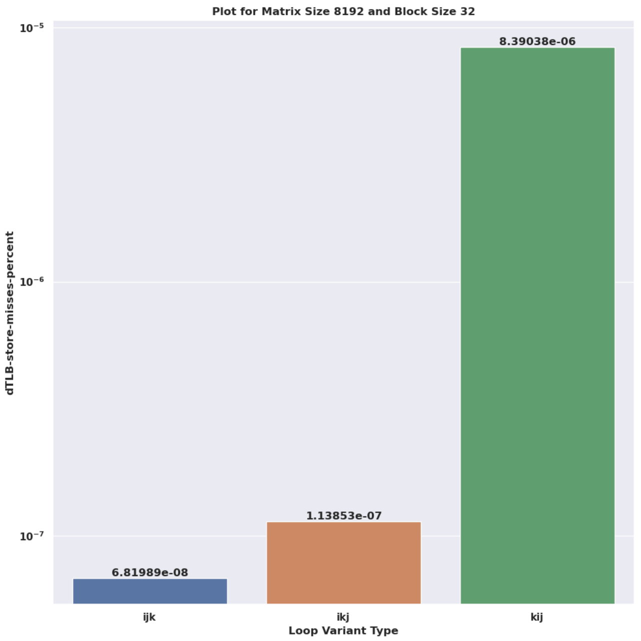
Table 1: Matrix - 2048 x 2048 | Tile-Size 32 | Rows are in order of ijk, ikj, kij

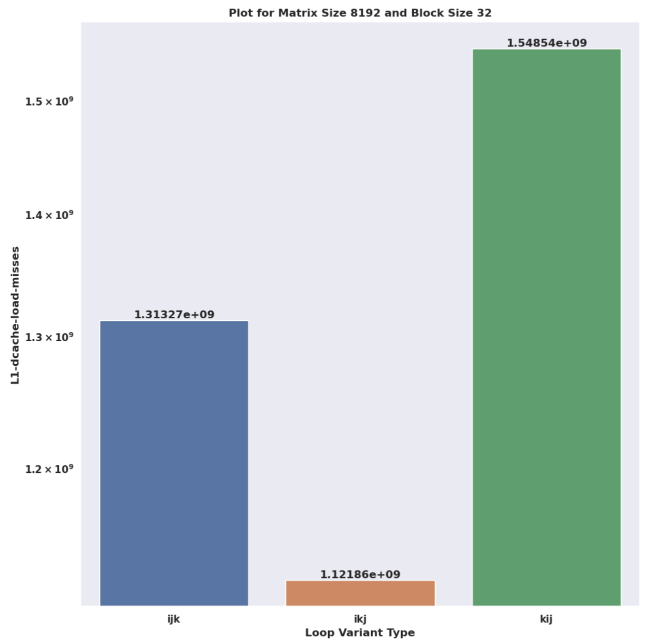
2. **For Matrix Size 8192 x 8192 and Block Size 32, below are the plots for various parameters/events like LLC misses, cycles, time taken to execute etc listed on y-axis, while the loop variant is listed on x-axis.**

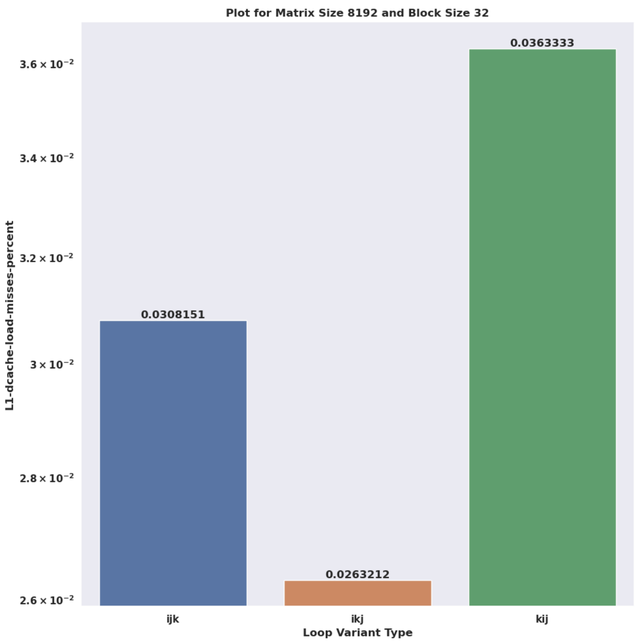

Plot for Matrix Size 8192 and Block Size 32
dTLB-load-misses
10^8
10^7
1.26488e+08
4.36777e+06
4.7899e+06
ijk
ikj
kij
Loop Variant Type

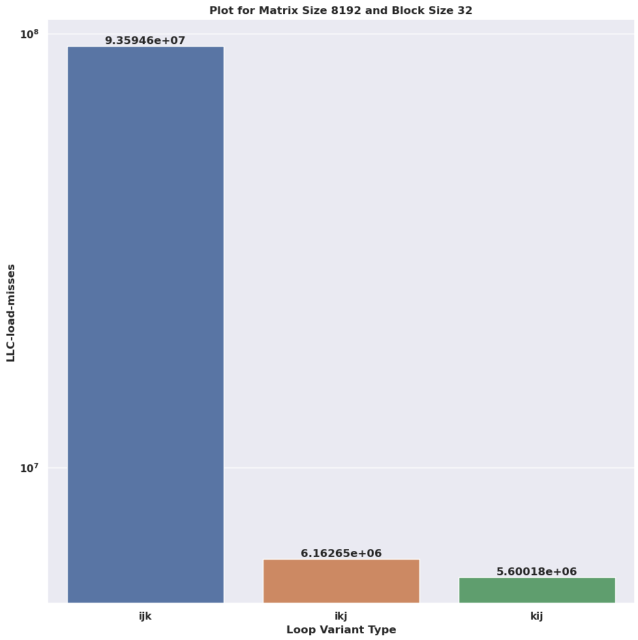

Plot for Matrix Size 8192 and Block Size 32
dTLB-loads-misses-percent
10^{-3}
0.00445151
0.000153772
0.000168555
ijk
ikj
kij
Loop Variant Type

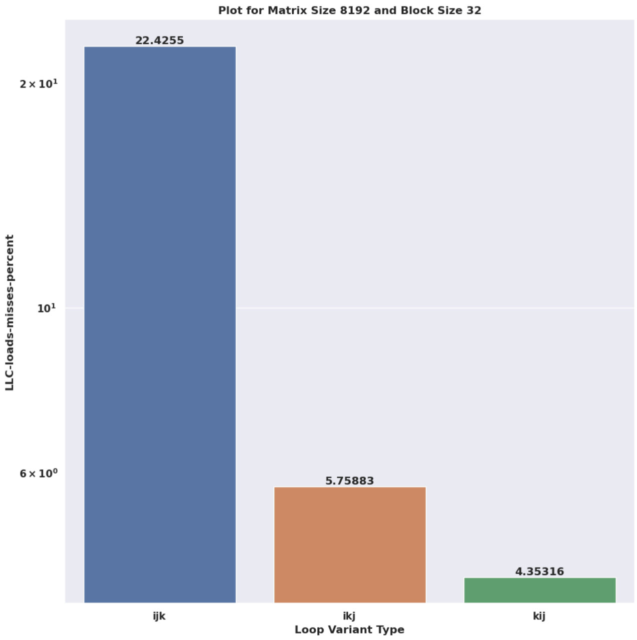

Plot for Matrix Size 8192 and Block Size 32
dTLB-store-misses
10^4
10^3
172
287
21161
ijk
ikj
kij
Loop Variant Type

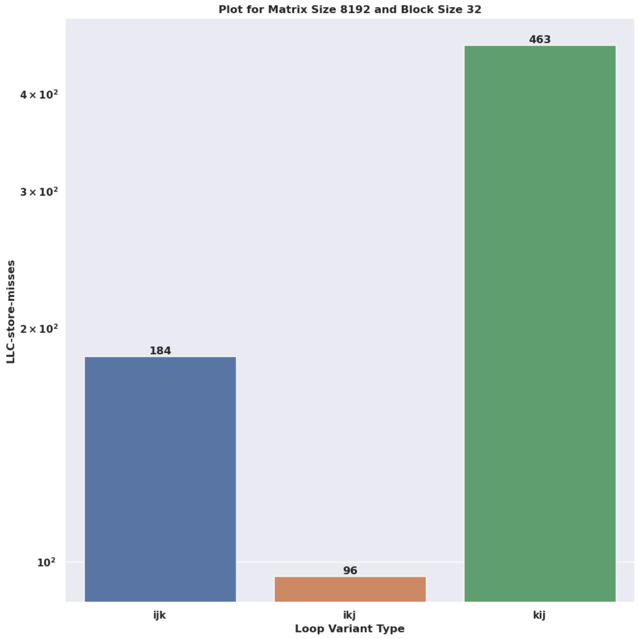

Plot for Matrix Size 8192 and Block Size 32
dTLB-store-misses-percent
10^{-5}
10^{-6}
10^{-7}
6.81989e-08
1.13853e-07
8.39038e-06
ijk
ikj
kij
Loop Variant Type

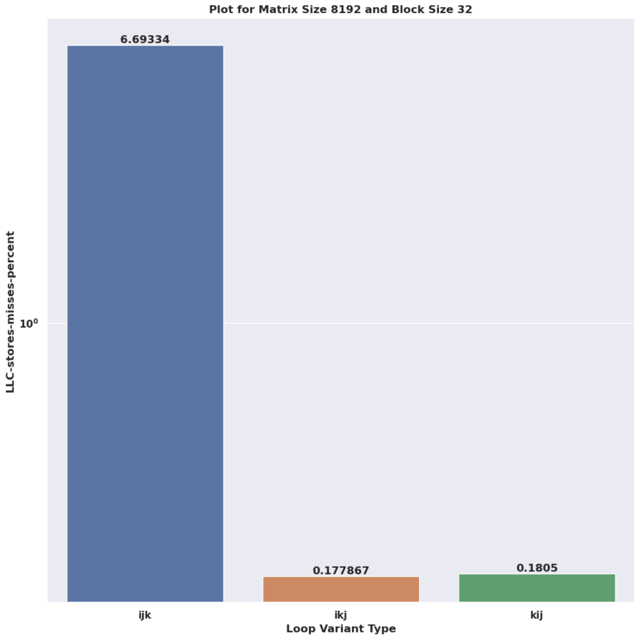

Plot for Matrix Size 8192 and Block Size 32
L1-dcache-load-misses
1.5×10^9
1.4×10^9
1.3×10^9
1.2×10^9
1.31327e+09
1.12186e+09
1.54854e+09
ijk
ikj
kij
Loop Variant Type

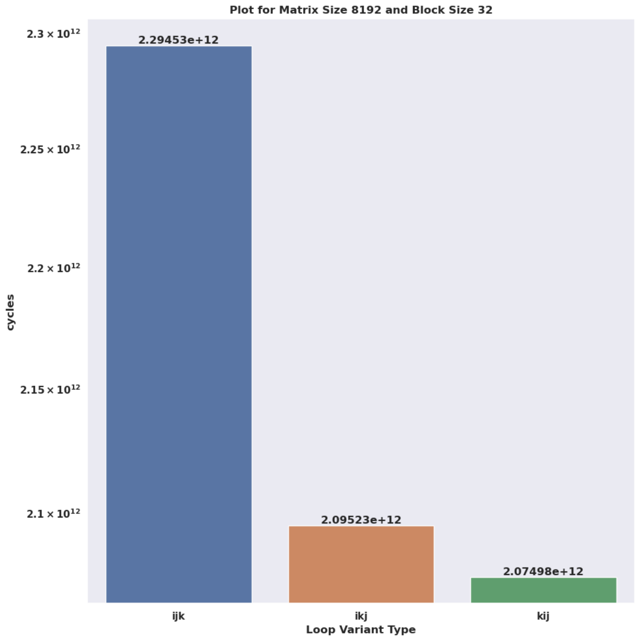

Plot for Matrix Size 8192 and Block Size 32
L1-dcache-load-misses-percent
3.6×10^{-2}
3.4×10^{-2}
3.2×10^{-2}
3×10^{-2}
2.8×10^{-2}
2.6×10^{-2}
0.0308151
0.0263212
0.0363333
ijk
ikj
kij
Loop Variant Type

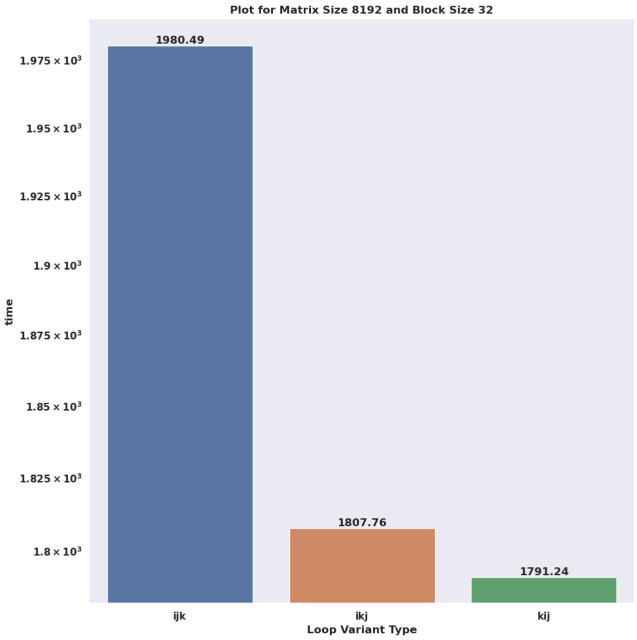

Plot for Matrix Size 8192 and Block Size 32


Plot for Matrix Size 8192 and Block Size 32


Plot for Matrix Size 8192 and Block Size 32


Plot for Matrix Size 8192 and Block Size 32


Plot for Matrix Size 8192 and Block Size 32


Plot for Matrix Size 8192 and Block Size 32

- The plots from left to right are event-misses and event-misses-percent. From top to bottom are different events that the perf_event_open captured.

- To justify that even when there are higher number of misses of a loop variant compared to others, miss percentage for that parameter is calculated using the formula that is mentioned previously, so that a fair comparison among percentages can be used when an loop invariant might have higher number of references when compared to its counterparts .

- From the above plots, we can observe that most of the time the kij variant is either having lower or close to lower number of misses in most of the events and execution time along with cycles is the lowest compared to the other variants. While in the plots sometime the kij bar is higher than others, if we observe closely its in 10^-3 to 10^-7 range.

- Few events like page faults are having either 0 or 1 as its values, thus overall not affecting the final decision.

- Thus, we can conclude from overall above plots that the **kij variant** performs better than other variants when matrix size is 8192 x 8192 and block size is 32.

- Below is the table which shows the actual values obtained for different events and its corresponding variants ordered in ijk, ikj and kij respectively for matrix size 8192 x 8192 and block size of 32. For full table with other events data please contact us in person.

LLC-loads	***LLC-load-misses***	***LLC-stores***	***LLC-store-misses***	***dTLB-loads***	***dTLB-load-misses***	***dTLB-stores***	***dTLB-store-misses***
417357757	93594649	2749	184	2841460509846	126487838	252203329818	172
107012160	6162653	53973	96	2840415998879	4367771	252078725986	287
128646367	5600182	256510	463	2841745513517	4789895	252205506343	21161

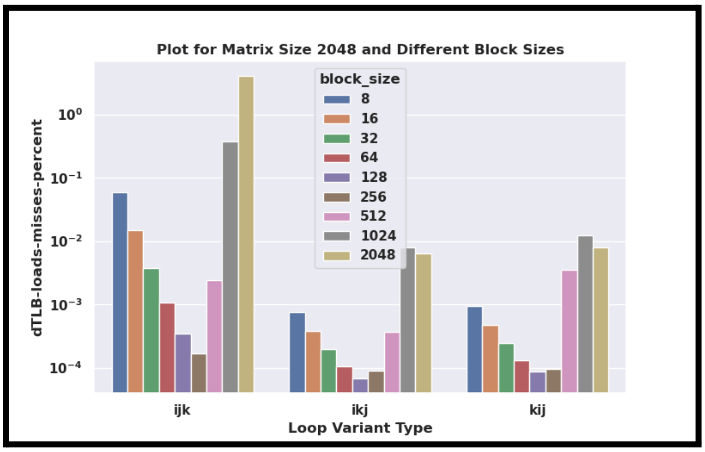
cycles	***page-faults***	***L1-dcache-loads***	***L1-dcache-load-misses***
2294531192141	1	4261780204048	1313272671
2095228382257	1	4262169001172	1121855321
2074981166472	0	4262048544614	1548541675

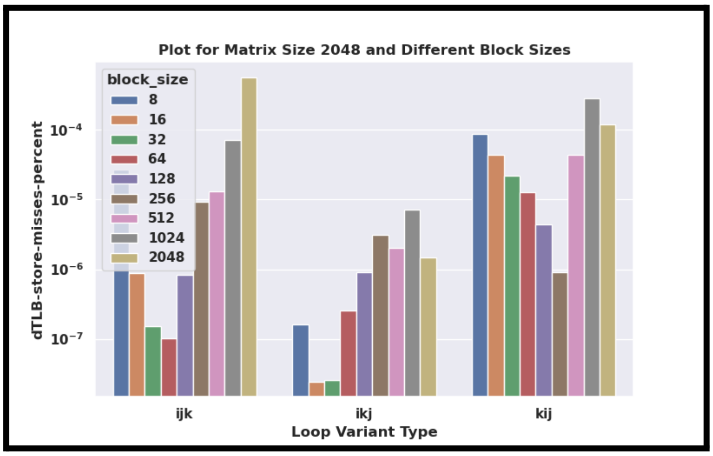
Table 2: Matrix - 8192 x 8192 | Tile-Size 32 | Rows are in order of ijk, ikj, kij

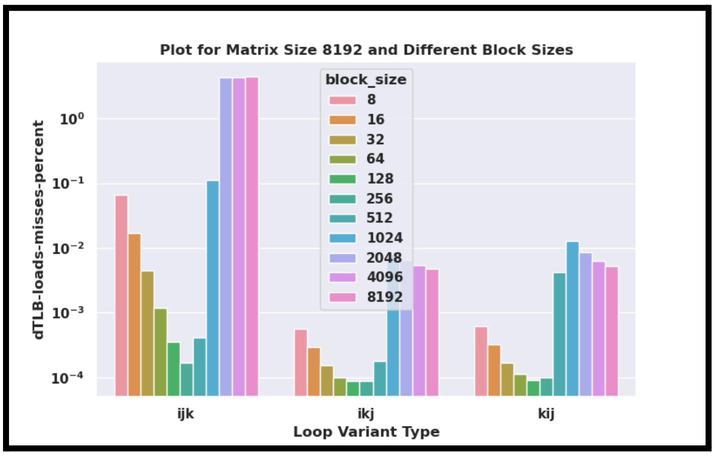
Final Conclusion :- To conclude, for **2048 matrix size with block size of 32, ikj variant** is performing better than its counterparts. While for **8192 matrix size with block size of 32, kij variant** is performing better than its counterparts.

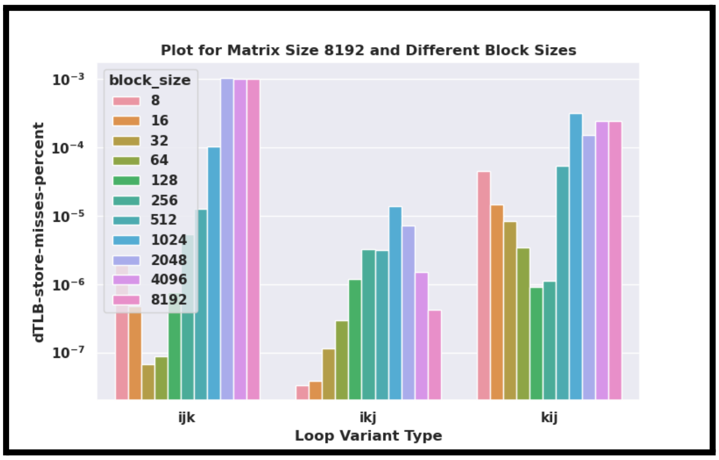
(1b) Comparing parameters for all the tile sizes:

1. dTLB:(Load and Store Misses)



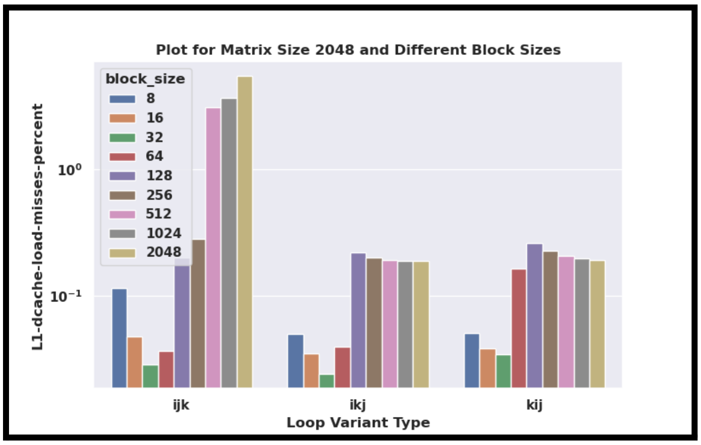


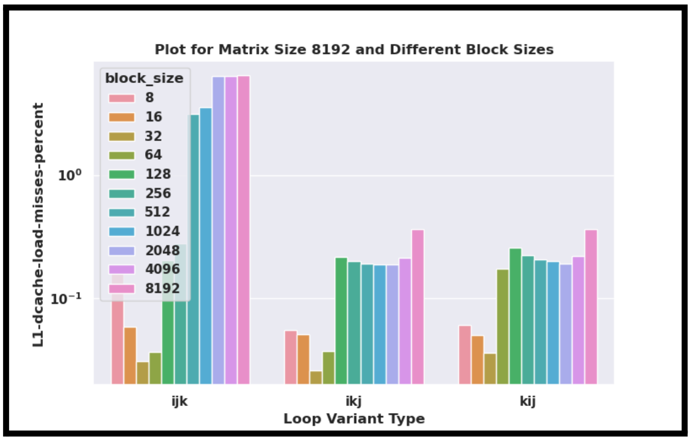




- Load Misses: On average **tile size 256 for 2048** is performing best, **tile size 256 for 8192** is performing the best.
- Store Misses : On average tile size 64 for 2048 is performing best, tile size 64 for 8192 is performing the best.

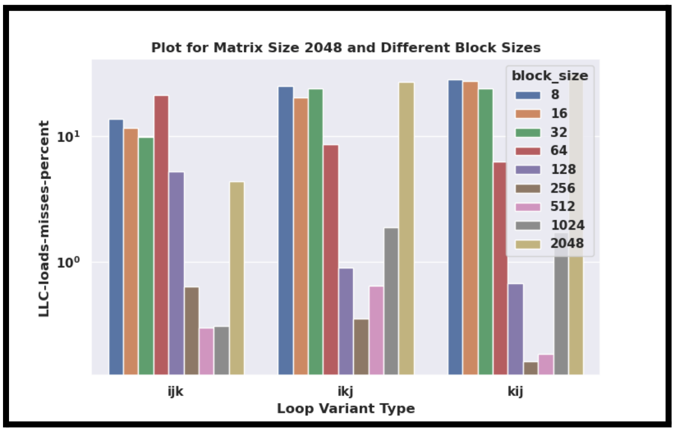
2. L1dcache:(Load Misses)

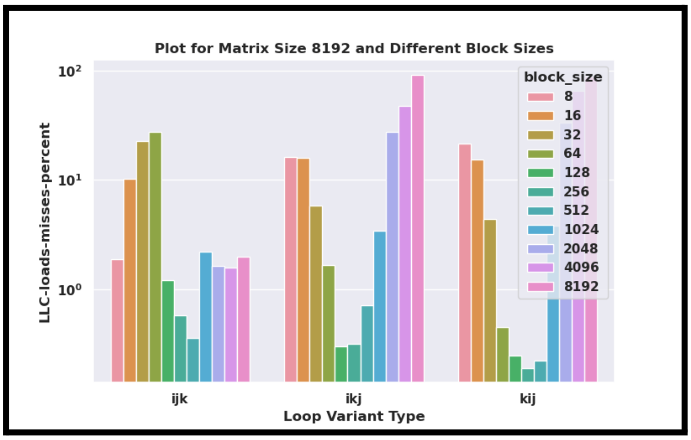


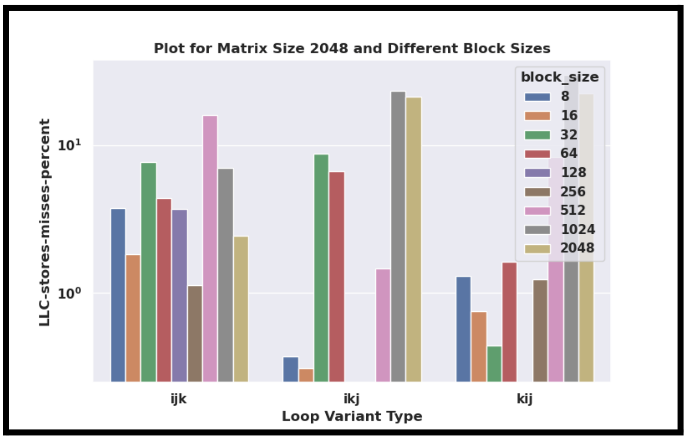


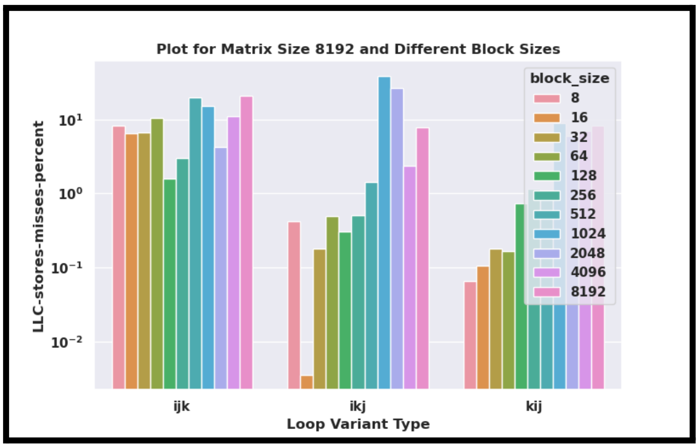
- **Tile size 32** for 8192 is better on average
- **Tile size 32** for 2048 is better on average

3. LLC



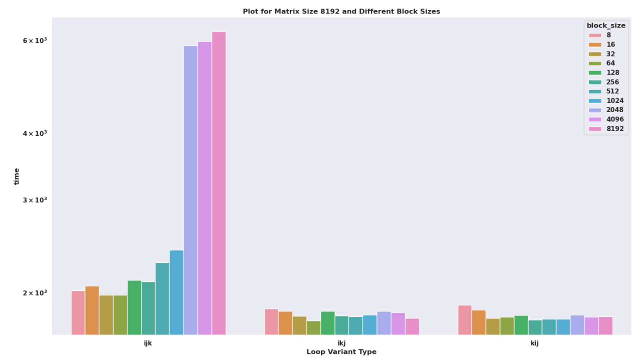




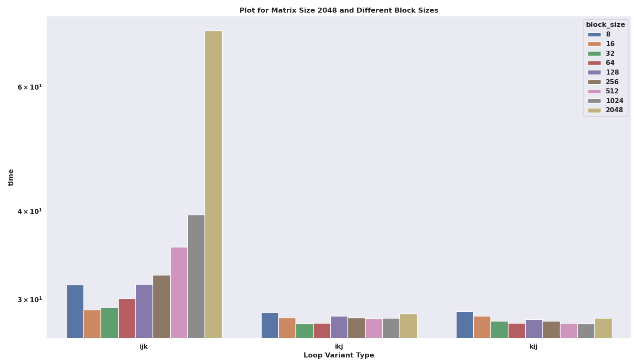


- Load misses: On average **tile size 256 for 2048** is performing best, **tile size 256 for 8192** is performing the best.
- Store misses: On average **tile size 256 (having 0 misses for ikj, kij) for 2048** is performing best, **tile size 128 for 8192** is performing the best.

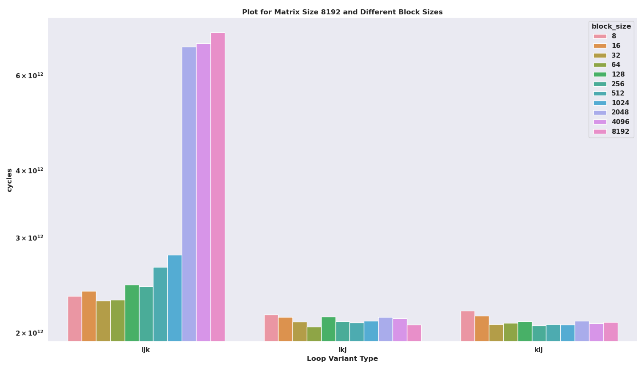
4. Total execution time & instructions:



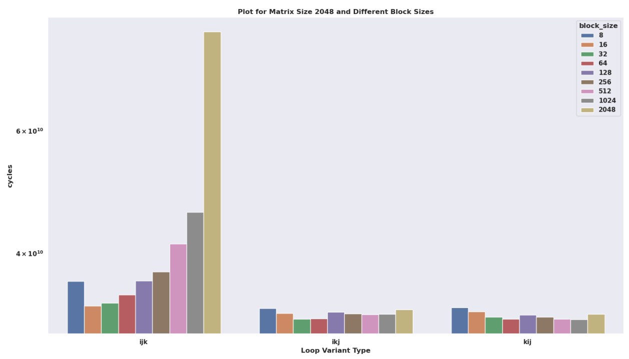
Execution time for 8192



Execution Time for 2048



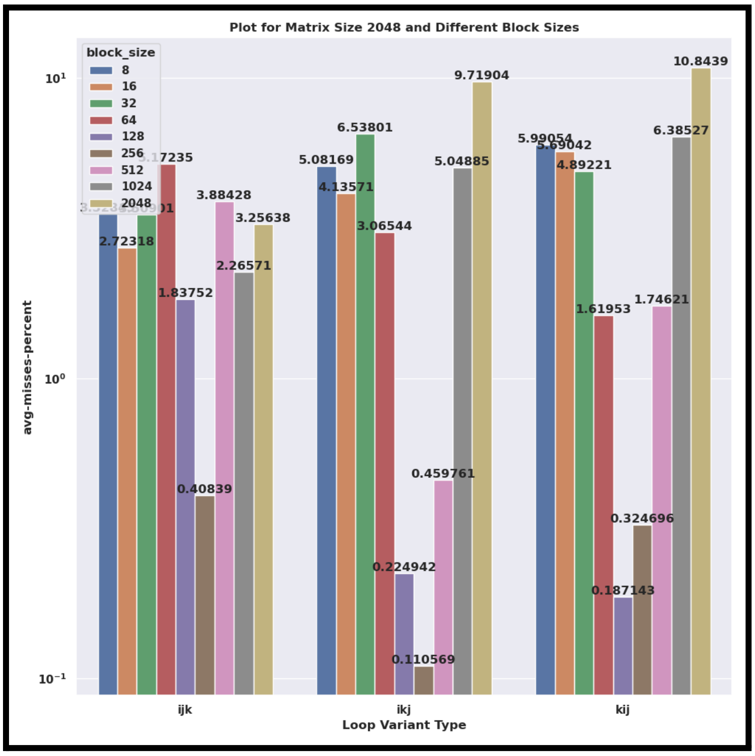
# cycles for 8192

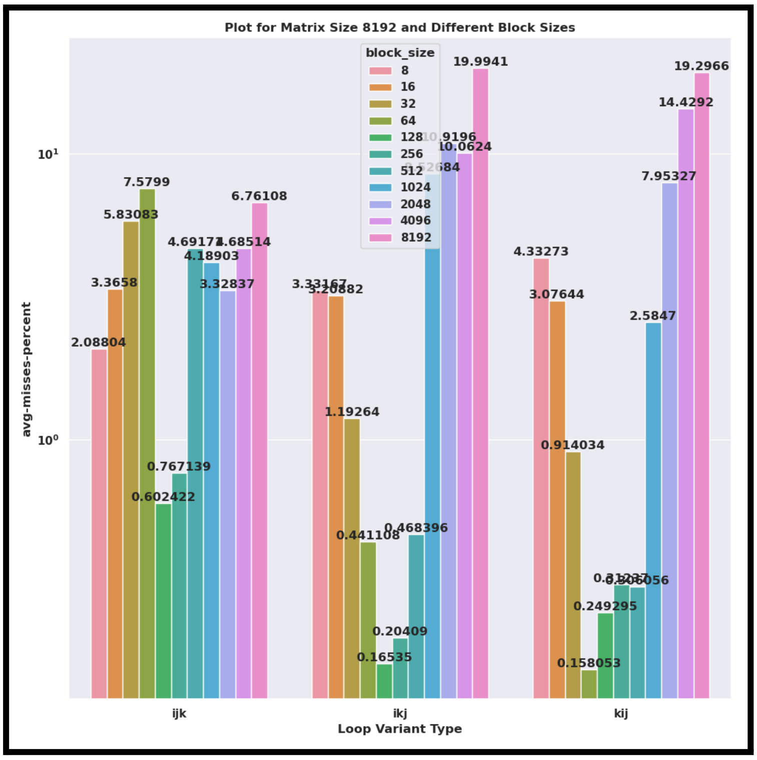


# cycles for 2048

- For 2048 tile 32 size has the best total execution and cycle time
- For 8192 tile 64 size has the best total execution and cycle time

5. Average Misses across all miss parameters:





- For 8192 tile size 128 performs the best on average.
- For 2048 Matrix tile size 256 performs the best on average
-

Final Conclusion :- Overall the tile size 256 performs the best across all the loop variants and matrix sizes, because if we incorporate the LLC misses and dTLB load and store misses which amount to most of the references generated. This concludes most of the references are data stores/loads to memory and there are high number of misses in top level caches due to which LLC is serving a lot of requests, the size of LLC in our system is 8MB

Question-2:

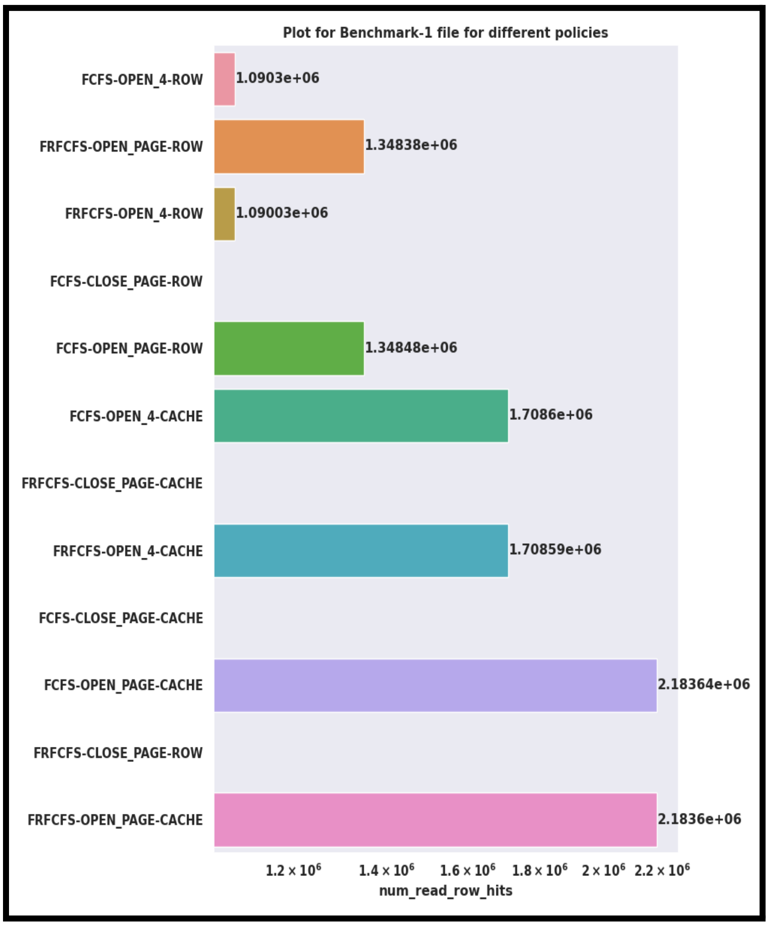
- We have modified the pintool and created our own custom program to generate the traces for all the loads and stores that misses the caches and went to main memory
- We ran the pintool on the three benchmarks and scaled down the timestamps as per below:
    - Cumulative diff(cd) = (timestamp of last instruction - timestamp of 1st instruction)
    - mean diff(md) = cd / # of instructions
    - (s)cale = 3
    - For every instruction (i > 1), **cycle = ((timestamp(i) - timestamp of 1st instruction) * s) / md**
- We modified the DRAM simulator to incorporate the cache interleaving addressing scheme by introducing lower and higher column bits in the config file and few other files.
- We implemented FCFS and Open_4 policy accordingly and ran the DRAM simulator on the above 3 traces using various combinations of addressing, row-buffer policies and memory scheduling policies.
- Below are the graphs for different benchmarks and the corresponding observations.
- Bank level parallelism is calculated as follows
    - 1 - (average of idle cycles of all banks in different ranks/number of cycles)
- Average memory access time (avg_mmr_access_time) is calculated as follow
    - # of reads done/(total reads and writes) * average_read_latency + # of writes done/(total reads and writes) * average_write_latency.

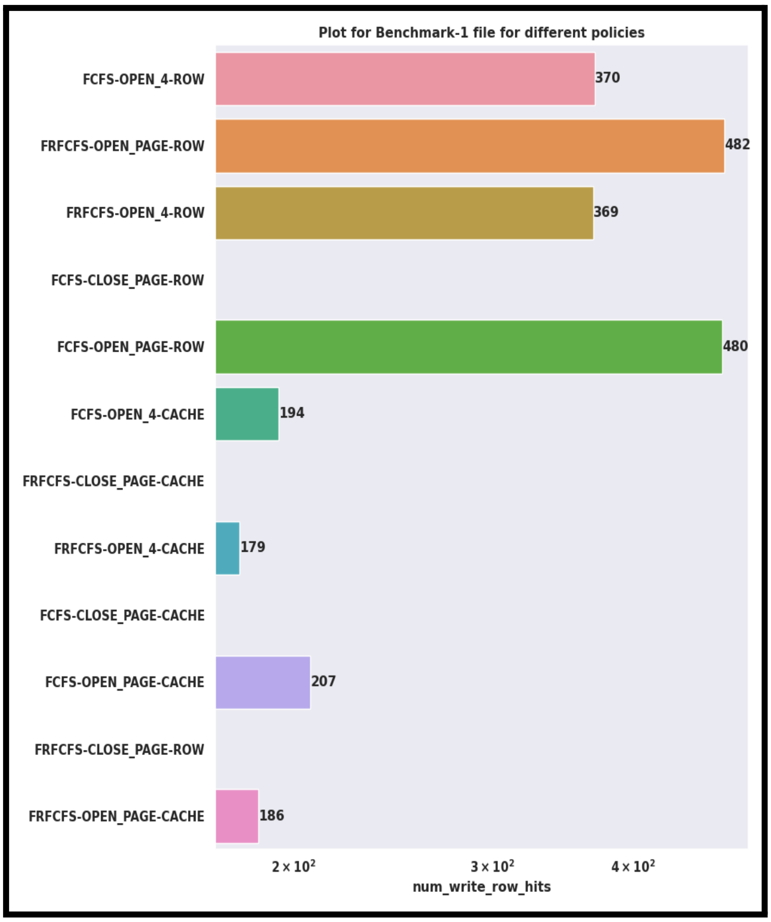
Benchmark-1

- For the Read Row hits FCFS along with Open_4 with cache interleaving outperforms others.

- For the Write Row hits FCFS/FRFCFS along with Open_PAGE with row interleaving give comparable performance and outperform the others.

- Close Page policies have 0 hits in both cases, which is expected since we precharge the row buffer immediately after a read/write to a row.

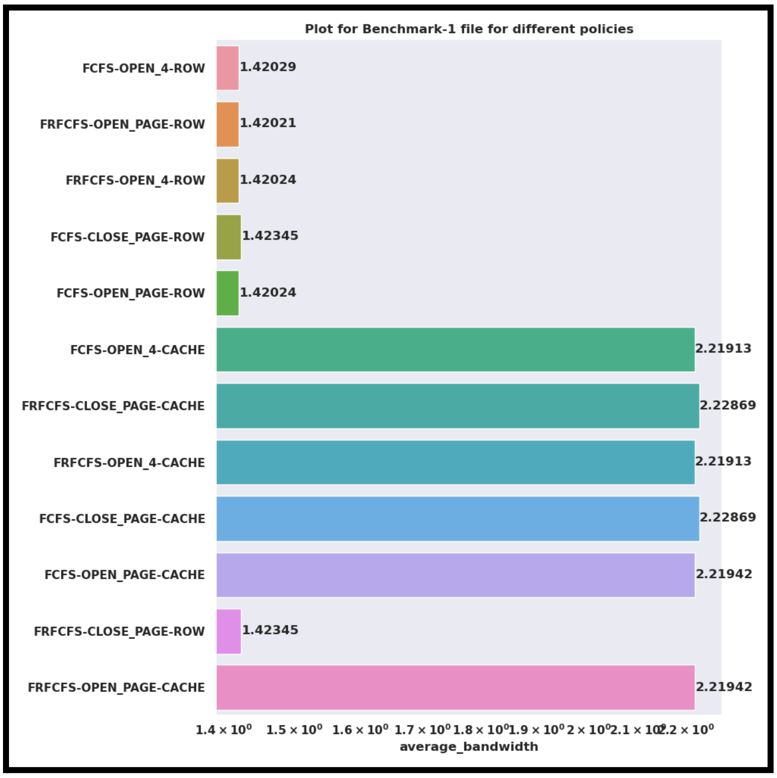


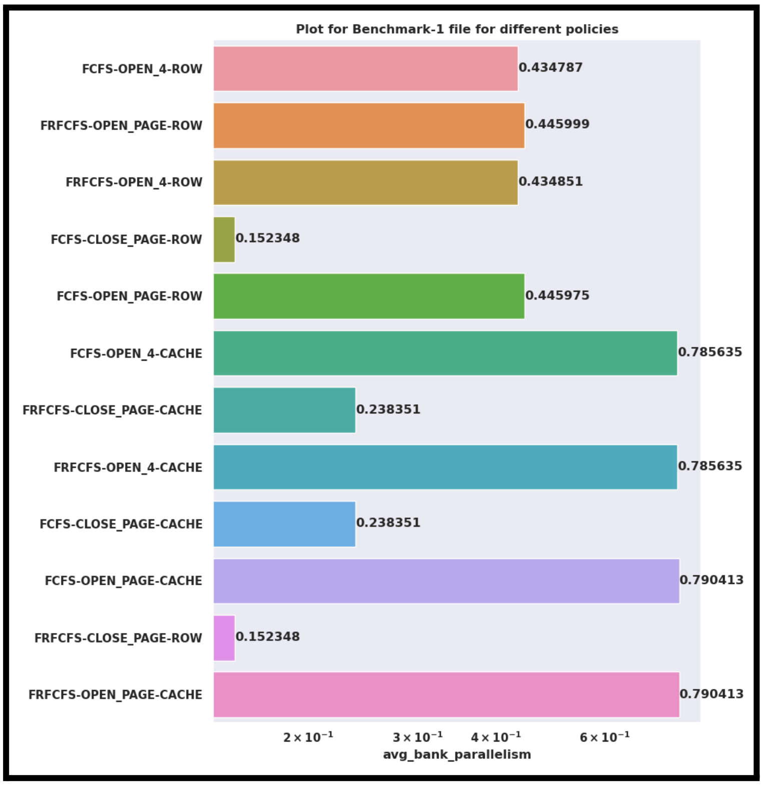


- High Bank-Level Parallelism is exhibited by Open_4 and Open Page policies with cache interleaving addressing scheme.

- Close page with FCFS and row interleaving offers close to no Bank-level parallelism.

- All the variants of Cache-interleaving addressing give really high bandwidth as compared to that of Row-interleaving.

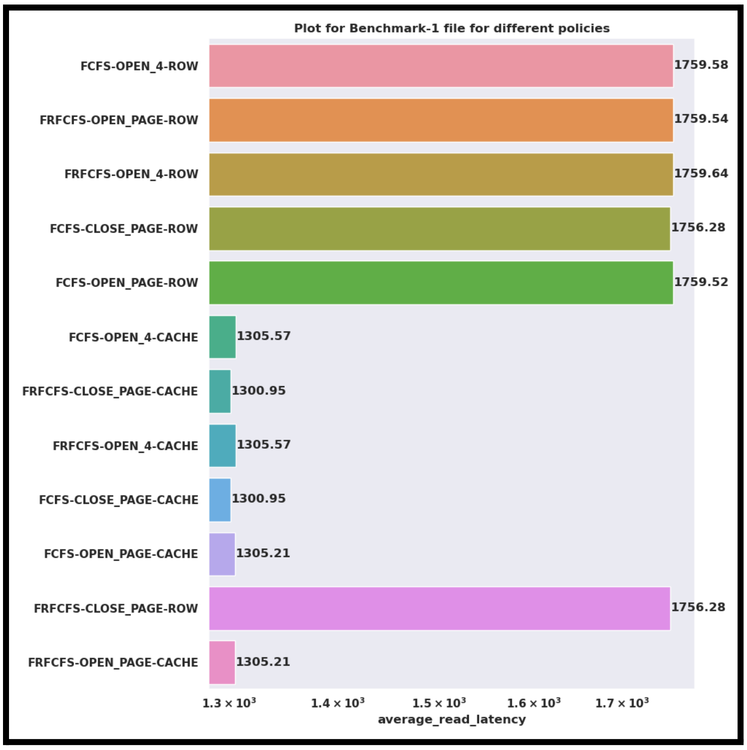


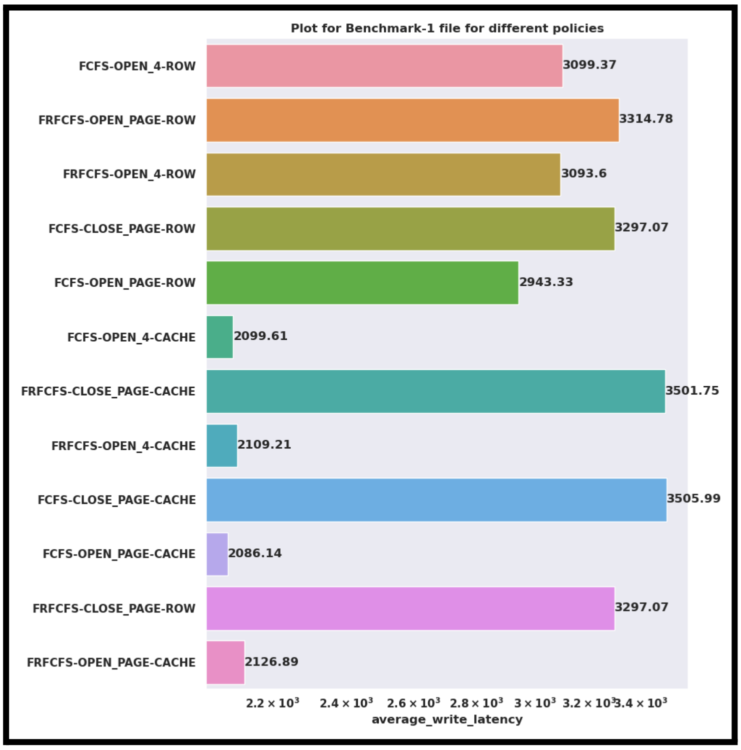


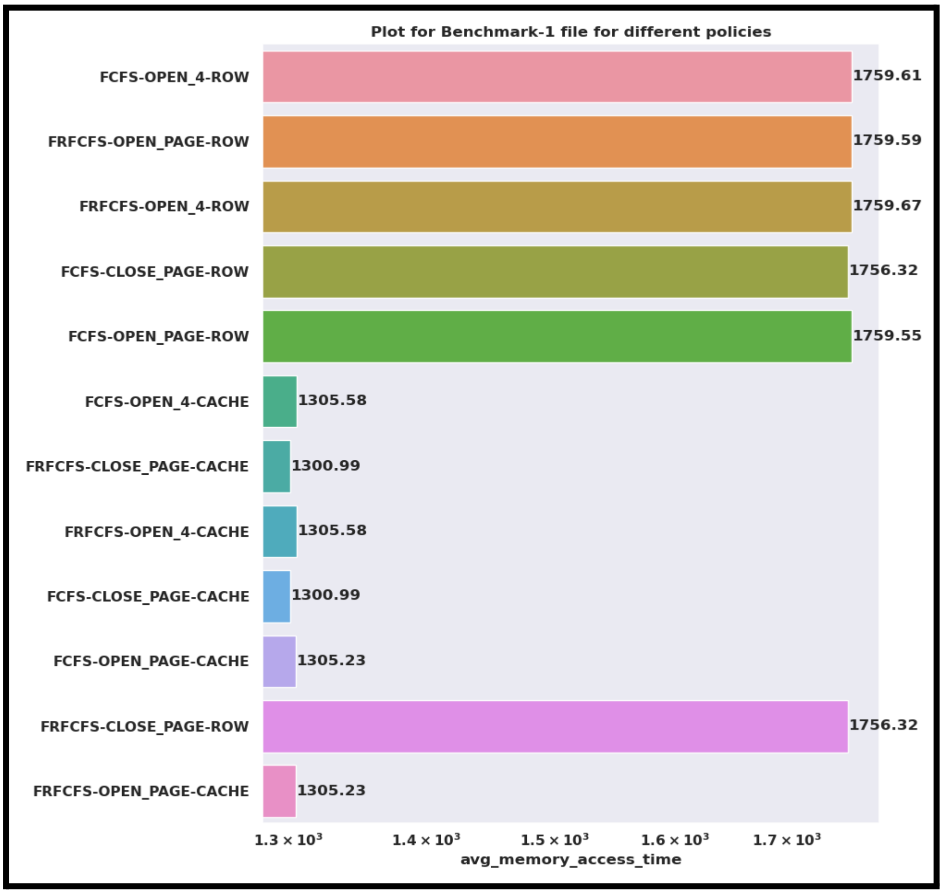
- Read Latency is high for row-interleaving policy as compared to that of Cache-interleaving.

- For Write latency the row-interleaving policies have high latency, but Cache-interleaving policy with FCFS/FRFCFS with Close page shows even higher write latency, higher than row interleaved.

- Close Page policies have higher latency since we are immediately precharging rows that adds to more pre-charge/activate in case of simultaneous accesses to the same row and bank.







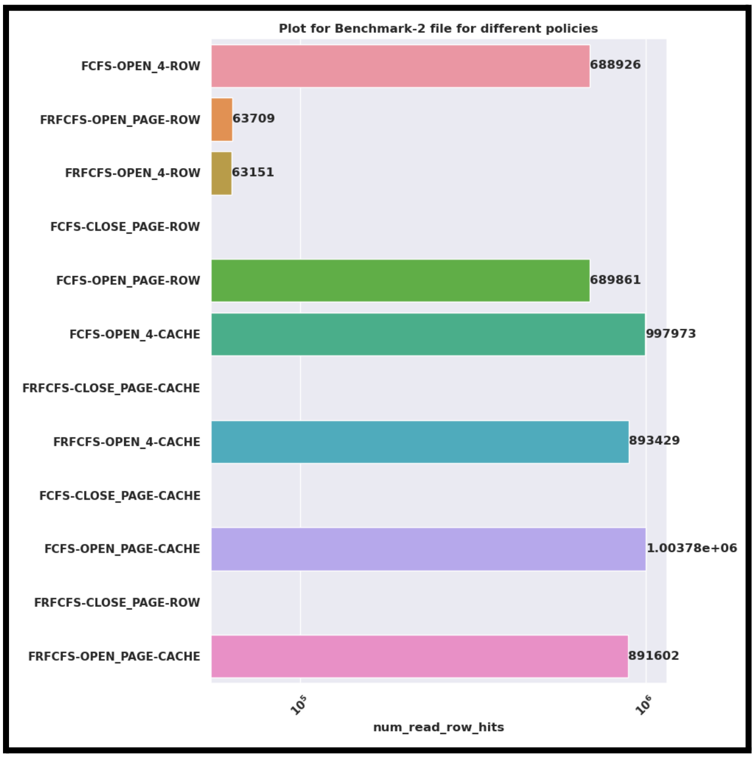
- Based on the above results we can conclude Cache-interleaving addressing provides us with least latency, highest Bank-level parallelism and average bandwidth.

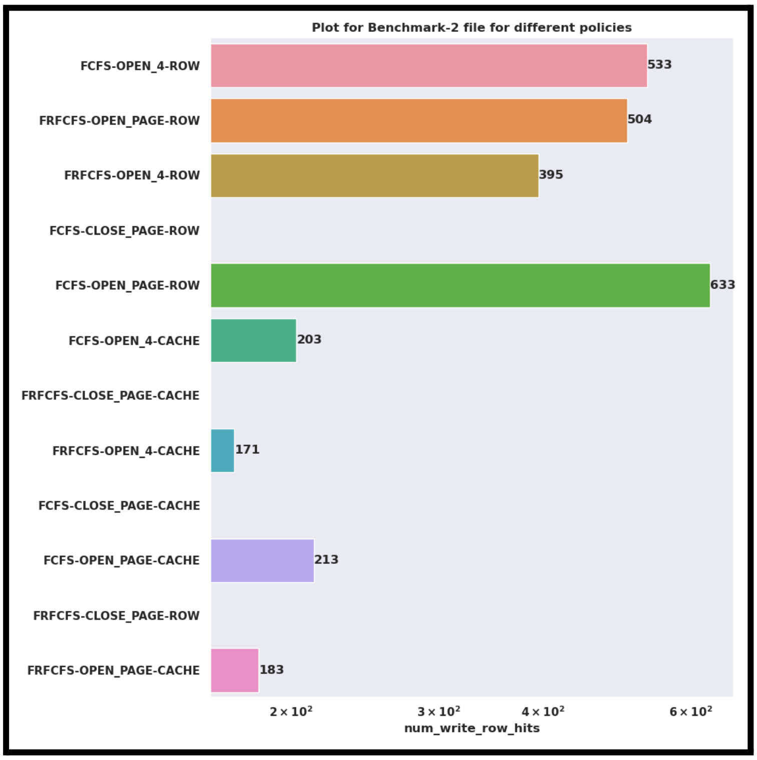
- Open-Page with FCFS/FRFCFS gives the highest number of read row hits followed by Open-4 policy. (Write row hits are very low as compared to read since this benchmark is read heavy (28618370 reads comprated to 506 writes).

Conclusion for Benchmark-1 :- Open Page with FCFS/FR-FCFS using cache interleaving gives the best performance for Benchmark-1 program compared to other policies and addressing schemes.

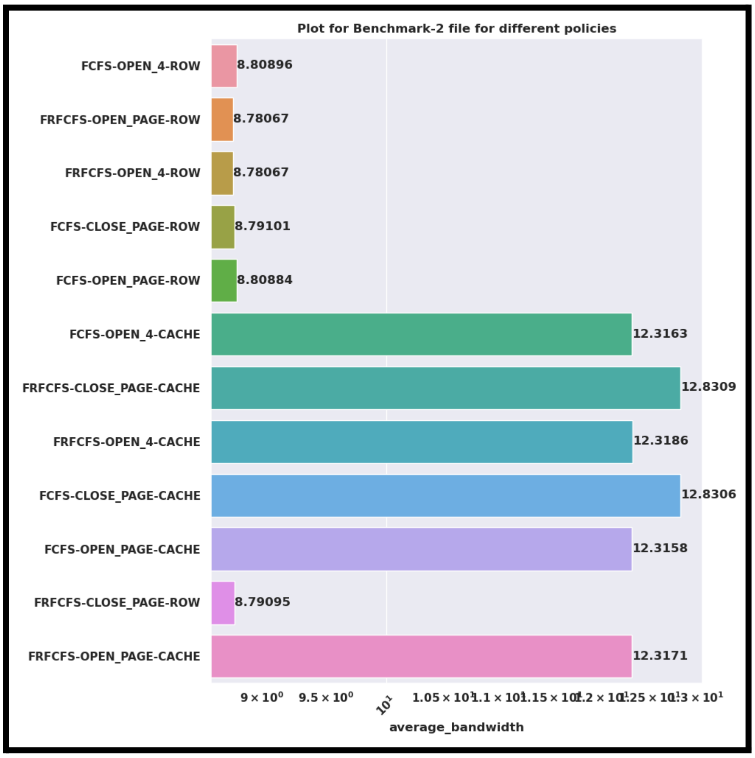
Benchmark-2:

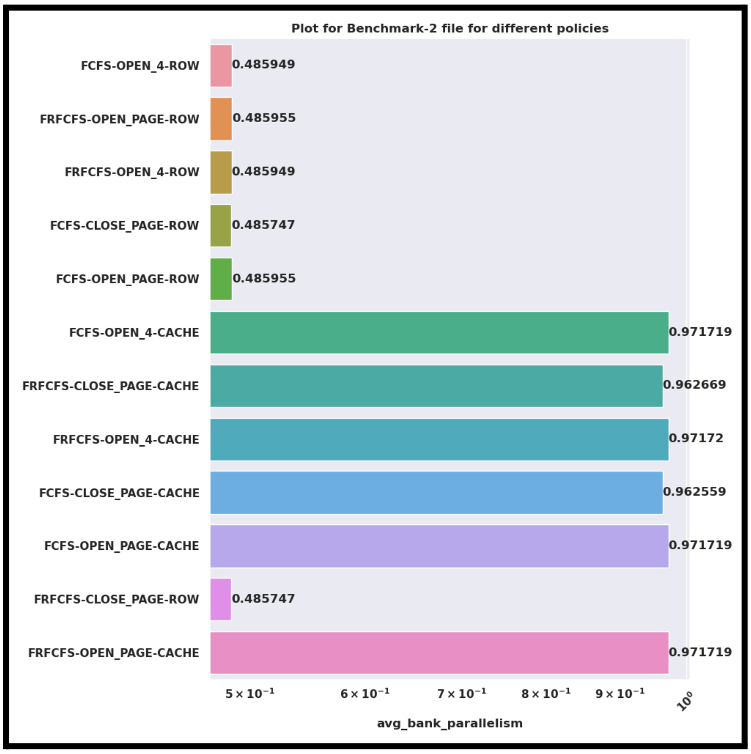
- Cache interleaving addressing yield the highest read row-buffer hits.
- For Row interleaving we do see FCFS along with OPEN_4 and OPEN_PAGE having high read row buffer hits.
- Contrastingly for write buffer hits, all Row-interleaving policy have high numbers as compared to cache interleaving even though the benchmark is read heavy.



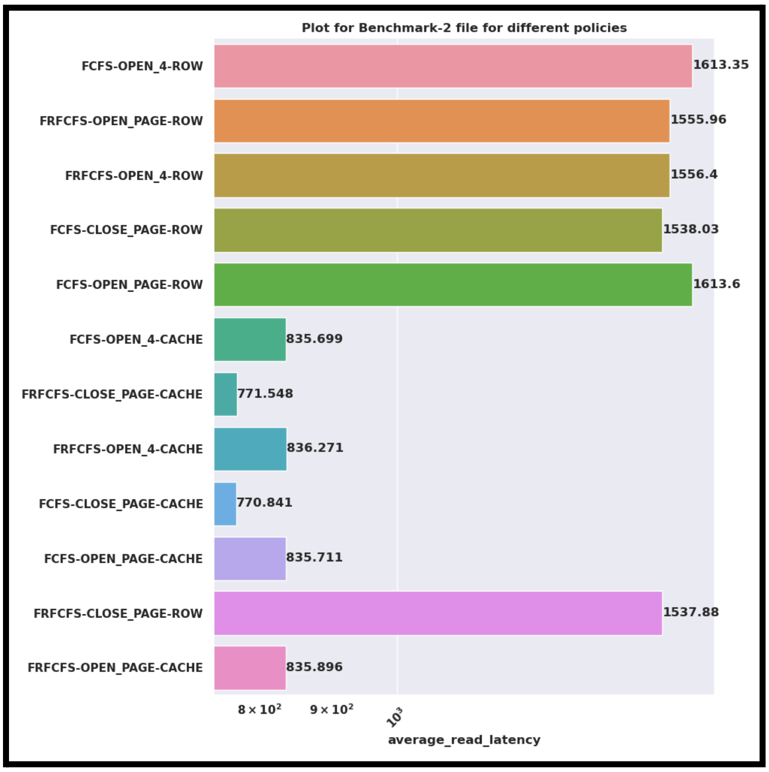


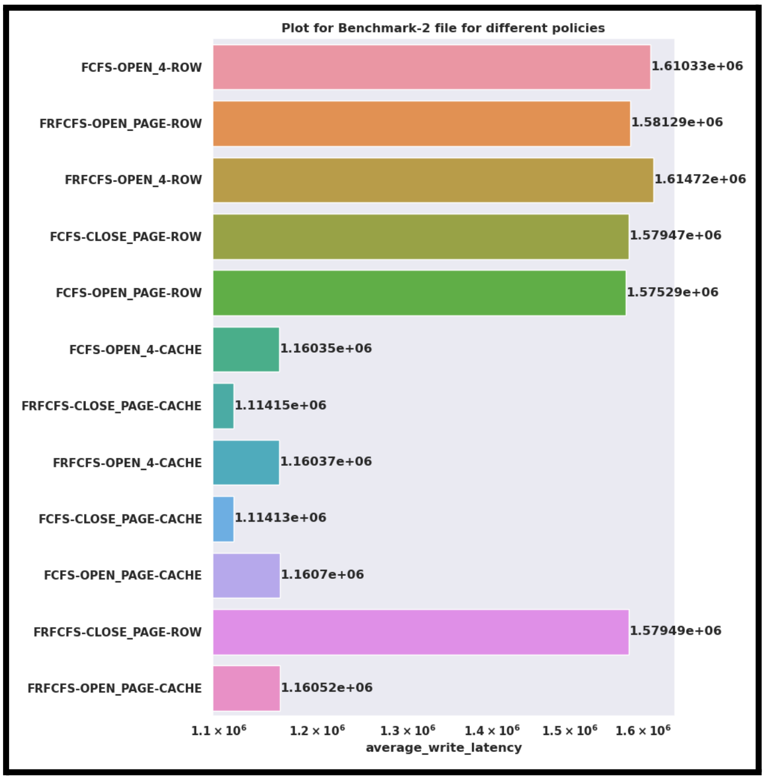
- Cache interleaving policy exhibit high average Bank Parallelism and bandwidth as compared to row interleaving.

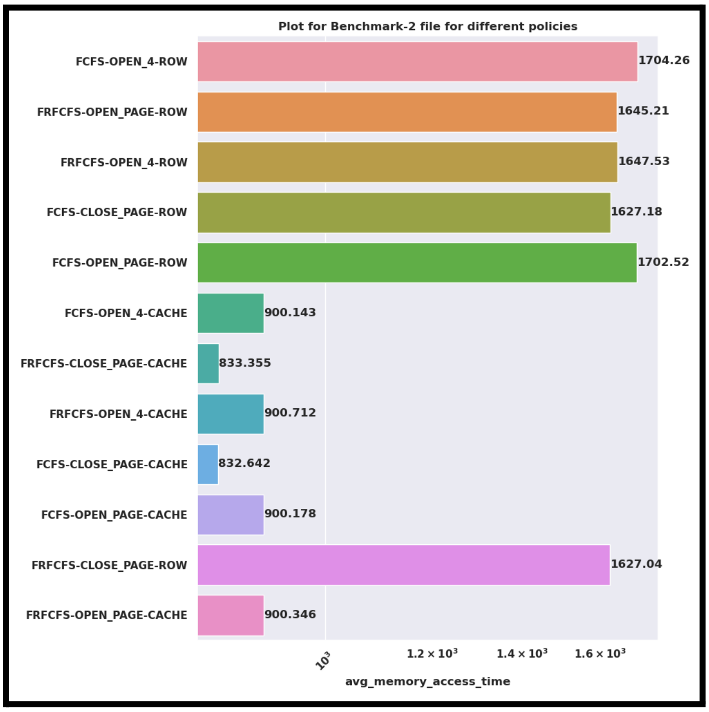




- Row interleaving policies read latencies are more than 2x than that of Cache interleaving policy, and the write latencies are also higher than its counterpart.







- With the above observations, we conclude that Cache interleaving policy yields the better performance along with FCFS/FR-FCFS and OPEN_PAGE/OPEN_4 policies.

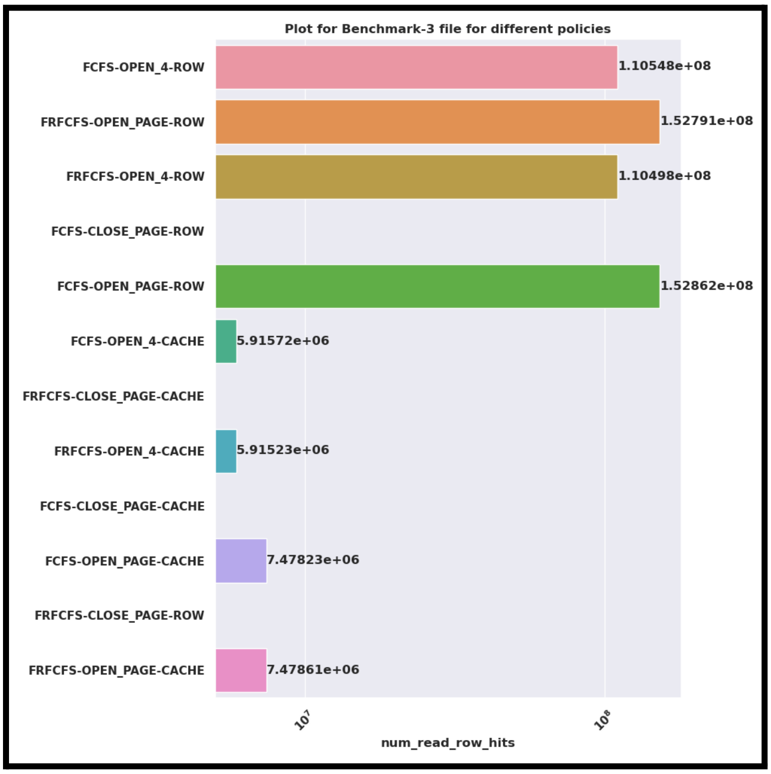
- The above mentioned policies only yield low write buffer hits, but since the benchmark being read heavy outperforms the other policies as measured on different parameters.

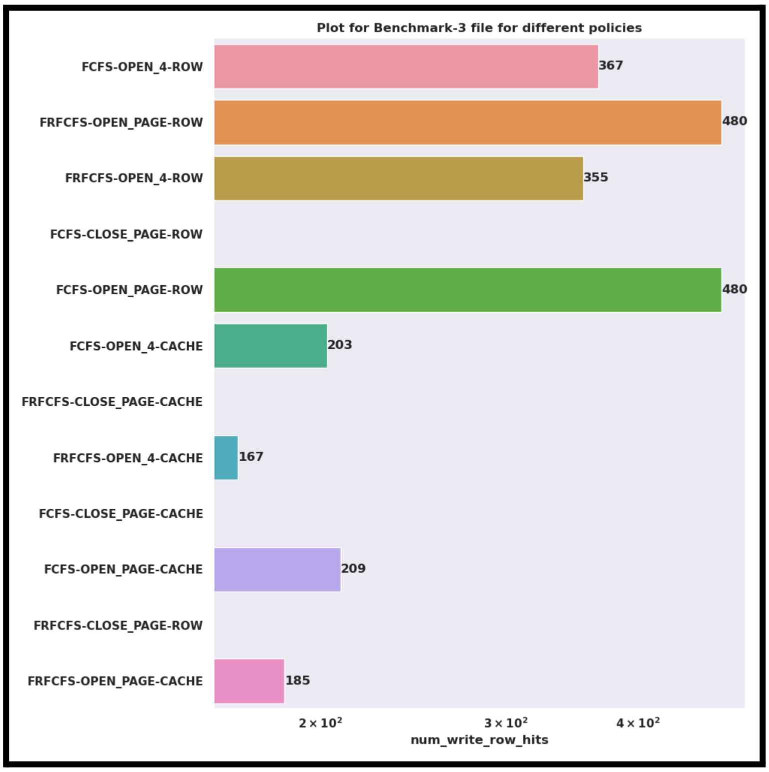
Conclusion for Benchmark-2 :- FCFS/FR-FCFS with OPEN_PAGE/OPEN_4, cache interleaved addressing scheme performs the best compared to other schemes.

Benchmark-3:

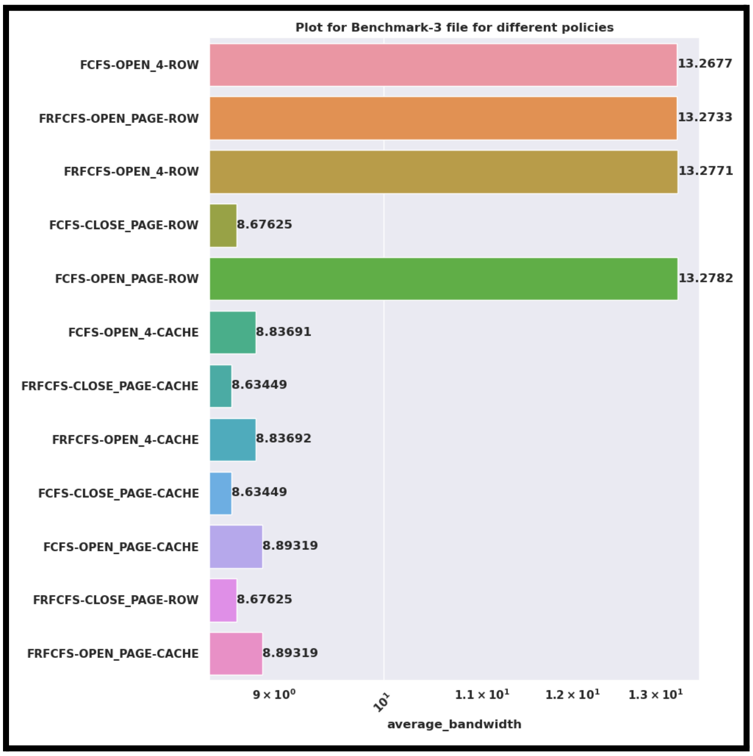
- Row interleaving addressing gives the highest read row buffer hits, with Open-page outperforming Open-4 by 25%.

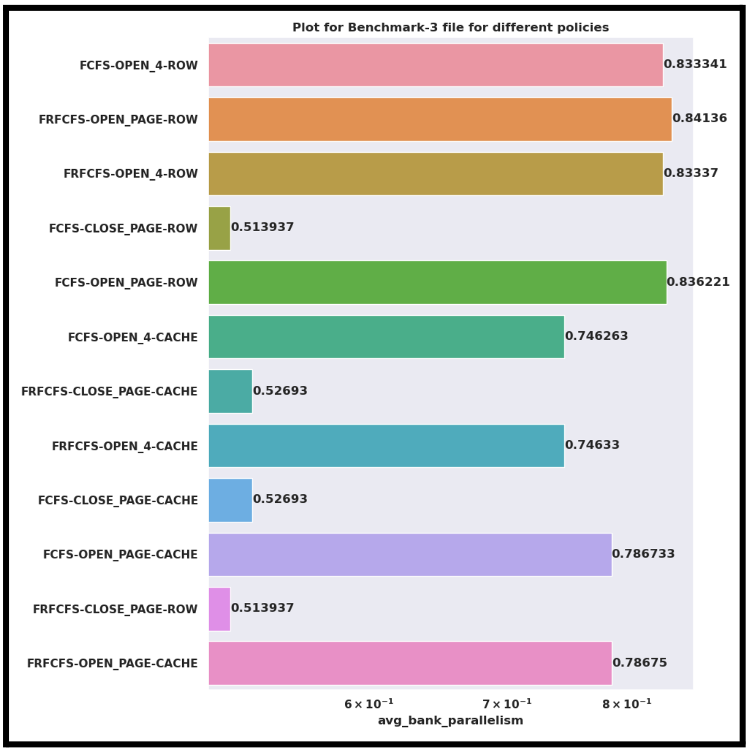
- Similarly for write buffer hits, Row interleaving gives the best performance.



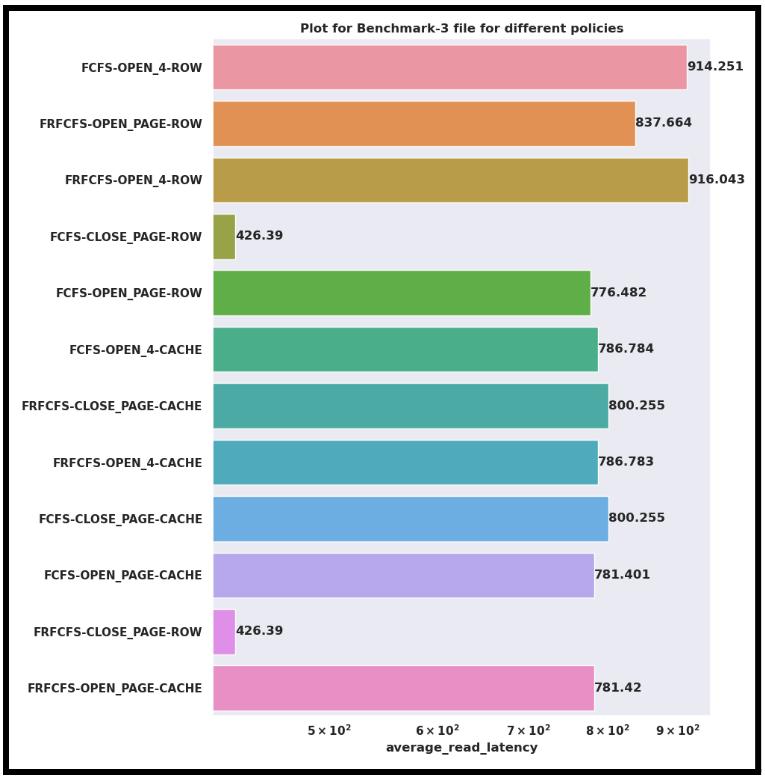


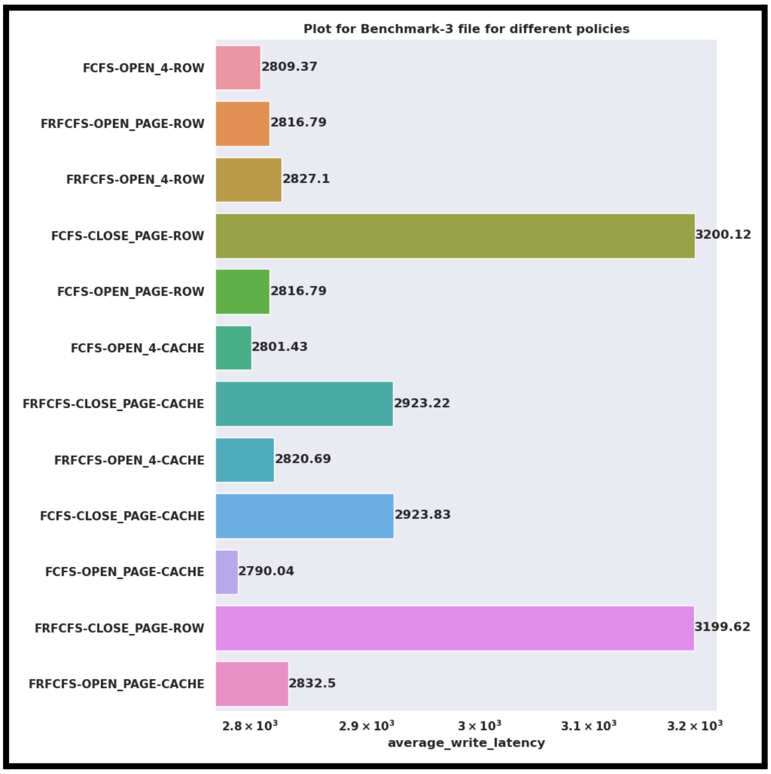
- Row interleaving policies provide the highest average bandwidth and bank level parallelism.
- Cache interleaving exhibit significantly high bank level parallelism but less(~7 to 10%) than Row interleaved.

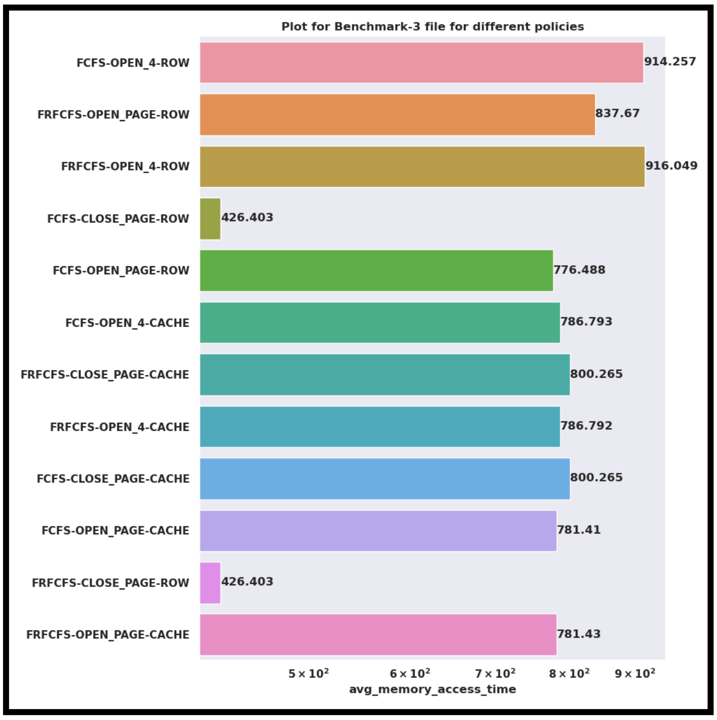




- Read latencies are high for all benchmarks except for Row interleaved, Close page
- Write latencies are abruptly high for row interleaved, close page.
- Average memory access times for row interleaving are higher (~13%) than cache interleaved.







- Bank-Level parallelism is comparable amongst the row and cache interleaved addressing modes.

- Overall the Row interleaved policy performs the best with only the read/write latencies and average memory access times being the worst performing parameters.

- While designing a DRAM we must look for higher Bank level parallelism, bandwidth and row buffer hits. Hence the Row interleaved addressing wins here.

Conclusion for Benchmark-3 :- FCFS/FRFCFS with OPEN_PAGE along with row interleaving performs the best here.

Final Conclusion:

- Closed page policies will have the lowest buffer hits as they pre-charge the page immediately after read/write.

- For most of the benchmarks we see that Open-Page slightly out-performs Open-4 policy. This is expected as Open-4 policy is a strict modification to that of Open-page where we pre-charge the row as soon as 4 requests are served, whereas in Open-page we keep the row open if we have more pending requests to the same row irrespective of the requests served, which can potentially help us increase buffer hits and reduce latency with fewer precharge and activate.