Cache Simulator - CO262 Course Project

Submitted by :

1. Ankit Jain (17CO208)
2. Arqum Shaikh (17CO241)
3. Vybhav Pai (17CO252)
4. Shrinidhi A Varna (17CO145)
5. Chaitany Pandiya(17CO112)
6. Avakash Bhat (17CO110)
7. Abhinav P Y (17CO103)

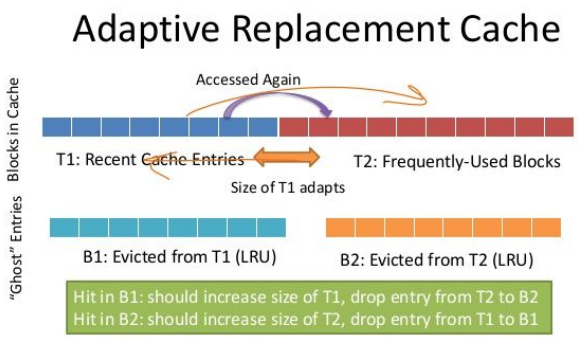
Introduction

The Cache simulator will take a trace of memory accesses, type of algorithm, number of ways and block size as input and it will output the hit ratio. Pinatrace program which is a pin tool, was used to generate the trace, which is a sequence of program counter values and corresponding memory accesses along with the type of accesses(read or write) using this we can output the number of reads and writes. The trace was generated for Double precision General Matrix Multiplication(DGEMM), DGEMM with row-major accesses only and Tiled DGEMM. Given below is the individual contribution for this project.

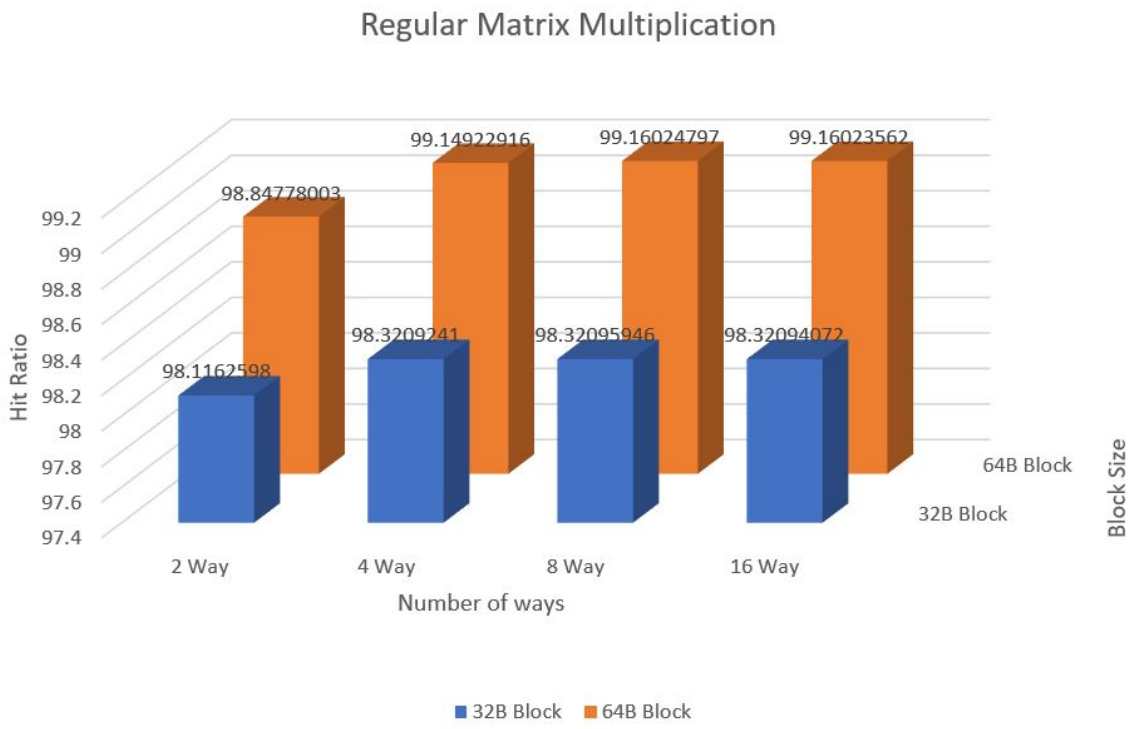
1. Ankit Jain

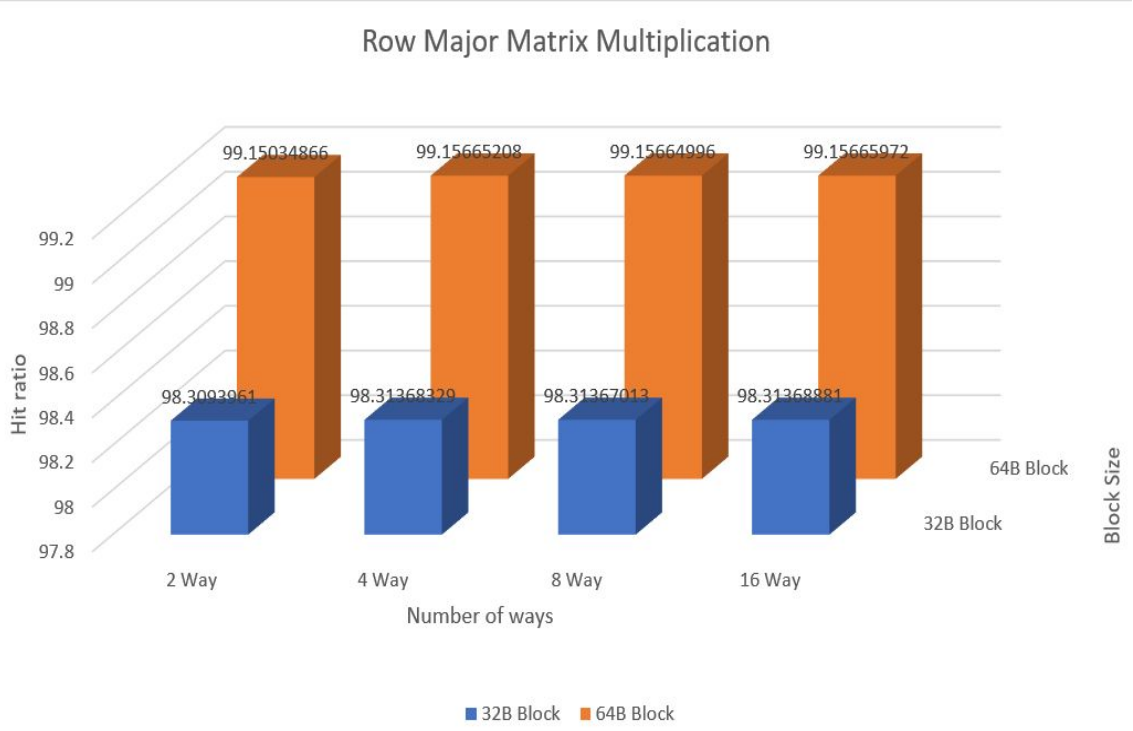
Contribution: ARC cache replacement algorithm and generating **trace for matrix multiplication with row-major accesses only**.

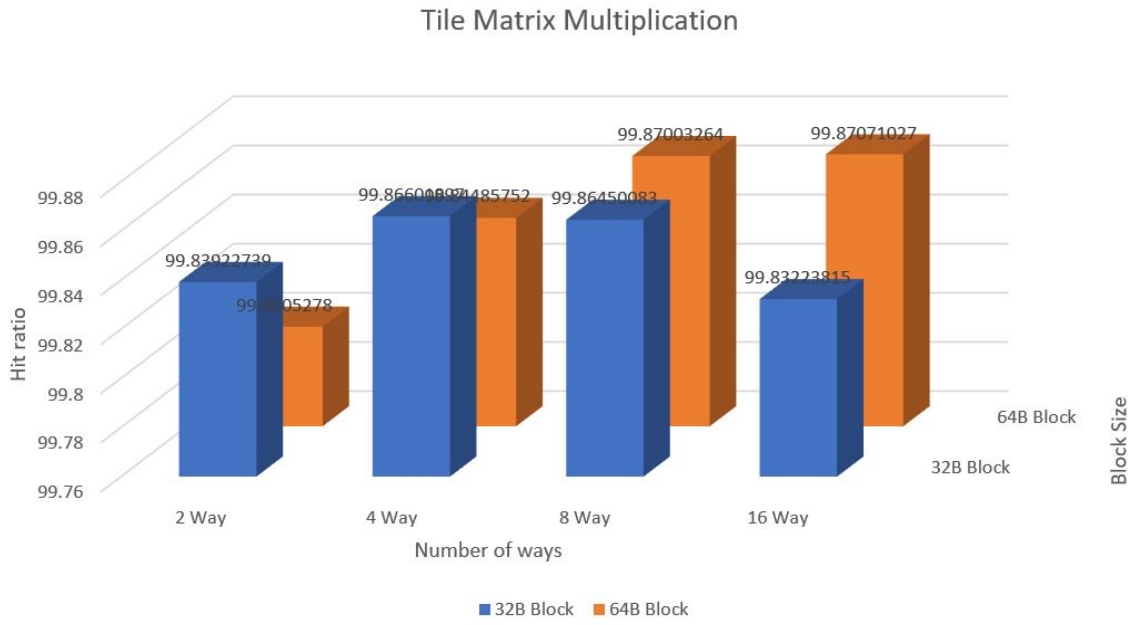
Adaptive Replacement Cache (**ARC**) is a state-of-the-art "adaptive" cache replacement algorithm invented to improve on the shortcomings of classical cache replacement policies such as LRU and LFU. This is accomplished by keeping track of both frequently used and recently used pages plus a recent eviction history for both (Ghost Entries). By separating out items that have been accessed only once and items that have been accessed more frequently, ARC is able to control the harmful effect of single-access items



flooding the cache and pushing out more frequently accessed items. In response to evolving and changing access patterns, ARC dynamically, adaptively, and continually balances between the recency and frequency components in an online and self-tuning fashion. The policy ARC uses a learning rule to adaptively and continually revise its assumptions about the workload ARC has been shown to outperform its classical and popular counterparts in practice. The algorithm was developed at the IBM Almaden Research Centre. In 2006, IBM was granted a patent for the adaptive replacement.
The plots of Hit Ratio vs Number of ways and Block Size for various Matrix Multiplication programs are shown in the following figures.







Analysis of results on ARC policy:

- The replacement algorithm works best for 8 way or 16 way set associative cache with 64B block size.
- Trace for Tile matrix multiplication gives the best hit ratio for both 32B and 64B cache block.

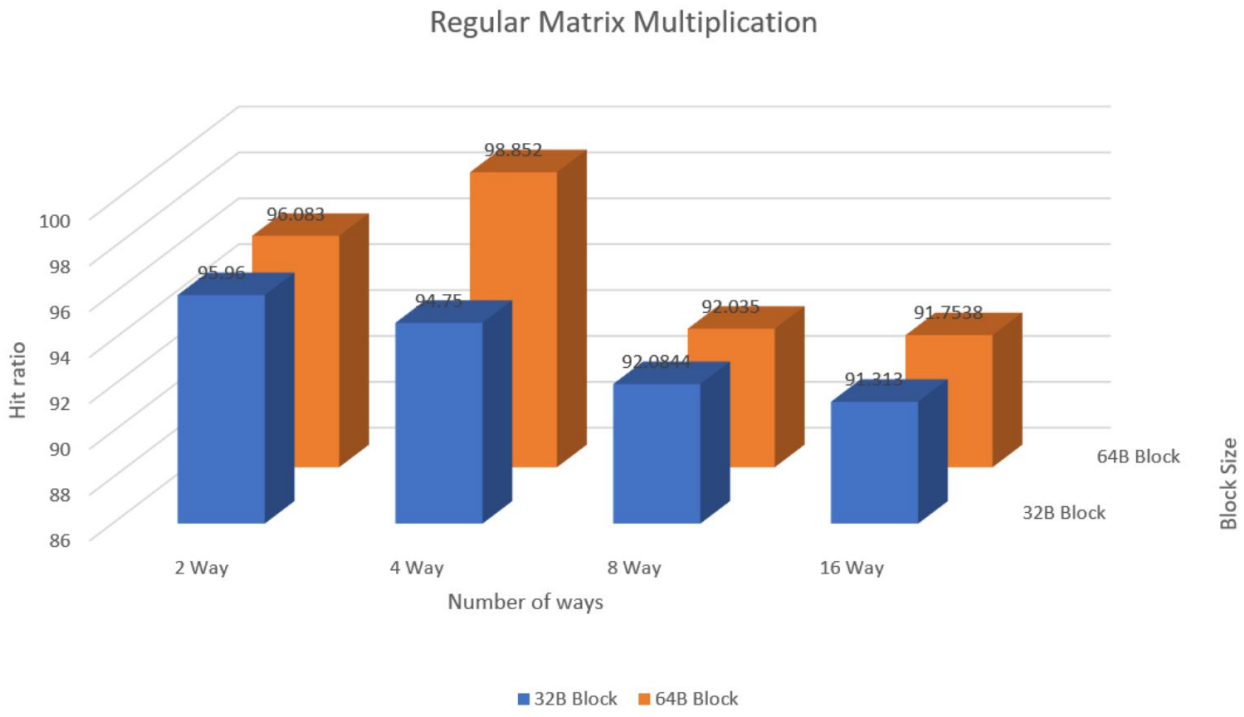
- Hit ration for 64B is always better except in the case of Tile matrix multiplication where 32B hit ratio is better for 2 and 4 way set associative cache.

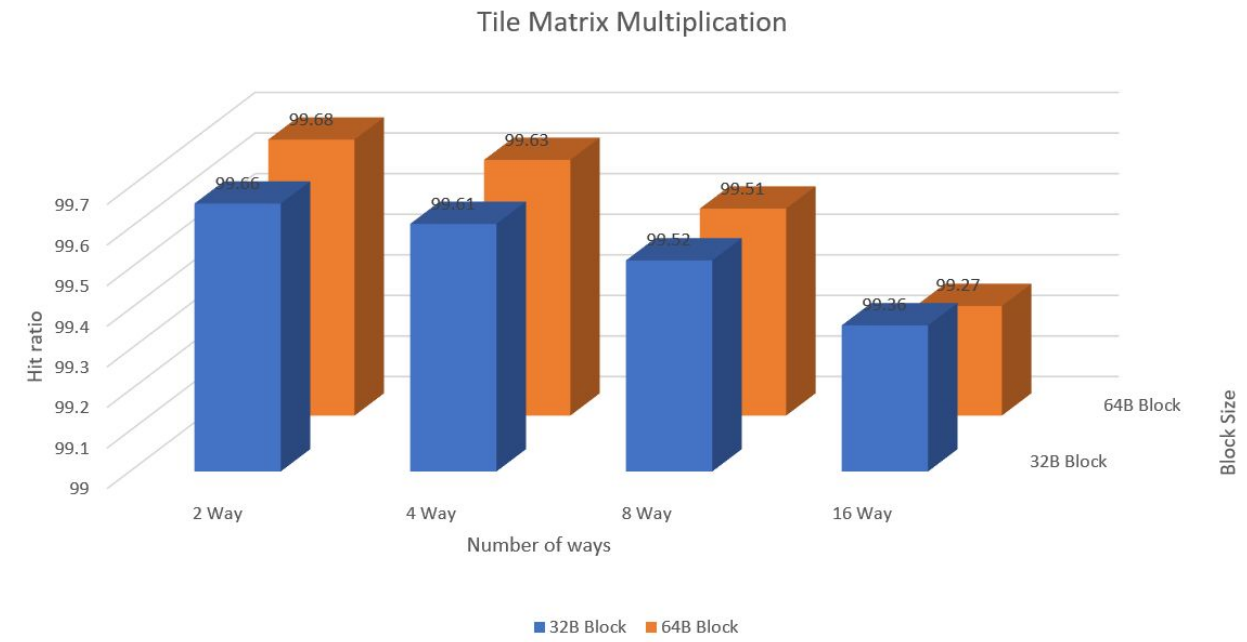
2. Arqum Shaikh

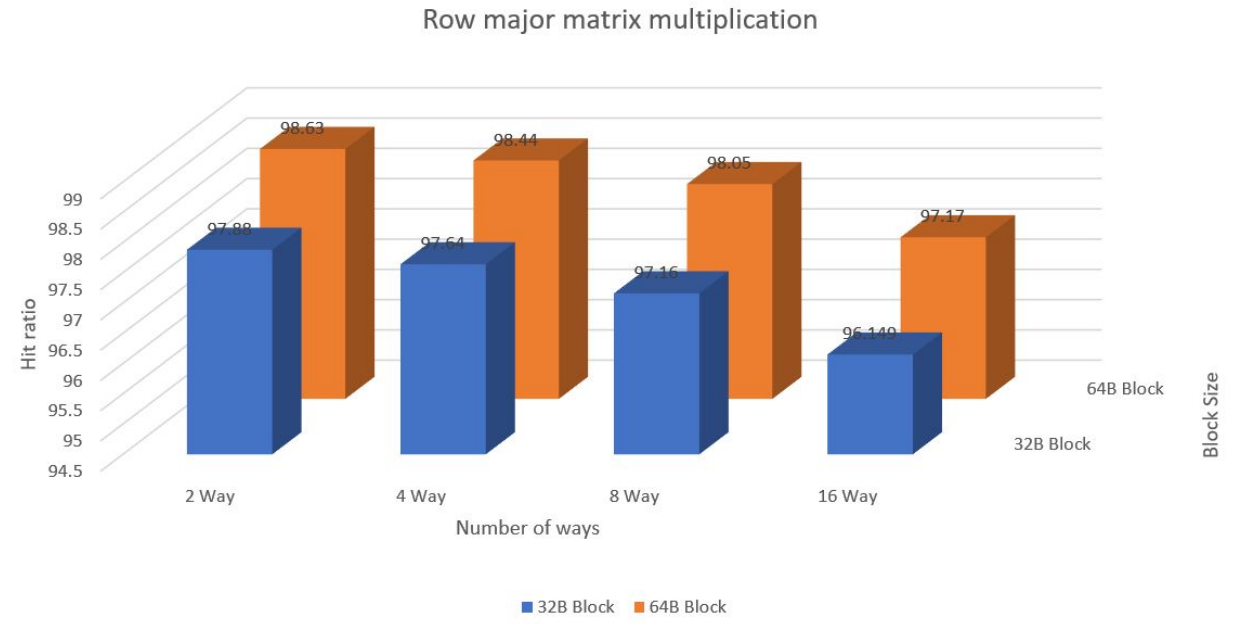
Least Frequently Used (**LFU**) is a type of cache algorithm used to manage memory within a computer. The standard characteristics of this method involve the system keeping track of the number of times a block is referenced in memory. When the cache is full and requires more room the system will purge the item with the lowest reference frequency.

Analysis of Hit Results on various trace simulations and parameters for LFU:

1. The best performance of Cache is observed in the case of tiled matrix multiplication trace, with results as high as 99%.
2. The next highest performance was observed in the case of Row major multiplication trace, with results at 97-98%.
3. The worst performance was observed in the case of Regular Matrix Multiplication trace with results falling as low as 90%.
4. As a general trend, increasing the number of ways of the set associative cache has decreased the cache performance.
5. 64B blocks showed relatively better performance as compared to 32B blocks.



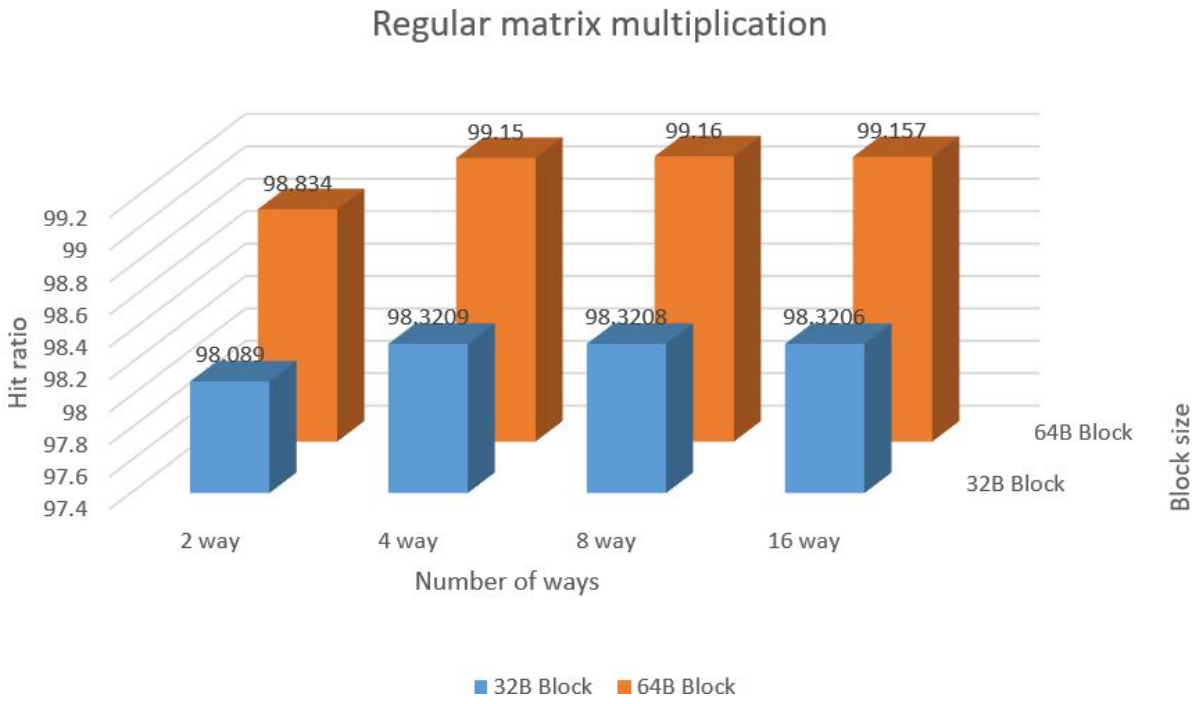


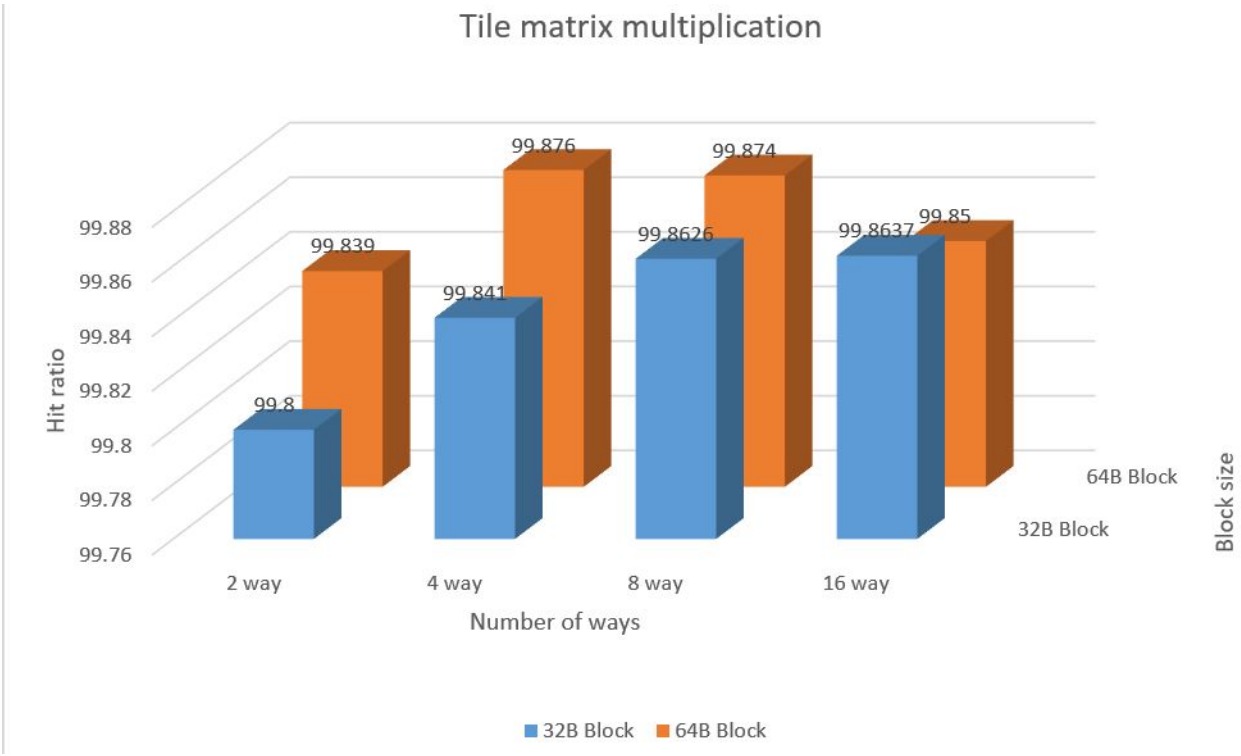


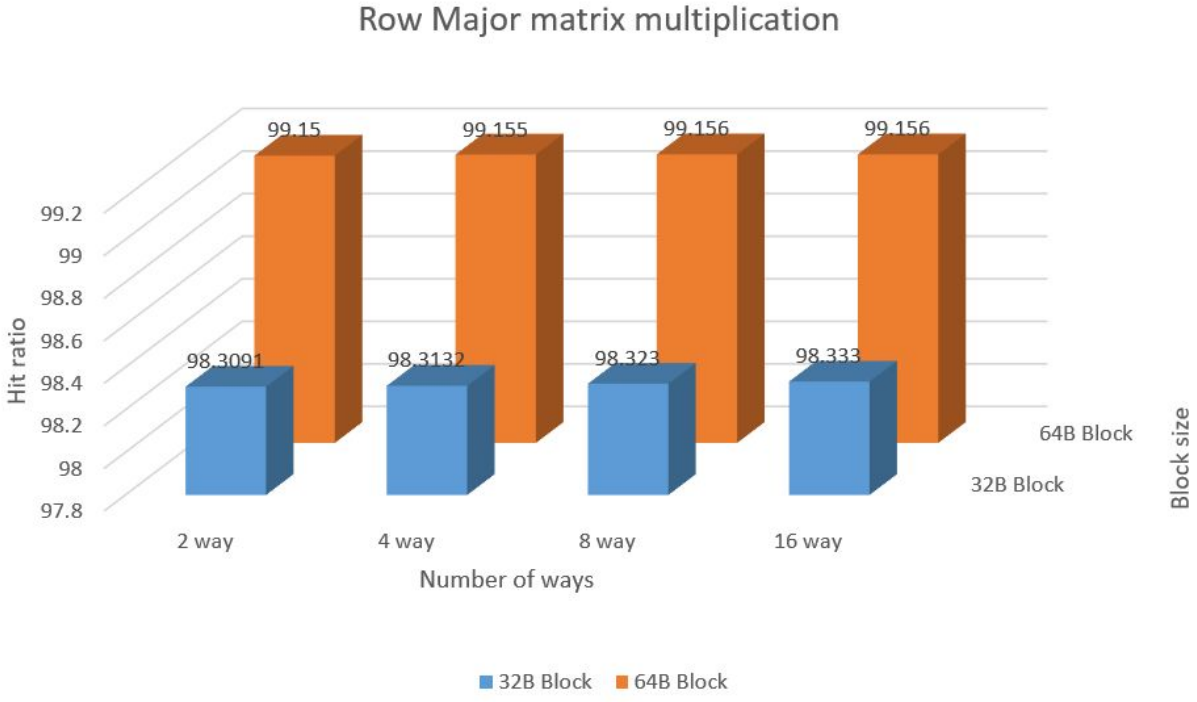
3. Vybhav Pai

Contribution: NRU cache replacement algorithm and simulator.

Bit-PLRU or **NRU** stores one status bit for each cache line. We call these bits MRU-bits. Initially all cache line MRU bits are set to 0. Every access to a line sets its NRU-bit to 1, indicating that the line was recently used. Whenever the last remaining 0 bit of a set's status bits is set to 1, all other bits are reset to 0. This means at every instance at least one line in the set has status set as 0. At cache misses, the line with lowest index whose MRU-bit is 0 is replaced.


Regular matrix multiplication
99.2
99
98.8
98.6
98.4
98.2
98
97.8
97.6
97.4
Hit ratio
98.089
98.834
98.3209
99.15
98.3208
99.16
98.3206
99.157
2 way
4 way
8 way
16 way
Number of ways
64B Block
32B Block
Block size
32B Block
64B Block


Tile matrix multiplication
99.88
99.86
99.84
99.82
99.8
99.78
99.76
Hit ratio
99.8
99.839
99.841
99.876
99.8626
99.874
99.8637
99.85
2 way
4 way
8 way
16 way
Number of ways
64B Block
32B Block
Block size
32B Block
64B Block


Row Major matrix multiplication
99.2
99
98.8
98.6
98.4
98.2
98
97.8
Hit ratio
98.3091
99.15
98.3132
99.155
98.323
99.156
98.333
99.156
2 way
4 way
8 way
16 way
Number of ways
64B Block
32B Block
Block size
32B Block
64B Block

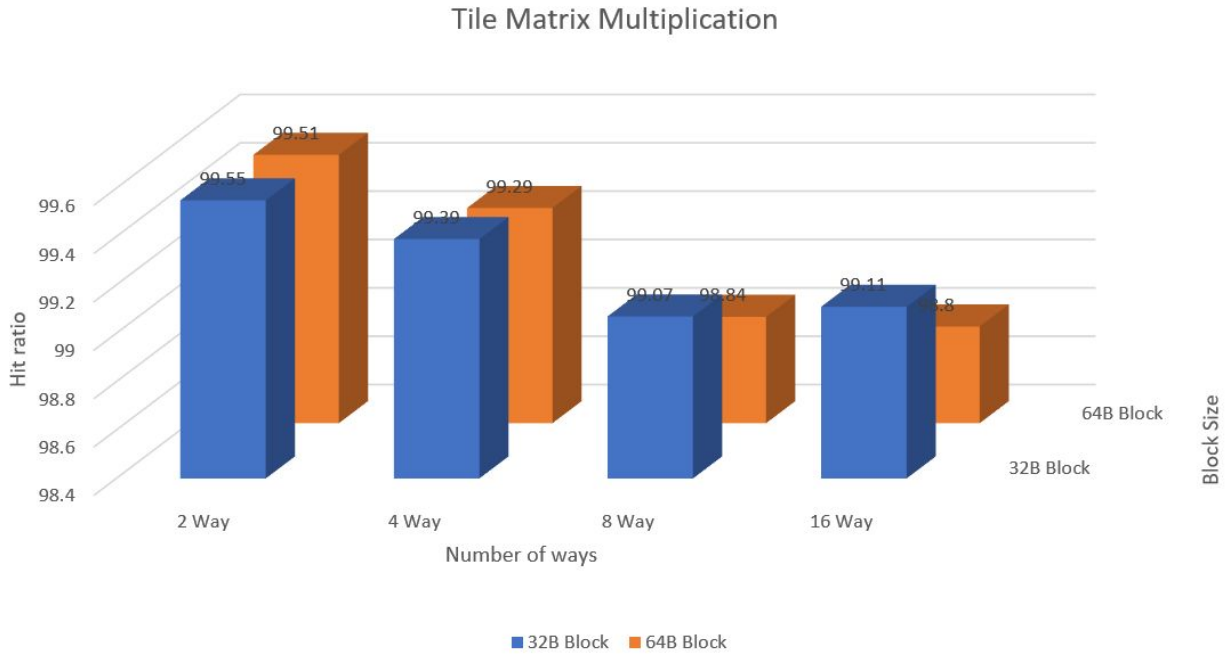
Analysis of Results on NRU cache replacement policy:

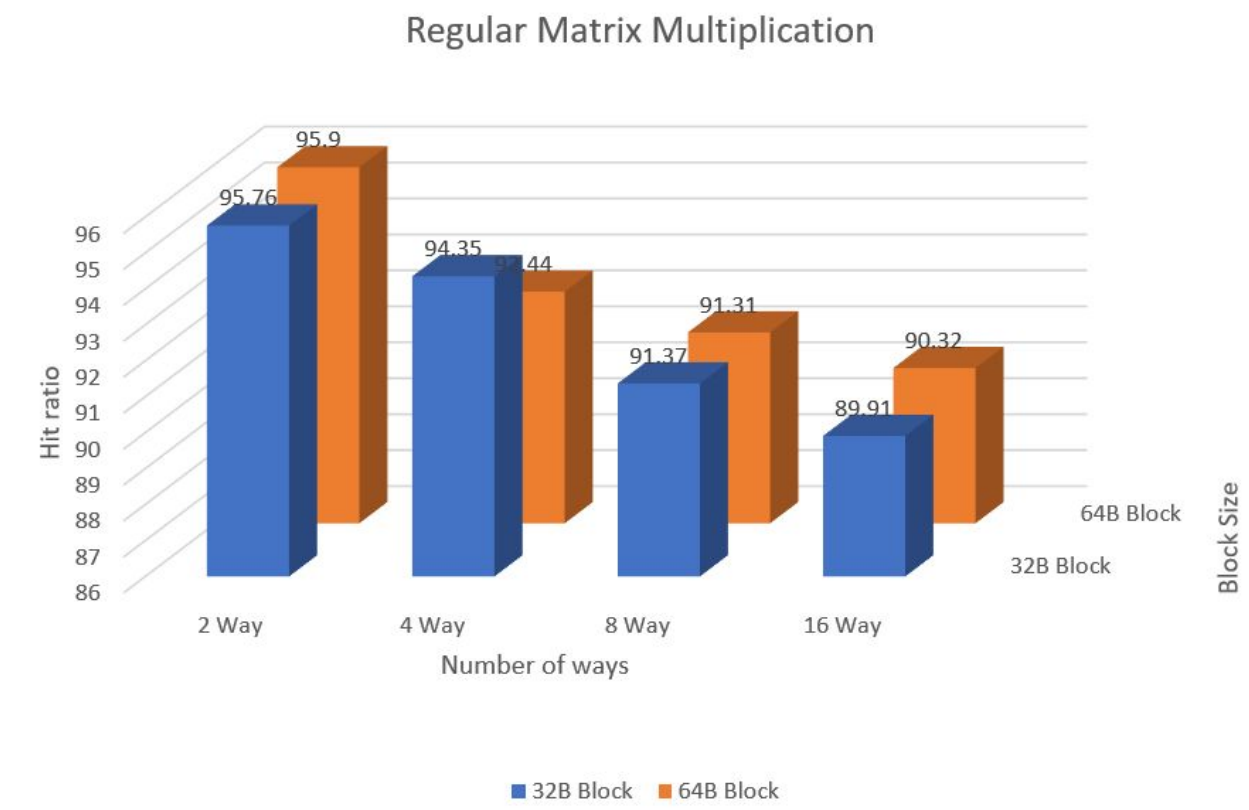
1. On an average the **best** performance for this cache replacement algorithm is observed for a **4** or **8 way** set associative cache having **64B block size**.
2. Tiled matrix multiplication gives the **best** performance on 4 way set associative cache with cache line size of 64B.
3. DGEMM on 2 way set associative cache with cache line size of 32B gives the **worst** performance.

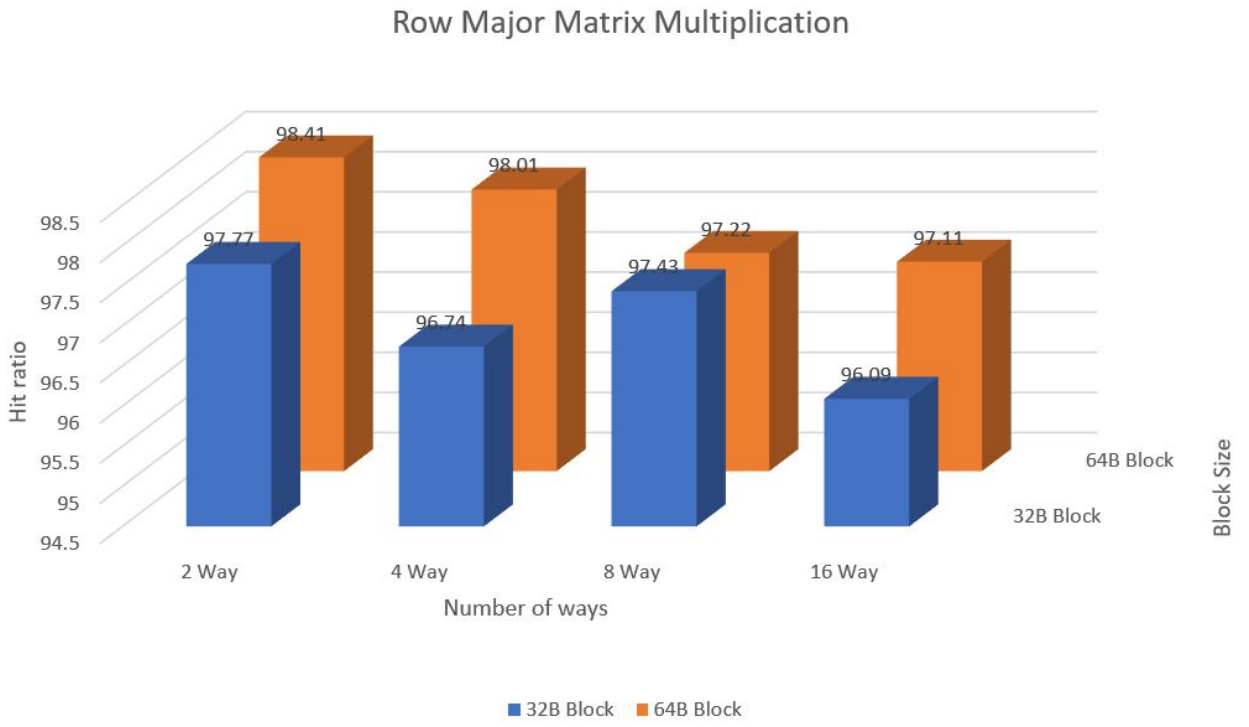
4. Shrinidhi Anil Varna

Contribution: LRU cache replacement algorithm and generating trace for regular matrix multiplication

Least recently used (LRU) cache replacement algorithm: In this algorithm, the instruction which is least recently used is replaced during a cache miss. During a cache hit, the recency of the instruction is updated. The algorithm keeps a valid bit, tag bit for each block in each set. The set number, tag value and the number of ways of the cache (cache configuration) is sent and the algorithm uses this policy to replace a victim block in case of a miss. The first execution of the cache for any configuration is a warm-up but later on cache is updated with instructions stored in it.




Regular Matrix Multiplication
95.9
95.76
94.35
91.31
91.37
90.32
89.91
96
95
94
93
92
91
90
89
88
87
86
Hit ratio
64B Block
32B Block
Block Size
2 Way
4 Way
8 Way
16 Way
Number of ways
32B Block
64B Block


Row Major Matrix Multiplication
98.41
98.01
97.77
97.22
97.43
97.11
96.74
96.09
98.5
98
97.5
97
96.5
96
95.5
95
94.5
Hit ratio
64B Block
32B Block
Block Size
2 Way
4 Way
8 Way
16 Way
Number of ways
32B Block
64B Block

Analysis:

1.On an average the best performance of this algorithm can be observed for 2 way set associative cache with 32B and 64B block sizes.
2.The tiled matrix multiplication has proven to be the best when this algorithm is used.
3.DGEMM on 16 way , 32B cache has given the worst performance upon using this algorithm.

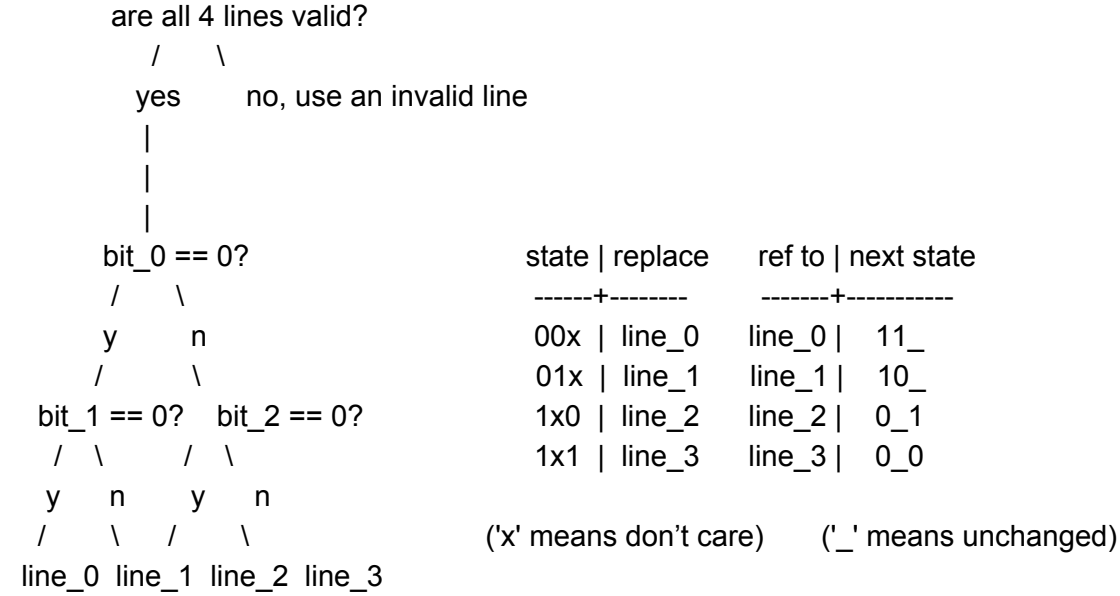
5. Chaitany Pandiya

Contribution: PLRU cache replacement algorithm.

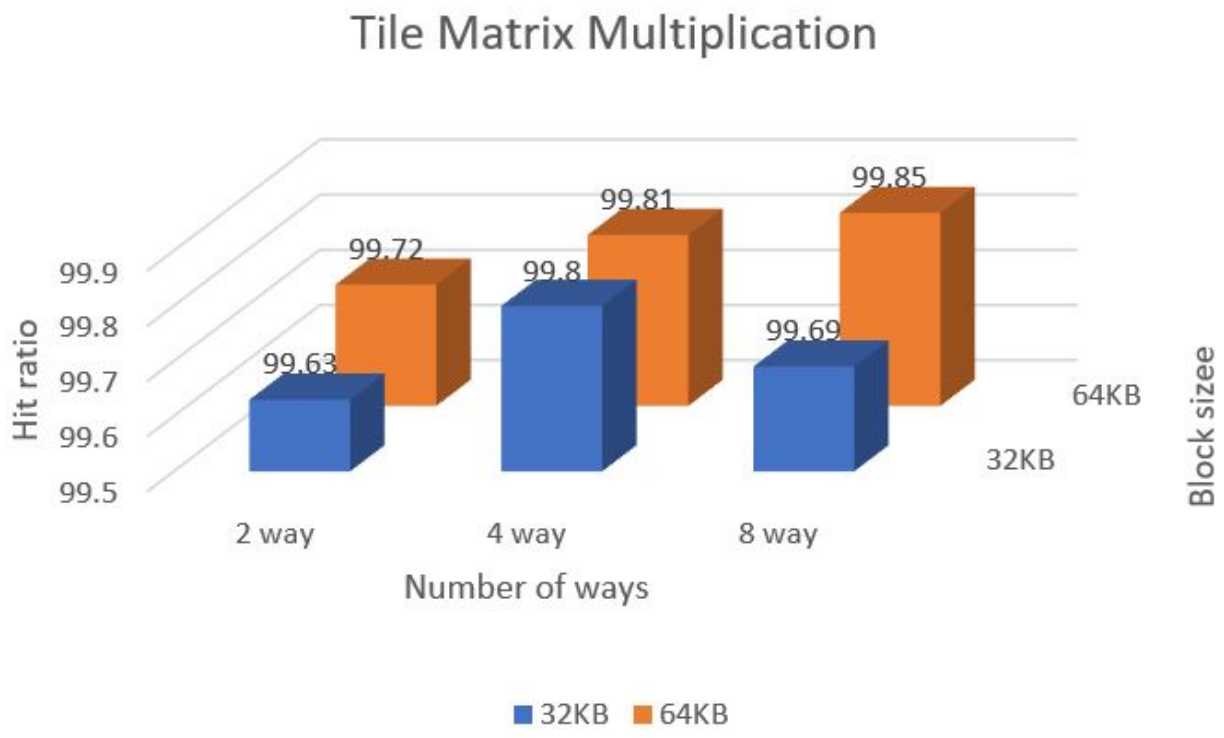
Pseudo-LRU or **PLRU** is a family of cache algorithms which improve on the performance of the Least Recently Used (LRU) algorithm by replacing values using approximate measures of age rather than maintaining the exact age of every value in the cache.

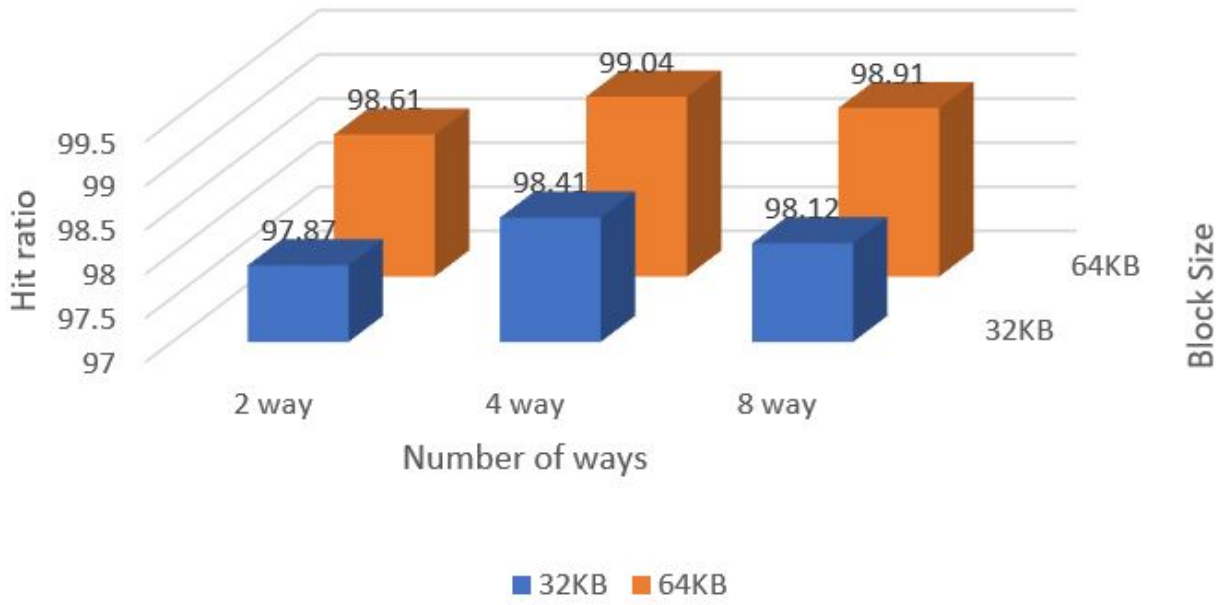
I have implemented by PLRU by storing the path(which tells which node is to be evicted) by doing simple bitwise operations after each access.

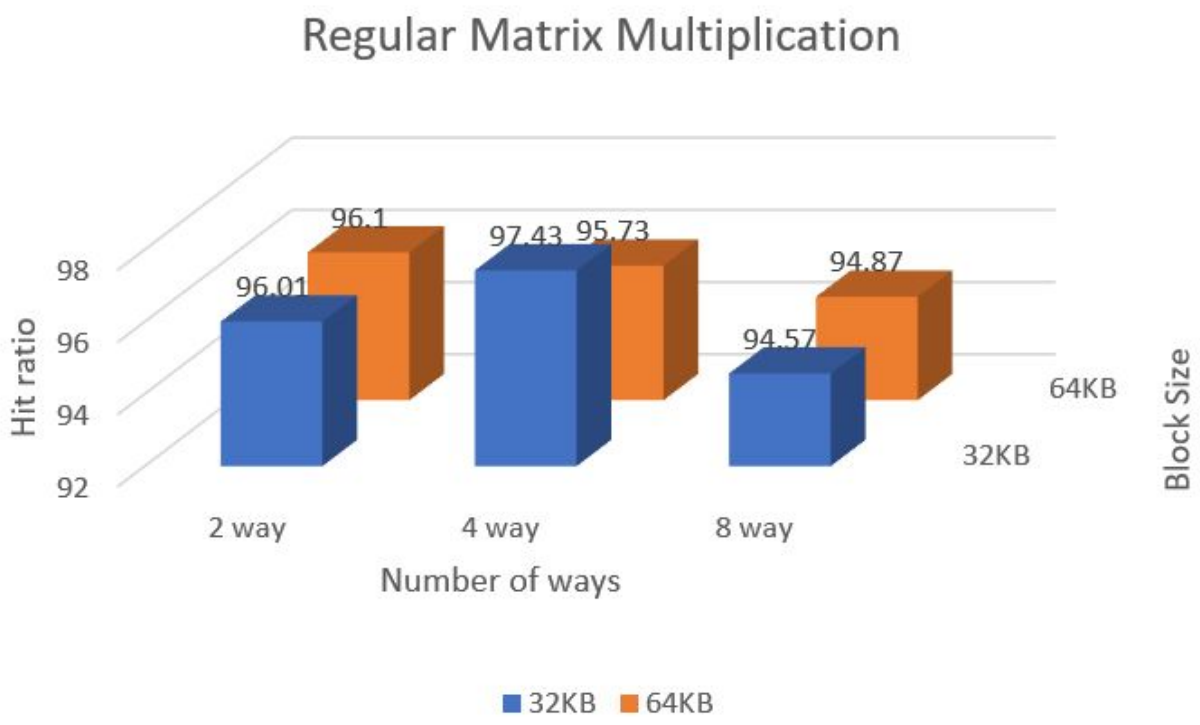
For example consider:
Four-way set associative - three bits
each bit represents one branch point in a binary decision tree; let 1 represent that the left side has been referenced more recently than the right side, and 0 vice-versa



2-way set associative - 1 = 1 bits
4-way set associative - 3 = 1+2 bits
8-way set associative - 7 = 1+2+4 bits
16-way set associative - 15 = 1+2+4+8 bits


Tile Matrix Multiplication
Hit ratio
99.9
99.8
99.7
99.6
99.5
99.63
99.72
99.8
99.81
99.69
99.85
2 way
4 way
8 way
Number of ways
64KB
32KB
Block sizee
32KB
64KB


Row Major Matrix Multiplication
Hit ratio
99.5
99
98.5
98
97.5
97
97.87
98.61
98.41
99.04
98.12
98.91
2 way
4 way
8 way
Number of ways
64KB
32KB
Block Size
32KB
64KB



Analysis of Results:

- 64 B Regular Matrix multiplication, Row major and Tiled in general perform better than that of 32 B with exception in number of ways=4. I think this is the ideal number of ways hence more sets increases the accuracy.

- Tiled Matrix multiplication shows the highest percentages and this should occur because tiled matrix multiplication is cache aware programming at its best.

- **Worst result** is given by Regular 32KB block size.

- **Best result** by 8 way Tiled multiplication.

6. Avakash Bhat

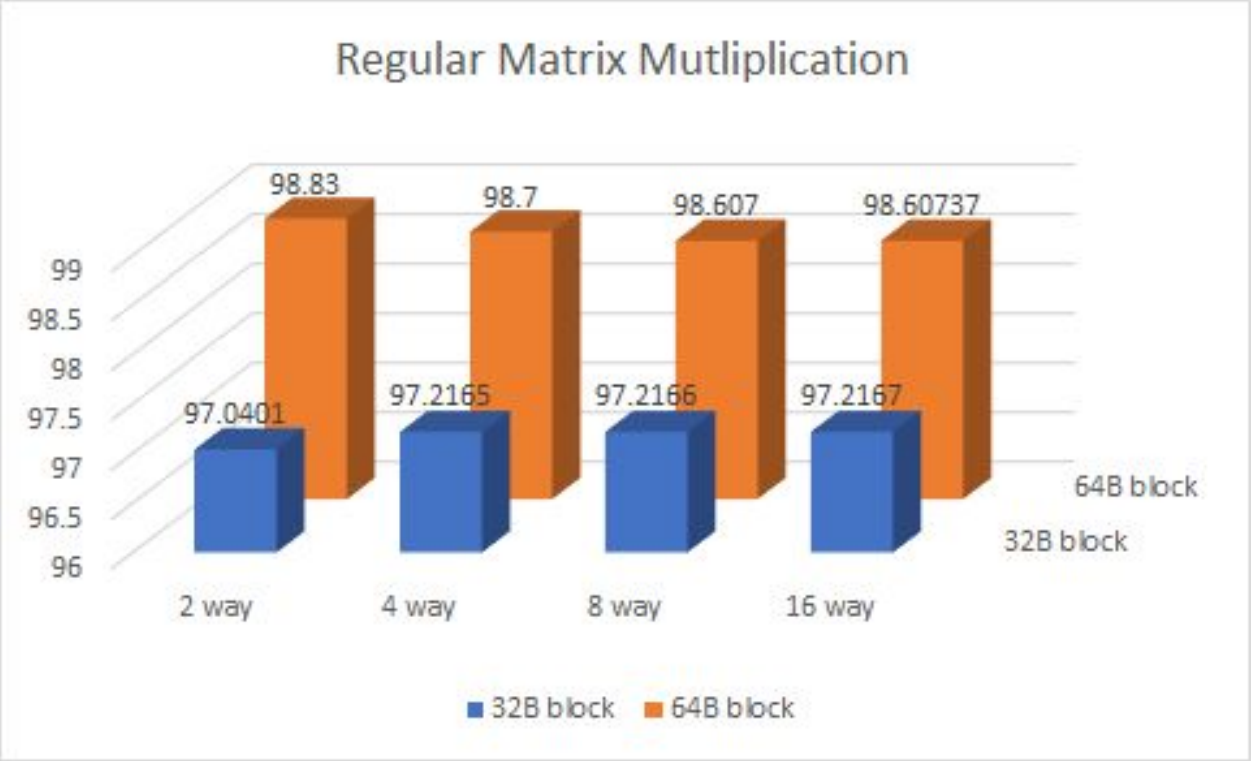
Contribution: FIFO cache replacement policy and generating the trace for tiled matrix multiplication.

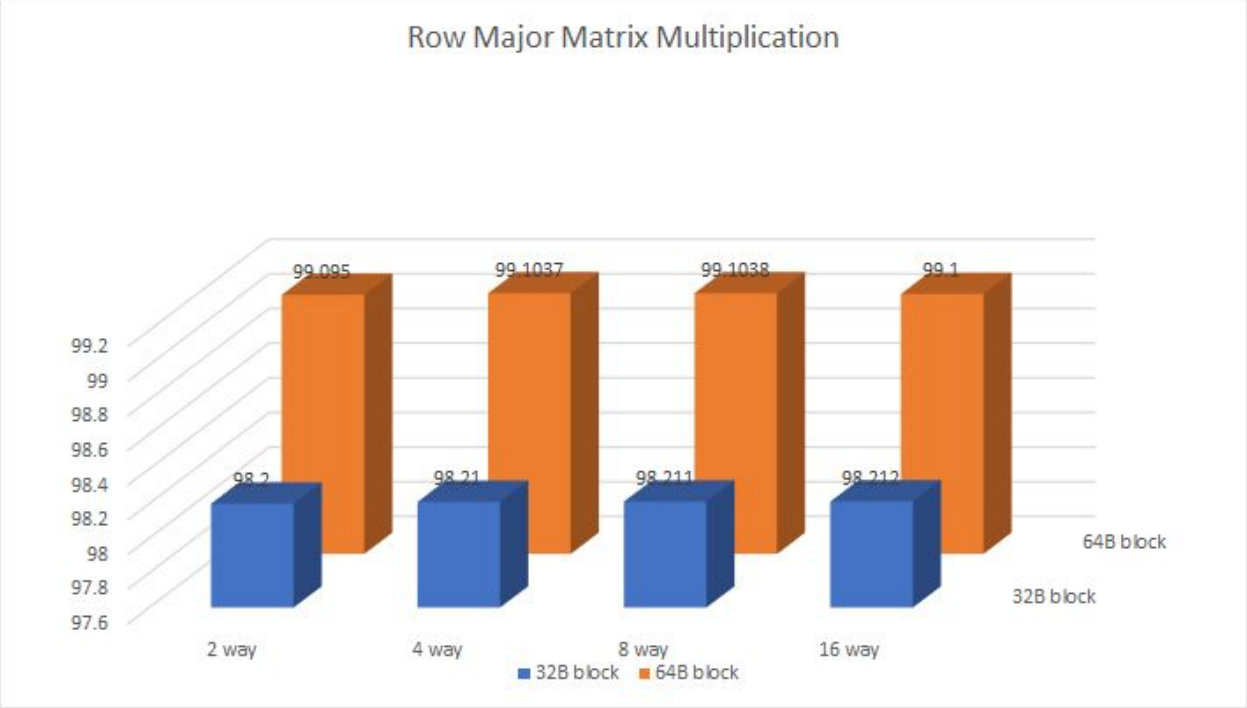
First in First out Cache Replacement Policy(FIFO): FIFO is one of the earliest page-replacement algorithm. FIFO requires very less bookkeeping and so is less of a burden to the operating system. The idea for this algorithm is that the pages that arrive are put in a queue and removed from the front. When a page is brought in it is added to the back of the queue and the back is incremented until the queue is full. Once the queue is full and another new page arrives, the page at the front of the queue is replaced. This page is also the oldest page. This algorithm experiences Bélády's anomaly.The graphs plot hitratio vs block size vs number of ways

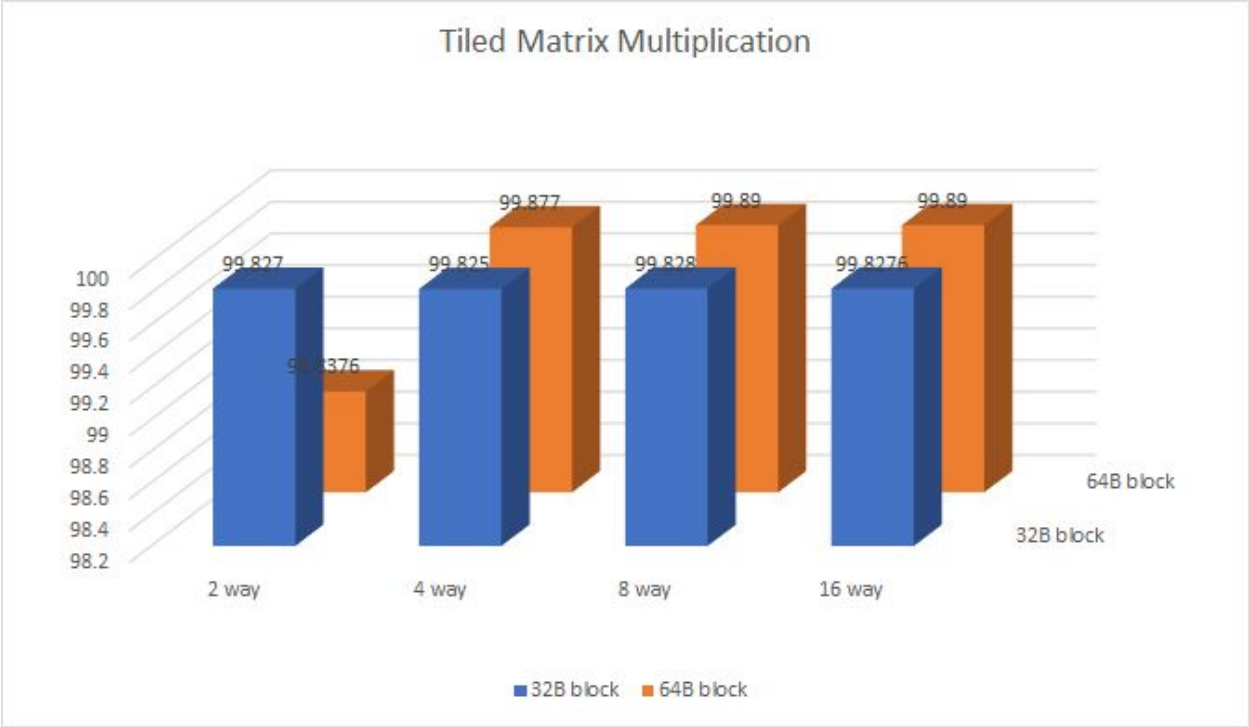
Analysis of Results:

- 64 B Regular Matrix multiplication, Row major and Tiled in general perform better than that of 32 B. This might be because of the larger capacity of each cache line
- In general the hit ratio continues to increase as the number of ways increase for both block sizes. The only anomaly is seen in Regular Matrix multiplication which might be due to Bélády's anomaly which FIFO is prone to.
- Tiled Matrix multiplication shows the highest percentages and this should occur because tiled matrix multiplication is cache aware programming at its best
- In general Tiled Matrix Multiplication performs better than Row major access matrix multiplication which performs better than Regular Matrix Multiplication

.


Regular Matrix Mutliplication
98.83
98.7
98.607
98.60737
97.0401
97.2165
97.2166
97.2167
99
98.5
98
97.5
97
96.5
96
64B block
32B block
2 way
4 way
8 way
16 way
32B block
64B block


Row Major Matrix Multiplication
99.095
99.1037
99.1038
99.1
98.2
98.21
98.211
98.212
99.2
99
98.8
98.6
98.4
98.2
98
97.8
97.6
64B block
32B block
2 way
4 way
8 way
16 way
32B block
64B block



7. Abhinav P Y:

Contribution: Static Re-Referemce Interval Prediction Cache Replacement Policy (SRRIP)

To **address the limitations of NRU**, we enhance the granularity of the re-reference prediction stored with each cache block. We propose cache replacement based on **Re-reference Interval Prediction (RRIP)**. RRIP uses M-bits per cache block to store one of 2M possible Rereference Prediction Values (RRPV). RRIP dynamically learns rereference information for each block in the cache access pattern. Like NRU, an RRPV of zero implies that a cache block is predicted to be re-referenced in the near-immediate future while RRPV of saturation (i.e., 2M–1) implies that a cache block is predicted to be re-referenced in the distant future. Quantitatively, RRIP predicts that blocks with small RRPVs are re-referenced sooner than blocks with large RRPVs. When M=1, RRIP is identical to the NRU replacement policy. When M>1, RRIP enables intermediate re-reference intervals that are greater than a near-immediate re-reference interval but less than a distant re-reference interval.

Since the re-reference predictions made by RRIP are **statically determined on cache hits and misses**, we refer to this replacement policy as **Static Re-reference Interval Prediction (SRRIP).**

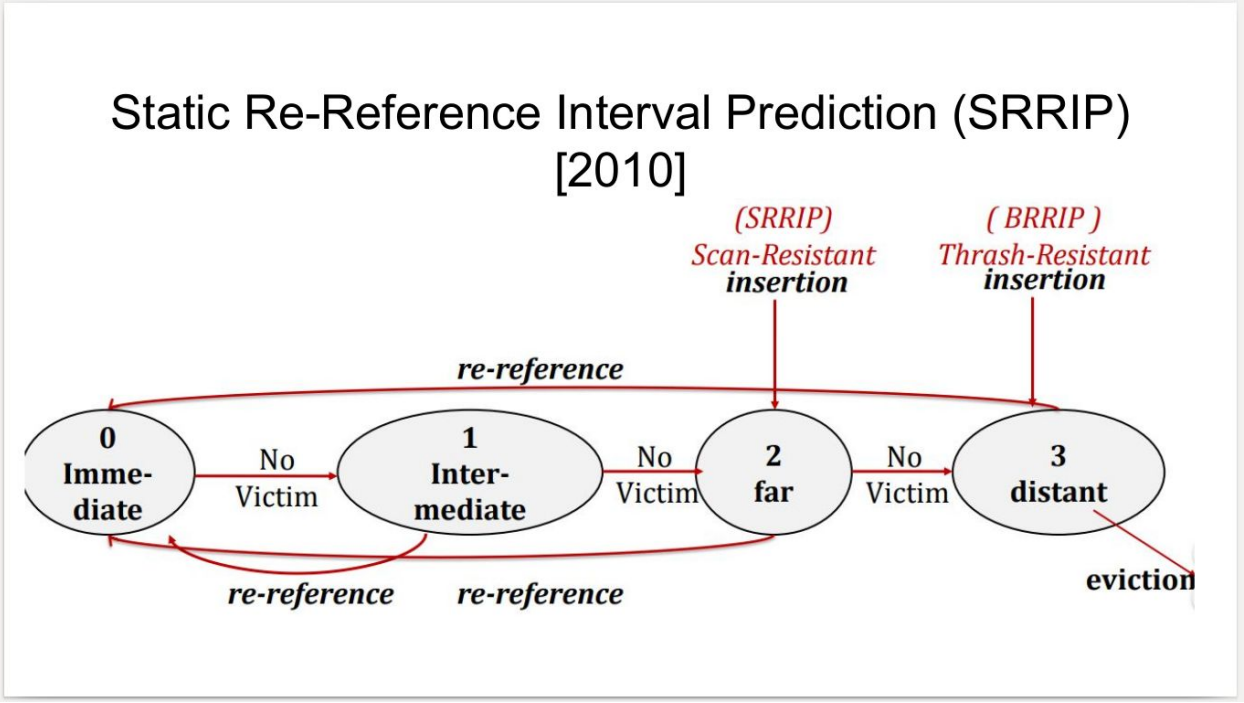
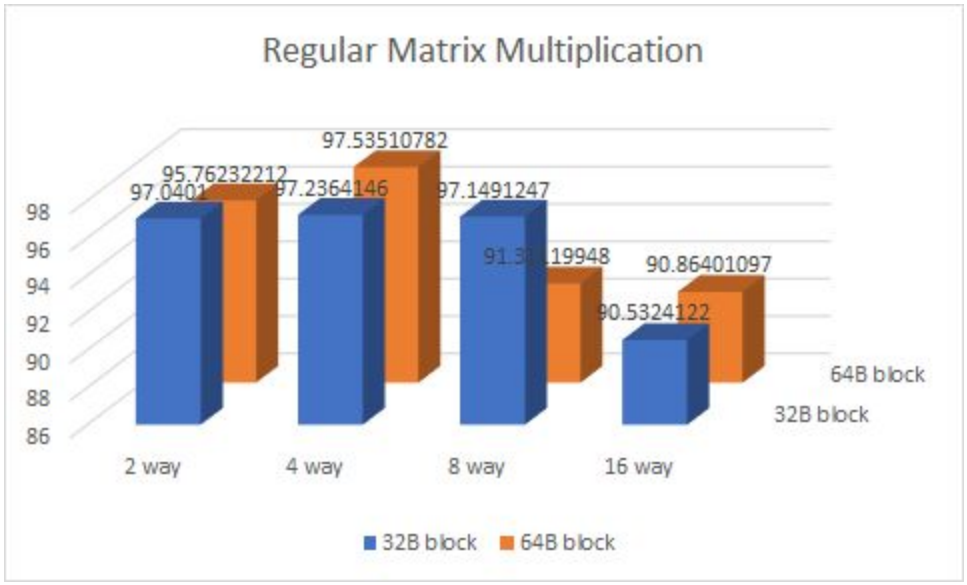


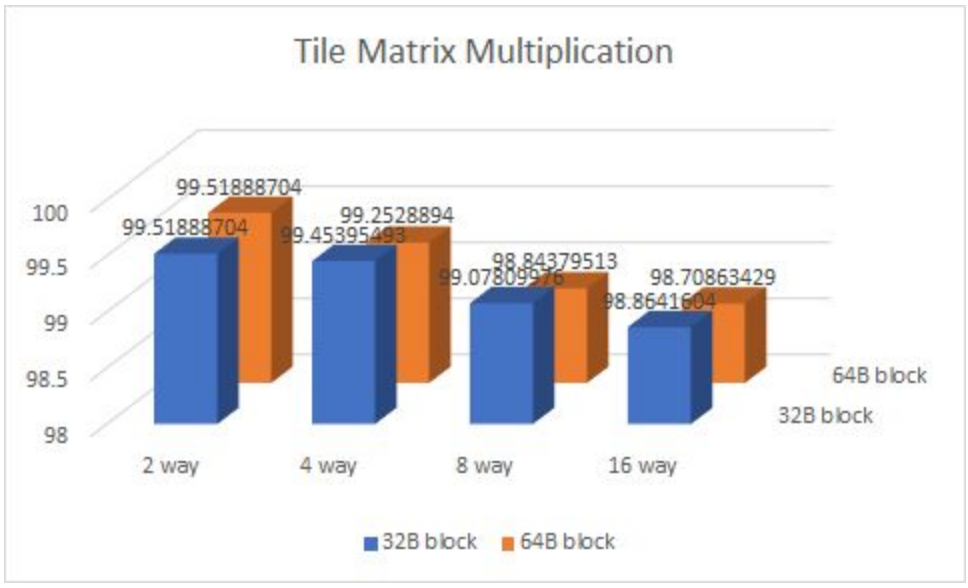
Image Courtesy: Aditya Kamath, NITK Surathkal

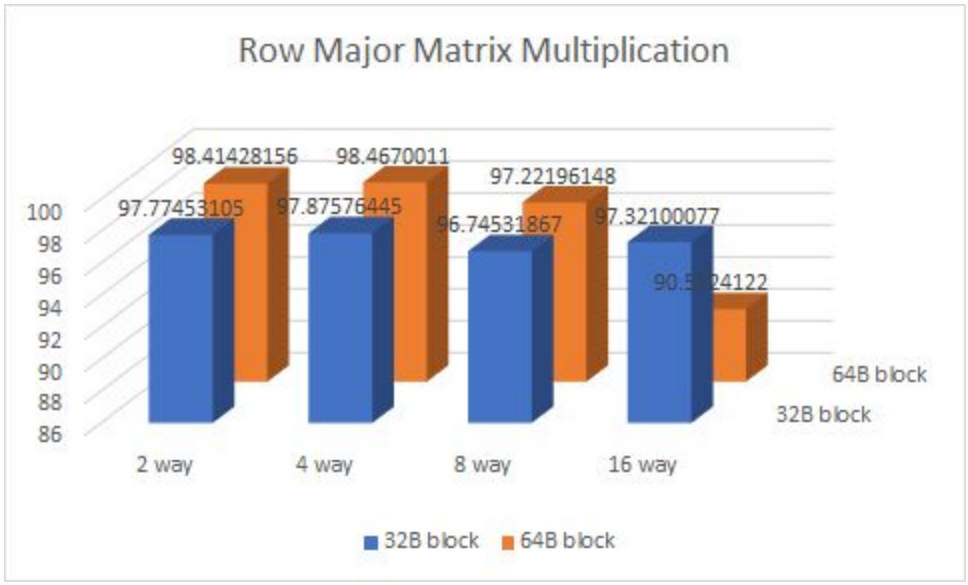
Analysis of Hit Results on various trace simulations and parameters for SRRIP:

1. We observe the **best performance** of Cache in **Tiled - Matrix multiplication's trace**, with over **99% hit rate** even when varying different ways and block size. We also observe that the hit rate drops with increasing number of ways, having worst performance with 16-way set associative cache.
2. We observe the **worst performance** of Cache in **Regular - Matrix multiplication's trace**, with results falling as low as 90%.
3. We observe a decently **good performance** of Cache in **Row Major Multiplication's trace**, with hit ratio's ranging from 97-98% on an average.
4. As a general trend across all traces, **increasing the number of ways** of the set-associative cache **decreses the hit ratio**.
5. Similiarly, **64B block sized** cache's showed **relatively better performances** than **32B block sized cache**.

The following graphs depict the results of our cache replacement policy:


Regular Matrix Multiplication
98
96
94
92
90
88
86
97.53510782
95.76232212
97.0401
97.2364146
97.1491247
91.3[illegible]19948
90.86401097
90.5324122
64B block
32B block
2 way
4 way
8 way
16 way
32B block
64B block


Tile Matrix Multiplication
100
99.5
99
98.5
98
99.51888704
99.51888704
99.2528894
99.45395493
98.84379513
98.70863429
98.8641604
64B block
32B block
2 way
4 way
8 way
16 way
32B block
64B block



Comparative study across various cache replacement algorithms which were implemented for 32B and 64B cache blocks: