CS 224 – Computer Organization

Section No: 4

Spring 2020

Lab No: 6

Mannan Abdul

21801066

Part 1)

1.

No.	Cache Size KB	N Way Cache	Word Size in Bits	Block Size	No. of Sets	Tag Size in Bits	Index Size	Word Block Offset Size in Bits	Byte Offset Size in Bits	Block Replacement Policy needed?
1	2	1	32	4	2^7	18	7	2	2	NO
2	2	2	32	4	2^6	19	6	2	2	YES
3	2	4	32	8	2^4	20	4	3	2	YES
4	2	FULL	32	8	1	24	0	3	2	YES
9	16	1	16	4	2^{11}	15	11	2	1	NO
10	16	2	16	4	2^{10}	16	10	2	1	YES
11	16	4	8	16	2^8	17	8	4	0	YES
12	16	FULL	8	16	1	25	0	4	0	YES

2.

a.

Instruction	**Iteration No.**				
	1	**2**	**3**	**4**	**5**
lw $t1, 0xA4($0)	Compulsory Miss	Hit	Hit	Hit	Hit
lw $t2, 0xAC($0)	Compulsory Miss	Hit	Hit	Hit	Hit
lw $t3, 0xA8($0)	Hit	Hit	Hit	Hit	Hit

b.

First, we can see that the cache is directly mapped because we have N = 1 which means that each every cache set will contain only a single memory data block at a time. The block size is 2 which means that there are always 2 data items in a set to allow for spatial locality. We have a byte offset of 2 because mips has byte addressable memory, we will need 1 bit to specify the block offset and 2 bits to specify the set no. This leaves us with 32 - 2 - 1 - 2 = 27 bits for the tag.

Each Set will need only 1 tag for both data items, that makes 27 bits for the tag, 1 valid bit and 64 data bits, 32 for each data. No of bits for 1 set in the cache is therefore 92 bits. Because we have 4 sets, that makes the **cache memory size 368 bits**.

c.

We will need 2 comparators for the comparison of the sets, 2 AND gates that check both the output of the comparators and the valid bit, 1 OR gate will be needed to check if data is found in a block and lastly we will need a 32 bit 2 – 1 Multiplexer to select the word output.

3.

a.

Instruction	**Iteration No.**				
	1	**2**	**3**	**4**	**5**
lw $t1, 0xA4($0)	Compulsory Miss	Capacity Miss	Capacity Miss	Capacity Miss	Capacity Miss
lw $t2, 0xAC($0)	Compulsory Miss	Capacity Miss	Capacity Miss	Capacity Miss	Capacity Miss
lw $t3, 0xA8($0)	Capacity Miss	Capacity Miss	Capacity Miss	Capacity Miss	Capacity Miss

b.

The cache capacity is 2 words and the block size is 1 word with only 1 set in the cache, this means that that N = 2 for this design of the cache and it is 2 way associative. This means that each set has 2 blocks. The byte offset for mips is always 2 bits, the block offset will be 0 because each block has only a single word. Since there is only 1 set, we don't need a bit for the index size in the tag either. That makes the tag have 32 – 2 = 30 bits. Each block will have 1 valid bit, 30 tag bits and 32 data bits which is equal to 63 bits for each block, since we have 2 blocks that totals 126 bits for 1 set. We need only 1 use bit because we have 2 blocks so adding that for the set, we have 127 bits. We have only 1 set, therefore the **cache size is 127 bits**

c.

Again, We will need 2 comparators for the comparison of the sets, 2 AND gates that check both the output of the comparators and the valid bit, 1 OR gate will be needed to check if data is found in a block and lastly we will need a 32 bit 2 – 1 Multiplexer to select the word output.

4. Assembly Code

```
        .text


main:
        li $v0, 4
        la $a0, Menu
        syscall

        li $v0, 4
        la $a0, prompt1
        syscall

        li $v0, 4
        la $a0, prompt2
        syscall

        li $v0, 4
        la $a0, prompt4
        syscall

        li $v0, 4
        la $a0, prompt5
        syscall

        li $v0, 4
        la $a0, prompt6
        syscall
```

```
        li $v0, 4
        la $a0, prompt7
        syscall

        li $v0, 4
        la $a0, prompt9
        syscall

        li $v0, 4
        la $a0, prompt8
        syscall

        li $v0, 5
        syscall

        beq $v0, 1, GetArraySize
        beq $v0, 2, InitializeArray
        beq $v0, 3, ShowArrayElement
        beq $v0, 4, RowMajorAdd
        beq $v0, 5, ColumnMajorAdd
        beq $v0, 6, ShowROrC
        beq $v0, 7, Exit

GetArraySize:
        li $v0, 4
        la $a0, prompt10
```

```
        syscall

        li $v0, 5
        syscall

        move $s0, $v0 #save N in $s0

        j Done
InitializeArray:
        addi $t0, $zero, 1
        addi $t3, $zero, 0
        mul $s1, $s0, $s0 #save no of elements in $s1
        mul $t1, $s1, 4
        bnez $s2, oldarray

        li $v0, 9
        move $a0, $t1
        syscall

        move $s2, $v0 #the starting address of the array
oldarray:
        li $t2, 0
        add $t2, $s2, $zero
startInitialize:
        beq $t3, $s1, Done

        li $v0, 4
```

```
        la $a0, prompt3
        syscall

        li $v0, 1
        move $a0, $t0
        syscall

        li $v0, 4
        la $a0, colon
        syscall

        li $v0, 5
        syscall

        sw $v0, ($t2)
        addi $t0, $t0, 1
        addi $t2, $t2, 4
        addi $t3, $t3, 1

        j startInitialize


ShowArrayElement:
        li $v0, 4
        la $a0, prompt11
        syscall
```

```
li $v0, 5
syscall

move $t0, $v0

li $v0, 4
la $a0, prompt12
syscall

li $v0, 5
syscall

move $t1, $v0

addi $t0, $t0, -1
addi $t1, $t1, -1
mul $t2, $s0, 4
mul $t1, $t1, $t2
mul $t0, $t0, 4
add $t3, $t0, $t1
add $t3, $s2, $t3

li $v0, 4
la $a0, prompt13
syscall

li $v0, 1
```

```
        lw $a0, ($t3)
        syscall

        li $v0, 4
        la $a0, newline
        syscall

        j Done

RowMajorAdd:
        li $t0, 1 #i = 1
        li $t1, 1 #j = 1
        li $t5, 0 #to compare with Nj
        li $t4, 0 #to comare with Ni
        li $t7, 0 #sum
computeRowMajor:
        addi $t0, $t0, -1
        addi $t1, $t1, -1
        mul $t2, $s0, 4
        mul $t1, $t1, $t2
        mul $t0, $t0, 4
        add $t3, $t0, $t1
        add $t3, $s2, $t3
        lw $t6, ($t3)
        add $t7, $t7, $t6

        addi $t5, $t5, 1
```

```
        addi $t1, $t5, 1
        addi $t0, $t4, 1
        bne $s0, $t5, computeRowMajor
        addi $t4, $t4, 1
        li $t5, 0
        addi $t1, $t5, 1
        addi $t0, $t4, 1
        bne $s0, $t4, computeRowMajor

        li $v0, 4
        la $a0, prompt14
        syscall

        li $v0, 1
        move $a0, $t7
        syscall

        li $v0, 4
        la $a0, newline
        syscall

        j Done

ColumnMajorAdd:
        li $t0, 0 #sum
        move $t1, $s2 #starting address of array
```

```
        li $t3, 0 #compare with no of elements

computeColumnMajor:
        lw $t2, ($t1)
        add $t0, $t0, $t2
        addi $t1, $t1, 4
        addi $t3, $t3, 1
        bne $s1, $t3, computeColumnMajor

        li $v0, 4
        la $a0, prompt15
        syscall

        li $v0, 1
        move $a0, $t0
        syscall

        li $v0, 4
        la $a0, newline
        syscall

        j Done

ShowROrC:
        li $v0, 4
        la $a0, prompt16
        syscall
```

```
        li $v0, 5
        syscall

        move $s4, $v0 #row or column

        li $v0, 4
        la $a0, prompt17
        syscall

        li $v0, 5
        syscall

        move $s3, $v0 #row or column number
        bgt $s3, $s0, error
        bnez $s4, displayRow
        j displayColumn
displayRow:
        move $t0, $s3#row number, i = $t0
        li $t1, 1 #j = 1
        li $t4, 0#to compare with N

computeDisplayRow:
        addi $t0, $t0, -1
        addi $t1, $t1, -1
        mul $t2, $s0, 4
        mul $t1, $t1, $t2
```

```
        mul $t0, $t0, 4
        add $t3, $t0, $t1
        add $t3, $s2, $t3
        lw $t6, ($t3)

        li $v0, 1
        move $a0, $t6
        syscall

        li $v0, 4
        la $a0, space
        syscall

        addi $t4, $t4, 1
        addi $t1, $t4, 1
        move $t0, $s3
        bne $t4, $s0, computeDisplayRow

        li $v0, 4
        la $a0, newline
        syscall

        j Done


displayColumn:
        move $t0, $s3#j = $t0
```

```
        li $t7, 0#to compare with N
        addi $t0, $t0, -1
        mul $t1, $s0, 4
        mul $t0, $t0, $t1
        add $t2, $s2, $t0
computeDisplayColumn:
        lw $t3, ($t2)

        li $v0, 1
        move $a0, $t3
        syscall

        li $v0, 4
        la $a0, newline
        syscall

        addi $t7, $t7, 1
        addi $t2, $t2, 4

        bne $t7, $s0, computeDisplayColumn

        j Done
error:
        li $v0, 4
        la $a0, prompt18
        syscall
```

```
        li $v0, 4
        la $a0, newline
        syscall

        j Done
Done:
        j main
Exit:
        li $v0, 10
        syscall

.data
newline: .asciiz "\n"
space: .asciiz "  "
colon: .asciiz ": "
Menu: .asciiz "               Menu            \n"
prompt1: .asciiz "1: Enter the size of the matrix in terms of its dimensions (N)\n"
prompt2: .asciiz "2: Enter the elements of the array colunmwise\n"
prompt3: .asciiz "\nEnter element "
prompt4: .asciiz "3: See element at position in Matrix[i,j] \n"
prompt5: .asciiz "4: Obtain row by row summation of matrix elements\n"
prompt6: .asciiz "5: Obtain column by column summation of matrix elements\n"
prompt7: .asciiz "6: Display a certain row or column, range should be within N\n"
prompt8: .asciiz "Choose your option by entering its number: "
prompt9: .asciiz "7: Exit\n"
prompt10: .asciiz "N = "
prompt11: .asciiz "i = "
```

prompt12: .asciiz "j = "

prompt13: .asciiz "\nM[i,j] = "

prompt14: .asciiz "\nRow major addition will give us: "

prompt15: .asciiz "\nColumn major addition will give us: "

prompt16: .asciiz "\nEnter 0 for column, 1 for row: "

prompt17: .asciiz "\nEnter row/column number: "

prompt18: .asciiz "\nrow/column number is out of bounds! "

Part 2)

a.

Row Major Addition

Matrix Size 75 x75

	Block Size (Number of Words)				
Cache Size (bytes)	16	32	64	128	256
1024	Miss Rate: 93% Miss Count: 5633	Miss Rate: 93% Miss Count: 5629	Miss Rate: 93% Miss Count: 5627	Miss Rate: 54% Miss Count: 3301	Miss Rate: 54% Miss Count: 3301
2048	Miss Rate: 90% Miss Count: 5471	Miss Rate: 93% Miss Count: 5629	Miss Rate: 93% Miss Count: 5627	Miss Rate: 54% Miss Count: 3301	Miss Rate: 27% Miss Count: 1651
4096	Miss Rate: 74% Miss Count: 4520	Miss Rate: 89% Miss Count: 5391	Miss Rate: 93% Miss Count: 5627	Miss Rate: 54% Miss Count: 3301	Miss Rate: 27% Miss Count: 1651
8192	Miss Rate: 7% Miss Count: 430	Miss Rate: 38% Miss Count: 2298	Miss Rate: 90% Miss Count: 5453	Miss Rate: 54% Miss Count: 3301	Miss Rate: 27% Miss Count: 1651
16384	Miss Rate: 7% Miss Count: 398	Miss Rate: 8% Miss Count: 495	Miss Rate: 42% Miss Count: 2578	Miss Rate: 30% Miss Count: 1821	Miss Rate: 15% Miss Count: 911

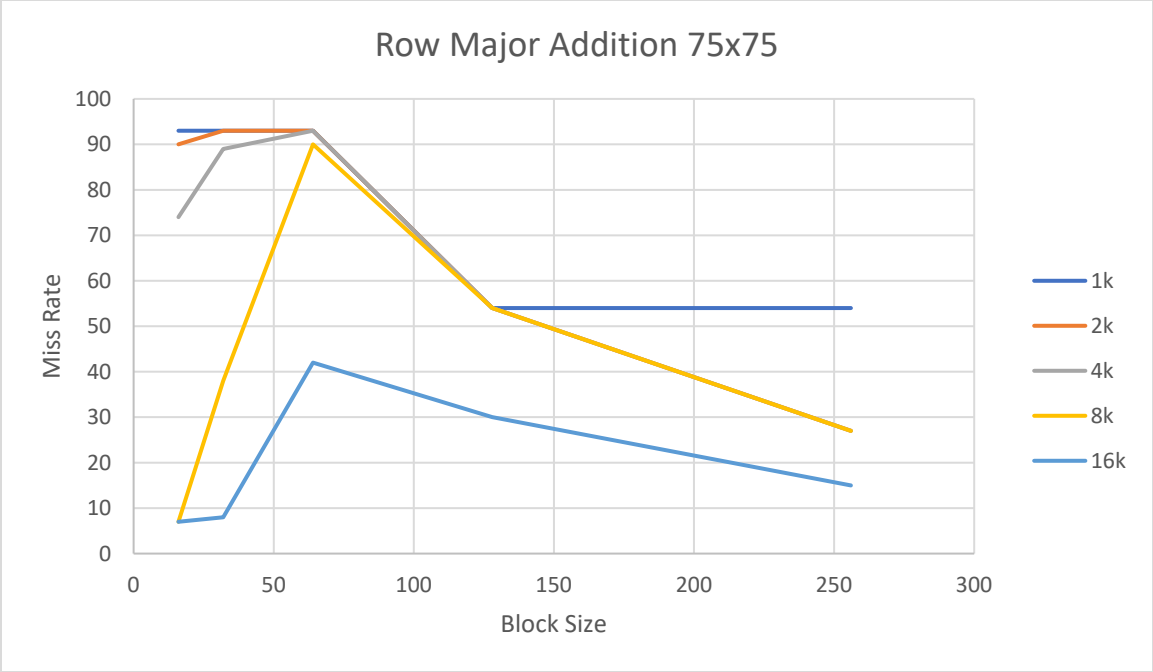

Figure 1

Matrix size 150x150

	Block Size (Number of Words)				
Cache Size (bytes)	16	32	64	128	256
1024	Miss Rate: 98% Miss Count: 22508	Miss Rate: 98% Miss Count: 22504	Miss Rate: 98% Miss Count: 22502	Miss Rate: 98% Miss Count: 22501	Miss Rate: 58% Miss Count: 13201
2048	Miss Rate: 98% Miss Count: 22508	Miss Rate: 98% Miss Count: 22504	Miss Rate: 98% Miss Count: 22502	Miss Rate: 98% Miss Count: 22501	Miss Rate: 58% Miss Count: 13201
4096	Miss Rate: 96% Miss Count: 21985	Miss Rate: 98% Miss Count: 22504	Miss Rate: 98% Miss Count: 22502	Miss Rate: 98% Miss Count: 22501	Miss Rate: 58% Miss Count: 13201
8192	Miss Rate: 81% Miss Count: 18685	Miss Rate: 90% Miss Count: 20590	Miss Rate: 98% Miss Count: 22502	Miss Rate: 98% Miss Count: 22501	Miss Rate: 58% Miss Count: 13201
16384	Miss Rate: 29%	Miss Rate: 59%	Miss Rate: 94%	Miss Rate: 98%	Miss Rate: 58%

	Miss Count: 6615	Miss Count: 13523	Miss Count: 21557	Miss Count: 22501	Miss Count: 13201

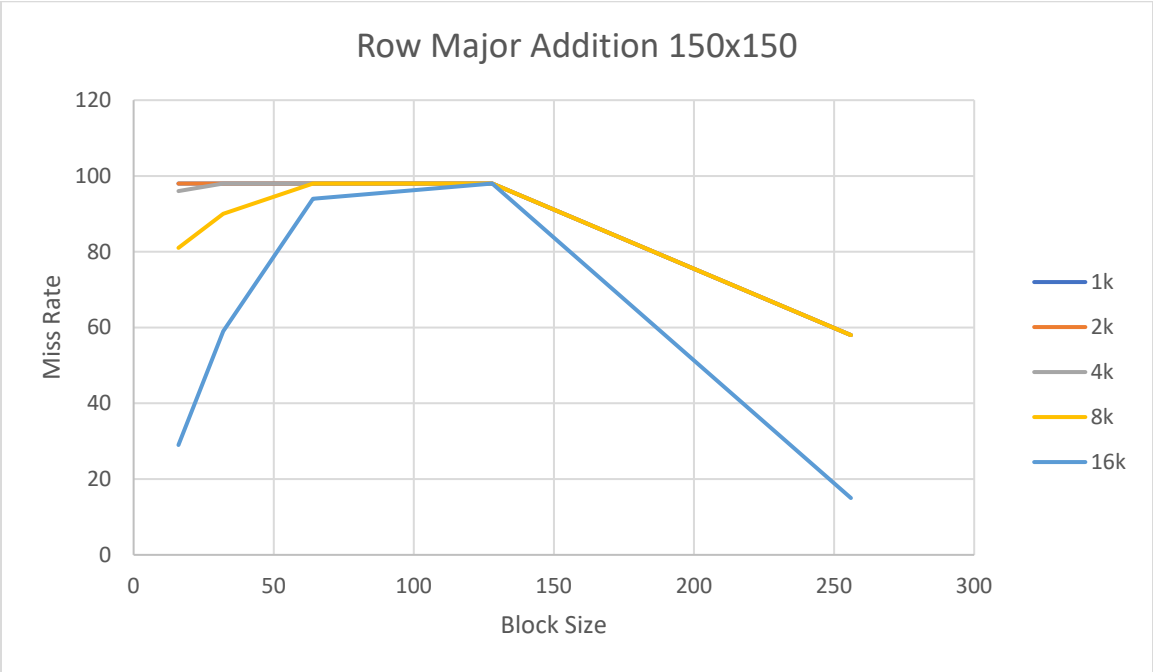

Figure 2

Column Major Addition

Matrix size 75x75

	Block Size (Number of Words)				
Cache Size (bytes)	16	32	64	128	256
1024	Miss Rate: 6% Miss Count: 361	Miss Rate: 3% Miss Count: 181	Miss Rate: 1% Miss Count: 91	Miss Rate: 1% Miss Count: 46	Miss Rate: 0% Miss Count: 23
2048	Miss Rate: 6% Miss Count: 361	Miss Rate: 3% Miss Count: 181	Miss Rate: 1% Miss Count: 91	Miss Rate: 1% Miss Count: 46	Miss Rate: 0% Miss Count: 23
4096	Miss Rate: 6% Miss Count: 361	Miss Rate: 3% Miss Count: 181	Miss Rate: 1% Miss Count: 91	Miss Rate: 1% Miss Count: 46	Miss Rate: 0% Miss Count: 23
8192	Miss Rate: 6% Miss Count: 361	Miss Rate: 3% Miss Count: 181	Miss Rate: 1% Miss Count: 91	Miss Rate: 1% Miss Count: 46	Miss Rate: 0% Miss Count: 23
16384	Miss Rate: 6%	Miss Rate: 3%	Miss Rate: 1%	Miss Rate: 1% Miss Count: 46	Miss Rate: 0% Miss Count: 23

	Miss Count: 361	Miss Count: 181	Miss Count: 91		

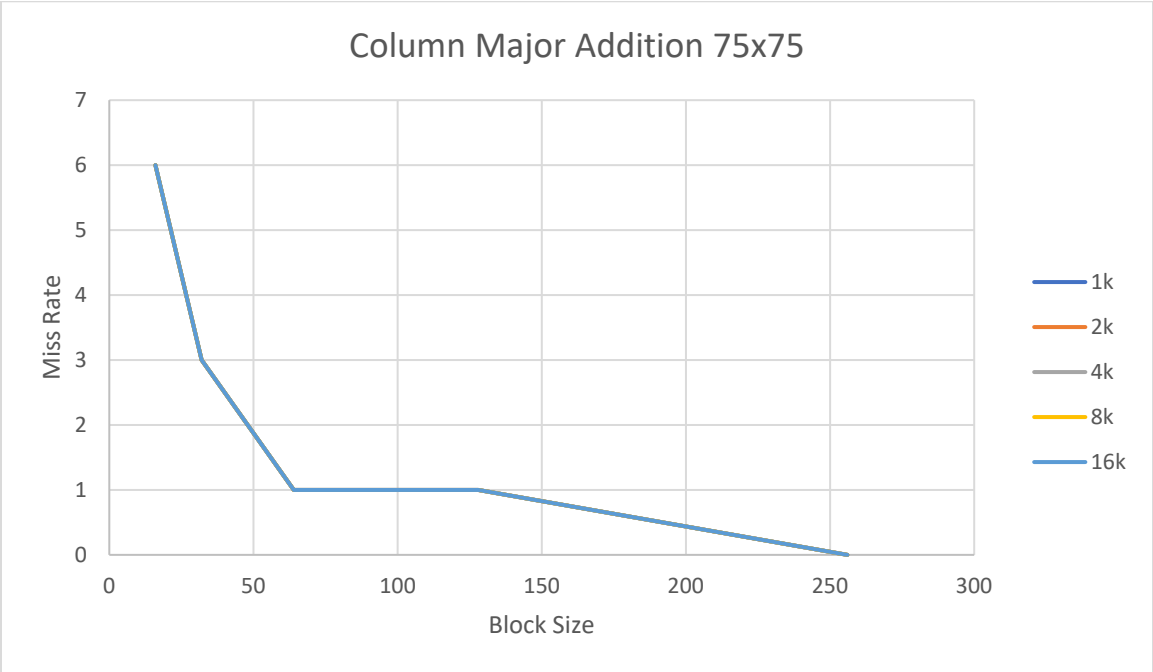

Figure 3

Matrix size: 150 x 150

	Block Size (Number of Words)				
Cache Size (bytes)	16	32	64	128	256
1024	Miss Rate: 6% Miss Count: 1416	Miss Rate: 3% Miss Count: 709	Miss Rate: 2% Miss Count: 355	Miss Rate: 1% Miss Count: 178	Miss Rate: 0% Miss Count: 89
2048	Miss Rate: 6% Miss Count: 1416	Miss Rate: 3% Miss Count: 709	Miss Rate: 2% Miss Count: 355	Miss Rate: 1% Miss Count: 178	Miss Rate: 0% Miss Count: 89
4096	Miss Rate: 6% Miss Count: 1416	Miss Rate: 3% Miss Count: 709	Miss Rate: 2% Miss Count: 355	Miss Rate: 1% Miss Count: 178	Miss Rate: 0% Miss Count: 89
8192	Miss Rate: 6% Miss Count: 1416	Miss Rate: 3% Miss Count: 709	Miss Rate: 2% Miss Count: 355	Miss Rate: 1% Miss Count: 178	Miss Rate: 0% Miss Count: 89
16384	Miss Rate: 6%	Miss Rate: 3%	Miss Rate: 2%	Miss Rate: 1%	Miss Rate: 0% Miss Count: 89

	Miss Count: 1416	Miss Count: 709	Miss Count: 355	Miss Count: 178	

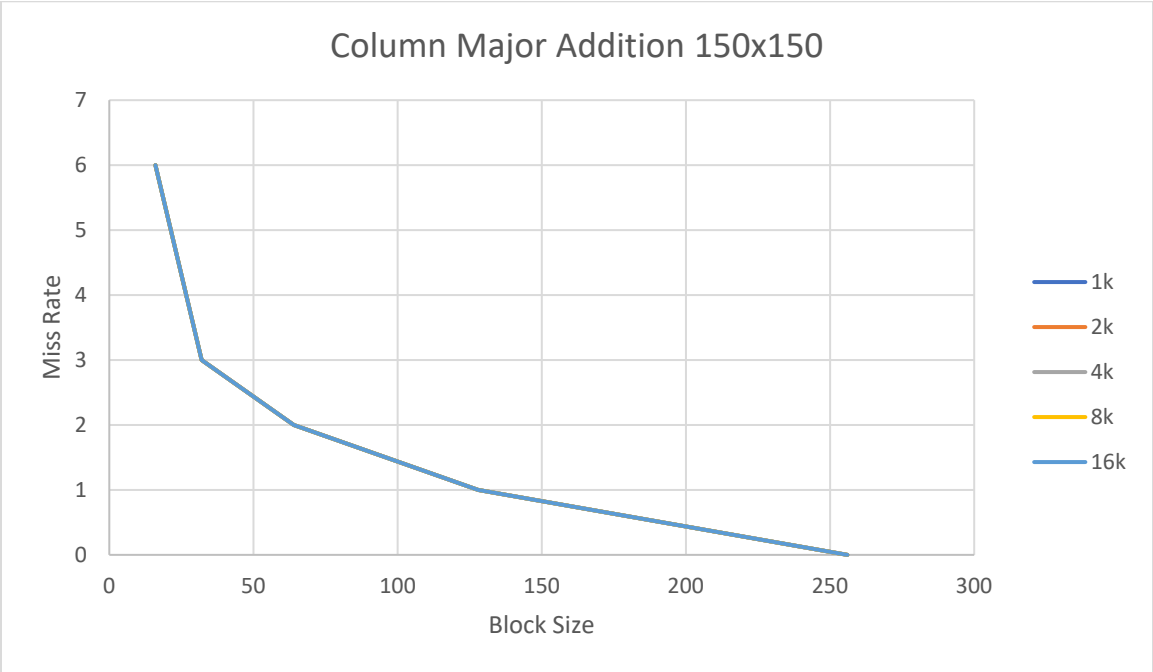

Figure 4

b.

75x75

	Cache Type		
Cache Size/ Block Size	**Direct Mapped**	**Fully Associative LRU**	**Fully Associative Random**
8192/16 (Good)	7%	7%	12%
2048/128 (Medium)	54%	54%	54%
1024/16 (Poor)	93%	93%	92%

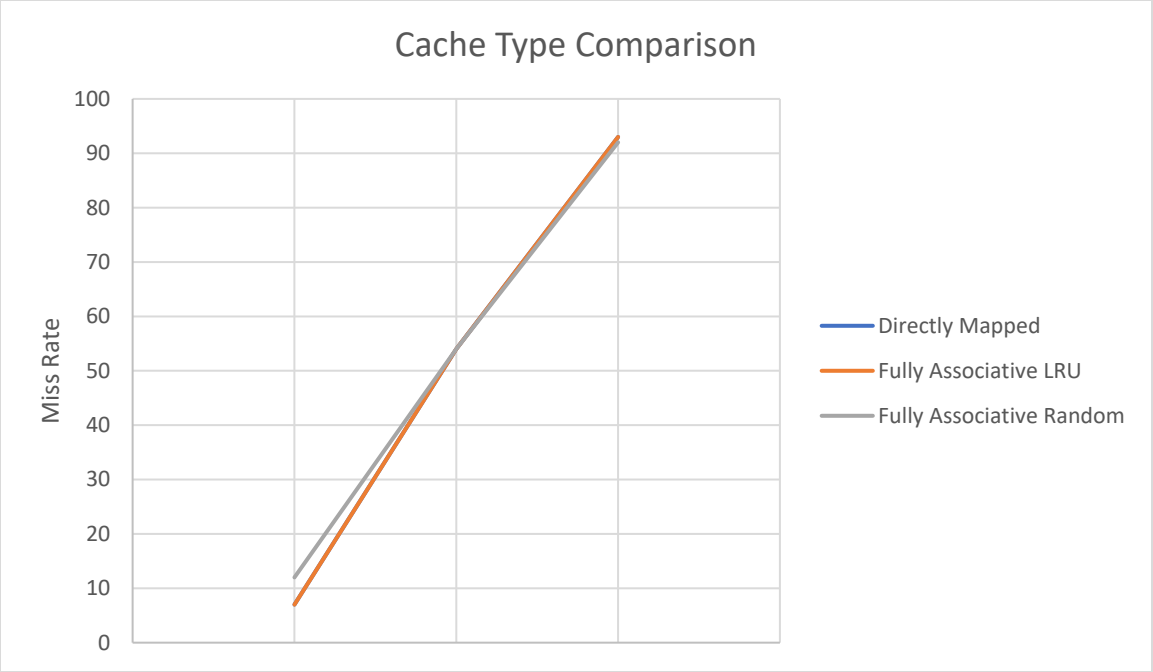

Figure 5

*1st data point is 8192/16, 2nd is 2048/128 and 3rd is 1024/16

150 x 150

	Cache Type		
Cache Size/ Block Size	**Direct Mapped**	**Fully Associative LRU**	**Fully Associative Random**
16384/16 (Good)	29%	7%	13%
4096/256 (Medium)	58%	58%	58%
1024/16 (Poor)	98%	98%	98%

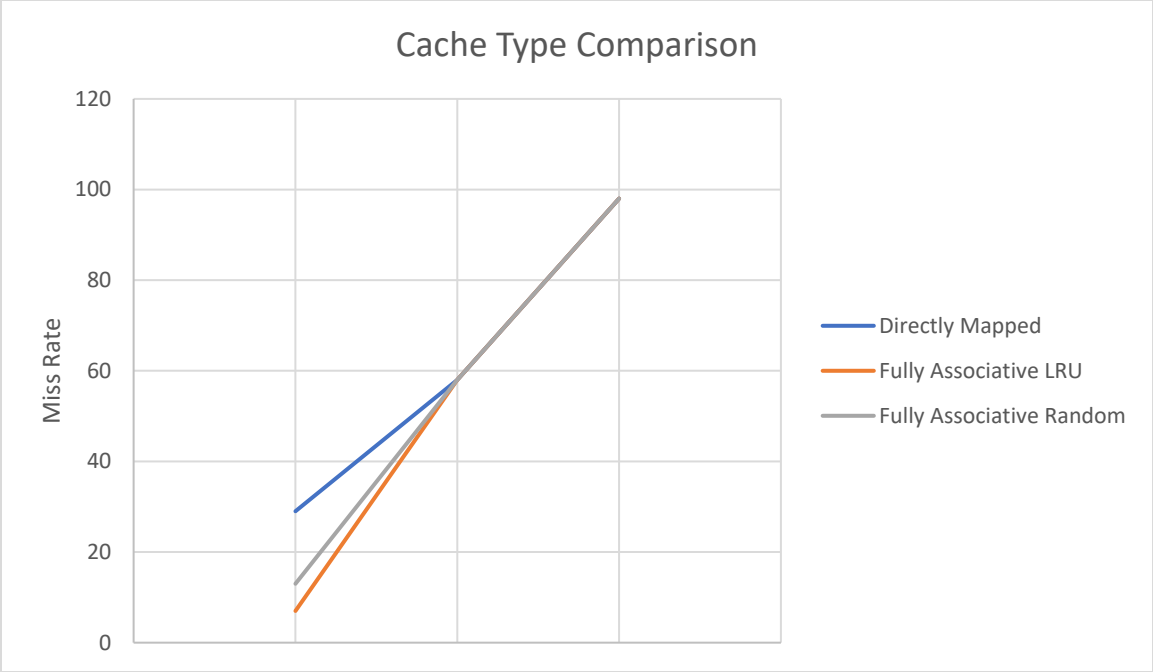

Figure 6

75 x 75)

As we can see from the 1st graph, the change to fully associative cache type doesn't really make a big difference but when it is set as random, it actually increases the miss rate for the best case for this matrix size. It also decreases the miss rate by 1% for the worst case. The middle case remains the same here.

150 x 150)

From the 2nd graph we can see a really big difference, for the best case, changing cache type to fully associative LRU brings down the miss rate from 29% to just 7% greatly making our best case even better. Changing placement policy to random though makes it worse bringing it to a 13% miss rate. The other 2 cases, medium and poor both remain the same otherwise.

c.

Row Major =>

Medium Configuration, 2048/128 (Cache Size/Block Size), Row Major Addition, 75 x75

N- Way Cache	2	4	8	16
Miss Rate	54%	54%	-	-
Hit Rate	46%	49%	-	-
Miss Count	3301	3301	-	-

*only 4 sets max are possible for this configuration.

Best Configuration, 8192/128 (Cache Size/Block Size), Row Major Addition, 75 x 75

N- Way Cache	2	4	8	16
Miss Rate	7%	7%	7%	7%
Hit Rate	93%	93%	93%	93%
Miss Count	430	430	430	430

Poor Configuration, 1024/16 (Cache Size/Block Size), Row Major Addition, 75 x 75

N- Way Cache	2	4	8	16
Miss Rate	93%	93%	93%	93%
Hit Rate	7%	7%	7%	7%
Miss Count	5633	5633	5633	5633

Medium Configuration, 4096/256 (Cache Size/Block Size), Row Major Addition, 150 x 150

N- Way Cache	2	4	8	16
Miss Rate	58%	58%	-	-
Hit Rate	42%	42%	-	-
Miss Count	13201	13201	-	-

*only 4 sets max are possible for this configuration.

Best Configuration, 16384/16 (Cache Size/Block Size), Row Major Addition, 150 x 150

N- Way Cache	2	4	8	16
Miss Rate	43%	7%	7%	7%
Hit Rate	57%	93%	93%	93%
Miss Count	9779	1546	1546	1546

Poor Configuration, 1024/16 (Cache Size/Block Size), Row Major Addition, 150 x 150

N- Way Cache	2	4	8	16
Miss Rate	98%	98%	98%	98%
Hit Rate	2%	2%	2%	2%
Miss Count	22508	22508	22508	22508