Name- Nishant
Roll No. 251110053

CS610 Assignment 1

Problem 1:

Cache size=128KB, Line size=128B and word size=4B
then LIne size(in words)=32 words , #lines in cache=2^10 lines
No of ways=8
No of sets=2^10/2^3=2^7 sets
tag=18bits,sets=7 bits and block=7 bits(last 2 are for words and the other five for block)
assume the address are block aligned: e.g address as 0x12340000 and 0xabcd0000
both A and B starting address are in 0th set
and both A and B compete with each other for cache space.
(Misses means A misses only).

Number of wraparounds=total number of elements/(number of elements accessed in each cache round)
set number of a element=(element number/number of elements in a line) %number of sets

Stride<=32:
We get a hit for every set in cache,as each line can hold 32 words
for stride =32 pattern of reads on A array is A[0],A[32],A[64],_ _ _ _ _ ,it means they get map to 0,1,2 ,_ _ _ sets
number of rounds around the cache is 2^15/2^12 =8 rounds
No of misses is 1 miss per set.
No of sets=2^7 and No of rounds =2^3
Total miss for 8 wraparounds=2^10=1024.
As there are 1000 iterations of these accesses.

and since each set maps 1 line of A and 1 line of B so after 4 wraparounds,cache is full and replacement of old blocks (lines) is started happening from round 5.
So misses in each iteration(of it=0 to 999)=1024
Total misses in 1000 iterations=1024000

Stride=64:

Access is of 0th set ,2nd set,4th set and so on,
So each access is a miss, and number of elements accessed is =2^15/2^6=2^9(element/stride), so 512 misses.
again number of rounds=number of elements/(# elements accessed in each cache round)=2^9/2^6=8
and again after 4 wrap rounds cache is full(8lines in set=4 lines occupied by A + 4 lines occupied by B)
So,after 8 rounds all blocks are replaced,leading to same number of misses for each next iteration
So,number of misses in 1000 iterations=512000

Stride=2K:
element 2^11 is in =2^11/2^5=2^6 set
next element in =0th set
so access pattern of these elements is:0,64,0,64, and so on
elements accessed=2^15/2^11=16
#misses=as each element access is a miss=16 miss
again wraparounds is =16/2=8
again after 4 round ,old blocks starting to get replaced , leading to miss in each iteration
Total miss in all iterations=16000 misses

Stride=8K:
elements accesses: 0th,2^13,2.2^13,3.2^13 and so on
so sets they map to is: 0,0,0 and so on((element/#words in block)%number of sets)
Number of elements accessed=2^15/2^13=4
miss for each element ,#misses in 1st iteration=4 miss
also wrap rounds=4/1=4 rounds
so after 4 rounds is full with each set has 4 elements of A and 4 of B
Now for each new iteration,there is no evictions,as they are in cache
So,Total miss in all iterations=4 miss

Stride	Misses of A in first iteration	Misses of A in all 1000 iterations
1	1024	1024000
16	1024	1024000
32	1024	1024000
64	512	512000
2K	16	16000
8K	4	4

Problem 2:

cache size=2^16 words
line size=16 words=BL
Cache size(in lines)=2^12 lines
size of array =2^20 elements, N=1024
and each access in row major order lead to =2^10/2^4=first 64 lines get filled

For explanation of my answers, i am taking order from most inner to most outer loop.
Also,here as row is big enough to not fit in cache line,column major order transveral lead to all misses.

For kij:

Direct mapped:

For A:

1. wrt j it is accessing the same first elemet A[0][0],so every iteration is hit, so j loop contribution is 1.
2. wrt i ,it is accessing in column major order,so i loop contribution is N.
3. wrt k,if we change k=0 to k=1 ,due to it is a direct mapped cache,due to column major order of i,the lines of cache mapped are (for A[0,0],A[1,0],A[2,0] and and so on): 0,64,128 and so on,so after 2^12/2^6=64 entries accessed in column major order the old lines are evicted ,leading to miss in row wise traversal also.

For B:

1. wrt j ,it is accessing in row major order,so BL blocks gets hit after each miss,thus j loop contribution is N/BL
2. wrt i, as cache is full upto 0th to 63th line ,and each ith iteration access the same blocks(lines),leading to all hits,thus i loop contribution=1
3. wrt k,as now we are accessing in columns ,the access pattern is A[1,0] in 64th line,A[2,0] in 128th line and so on,so each iteration leads to miss,then k loop contribution is N.

For C:

1. wrt j,it access in row major order so j loop contribution is N/BL(1 miss for BL number of accesses)
2. wrt i,it is in access across column, so i loop contribution is N.
3. wrt K,as matrix is big enough to not fit in cache ,so blocks get replaced and

contribution of k loop is N.

	A	B	C
i	N	1	N
j	1	N/BL	N/BL
k	N	N	N
	N^2=2^20	N^2/BL=2^16	N^3/BL=^26

Fully Associative:

For A:

1. wrt j, it is accessing the same first elemet A[0][0],so every iteration is hit, so contribution of j loop is 1.
2. wrt i ,it is accessing in column major order,so every access is miss,thus i loop contribution is N. Also line allocation pattern is 0,1,2,3, and so on(as it is fully associative cache,the 2nd block is in 2nd line ,and so on).A[0,0] in 1st line,A[1,0] in 2nd line, _ _ _ and A[1023,0] in 1024th line.
3. wrt k,it is accessing row wise,e.g. A[0,0],A[0,1],_ _ _,A[0,15],there is hit on these next 15 blocks(BL-1),so its contribution to miss rate is N/BL.(k loop contribution)

For B:
same as direct cache case for each B array entry.

For C:
same as the direct cache case for each C array entry.

	A	B	C
i	N	1	N
j	1	N/BL	N/BL
k	N/BL	N	N
	N^2/BL=2^16	N^2/BL=2^16	N^3/BL=2^26

For jki:

Direct Mapped:

For A:

1. wrt i, we are accessing elements in column major order leading to all misses,so i loop contribution is N. Also mapping of A[0,0],A[1,0],A[2,0] and so on is 0,64,128 and so on.So ,older blocks also gets replaced,as number of wrap rounds is 2^10/2^6=16 rounds(64 elements are accessed in 1 round).
2. wrt k, as above 16 wrap rounds has happened,so accessing in row manner now lead to all misses, so k loop contribution is also N.
3. wrt j,as matrix is big enough to not fit in cache ,so blocks get replaced and contribution of j loop is N.

For B:

1. wrt i, as we are only accessing B[0,0] element,hence i loop contribution is 1.
2. wrt k,we are accessing in column major fashion so k loop contribution is N.
3. wrt j,as it is direct mapped so lines that are mapped are 0,64,128 etc,and this lead to 16 wraparounds,so older blocks getting evicted and no hits for row wise access now,and thus j loop contribution is N.

For C:

1. wrt i,we are accessing elements in column major,and so i loop contribution is N.
2. wrt k,as older blocks are evicted due to wraparounds,and so k loop access to them lead to all misses ,and thus k loop contribution is N.
3. wrt j,as due to wraprounds older blocks are replaced,so contribution of j loop is N.

	A	B	C
i	N	1	N
j	N	N	N
k	N	N	N
	N^3=2^30	N^2=2^20	N^3=2^30

Fully Associative:

For A:

1. wrt i ,we are accessing elements in column major fashion,so i loop contribution is N.
2. wrt k,due to fully associative cache,A[0,0] map to 0th line,A[1,0] map to 1st line and _ _ _ A[1023,0] map to 1023th line,and so access to A[0,1] to

a[0,BL-1] gets hit.So contribution of k loop is N/BL.
3. wrt j,as matrix is big enough to not fit in cache ,so blocks get replaced and contribution of j loop is N.

For B:
1. wrt i,we are only accessing B[0,0] element,i loop contribution is 1.
2. wrt k, we are accessing in column major fashion so k loop contribution is N.
3. wrt j, due to fully associative cache,B[0,0] in 0th line,B[1,0] in 1st line, and and B[1023,0] in 1023th line. So access to BL elements along the row lead to hits and hence contribution of j loop is N/BL.

For C:
1. wrt i, we are accessing in column major fashion, so i loop contribution is N.
2. wrt k, we are accessing the elements C[0,0] to C[1023,0] again and again,as it is fully associative so C[0,0] is in 0th line,_ _ _ ,and and C[1023,0] in 1023th line.So this gets all hits, and thus contribution of k loop is 1.
3. wrt j,as all rows are in cache,so access to BL elements along row lead to hits and hence contribution of j loop is N/BL.

	A	B	C
i	N	1	N
j	N	N/BL	N/BL
k	N/BL	N	1
	N^3/BL=2^26	N^2/BL=2^16	N^2/BL=2^16

Problem 3:

Compilation Command :
g++ -std=c++17 -pthread 251110053.cpp -o 251110053
./251110053 <input_file_path> <number_of_Producer_Threads> <Lmin> <Lmax> <Buffer_size> <output_file_path>

Sample Output:

./251110053 test_input.txt 6 2 5 10 test_output.txt

test_input.txt:
line1
line2
line3

line4
line5
line6
line7
line8
line9
line10
line11
line12
line13
line14
line15
line16
line17
line18
line19
line20

test_output.txt:

line9
line10
line11
line1
line2
line3
line4
line5
line12
line13
line14
line15
line6
line7
line8
line16
line17
line18
line19
line20

Synchronization used in Program:

Primitive	Role

mtxinput	serializes access to the input stream so only one producer reads from the file at a time.
mtxbuffer	Protects queueBuffer and the linesUsed counter .
mtxProducer	prevents interleaving during queue operations
mtxOutput	Ensures only one thread(consumer) writes to the output stream at a time.
condVar	Coordinates producers and consumers: producers wait when linesUsed + chunk.size() > bufCapacity; consumers wait when the buffer is empty until new data arrives or all producers finish.