CS18BTECH11039
CA_HW4

Name: - Raj Patil
Roll no: - CS18BTECH11039

[Question 1]

Solving for the time required to multiply two nxn arrays using a systolic array processor: - (Let the two matrices be labelled P and Q); Explanation:-

P is fed in row wise and Q- column wise

- the initial element from row 0 of P takes n cycles to reach till the end
- the total number of cycles are represented by the number of cycles taken by the last entry of last row of P in the systolic array (similar for Q). it enters the matrix after 2*n-2 cycles (from T =0 when the operations begin)
    - n-1 cycles for the first element of the last row to enter
    - n-1 cycles from here for the last element of the last row to enter
- further n cycles for this last element to be fully consumed.
- total cycles needed: - n-1 + n-1 + n = 3*n-2

Hence, for multiplication of a square matrix of size n*n, the systolic array processor takes ***3*n-2*** cycles for the complete operation.

[Question 2]

Original code (guided by indents)

```
for ( row=0; row<R; row++)
        for ( col =0; col<C; col++)
                for ( to=0; to<M; to++)
                        for ( ti =0; ti <N; ti++)
                                for ( i =0; i<K; i++)
                                        for ( j =0; j<K; j++)
                                                Output_fmaps [to] [row] [col] +=
                                                Weights [to] [ti] [i] [j] * Input_fmaps[ti] [S*row+i] [S*col+j]
```

Code tiled for first loop (changes highlighted)

```
for(RR=0;RR<R;RR+=T)//T is the tiling factor
        for ( row=RR; row<min(RR+T,R); row++)
                for ( col =0; col<C; col++)
                        for ( to=0; to<M; to++)
                                for ( ti =0; ti <N; ti++)
                                        for ( i =0; i<K; i++)
                                                for ( j =0; j<K; j++)
                                                        Output_fmaps [to] [row] [col] +=
                                                        Weights [to] [ti] [i] [j] * Input_fmaps[ti] [S*row+i] [S*col+j]
```

[Question 3]

Writing only the declarations of the asked functions: -

__device__ void addFunc1(int *a, int *b, int *c)

__global__ void addFunc2(int *a, int *b, int *c)

__host__ void random_ints(int* x, int size)

__host__ int main(void)

[Question 4]

Variables	Location
x_dim	register
y_dim	register
iteration	register
pqr	local
ABC	Global
maxValue	Global

[Question 5]

(A)

As we are given arrLen = 16, we are working with a 16x16 matrix.

we are using the int type and the cache size is 32 such entries :- the cache size is 32*4B = 128 Bytes

(INPUT and TRANSPOSE are initialized as int** : - even though a string is printed in main that says we are using 8B integers→using 4B integers.)

(B)

summarizing parameters :-

Cache size :- 32 entities (8B integers)
blocking factor :- 4

pattern of hits/misses repeats after 8 accesses(4 each for input and output matrix) when not blocking (cache is effectively refreshed after these 8 accesses)

observing only for the first 4 elements of the array and scaling accordingly:

Without Blocking	For input matrix	For output matrix	Total
Hits	192	0	192
Misses	64	256	320

Pattern hits/misses repeats after 32 accesses (16 each for input and output matrix) when we are using a blocked algorithm (cache is effectively refreshed after these many accesses)

Observing only for one block (4*4 elements of array) and scaling accordingly:

With Blocking	For input matrix	For output matrix	Total
Hits	192	192	384
Misses	64	64	128

[Question 6]

Initializing arrays :-

A = [[A00,A01]
,[A10,A11]]

B = [[B00,B01]
,[B10,B11]]

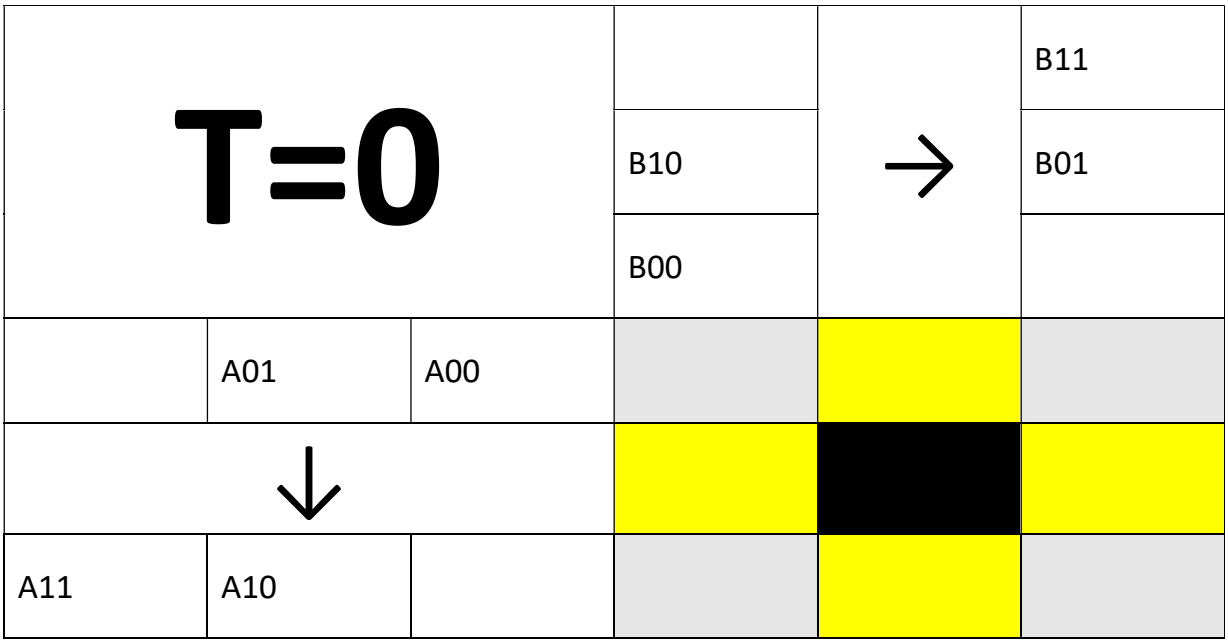
Displaying snaps for systolic array processor at different times: -

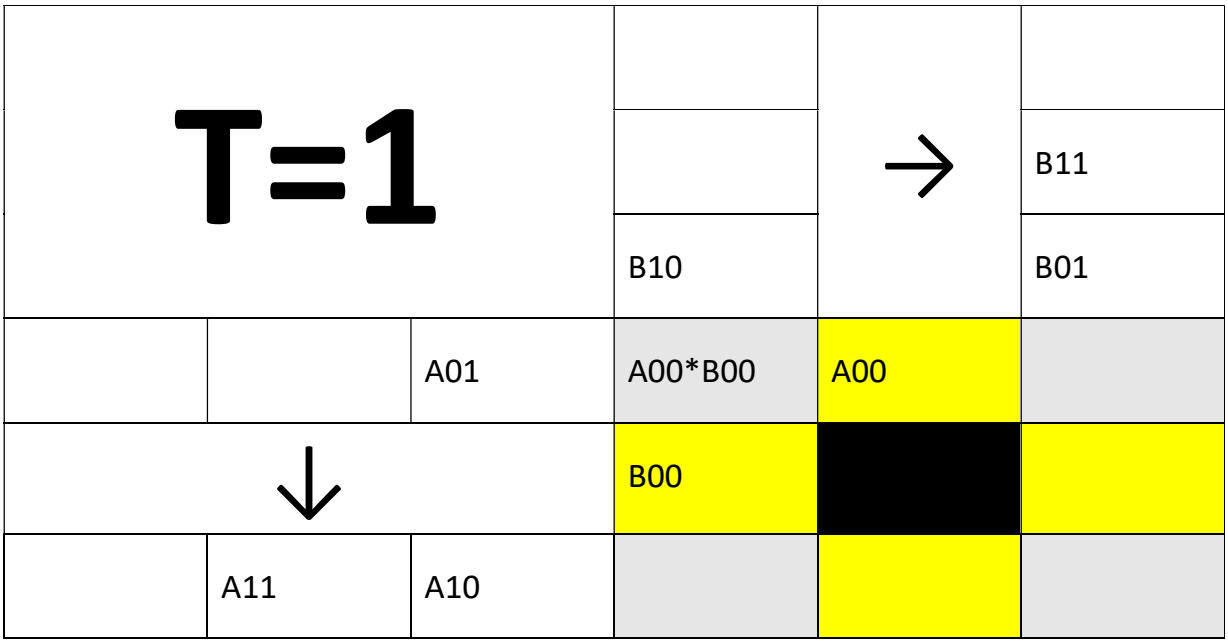
The gray boxes indicate the indexes of the output matrix just as they are arranged spatially.

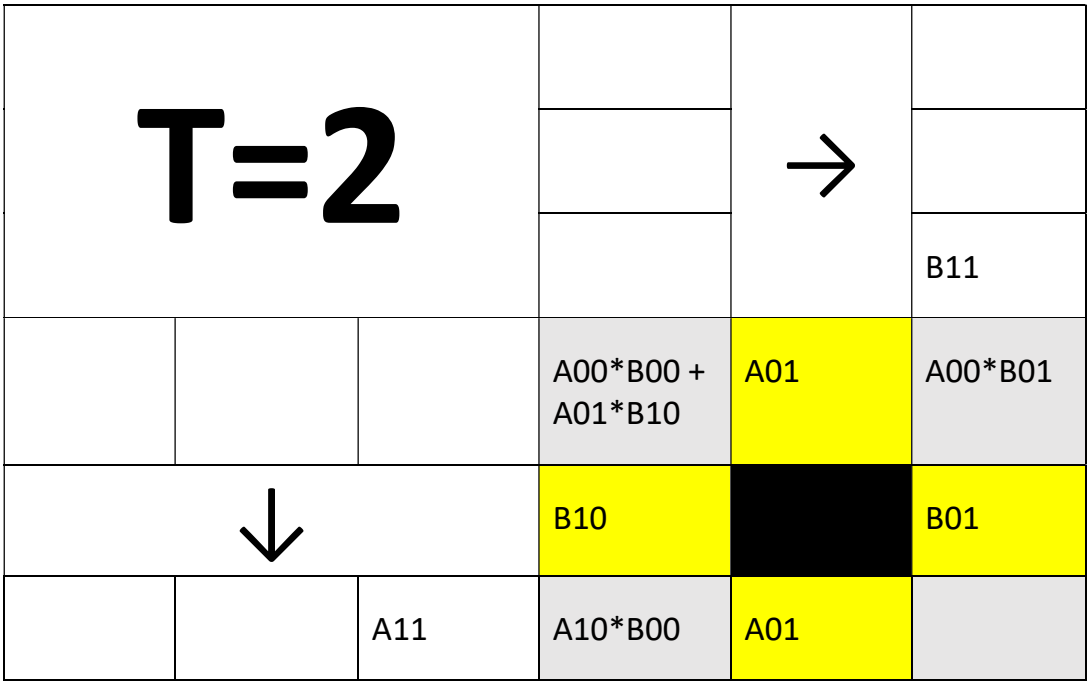
The black box is a filler to adjust alignment

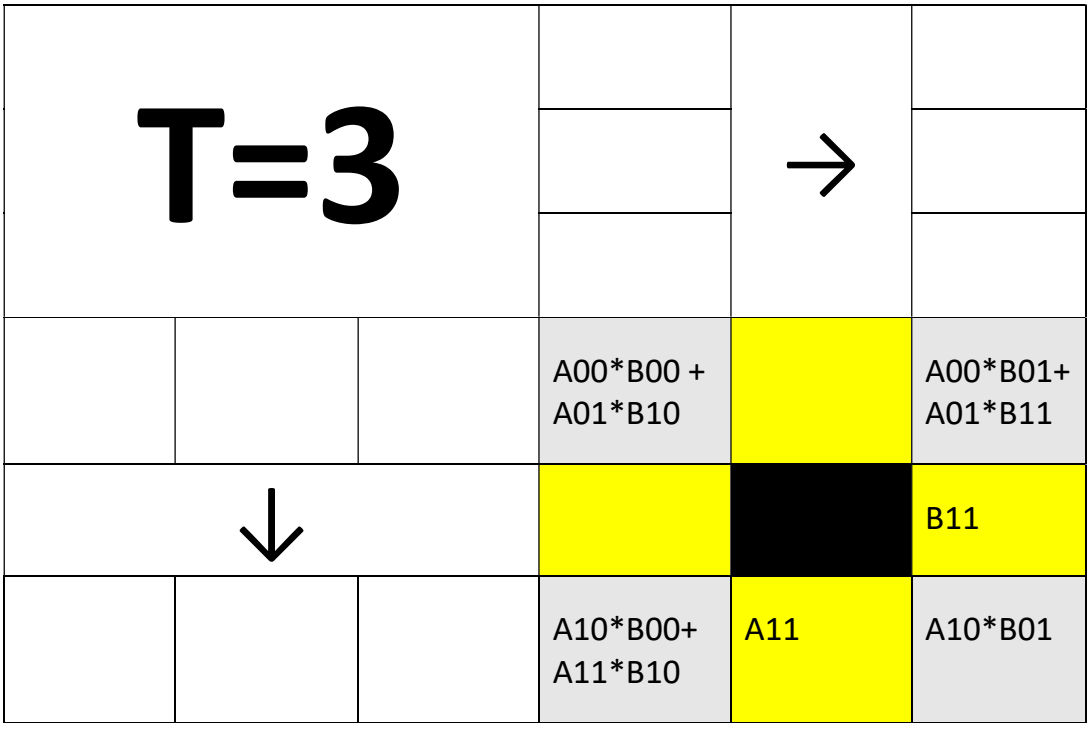
The yellow boxes indicate what element is being passed on in that specific row/col of the array

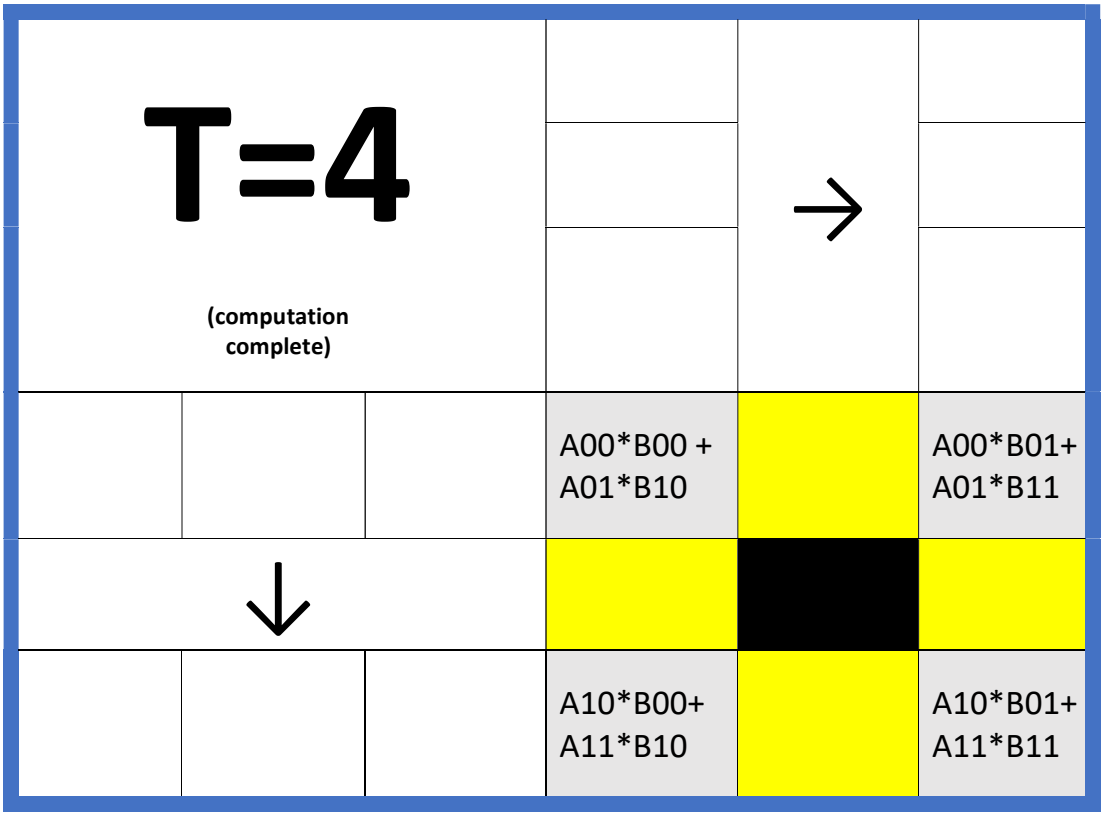
The arrows indicate the direction of movement corresponding to the yellow boxes with which they are in the same row/column.


T=0
B10
B11
B01
B00
A01
A00
A11
A10


T=1
B10
B11
B01
A01
A00*B00
A00
B00
A11
A10


T=2
B11
A00*B00 +
A01*B10
A01
A00*B01
B10
B01
A11
A10*B00
A01


T=3
A00*B00 +
A01*B10
A00*B01+
A01*B11
B11
A10*B00+
A11*B10
A11
A10*B01


T=4
(computation complete)
A00*B00 + A01*B10
A00*B01+ A01*B11
A10*B00+ A11*B10
A10*B01+ A11*B11

[Question 7]

Given:

32b value Z=0 10000001 01101100110011001100110 (5.7)

IEEE754 FP 32 number:-

Exponent bits :- 8
Sign bit :- 1
mantissa bits :- 23

(A) fetching leftmost 9 bits

Z_A = 0 10000001 00000000000000000000000 represents the value: - **4.0**

(B) fetching leftmost 12 bits

Z_B = 0 10000001 011 00000000000000000000 represents the value: - **5.5**

(C) fetching leftmost 15 bits

Z_C = 0 10000001 011 011 00000000000000000 represents the value: - **5.6875**

(D) fetching leftmost 18 bits

Z_D = 0 10000001 011 011 001 00000000000000 represents the value: - **5.6953125**

[Question 8]

Given vector width:-2; loading contiguous data with initial index at r2+20 into the vector register vr1

The instruction which does this is

v.ld vr1, 20[r2]

Semantics: -

Vr1 ← ([20+r2],[24+r2])

[Question 9]

Given the following code :-

```
for(i=0; i<N; i++)
        for(j=0; j<N; j++)
                X[i][j] = Y[i][j]*Z[i][j];
```

(A)

Case 1:

For an iteration: -

no of Dram bytes accessed = 3*8 = 24 bytes (assuming the arrays use IEEE754 FP64 (DP))

no. of FLOPs = 1 (the multiplication) (rest are integer operations)

Hence, AI for case 1 = **1/24 (FLOPs/Bytes)**

Case 2:

Total accesses: - $3*N^2$(stay the same) → total DRAM bytes fetched = $24*N^2$

Total FLOPS: - $N^2/4$

Hence, AI for case 2 = $N^2/(4*24*N^2)$ = **1/96 (FLOPS/Bytes)**

(B)

In this case when the elements of Z and Y are Zero, only 2 DRAM fetches occur: - i.e. 16 bytes whereas we access the normal 24 bytes for non-zero fetches of Z and Y

Total accesses: - $N^2/4 *24 + 3* N^2/4 * 16 = N^2*(6+12) = 18* N^2$

Total FLOPS: - $N^2/4$

Hence, AI for modified case 2 = $N^2/(4*18*N^2)$ = **1/72 (FLOPS/Bytes)** **(AI increased)**

[Question 10]

(A)

In 1 sec, no. of operations completed = 0.75*66 TOPS = 49.5 TOPs

OPs required for classifying one image = 1.5 GOPs

No. of images classified in 1 second = 49.5 TOPs/1.5 GOPs = 49.5/1.5 * 1000 = 33000 images

(B)

Finding out the arithmetic intensity when using AlexNet to classify one image: -

AI = no. of OPS for one image/ DRAM footprint in bytes

(IMPORTANT) (note: - mega is taken as 2^{20} and not 10^6)

For 8b fixed point version: -

AI = 1.5 * 10^9 OPS / 50 MB = (1.5*10^9/ 50*2^{20}) OPS/B

= **28.6102294921875 OPS/B** (take note of the note (affects answer))

For binarized version: -

AI = 1.5 * 10^9 OPS / 7.4 MB = (1.5*10^9/ 7.4*2^{20}) OPS/B

= **193.3123614336993243 OPS/B** (take note of the note (affects answer))

[Question 11]

Using AI*bandwidth = peak FLOP/s

Using MCDRAM: -

AI = 2199 GFlop/s /(372 GB/s) = **5.911 Flops/B**

Using DRAM: -

AI = 2199 GFlop/s /(77 GB/s) = **28.558 Flops/B**