a) Now given the following concrete image and kernel matrix, calculate the convolution result:

4	1	0
3	0	5
2	6	1

*

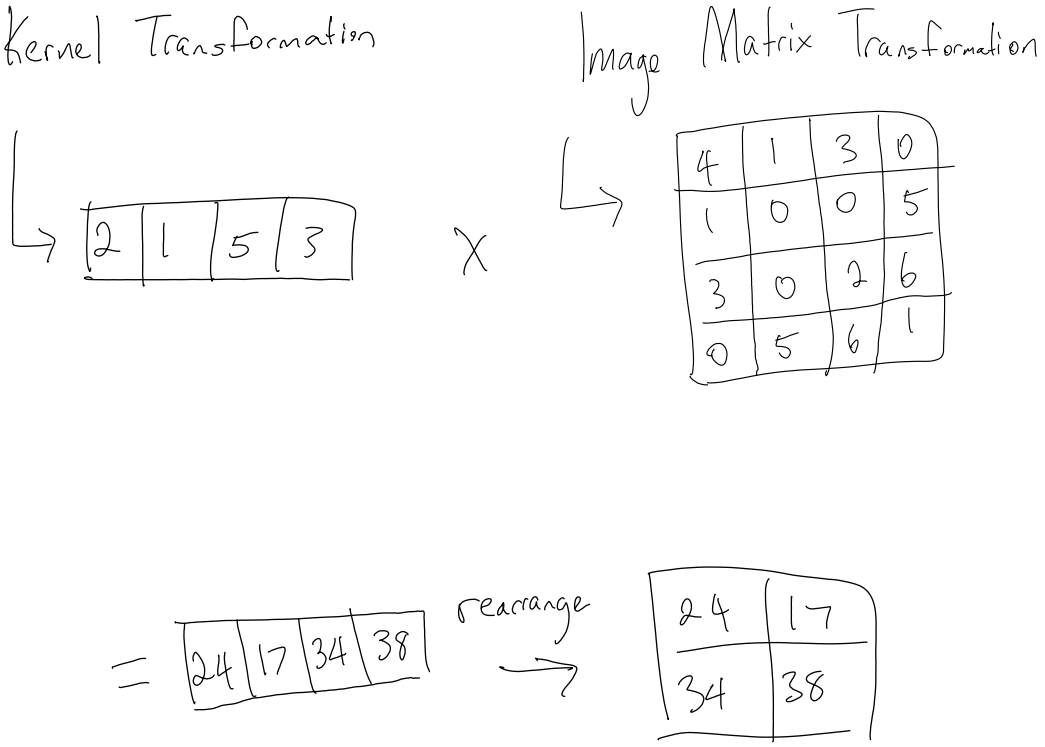
2	1
5	3

$8+1+15+0$	$2+0+0+15$
$6+0+10+18$	$0+5+30+3$

$=$

24	17
34	38

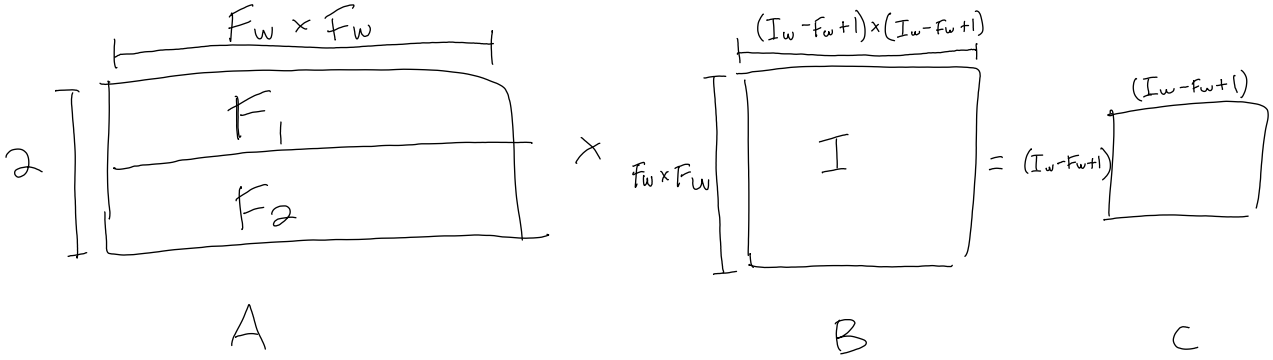
b) As discussed in class, such a convolution operation can be done by transforming it into a general matrix-matrix multiplication (GEMM). Show how this transformation and re-arrangement is performed on the example in a). What would be the matrix A and matrix B to be multiplied? Explain and draw figures as necessary. What are the pros and cons of casting the convolution as GEMM?



Benefits: Allows convolution to take advantage of software libraries or hardware accelerators focused on matrix multiplication.

Cons: More storage required to complete operation

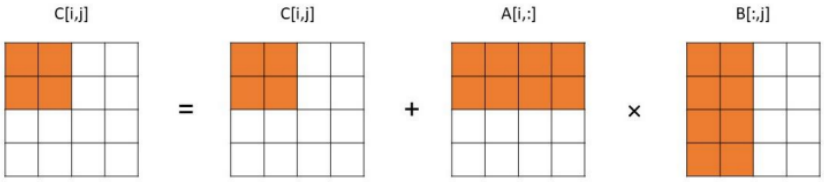
c) Now assume an input image I with 1 channel of general dimensions I_wxI_w, and two filters F_1 and F_2 each with 1 channel and dimensions F_wxF_w. Assuming no zero-padding and a stride of 1, what are the dimensions of the output feature map O? Write down the pseudocode to first transform I, F_1 and F_2 into A and B, and then perform the GEMM of A and B such that the output feature is calculated. What are the dimensions of A, B and C, where C is the result of the GEMM?



A: `A = new array [2][Fw x Fw]`
`A[0] = F1.reshape(1, Fw x Fw)`
`A[1] = F2.reshape(1, Fw x Fw)`

B:
```
B = new array [Fw x Fw] [(Iw-Fw+1)(Iw-Fw+1)
col_index = 0
for col in range(0, Iw-Fw+1):
    for row in range(0, Iw-Fw+1):
        window = I[row:row+Fw][col:col+Fw]
        B[:][col_index] = window.reshape(Fw x Fw, 1)
        col_index += 1
```

d) GEMM is one of the most fundamental operation in most applications in machine learning and beyond. As discussed in class, GEMM is inherently compute-bound with ample parallelism and locality. However, with the large matrix sizes, performance is often limited by on-chip memory size and/or memory bandwidth. In order to maximize data reuse, so-called blocked matrix-multiplication algorithms are often used. Shown below is the case of blocking a 4x4 matrix multiplication using a block size of 2:



Write down a pseudocode for performing a simple blocked matrix multiplication, assuming a block size b and square matrices A and B, both with dimensions equal to NxN (where $N>>b$ and $N\%b=0$). Feel free to use any blocking technique you like. In general, many blocking algorithms have been studied depending on the dimensions along which matrices are blocked, and their relative performance is correlated to the underlying memory architecture, e.g., cache hierarchy and configurations in a CPU.

```
for (row = 0; row <= N-b; row += b)
    for (col = 0; col <= N-b; col += b)
        C[row:row+b][col:col+b] =
            matrix_multiply(A[row:row+b][:],
                            B[:][col:col+b])
```

a) Study the code and observe the program output. You can also use your favorite debugger (e.g., using `ddd` as a graphical frontend for `gdb`) to walk through the example. Briefly explain the program behavior and output, e.g. by drawing a trace of SystemC thread executions.

1. Producer runs and fills FIFO with 'Visit www.' FIFO is full, producer waits for read event and suspends.

2. Consumer reads and prints out FIFO. FIFO becomes empty, and consumer waits for write event and suspends.

3. 1 & 2 are repeated until FIFO empties.

b) Modify the example to replace the custom FIFO channel with a corresponding `sc_fifo<char>` channel from the SystemC standard channel library. Compile and simulate the code to verify correctness. Vary the FIFO depth/size and observe the resulting behavior. Briefly explain any differences in behavior you see as a function of FIFO size/depth.

Depth = 1
Visit www.accellera.org and see what SystemC can do for you today!

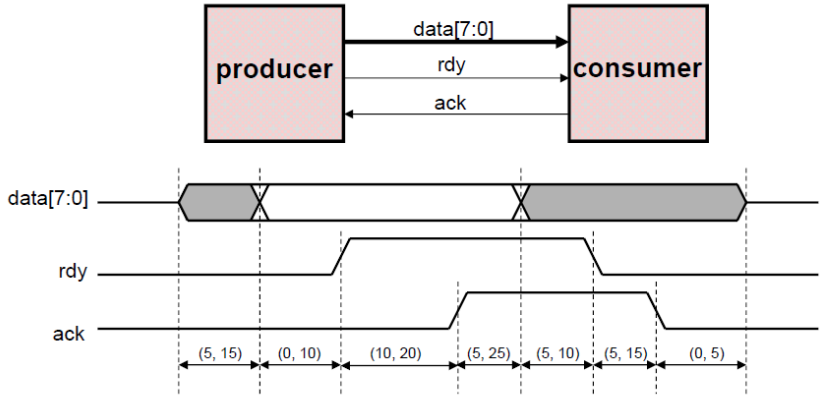
Depth = 2
V<1>is<1>it<1> w<1>ww<1>.a<1>cc<1>el<1>le<1>ra<1>.o<1>rg<1> a<1>nd<1> s<1>ee<1> w<1>ha<1>t
<1>Sy<1>st<1>em<1>C <1>ca<1>n <1>do<1> f<1>or<1> y<1>ou<1> t<1>od<1>ay<1>!

Depth = 3
Vi<1>sit<1> ww<1>w.a<1>cce<1>lle<1>ra.<1>org<1> an<1>d s<1>ee <1>wha<1>t S<1>yst<1>emC<1> ca<1>n d<1>o
f<1>or <1>you<1> to<1>day<1>!

Depth = 12
Vis<9>it www.a<1>ccel<9>lera.org<1> and<9> see wha<1>t Sy<9>stemC ca<1>n do<9> for you<1> today!<1>

Depth = 100
Visit www.accellera.org and see what SystemC can do for yo<9>u today!<1>

c) Now replace the `sc_fifo<char>` FIFO queue channel with `sc_signal<char>` and `sc_signal<bool>` channels that model the wires of a simple timed double handshake protocol connecting the *producer* and *consumer* modules, e.g. following this timing diagram (where tuples indicate minimum/maximum allowed times between events on the wires):



Insert code into the *consumer* that prints the current simulation time every time a character is received. Compile and simulate your design to verify its correctness and report on the simulation output.

```
V 21ns
i 52ns
s 81ns
i 112ns
t 141ns
 172ns
w 201ns
w 232ns
w 261ns
. 292ns
a 321ns
c 352ns
c 381ns
e 412ns
l 441ns
l 472ns
e 501ns
r 532ns
a 561ns
. 592ns
o 621ns
r 652ns
g 681ns
 712ns
a 741ns
n 772ns
d 801ns
 832ns
s 861ns
e 892ns
e 921ns
 952ns
w 981ns
h 1012ns
a 1041ns
t 1072ns
 1101ns
S 1132ns
y 1161ns
s 1192ns
t 1221ns
e 1252ns
m 1281ns
C 1312ns
 1341ns
c 1372ns
a 1401ns
n 1432ns
 1461ns
d 1492ns
o 1521ns
 1552ns
f 1581ns
o 1612ns
r 1641ns
 1672ns
y 1701ns
o 1732ns
u 1761ns
 1792ns
t 1821ns
o 1852ns
d 1881ns
a 1912ns
y 1941ns
! 1972ns

 2001ns
```

d) Now go back to the original custom FIFO design from a). Can you modify this design by inserting time `wait()` statements into the custom queue channel to match the timing, behavior and output of the design from c)?

```
V 60ns
i 90ns
s 120ns
i 150ns
t 180ns
 210ns
w 240ns
w 270ns
w 300ns
. 330ns
a 360ns
c 390ns
c 420ns
e 450ns
l 480ns
l 510ns
e 540ns
r 570ns
a 600ns
. 630ns
o 660ns
r 690ns
g 720ns
 750ns
a 780ns
n 810ns
d 840ns
 870ns
s 900ns
e 930ns
e 960ns
 990ns
w 1020ns
h 1050ns
a 1080ns
t 1110ns
 1140ns
S 1170ns
y 1200ns
s 1230ns
t 1260ns
e 1290ns
m 1320ns
C 1350ns
 1380ns
c 1410ns
a 1440ns
n 1470ns
 1500ns
d 1530ns
o 1560ns
 1590ns
f 1620ns
o 1650ns
r 1680ns
 1710ns
y 1740ns
o 1770ns
u 1800ns
 1830ns
t 1860ns
o 1890ns
d 1920ns
a 1950ns
y 1980ns
! 2010ns
```