Accessing IPython cluster clients and printing their ids to check.

In [69]:
```
import ipyparallel as parallel

clients = parallel.Client()
clients.block = True  # use synchronous computations
print(clients.ids)
```

```
[0, 1, 2]
```

Importing mpi4py and numpy.

In [70]:
```
%%px
from mpi4py import MPI
import numpy as np
```

Implementing Convolution, you don't need to modify this code.

In [71]:
```
%%px

def convolve_func(main, kernel, KERNEL_DIM, DIMx, DIMy, upper_pad, lower_pad
    num_pads = int((KERNEL_DIM - 1) / 2)
    conv = np.zeros(main.shape, dtype=int)
    main = np.concatenate((upper_pad, main, lower_pad))
    for i in range(DIMy):
        for j in range(DIMx):
            for k in range(KERNEL_DIM):
                for l in range(KERNEL_DIM):
                    if j + l <= DIMx + 1 and i + k >= num_pads and i + k <=
                        conv[j * DIMy + i] += main[(j + l) * DIMy + i - num_
    return conv


def convolve_func(main, kernel):
    DIMx, DIMy = main.shape
    convDimX = DIMx - (kernel.shape[0] - 1)
    convDimY = DIMy - (kernel.shape[1] - 1)
    conv = np.empty([convDimX, convDimY], dtype='int64')
    conv.fill(0)
    for i in range(convDimX):
        for j in range(convDimY):
            for k in range(kernel.shape[0]):
                for l in range(kernel.shape[1]):
                    conv[i, j] += main[i + k, j + l] * kernel[k, l]
    return conv
```

5 points: Load MPI communicator, get the total number of processes and rank of the process

In [72]:
```
%%px
#and also print total number of processes and rank from each process
comm = MPI.COMM_WORLD
print(f'total process number: {comm.Get_size()}, printing from process: {com
pass
```

```
[stdout:0] total process number: 3, printing from process: 0
[stdout:1] total process number: 3, printing from process: 1
[stdout:2] total process number: 3, printing from process: 2
```

5 points: Load or initialize data array and kernel array only in process 0(rank 0)

In [73]:
```
%%px
DIMx = 0
DIMy = 0
KERNEL_DIM = 0

#Add a condition such that these intializations below should happen in only
if comm.Get_rank() == 0:
    img = np.array(
        [[3, 9, 5, 9], [1, 7, 4, 3], [2, 1, 6, 5], [3, 9, 5, 9], [1, 7, 4, 3
         [2, 1, 6, 5]])
    kernel = np.array([[0, 1, 0], [0, 0, 0], [0, -1, 0]])
    DIMx = img.shape[0]
    DIMy = img.shape[1]
    KERNEL_DIM = int(kernel.shape[0])
```

10 points: Broadcast data and kernel array sizes from process 0 to all other processes

In [74]:
```
%%px
#broadcast data and kernel array sizes (think why we are broadcasting sizes)
DIMx = comm.bcast(DIMx, root=0)
DIMy = comm.bcast(DIMy, root=0)
KERNEL_DIM = comm.bcast(KERNEL_DIM, root=0)

# print(img.shape)
# print(DIMy)

pass
```

Initialize empty kernel array for all processes except rank = 0, why we are not initialzing kernel array for rank 0?

Ans:

- Because rank0 keeps the original kernel array to be broadcasted to other process later.

In [75]:
```
%%px
#initialize empty kernel array except for process 0(rank=0)
if comm.Get_rank() != 0:
    kernel = np.empty([KERNEL_DIM, KERNEL_DIM], dtype='int64')

pass
```

10 points: Broadcast Kernel array from rank 0 to all other processes.

In [76]:
```
%%px
#broadcast kernel array from rank 0 to all other processes
comm.Bcast(kernel, root=0)

pass
```

25 points: Split the rows in data array equally and scatter them from process 0 to all other process. To split them equally, number of rows in the data array must be a integral multiple of number of processes. MPI has ways to send unequal chunks of data between processses. But for here you can do with equal number.

In [77]:
```
%%px
#split and send data array to corresponding processses (you need to initiali
#process 0, similar to the random initializing done for kernel array)

sendbuf = None
if comm.Get_rank() == 0:
    sendbuf = img
inputData = np.empty([round(DIMx / comm.Get_size()), DIMy], dtype='int64')
comm.Scatter(sendbuf, inputData, root=0)

#Here does we initialize buffer for process 0 also, if so why?(Hint: because
#and receieve data)
pass
```

25 points: For convolution of kernel array and data array, you have to pass the kernel padding rows from one process to another. please see objective for more details. Send and Recieve rows from one process to other. Careful with the data size and tags you are sending and receiving should match otherwise commincator will wait for them indefintely.

In [78]:
```
%%px
#send padding rows from one process to other (carefully observe which proces
# which process receives the data)

if comm.Get_rank() == 0:
    paddingFirst = np.zeros(DIMy)
    sendArray = paddingFirst
    comm.Send([sendArray, MPI.INT], dest=comm.Get_size() - 1, tag=33)

    inputData = np.insert(inputData, 0, values=paddingFirst, axis=0)


elif comm.Get_rank() == comm.Get_size() - 1:
    receiveArray = np.empty(DIMy, dtype='int64')
    comm.Recv(receiveArray, source=0, tag=33)

    paddingLast = receiveArray
    inputData = np.append(inputData, [paddingLast], axis=0)

print(inputData)

pass
```

```
[stdout:0]
[[0 0 0 0]
 [3 9 5 9]
 [1 7 4 3]
 [2 1 6 5]]
[stdout:1]
[[3 9 5 9]
 [1 7 4 3]
 [2 1 6 5]]
[stdout:2]
[[3 9 5 9]
 [1 7 4 3]
 [2 1 6 5]
 [0 0 0 0]]
```

Why we are loading data into process 0 and broadcasting input data to all other processes? are there any other methods to load data into all processes (not for evaluation)

Ans:

1. Process 0 is playing the master role in the cluster. It is process 0's responsibility to coordinate and schedule data and tasks.
2. Yes, data could be kept in a central database that every process could access. Or store the split data in to a distributed database allocated for each process.

5 points: Perform Convolution operation by calling convolve_func() provided for each of the process with corresponding rows as arguments.

In [79]:
```
%%px
#convolution function arguments
#main - data array (flattened array), only the part of the data array that i
#kernel - kernel array
#DIMy - ColumnSize
#Dimx - RowSize
#upper_pad = upper padding row
#lower_pad = lower padding row

rank = comm.Get_rank()
size = comm.Get_size()

if comm.rank != comm.Get_size() - 1:
    sendArray = inputData[-1]
    print(f'sending to dest:{rank + 1}, tag:{11 * rank + 1}')
    comm.Send([sendArray, MPI.INT], dest=rank + 1, tag=11 * rank + 1)

    receiveArray = np.empty(DIMy, dtype='int64')
    receiveArray.fill(0)
    comm.Recv(receiveArray, source=rank + 1, tag=11 * (rank + 1) - 1)
    print(f'receive from dest:{rank + 1}, tag:{11 * (rank + 1) - 1}')
    inputData = np.append(inputData, [receiveArray], axis=0)

if rank != 0:
    sendArray = inputData[0]
    print(f'sending to dest:{rank - 1}, tag:{11 * rank - 1}')
    comm.Send([sendArray, MPI.INT], dest=rank - 1, tag=11 * rank - 1)

    receiveArray = np.empty(DIMy, dtype='int64')
    receiveArray.fill(0)
    comm.Recv(receiveArray, source=rank - 1, tag=11 * (rank - 1) + 1)
    print(f'receive from dest:{rank - 1}, tag:{11 * (rank - 1) + 1}')
    inputData = np.insert(inputData, 0, values=receiveArray, axis=0)

convResult = convolve_func(inputData, kernel)
print(convResult)

pass
```

```
[stdout:0]
sending to dest:1, tag:1
receive from dest:1, tag:10
[[-7 -4]
 [ 8 -1]
 [-2 -1]]
[stdout:1]
sending to dest:2, tag:12
receive from dest:2, tag:21
sending to dest:0, tag:10
receive from dest:0, tag:1
[[-6  2]
 [ 8 -1]
 [-2 -1]]
[stdout:2]
sending to dest:1, tag:21
receive from dest:1, tag:12
[[-6  2]
 [ 8 -1]
 [ 7  4]]
```

10 points: Gather the computed convolutional matrix rows to process 0.

In [80]:

```
%%px
#To receive data from all processes, process 0 should have a buffer

size = comm.Get_size()
rank = comm.Get_rank()

gatherSendArray = convResult.flatten()
gatherReceiveArray = None
if rank == 0:
    gatherReceiveArray = np.empty([img.shape[0] + 2 - (kernel.shape[0] -1),
                                   img.shape[1] - (kernel.shape[1] - 1)], dt

comm.Gather(gatherSendArray, gatherReceiveArray, root=0)

print(gatherReceiveArray)
pass
```

```
[stdout:0]
[[-7 -4]
 [ 8 -1]
 [-2 -1]
 [-6  2]
 [ 8 -1]
 [-2 -1]
 [-6  2]
 [ 8 -1]
 [ 7  4]]
[stdout:1] None
[stdout:2] None
```

Reshape the flattened array to match input dimensions

In [83]:
```
%%px
#Reshape the collected array to the input image dimensions
if rank == 0:
    gatherReceiveArray = np.reshape(gatherReceiveArray, (-1, DIMx))
pass
```

5 points: Test to check sequential convolution and MPI based parallel convolution outputs

In [84]:
```
%%px
if rank == 0:
    #main_grid is the actual input input image array that is flattened
    #convolution function arguments
    #main_grid - data array (flattened array)
    #kernel - kernel array
    #DIMy - ColumnSize
    #Dimx - RowSize
    #upper_pad = upper padding row
    #lower_pad = lower padding row

    #rename the below arguments according to your variable names

    #Entire convolution in a single process
    # conv1 = convolve_func(main_grid, kernel, KERNEL_DIM, DIMx, DIMy, upper
    conv1 = convolve_func(np.concatenate(([np.zeros(DIMy, dtype='int64')], i
    conv1 = np.reshape(conv1, (-1, DIMx))
    #recvbuf is the convolution computed by parallel processes and gathered
    #if you named it different, modify that name below

    #Checking with parallel convolution output
    print(np.array_equal(conv1, gatherReceiveArray))
```

```
[stdout:0] True
```

In [ ]: