ECE 562/662 - Hardware for Machine Learning

Lab 2: Parallelization using MPI

Goal: In this assignment, you will experiment with a parallelization technique to speed up a program. You will analyze the algorithm and identify the parts of the algorithm that can be parallelized. In particular, you will parallelize the convolution process by splitting the convolution matrix computation into rows between the processes.

Requirements: Python, IPython, Jupyter notebooks, mpi4py

Use Anaconda for managing python packages.
Follow the instructions provided in the below link to install Anaconda on Windows, macOS, and Linux Operating Systems- Please note that this lab does not work on Google Colab due to library issues.
Please check your code regularly in GitHub.

Anaconda Installation

Run the following commands in the terminal(Ubuntu & macOS) or Anaconda prompt(Windows)- Setup

1. conda create --name myenv
2. conda activate myenv
3. conda install -c anaconda ipython
4. conda install -c conda-forge jupyterlab
5. conda install -c conda-forge mpi4py
6. conda install -c anaconda ipyparallel
7. conda install numpy

To launch ipcluster and start running the jupyter notebook-

Ubuntu & macOS -

1. ipcluster start -n 3 --engines=MPIEngineSetLauncher & jupyter notebook
   Make sure that the console displays "Engines appear to have started successfully" before you proceed further.

Windows-

Run the following commands in the Anaconda prompt

1. start /b ipcluster start -n 3 --engines=MPIEngineSetLauncher --cluster-id='mpi'
   Once you run the above command, you must wait until the console displays "Engines appear to have started successfully" before you run the next command.
   If you see the error "[IPClusterStart] CRITICAL | Cluster is already running with [pid=<id here>]. use "ipcluster stop" to stop the cluster" when you try to run "start /b ipcluster start -n 3 --engines=MPIEngineSetLauncher --cluster-id='mpi'" then you should type this: "ipcluster stop --cluster-id=mpi" and then rerun the start command.

2. jupyter notebook
   For windows, please modify the following line:
   ```
   clients = parallel.Client()
   ```
   to the one below:
   ```
   clients = parallel.Client(cluster_id='mpi') #name with which you started the cluster
   ```

After running the above commands, jupyter notebook will be launched in the default browser.

Next navigate to the location you downloaded "Conv_MPI.ipynb" and click it to launch it.

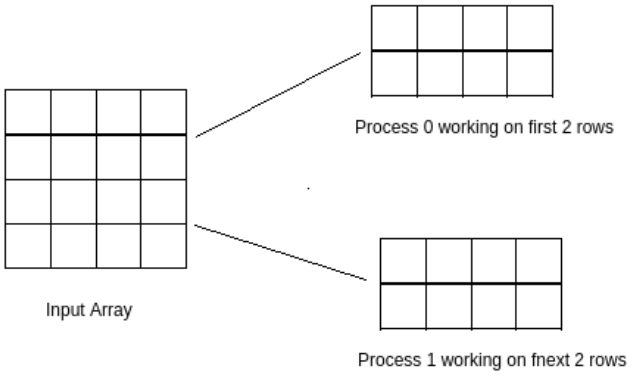
In this assignment, you will write an MPI code that computes matrix convolution parallely using 3 processes.

Inputs: Data array and Kernel array
Outputs: Convolution output

Steps: Tasks are divided such that you will be familiarized with MPI functions like broadcast, scatter, gather, send, and receive.

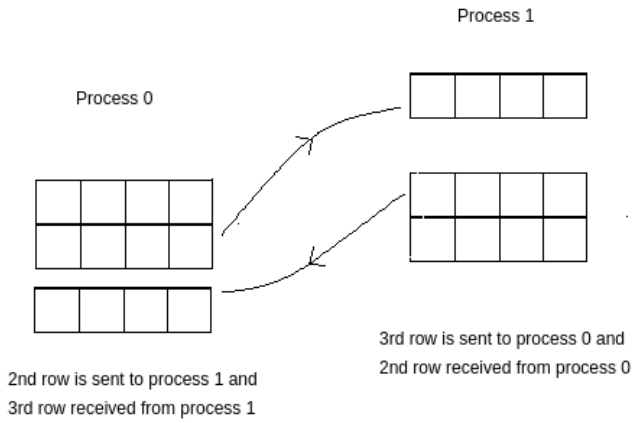
1. Load the input data, kernel data into process 0.
2. Broadcast data array and kernel array dimensions to all processes (from process 0 to all other processes).
3. Broadcast kernel array to all processes (from process 0 to all other processes).
4. In process 0, split the data array rows equally (Each split should contain number of array rows that is an integral multiple of the total number of processes)
5. From process 0, scatter each split to its corresponding processes.
   In the following example, the data array containing 4 rows should be split across 2 processes (row 1 and row 2 to process 0, row 3 and row 4 to process 1). First two rows of data

array should be scattered to process 0, next two rows to process 1

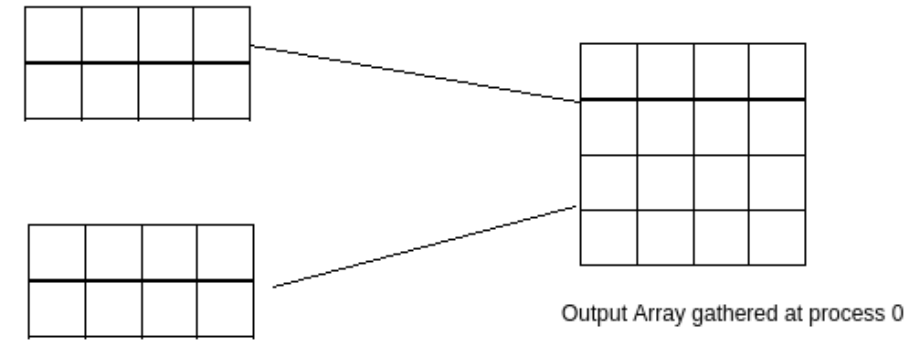


input array containing 4 rows is split between 2 processes each working on 2 rows

6. Now, each process has the kernel array and part of the data array. Kernel matrix should be applied on the data array. Input array is padded by zeros on the first and last rows of the data array (Check convolution algorithm about padding - process 0 has the first row of the data array part padded by zeros and the last process has the last row padded by zeros).
7. Assuming a kernel size of 3x3, for computing the kernel matrix, process 0 needs the 3rd row from process 1 and process 1 needs 2nd row from process 0. Send 3rd row from process 1 and receive that row at process 0 (In MPI, a send data should be received at the other side, so every transmission has a send command from source and receive command at destination). Send and receive all the rows that are needed by each process (Check convolution algorithm, for applying kernel array mask, boundary row in the data array part needs a row from the other process).



8. Compute convolution at each process (apply kernel mask to the data array part at the process). After this step, each process has a part of the data array of the convolution output.
9. Gather all the rows of the output array from each process at process 0.



Output Array gathered at process 0

Individual processes outputs