AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING

Computer Engineering and Software Systems Program
i-Credit Hours Engineering Programs (CHEP)

CSE 415, High Performance Computing

Project
High Pass Filter

Team Members:

Mohamed Ashraf Abdulhamid	17P6009
Ahmed Wael Fikry	17P6000

Submitted to:

Dr. Tamer Mostafa

TA Mohamed Atef

Contents

Description

High pass filters are used to pass high frequency components by blocking the low frequencies components, it only passes signals above the threshold making their value equal 1 and the signals below the determined threshold equal 0.

Edges are high frequency content. In edge detection, we want to retain these edges and discard everything else. hence, we should build a kernel that is equivalent of a high pass filter, as high pass filter is like an edge detector. It gives a high when there is a significant change in the adjacent pixel values.

Thus, high pass filter can be implemented by applying different filters such as Laplacian, Sobel or Scharr.

Our project aims to implement High pass filter using parallel programming by splitting the image into chunks by the master and then distribute them to the other processes, hence they apply the filter, Laplacian or Sobel then return the image part to the master(rank=0) to concatenate it and produce the final image after filtering.

The filter is build using OpenCV, python and MPI for applying the filter in parallel

Analysis

We needed to implement the problem in parallel programing using MPI, which is a portable library used for writing parallel programs using the message passing architecture, so we used Python programming language, allows the use of python's efficient high-level data structures and approach to object-oriented programming with dynamic typing and dynamic binding and it has mpi4py package, mpi4py can be used for the processes communication and the heavy computational part can be implemented in Python, This package builds on the MPI specification and provides an object oriented interface resembling the MPI-2 C++ bindings. It supports point-to-point (sends, receives) and collective (broadcasts, scatters, gathers) communication of any picklable Python object, as well as efficient communication of Python objects exposing the Python buffer interface (e.g. NumPy arrays and builtin bytes/array/memoryview objects).

Design

Mainly, the design to parallelize the code is done by **mpi4py** package by using:

- **MPI.COMM_WORLD** contains all the process in mpi4py
- **Get_size()** Get how many processes are running in a given communicator. Size: the total number of processes in a communicator
- **Get_rank()** the rank of the calling process within that communicator.
- **comm.send()&comm.recv()** to send and receive data between processors

Also, we have used **OpenCV** library, It was generated to support a common infrastructure for computer vision operations and use system behaviour in financial products. It generally targets image processing, faces recognition, video capture and many other applications.

We need it in this project to use:

- **cv2.imread()**, take path string as input representing the path of the image to be loaded
- **cv2.imshow()**, to display the image in a new window
- **cv2.Laplacian()**, to apply Laplacian kernel
- **cv2.Sobel()**, to apply Sobel kernel

NumPy library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

Used it for:

- **np.array_split()** to split the input image into sub images
- **np.concatenate()** to append the image chunks at the master

Parallelization idea

To speed up the process we can divide the image into smaller images and apply the filter to them at the same time

At rank 0 (Master)

1. read the image
2. divide the image into chunks
3. distribute the chunks to other processors
4. receive the results from each slave node
5. show the result

Other processors

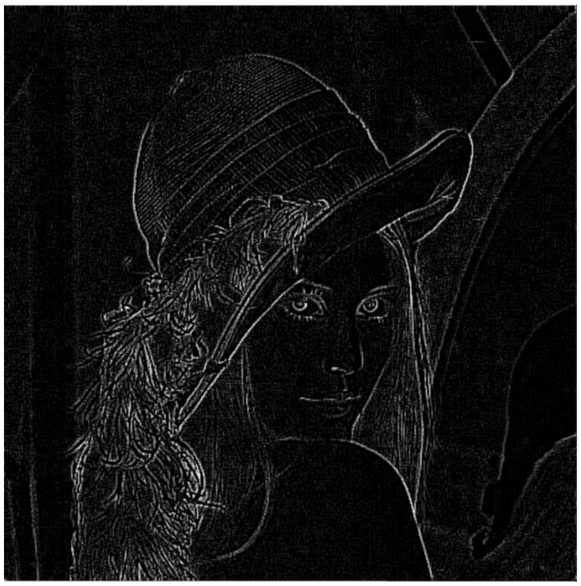
1. Receive the sub image from the master node (rank 0)
2. apply the filter (Laplacian or Sobel)
3. send the filtered sub – image to the master

I/O Example

- **Input**

➢ **Output**

Laplacian



Sobel