Introductory Python-based MPI Assignment

Alina Lazar
Department of SCSIET
Youngstown State University
Youngstown, OH
alazar@ysu.edu

***Abstract*—The purpose of this paper is to describe an assignment used for assessing student learning about parallel programming in an introductory programming course. However, the assignment could be easily adapted for upper division undergraduate courses. For this assignment students are required to run sequential and parallel Python code to generate the Mandelbrot set image. Using the Python framework IPython Parallel together with mpi4py, students can become familiar with parallel and distributed processing. Furthermore, the assignment introduces students to the basics of parallel computing through an interesting example and also shows them how to evaluate the execution time of sequential and parallel implementations.**

I. Motivation for the Assignment

As multi-core hardware platforms evolve and become widely available, people will have high-performance computing at their fingertips. To prepare students for careers in computing and particularly in parallel and distributed computing (PDC), computer science (CS) educators are finding it increasingly challenging to decide what to teach and how. The college level introductory CS course sequence only exposes students to sequential type programming and data structures concepts, no matter what programming language is used. PDC is mainly taught in upper division courses such as parallel computing, operating systems and computer architecture. The community has long recognized [1], [2] the benefits of exposing students to PDC concepts as early as possible during their studies.

Currently, Python is the most popular programming language according to TIOBE Index [3] and also one of the favourite programming language to teach in the college introductory programming courses CS1 and CS2. Python's simpler syntax lets students get started faster and provides an easier learning curve for those with no prior programming experience. It also sets the stage for later coursework in object-oriented programming, parallel computing, data science, AI, and machine learning. Not only that, but integrated development environments (IDEs) such as Jupyter Notebooks and VS Code have only helped increase Python's popularity. In the same time, as all the modern computing devices in use today have multiple cores as well as GPU in many cases there is a push to integrate parallel and distribute computing in the early CS courses. However, in most cases CS1 and CS2 still teaches students to solve problems using only sequential thinking or in the best case concurrency in problem solving is demonstrated using unplugged activities [4]. Many Python libraries for Just In Time (JIT) compilation, multiprocessing and high performance computing exist, however IPython Parallel [5] makes explicit parallel computations interactive which means it is very appealing for beginners because it removes the barrier of accessing remote computers.

With IPython Parallel, you have all the power of IPython's inspection, interactivity, magic, and debugging at your fingertips, regardless of where you run your code. This makes it especially well-suited to prototyping and experimentation. IPython Parallel also presents standard APIs such as Python Executors, compatible with many other implementations, to make it easy to migrate to and from IPython Parallel, enabling developers to write code that uses a single multi-core laptop and to deploy it to a thousand cores on an HPC cluster or in the cloud.

For this assignment we are using the IPython Parallel package together with the mpi4py package. IPython Parallel is a light weighted framework for controlling clusters of IPython processes, that works well on a single machine by using all it's cores to run computations in parallel. Once you develop your parallel code locally, IPython Parallel lets you scale it up resiliently and elastically to clusters with many nodes for speeding up your solution or for solving even larger problems. Under the Anconda platform, IPython Parallel can be installed with a single conda or pip install command, under a separate conda environment. This installation includes an extension for both the classic Jupyter Notebook and JupyterLab. The IPython's parallel computing architecture has been designed from the ground up to integrate with the Message Passing Interface (MPI). To use MPI with IPython, you will need to install the mpi4py package.

II. Assignment

The goal of this assignment is to use the IPython Parallel together with mpi4py package to introduce students in introductory programming classes to parallel computing concepts. The starting point of this assignment is sequential Python code in Listing 1, that computes and displays the Mandelbrot set in Figure II. The Mandelbrot set is considered the most popular fractal. Generating the image of the Mandelbrot set is a widely used benchmark to test computer systems as well as programming language implementations because it is computational intensive.

The Mandelbrot set computation is relatively easy to parallelize using MPI because the value of each pixel in the image can be calculated independently, without any information

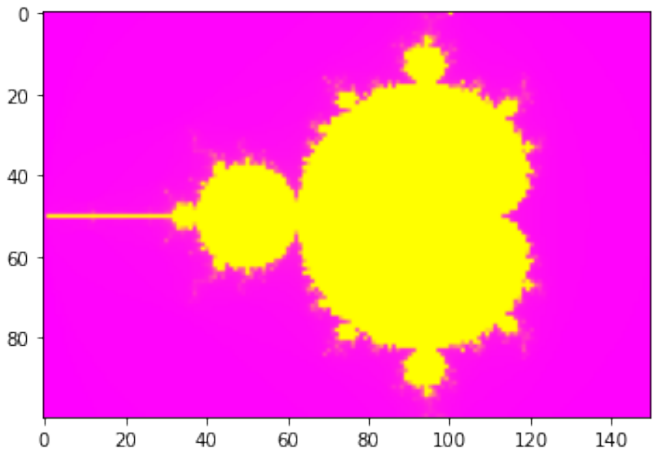

Fig. 1. Mandelbrot set

about the values of any surrounding pixels. The main idea of this embarrassingly parallel example is to group the pixels in a square region together and assign them to be processed by the same process or core. To make it simple and easier for students, the parallel MPI implementation is done in a Jupyter notebook using IPython Parallel. A static type MPI parallel implementation of the Mandelbrot set is also provided to the students. The main goal of the assignment is to run this parallel implementation using different number of processes, i.e 2, 4, 8, 16. Students are asked to run these experiments and to plot a bar plot to show the time it takes to finish the computation using different number of computing cores.

Listing 1. Sequential Mandelbrot set Python code

```
MAX_ITER = 100

def mandelbrotSet(complex):
    z = 0
    i = 0
    while abs(z) <= 2 and i < MAX_ITER:
        z = z**2 + complex
        i += 1
    return i
```

This challenging assignment introduces students to parallel and distributed computing in an easy, unintimidating way. Students should be familiar with Python and Numpy by the time in the semester when they have to solve this assignment. This assignment only requires access to a local computer and the entire code can be run in Jupyter Notebooks. The skills acquired while solving this assignment will be useful for understanding how parallel code is designed. The assignment can be extended to present a dynamic task assignment parallel solution for the Mandelbrot set. The same approach can be used to present the generation of other fractals, such as Julia or other problems that require a high amount of computations.

REFERENCES

[1] X. Suo, O. Glebova, D. Liu, A. Lazar, and others, "A survey of teaching pdc content in undergraduate curriculum," *2021 IEEE 11th Annual*, 2021.

[2] T. Newhall, K. C. Webb, V. Chaganti, and A. Danner, "Introducing parallel computing in a second CS course," in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. ieeexplore.ieee.org, May 2022, pp. 321–329.

[3] "TIOBE index," https://www.tiobe.com/tiobe-index/, Dec. 2021, accessed: 2023-1-27.

[4] S. K. Ghafoor, D. W. Brown, M. Rogers, and T. Hines, "Unplugged activities to introduce parallel computing in introductory programming classes: an experience report," in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '19. New York, NY, USA: Association for Computing Machinery, Jul. 2019, p. 309.

[5] "Using IPython for parallel computing — ipyparallel 8.4.1 documentation," https://ipyparallel.readthedocs.io/en/latest/, accessed: 2023-1-27.