Parallel Processing of GDP Model based on the growth of income annually by Country

Venkata Durga Narendra Kumar Koppuravuri (U00337159) April 20,2022

***Abstract – This paper shows the implementation and result that was achieved for my class Project of Parallel Computing CS523. The goal of this project was to achieve data parallelization using a multiprocessing library that is built in python and learning about its available classes. I found that the Pool class allows you to do multiple jobs per process which makes it more efficient than Process class.***

# INTRODUCTION

This project mainly aims at learning and understanding the Multiprocessing in python using python’s built-in library. I started the project with the idea learning the code for building a GDP model in python. Then to specifically determine what kind of Model do I had to implement since there are many factors associated with the word GDP itself, I decided to check the GDP growth of each country annually over the past 50 years.

Once that was decided I built a code for GDP model. For that model, I have outlined and decided the requirements that I needed for the dataset to be used. Once this was achieved, I started to code serially first to make the code working.

After the successful Serial code, I then started to implement the multiprocessing library in python, and which lead to me in learning more about the classes and how they worked. Initially I used Process Class but since I had to do data parallelization it was not an effective method to follow. Hence, I used Pool Class in my project. Also, while working with Process and Pool class I observed few time differences while running the code and plotting of the graph. Further in this paper I will add what were the libraries used in this project and explain the Process and Pool class of multiprocessing library I python.

# LIBRARIES AND METHODS

## Matplotlib

Matplotlib is a Python graphing library dedicated to creating animations. The module only has two methods. One creates animations by collecting a series of Matplotlib artist objects (like the Axes or Line2d objects that represent either the locations of the things you are graphing or the way they are graphed). The other applies a function you supply it to repeatedly update a graph. In my project, I have used the. FuncAnimation() function, which will repeatedly call a function to update the graph’s information. Matplotlib will display animations on its own.

## Time

Time is a library in python that is used to handle time related tasks. I used time() function to measure the performance of the program.

## tkinter

The tkinter library also known as “Tk interface” is a GUI toolkit. I used tkinter to create the front end for accepting the inputs (in my case, countries of which we want to plot the graph).

## Multiprocessing

Multiprocessing refers to the ability of a system to support more than one processor at the same time. Applications in a multiprocessing system are broken into smaller routines that run independently. The Multiprocessing library in Python supports spawning processes to achieve parallelism. Python’s Multiprocessing module contains two classes, the Process and the Pool, that can allow us to run a certain section of code simultaneously.

## Process Class

Process class works better when processes are small in number and IO operations are long. It basically suspends the process executing I/O operations and schedules another process parallel. Hence, it allots memory to all the processes, which might cause loss of memory if there are large or more repetitive tasks in hand. It's used when function-based parallelism is required, where I could define different functionality with parameters that they receive and run those different functions in parallel which are doing various kinds of computations. Here two functions to pay attention are .start() and .join()

.start() helps in starting a process and that too asynchronously.

.join() method on a Process blocks execution of the next process until the previous process has finished, but because we called .start() on both p1 and p2 before joining, both processes will run asynchronously. The interpreter will, however, wait until p1 finishes before attempting to wait for p2 to finish.

Note: A process cannot join itself because this would cause a deadlock. It is an error to attempt to join a process before it has been started.

## Pool Class

Pool class works better when there are more processes and small IO wait, hence offering a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes i.e. data based parallelism. It waits till the I/O operation is completed & does not schedule another process. Only the process under execution is kept in the memory. Pool class is easier to use than the Process class because you do not have to manage the processes by yourself. It creates the processes, splits the input data, and returns the

result in a list. There are four variant functions provided with Pool class in Python.

.apply() Call function with arguments args. It blocks until the result is ready.

.apply\_async() It is better suited for performing work in parallel.

.map() A parallel equivalent of the map() built-in function (it supports only one iterable argument though, for multiple iterables). But it blocks. it distributes a bunch of arguments to the same function.

.map\_async() It is better suited for performing work as map in parallel.

# Pool and Parallelization Code used



This constructor creates a pool of processes where the number of processes to keep in a pool is specified by processes parameter. If we don't provide processes parameters, then output of os.cpu\_count() method is used to specify processes.

Text

Description automatically generated

apply\_async() takes an args argument that accepts the parameters passed to the ‘function- to-be-parallelized’ as an argument. It lets you execute the processes in parallel asynchronously, that is the next process can start as soon as the previous one gets over without regard for the starting order.

Graphical user interface, application

Description automatically generated

Using read\_excel() function, you can read one or many sheets present in excel I am storing them in list of lists country wise.

While developing the parallelization section of code I came across the error as follows

*“if name == main*

*An attempt has been made to start a new process before the current process has finished its bootstrapping phase. This probably means that you are not using fork to start your child processes and you have forgotten to use the proper idiom in the main module: RuntimeError on windows trying python multiprocessing*

*AttributeError: partially initialized module has no attribute 'main'”*



So, it is important when using a multiprocessing library to spawn additional processes, it’s important to enclose the main body of the program in an if statement. To check if the special name variable is equal to main. When the MP code starts a new process, it tells it to execute the Read\_excel function. However, that new python process does not jump straight into the Read\_excel Function, because it does not know what that is. It needs to run through the entire script first to find the read\_excel function and become aware of any other dependencies. Python gives those processes that are spawned a different name than the main module so that I can tell them apart. If I did not include the if statement, then all those processes that are spawned would execute the section of the code within it. Spawning more and more processes is a problem and Python will throw an error if this line is not included which prevents that from happening.

# RESULT

The below image is of the front-end part of the code where you can select the countries from the dropdown list, also you can add as many countries as you want and then submit it. As for the dataset, there are a total of 218 countries. All the countries you select will be stored in a list and then that list will be passed to the multiprogramming part of the code which was used by the main method and pool class.

Graphical user interface, application

Description automatically generated

This is the output that is generated, that is the GDP growth of all the selected countries are plotted simultaneously with each line representing different country using matplotib library.

Chart, line chart

Description automatically generated

Text

Description automatically generatedFollowing are the results when the serialized and parallelized code was run with cores 4,6 and 8 respectively

Text

Description automatically generated Text

Description automatically generated

The above results were observed when I passed total of 4 countries in each execution. There are multiple factors that needed to consider when running the code on different machine there are different background processes running on a system that could impact the duration of the execution of the code and multiple different factors needed to be considered of the configuration of the system that you are running the code on.

# CONCLUSION

The importance was not just to achieve the processing speed of the program, but also to develop a deep understanding of the program’s behavior. In my opinion, choosing an appropriate approach depends on the task in hand. In my learning, in some cases, you might see that the serial

code runs faster than multiprogram code, because there aren’t any complicated tasks at hand. But you will see the results as the tasks become more and more complicated. The Pool allows you to do multiple jobs per process, which makes it easier to parallelize your program if you have a million tasks to execute in parallel.

# REFERENCES

1. “[multiprocessing](https://docs.python.org/dev/library/multiprocessing.html#module-multiprocessing) — Process-based parallelism”
2. “Introduction to Multiprocessing and Process in Python <https://www.codesdope.com/blog/article/introduction-to-multiprocessing-and-process-in-pyt/>”
3. “Parallel Processing in Python – Practical Guide with Examples” by Selva Prabhakaran
4. “Ellicium – Python Multiprocessing”
5. Dataset - ﻿<https://data.worldbank.org/indicator/SP.POP.GROW?end=2020&start=2000>