Doctoral School in Informatics for Climate Change

******

Master in Informatics for Climate Change

******

Module CCI – 14: High Performance Computing Systems

******

Project

Parallel programming with Python, Case study:

GlobalLandTemperatureByCity.csv

Submitted by:

Christiane COULIBALY

Adamou MOUSSA ISSAKA

Juste GARBA

David DEGBEY

Lecturers:

Dr. Tegawandé F. BISSIYANDE

Abdoul Kader KABORE

June 2021

Contents

List of figures

Introduction

Nowadays, in the domain of Climate, we use a lot of big data. Processing these data requires very powerful machines. This is the reason why in most of the climate data research center, High Performance Computing Systems, to process climate data. Wascal which is one of the west african reference center in the climate area, owns its HPC and this is why basically the training of Wascal include a course on HPCS. To program HPCS, we use parallel programming.

Parallel computing refers to the process of breaking down larger problems into smaller, independent, often similar parts that can be executed simultaneously by multiple processors communicating via shared memory, the results of which are combined upon completion as part of an overall algorithm. The primary goal of parallel computing is to increase available computation power for faster application processing and problem solving.

As a proof of performance, a project was given to us at the end of the course. The project is consisted of 2 exercises. After giving the environment and tools used for both, we intend to focus on an approach to solve the exercise 2 in this document.

1. Environment & Tools

To conduct the project, I used the following equipment and tools:

1.1. Hardware

- Laptop Lenovo Thinkpad (provided by Wascal)
- CPU : Intel Core i7
- Processor speed : 1.8 GHz (by core)
- Ram : 16 Gb

1.2. Software

- *For both exercises*:

  OS: Ubuntu 20.04 (Focal fossa distribution)

- *For exercise 1*:

  Compiler: g++-9 --- 9.3.0-17

  (Result of the command “dpkg --list | grep compiler”)

  File editor: nano

- *For exercise 2*:

  Anaconda 3 environment

  Python 3.8

  Jupyter notebook

2. Answer to the questions

2.1. Exercise 1

The purpose of this exercise is to help us deepen our comprehension of parallel programming in the low-level language c++ using the ***pthread*** library. Another purpose is also to let us be familiar to modular programming by separating our main program from the helpers file, which contains the treatment. At this level, as required by the lecturer, we provided the source code, the makefile and the file result.txt. We used the command ***time*** before running ./pi_estimator.bin and the command ***top | grep pi_estimator*** to monitor **in real time** the memory and the cpu usage as well as the time of execution. All the output is redirected in result.txt.

2.2. Exercise 2

The purpose of this exercise is to introduce us to parallel programming in python. Indeed, after seeing the basic principles in c++ language, we intend to see how to use parallel computing libraries in Python.

2.2.1. Calculation and display of the average temperature for each country

To do that, as asked by the lecturer, we used the library ***multiprocessing*** from python libraries. We used two main classes to do the work: The Class ***Process***, and the

class ***Lock***. We provided the well-commented source code as a jupyter notebook file. Here are the different steps:

- **Importing libraries**

First, we imported the ***pandas*** library to manipulate pandas dataframes (and series) with specific functions. **Second, we imported the necessary library for parallel programming in python: *multiprocessing***. We specifically imported the class Process and Lock.lhll

- **Importing the file and storing data in pandas dataframe**

Then, we imported the data with the function ***read_csv*** of the pandas library of python. So Then we store the data in a dataframe that we called ***df***. This will be our general dataframe during this exercise.

- **Storing the countries in a list**

This step consisted in getting all the distinct countries in a series and storing them in a list for further more complex manipulations. We use the method ***.append()*** of the list object to construct the list of countries called l during all this exercise.

- **External function : compute_mean()**

This step is where we define the function, which does all the treatment itself. We used a **for loop** to browse each country of a list of 20 countries at once and calculate the average of each. We did an analogy with the c++ program. Here the number of countries is 159 **so we decided to choose x = 20 countries** since we have a maximum of **8 threads** created at the same time. So each thread will deal with x = 20 countries at once until the last list which will contain exactly 19 countries. We provide more details in the main function section. Here also, one important thing we used is the Lock class by calling the methods .acquire() and .release() to “protect” the print statement which is a critical code.

- **Main function : main()**

Here, with the ***“if __name__ == "__main__":”*** statement, we confirm if it is the main process, which is running. The next statement is to declare and instantiate a ***Lock***

object. The next statement is to use the function ***cpu_count()*** of the multiprocessing library to get the maximum number of concurrent threads/processes that we can use at the same time. It is the ***equivalent of the function omp_get_max_threads()*** in omp.h library in c++. This max number is stored in a global variable called ***thread_max***. As we said above, we did an analogy with our c++ program to get thread_max and to get consequently the number of countries to treat by one process/thread at once. The next statement is to store the number of countries in the variable called ***n_countries*** with the function nunique(). The next statement is used to get the incremental step count, which represents the x we chose here. Then we can say that **count == x**.

The next block of statements is to create a list L, which contains 8 lists of 20 countries each.

Then the next block of instructions is to set to instantiate an object of the class ***Process*** but without any argument for the moment and to start it.

The next block of statements is meant to instantiate the process objects but now with arguments. The arguments are ***lock*** (the lock variable to protect critical code) and ***lst*** which will browse each of the sub lists of (count =) 20 countries in the great ***list L***. We put intentionally a variable ***proc_id***, which is used to check which process is started and which one is ended. We start all the processes with the method ***.start()*** of the class Process

The next block of instructions is set to stop all the created processes, especially with the method ***.join()*** of the class Process.

Here is a partial screenshot of the output of this program:

```
Average of  Denmark  is 7.8026399241945725
Average of  Afghanistan  is 13.816496896263587
Average of  Uzbekistan  is 11.946573813309133
Average of  Ghana  is 26.319964426877508
Average of  Taiwan  is 21.68291664251478
Average of  Poland  is 7.671972296106832
Average of  Jordan  is 18.360980886539238
Average of  Panama  is 26.745667996011946
Average of  Argentina  is 16.999215885618266
Average of  Turkey  is 12.951888167466613
Average of  Ethiopia  is 20.611525296730854
Average of  Philippines  is 26.51646246746454
Average of  Bolivia  is 11.352980175438574
Average of  Ukraine  is 7.8221839538832425
Average of  Djibouti  is 29.152790108564506
Average of  Suriname  is 26.429474975938355
Average of  Azerbaijan  is 11.11366381418095
Average of  Australia  is 16.701462142476366
Average of  Kazakhstan  is 4.340299609693073
Average of  Madagascar  is 22.11697617917819
Average of  Guinea Bissau  is 27.05718546231962
Average of  Sudan  is 28.072830827505655
Average of  Guinea  is 25.509399201596743
Average of  Cambodia  is 26.918136297728335
Average of  Moldova  is 8.672153295430638
Average of  Yemen  is 25.76840766445383
Average of  Chile  is 11.770133066906789
Average of  China  is 12.542541204592437
Average of  Malawi  is 21.34787202649801
Average of  Ireland  is 9.15390603284904
Average of  Lithuania  is 6.10594150347443
Average of  Montenegro  is 10.22104011370813
```

Figure 1: Output for question 1

This display continues for the 159 countries of the dataset. As you can notice when running the code, the execution is not sequential but parallel. Each thread work on x = 20 countries at once.

2.2.2. Showing the highest temperature with the name of the country and the date associated

Here are the different steps to solve question 2:

- **Importing the libraries**

We consider that the libraries we imported at the beginning are still available. But, if not we just reimport them. We are talking about the ***pandas*** library and the ***multiprocessing*** library.

Presenting the libraries used

Multiprocessing is a package that supports spawning processes using an API similar to the threading module. The multiprocessing package offers both local and remote concurrency, effectively side-stepping the Global Interpreter Lock by using subprocesses instead of threads. Due to this, the multiprocessing module allows the programmer to fully leverage multiple processors on a given machine (docs.python.org).

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language (pandas.pydata.org).

- **Creation of a sub dataframe containing only 4 columns**

We get the four columns AverageTemperature, Country, City and dt in a subset of the original dataset ***df*** that we called sub_dataset. Then, we create a new pandas series called max_c which contains each country (as index) and its corresponding maximum average temperature (as values).

- **Creation of the great list M**

The next block of instructions is, **by analogy to the creation of the L list in the previous code (2.1)**, meant to create a list ***M*** of 8 series containing each of them a series 20 countries and their corresponding maximum average temperature. The next statement is to initialize a ***global variable of type dataframe*** called df2 with the four needed columns entirely empty. Actually df2 will store immediately all the results.

- **External function: computemaxi()**

Then, we create the function ***compute_maxi()*** which will do all the treatment. The arguments of the function are ***l*** for the lock variable (that we have already explained the utility), ***dframe1*** and ***dframe2*** which are two dataframe arguments, and ***liste*** which is any list. As stated, df2 is a ***global variable*** so if we want to use it inside the function and modify its value we have to add the key name global. Then, in a for loop, we create progressively our dataframe with the expected 4 columns. Out of this for loop, we display now df2.

- ***The main function: main()***

Here the main function is almost the same than the previous one (2.1). The only difference here is the argument of the Process objects passed when we want to instantiate them with arguments. These arguments are lock (already explained), ***sub_dataset***, ***M[L.index(lst)]*** and ***lst***. The index of ***M*** is important and refers to the country corresponding exactly to the one in the ***liste lst***. Then, the rest remains unchanged.

Here is a partial screenshot of the output of this program (the order may change for another run):

	AverageTemperature	Country	City	dt
6127283	32.950	Afghanistan	Qandahar	1997-07-01
2127753	27.361	Albania	Durrës	1757-07-01
2195374	27.361	Albania	Elbasan	1757-07-01
7621892	27.361	Albania	Tirana	1757-07-01
8148236	39.651	Algeria	Warqla	1761-07-01
827518	28.085	Angola	Benguela	2013-03-01
4325574	28.085	Angola	Lobito	2013-03-01
4128814	29.509	Argentina	Lambaré	2009-11-01
2760223	24.958	Armenia	Gyumri	2006-08-01
7963321	24.958	Armenia	Vanadzor	2006-08-01
8394081	24.958	Armenia	Yerevan	2006-08-01
7732276	28.575	Australia	Townsville	1935-02-01
8020065	23.017	Austria	Vienna	1992-08-01
577857	27.000	Azerbaijan	Baku	2010-07-01
5138622	30.307	Bahamas	Nassau	1798-08-01
4583164	37.447	Bahrain	Manama	2012-07-01
6808986	32.815	Bangladesh	Satkhira	1957-05-01
2616187	24.278	Belarus	Gomel	2010-07-01
4239330	22.812	Belgium	Liège	2006-07-01
2042868	32.102	Benin	Djougou	1998-04-01
3621666	32.102	Benin	Kandi	1998-04-01
7471408	20.786	Bolivia	Tarija	2009-11-01
4960432	28.496	Bosnia And Herzegovina	Mostar	2003-08-01
2453556	26.985	Botswana	Gaborone	1992-02-01
4957127	30.495	Brazil	Mossoró	1998-02-01
5913481	26.901	Bulgaria	Pleven	2012-07-01
	AverageTemperature	Country	City	dt
3822277	21.634	Rwanda	Kigali	1997-09-01
1210389	38.049	Saudi Arabia	Buraydah	2012-07-01
3640282	31.823	Senegal	Kaolack	2005-05-01
5342430	25.715	Serbia	Novi Sad	1992-08-01
2395471	29.087	Sierra Leone	Freetown	2010-05-01
7028160	28.880	Singapore	Singapore	1998-05-01
1107463	25.181	Slovakia	Bratislava	1992-08-01
4318077	23.869	Slovenia	Ljubljana	2003-08-01
840382	34.137	Somalia	Berbera	2011-07-01
5843684	27.389	South Africa	Phalaborwa	1992-02-01
6850851	26.791	South Korea	Seoul	2013-08-01
1249367	30.659	Spain	Córdoba	1761-07-01
2640209	30.659	Spain	Granada	1761-07-01
3311841	30.659	Spain	Jaén	1761-07-01
7758432	32.075	Sri Lanka	Trincomalee	2012-06-01
3778208	35.700	Sudan	Khartoum	2012-06-01
7889914	35.700	Sudan	Umm Durman	2012-06-01
5692464	28.807	Suriname	Paramaribo	2003-10-01
4627572	27.221	Swaziland	Manzini	1987-02-01
4572943	20.922	Sweden	Malmö	1997-08-01
2541475	22.888	Switzerland	Geneva	2003-08-01

Figure 2: Output for question 2

The display continues for the 159 countries. It is very important to notice that our program is even able to detect the duplicate max AverageTemperature. We mean that even if for one country there are several maxima of AverageTemperature for different cities or different dates, the program will show all of the instances.

2.2.3. What we learned

- We learned how to use the parallel programming in Python. It was very interesting.
- We learned more about the manipulation of the dataframe object in an object oriented aspect.
- We learned about the use of global variable manipulated by both the external function and the main function
- We learned about the Lock class which is the equivalent of the mutex (Mutual Exclusion) in c++. This class with its two methods ***.acquire()*** and ***.release()*** **are very useful to "protect" a piece of Critical code in parallel programming**. We used it here to synchronize the display of our results, which was done in anarchy without using the lock variable.

2.2.4. Difficulties

- The first difficulty task was in question 1 about how to parallelize the code. Indeed, calculate the mean of global average temperature of each country was easily done directly by a piece of code. But when using the multiprocessing library it was a bit difficult for us. But we succeeded.
- The second difficulty was about how to separate the main function and the treatment functions in different files in the case of a parallel program because we did it in class but in one file.
- The third difficulty we faced, was to get the city, the name, and the date of the maximum temperature in a country. This was very hard because we used an intermediary dataframe named M, which was a list of series.

Conclusion

The two exercises of this project allowed us to really understand the concept of parallel programming. We deeply understand it in 2 different languages and tried also many combination in each of c++ and Python. Our personal remark is that the execution of the last program (in python) though it presents a bigger complexity, it runs very fast when we share the task using parallel computing. We hope to apply also these gained knowledge to a real HPC like Wascal's one.