Parallel analysis on NetCDF files

Brando Chiminelli e Alessio De Paoli

December 14, 2021

Contents

1 Introduction **2**

2 Problem analysis **3**
2.1 Serial code 3

3 Main steps **4**
3.1 Design of the parallel solution 4
3.2 Implementation 5
3.3 Benchmark on the HPC@UniTrento cluster 6
3.3.1 Speedup and Efficiency 7

4 Final discussion **9**

5 References **10**

1 Introduction

The project analyzed in this report is about the exploitation of **HPC** (High Performance Computing) infrastructure and parallel computing paradigms for the application of data science methods to big datasets about climate. In particular, our domain is the daily minimum near-surface (usually, 2 meters) air temperature (**tasmin variable**), measured on a Gaussian grid of the Earth. We handle both historical and future data (from 2015 to 2100) collected by the **EC-Earth-Consortium**, to have a better understanding of past, present, and future climate.

The datasets used in this project are taken from a shared folder in the HPC Cluster at the University of Trento (*/shares/HPC4DataScience/indices*).

The goal of the project is to compute the **average** of the tasmin variable (on a 160x320 latitude-longitude grid) of one or more years, inserted by the user on the terminal. To determine the average we perform a **Roll up** on the variable time that contains the values of 365 days, reducing the dimensions of our data from 3 (time, latitude, longitude) to 2 (latitude, longitude), as displayed on Figure 1.

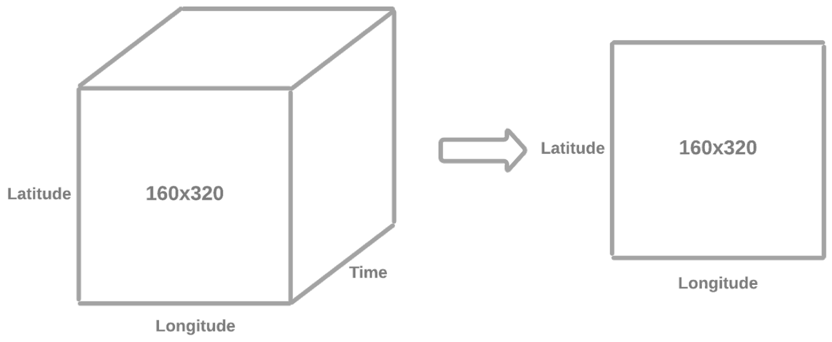


Figure 1: Roll up on Time dimension

The output is a file in the **NetCDF** (Network Common Data Form) format, that defines and describes the variables (latitude, longitude and tasmin) and their values. The file is then visualized with **Panoply**, a software developed by NASA, that plots our geo-referenced array of the average tasmin variable on a map scaled on different colors to produce a pleasant and easy to understand output.

The parallel approach we adopted is the following: we decided to parallelize the reading and analysis part, as they are the heaviest task on our program; the writing part is, instead, serial. For the parallelization we adopted the **hybrid parallelization** technique, applying both **MPI** and **OpenMP**.

We decided to work on this project because the topic of climate change is a big, worldwide and current challenge and continuous climate data analysis can provide insights to better understand and address this problem. Moreover, we were curious to work with the NetCDF format as it is the most commonly used in climatology, meteorology and oceanography applications (e.g., weather forecasting, climate change).

2 Problem analysis

Our starting point is a netCDF file, i.e. a file which allows the sharing of array-oriented scientific data. In our case, a file corresponds to a time span of one year and has 365 instances in which we will have the tasmin (air temperature) values in two dimensions: latitude and longitude (as displayed in Figure 2). Our goal, using these files, is to obtain the average of tasmin values in a number of years given as input. The first step is to analyse the problem serially.

2.1 Serial code

Firstly, we take the file we need to study as input and open it for reading. Then we create indices that allow us to cycle through the file (start, count).

Secondly, we start reading the file using the *nc_ get_ vara_float* primitive of the library netCDF, which allows us to obtain an instance. We save the data in an array that reflects all cells on the globe. After summing the tasmin values, day by day, we calculate the average of it in each cell of the matrix.

Finally we create a netCDF file, where we are going to insert our results, setting the dimensions and the variables. We use the *nc_ put_ vara_float* primitive which allows us to insert the matrix containing the averages into the file.

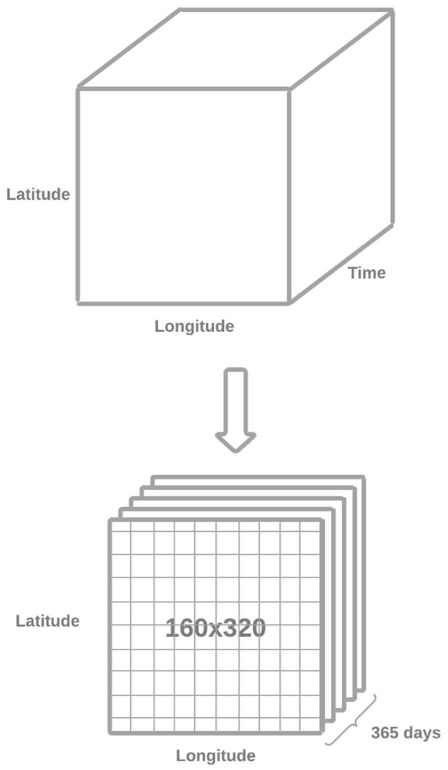


Figure 2: Serial analysis on a NetCDF file

3 Main steps

3.1 Design of the parallel solution

As mentioned in the introductory part (1), the approach adopted for the parallel solution is a **hybrid parallelization** which consists in using MPI and OpenMP jointly.

MPI processes are involved in the reading part, meaning that for a single file (which contains data for 365 days) each process retrieves the value of tasmin on the whole latitude-longitude grid for a pool of days. The days of one year are divided equally among the available processes (in Figure 3, two: p0 and p1), in this way each process can read in parallel a different section of the file.

On each day (displayed as a slice on Figure 3), then, the values of tasmin are summed up for every location on the grid, and this is done with the help of OpenMP that launches 4 threads for each MPI process.

OpenMP is also involved toward the end of the analysis part where the sum of the tasmin variable, coming from all the different processes (which is obtained with an *MPI Reduce*), is divided by 365 days for each location on the grid, getting in this way the average.

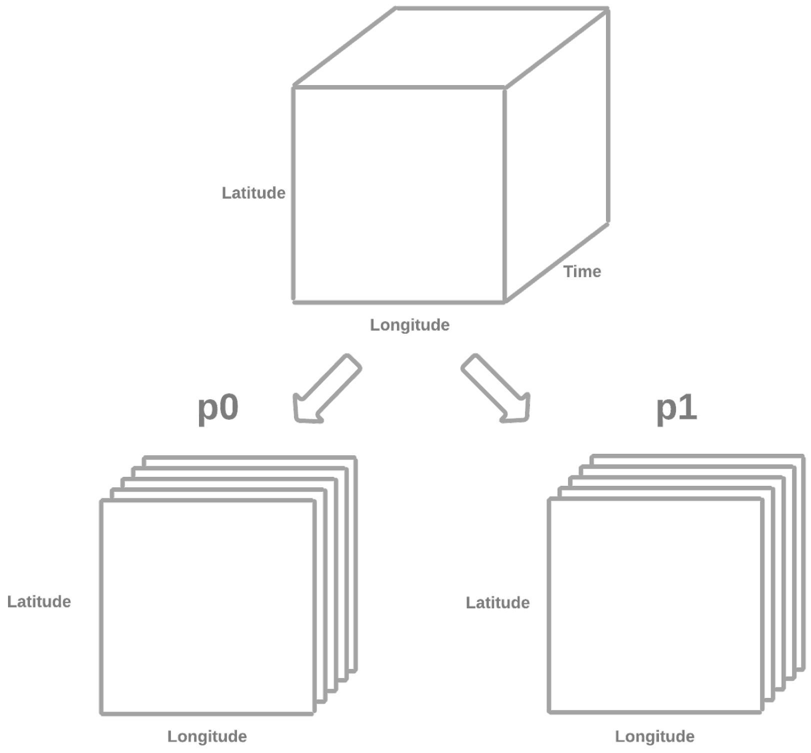


Figure 3: Parallel analysis on 2 MPI processes

3.2 Implementation

In order to understand the level of optimization that the parallel solution brings, it is necessary to work with more data than just one year, so we implemented in our code the possibility to deal with more than one file. In this way, we can measure the different performances in dealing with one year, two years and even four years and, by changing the number of cores involved, we can analyze how efficiently the work is spread among the available processes.

Firstly, the years are inserted manually in the terminal by the user, in this way we can build the corresponding strings to access the files in the shared folder of the cluster (*/shares/HPC4DataScience/indices*).

The program, written in C, can be divided into three main parts:

- The **reading** part
- The **analysis** part
- The **writing** part

In the **reading** part, the NetCDF variables, IDs and dimensions are declared in order to read the files and store the value of the variables. Then, MPI is initialized and the Communicator is set, from now on, the operations are done in parallel on each process.

With the primitive below, every process opens a netCDF file based on its rank, so as to split the reading among the processes (for example, if we allocate 8 cores and we want to read 2 files, we'll have 4 cores reading one file and the other 4 cores reading the second file).

```
nc_open(filenames[rank], NC_NOWRITE, &ncid_r)
```

In order for each process to read a subset of days (a fraction of a file), two arrays of size_t integers must be set: count and start, with the dimension equal to the number of variables (time, latitude and longitude, 3 in total). The **count** array tracks the number of elements that must be read: one for each value of time, 160 for the latitude variable and 320 for the latitude. In this way on each day, we retrieve the values of tasmin for a grid of 160x320 values. The **start** array tells to every process where to start reading the data: for the latitude-longitude grid, it reads from the position (0,0) on the grid; for the time dimension, it starts reading from the first value assigned to each process.

After having defined the two arrays, we need to iterate the time dimension with a for loop on every single process. Inside the loop, for every single day, we use the following primitive to retrieve the values of tasmin on the 160x320 latitude-longitude grid, and save them in the temp_in array.

```
nc_get_vara_float(ncid_r, temp_varid_r, start, count, &temp_in[0][0])
```

In the **analysis** part, all the values of tasmin for each position on the latitude-longitude grid are summed up on the temp_out array. Once all the processes have completed their sums and stored the values on their temp_out array, we need to sum all these partial contributions into one single array (the temp_sum) in order to get the average. We do that by exploiting a powerful MPI collective function, the following **MPI_Reduce**, which collects the different sums from each process on process 0.

```
MPI_Reduce(&temp_out, &temp_sum, 51200 , MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);
```

By dividing each value of the temp_sum array in 365*(number of years), we get the average. To optimize the time to carry out these operations, which are executed in a for loop, we make use of **OpenMP**, launching four threads in the following way.

```
#pragma omp parallel for num_threads(4) private(i, k)
```

We need to set the loop indexes as private to ensure that the threads will access only their private indexes.

The last part is the **writing** one. In this section we write a NetCDF file in a serial way by using only one process, which is the one with rank 0 as it contains the temp_sum array. After having defined the variables, dimensions and attributes of the NetCDF file to write as output, we use the following primitive to insert the array with the averages.

```
nc_put_vara_float(ncid, temp_varid, start, count, &temp_sum[0][0])
```

Finally, MPI is finalized, and everything that has been allocated is cleaned up.

3.3 Benchmark on the HPC@UniTrento cluster

In the following table we summarise where we have tested the performance of our code. We can see that in the serial case there are submissions with only one node, while in the parallel case we assume that a year must have at least one MPI_process analysing it.

	SERIAL			**PARALLEL**		
	1Y	***2Y***	***4Y***	***1Y***	***2Y***	***4Y***
1p				NO	NO	NO
2p	NO	NO	NO			NO
4p	NO	NO	NO			
8p	NO	NO	NO			
16p	NO	NO	NO			

In order to obtain the performance, we made 7 submissions for each case. We then proceeded to remove the maximum and minimum value and at the end average the resulting 5 items.

The time values to measure the different performance are taken using the following primitive, that retrieves the time in microseconds before the reading part and after the analysis part.

```
gettimeofday(&now, NULL);
```

3.3.1 Speedup and Efficiency

Speedup is a number that measures the relative performance of two systems processing the same problem, in this case it measures the difference between the serial solution and the parallel approach.

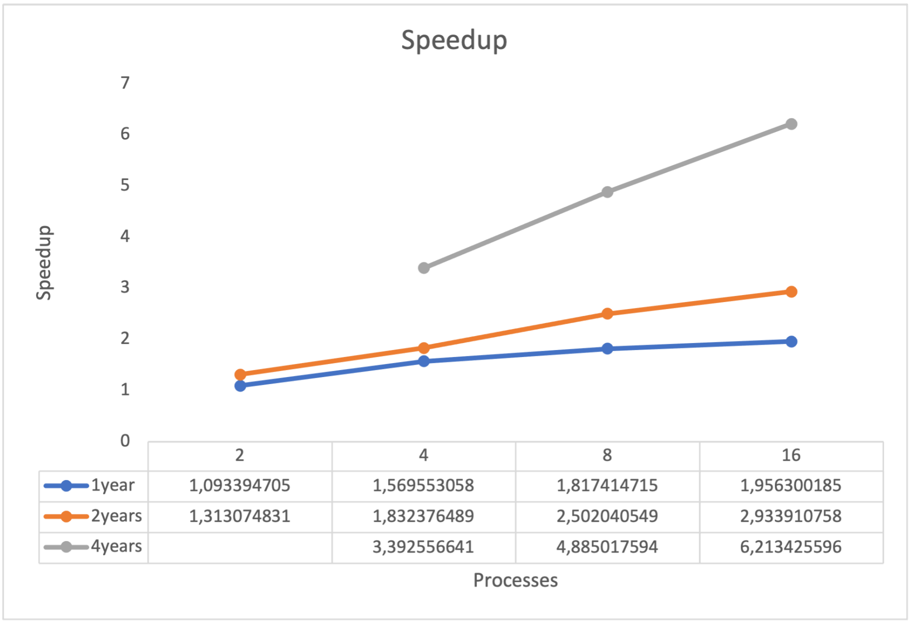
We define three variables:

- p = Number of cores
- T_{serial} = Serial run-time
- $T_{parallel}$ = Parallel run-time

The formula for obtaining the speedup is:

$$Speedup = \frac{T_{serial}}{T_{parallel}}$$

As we can see in Figure 4, the speedups of the three cases under consideration grow as the number of MPI_processes increases, because the parallel execution time decreases. Each process will have fewer days to analyse and will consequently be faster.



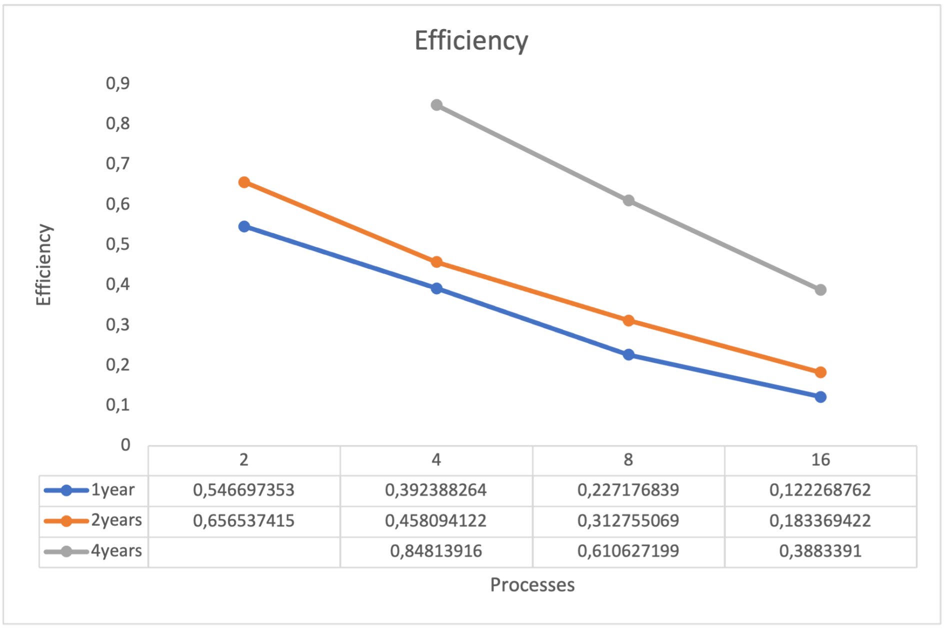
	2	4	8	16
1year	1,093394705	1,569553058	1,817414715	1,956300185
2years	1,313074831	1,832376489	2,502040549	2,933910758
4years		3,392556641	4,885017594	6,213425596

Figure 4: Speedup on the performance of parallel solution

Efficiency is the ratio of useful work to resources expended, so it is the proportion between the speedup and the number of MPI_processes used.

$$Efficiency = \frac{Speedup}{p} = \frac{T_{serial}}{p * T_{parallel}}$$

As displayed in Figure 5, we see that efficiency drops as the number of processes increases. From this we can deduce that it is not enough to increase the number of processes to obtain a good result, but we must also balance the number of resources used.



	2	4	8	16
1year	0,546697353	0,392388264	0,227176839	0,122268762
2years	0,656537415	0,458094122	0,312755069	0,183369422
4years		0,84813916	0,610627199	0,3883391

Figure 5: Efficiency of the parallel approach

4 Final discussion

In chapter 3.3.1, we presented two graphs: one that displayed the speedup and the second the efficiency of our parallel approach. About the **speedup** (Figure 4) we can say that it increases with the increase in the number of processes involved, as expected. However, we can see that there's little improvement with one and two years, but already with four year the improvement grows. We can than estimate that we'll see better speedups if we deal with even more than four years.

In the **efficiency** graph (Figure 5) it is very clear that with 4 processes and 4 years we have the highest efficiency, almost 85%. This result means that this particular setting for the parallel approach is 3.4 times faster than the serial one (if it were 4 times faster we would have the perfect optimization).

As anticipated in chapter 1, the NetCDF output file is read by **Panoply** to produce a map scaled on different colors. The map, displayed in Figure 6, uses a Kelvin scale for the tasmin variable and is based on the Gaussian grid (160x320).

By comparing the final map of different years we can produce reports on the change in temperature over time on different locations and understand future climate response to changes in human-induced factors such as greenhouse gases (as water vapor, carbon dioxide and methane).

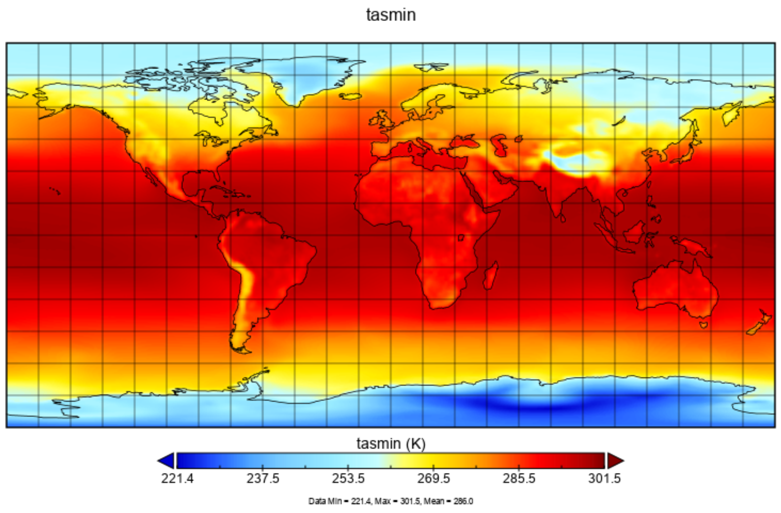


Figure 6: Example of a panoply map (year 2057)

5 References

1. NetCDF Documentation

   https://docs.unidata.ucar.edu/netcdf-c/current/index.html

2. Writing a NetCDF file example in C

   https://docs.unidata.ucar.edu/netcdf-c/current/pres__temp__4D__wr_8c_source.html

3. Reading a NetCDF file exapmle in C

   https://docs.unidata.ucar.edu/netcdf-c/current/pres__temp__4D__rd_8c_source.html