Analysis, refactoring and optimization of the Multiscale Online Nonhydrostatic AtmospheRe CHemistry model

Joan Bernat Gonzalez

Summary– This project studies the performance of the Multiscale Online Nonhydrostatic AtmospheRe CHemistry model (MONARCH). This model is one of the multiple Earth System Models (ESM) developed at Barcelona Supercomputing Center (BSC), and used daily by a large number of users. It has a large computational cost due to the multiscale nature of the physical processes, so it is of vital importance to maximize its performance. To achieve this, different BSC tools are used throughout the project: autosubmit, extrae and paraver. MONARCH runs in a parallel environment like the EuroHPC supercomputer Marenostrum5 (MN5), and utilizes the Message Passing Interface (MPI) to perform the communications between the processes. Performance issues related to MPI communications or the model itself are detected and improved in multiple parts of the code.

Keywords– MONARCH, ESM, HPC, parallel, performance, MPI, extrae, paraver, communications, chemistry, traces

1 Introduction

CUTTING edge expensive numerical simulations are required across various sectors to solve complex problems in our society, such as predicting air quality and climate change. High Performance Computing (HPC) utilizes supercomputers to run these massively parallelized codes [1], but even with the most modern machines, we are unable to fully resolve complex meteorological systems. Therefore, to make the most of these huge and expensive machines, it is essential to optimize the applications' performance.

The application optimized during this project is the MONARCH [3] model, which is a chemical weather prediction system. Within the world of meteorology these models are of vital importance, MONARCH simulates air quality and the concentration of pollutants, which can help by taking measures to protect human health and the environment.

This model is used among different projects at the European level, one of which is at the Earth Sciences Department of the BSC. This department focuses on the research, development and implementation of environmental models and forecasts using models based on numerical processes.

In this work we improve the performance of the various MONARCH components, utilizing one of the most powerful supercomputers in Europe, MareNostrum5 (MN5), which is located at BSC and designed to run massively parallel codes, and store and manage large quantities of data.

The project begins with an overview of the state of the art and a clear definition of the project's objectives. This is followed by a detailed explanation of the methodology used throughout the work. Next, a comprehensive section is dedicated to model performance analysis, starting with a general overview and then delving into specific topics identified during the general evaluation. The main improvements made to the model, which significantly improve the scalability of various model versions, are then summarized. Finally, the project concludes with a discussion of future work and a summary of the key conclusions.

2 State of the Art

ESMs attempt to represent the interactions between different components of the Earth system: the atmosphere, the ocean, the biosphere, sea ice, the land surface, and the ice sheets. A large part of the computational cost of these models comes from the atmosphere component, which solves the physical, chemical, and radiative processes with computational fluid dynamics (CDF) being utilized to model turbulence [4]. Numerical modeling is needed to fully describe the atmosphere, which has a high computational cost, as there are no analytical solutions.

• Contact E-mail: joan.bernat@autonoma.cat, joan.bernat@bsc.es
• Project supervised by: Hannah Elizabeth Ross and Mario Acosta (Barcelona Supercomputing Center)
• Project tutored by: Eduardo Cesar Galobardes (Architecture and Computer Technology Area)
• Course 2024/25

Due to the large computational complexity of atmosphere models, they must run on supercomputers that have distributed memory. Therefore, they are usually parallelized using a domain decomposition strategy, using Message Passing Interface (MPI) for performing the communications among processes, since the systems on which MONARCH runs have distributed memory. Each of the processes solves a part of the domain, dividing it into a network of processes that exchange information between one another during execution.
Recently, for many numerical codes, porting to GPUs has been a common way to achieve a performance increase, there is a GPU implementation of MONARCH in development (Chemistry across multiple phases CAMP). However, there are still many supercomputers with CPU architecture, so there are many users who will keep using the CPU implementation. That is why it remains important to run efficiently on them too, in addition to taking advantage of newer GPU resources.
Among the different components of atmospheric models, chemistry usually stands out in computational complexity, taking up to a 90% of the total computation time in some models [5]. This part of the model within MONARCH is the one that has been implemented at BSC, and will be the focus of this work.
MONARCH is a chemical climate prediction system that can run either globally or regionally. This model is based on the online coupling of the Non-hydrostatic Multiscale Model on the B-grid (NMMB) meteorological model with a complete aerosol chemistry module.
MONARCH contributes to several prediction activities, providing daily predictions produced at BSC to the Barcelona Dust Regional Center[7], air quality predictions to the multi-model system of the EU Copernicus program from the Copernicus European Air quality service [8], and Global daily prediction of aerosols for the International Cooperative for Aerosol Prediction [9].
MONARCH contains several modules, which have already been parallelized, such as the chemical module. In this work, the optimization will focus on the various modules developed at BSC.

3 OBJECTIVES

The main objective of this project is to optimize and improve the performance of the different modules of MONARCH, allowing the users to run more simulations in a shorter time and also waste fewer resources. To achieve this, some other objectives have to be fulfilled:

- Adapt to the BSC work environment and learn how to use the highly specialized tools provided.
- Gain foundational knowledge about ESM and HPC environments.
- Understand the different MPI communications between the processes, discover where the bottlenecks are and try to reduce their impact.
- Share knowledge gained with colleagues working on MONARCH to help them to run more efficiently.

4 WORK METHODOLOGY

MONARCH is mainly developed in Fortran, a language compatible with MPI and other parallelization tools, and also commonly used for meteorological codebases. The model contains many modules, all of them parallelized. Even though there are different parallel programming models, in this work the focus is on the optimization on CPUs using MPI. MONARCH also can utilize OpenMP, but since it has to run with shared memory, MPI is more likely to improve the performance of the model on distributed memory.[6]
The tests done for this work will be executed on MN5, using a workflow management system developed entirely by BSC, autosubmit [10], which facilitates all the work of submitting jobs to the supercomputer, as well as the management of the various experiments to be performed.
To perform the profiling and analysis of the model, the Extrae and Paraver [11] are the main tools used. These tools are designed for the analysis of parallel execution processes, and provide a complete report of the model executions, especially of the MPI communications between the various nodes. Extrae is used to generate traces of all the events that occur in a Fortran or C executable file. Paraver is the interface used to visualize and analyze the traces generated by Extrae. Among the extrae and paraver functionalities that can be used to analyze program performance, three will be mainly used:

- ***MPI Calls:*** Collects the *MPI Calls* that are carried out by the processes during the execution of the program, as well as how much time is spent inside them.
- ***MPI Callers:*** Collects from which subroutine or function these *MPI Calls* are executed for every process.
- ***User Functions:*** Allows the user to specify which subroutines to follow during execution and generates a trace that tells at what point the various processes go through the specified subroutines.

The processes that are running MONARCH are divided into forecast and quilting tasks. This is a commonly used strategy in these types of models where some tasks are completely dedicated to perform all the I/O operations of the model while the rest perform the computation.
For studying the performance of MONARCH, first, a general analysis to locate parts of the model that can be optimized further is done, followed by an in-depth investigation of these possible performance issues. The traces generated show the behavior of the processes (vertical axis) during the simulation (horizontal axis). These traces are zoomed in to see the more interesting parts, also histograms showing the general behavior of the processes are shown during the project.

5 MONARCH PERFORMANCE ANALYSIS

As mentioned above, MONARCH contains several components, such as the chemistry and radiation modules. In this case, the focus of the work is the chemistry module, which can significantly impact the performance of

MONARCH. Initially, the model does the spin-up, which is similar to dust-only simulations as it does not include chemistry, therefore having a lower computational cost. Once the spin-up is done, the model starts computing the chemistry. Model's performance can vary depending on the domain conditions and the resources used, in this case, the model runs in a regional domain and with an intermediate number of processors. The configurations selected are those regularly used by domain scientists to test the model.

5.1 General analysis

Figure 1 shows an example of a complete trace for the chemistry configuration. To find out what color the MPI calls are, the legend can be seen on the right side of the image.

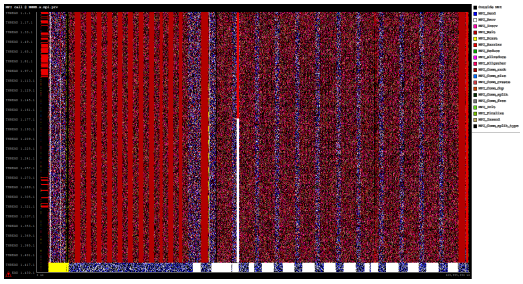

Fig. 1: Complete trace of CHEM regional model configuration.

It is important to note that this is a 24-hour simulation, which can be identified due to the horizontal red stripes. It can also be seen that the last threads behave differently from the rest, this is because they correspond to the threads (blue and white color based on the plot) that carry out all the writing of the model output files, while the rest carry out the computational work.

The first thing that draws attention just by observing this trace is that the first 40% of the model execution works differently from the rest. This part corresponds to the first 12 hours simulated, which are the chemistry spin-up hours. Marked mainly by the uniform red bands on the computing threads that indicate that these processes called *MPI_Wait*, that is, they are waiting for some communication with other threads to finish. All this time, the threads are completely stopped, which has a big impact on the performance of the model.

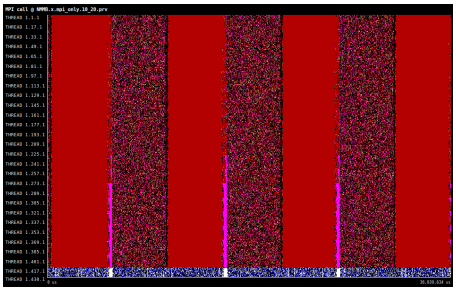

Fig. 2: CHEM regional model trace between the 10-20% of the execution time, which includes hours 2-6.

Figure 2 shows a cut of the previously mentioned part. We see how all the computing processes are waiting while the only ones working are the threads that write the model output. In addition, if we count the red bands in Figure 1 at the beginning, there are exactly 12, which coincides with the number of hours that are run without chemistry. It should also be noted that the write tasks write the output every simulated hour, saving various aspects of the simulation. Taking all this into account, the writing processes can slow down some parts of the simulations, this will be further studied in section 5.2. Inside the same trace, if we zoom in, we can see an individual timestep in more detail, the figure 3 shows a single chemistry timestep.

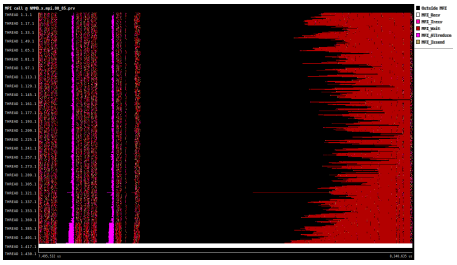

Fig. 3: Complete timestep of the chemical part of CHEM Regional model.

All the time steps of the model have a very similar behavior, here we can see factors that may be affecting the overall performance of the model. At the beginning two *MPI_AllReduce* (pink) occur, in which the last processes seem to be arriving much earlier than the others, as can be better seen in the figure 4.

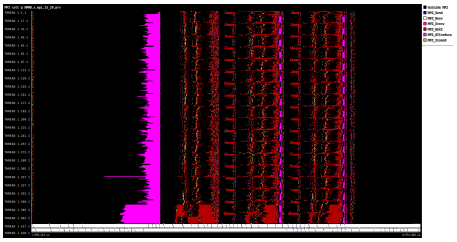

Fig. 4: Trace from figure 3 zoomed in.

This behavior, which is observed at all model time steps, is peculiar because the change in the work performed does not occur gradually, but rather only a block of the last processes seem to have less work.

The behavior of the processes at the end of the timestep also seems important, it can be seen at the end of the figure 3 that after performing all the necessary computations within the timestep, they arrive very irregularly at the end of it, which causes some processes to be waiting for the slower processes for up to 0.2 seconds. The two factors mentioned may be slowing down the performance of the model, this will be studied in the section 5.3.

Extrae also provides a view called MPI Profile, which not only shows the when the MPI Calls were made, but also all the time spent inside them. In figure 5 you can see the MPI Profile corresponding to the same trace in figure 1.

This trace provides new information, firstly, during all time steps the computational processes are sending information in a non-blocking manner with *MPI_Isend*, we can also see how at the end of each hour in the part of

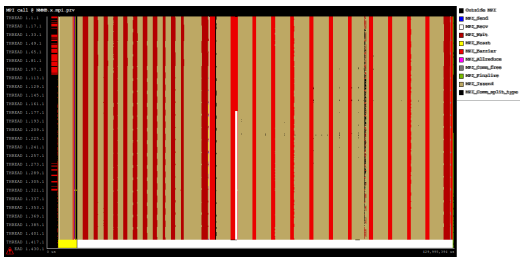

Fig. 5: MPI Profile from figure 1

the model with chemistry (the last 12 hours) the computational processes call *MPI_Barrier*, possibly also losing computational time due to waiting for other processes. This is not seen in the trace of figure 1, which is why it is important to use different metrics, since it can provide us with information that we do not see with the MPI Calls, this will be done in the section 5.4.

5.2 Writing processes

To confirm if the processes dedicated to the writing are slowing down the simulations, several strategies can be followed, in this case the performance of the model is measured, both the total time and the time dedicated to writing, by increasing only the number of writing threads, so we can see the impact it has on the performance, the metric used for measuring the impact is the speedup, given by:

$$Speedup = \frac{\text{Old execution time}}{\text{New execution time}}$$

In figure 6 we can see how the speedup of the model evolves taking into account the total execution time. In this graph there are two lines, the blue one corresponds to the speedup of the model simulating 24 hours (12 without chemistry and 12 with chemistry) and the orange one corresponds to the model simulating the first 12 hours (only the part without chemistry). In small text below each point you can see the computation time of each of the models in seconds.

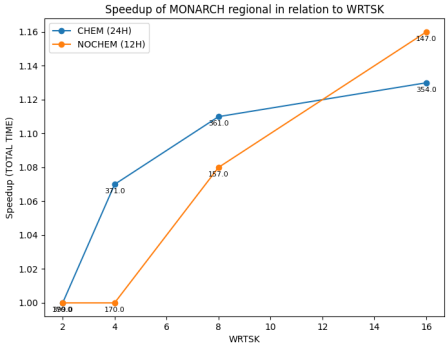

Fig. 6: Total time speedup for the model with and without chemistry increasing the number of writing tasks.

The graphs in figures 6 show that there is a non-negligible speedup in both configurations, but it is not very significant, not reaching much more than 1.1 by increasing the tasks from 2 to 16. This leads us to think that the problem is likely somewhere else, we will analyze how the writing processes work and how they carry out their work.

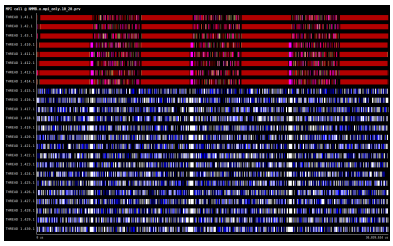

Fig. 7: Trace between hours 2-6 of simulation where writing threads are active (blue) and a few computing threads (red).

In figure 7 you can see the clear difference between the computation (red) and writing tasks (blue), while the computation tasks carry out all kinds of MPI communications, the writing tasks are all the time sending and receiving information in a blocking manner. If we look at all the writing tasks, we can see a slightly different behavior for the first task, to further see how they behave, we will analyze the time they use for each of the communications.

In table 1 we can see the percentage of time spent in each of the MPI calls for the same processes shown in figure 7. We can see that, as was already seen in the calls, the computing processes spend most of the time inside *MPI_Wait*, an average of **64%**, which means a large drop in performance. For the writing processes, on the other hand, a behavior is seen that was not seen in the calls themselves, we can see how most of these spend **97%** of the time waiting to receive information inside *MPI_Recv*, while the first behaves completely differently, being most of the time outside MPI and only **4%** of the time receiving information from the simulation.

This behavior leads us to think that the first process acts as the master of the writing processes, writing itself the output file, while the other remain waiting for a lot of time inside *MPI_Recv* waiting for some information from the computing tasks.

	Outside MPI	MPI_Send	MPI_Recv	MPI_Irecv	MPI_Wait
THREAD 1.41.1	35.10 %	-	-	0.08 %	61.88 %
THREAD 1.42.1	35.32 %	-	-	0.09 %	61.65 %
THREAD 1.43.1	35.08 %	-	-	0.08 %	61.85 %
THREAD 1.410.1	27.62 %	-	-	0.06 %	64.69 %
THREAD 1.411.1	27.52 %	-	-	0.06 %	64.83 %
THREAD 1.412.1	27.42 %	-	-	0.06 %	64.87 %
THREAD 1.413.1	27.55 %	-	-	0.06 %	64.77 %
THREAD 1.414.1	28.78 %	-	-	0.08 %	64.25 %
THREAD 1.415.1	94.78 %	0.57 %	4.65 %	-	-
THREAD 1.416.1	4.47 %	0.19 %	95.33 %	-	-
THREAD 1.417.1	2.62 %	0.13 %	97.25 %	-	-
THREAD 1.418.1	2.60 %	0.14 %	97.26 %	-	-
THREAD 1.419.1	2.60 %	0.13 %	97.27 %	-	-
THREAD 1.420.1	2.60 %	0.13 %	97.26 %	-	-
THREAD 1.421.1	2.60 %	0.13 %	97.26 %	-	-
THREAD 1.422.1	2.61 %	0.14 %	97.25 %	-	-
THREAD 1.423.1	2.62 %	0.13 %	97.25 %	-	-
THREAD 1.424.1	2.61 %	0.14 %	97.25 %	-	-
THREAD 1.425.1	2.61 %	0.14 %	97.26 %	-	-
THREAD 1.426.1	2.70 %	0.14 %	97.16 %	-	-
THREAD 1.427.1	2.55 %	0.14 %	97.32 %	-	-
THREAD 1.428.1	2.53 %	0.14 %	97.33 %	-	-
THREAD 1.429.1	2.54 %	0.14 %	97.32 %	-	-
THREAD 1.430.1	1.48 %	0.13 %	98.40 %	-	-

Table 1: Percentage of time within each MPI Call by the writing threads and some computing threads between hours 2–6 of simulation.

	MPI_Recv	MPI_Send
THREAD 1.415.1	244,554	244,258
THREAD 1.416.1	16,604	16,283
THREAD 1.417.1	16,468	16,283
THREAD 1.418.1	16,467	16,284
THREAD 1.419.1	16,467	16,284
THREAD 1.420.1	16,468	16,284
THREAD 1.421.1	16,468	16,284
THREAD 1.422.1	16,467	16,284
THREAD 1.423.1	16,468	16,284
THREAD 1.424.1	16,468	16,284
THREAD 1.425.1	16,468	16,284
THREAD 1.426.1	16,467	16,284
THREAD 1.427.1	16,463	16,284
THREAD 1.428.1	16,463	16,284
THREAD 1.429.1	16,463	16,284
THREAD 1.430.1	16,468	16,284

Table 2: Number of *MPI_Send* and *MPI_Recv* calls from the writing processes in the 10-20 segment.

In table 2 we can see the number of times each of the writing threads calls *MPI_Send* and *MPI_Recv*, the first thread is calling them much more than the rest, in fact, the

number of calls of the first thread is the sum of all the rest. All this indicates that the first one effectively acts as *master*, performing all the writing to output files of the information sent by all the writing tasks. Since these threads are sending a lot of information, it is difficult for a single process to deal with all the communications and writing.

The model already has the option to change the number of I/O servers implemented, splitting the writing tasks in various MPI communicators, each with their respective master threads doing the MPI communications and writing. We tried increasing the number of nodes to see the traces and how the speedup of the model evolves.

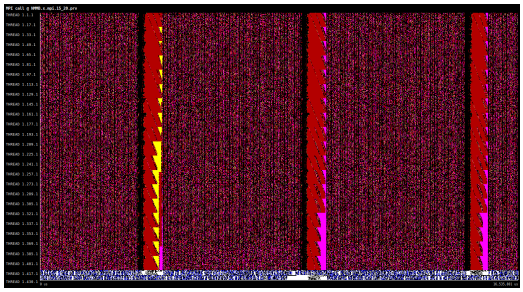

Fig. 8: Regional CHEM model trace between 15-20% execution time with 2 I/O servers, spanning simulated hours 2-5.

The trace of figure 8 corresponds to a simulation run with 2 I/O servers rather than 1, it is shown that the *MPI_Wait* at the end of each hour is significantly reduced, therefore, the computing threads are waiting significantly less time. With this straightforward change, it seems that the performance of the program has increased considerably with 2 I/O servers, and to further confirm this we can see the speedup in figure 9.

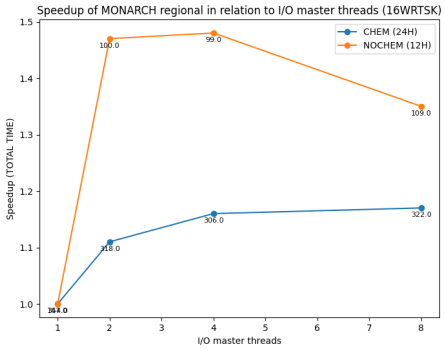

Fig. 9: SpeedUp of the total execution time of the model with and without chemistry by increasing the number of I/O servers.

The graph in figure 9 already shows a high speedup, of almost **1.6x**, for the part without chemistry, with configurations of 2 and 4 I/O servers, but it should also be noted that if we increase the number of I/O servers a lot then there will not be as many processes doing the communications to the master writing process, which can lower the performance, this is seen with 8 I/O servers. For this reason, a balance must be sought between processes performing the simulation, the processes receiving information from the computation tasks and the processes writing the output files.

On the other hand, it also shows that this bottleneck does not occur in the part with chemistry, since the speedup is not as high when both parts are taken into account, this is because when chemistry occurs the time steps carried out by the computing threads are longer, so the writing tasks have more time for finishing their work, therefore with a single I/O server is enough to perform all the writing.

This method, however, has a disadvantage due to the strategy that the model follows for writing when the writing threads are separated into several I/O servers. What the model does is making the different I/O servers turn the write for the simulated hours of the model, so each master thread from the different I/O servers creates a different output file of *netcdf* [12], where they write the results of each hour in turns. That is, if there are two I/O servers, two output files are generated, where the first will contain the results of the hours written by the first master thread, which will be 0, 2, 4, 6..., and the second file will contain the results of the hours written by the second, in this case 1, 3, 5, 7...

This can lead to extra post-processing work since the output expected by the model is just one netcdf file, so the different files generated should be merged after the simulation, which takes some more time. This means that in models with chemistry, where only the first 12 hours are optimized for the change, it may not be cost-effective to use more I/O servers.

It might be said that with this strategy the performance of the writing of the model is not improved, what happens is that every I/O server has more time for performing the write of a simulated hour, since it has to write fewer simulated hours. This allows the computing tasks to keep on with the simulation while some of the I/O servers are writing, since they don't have to wait for the writing to finish at every simulated hour. Some ideas of how to manage writing in a future implementation are detailed in section 7.

There are cases where this change will have a greater impact, since they do not use chemistry, and timesteps are shorter, such as the global and regional DUST simulations, which some users from MONARCH also require. We will perform the same experiment in these cases to observe the impact and assess the results.

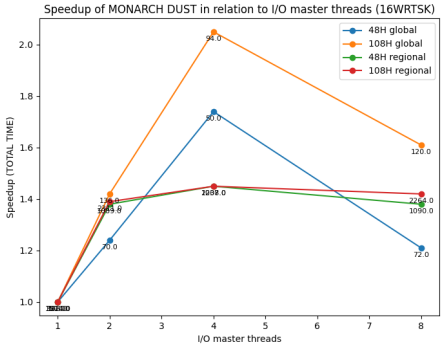

Fig. 10: SpeedUp of the total execution time of the DUST model in regional and global domains by increasing the number of writing tasks.

Figure 10 shows the speedup in the DUST model with a global and a regional domain in two different simulations, one of 108 hours and another of 48 hours. Here we see

that for the global domain the improvement is impressive for both cases, arriving to **2.00x** speedup with 4 I/O servers for the simulation of 108 hours and to nearly **1.8x** speedup for the 48 hours simulation.
However, for the regional domain the improvement isn't as significant, this may be due to the fact that the model time steps take more computing time in this case, so the computing tasks do not longer wait for the writing to finish. This may be due to several factors, such as the domain resolution being much higher, for this reason the scalability of the model with regional domain will be measured with different numbers of I/O servers. This will show if changing the number of I/O servers affects this simulation.

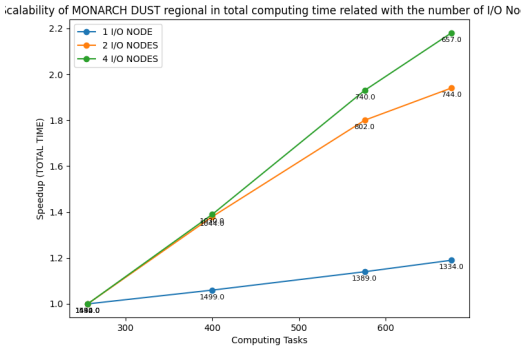

Fig. 11: Scalability of the Regional DUST model by varying the number of processes in relation to the I/O servers.

Figure 11 clearly shows how increasing the number of I/O servers helps improving the scalability of the model for this configuration. With this information, we can conclude that, although the post-processing may take some extra time, it is worth increasing the number of I/O servers for configurations in which the time steps are not very long, for example, in the case of DUST simulations.

5.3 CHEM time steps irregularity

In Section 5.1 reference is made to a periodic behavior in the time steps of the simulation that may be slowing down the simulation, (see figures 3 and 4). We first repeated the experiment to see if the behavior of the processes changed, displaying a similar but not identical behavior, with the same distribution of timings for the threads. This indicates that the problem is not related to the *hardware*, as it is extremely unlikely the same threads would be assigned the same nodes again. Therefore, we will study the model to see where this inequality may come from.

To locate where these calls are coming we will use the extrae *callers*, described at section 4.

Figure 12 shows the *callers* of figures 3 and 4, which indicate the functions from where the calls of MPI are happening, which is why the shapes of the trace are so similar disregarding the colors. The first *MPI_Wait* in figure 12 is located in the *exch3* subroutine in the *exchange* module.
This module contains all the subroutines needed to perform the exchanges in any dimension of the boundaries between parts of the grid stored in different MPI processes. Each

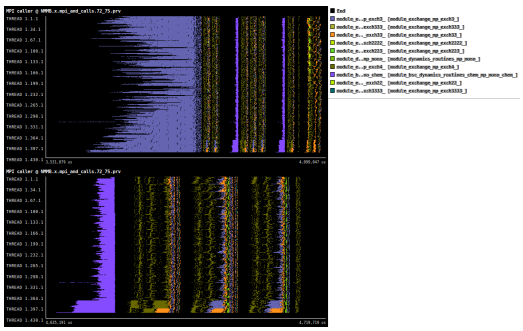

Fig. 12: Figure 3 and 4 callers.

of the processes simulates a region of the grid, and needs information from the boundaries of the neighboring regions in order to perform the calculations, which must be updated with the information from the processes that compute these cells every timestep.
After investigating this module, we concluded that it is already quite optimized, and therefore the way of performing the communications is not the problem. However, since there is an *MPI_Wait*, this is where the processes are synchronized from the previous work, and therefore where they will have to wait for those that take longer to perform the calculation.
From this we can conclude that the real problem is the difference in computation prior to the exchange, since if the computation time was balanced, the processes would not have to wait so long for the slowest ones. To see how the simulation is affected by the inequality between the processes, we can use a histogram of the time of each of the processes within the various *calls* of MPI.

	Outside MPI	MPI_Wait	MPI_Recv	MPI_Barrier	MPI_Allreduce	MPI_Issend	MPI_Irecv	MPI_Bcast	MPI_Send
Num. Calls	414	414	414	414	414	414	414	414	414
Total	22,691.19 %	8,072.34 %	5,016.64 %	4,476.93 %	623.11 %	257.38 %	153.50 %	71.91 %	36.30 %
Average	54.81 %	19.50 %	12.12 %	10.81 %	1.51 %	0.62 %	0.37 %	0.17 %	0.09 %
Maximum	64.60 %	38.00 %	21.76 %	20.96 %	4.46 %	0.86 %	0.41 %	0.43 %	8.68 %
Minimum	33.47 %	10.09 %	2.03 %	0.98 %	0.44 %	0.25 %	0.19 %	0.00 %	0.02 %
StDev	4.08 %	3.77 %	5.82 %	5.83 %	0.71 %	0.12 %	0.04 %	0.12 %	0.44 %
Avg/Max	0.85	0.51	0.56	0.52	0.34	0.72	0.90	0.41	0.01

Table 3: Percentage of time within each of the *MPI calls* for the twelve hours with chemistry in the model.

From the histogram in the table 3 we can extract very important metrics, for example, by looking at the average time of the processes outside of MPI we can determine the parallel efficiency of the program, since all the time that is inside MPI calls is time lost in communications, in which no computation is performed. In our case, it is **55%**, which indicates that there is quite a bit of room for improvement, since, in an ideal case in which there were no communications, the speedup would be approximately **1.81x**.

From here, we can also extract the load imbalance of the program, shown by the Avg/Max metric outside of MPI, which is basically the division of the average between the maximum of all processes. In an ideal case it would be 1.00, since we want that all the processes to spend the maximum time outside MPI, while ours is **0.85**, which indicates that there is some imbalance between the processes, as previously stated.

Having analyzed how this inequality affects the simu-

lation, we will move on to see which functions are called in this section. This will help us pinpoint the issue and determine if it is worth working on improving process load balancing. This is where we will use the *user functions* mentioned in 4 section.

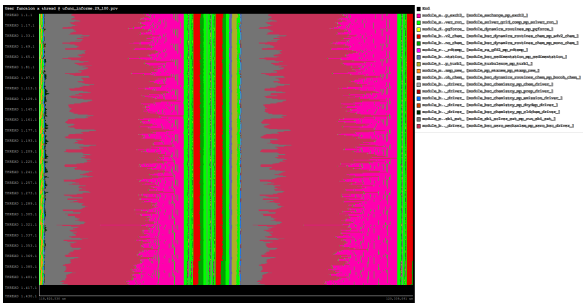

Fig. 13: User functions for two chemistry timesteps.

Figure 13 shows the *user functions* for two timesteps of the model with chemistry, we can easily see the pattern of functions that follow the time steps with chemistry and appreciate the similarity in terms of shape with the figures 3 and 4.
We can see the shapes of the functions where the *MPI Calls* are performed, as well as the functions prior to these, among the user functions shown, the most notable ones for their computational time at a glance and the ones that seem to be more load imbalanced are *aero_bsc_driver*, which contains the drivers of chemical mechanisms of aerosols, and *run_ebi_ext*, which calls the EBI solver, the main function that solves chemistry. Since this trace does not provide concrete numbers about how the processes behave in the functions, we can extract a histogram on the section of the trace shown on figure 20, shown in the section A, with several interesting measures.

Due to the size of this histogram, it can be seen in the appendix A, figure 21 confirms that the functions *aero_bsc_driver* and *run_ebi* are among those that take the most computing time, with an average of **33.63%** and **12.56%**. We can also see *solver_run*, which corresponds to the main interface, and *exch3*, which is where most of the synchronization of the processes is performed.

Domain scientists have confirmed that the difference between the time consumed in these solving functions is likely due to the different conditions within the cells, whether it is the position on the grid or the different physical quantities (i.e. pressure, temperature etc) that can affect the solution time of the aerosols and the chemistry, all of which can cause some cells to be more expensive and require more iterations to converge on a solution. In the meantime, there is no solution that removes time from the most expensive cells in a simple way without affecting to the model solution, so we must try to deal with these in the most optimal way possible.

With this information, we can propose a solution involving the implementation of a method to balance the load between the processes. A good solution for this type of problem is implementing some kind of process planification algorithm.
There are several algorithms for this problem, we suggest:

- Round Robin: Has a queue of MPI ranks and assigns cells to the first process on the queue, if the first process hasn't finished the computation of its cell, it is just sent to the end of the queue and goes to the next one.
- Knapsack: A more complex algorithm that predicts the cost of each cell based on the initial data or the previous timestep, then distributes the cells from the most to the least expensive to different MPI ranks. It could give better results than *round robin*, but it is strongly dependent on how good the cell prediction is.

These strategies often lead to neighboring cells being distributed to different nodes in the machine, which results in more expensive MPI communications between them due to them not having shared memory. This can decrease the performance in a lot of models, but in this case, there are communications only at the start and end of the timesteps, so it is not an issue.
Both strategies have a drawback in practice, due to the great complexity of the model they would be very difficult to implement, since the way the model cells are distributed would have to be completely changed, and this is beyond the scope of this work. Therefore, to determine if this implementation is really worthwhile, we estimate the maximum performance improvement achievable here.
To achieve this, we will use the previous *user functions* and *callers*, since with these we have detected which functions the inequality comes from, as well as the functions that deal with this inequality by containing the synchronizing *MPI Calls*. We will look more closely at the functions that deal with the inequality of the processes, since they can be a good measure of how this affects performance, specifically, the functions corresponding to the *exch* module mentioned above, since these do not perform any computation, so almost all the time they occupy is due to the *MPI Calls*.

	module_s...ver_run_	module_e...p_exch3_	module_e...p_exch4_	module_w...ite_run_	module_e...exch33_	module_e...xch3333_
Num. Calls	414	414	**414**	415	414	414
Total	29,802.95 %	7,421.95 %	2,603.38 %	819.24 %	528.36 %	324.12 %
Average	71.99 %	17.93 %	6.29 %	1.97 %	1.28 %	0.78 %
Maximum	84.40 %	39.74 %	7.95 %	100 %	4.63 %	2.38 %
Minimum	48.92 %	5.55 %	3.63 %	0.36 %	0.50 %	0.04 %
StDev	4.16 %	4.27 %	0.66 %	4.89 %	0.70 %	0.69 %
Avg/Max	0.85	0.45	0.79	0.02	0.28	0.33

Table 4: User functions of the CHEM model for the functions of the exch module.

In the histogram of figure 4 we can see how among all the used functions of the module *exch* consumes a **25%** of the time, taking this percentatge as a reference, the maximum speedup if there weren't any communications would be **1.33x**, this speedup can vary according to the configuration of the experiment, being able to introduce more components to solve, changing the time interval or the domain, since this can influence the inequality between the different processes. However, it is impossible to reach this speedup, since there will always be communications between the processes, and it is not possible to eliminate all the load imbalance, so for a future implementation, we should take into account that the performance improvement will be lower.

5.4 MPI Profile CHEM analysis

In this section, we will delve deeper into the MPI Profile of the chemistry model to identify more performance problems and find solutions. We will focus on the part that corresponds to the hours where the chemical part of the model

is used, since the performance problems of the part without chemistry, that is, the *MPI_Waits* at each hour have mainly been studied in the section 5.2. We saw at figure 5 that there were big *MPI_Barrier* (red) every hour at the hours where chemistry was computed, to see if this red bands can affect the scalability of the model also, we can see its behaviour when the number of computing tasks increments.

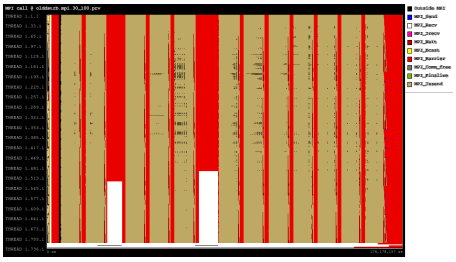

Fig. 14: MPI Profile of chemistry hours with 720 computing tasks.

In figure 14 we see that the *MPI_Barrier* bands are even bigger if the number of computing tasks increases. This means that while the rest of the model is scaling, these communications aren't. Normally, communications take slightly more time when the number of MPI ranks increases, but in this case, they are drastically affecting the performance. To see what is exactly happening, we can zoom in on the red bands.

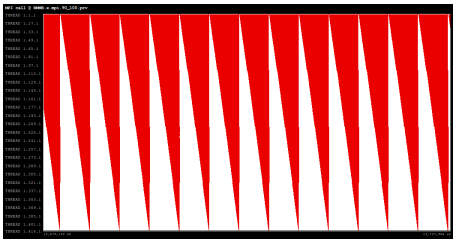

Fig. 15: Red bands of figure 14 zoomed in.

Figure 15 shows the trace when we zoom in on any of the red bands, we can see triangles turning calls of *MPI_Recv MPI_Barrier* which indicate that most MPI ranks remain waiting while some other rank is doing more work than them. This behaviour is inefficient, to quantify how it affects the simulation we can use the histogram of MPI calls of figure 14.

	Outside MPI	MPI_Wait	MPI_Barrier	MPI_Recv	MPI_Allreduce
Num. Cells	719	719	719	719	719
Total	33,774.81 %	15,485.89 %	12,455.14 %	8,395.74 %	1,175.96 %
Average	46.97 %	21.54 %	17.32 %	11.68 %	1.64 %
Maximum	58.78 %	37.76 %	26.54 %	23.96 %	4.08 %
Minimum	29.01 %	10.30 %	4.62 %	2.88 %	0.44 %
StDev	4.37 %	4.00 %	7.21 %	7.16 %	0.68 %
Avg/Max	0.80	0.57	0.65	0.49	0.40

	Outside MPI	MPI_Wait	MPI_Barrier	MPI_Recv	MPI_Allreduce
Num. Cells	719	719	719	719	719
Total	60,179,340,069.84 us	27,592,479,744.86 us	22,192,345,528.26 us	14,959,381,121.79 us	2,095,295,905.66 us
Average	83,698,664.91 us	38,376,188.80 us	30,865,570.97 us	20,805,815.19 us	2,914,180.68 us
Maximum	104,736,724.88 us	67,277,086.55 us	47,297,234.60 us	42,692,296.33 us	7,275,082.95 us
Minimum	51,694,921.19 us	18,358,058.73 us	8,232,147.49 us	5,128,631.16 us	776,471.87 us
StDev	7,791,755.20 us	7,118,828.63 us	12,837,977.20 us	12,764,435.40 us	1,211,851.79 us
Avg/Max	0.80	0.57	0.65	0.49	0.40

Table 5: Percentage and total time (microseconds) consumed by the different MPI calls.

In table 5 we see how the parallel efficiency is only **47%** for the hours solving chemistry, having an average of **17%** consumed by *MPI_Barrier* and **11%** by *MPI_Recv*, with both of them exhibiting a huge difference between the maximum and minimum percentatges, explained by the triangles of figure 15. This confirms that this part of the code is executing in a sequential manner, as one MPI rank does most of the work.

After using the callers functionality shown in figure 12, it has been concluded that these communications come from the subroutine *dstrb*. This subroutine carries out the MPI communications to distribute the relevant sections of a global array from the master process to the other MPI ranks. This subroutine is used frequently throughout the execution, specifically every time binary files have to be read, which happens during initialization and in certain simulations also at the start of each hour, as is the case here. Being a function of a fairly general nature and that is used in several parts of the simulation, it is worth studying it in depth and seeing if it can be optimized.

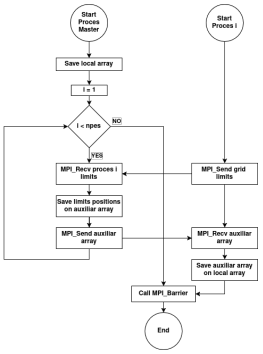

Fig. 16: ***Dstrb*** function flux diagram.

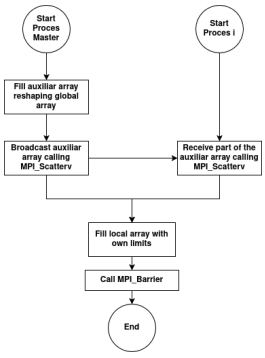

Fig. 17: ***Dstrb*** new implementation flux diagram.

Figure 22 shows the flow diagram of the function *dstrb* mentioned above. As concluded from the trace in the figure 15, it is done sequentially, so that the master process processes the arrays of each of the processes in the grid, receiving the limits from them and sending the corresponding positions to each one. This implementation, in addition to being inefficient due to being sequential, is also inefficient due to the large amount of communications that the master process must make with the rest.

Since the master process already has all the information of the grid distribution, it doesn't have to receive the limits from every MPI rank individually, so this communications can be changed to a simple *MPI_Scatterv* sending to each rank the part of the array that belongs to the grid cells that it is computing. To do this, the global array has to be preprocessed by the master process, but this is more efficient than iterating over all the ranks as it was doing. The new implementation is shown in figure 23.

This new implementation leads to a great improvement in the performance of the program, since the master process no longer has to iterate through the rest. The trace corresponding to the new execution corresponds to the figure 18, which if we compare it with the initial 14, we can see a great improvement, so that at first glance the red bands corresponding to the distributions of the global arrays are almost not visible.

The traces of figure 18 show that the new implementation reduced the time the processes spend distributing the global

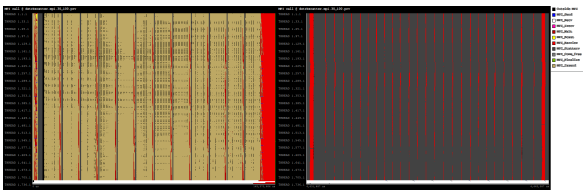

Fig. 18: Resulting traces after improving the *Dstrb* function.

arrays. As we see in the zoomed trace (right one), they still spend time on *MPI_Scatterv* waiting for the master process, but it is significantly less than before.

	Outside MPI	MPI_Wait	MPI_Scatterv	MPI_Barrier	MPI_Allreduce
Num. Cells	719	719	719	719	719
Total	41,055.63 %	19,558.22 %	4,847.03 %	4,275.90 %	1,484.55 %
Average	57.10 %	27.20 %	6.74 %	5.95 %	2.06 %
Maximum	71.06 %	47.24 %	6.92 %	6.53 %	5.13 %
Minimum	34.99 %	13.90 %	6.56 %	5.28 %	0.57 %
StDev	5.29 %	4.81 %	0.07 %	0.33 %	0.86 %
Avg/Max	0.80	0.58	0.97	0.91	0.40

	Outside MPI	MPI_Wait	MPI_Scatterv	MPI_Barrier	MPI_Allreduce
Num. Cells	719	719	719	719	719
Total	59,766,166,567.80 us	28,471,603,079.60 us	7,056,000,804.22 us	6,224,575,548.32 us	2,161,118,203.70 us
Average	83,124,014.70 us	39,598,891.63 us	9,813,631.16 us	8,657,267.80 us	3,005,727.68 us
Maximum	103,442,122.91 us	68,768,101.42 us	10,069,791.24 us	9,502,176.92 us	7,468,113.03 us
Minimum	50,936,026.80 us	20,228,949.34 us	9,556,366.15 us	7,688,017.61 us	832,552.79 us
StDev	7,703,261.44 us	6,995,660.40 us	101,843.25 us	476,427.98 us	1,249,304.09 us
Avg/Max	0.80	0.58	0.97	0.91	0.40

Table 6: Percentage and total time (microseconds) consumed by the different MPI calls with the new implementation.

The table 6 show that the percentage of time consumed by *MPI_Scatter* and *MPI_Barrier* is a **7%** and **6%**, which is much lower than the **11%** and **17%** consumed previously by *MPI_Recv* and *MPI_Barrier*. But since the percentatges are relative to the total time of execution, we can compare the average total times for both, which are **18,5** seconds for new implementation and **51** seconds for the old one, what means an improvement of **32** seconds for the communications inside this function.
Also, after using the histograms of total time for checking the total running time of the simulation, it went down from **178,18** seconds to **145,57** seconds, so the new implementation meant a **1.22x** speedup in this case. This affects all the simulations as said before, but the speedup can vary depending on how many times it is called in relation on how many time is spent in the rest of the simulation, so for knowing exactly how the dstrb function improved we can use the user functions for measuring the time the different MPI ranks spend inside it.

	module_d..p_dstrb
Num. Cells	719
Total	24,890,593,143.49 us
Average	34,618,349.30 us
Maximum	34,832,799.49 us
Minimum	34,529,613.50 us
StDev	41,495.14 us
Avg/Max	0.99

	module_d..p_dstrb
Num. Cells	719
Total	1,416,894.92 us
Average	1,970.65 us
Maximum	1,982.85 us
Minimum	1,965.60 us
StDev	2.36 us
Avg/Max	0.99

Table 7: Total time spent on *dstrb* and average time per call to *dstrb* for the old implementation.

	module_d..p_dstrb
Num. Cells	719
Total	12,429,251,331.74 us
Average	17,286,858.60 us
Maximum	31,584,279.83 us
Minimum	16,793,311.76 us
StDev	1,691,851.05 us
Avg/Max	0.55

	module_d..p_dstrb
Num. Cells	719
Total	707,534.09 us
Average	984.05 us
Maximum	1,797.93 us
Minimum	955.96 us
StDev	96.31 us
Avg/Max	0.55

Table 8: Total time spent on *dstrb* and average time per call to *dstrb* for the new implementation.

Tables 7 and 8 show the total time of *dstrb* went down from an average of **34,62** seconds to **17,29** seconds, and average time per call from $\mathbf{1{,}97 \times 10^{-3}}$ seconds to $\mathbf{0{,}98 \times 10^{-3}}$, which means a speedup of **2.00x** for this function.

6 Main Improvements

During the project two major changes have been applied to the model:

1. Splitting the writing tasks into different I/O servers gave a maximum speedup for the DUST model of **2.0x** for the global domain and **2.2x** speedup for the regional domain when incrementing the number of tasks to 676, also a **1.2x** for CHEM.

2. Changing the distribution of global arrays in the model. This gave an overall speedup **1.22x** on 720 processes.

To show how the number of processes impacts the performance improvement,t the scalability of both DUST and CHEM before and after the changes is plotted in figure 19. The solid lines represent the results before the changes and the dotted lines the results after. The total times of the simulations are shown in the annex A.

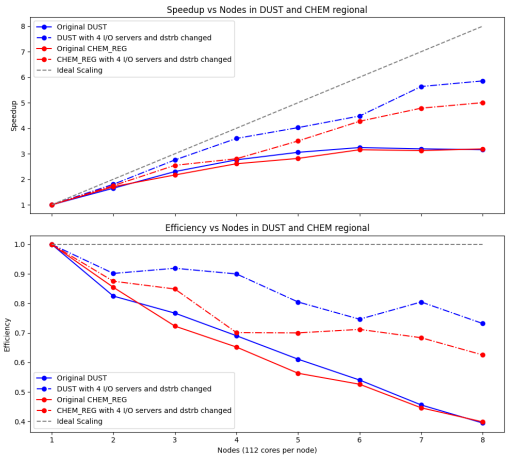

Fig. 19: Time, Speedup and efficiency running CHEM model in regional domain for 1-8 nodes (112 cores per node).

These figures show that the changes do not have a significant impact on both CHEM and DUST models when running with fewer cores, but the performance improves once we increase the number of nodes, shown by the increase in speedup and efficiency, demonstrating improved scalability. Even though there were changes focused on each of the models, both are affected by all of them. The improvements described in this work will be extremely valuable to MONARCH users, as one of the most important and difficult things to improve in these type of models is the scalability, since there is always a bottleneck caused by the communication overhead. These changes improved the communications, which will allow the users to run the model at a higher resolution or more times with fewer resources.

7 Future work

This project also leaves some topics open for future work, the first one was introduced in 5.2, referring to the program writing different output files when splitting the writing tasks in various MPI communicators, making them turn the writing of the simulated hours. This can be solved easily by

running a simple command of the CDO module [13] for performing the merge of the netcdf files by the time variable, since they don't have any conflict on the hours written. Althought this can serve as a fast fix, it takes some extra time, so the ideal solution is to change the model so that the writing tasks perform the writing in parallel with netcdf support, this way the writing itself would be faster and maybe it wouldn't be necessary to perform additional post-processing steps.
In addition, in section 5.3 the possibility of implementing some kind of workload manager was introduced. The idea given was to implement a load-balancing algorithm for managing the grid distribution. Both *knapsack* and *round robin* have the potential to solve this issue, but *knapsack* depends on the fact that we can estimate the cell costs, which can also be a bit computationally expensive and potentially unreliable, so *round robin* is easier to implement and should provide good results, not that different from *knapsack*. We thus propose that *round robin* would be the first choice and starting point.
There are also other versions of the model still not reviewed in this project, in particular the new GPU chemistry solver CAMP model and the nested version of the model, which has many more communications between parent and child domains will also be profiled and analyzed.

8 Conclusions

In conclusion, the objectives raised at the start of the project were met, first understanding how the model works and how the performance can be improved for various modules, for then applying the changes needed to reduce the performance issues and improve the performance and scalability of the model. These changes will allow the users to run more simulations with more processors or higher resolutions without affecting to the scientific results of it.
This report will be used to disseminate the knowledge obtained to other developers and users. Sharing the details of changes applied, as well as how and why they affect the performance, is very important to avoid the repetition of the work. Since there are other similar models to MONARCH in the HPC community, it's possible that they will also benefit from the results of this project.

Acknowledgements

I would like to express my gratitude to all the people who helped me during the course of this project.
First and foremost, I would like to acknowledge my tutor Eduardo Cesar Galobardes, who helped me to get this opportunity inside BSC, as well as guided and advised in achieving all the objectives during the project.
Also, I am so grateful to all my colleagues of BSC, who provided all the help and advice i needed since I started working here. But especially to my mentors Mario Acosta and Hannah Ross, who also guided and advised me during these last months. I would like to thank also the MONARCH unit, led by Hannah Ross, and all the people involved in the MONARCH development, who answered and helped me with all the questions I had about this large and complex model.
Finally, I want to thank my friends and family; without their constant support and guidance, this project wouldn't have been possible.

References

[1] H. Casanova, A. Legrand, and Y. Robert, Parallel Algorithms. New York: Chapman and Hall/CRC, Oct. 2011. doi:10.1201/9781584889465

[2] Palmer, T. Climate forecasting: Build high-resolution global climate models. Nature 515, 338–339 (2014). https://doi.org/10.1038/515338a

[3] Badia, A., Jorba, O., Voulgarakis, A., Dabdub, D., Pérez García-Pando, C., Hilboll, A., Gonçalves, M., and Janjic, Z.: Description and evaluation of the Multiscale Online Nonhydrostatic AtmospheRe CHemistry model (NMMB-MONARCH) version 1.0: gas-phase chemistry at global scale, Geosci. Model Dev., 10, 609–638, https://doi.org/10.5194/gmd-10-609-2017, 2017.

[4] H. Zhang, J. C. Linford, A. Sandu, and R. Sander, "Chemical Mechanism Solvers in Air Quality Models," Atmosphere, vol. 2, no. 3, pp. 510–532, Sep. 2011, number: 3 Publisher: Molecular Diversity Preservation International. [Online]. Available: https://www.mdpi.com/2073-4433/2/3/510

[5] M. Christou, T. Christoudias, J. Morillo, D. Alvarez, H. Merx, Earth system modelling on system-level heterogeneous architectures: EMAC (version 2.42) on the Dynamical Exascale Entry Platform (DEEP), *Geoscientific Model Development*, **9**(9), (2016), 3483–3491. Copernicus GmbH. doi:10.5194/gmd-9-3483-2016

[6] B. Armstrong, S. W. Kim, and R. Eigenmann, "Quantifying Differences between OpenMP and MPI Using a Large-Scale Application Suite," in High Performance Computing, ser. Lecture Notes in Computer Science, M. Valero, K. Joe, M. Kitsuregawa, and H. Tanaka, Eds. Berlin, Heidelberg: Springer, 2000, pp. 482–493.

[7] Barcelona Dust Regional Center. https://ess.bsc.es/our-work/services/barcelona-dust-regional-center

[8] Copernicus air quality center. https://atmosphere.copernicus.eu/air-quality

[9] International Cooperative for aerosol emission. https://aero.und.edu/atmos/icap/index.html

[10] Autosubmit docs. https://www.bsc.es/research-and-development/software-and-apps/software-list/autosubmit

[11] Extrae and Paraver docs. https://tools.bsc.es

[12] NetCDF library and files explained. https://www.unidata.ucar.edu/software/netcdf/

[13] CDO docs. https://code.mpimet.mpg.de/projects/cdo/embedded/inde

A ANNEX

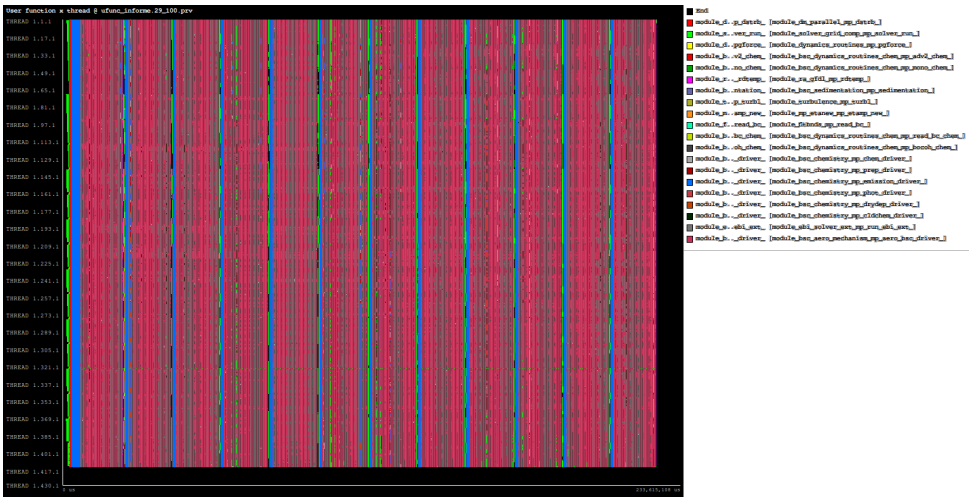

Fig. 20: Full trace of *user functions* for simulated hours with chemistry.

Fig. 21: Time percentatge inside all the MPI calls for the twelve hours of the simulation which compute chemistry.

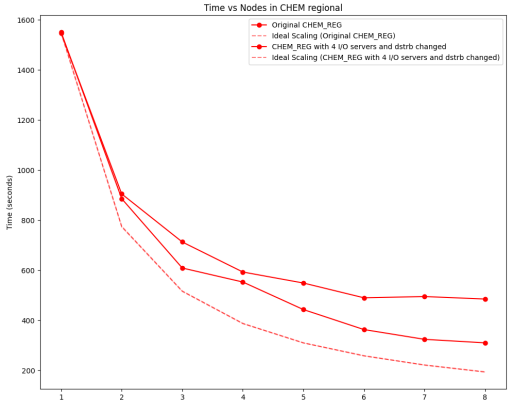

Fig. 22: Scalability of total elapsed time simulation time for chemistry simulation.

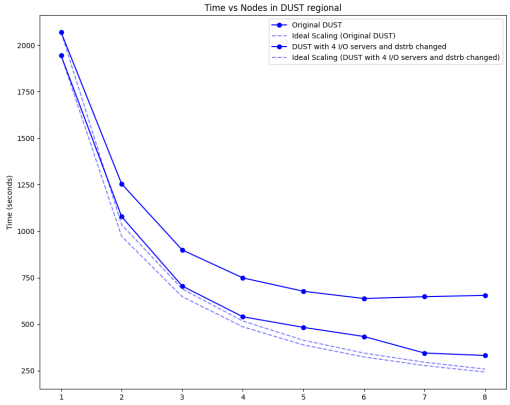

Fig. 23: Scalability of total elapsed time by simulation time for DUST simulation.