Journal of Computational Science 36 (2019) 100864

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Finding, analysing and solving MPI communication bottlenecks in Earth System models

Oriol Tintó Prims[a,b,*], Miguel Castrillo[a], Mario C. Acosta[a], Oriol Mula-Valls[a], Alicia Sanchez Lorente[a], Kim Serradell[a], Ana Cortés[b], Francisco J. Doblas-Reyes[a,c]

[a] Barcelona Supercomputing Center, Spain
[b] Universitat Autònoma de Barcelona, Spain
[c] Catalan Institution for Research and Advanced Studies, ICREA, Spain

ARTICLE INFO

Article history:
Received 29 July 2016
Received in revised form 11 March 2017
Accepted 18 April 2018
Available online 22 April 2018

Keywords:
Earth System modelling
Ocean modelling
Performance analysis
Performance optimization
MPI optimization

ABSTRACT

It is a matter of consensus that the ability to efficiently use current and future high performance computing systems is crucial for science, however, the reality is that the performance currently achieved by most of the parallel scientific applications is far from desired. Despite inter-process communication has already been a matter of study in many different works, it is a fact that their recommendations are not taken into account in most of computational model development processes, at least in the case of Earth Science. This work presents a methodology that aims to help scientists working with computational models using inter-process communication, to deal with the difficulties they face when trying to understand their applications behaviour. Following a series of steps that are presented here, both users and developers will learn how to identify performance issues by characterizing applications scalability, identifying which parts present a bad performance and understand the role that inter-process communication plays. In this work, the Nucleus for European Modelling of the Ocean (NEMO), the state-of-the-art European global ocean circulation model, will be used as an example of success. It is a community code widely used in Europe, to the extent that more than a hundred million core hours are used every year in experiments involving NEMO. In the analysis exercise, it is shown how to answer the questions of where, why and what is degrading model's scalability, and how this information can help developers in finding solutions that will mitigate their eventual issues. This document also demonstrates how performance analysis carried out with small size experiments, using limited resources, can lead to optimizations that will impact bigger experiments running on thousands of cores, making it easier to deal with the exascale challenge.

1. Introduction

Until 2005, processor clock frequencies had been increasing exponentially allowing codes of any kind to improve their performance without almost any effort from the code developer. However, the difficulty in handling the heat dissipation prevented from further increasing the computing power through this way and forced both hardware and software to drift from a sequential to a parallel paradigm, which forced hardware to evolve from the almost hegemonic single-core to different multi-core architectures. In order for the software to take advantage of the computing capacity of these multi-core architectures it necessarily had to exploit both intra and inter-node parallelism and to do so different programming models were developed. For intra-node parallelism, Open Multi Processing (OpenMP) was the most commonly used [16] and with the adoption of graphics processing units (GPUs) for general computation, CUDA[1] and OpenCL[2] were also created [13]. However, among this diversity, Message Passing Interface (MPI) has become the standard paradigm to exploit inter-node parallelism and it now has the role of being the most used parallel programming model [10,27]. The reason for such a success relies on its widely known advantages: it is portable, it has a long history and it is compatible with other paradigms such as shared memory. As a result

[1] CUDA: is a API created by Nvidia. It allows developers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing.
[2] OpenCL:is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors or hardware accelerators.

* Corresponding author.
E-mail address: oriol.tinto@bsc.es (O. Tintó Prims).

https://doi.org/10.1016/j.jocs.2018.04.015

of this popularity, an MPI code will run on most clusters around the world. Nevertheless, MPI also has some drawbacks: its implementation is explicit and it can produce a considerable overhead, making its efficiency depend on the developer's ability.

The need to change to a parallel paradigm made necessary to adapt already existing computational models. Earth System models (ESMs), which nowadays are the most powerful tools to forecast tomorrow's weather and study the future climate, were not an exception. These ESMs are made up from multiple mathematical models that describe the different facets of the Earth System. Not all ESMs include the same components, but their core is always an Atmosphere-Ocean Global Circulation model, which is an atmospheric model coupled with an oceanic one, which can be complemented with other submodels to better describe other important processes as sea-ice, land, biosphere, etc.

Scientific performance of ESMs has been for decades in a sustained path of improvement, fundamentally based on the use of ensemble methods, resolution increments of the grid used to solve the equations [25], or the addition of new features, demanding all these elements more computational power. To summarize, the improvement of the ESMs was sustained by an increase of the computational power, that at one point was only possible to be exploited by switching to the parallel paradigm and using high performance computing (HPC) systems [18,33,34,14].

In many cases the change was not trivial since not every serial algorithm can be easily converted into a parallel version. For example, computational models that use implicit or semi-implicit methods to discretize the governing equations are harder to parallelize than explicit cases [24,21,15]. Implicit or semi-implicit algorithms need to solve systems of equations, generally using iterative methods, involving information from the whole domain that require expensive communication/synchronization operations to be parallelized. As a consequence, parallel implementations have an associated overhead that can increase both the execution time of each solver iteration [2] and the number of iterations required to converge to an acceptable solution [4,19,23,30]. For example, some works [4,7] present a parallel matrix vector multiplication procedure whose implementation has a scalability limited by communication's overhead.

Despite the difficulties presented above, new codes would be developed taking into account all the necessary considerations to exploit better the available resources. However, legacy software was classically designed with sequential flows at a time when parallel implementations were nonexistent or offered a low payback [3] and therefore will have more problems to efficiently use modern computers. These problems may be of different nature: a non-scalable algorithm, a bad implementation or, the most usual, a mixture of these two cases [6], not being always obvious in which of these situations a model is.

This is not the first work that tries to address computational problems with the parallel implementations of ESMs. Some illustrative examples could be [12], where a low-resolution climate model is analysed and major code modifications are proposed in order to increase the performance on a specific machine, and [6], where the different components of the Community Climate System Model are stress tested to find scalability issues. One of the conclusions of this publication was that in most cases at the root of scalability issues there are design choices made when the need for parallelization was less or even nonexistent, and that little modifications could solve many of these problems.

The approach of performing major code modifications is usually an unaffordable solution for most of the scientific models, taking into account that some studies quantified the development cost of a state-of-the-art ESM to be between 500 and 1000 person years [22]. Furthermore, putting significant efforts to optimize a model to best fit into a specific architecture may be not so much attractive, taking into account that the lifetime of the computer systems is usually much shorter than the models'. Then, its portability is most valuable than its capacity to exploit a very specific machine.

The authors believe that in the ESMs case an in-depth performance analysis can lead to feasible and productive solutions that do not require a full rewrite of the code while effectively improving the performance of the model. Knowing that the work of understanding the computational behaviour of applications running in HPC systems is tough, a methodology to undertake this analysis is presented in this work, with the aim to help computational scientists to deal with it. To demonstrate this, the NEMO model is used as a case study. It is a state-of-the-art ocean model based on the Navier–Stokes equations and is being used by a wide community involving hundreds of institutions [1]. Although there have been some previous performance analyses on NEMO [26,8], the approach presented in this work allows to highlight problems not previously identified. In short, this paper shows how a deep analysis of the communications implementation in a ESM can be used to successfully identify scalability bottlenecks. It explains the reasons for their occurrence and proposes optimizations to substantially improve the computational performance of the model.

Section 2 is devoted to introduce the steps necessary to reveal bottlenecks and understand the impact of inter-process communication in the performance of the model. Section 3 contains a description of the model used to do a demonstration of the method. In Section 4 there is an analysis carried out using the proposed methodology. Finally, in Section 5 the conclusions of the work are exposed.

2. The methodology

The methodology proposed in this section intends to be useful to scientists that are running computational models on HPC systems and are willing to understand its computational performance, uncovering communication related issues that are potentially hindering its efficiency.

2.1. Analysis steps

Following the steps stated below, one should be able to identify what, where, and how is constraining the model's scalability, highlighting the role of communication.

2.1.1. Step 1: Determining maximum throughput

A good starting point for learning about the issues that are constraining the performance of a model is to know how fast it can perform when using different amount of resources, and at what point it reaches its peak. There are different metrics that can be used for the model throughput, being many of them field-specific. In the case of climate models a widely used metric is Simulated Years per Day (SYpD), which indicates the number of simulation years that can be completed in one day of wall-clock time. When it is time to measure the throughput of a model, it has to be taken into account that most models have initialization and finalization phases that may not be relevant in time in real simulations but that can be relevant in short tests.

2.1.2. Step 2: Finding the bottlenecks

Once it is known how the model throughput evolves and how many cores have to be invested to reach its peak value, the next step is to identify the code regions that do not scale properly. If they exist, these will be those representing a substantially higher proportion of time when the model speed reaches its peak, in comparison to a smaller case. To identify these regions, the only required information is the time spent inside each function for two different cases, usually the single-node case and another one closer

to the maximum model throughput. There are several tools that can help to collect this information, being the most common gprof [9]. However, in this work the tools used were Extrae, to collect the information, and Paraver, to visualize it, both from the BSC tools suite.

2.1.3. Step 3: Bottleneck deeper insight

Once the bottlenecks are localized, the next step is to find the role that inter-process communication plays on them and, therefore, MPI information of these regions has to be collected. While MPI statistics could certainly give valuable information, sometimes they do not uncover the real problem. In these cases, more detailed information from all the individual MPI calls would give more insight into the structure of the application, providing a meaningful understanding of what is happening during the execution. For this reason, advanced tracing tools like those included in the BSC toolkit become very useful.

To provide an example, the percentage of the time spent in communication is a good metric of the parallel efficiency, and a high value of this quantity is a clear sign of issues related with communication, but does not describe well the nature of these problems. There are many different MPI operations: sending and receiving point to point messages, collective operations, waiting for operations to finish and all these operations use to be accounted as communication time, even though, it is not the same to be transferring data than being waiting for a message from another process.

2.2. Tools

For the task of analysing the model computational performance in an HPC platform, the set of performance tools developed at the Computer Science Department of the Barcelona Supercomputing Center (BSC-CNS) was used. This suite, that is open-source and can be freely downloaded, includes a tool to collect information from the model execution (Extrae) [31], a tool to simulate the behaviour of eventual runs in different hardware conditions (Dimemas), and also a tool to visualize the performance data collected or simulated (Paraver) [17].

Extrae collects information from an application and stores it in form of traces. This information can be of three different types:

1. Timestamps of the events occurring during the execution (up to nanoseconds).
2. Performance and other counter metrics using the Performance Application Programming Interface (PAPI) and the Performance Metrics Application Programming Interface (PMAPI).
3. References to the source code to relate it with the performance.

The Dimemas tool is able to estimate the behaviour of a message-passing program in a target platform. It is used to simulate what-if cases on different classes of architectures including clusters, SMPs, heterogeneous systems, etc. Finally, Paraver is used to visualize the information stored in the traces, both those coming from real executions and those output from performance simulations. One of its main features in comparison to other tools is that it offers a great flexibility to explore the data contained in the traces. A more complete description of these tools and several others can be found in http://tools.bsc.es.

3. Case study

This section contains a brief description of NEMO, that was chosen as case study, to be analysed using the proposed methodology. Here, then, what can be found is some information about this model, details about the specific configuration used for the simulations and information about HPC environment where the experiments were performed.

3.1. About NEMO

The Nucleus for European Modelling of the Ocean (NEMO) [20] is a state-of-the-art modelling framework for oceanographic research, operational oceanography seasonal forecast and climate studies, which is used by a large community: about 100 projects and 1000 registered users around the world. It is controlled and maintained by an European consortium, made up by CNRS and Mercator-Ocean from France, NERC and Met Office from the United Kingdom and CMCC and INGV from Italy [1].

The framework includes five major components, some of them can work standalone and others need to run on top of others. The principal components are:

- OPA: dynamics and thermodynamics of the ocean.
- LIM: dynamics and thermodynamics of the sea-ice.
- TOP: biogeochemistry.
- AGRIF: adaptive mesh refinement.
- TAM: data assimilation component.

The core of NEMO is the OPA module. It solves the Navier–Stokes equations from regional to global scales using Euler first-order discretization methods on a three-dimensional (3D) grid. However, despite the domain being represented in three dimensions, the dynamics and thermodynamics of the ocean involve very diverse phenomena, some of them being simulated over horizontal planes while the others are simulated over the vertical axis.

This diversity entails that not all the routines compute over the full 3D domain and some of them only act over 2D surfaces. The model was parallelized and is able to be executed in both shared and distributed memory environments, using a 2D-Pencil domain decomposition method (Fig. 1) that splits the global three-dimensional domain in two dimensions, keeping all the vertical levels from the global domain inside each one of the sub-domains. Domain decomposition methods require the sub-domains to communicate in order to exchange boundary conditions (dark grey regions in Fig. 1) between neighbour subdomains. In NEMO this operation is performed by means of a module using MPI.

The LIM model [32,28] solves the dynamics and thermodynamics of the sea-ice. It has to be coupled with the OPA module and uses the same grid but only operates on the surface layer, implying that computations are performed mainly over 2D arrays.

The model is able to run on different grids, the most commonly used being the ORCA irregular grid, which has three poles to avoid numerical instabilities in the north, and is available in several resolutions. A scientist can use a low resolution grid model to study the general ocean circulation running the model on his laptop while another one can study coastal current vorticity in a high resolution grid using a supercomputer. It becomes clear that the configuration used is fundamental for the computational cost of the simulation and the performance issues that can arise.

The computational cost is estimated to be proportional to the number of grid points, so an ultra-high resolution grid ORCA12 requires 1239.5 times more computational power to perform a time-step than a low-resolution ORCA2 grid. Furthermore, an oceanic model is subject to the Courant–Friedrichs–Lewy (CFL) condition, which forces the time step length in oceanic modelling to satisfy multiple criteria associated with different physical processes in order to guarantee numerical stability [11]. Due to CFL, the use of a higher spatial resolution implies that the maximum time step length that can be used is reduced, this value being determined by the maximum propagation speed of internal waves [29]. For this

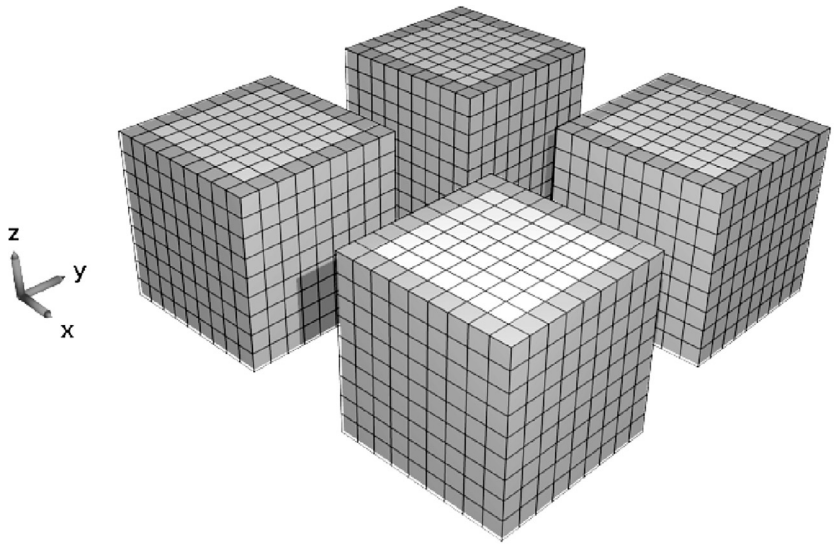

Fig. 1. Vertical 2D-Pencil domain decomposition for three-dimensional grids. In dark gray, the boundary elements that have to be interchanged between the sub-domains.

Table 1
Summary of different available ORCA grid resolutions. The included information tells us the approximate separation between grid points near the equatorial line, the number of vertical levels, how much time does a model step simulate and finally the number of grid points that a specific resolution has.

Name	Horizontal resolution (degrees)	Number of vertical levels	Time-step length (s)	Total points
ORCA2	2	31	5760	0.8M
ORCA1	1	75	3600	7.9M
ORCA025	1/4	75	900	110.4M
ORCA12	1/12	75	3006	991.6M

reason, the number of iterations that the model has to perform to simulate the same time lapse increases with the spatial resolution.

Table 1 shows that while ORCA2 has a default time step length of 5760 s, the usual time step lengths for higher resolutions are 3600, 900 and 300 s for ORCA1, ORCA025 and ORCA12 respectively, making necessary to do 1.6, 6.4 and 19.2 times more iterations for the same experiment length. This not only implies that higher resolutions have much more computational workload per time step, but also that they have to perform more time steps to simulate the same experiment. Taking into account both spatial and temporal resolution, the final cost of an ORCA12 simulation is at least 23,798.4 times higher compared to an ORCA2 simulation.

3.2. Configurations

As it was stated above, the grid resolution is the main conditioning factor for the computational cost of a simulation, but not the only one. There are other features (modules, runtime options, etc.) that have also an important impact. Since the interest is to stress the impact of the communication on the model scalability, a low-resolution configuration was chosen, considering that the communication overhead represents a bigger proportion of the time than in higher resolutions. However, the optimizations proposed have also been tested on a high-resolution configuration to evaluate the impact of these optimizations using different resolutions. The low resolution experiments use an ORCA2-LIM3 configuration, and they can be easily reproduced since the configuration is contained in the model sources and the input files are available in the NEMO website. The only modification required is the activation of a performance optimization parameter in the namelist, which is disabled by default. This can be done by setting the flag ln_nnogather = .true.. For the high resolution case, no official configuration is provided with the model sources. However, there is a benchmark configuration based on an ORCA025 grid and created for performance purposes, available in the NEMO repository. Both low and high resolution experiments use the OPA and LIM3 modules, the former for the ocean, the latter for the sea-ice. The reason for using these specific modules and not the other options, is that those are used in the Earth System coupled model developed by the EC-Earth consortium, to which the BSC-ES department belongs, and that is key piece for CMIP6 experiments. Besides, the sea-ice model LIM3 is quite new and it is more computationally expensive than the previous versions [28]. Further studies considering the other modules are foreseen in the future.

3.3. Environment

The experiments have been executed in the Marenostrum III supercomputer, hosted in the BSC-CNS. Marenostrum III is a 1,1 petaFLOPS machine with 48,896 processor cores spread over 3056 nodes. Each computing node has two Intel Sandy Bridge-EP E5-2670/1600 processors with 8 cores running at 2.6 GHz each, and with 8x4GB DDR3-1600 DIMMS memory (2 GB/core). The nodes are coupled with a high-performance Infiniband FDR10, and an auxiliary Gigabit Ethernet network is used for the shared file system. For these experiments Intel compilers v16.0.1 and the Intel MPI library v5.1.2.150 have been used.

4. Analysis and results

This section sets out in Section 4.1 the analysis of NEMO, the case study, following the steps presented in Section 2. Additionally, in Section 4.2 there is a little further analysis of the code and some solutions for the problems identified during the analysis are proposed, including an evaluation of their impact.

4.1. Analysis

4.1.1. Step1: Determining maximum throughput

The first step of the analysis must be to determine how fast the model can simulate when using different number of resources and when it achieves its maximum throughput. To collect this information it was not necessary to use additional tools, whereas the system

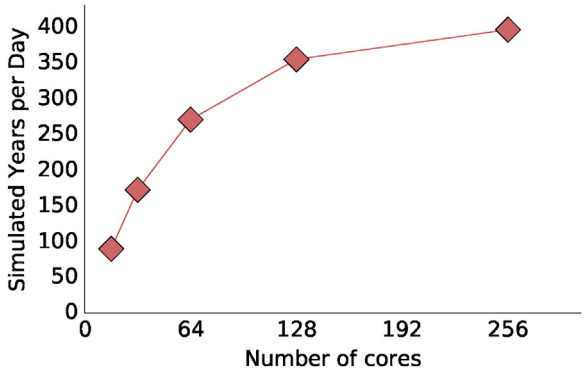

Fig. 2. Throughput in Simulated Years per Day achieved by the NEMO model using an ORCA2-LIM3 configuration with different number of resources.

Table 2
Model throughput in Simulated Years per Day using an low-resolution ORCA2-LIM3 configuration for different number of cores. The relative efficiency is the comparison of the amount of simulated years that can be produced with a specific amount of resources, normalized with the single-core case.

Number of cores	Throughput (Simulated Years per Day)	Relative efficiency
16	89.73	100%
32	172.45	96%
64	270.59	75%
128	354.96	49%
256	396.40	28%

time was written at each time step, deriving from this information the model throughput. This approach allows to discard the time spent in both the model initialization and finalization while capturing the simulation throughput in shorter-duration executions. For each one of the different evaluated instances (number of cores used) samples of ten measures (measured speed of an execution) were taken.

Looking at the Fig. 2 and Table 2 it can be seen how the model throughput evolves, as more resources are being used. Running on a single node the model throughput is almost 90 SYpD, reaching 355 SYpD when using 8 nodes (128 cores), being very small the improvement beyond this point only 11% faster when doubling the resources used, which is a huge drop in the efficiency. It can also be seen that the scalability of the model is far from being linear, being the efficiency of the 8 node case just below 50% of the single node case efficiency.

4.1.2. Step 2: Finding the bottlenecks

In the second step, the specific routines that do not scale properly (if that is the case) must be identified. The previous step analysis showed that the model reaches almost its maximum throughput when using 8 nodes (128 cores). Having this information, and in order to proceed with the second step, function information has to be collected for both single and 8-node executions. Nevertheless, function information for other cases has also been collected and represented in Fig. 4 to have a clearer picture of how the time consumed by each function evolves (Tables 3 and 4).

Looking at the distribution of the time spent in each function (see Fig. 3), in the single node case the model has a very flat profile, with the most expensive routine spending only 14.6% of the execution time. However, in the 8 node run, there is a qualitative change, being most of the time consumed by few routines. This fact indicates that the lack of model scalability is mostly caused by these routines, that become the performance bottleneck. As it can be seen in Fig. 4, the functions that are relevant because of their bad

Table 3
Throughput of the NEMO model with a ORCA2-LIM3 configuration for the different versions of the code using different number of cores. The version v0 corresponds to the original code without optimizations, the version v1 includes message aggregation, the versions v2 include the reduction of the global sums in the sea-ice diffusion routine and the version v3 includes also the reorganization of the sea-ice diffusion routine. v2a, v2b and v2c correspond to the same version of the code being different the frequency at which the global sums are performed, in the a case.

No. of cores	Version					
	v0	v1	v2a	v2b	v2c	v3
16	89.73	90.16	90.70	90.97	90.80	87.80
32	172.45	174.69	176.34	177.25	177.81	174.43
64	270.59	276.99	289.31	295.93	297.66	290.22
128	354.96	394.16	406.46	426.60	429.83	461.96
256	396.40	455.46	483.92	511.37	520.06	597.60

Table 4
Throughput of the NEMO model with a ORCA2-LIM3 configuration for the different versions of the code using different number of cores.

No. of cores	Version			
	v0	v1	v2	v3
256	1.67	1.69	1.78	1.65
512	2.68	2.70	2.89	2.87
1024	3.78	3.94	4.28	4.55
2048	4.16	4.39	5.20	6.08

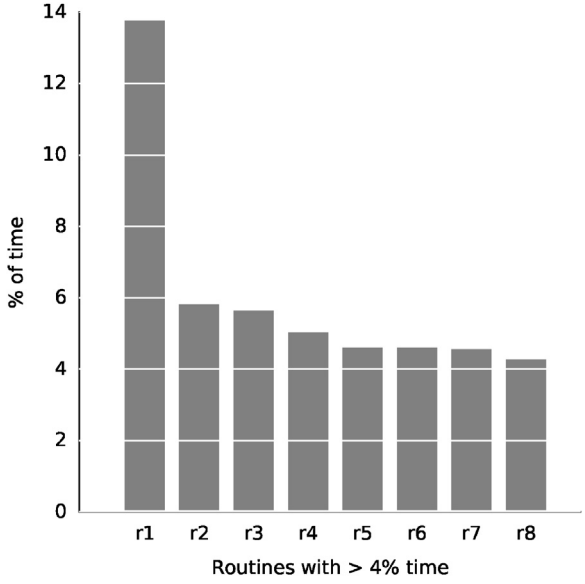

Fig. 3. Percentage of time spent in the eight most costly routines for an ORCA2-LIM3 simulation using a single-node (16-cores) of Marenostrum III. The profile is very flat and the most costly routine (the surface pressure gradient) does not even reach a 14% of the total execution time.

scalability are mainly three, representing together 24% of the time when running on a single node but more than 50% in the 8-node case.

The first of these routines (surface pressure gradient) solves the dynamics of the ocean surface pressure gradient and belongs to the OPA module, while the second and the third belong to the LIM3 module and compute the horizontal diffusion (sea-ice horizontal diffusion) and rheology (sea ice rheology), respectively.

Once the worst scaling code regions have been determined, a further study of these regions is needed.

4.1.3. Step 3: Bottleneck deeper insight

The third step is devoted to explore the role that inter-process communication plays on the bad scalability of the regions previ-

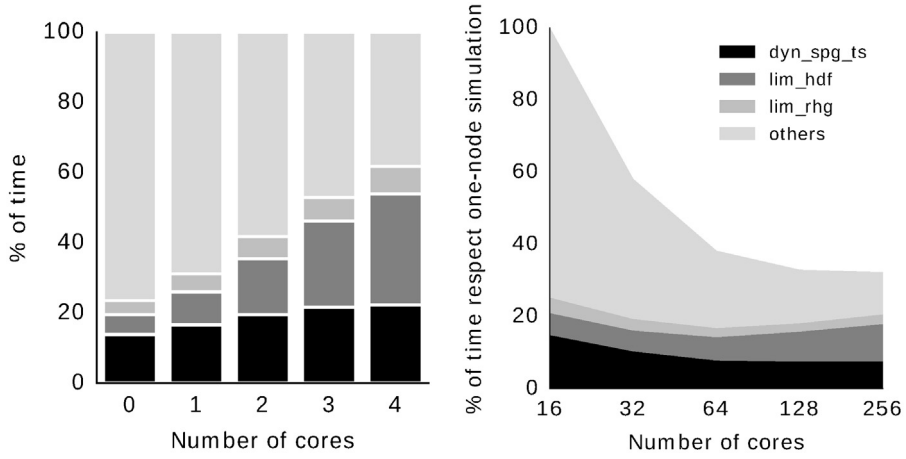

Fig. 4. (A) Percentage of time spent in sea surface pressure gradient (dyn_spg_ts), sea-ice horizontal diffusion (lim_hdf), sea-ice rheology (lim_rhg) and all the other routines (others) when using different number of cores. (B)Time spent in sea surface pressure gradient (dyn_spg_ts), sea-ice horizontal diffusion (lim_hdf), sea-ice rheology (lim_rhg) and all the other routines (others) when using different number of cores compared with a single node execution.

ously identified. MPI information from these regions can be used to compute metrics that are useful to characterize the impact of the communication overhead and, with a further analysis, to determine the potential causes of that behaviour. With regard to the case study, in the second step three main routines were identified: sea surface pressure gradient, sea-Ice horizontal diffusion and sea-ice rheology. For these regions, MPI information was collected as it is explained in Section 2.1.3.

In the case of the **surface pressure gradient** routine, using the Paraver tool to visualize the MPI information collected through different experiments (Fig. 5), it was possible to see that, even in the one-node case, the time spent in communication exceeds 30% of the time. The potential of these tools is that, even without having any previous knowledge of the code, it is possible to see which is the origin of performance loss. In this case the tool revealed that, inside this routine, most of the execution time is spent in a single loop containing four computation phases separated by four communication phases. There are seven border updates distributed over them: three phases with two interchanges, and one phase with a single interchange. Having this seven intercommunications taking place in each every loop iteration, it is clear that what is constraining this routine to scale is the high rate of communication.

For the second routine identified, **sea-ice horizontal diffusion**, the previous step of the analysis showed a very bad scalability. Using Paraver (see Fig. 6), it is possible to identify a loop structure consisting in two computational phases with very short duration, separated by two communication regions. The former communication region corresponds to border interchange while the latter consists in one collective operation which requires synchronization of all the processes. Like in the first routine, there is an issue with the high amount of communications, but in this case these concentration is even more harmful because of the requirement of global synchronization. Additionally, it can be observed that the four last processes spend more time in the border interchanges, and in consequence the other processes have to wait more time in the collective communication.

The third one of the routines, **sea-ice rheology** (see Fig. 7), presents the same issues that the sea surface pressure gradient, it is a high rate of communication.

At this stage and only following the three steps stated in the method proposed, it is already clear where and what is constraining the model's scalability.

4.2. Further exploration and optimizations

Whilst the purpose of the proposed methodology is to uncover communication issues in a simple manner, this section goes a little further and shows how the revealed information can lead to solutions able to improve the model performance.

The main analysis' outcome indicates that mainly three routines were identified as bottlenecks and in the three cases the main reason was the high rate of communication, additionally having in one of the cases the presence of collective operations making the problem worse. The analysis' main outcome indicates that essentially three routines were identified as bottlenecks and in all the cases the fundamental reason was the high rate of communication, with the aggravating circumstance of having collective operations in one of the routines. This information is very important to understand what is happening with regard to the performance, but to know what changes can be done in each case it is necessary to further study the code and understand what each routine is doing.

The first of the routines identified, the **surface pressure gradient**, is part of the OPA module and is in charge of updating the dynamic trend with the lateral diffusion. Out of the different phenomena simulated in the dynamical core, the barotropic flow has a higher speed than the rest. As it was explained, CFL conditions make higher speeds to require smaller time steps to avoid numerical instabilities. To overcome this issue, the current solution consists in computing separately the equations involving this barotropic flow by using a time-splitting strategy, along with a split-explicit scheme iterative algorithm. With this method, the resolution of the implicit scheme system and the overall time-step reduction are avoided at the cost of doing extra iterations in the routine. Event this routine implements the time-splitting strategy, it still has to perform many iterations per time-step, involving communication among neighbour subdomains. However, there is not a sound reason to have consecutive interchanges of different variables performed in different messages, since it is not enforced by the algorithm. The same situation of consecutive interchanges with the neighbours happens in the **sea-ice rheology** routine, that computes rheology processes on sea-ice and requires also an iterative process to solve the momentum equation.

In both cases this approach is likely to have been followed simply because it was easier for the developers, who probably did not expect the impact it would have in the performance. This point suggests that a method called message aggregation can be applied. It consists in gathering all the different consecutive messages going

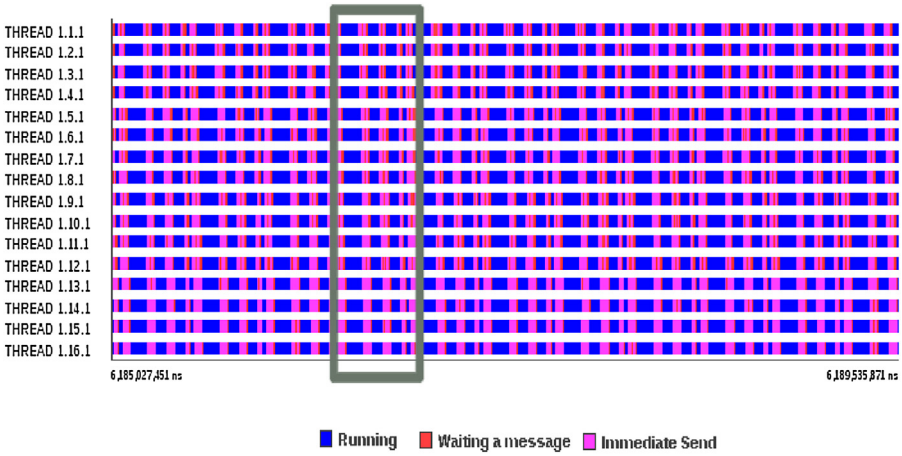

Fig. 5. Paraver view of the sea surface pressure gradient routine. Inside the gray rectangle, it can be seen the pattern that it is repeated consisting on four computation phases (in blue) separated by communication phases (red and pink). The different lines in the *y* axis represent the different threads (16 in the single node case) and the *x* axis represents the time. Each line shows what is happening in each one of the threads among the time, in this case inside the sea surface pressure gradient routine. The whole figure corresponds to 4.5 ms of simulation. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

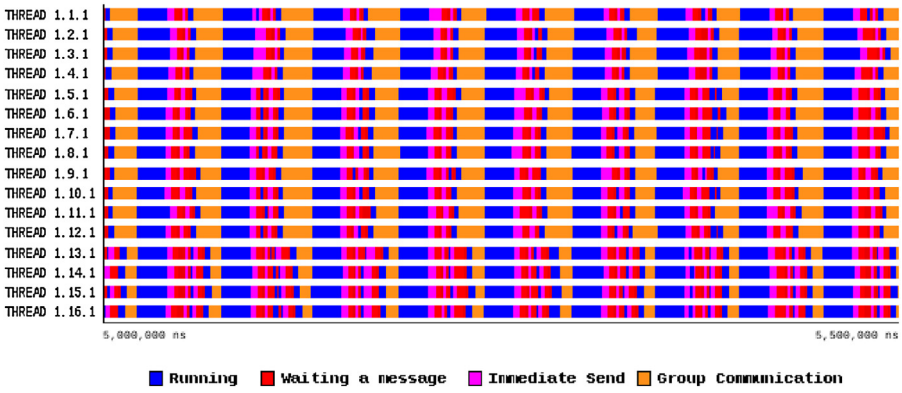

Fig. 6. Paraver view of the sea-ice horizontal diffusion routine. It can be seen a iterative structure consisting in a computational phase (in blue), a border interchange (red and pink) a very short computational phase and a group communication (in orange).The different lines in the *y* axis represent the different threads (16 in the single node case) and the *x* axis represents the time. Each line shows what is happening in each one of the threads among the time, in this case inside the sea surface pressure gradient routine. The whole figure corresponds to 0.5 ms of simulation. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

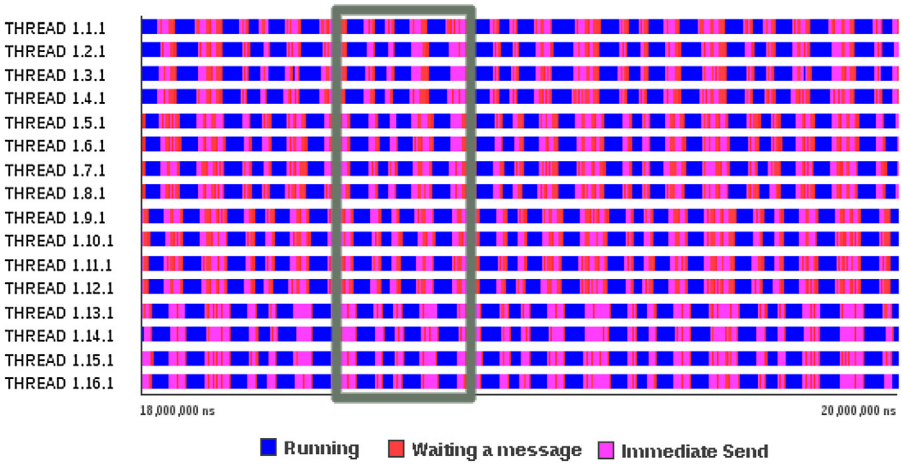

Fig. 7. Paraver view of the sea-ice rheology routine. Inside the gray rectangle It can be seen the iterative structure consisting in four computational phases (in blue) separated by border interchanges (red and pink). The different lines in the *y* axis represent the different threads (16 in the single node case) and the *x* axis represents the time. Each line shows what is happening in each one of the threads among the time, in this case inside the sea surface pressure gradient routine. The whole figure corresponds to 2 ms of simulation. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

to the same destination in one single message, making it possible to pay the latency cost once only. To do this, one routine was adapted to interchange an arbitrary number of variables with the neighbour subdomains in a single call. It was possible to apply message aggregation not only in the identified routines, but also in some more places of the code, although is in these places where the expected impact is bigger.

In order to test the impact of this optimization, the throughput of the modified code was compared with the original version. Fig. 7 shows that a 15% throughput improvement is achieved using 256 cores. In the same sense, Fig. 8 shows that in high resolution the maximum improvement is achieved when running 2048-cores, having the version including message aggregation (v1) 5% more throughput than the original (v0). It can be observed that this technique has little impact in low core-counts (big subdomains and bigger computation over communication ratio) because it has more effect when the subdomains are small and communication spends a bigger proportion of the time. The absolute impact of this optimization is not outstanding, but out of any doubt is relatively efficient, taking in account the little effort its implementation demands.

In the **sea-ice horizontal diffusion** routine, the second region of interest and the one that presents the worst scaling behaviour in the tested configuration, it was also observed a high rate of communication, but additionally, an abuse in the usage of collective operations that required synchronization was detected. Looking into the code it can be seen that this routine, part of the sea-ice module LIM3, computes the diffusion trend on the sea-ice variables. It uses a second order diffusive operator evaluated using a Crank–Nicholson method [5], a semi-implicit scheme that requires the use of an iterative approach. In the analysed implementation a boundary interchange was required in all the iterations, as well as an evaluation of the exit condition. And this routine was executed once for each one of the variables on which diffusion has to be applied, whose number depends on the number of sea-ice categories and layers chosen. In the configuration used, this routine acts on 41 different variables. The commented implementation may not be problematic when the communication overhead is some orders of magnitude smaller than the computation time (big subdomains) but it becomes problematic when using more processors and the subdomains size is reduced, i.e. when the demands on the parallelization are increased. To allow this routine to scale, it was necessary to reduce both the use of point-to-point messages and the number of collective communications.

The simplest way to reduce the number of collective communication consisted of a reduction of the frequency of the exit condition evaluation, performing the check not at every single sub-time step, but only at few of them. This is an acceptable strategy since performing some extra iterations does not degrade the quality of the solution and the performance improvement expected is important, considering that the global additions represent a big proportion of the time. The impact of this measure can be seen in Fig. 8. It puts in relief that the reduction of the check frequency has a big impact in the performance when the subdomains are small, reaching a 14% increase in the maximum throughput with the convergence check performed only one time every 8 loop iterations (v2c) with respect to the version with message aggregation (v1), and a 31% with respect to the original one (v0). Along the same lines, Fig. 9 shows that the impact is also important in high resolution, where using a convergence check frequency value of 5 (v2) leads to a 18% improvement in the 2048-case with respect to the message aggregation case, and a 23% with respect to the original case.

On the contrary, there is no trivial way to reduce the amount of messages in this routine since there is a single border interchange at each loop iteration. To reduce the number of point-to-point messages by merging them, a reorganization of the routine code is unavoidable. In this case, the absence of dependencies between

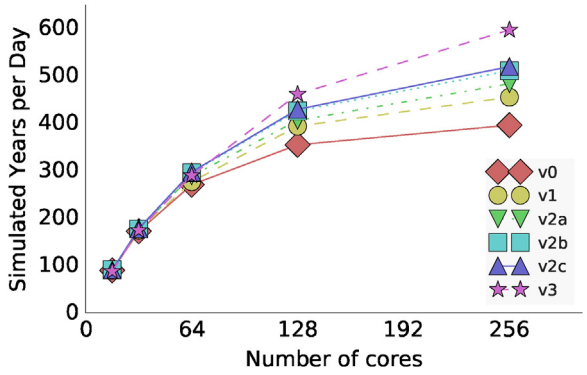

Fig. 8. Model throughput in Simulated Years per Day using an ORCA2-LIM3 configuration for the different versions of the code with the different optimizations and for different number of cores.

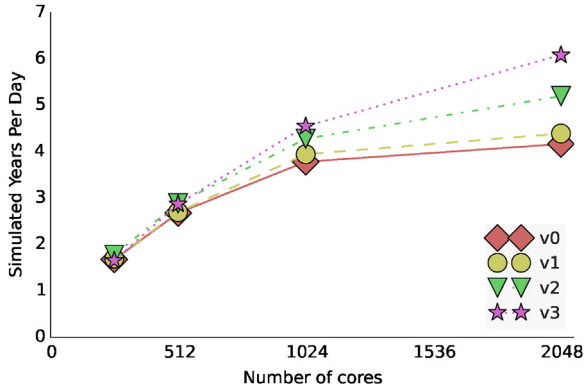

Fig. 9. Model throughput in Simulated Years per Day using an high resolution ORCA025-LIM3 configuration for the different versions of the code with the different optimizations and for different number of cores.

the different variables can be exploited to group the computation phases on one side, and the communication phases, on the other side, to allow the merge of messages and collective operations. The idea is to do as much work as it is possible to do, without communicating with the neighbours, and then communicate with a single call the border information of all the variables. By reordering the code using this strategy it would preserve exactly its numerical results, and at the same time allow the use of message aggregation and reduce by more than one order of magnitude the number of messages.

The impact of this optimization (v3) with respect to the other versions can be seen in Figs. 8 and 9 . The speed up improvement in comparison with the second optimization version (v2) is 15% and 17% for the low- and high-resolution configurations, respectively. Compared to the original (v0), the improvement reaches 49% and 46% for the low- and high-resolution configurations, respectively. Nevertheless, it is true that for low core counts this modification does not improve the performance but even slows it down a little bit. This is probably explained by the fact that, when the reorganization is applied, the counterpart of reducing the number of communications is that the computation inside the sea-ice horizontal diffusion routine does a worse use of the memory hierarchy. When the size of the subdomains is big and therefore, the communication overhead has little importance this can effectively slow down the simulation. Even having this in consideration, high resolution experiments do take a lot of advantage of this optimization.

5. Conclusions

High performance computing plays and will play a central role on science and, on the road to exascale, the ability to efficiently use the available resources will be crucial. However, many cutting-edge scientific tools, of which Earth System models are not an exception, still present difficulties to scale, tens that could prevent them from exploiting future supercomputers. Besides, most of these models rely on legacy codes, originally thought to be executed in small clusters or even in serial machines, having inefficient implementations that in most cases present issues related with inter-process communication. While it is true that exascale will require of new methods and algorithms, it is not less true that helping developers of legacy software to adapt their models will be key for the excellence of science. To make progress in this direction, this work presents a methodology with easy-to-follow steps to find performance bottlenecks and uncover the role that communication plays in them, with the aim to help this needy applications. To demonstrate the usefulness of this work, the methodology was applied to the analysis of the NEMO model, having satisfactory results. The analysis helped to identify performance bottlenecks, showing that the high frequency of communications that some routines presented, and the use of collective operations was preventing the model to scale. Thanks to the outcomes of this analysis, some solutions were proposed to reduce the amount of communications, achieving a significant improvement in the maximum model throughput, both in low and high resolution configurations, with 49% and 46% improvement respectively. It is worth noting that the analysis in a low resolution configuration led to optimizations which had an impact in high resolution.

Acknowledgements

The research leading to these results has received funding from the EU ESiWACE H2020 Framework Programme under grant agreement no. 675191, from the Severo Ochoa (SEV-2011-00067) programme of the Spanish Government and from the Ministerio de Economia y Competitividad under contract TIN2014-53234-C2-1-R.

References

[1] NEMO Website. www.nemo-ocean.eu.
[2] P. Amestoy, I. Duff, J.-Y. L'Excellent, Multifrontal parallel distributed symmetric and unsymmetric solvers, Comput. Methods Appl. Mech. Eng. 184 (2–4) (2000) 501–520.
[3] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, K. Yelick, A view of the parallel computing landscape, Commun. ACM 52 (10) (2009) 56–67.
[4] B. Michele, Preconditioning techniques for large linear systems: a survey, J. Comput. Phys. 182 (2) (2002) 418–477.
[5] J. Crank, P. Nicolson, A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type, Math. Proc. Cambr. Philos. Soc. 43 (1) (1947) 50–67.
[6] J.M. Dennis, R.D. Loft, Refactoring scientific applications for massive parallelism, Numerical Techniques for Global Atmospheric Models, vol. 80 (2011) 539–556.
[7] V. Eijkhout, Beware of unperturbed modified incomplete factorizations, Iter. Methods Linear Algebra 58 (1991) 3–591.
[8] I. Epicoco, S. Mocavero, F. Macchia, M. Vichi, T. Lovato, S. Masina, G. Aloisio, Performance and results of the high-resolution biogeochemical model PELAGOS025 within NEMO, Geosci. Model Dev. Discuss. 8 (12) (2015) 10585–10625.
[9] J. Fenlason, R. Stallman, GNU gprof, 2000, pp. 1–48.
[10] O. Fuhrer, C. Osuna, X. Lapillonne, T. Gysi, M. Bianco, A. Arteaga, T.C. Schulthess, Towards a performance portable, architecture agnostic implementation strategy for weather and climate models, Supercond. Front. Innov. 1 (1) (2014) 45–62.
[11] S.M. Griffies, A.J. Adcroft, Formulating the Equations of Ocean Models, 2008, pp. 281–318.
[12] P. Hanappe, A. Beurivé, F. Laguzet, L. Steels, N. Bellouin, O. Boucher, Y.H. Yamazaki, T. Aina, M. Allen, FAMOUS, faster: using parallel computing techniques to accelerate the FAMOUS/HadCM3 climate model with a focus on the radiative transfer algorithm, Geosci. Model Dev. Discuss. 4 (2011) 1273–1303.
[13] T. Henderson, J. Middlecoff, J. Rosinski, P. Madden, Experience Applying Fortran GPU Compilers to Numerical Weather Prediction, 2012, pp. 34–41.
[14] B. Hess, C. Kutzner, D. Van Der Spoel, E. Lindahl, GRGMACS 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation, J. Chem. Theory Comput. 4 (3) (2008) 435–447.
[15] J. Legaux, S. Jubertie, F. Loulergue, Experiments in parallel matrix multiplication on multi-core systems, in: Y. Xiang, I. Stojmenovic, B.O. Apduhan, G. Wang, K. Nakano, A. Zomaya (Eds.), Algorithms and Architectures for Parallel Processing. ICA3PP 2012. Lecture Notes in Computer Science, Vol. 7439, Springer, Berlin, Heidelberg, 2012.
[16] H. Jin, M. Frumkin, The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance, 1999.
[17] J. Labarta, J. Gimenez, E. Martínez, P. González, H. Servat, G. Llort, X. Aguilar, Scalability of Tracing and Visualization Tools, 2005.
[18] L. Jeffrey, Benefits of investing in weather forecasting research: an application to supercomputing, Yuejiang Acad. J. 2 (2010) 18–39.
[19] M.M.m. Made, H.A. van der Vorst, ParIC: A Family of Parallel Incomplete Cholesky Preconditioners, vol. 1823, 2000, pp. 89–98.
[20] G. Madec, NEMO Ocean Engine, No. (27), 2015.
[21] P.D. Meyer, A.J. Valocchi, S.F. Ashby, P.E. Saylor, A numerical investigation of the conjugate gradient method as applied to three-dimensional groundwater flow problems in randomly heterogeneous porous media, Water Resour. Res. 25 (6) (1989) 1440–1446.
[22] S. Joussaume, B. Lawrence, J. Mitchell, R. Budich, J. Marotzke, Infrastructure strategy European earth system modelling community 2012-2022, ENES Report Series 1 (2012), 33 pp.
[23] R.L. Naff, J.D. Wilson, A comparison of preconditioning techniques for parallelized PCG solvers for the cell-centered finite-difference problem, XVI International Conference on Computational Methods in Water Resources, No. (1) (2006) 18–22.
[24] N.H. Naik, V.K. Naik, M. Nicoules, Parallelization of a class of implicit finite difference schemes in computational fluid dynamics, Int. J. High Speed Comput. 5 (1) (1993) 1–50.
[25] C. Prodhomme, L. Batté, F. Massonnet, P. Davini, O. Bellprat, V. Guemas, F.J. Doblas-Reyes, Benefits of increasing the model resolution for the seasonal forecast quality in EC-Earth, J. Climate 29 (24) (2016) 9141–9162.
[26] F.J.L. Reid, NEMO on HECToR A dCSE Project, 2009.
[27] B. Rockel, A. Will, A. Hense, The regional climate model COSMO-CLM (CCLM), Meteorol. Zeit. 17 (4) (2008) 347–348.
[28] C. Rousset, M. Vancoppenolle, G. Madec, T. Fichefet, S. Flavoni, A. Barthélemy, R. Benshila, J. Chanut, C. Levy, S. Masson, F. Vivier, The Louvain-La-Neuve sea ice model LIM3.6: global and regional capabilities, Geosci. Model Dev. 8 (10) (2015) 2991–3005.
[29] A.F. Shchepetkin, J.C. McWilliams, The regional oceanic modeling system (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model, Ocean Modell. 9 (4) (2005) 347–404.
[30] K. Teranishi, P. Raghavan, A hybrid parallel preconditioner using incomplete cholesky factorization and sparse approximate inversion, in: Domain Decomposition Methods in Science and Engineering XVI, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 755–762.
[31] Tools@bsc.es. EXTRAE User Guide.
[32] M. Vancoppenolle, T. Fichefet, H. Goosse, S. Bouillon, G. Madec, M.A.M. Maqueda, Simulating the mass balance and salinity of Arctic and Antarctic sea ice. 1. Model description and validation, Ocean Modell. 27 (1–2) (2009) 33–53.
[33] P. Wang, Y. Tony Song, Y. Chao, H. Zhang, Parallel computation of the regional ocean modeling system, Int. J. High Perform. Comput. Appl. 19 (4) (2005) 375.
[34] T. Wilhelmsson, Parallelization of the HIROMB Ocean Model, 2002.

Oriol Tintó Prims is a PhD candidate researcher at the Performance Team inside the Computational Earth Sciences group from the Earth Sciences Department of the Barcelona Supercomputing Center. He is following the PhD program from the Computer Architecture and Operating Systems group at the Universitat Autònoma de Barcelona, with who he is actively collaborating. His background is Bs in Physics and a Ms in Mathematical Modeling and his current research topics include High Performance Computing, Earth System Models, performance analysis, performance optimization and mixed precision algorithms.

Miguel Castrillo holds an MSc in computer science from the University of León. Having more than five years of experience as software analyst and developer for different companies in the private sector, he joined the Computational Earth Sciences group at the Earth Sciences department of the Barcelona Supercomputing Center (BSC) in 2012, where he has been specializing in HPC and Earth Sciences modelling. His extensive expertise in the sector ranges from HPC data management and visualization tools, to parallel applications performance. He developed the CALIOPE air quality system mobile application, winner of the European Commission MYGEOSS project (2015–2016) for innovative applications using open data. In the last five years he has been intensely focused on HPC performance and model workflows, being involved in the IS-ENES2 and ESiWACE European projects and collaborating with the EC-Earth and NEMO models development teams. Currently he is head of the Models and Workflows Team in the Earth Sciences Department, as well as permanent member of the NEMO HPC working group and EC-Earth technical group.

Mario C. Acosta is a postdoctoral researcher and the leader of the Performance Team in the Computational group at the BSC (Barcelona Supercomputing Center) Earth Science Department. He obtained his PhD from University of Granada (Spain) in 2015, on High Performance Computing applied to Earth System Modeling. This expertise includes wide knowledge in numerical models (governing equations, numerical algorithms and computational implementation) and how to adapt them efficiently to actual and new HPC resources.

Oriol Mula-Valls holds a BSc in Computer Science by the Universitat Autonònoma de Barcelona and a MSc by the Universitat Politècnica de Catalunya. After a period in the private sector, he started working at the Institut Català de Ciències del Clima (IC3) where he was developing Autosubmit and managing the infrastructure as well as the cluster. After co-leading the Computational Earth Sciences group he moved back to the private sector. He currently works at HPCNow! and actively collaborates with the Barcelona Supercomputing Center Earth Science department. His main research interests are HPC, workflows and infrastructure.

Kim Serradell Maronda is currently managing the Computational Earth Sciences group in the Earth Sciences department at Barcelona Supercomputing Center (BSC-CNS). In the last years, he has been in charge for the system administration of all the computational resources of the department and he was also responsible of supervising the operational runs of the NMMB/BSC-CTM model and CALIOPE Air Quality System at BSC. In that sense, he was also involved in the analysis of the models to improve their performance and developed strong skills of compilation and scripting. Furthermore, he's focused in deploying different earth system models (dust transport, climate or weather forecast) required by the department in a wide range of HPC architectures. He succeeds porting different these models in next HPC architectures like Montblanc cluster (ARM Based). He applied with success these skills in IS-ENES2, SDS-WAS or ESiWACE projects.

Ana Cortés received both her first degree and her PhD in Computer Science from the Universitat Autònoma de Barcelona (UAB), Spain, in 1990 and 2000, respectively. She is currently associate professor of Computer Science at the UAB, where she is a member of the High performance Computing Applications for Science and Engineering of the Computer Architecture and Operating Systems Department. Her current research interests concern performance engineering of high performance computing environmental sciences applications.

Francisco Doblas-Reyes started working on climate variability at the Universidad Complutense de Madrid (Spain) in 1992, where he did his PhD. He then worked as a postdoc in Météofrance (Toulouse, France), at the Instituto Nacional de Técnica Aerospacial (Torrejón, Spain) and for ten years at the European Centre for Medium-Range Weather Forecasts (Reading, UK). He led the Climate Forecast Unit at the Institut Català de Ciències del Clima (IC3) from 2010 to 2015. He is currently the head of the Department of Earth Sciences of the Barcelona Supercomputing Center (BSC-CNS). The Department hosts more than 70 engineers, physicists, mathematicians and social scientists who try to bring the latest developments in supercomputing and data analysis to provide the best information and services on climate and air quality. He is author of more than 130 peer-reviewed papers (h index 36, scopus), member of several international scientific committees and supervisor of several postdocs, engineers and two PhD students.