Universitat Autònoma de Barcelona

NEMO: computational challenges in ocean simulation

Oriol Tintó Prims

Universitat Autònoma de Barcelona

Escola dEnginyeria

Departament dArquitectura de Computadors i Sistemes Operatius

NEMO: computational challenges in ocean simulation

Thesis submitted by

Oriol Tintó Prims

in fulfillment of the requirements for the doctoral degree from

Universitat Autònoma de Barcelona

advised by

Dra. Ana Cortés Fité

Dr. Mario C. Acosta

Dr. Francisco J. Doblas-Reyes

Barcelona, Spain, October 2019

NEMO: computational challenges in ocean simulation

Thesis submitted by Oriol Tintó Prims in fulfillment of the requirements for the doctoral degree from Universitat Autònoma de Barcelona. This work has been developed under RD 99/2011 in the Computer Science doctoral program and presented to the Computer Architecture & Operating Systems Department at the Escola d'Enginyeria of Universitat Autònoma de Barcelona. This thesis was advised by Dr. Ana Cortés Fité, Dr. Mario C. Acosta and Dr. Francisco J. Doblas-Reyes.

Advisors signature

Author signature

Tutor signature

NEMO: computational challenges in ocean simulation

Resum

Els oceans juguen un paper molt important a l'hora de modular la temperatura de la Terra absorbint, emmagatzemant i transportant l'energia que ens arriba del sol. Entendre millor la dinàmica dels oceans ens pot ajudar a millorar les prediccions meteorològiques i a comprendre millor el sistema climàtic, dues qüestions d'especial rellevància per la societat. Els models oceànics han esdevingut eines extremadament útils, ja que són un marc sobre el qual s'ha pogut construir coneixement. Utilitzant ordinadors ha sigut possible resoldre numèricament les equacions que descriuen la dinàmica dels oceans, i millorant com els models oceànics exploten els recursos computacionals, podem reduir el cost de les simulacions alhora que fem possibles nous desenvolupaments que milloraràn la qualitat científica dels models. Enfrontant els reptes computacionals de la simulació oceànica podem contribuir en camps que tenen un impacte directe en la societat mentre reduïm el cost dels nostres experiments. Per ser un dels principals models oceànics i el principal model col·laboratiu europeu, la tesis s'ha centrat en el model NEMO (Nucleus for European Modelling of the Ocean). Per trobar una manera de millorar el rendiment computacional dels models oceànics, un dels objectius inicials va ser entendre millor el seu comportament computacional. Per tal de fer-ho, es va proposar una metodologia d'anàlisis, posant especial atenció en les comunicacions entre processos. Utilitzada amb NEMO, la metodologia va ajudar a resaltar diverses ineficiències en la implementació, que un cop sol·lucionades van portar a una millora del 46-49% en la velocitat màxima del model, tot millorant la seva escalabilitat. Aquest resultat ilustra que aquest tipus d'anàlisis poden ajudar molt als desenvolupadors de models a adaptar-los tot mostrant l'origen dels problemes que pateixen. Un altre dels problemes detectats va ser que l'impacte d'escollir una descomposició de domini concreta havia estat molt subestimat, ja que en certes circumstàncies el model triava una descomposició sub-optima. Tenint en compte els factors que fan que una descomposició concreta afecti el rendiment del model, es va proposar un mètode per fer una selecció òptima. Els resultats mostren que parant atenció a la descomposició no només es poden estalviar recursos sinó que la velocitat màxima del model també se'n beneficia, arribant al 41% de millora en alguns casos. Després dels èxits aconseguits en la primera part de la tesis, arribant a doblar la velocitat màxima del model, l'atenció es va posar sobre els algoritmes de precisió mixta. Idealment, un ús adequat de la precisió numèrica ha de permetre millorar el rendiment computacional d'un model sense perjudicar-ne l'exactitud dels resultats. Per tal d'aconseguir-ho en models oceànics, es va presentar un mètode que permet determinar quina és la precisió necessària en cada una de las variables d'un codi informàtic.

El mètode es va posar a prova amb NEMO i ROMS (Regional Ocean Modelling System) mostrant que en ambdós models la major part de les variables pot utilitzar sense problema menys precisió que els 64-bits habituals, mostrant que potencialment els models oceànics es poden beneficiar molt d'una reducció de la precisió numèrica. Finalment, durant el desenvolupament de la tesi es va observar que degut a la no-linealitat dels models oceànics, determinar si un canvi en el codi informàtic perjudica o no la qualitat dels resultats esdevé molt complicat. Per solucionar aquest problema, es va presentar un mètode per verificar els resultats de models no-lineals. Encara que les contribucions que donen forma a aquesta tesis han sigut diverses, conjuntament han ajudat a identificar i combatre els reptes computacionals que afecten els models oceànics. Aquestes contribucions no només han resultat en quatre publicacions i múltiples participacions en activitats de divulgació sinó que també han resultat en la contribució al codi informàtic de NEMO i del consorci EC-Earth. Aquest fet es tradueix en que els resultat de la recerca realitzada a hores d'ara ja estan tenint un impacte positiu en la comunitat, ajudant als usuaris dels models a estalviar recursos i temps. A més a més, aquestes contribucions no només han ajudat a millorar significativament el rendiment computacional de NEMO sinó que han sobrepassat l'objectiu inicial de la tesis i poden ser fàcilment transferibles a altres models computacionals.

Paraules clau: Computació d'altes prestacions, simulació, oceà, NEMO, ROMS, precisió mixta

NEMO: computational challenges in ocean simulation

Abstract

The ocean plays a very important role in modulating the temperature of the Earth through absorbing, storing and transporting the energy that arrives from the sun. Better understanding the dynamics of the ocean can help us to better predict the weather and to better comprehend the climate, two topics of special relevance for society. Ocean models had become an extremely useful tools, as they became a framework upon with it was possible to build knowledge. Using computers it became possible to numerically solve the fluid equations of the ocean and by improving how ocean models exploit the computational resources, we can reduce the cost of simulation whilst enabling new developments that will increase its skill. By facing the computational challenges of ocean simulation we can contribute to topics that have a direct impact on society whilst helping to reduce the cost of our experiments. Being the major European ocean model and one of the main state-of-the-art ocean models worldwide, this thesis has focused on the Nucleus NEMO. To find a way to improve the computational performance of ocean models, one of the initial goals was to better understand their computational behaviour. To do so, an analysis methodology was proposed, paying special attention to inter-process communication. Used with NEMO, the methodology helped to highlight several implementation inefficiencies, whose optimization led to a 46-49% gain in the maximum model throughput, increasing the scalability of the model. This result illustrated that this kind of analysis can significantly help model developers to adapt their code highlighting where the problems really are. Another of the issues detected was that the impact of the domain decomposition was alarmingly underestimated, since in certain circumstances the model's algorithm was selecting a sub-optimal decomposition. Taking into account the factors that make a specific decomposition impact the performance, a method to select an optimal decomposition was proposed. The results showed that that by a wise selection of the domain decomposition it was possible not only to save resources but also to increase the maximum model throughput by a 41% in some cases. After the successes achieved during the first part of the thesis, that allowed an increase of the maximum throughput of the model by a factor of more than two, the attention focused on mixed-precision algorithms. Ideally, a proper usage of numerical precision would allow to improve the computational performance without sacrificing accuracy. In order to achieve that in ocean models, a method to find out the precision required for each one of the real variables in a code was presented. The method was used with NEMO and with the Regional Ocean Modelling System showing that in both models most of the variables could use less than the standard 64-bit without problems. Last but not least, it was found that being ocean models nonlinear it was not straightforward to determine whether a change made into the code was deteriorating the accuracy of the model or not. In order to solve this problem a method to verify the accuracy of a non-linear model was presented. Although the different contributions that gave form to this thesis have been diverse, they helped to identify and tackle computational challenges that affect computational ocean models. These contributions resulted in four peer-reviewed publications and many outreach activities. Moreover, the research outcomes have reached NEMO and EC-Earth consortium codes, having already helped model users to save resources and time. These contributions not only have significantly improved the computational performance of the NEMO model but have surpassed the original scope of the thesis and would be easily transferable to other computational models.

Keywords: High Performance Computing, ocean simulation, NEMO, ROMS, mixed-precision

Contents

List of Figures

List of Tables

Awnoledgements

The studies included in this PhD-Thesis were supported by the Spanish Ministerio de Ciencia, Innovación y Universidades through the grant BES-2015-074679.

I would like to express my sincere gratitude to my advisors Dr. Ana Cortés Fité, Dr. Francisco J. Doblas-Reyes and Dr. Mario C. Acosta Cobos for their guidance and support during my thesis. When I was in the middle of a tough year during my master's, Anna gave me the opportunity to have an internship at IC3, and to continue with my research starting this thesis. Without her help this thesis would not exist. At the beginning at IC3 and latter at the Earth Sciences department of the Barcelona Supercomputing Center I could witness how Paco made a clear and brilliant demonstration on how to build a team (a very big team) of amazing researchers, building an environment that without any doubt made my work many times easier and a lot more interesting. After we finally got installed at BSC, Mario arrived to lead the performance team. Since then he helped with my research. Thank you, specially for your comprehension and closeness. I've been your first advisee and I hope that your experience has been as positive as mine, and that what we learned together during this process will help you with your future career in research.

During the thesis I had the opportunity to meet many colleagues at the Universitat Autònoma de Barcelona, at the IC3 and at Barcelona Supercomputing Center. It is true that the Earth Sciences Department has grown so much during my PhD that it has been hard to really interact as much as i'd liked with all of them, but anyway it has been a great experience and a nice opportunity to meet so many researchers from all around the world with so many interesting things to share. To all of them, thank you for making my time better.

From all the colleagues that I've met during these years I would like to specially thank a few of them. After my landing at IC3, Oriol Mula helped me a lot by guiding me when HPC and ocean simulation were new for me, he taught me that every question has an answer if you search well enough, and that any bar is too low regarding to the level of a joke, thank you!

When we finally moved to BSC, I had the luck to have a seat near Miguel and Kim. I've seen Kim responsibilities grow a lot having now a big team to lead, thank you for helping me personally but above all for making CES the great group that is now. All these years I've been trying hard to distract Miguel but somehow he still manages to be efficient, he hasn't been officially an advisor but for sure he advised me wisely, many thanks for your help and above all for our friendship.

I would like to thank all the colleagues that helped me somehow during the writing of my publications and my thesis. During the time that we had Alícia in our team, I've always appreciated his kindness and attention to everybody's well being, thank you!! Even though Stella hasn't been too much time around, her feedback definitely helped me, thank you! I would like to thank Marcus too, I appreciate the time and effort that you spent helping me make my English a little bit less scary, for sure I owe you a few beers.

During the PhD I had the opportunity to spend a few months working at the University of California Santa Cruz. Thank you Andy for giving me the opportunity to do research in such a beautiful place and for your help and feedback during my time there. I would like to thank too my amazing housemate during my time there. Maryanne, your energy and impulse are out of this world, thank you for making my California experience so awesome, for teaching me the baseball rules and for all the volleyball games at the beach. I'm afraid that I've already forgotten the rules of the card game we used to play with John, which I would like to thank too. I hope that I'll see you again soon.

I would like to thank too all the researchers from other institutions that have immensely helped to develop my ideas with fruitful discussions and valuable feedback.

Being raised in a small town, your childhood friends end up being a huge family. Thank you for supporting me and being there even though I'm sometimes a little bit "know-it-all". I hope that we will keep the many positive things of being such a huge group of friends while improving the few things that we still need to get better, like showing ourselves how much we appreciate each other.

I would like to thank too my teammates, coaches and members of Basquet Berga which had me playing basketball all these years even though I couldn't train as much as I'd like to.

I would like to thank my colleagues from Physics at UAB, it's good to see that everybody in this amazing group of people goes one with their own paths but we still manage to meet and be a part of each others lives. I should be dedicating a full chapter to thank Linus and Pau for everything we shared and everything I'm grateful for, but you two already know that I'm not the prolific writer that I should be. I will only say that I really appreciate the time we spent together eating braves, riding waves and letting my crazy mind fly discussing about dystopian worlds. I hope that you already know how much I appreciate you.

I would like to thank too the amazing flatmates that I had during all these years. When I first came to Barcelona to live with Joan and Marc I was not in my best moment, and living with them helped me to recover my optimistic self. I value a lot the time we lived together, thank you! After my California dream and landing back to Barcelona, Marc had the generosity to find us a new place (thank you very much) and we had the opportunity to meet new flatmates. Oriol's social life has been legen(wait-for-it)dary and his determination following what makes him happy still amazes me, thank you for being such a great example. Finally Susanna had the courage to put plants inside our flat and the determination to keep them alive, thank you! I hope that I have been as half as good as a flatmate as you are.

I want to thank my girlfriend Marta for sharing these years with me. Thank you for making this time so much better and for understanding my headaches with the thesis. You've helped me much more than you think, I love you!

I would like to thank my family for their support, especially during the hard times. To my grandfather Lluís for being such a great example. To my father Joan for making me feel unconditionally loved and supported.

To my mother, whose absence has been the hardest thing of all these five years, I miss you so much. Your memories helped me to go on, I know that you would have been proud of who I am today, and this belief alone is more than enough for me.

I would like to thank all the people that helped me to finish this thesis but beyond that I would like to thank all the people that made my time worth living.

Chapter 1

Motivation and Objectives

The birth of computational science has been one of the most game-changing achievement of science in the XX century [1]. The capacity of numerically solving mathematical models made possible the virtual experimentation of systems that are too big, too small or simply too complicated to be treated with classical experimentation methods. These fields include many diverse topics such as DNA folding, physical cosmology or the Earth climate.

Computers have made possible what was previously thought as impossible, allowing to numerically solve problems that were simply unattainable before. Although this has opened a new window of opportunities, it has its own limitations. Computers have a finite amount of memory and can perform a limited number of operations over a finite time, meaning that the kind of problems that can be solved using a computer will still be limited by the capacity of the computer itself. Since there are problems that can only be solved beyond a certain threshold of computational power, sometimes not only a quantitative change is brought about by a more powerful computer but a qualitative one.

A good example is weather prediction. The hypothesis that the evolution of the atmosphere's state could be solved numerically (and thus weather prediction is possible) was formulated several years before the first computer was built. As soon as it was built the first simulations took place [1] proving that it was possible, but with the limited computing power the length of time it took to complete the simulations made them useless for weather forecasting. However, when the computational power enabled weather simulations to be produced fast enough it was a quantitative difference in computing power that resulted in a qualitative difference in the worth of the information provided by the simulations.

Since then, the skill predicting the weather has been improving at a rate of about one day per decade becoming an extremely useful resource for society, where the benefits largely outweigh its costs. The improvement of the weather models came from several developments (higher model complexity, ensemble methods, data assimilation) which all required additional computational power [2].

The weather refers to the state of the atmosphere, but the atmosphere is not an isolated system. The atmosphere is just one of several parts of the Earth system, whose usual description also includes the hydrosphere, the cryosphere, the lithosphere and the biosphere (see Figure 1.1). These systems interact in many ways, they are interconnected and they interchange energy and mass.

For time-scales up to few days, these interactions are important but the dynamics of the other systems are much less relevant than atmosphere's because in general are much slower. However, there are short time-scale phenomena in these systems that can be relevant: small scale ocean-eddies that can happen in short time-scales can deeply affect atmosphere circulation [3], and in extreme weather events like hurricanes, the sea-air interaction plays a major role in particular to dictate the trajectory of the cyclones and ocean dynamics became relevant even in short time-scales [4]. For longer time-scales, however, the dynamics of the other parts of the Earth system become more important. To simulate how the atmosphere and the whole Earth system evolve in longer time scales, understanding the dynamics of the other systems and their interactions is essential.

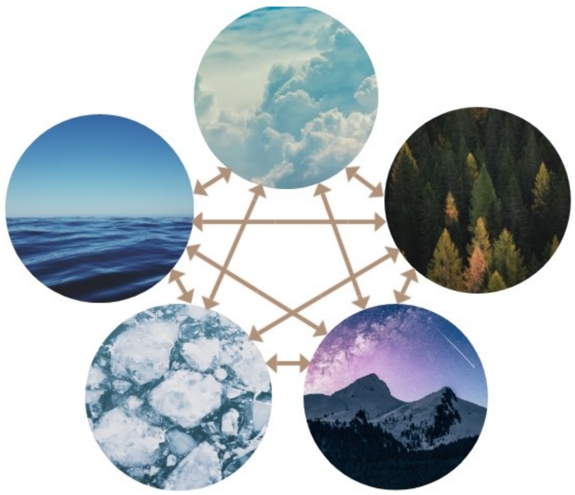

Figure 1.1: "The five components of the climate system and their interactions" by Femkemilene [CC BY-SA 4.0]

One of the components that has particular relevance in the Earth System is the ocean. The oceans cover up to 71% of the Earth surface, and its high heat capacity and fluidity makes it able to absorb, store, and transport the energy that our planet receives from the sun. The ocean and the atmosphere are the two fluid components of the Earth system, but the fact that the ocean heat capacity is more than 1000 times bigger than that of the atmosphere means that the ocean has a dominant role in moderating the temperature of the planet, and therefore having major relevance for the climate [5].

While it is widely known that the understanding of the weather is hugely valuable for society [2], climate science is raising a lot of interest in the general public due to the climate crisis caused by anthropogenic greenhouse gases.

In recent decades, humanity has gone from seeing our own planet Earth as something extremely big and resilient to realize that we, as a species, are already altering and reshaping it at a tremendous rate. We have realized that our actions have the potential to destroy ecosystems, drive species to extinction [6] [7], boost desertification [8], and even change the climate of the entire planet [9]. Climate change represents a great challenge, not only for the huge threat that this represents for the environment but also the severe social [10], economic [11] and life threatening consequences [12]. Two approaches are needed in order to face the climate crisis. On one hand we need to mitigate our impact in the climate by reducing our emissions of greenhouse gases, and on the other hand we need to adapt to the changes in climate that are already happening.

Since a better capacity to simulate the ocean dynamics is necessary to improve our skill in predicting the weather and our understanding and capacity to predict climate, improving ocean models is of general interest since both fields have and will have a direct impact on society.

There's not one single method to improve a dynamical ocean model, but most will imply an increased computational cost. Since having more resources available will make getting rid of approximations or increasing the complexity of the model by including new phenomena possible, making more resources available is a good an necessary way to improve a model. Being able to use more computational power its not only a matter of having a more powerful computer, it depends also on how the system is used. Then, it is possible to increase the computational resources available by improving the model's computational performance. Moreover, at times it is not only possible but necessary, since computational models might be unable to use more resources without the proper adaptation [13]. Also, an improvement of the computational performance of the model is not only necessary to improve the model skill, but it can also reduce the resources needed to run simulations, reducing its carbon fingerprint and the required budget.

As a consequence, improving the computational performance of the ocean models will not only enable new kinds of experiments but also will allow to carry out current simulations with a reduced cost. By facing the computational challenges in ocean simulation we can contribute to topics that are very relevant for society whilst helping to reduce the impact of our experiments.

There are many computational models that numerically solve the dynamics and thermodynamics of the ocean (see Table 1.1). Although these models have many differences, the theoretical basis of the ocean dynamics is the same, in the sense that they solve the Navier-Stokes differential equations over the ocean. Their main differences are the specific numerical methods used to solve the same problem, and how these methods have been implemented into a code.

Acronym	Name	nº projects
MOM	GFDL Modular Ocean Model	8
NEMO	Nucleus for European Modelling of the Ocean	5
POP	The Parallel Ocean Program	2
MICOM	Miami Isopycnic Coordinate Ocean Model	1
MPIOM	MPI Ocean Model	1
CanOM	Canadian Ocean Model	1
LICOM	LASG/IAP Climatesystem Ocean Model	1
Russel Ocean	-	1
COCO	CCSR Ocean Component Model	1
MRI-COM	Meteorological Research Institute Community Ocean Model	1
Total		22

Table 1.1:: List of ocean models used in CMIP 5.

In order to numerically solve differential equations using a computer, the domain and the equations first need to be discretized. In order to discretize the domain, we need to describe its physical quantities of the domain on a finite number of points. To discretize the differential equations, the derivatives are approximated using finite differences. There is not one single way to approximate a derivative but many, which will require information from different points and will have different truncation errors (see Figure 1.2 and Table 1.2). For spatial derivatives of ocean models, the most commonly used scheme is the second-order central scheme which for a certain point i will require information from adjacent points $i-1$ and $i+1$.

Forward difference: $\frac{\partial f'(x)}{\partial h} = \frac{f(x+h)-f(x)}{h} + O(h)$

Backward difference: $\frac{\partial f'(x)}{\partial h} = \frac{f(x)-f(x-h)}{h} + O(h)$

Central difference: $\frac{\partial f'(x)}{\partial h} = \frac{f(x+h)-f(x-h)}{2h} + O(h^2)$

Table 1.2:: Forward, backward and central finite difference schemes.

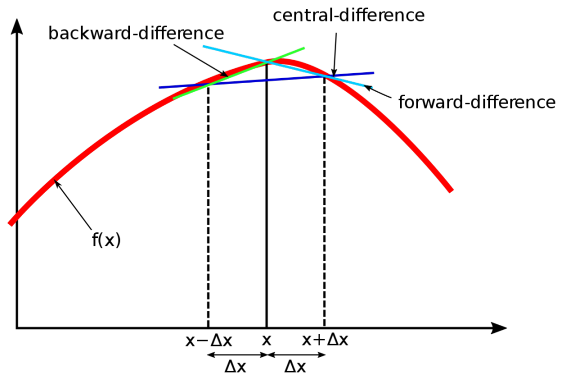

Figure 1.2: Visual representation of three different finite difference approximations of derivative in a one-dimensional function. Figure by Kakitc [CC BY-SA 4.0]

To parallelise the models so that they can be used in parallel systems, domain decomposition strategies are often used. Domain decomposition consists of splitting the original domain in smaller parts that are solved separately, i.e. having exactly the same code but using a different sub-domain. This method usually requires the different sub-domains to interchange information to fulfill their boundary conditions. For example: when a sub-domain needs to compute a spatial derivative of a grid-point of the border, it will require information from the neighbouring sub-domain.

From several existing ocean models, this thesis has focused on the Nucleus for European Modeling of the Ocean: NEMO. NEMO is a framework that includes several sub-models related with the ocean. Its nucleus is the OPA engine, a computational model that simulates the dynamics and thermodynamics of the water ocean. Other widely used components are the sea-ice model (at the beginning of the thesis was LIM and now it's called SI3) or the tracer/biogeochemistry (TOP-PISCES). The development of some of the parts that now integrate NEMO date back to mid 80s, and were originally programmed using Fortran in its Fortran77 standard. NEMO has a large network of users, and it is used in many projects, being the main European ocean model. A wider description can be found in Chapter 2.

And how one can start improving the computational performance of an ocean model? Since its creation, the evolution of ocean models has been influenced by the computer systems available. To understand the kind of issues that can be limiting the computational performance of the model it is good to have a background about how computer systems and models evolved.

The decades leading up to 2005 was a period in which the improvement of chips was mainly propelled by the miniaturization of transistors, which allowed clock frequency to increase providing more computing power. During this time, every new generation of processes managed to reduce the dimension of the transistors by 30%, leading to an area decrease of 50%, allowing the signal delays between transistors to be reduced by 30% and increasing the frequency of the clock by 40%. It also allowed to reduce the required voltage and the energy consumption by 65%. In practice, usually what was done was to maintain the size and the energy consumption of the chip by increasing the number of transistors [14]. However, some physical limitations prevented these changes to be held proportionally in much smaller scales. At these scales, electric current leakage causes the chip to heat up, increasing the energy costs. In consequence, building smaller transistors leads to a higher energy density that needs to be dissipated, making this a limiting factor that doesn't have an obvious solution. Therefore, the trend of the last decades that mostly based the increment of speed to the increment of the clock frequency could not continue for much longer. To find an alternative way to continue improving processors computing power, the paradigm had to switch to explicit parallelism. Instead of having a faster processing unit, manufacturers started to include multiple processing units in their chips. This is true for individual chips but even more so for supercomputers. What makes a supercomputer special compared to any regular desktop computer is the fact that many individual-computers can cooperate to complete a task. This cooperation can happen due to a high-speed interconnection that allows the components of the supercomputer to interchange information efficiently. The current trend of increasing the number of processing units forming a supercomputer has been ongoing for decades and it is expected to continue in the future. This implies that in order to exploit HPC systems one must be able to exploit parallelism, a feature that will gain importance in the future.

Most state-of-the-art models had been in development for several decades with the participation of many scientists and engineers. The systems that were available back then were several orders of magnitude less powerful than today's, and in consequence also the kind and size of experiments that were taking place back then were several orders of magnitude less demanding. Nevertheless, even though the evolution of the computer systems has been huge, there are parts of the code that hadn't changed, and while this is not necessarily bad, sometimes it can cause problems. For example, an algorithm that was designed to solve a problem of a certain size under certain conditions, and which was working optimally under these conditions, when used in different conditions may struggle to exploit the resources properly.

When a model that has been developed for a long time using a specific kind of architecture needs to be executed in a different system, it usually faces issues that prevent it to exploit the new system optimally. An example of this is what happened when the paradigm drifted from single-threaded processors to many-core. Converting a sequential algorithm to a parallel one is not always trivial, in fact it usually isn't. From the point of view of the applications it might be difficult to translate originally sequential algorithms into distributed ones, and the fact that it is not straightforward for developers to infer the computational behaviour of their codes prevents them from obtaining optimal implementations. High level programming languages allow the transformation of ideas into working codes with easiness, being a major step for productivity, but these high level programming language standards focus on more abstract concepts such as logic and mathematics than instructions and memory locations [15]. Interpreters and compilers have taken the responsibility of translating codes to the machine language, maximizing the efficiency of the programs. However, doing so developers have sacrificed understanding and control of what is happening at processor level. To complicate things further, programming models used to exploit parallel architectures can add an extra layer of complexity. If understanding how a computer code will behave is difficult, understanding how many instances of the same code running in parallel will interact is even more so. These points explain why state-of-the-art ocean models have computational inefficiencies to face, but knowing the specific challenges requires a deeper analysis of the model.

In a vast range of computational science fields researchers have had to find a compromise between what they wanted to simulate and what the computational resources allow them to do [16]. For those researchers, pushing the limits of what can be done using a computer by understanding the computational behaviour of their codes for the purpose of optimization has been crucial. Since many groups have had the necessity to understand the computational behaviour of their parallel applications, some of them opted for developing their own tools (see Table 1.3). Within these tools especially designed to help understand the behaviour of parallel applications there are the tools developed at the Barcelona Supercomputing Center, which has its own research group on performance tools. Its main tools are Extrae [17], Paraver [18] and Dimemas, which allow the collection of information during the execution, and a post-mortem visualization and manipulation of this data. The fact that those tools are very powerful, demonstrated in the past its usefulness in Earth Science codes and also the opportunity to collaborate with the tool developers made them the preferred choice in this thesis.

During a project that preceded this thesis an initial analysis of NEMO was performed and it had two main conclusions. Firstly, the computational time invested in the different regions of the code was quite evenly distributed, where no single region took up a considerable part of the computation. Secondly, inter-process communication seemed to play a key role in preventing the code to efficiently use more computational resources [19].

Therefore, as the objective of this thesis is to face the computational challenges of ocean simulation working with the NEMO model, two main objectives were pursued. Firstly, better understand the computational behaviour of the application, with an

Tool	Open Source
Arm MAP (Part of Arm Forge)	No
Arm Performance Reports	No
Dimemas	Yes
Extra-P	Yes
mpiP	Yes
Open—SpeedShop	Yes
Paraver	Yes
Periscope	Yes
Scalasca	Yes
TAU	Yes
Vampir	No

Table 1.3:: Parallel performance tools listed by the Virtual Institute of High Productivity Supercomputing.

special focus on communication. Secondly, to study ways to improve the computational performance of the model having in mind that to achieve a significant impact it needs to have an effect on most regions of the code.

To advance in the fulfillment of the thesis goals, the author has worked on several contributions, four of which have been published under a peer-reviewed procedure and are the main building blocks of this thesis document, where each occupies an entire chapter.

As stated before, the ability to efficiently use computer systems will be crucial for the progress of science. Since there was not a well established method to help model developers to understand the computational behaviour of Earth Science models that put emphasis on inter-process communication, the first of the contributions to the thesis proposes a method to do it. Inter-process communication plays a crucial role in massive-parallel systems since it requires the minimizing of the overhead of distributing the work along different processors. Using the developed methodology, it is possible to conclude that state-of-the-art ocean models still use algorithms that are not best suited for massive-parallel systems. Nevertheless, with a proper analysis methodology the issues can be characterized, leading to productive optimization that improved the efficiency of the model.

After the objective of the aforementioned objective was fulfilled, the focus of the thesis switched to specific challenges that could help us to improve the efficiency of the model. We kept in mind that the problem in question should be affecting the entire model.With this in mind, the second contribution emerged from the realization that the algorithm used to select a domain decomposition could result in a very inefficient choice, meaning that the specific division proposed by the algorithm could be much worse than a different decomposition using the same or even less resources. In addition to that, even though it is possible to exclude the land-only processes from the computation, the method to do so was not straightforward at all, making it difficult for users to select the best domain decomposition choice given a number of available resources. In this second contribution, this problem was solved by proposing a method to select a decomposition taking into consideration the possibility of discarding land-only sub-domains and the overlap.

The success in the first and the second contributions helped to gain the necessary confidence to attempt the challenge of exploring the numerical precision required to perform simulations. It is a risky proposal, since all previous attempts of reducing the numerical precision used in NEMO had failed. Due to the reluctancy to publish negative results in the scientific community, it was impossible to find documented details of these previous attempts. However, the success achieved by the European Centre for Medium-Range Weather Forecasts achieving significant computational performance improvement of the IFS model by reducing the numerical precision used [20] was enough motivation despite previous failures.

The main objective of adjusting the numerical precision is to achieve computational performance gains, by reducing the amount of data movement and to enabling a better exploitation of vector operations. However, a floating point representation of a real number is an approximation, and pushing further the approximation of numbers by reducing the amount of bits that we spend on it can have undesired effects. For this reason, when adjusting the numerical precision it is necessary to find a way to safely do it. In the third contribution , a method to automatically determine whether the variables in a Fortran code can use a desired precision is presented. The method was put to test not only with NEMO but also with ROMS, showing very promising results.

Although in the third contribution a method to determine which variables can use reduced precision was proposed, a way to determine whether the results of a simulation are good is still required. Discriminating the accuracy of a ill-conditioned non-linear model is not a trivial issue, and sometimes it requires an extended prior-knowledge about the specific field in order to be able to do it. In the fourth contribution, a method to verify the results of a computational model with a minimal field knowledge was proposed. This kind of method would be used along with the third contribution to achieve a Mixed-Precision version of NEMO, but the potential of such method goes beyond that. In many cases, computer scientists and performance engineers that could potentially have ideas to improve the computational performance of a scientific code struggle to know if the results are good once the bit-to-bit reproducibility is lost. Enabling them to know whether a code-change is acceptable or not can boost cooperation between computer engineers and computational scientists working in Earth Science.

As a summary, this thesis is presented as a compendium of publications, in which each one of the peer-reviewed publications occupy a chapter. The current chapter includes the motivations and objectives of the thesis, along with a brief summary of the contributions. Following this lines, chapter 2 contains a description of the NEMO model. Chapter 3 corresponds to the first contribution which addresses the issue of understanding the performance behaviour of a parallel Earth Science code, with special focus on inter-process communication. In chapter 4 a method to select the proper domain decomposition is proposed. The most important contribution of this thesis occupies chapter 5 and harnesses the issue of finding the required numerical precision in each part of an Earth Sciences code. Finally, in chapter 6 a method to verify the accuracy of computational models of non-linear systems is introduced. To finish, in the chapter 7 the conclusions and open lines can be found.

Chapter 2

Model Description

2.1 Introduction

The variety of phenomena happening at the ocean is astonishing. If an ambitious scientist pretends to simulate the ocean, definitely the best aim would be not to not start by trying to include all these phenomena but to start with a simple model on top of which more knowledge can be built. That's what happened with NEMO, at the beginning it was a model that tried to solve numerically the Navier-Stokes equations of the fluid ocean, as it was evolving more components were added becoming a framework for ocean modelling more than a model itself. NEMO has been used by many different users with many different purposes and that is the reason why several components have been incorporated along the time. The main core of the framework is the OPA model, a physical model that simulates the dynamics and thermodynamics of the water ocean. It solves the Navier-Stokes equations of the sea-water. Besides OPA, there are two more physical core engines, the model LIM for the sea-ice and the model TOP-PISCES for the passive tracers and biochemistry. Additionally, a two-way nesting system was integrated to allow adaptive mesh refinement (use more detailed grids in specific regions of interest) and few data assimilation interfaces too. Due to the complexity of each one of the components, this chapter will focus only on the two most used ones that are the most widely used.These two components are the ocean and sea-ice physical cores, OPA and LIM. OPA is the ocean physical core and therefore crucial in almost all the simulations performed with NEMO. LIM is the most-used sea-ice model in NEMO simulations, and has an important computational cost that makes it relevant for our study.

2.2 OPA

2.2.1 Mathematical model

OPA is the ocean physical core of NEMO. As it is described in the OPA reference documentation [21], it solves the primitive equations of the ocean, that fairly relate its behaviour. These primitive equations are mainly the Navier-Stokes equation plus the non-linear equation of state, that is used to couple the active tracers (temperature, and salinity) to the momentum equation. The resulting equations are further simplified using the following approximations:

- Spherical Earth hypothesis: the Earth is considered to be a sphere, so that gravity is parallel to the Earth's radius. (equator radius / polar radius $=$ 1.00346)
- Ocean shell hypothesis: Ocean depth is much smaller than the Earth radius. (max(ocean depth / earth radius) $= 0.172$ %)
- Turbulent closure hypothesis: the turbulent fluxes are parameterized. (Big source of uncertainty)
- Boussinesq hypothesis. (The approximation is extremely accurate)
- Hydrostatic hypothesis: the vertical momentum equation is reduced to a balance between the vertical pressure gradient and buoyancy force (this removes convective processes from the initial Navier-Stokes equations: they must be parameterized). (Problematic in fine resolutions (¡10km))
- Incompressibility hypothesis: the three dimensional divergence of the velocity vector is assumed to be zero. (condition: ocean depth h is small compared to a half of the wavelength λ of the ocean wave).

Applying these approximations, the resulting equations are:
Momentum balance:

$$\frac{\partial \mathbf{U}_h}{\partial t} = -\left[(\nabla \times \mathbf{U}) + \frac{1}{2}\nabla(\mathbf{U}^2)\right]_h - f\mathbf{k} \times \mathbf{U}_h - \frac{1}{\rho_0}\nabla_h p + \mathbf{D}^{\mathbf{U}} \tag{2.1}$$

Hydrostatic equilibrium:

$$\frac{\partial p}{\partial z} = -\rho g \tag{2.2}$$

Incompressibility:

$$\nabla \cdot \mathbf{U} = 0 \tag{2.3}$$

Heat conservation:

$$\frac{\partial T}{\partial t} = -\nabla \cdot (T\mathbf{U}) + D^T \tag{2.4}$$

Salt conservation:

$$\frac{\partial S}{\partial t} = -\nabla \cdot (S\mathbf{U}) + D^S \tag{2.5}$$

Equation of state:

$$\rho = \rho(T, S, p) \tag{2.6}$$

These equations fairly describe the behaviour of the ocean for scales that are bigger than a few kilometers in the horizontal scale, few meters in the vertical scale and several minutes for the temporal scale. However, there are other things that need to be considered in order to use them in a model. In continuation a brief summary of several of these issues can be found, a deeper description can be found in the NEMO reference publications [22].

Boundary conditions

The ocean is bounded by two surfaces, the sea-surface at the top and the sea-floor at the bottom. Through this two boundaries, the ocean interchanges energy (heat, momentum, . . .) and mass (precipitation, evaporation, . . .) , in interactions that can be with the land, the solid earth, the atmosphere and the sea-ice. Some of these interchanges can be negligible even in climate time-scales, but many others can not be neglected. The most relevant interaction with the land occurs due to the river runoff pouring fresh water into the ocean modifying its salinity. Even though this interaction is not relevant in short time-scales, in longer time-scales it is. In the boundary with the solid earth (sea-bottom), the heat and salt fluxes are very small and are usually neglected, becoming the boundary condition to have no fluxes across the border. For the momentum, the flow is also zero but there is an interchange of momentum due to frictional processes. These processes occur at sub-grid scales and thus have to be parametrized. The most important boundary interaction with the ocean occurs with the atmosphere. Through the atmosphere, energy and mass fluxes occur. The mass fluxes occur due the P - E (precipitation minus evaporation), the energy fluxes occur through heat and horizontal momentum (wind stress) interchanges. Finally, there are regions where the ocean also interacts with sea-ice. The sea-ice and the ocean interchange heat, salt, fresh water and momentum and also constrain the temperature in the interface to be at the freezing point. Horizontal Pressure Gradient: Given the hydrostatic approximation, the pressure can be calculated by adding the pressure at the surface plus the hydrostatic pressure due to the depth. The hydrostatic term is trivial but the surface term requires a more specific treatment. Currently, this term is handled in the model using the free surface formulation. Allowing the air-sea interface to move introduces the external gravity waves (EGWs), that are barotropic with a quite high phase speed. Depending on the physical processes of interest it is possible to filter these EGWs but if one wants to include them in the model it is necessary to use explicit time scheme with a very small time step or to rely on split-explicit methods with reasonably small time-step, which imply an increased computational cost.

Subgrid Scale Physics

In global ocean models, the primitive equations (ref to eq section) are used to simulate the behaviour of the ocean with a horizontal scale of the order of kilometers (state-of-the-art models run experimental setups of around 1 km and low resolution experiments use horizontal resolutions of ¿200 km), a vertical resolution of the order of meters for the finer layers and a time scale of the order of minutes. The effects of smaller scale motions need to be represented in terms of the large-scale patterns to close the equations. The name of this effects is usually called subgrid scale physics. The importance of the gravity produces a big difference between the horizontal and vertical directions, and for this reason these two are treated separately. In the vertical axis, the turbulence occur in scales that are always smaller than the model resolution and are never explicitly solved. The fluxes are assumed to be linearly dependant on the gradients of the large-scale quantities scaled by some coefficients that can be assumed to be either constant, a function of the local fluid properties or computed from a turbulent closure model. In the horizontal planes the things are a bit different, the lateral turbulence can be approximately divided between mesoscale and sub-mesoscale turbulence. If the horizontal resolution is sufficiently fine, the former can be explicitly simulated, while the latter will be never explicitly solved. Thus, the specific formulation will depend on whether the mesoscale eddies are resolved or not. Since parameterizations are a source of uncertainty, the use of finer resolutions can potentially reduce it.

2.2.2 Numerical description

To use this differential equations to simulate a given domain , first the equations are translated to a coordinate system that can adjust to the domain itself, and after that numerical methods are used in order to translate this differential equations to discrete difference equations that can be used to construct a computational code. The specifics of the methods used are described in this part of the chapter.

Tensorial formalism

Being able to solve the primitive equations in various curvilinear systems is important in order to overcome specific issues that appear when using specific systems of coordinates (polar singularity, regions of enhanced dynamics, ...). To do so the tensor formalism is adopted as a way of doing appropriate coordinate transforms between any multidimensional curvilinear coordinate system. Let (λ, ϕ, z) be the geographical coordinate system in which a position is defined by the latitude $\phi(i,j)$, the longitude $\lambda(i,j)$ and the distance from the centre of the earth $a + z(k)$ where a is the earth's radius and z the altitude above a reference sea level. The local deformation of the curvilinear coordinate system is given by $e1$, $e2$ and $e3$, the three scale factors:

$$e_1 = (a+z)\left[\left(\frac{\partial\lambda}{\partial i}cos\phi\right)^2 + \left(\frac{\partial\phi}{\partial i}\right)^2\right]^{1/2} \tag{2.7}$$

$$e_2 = (a+z)\left[\left(\frac{\partial\lambda}{\partial j}cos\phi\right)^2 + \left(\frac{\partial\phi}{\partial j}\right)^2\right]^{1/2} \tag{2.8}$$

$$e_3 = \left(\frac{\partial z}{\partial k}cos\phi\right) \tag{2.9}$$

This leads to the operators:

$$\nabla q = \frac{1}{e_1}\frac{\partial q}{\partial i}\mathbf{i} + \frac{1}{e_2}\frac{\partial q}{\partial j}\mathbf{j} + \frac{1}{e_3}\frac{\partial q}{\partial k}\mathbf{k} \tag{2.10}$$

$$\nabla\mathbf{A} = \frac{1}{e_1e_2}\left[\frac{\partial(e_2a_1)}{\partial i} + \frac{\partial(e_1a_2)}{\partial j}\right] + \frac{1}{e_3}\left[\frac{\partial a_3}{\partial k}\right] \tag{2.11}$$

$$\nabla\times\mathbf{A} = \left[\frac{1}{e_2}\frac{\partial a_3}{\partial j} - \frac{1}{e_3}\frac{\partial a_2}{\partial k}\right]\mathbf{i} + \left[\frac{1}{e_3}\frac{\partial a_1}{\partial k} - \frac{1}{e_1}\frac{\partial a_3}{\partial i}\right]\mathbf{j} + \frac{1}{e_1e_2}\left[\frac{\partial(e_2a_2)}{\partial i} - \frac{\partial(e_1a_1}{\partial j}\right]\mathbf{k} \tag{2.12}$$

$$\Delta q = \nabla\cdot(\nabla q) \tag{2.13}$$

$$\Delta\mathbf{A} = \nabla\cdot(\nabla\mathbf{A}) - \nabla\times(\nabla\times\mathbf{A})) \tag{2.14}$$

Where q is a scalar and $\mathbf{A} = (a1, a2, a3)$ a vector in the (i,j,k) coordinate system.

Using this operators to represent the prognostic equations using the tensor formalism, the final form of the equations becomes:

For the vector invariant form:

$$\frac{\partial u}{\partial t} = +(\zeta+f)v - \frac{1}{2e_1}\frac{\partial}{\partial i}(u^2+v^2) - \frac{1}{e_3}w\frac{\partial u}{\partial k} - \frac{1}{e_1}\frac{\partial}{\partial i}\left(\frac{p_s+p_h}{\rho_o}\right) + D_u^{\mathbf{U}} + F_u^{\mathbf{U}} \tag{2.15}$$

$$\frac{\partial u}{\partial t} = -(\zeta + f)u - \frac{1}{2e_2}\frac{\partial}{\partial j}(u^2+v^2) - \frac{1}{e_3}w\frac{\partial v}{\partial k} - \frac{1}{e_1}\frac{\partial}{\partial i}\left(\frac{p_s+p_h}{\rho_o}\right) + D_v^{\mathbf{U}} + F_v^{\mathbf{U}} \quad (2.16)$$

For the flux form:

$$\frac{\partial u}{\partial t} = +\left(f + \frac{1}{e_1 e_2}\left(v\frac{\partial e_2}{\partial i} - u\frac{\partial e_1}{\partial j}\right)\right)v - \frac{1}{e_1 e_2}\left(\frac{\partial(e_2 uu)}{\partial i} + \frac{\partial(e_1 vu)}{\partial j}\right) - \frac{1}{e_3}\frac{\partial(wu)}{\partial k} - \frac{1}{e_1}\frac{\partial}{\partial i}\left(\frac{p_s+p_h}{\rho_0}\right) + D_u^{\mathbf{U}} + F_u^{\mathbf{U}} \quad (2.17)$$

$$\frac{\partial v}{\partial t} = +\left(f + \frac{1}{e_1 e_2}\left(v\frac{\partial e_2}{\partial i} - u\frac{\partial e_1}{\partial j}\right)\right)u - \frac{1}{e_1 e_2}\left(\frac{\partial(e_2 uv)}{\partial i} + \frac{\partial(e_1 vv)}{\partial j}\right) - \frac{1}{e_3}\frac{\partial(wv)}{\partial k} - \frac{1}{e_2}\frac{\partial}{\partial j}\left(\frac{p_s+p_h}{\rho_0}\right) + D_v^{\mathbf{U}} + F_v^{\mathbf{U}} \quad (2.18)$$

Where ζ is the relative vorticity given by:

$$\zeta = \frac{1}{e_1 e_2}\left[\frac{\partial(e_2 v}{\partial i} - \frac{\partial(e_1 u}{\partial j}\right] \quad (2.19)$$

And p_s the surface pressure given by:

$$p_s = \begin{cases} \rho g \eta & \text{standard free surface} \\ \rho g \eta + \rho_0 \mu \frac{\partial \eta}{\partial t} & \text{filtered free surface} \end{cases} \quad (2.20)$$

The equations that describe the vertical velocity and the hydrostatic pressure with this vector invariant form are:

$$\frac{\partial w}{\partial k} = -\chi e_3 \quad (2.21)$$

$$\frac{\partial p_h}{\partial k} = -\rho g e_3 \quad (2.22)$$

Where χ is given by

$$\chi = \frac{1}{e_1 e_2}\left[\frac{\partial(e_2 u)}{\partial i} + \frac{\partial(e_1 v)}{\partial j}\right] \quad (2.23)$$

Finally, the tracer equations become:

$$\frac{\partial T}{\partial t} = -\frac{1}{e_1 e_2}\left[\frac{\partial(e_2 T u)}{\partial i} + \frac{\partial(e_1 T v)}{\partial j}\right] - \frac{1}{e_3}\frac{\partial(Tw)}{\partial k} + D^T + F^T \quad (2.24)$$

$$\frac{\partial S}{\partial t} = -\frac{1}{e_1 e_2}\left[\frac{\partial(e_2 S u)}{\partial i} + \frac{\partial(e_1 S v)}{\partial j}\right] - \frac{1}{e_3}\frac{\partial(S w)}{\partial k} + D^S + F^S \tag{2.25}$$

$$\rho = \rho(T, S, z(k)) \tag{2.26}$$

Where the **D** terms represent the subgrid scale parameterization and the **F** terms the surface forcing terms.

Vertical coordinate systems

If one looks at an ocean column in an arbitrary place, the situation found can be really diverse depending on the place. From shallow waters to deep trenches of several thousands of meters where the water can be stratified in layers or not, with a surface that can move. There are several ways to define a vertical coordinate that can be more useful depending on the specific case and its choice is one of the most important aspects of an ocean model design [23]. In OPA, currently the generally used option is the z*–coordinate System described in [24]. Since the version 3.4 of NEMO the z˜–coordinate is also available in NEMO but it is considered not robust enough to be used in all possible configurations [22].

Domain discretization: Time

The domain has three spatial dimensions plus a time dimension. In order to discretize the time, instead of having it as a continuous variable, it is sliced into a finite number of time steps $t \rightarrow 0, \Delta t, 2\cdot\Delta t, 3\cdot\Delta t, 4\cdot\Delta t, \cdots, n\cdot\Delta t$ .Therefore, the prognostic variables are now defined as follows:

$$x(t) \rightarrow x^n = x(n\cdot\Delta t) \tag{2.27}$$

where x stands for u, v, T or S and Δt is the time step length.

To define the evolution of a given variable x in this discrete time framework, the value of the variable at the next time-step is computed as follows:

$$x^{t+\Delta t} = x^{t-\Delta t} + 2\Delta t \cdot \mathrm{RHS}_x^{t-\Delta t, t, t+\Delta t} \tag{2.28}$$

where x stands for u, v, T or S. RHS refers to Right-Hand-Side of the corresponding time evolution equation, Δt is the time step and the superscripts indicate the time at which the RHS is evaluated, that will depend on the physics with which it is associated.

For the physics related with non-diffusive processes, the time stepping used is the leapfrog scheme which is time centered (RHS evaluated at time step n).

$$\frac{x^{n+1} - x^{n-1}}{2\Delta t} = RHS_x^n x^{n+1} = x^{n-1} + 2\Delta t \cdot RHS_x^n \tag{2.29}$$

However, the applicability of this scheme is limited by several factors, the large phase-speed error, the unsuitability for representation of diffusion and Rayleigh damping processes. To prevent some of the numerical issues appearing, this scheme is often used with a Robert-Asselin time filter, a kind of laplacian diffusion in time that mixes odd and even time steps:

$$x_F^n = x^n + \gamma \cdot \left[x_F^{n-1} - 2x^n + x^{n+1}\right] \tag{2.30}$$

where γ is the Asselin coefficient and the F subscript denotes filtered values. The addition of the filter degrades the accuracy of the calculations from second to first order but the second order truncation error is proportional to the Asselin coefficient which is usually small compared to 1.

For processes that are diffussive or damping, the leapfrog scheme is unsuitable. For those processes the forward time differencing scheme is used:

$$x^{n+1} = x^{n-1} + 2\Delta t \cdot RHS_x^{n-1} \tag{2.31}$$

This scheme is diffussive in time and conditionally stable, with the conditions for stability being:

$$A^h < \begin{cases} \frac{e^2}{8\Delta t} & \text{laplacian diffussion} \\ \\ \frac{e^4}{64\Delta t} & \text{bilaplacian diffussion} \end{cases} \tag{2.32}$$

where e is the smallest grid size in the two horizontal directions and A_h is the mixing coefficient. If the conditions are not satisfied, the model becomes wildly unstable. This can be solved by reducing by either reducing the length of the time steps or reducing the mixing coefficients. The conditions are necessary but not sufficient.

For the vertical diffusion terms, a forward time can be used but usually the numerical stability conditions impose a strong constraints on the time step. In NEMO there are two solutions available to overcome this. The first solution is to use a time splitting technique and the second solution is to use an implicit time differencing scheme. In the first approach the timestep is cut into smaller steps to fulfill the stability criterion. The second solution uses an implicit scheme that requires more time to complete a time-step, but can be worth if the number of necessary steps for the alternative method is bigger than 2.

In order to extend the range of stability of the Leap-Frog scheme, the hydrostatic pressure gradient term is treated separately using a semi-implicit scheme. To do so, the pressure is evaluated using a linear combination of values at $t - \Delta t$, t and $t + \Delta t$, as an alternative of evaluating it only at t. However, the use of this technique is limited to the range of applications where the factor that limits the time-step length are the internal gravity waves. In global low resolution cases the internal oscillations near the North Pole are the limiting factor and in very high resolution configurations usually the limiting factor are the strong currents.

Domain discretization: Space

To discretize the spatial domain, the approach used to compute the variable derivatives is the traditional centered second-order finite difference approximation. For a given dependant variable $f(x)$ the derivative is approximated in the discrete domain as follows:

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} + O(\Delta x^2) = \frac{f_{i+1} - f_{i-1}}{2\Delta x} + O(\Delta x^2) \tag{2.33}$$

where Δx is the grid spacing and i the grid index for the x dimension in the discrete space. The prognostic equations have different dependant variables whose time evolution depend on other dependant variables. Using the scheme described in equation 2.33 we can see that the variable derivatives are computed without using any information from the same point where the derivative is computed. Since the spatial derivatives of the variables are used in equations where the variable itself does not appear, it is possible to define different grid points for the different dependant variables. For example, for the 1d gravity wave equation:

$$\frac{\partial u}{\partial t} = -g \cdot \frac{\partial h}{\partial x} \ , \ \frac{\partial h}{\partial t} = -H \cdot \frac{\partial u}{\partial x} \tag{2.34}$$

where u and h are dependant variables and g and H are constants.

Discretizing the equation 2.34 using the scheme described in equation 2.33 we obtain:

$$\frac{\partial u_i}{\partial t} = -g \cdot \frac{h_{i+1} - h_{i+1}}{2\Delta x} \ , \ \frac{\partial h_i}{\partial t} = -H \cdot \frac{u_{i+1} - u_{i+1}}{2\Delta x} \tag{2.35}$$

Looking at how the discrete equations look using the scheme (scheme ref) we can see that to compute the time derivative of u at a given point i we don't need any h information from this same point. It is possible thus to build a staggered grid where the location of the u points and the h points don't coincide in space. In the case of the Navier-stokes equations described in the first part of this chapter the arrangement of the variables is done following generalization to the three dimensions of the Arakawa C grid [25].

2.2.3 Computational implementation

In this section the computational implementation of the numerical methods (section 2.2.2) used to solve the mathematical equations (2.2.1) is briefly described.

In order to compute everything, the OPA engine follows the next steps:

1. Update forcings and data.

2. Update ocean physics (mostly sub-grid scale physics).

3. Compute active tracer trends.
4. Update active tracers.
5. Compute momentum trends.
6. Update horizontal velocity.
7. Compute the diagnostic variables (rd, N2, div, cur, w)
8. Outputs and diagnostics

The model includes several options that can be activated through the model namelist, and depending on the options used, the computation of several of these steps will be using different parts of the code.

From the steps listed before, the more relevant parts here are from 3 to 6. To update the active tracers, we first need to compute the tracer trends, that will include contributions from different phenomena. Some of these phenomena are always taken into account while others will be used or not depending on the options selected.

Tracers

As was stated in section 2.2.1, the equations that describe the time evolution of the active tracers are equations 2.4 and 2.5. The RHS of these equations has different terms: the advection ($-\nabla \cdot (T \cdot \mathbf{U})$ and $-\nabla \cdot (S \cdot \mathbf{U})$) , the subgrid-scale physics ($\mathbf{D}^T$ and $\mathbf{D}^S$) and the forcings ($\mathbf{F}^T$ and $\mathbf{F}^S$). In the code, the lateral and vertical diffusion of the $\mathbf{D}^S$ term are computed separately. For the $\mathbf{F}^S$ term, the surface boundary conditions are always used but there are several contributions that are only used if explicitly specified in the simulation parameters, these contributions are the penetrative solar radiation, the bottom heat flux, the bottom boundary layer scheme or the use of internal damping terms. For the advection term, there are several schemes implemented that can be selected: a 2nd or 4th order centered scheme, a 2nd or 4th order flux corrected transport scheme, the MUSCL scheme [26], the 3rd order upstream biased scheme and the QUICK scheme [27]. Choosing one or another is a complex matter which depends on the model physics, model resolution, type of tracer, as well as the issue of numerical cost [22]. For ORCA configurations, the recommended scheme is the 2nd or 4th order flux corrected transport scheme.

For the lateral diffusion term, it is also possible to choose between several schemes for the spatial derivatives and for time derivatives the forward scheme is used.

For the vertical diffusion term, the large eddy coefficient found in the mixed layer plus the high vertical resolution may imply that the use of an explicit time stepping scheme would require too short time-steps. To overcome this issue the use of the implicit time stepping is preferred although an alternative explicit forward time differencing scheme is available using a time splitting technique.

For the tracers, the boundary condition terms are implemented in a separate module instead of entering as a boundary condition on the vertical diffusion operator [22]. The tracer surface boundary interchanges depend on the tracer content not linked to mass interchanges and tracer content of the mass exchange. For temperature and salinity, both terms are included in the surface heat flux (F^{heat}) and surface salt flux (F^{salt}).

Dynamics

The dynamics modules solve the momentum, the hydrostatic pressure and the continuity equations. The components of these equations are solved in several modules which are the hydrostatic pressure gradient, the surface pressure gradient and the lateral and vertical diffusion terms, that are computed always regardless of the formulation used, and the advection and coriolis terms that depending on the formulation used are computed on one way or another. These advection and coriolis terms in the vector invariant formulation are decomposed in the vorticity part, the kinetic energy part and the vertical advection part while in the flux form the terms are decomposed in the coriolis part and advection part. The recommended formulation for ORCA configurations is the vector formulation.

The hydrostatic pressure gradient term is computed differently depending on the type of vertical coordinate used. There are several implementations that are either using a leapfrog scheme or a semi-implicit scheme and there are also several choices to describe the lateral boundary conditions. The default time scheme used is the leapfrog 2.36, but in some specific cases the physical phenomena that can be more restrictive with the time step length are the internal gravity waves, in which case a semi-implicit scheme 2.37 can be used to double the stability limit. While the leapfrog scheme evaluates the density at the t moment, the semi-implicit approach computes the hydrostatic pressure gradient as an average over the three time levels $t - \Delta t, t, t + \Delta t$.

$$\frac{u^{t+\Delta t} - u^{t-\Delta t}}{2\Delta t} = ... - \frac{1}{\rho_0 e_{1u}} \delta_{i+1/2}[p_h^t] \tag{2.36}$$

$$\frac{u^{t+\Delta t} - u^{t-\Delta t}}{2\Delta t} = ... - \frac{1}{\rho_0 e_{1u}} \delta_{i+1/2}[p_h^{t+\Delta t} + 2p_h^t + p_h^{t-\Delta t}] \tag{2.37}$$

There is a special situation that requires a special treatment, the presence of ice shelves. In presence of ice shelves, the HPG term is computed by adding the pressure gradient due to the shelves and the pressure gradient due to the ocean.

The surface pressure gradient

How the surface pressure gradient term is computed depends on the representation of the free surface. External gravity waves are allowed in the equations, which impose a very small time step when using an explicit time step. To avoid that, there are two methods available: the explicit and the split-explicit methods. Due to the large speed of the barotropic waves, the CFL condition for the explicit method forces a very short time-step. Its implementation is very simple updating the horizontal velocity fields with the pressure gradient term.

The split-explicit method implemented in OPA was inspired by Shchepetkin and McWilliams 2005. The idea behind the method is that the barotropic and baroclinic modes of the equations can be treated separately, resolving the faster barotropic mode with a smaller time-step, being possible to keep the rest of the model using a larger time-step. The routine in charge of applying this method is dynspg_ts and its purpose is to compute the trend in time t due to the explicit time stepping of the quasi-linear barotropic system. To do it, the barotropic variables are advanced from internal time steps n to $n+1$ or from $n-1$ to $n+1$ depending on the choice between forward or backward time stepping. The specific actions performed are update the filtered free surface at step $n+1$, update the filtered barotropic velocities at step n+1, compute the barotropic advective velocities at step n and update the 3d trend with the barotropic component. The routine exceeds the 1000 lines and involves many 2D and 3D variables. Also, it requires to update the lateral boundary conditions several times, also inside the inner loops.

Lateral diffusion

The code has two different implementations, the laplacian or the biharmonic operators that can be selected through the list of parameters. The biharmonic operator consists in applying two times the laplacian operator and require an update of the lateral boundary conditions.

Vertical diffusion

To compute the vertical diffusion there are two implementations that can be selected, an explicit scheme or an implicit scheme. The use of an explicit scheme would constrain the length of the time-step, however a splitting method to allow a bigger time-step is also coded. The implicit scheme does not constrain the time-step length and computes the vertical diffusion term solving systems of matrices. Since the systems of equations only bound grid point elements of the same column, the process of solving these matrices do not require inter-process communication.

2.3 LIM

The LIM model (from The Louvain-la-Neuve Sea Ice Model) is a sea ice computational model that is part of NEMO in which it is coupled to OPA. It is designed for climate studies and operational oceanography, has been used in many global climate models contributing to the Intergovernmental Panel on Climate Change reports and also its used in the operational oceanography system MERCATOR. At the time of redaction of this manuscript the stable version of NEMO was 3.6 and included two versions of the model, LIM2 and LIM3. [28]

For the experiments done through the different chapters of this document the most recent version LIM3 was used. LIM3 is a model that solves the dynamics and thermodynamics of the sea-ice, including the representation of the subgrid-scale distributions of ice thickness, enthalpy, salinity and age [29].

2.3.1 Mathematical model

LIM3 is a C-grid dynamic-thermodynamic sea ice model with thickness, enthalpy, salinity and age distributions [30]. The sea ice is described with the velocity vector and several sea-ice state variables.

Ice dynamics

Wind and ocean currents makes ice to move and deform. Ice speed is determined by the conservation of linear momentum involving these concepts. To describe it, LIM assumes that the sea-ice behaves like an Viscous-Plastic material (VP) with the following momentum equation (per unit area) :

$$m \cdot \frac{\partial \mathbf{u}}{\partial t} = \nabla \cdot \boldsymbol{\sigma} + A(\boldsymbol{\tau}_a + \boldsymbol{\tau}_w) - mf\mathbf{k} \times \mathbf{u} - mg\nabla\eta \tag{2.38}$$

Where m is the mass, u is the horizontal ice velocity, σ is the internal ice stress tensor, A is the ice concentration, τ_w is the ocean stress, $\tau_a$is the atmosphere stress, f is the Coriolis parameter, k is an unit vector normal to the surface of the Earth, g is the gravity and η is the sea surface elevation. The advection of momentum is considered negligible.

Ice state variables

Instead of considering a single set of ice variables, LIM3 splits the sea-ice in thickness categories, having each category a mean thickness that is constrained to be between the category limits. Each category has its own set of state variables, that include ice concentration,ice volume, ice enthalpy, ice salt content, ice age, snow volume and snow enthalpy. Each category is splitted in vertical layers. The state variables are modified by transport, thermodynamic and mechanical redistribution processes leading to the conservation equation [30]:

Where X refers to any sea-ice state variable, the term $-\nabla \cdot (X\mathbf{u})$ corresponds to the transport (limited to the advection of the variables), $\mathbf{\Psi}^X$ represents the effect of mechanical redistribution processes on X and $\mathbf{\Theta}^X$ represents the thermodynamic effects on X.

Transport

The transport of the sea-ice state variables is limited to the advection.

Thermodynamics

The sea-ice thermodynamic processes include vertical diffusion of heat in the snow/ice system, snow/ice growth and decay, creation of new ice in open water, brine drainage and ice natural ageing. To compute these terms, the interfacial energy budgets, the diffusion of heat, the radiative transfer and the desalination of the ice must be taken into account. All these processes are assumed purely vertical in LIM (no lateral dependence in the computational algorithms). An extensive description of the processes involved can be found in the reference publication of LIM [30]. Rheology

“Sea ice resists deformation with a strength which increases monotonically with ice thickness and concentration.” “Calculation of sea ice internal forces in LIM uses the elastic-viscous (EVP) [31], which is a regularization of the VP model.”

2.3.2 Numerical description

To compute the sea ice rheology, LIM3 uses a method described in Bouillon et al. 2009. To solve the momentum equation it employs a sub-time stepping to achieve convergence of the relaxation algorithm used, including lateral boundary condition updates within. The number of sub-cycles used to solve the elastic-viscous-plastic momentum equation can be changed through the LIM namelist and by default it is set to 120.

The sea ice transport is computed in a separated routine that includes the computation of the advection and the diffusion of the sea ice. The code checks that the CFL condition is fulfilled, and if it is not then the time-step is reduced by half to ensure stability. The advection of the variables uses a Prather scheme [32], and alternates the order of the axes in which it performs the computation. For each sea-ice category it computes the advection for the ice volume, snow volume, ice salinity, ice age, ice concentrations, snow heat contents and ice heat contents, twice for each of these variables (one for each axis). The advection routine do require an update of the lateral boundary conditions for several variables. After the advection, a diffusion operator is applied to these variables. It is a second order diffusive operator evaluated using a Crank-Nicolson time scheme. Although it is unconditionally stable, it uses an iterative process with a convergence exit condition. Within the inner loop there are lateral boundary condition updates and also a global operation to check the convergence in the full domain.

2.3.3 Computational implementation

The code of LIM3 completes the necessary steps in the following order:

1. Compute dynamics
2. Compute advection/diffusion
3. Compute ice thermodynamics
4. Compute ice/ocean fluxes
5. Save outputs

Many of these steps require communication and since many of the numerical methods used are iterative, this implies that during this routine many inter-process messages are needed.

Chapter 3

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

This thesis chapter originally appeared in the literature as Oriol Tintó Prims, Miguel Castrillo, Mario C. Acosta, Oriol Mula-Valls, Alicia Sanchez Lorente, Kim Serradell, Ana Cortés, Francisco J. Doblas-Reyes, ***Finding, analysing and solving MPI communication bottlenecks in Earth System models***, *Journal of Computational Science, 2018, ISSN 1877-7503, DOI:10.1016/j.jocs.2018.04.015.*

Extended Abstract: Understanding the computational behaviour of a computational model can help its users and developers to better exploit computational resources. The work presented in this chapter, that has been already published in a peer-review journal [33] 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, we can learn how to identify performance issues by characterizing the scalability of an application, 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, has been used as an example of success. 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 optimization that will impact bigger experiments running on thousands of cores, making it easier to deal with applications that aspire to run in massive-parallel

systems. The analysis of NEMO 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.

3.1 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 [34] and with the adoption of graphics processing units (GPUs) for general computation, CUDA [1] and OpenCL[2] were also created[35]. 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[36][37]. 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 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

[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.

complemented with other submodels to better describe other important processes as sea-ice, land, biosphere...

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 [38], 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 [39] [40] [41] [39][42].

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 [43] [44][45]. 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 [46] and the number of iterations required to converge to an acceptable solution [47] [48] [49] [50]. For example, some works [47] [51] 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 [52] 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 [13], 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 [53], 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 [13], 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 to 1000 person years [54]. 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 [55]. Although there have been some previous performance analyses on NEMO [56] [57], 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.

The second section 3.2 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 3.3 contains a description of the model used to do a demonstration of the method. In section 3.4 there is an analysis carried out using the proposed methodology. Finally, in section 5 the conclusions of the work are exposed.

3.2 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.

3.2.1 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.

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.

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 [58]. However, in this work the tools used were Extrae, to collect the information, and Paraver, to visualize it, both from the BSC tools suite.

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.

3.2.2 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)[17], 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)[18].

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... 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 .

3.3 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.

3.3.1 About NEMO

The Nucleus for European Modelling of the Ocean (NEMO) [22] 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. [55].

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 (Figure 3.1) 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 figure 3.1) between neighbour subdomains. In NEMO this operation is performed by means of a module using MPI.

The LIM model [30][29] 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

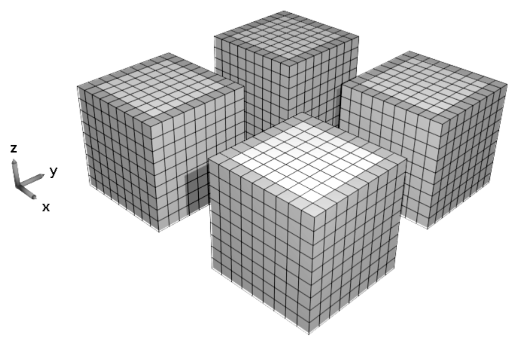

Figure 3.1: Vertical 2D-Pencil domain decomposition for three-dimensional grids. In dark gray, the boundary elements that have to be interchanged between the subdomains.

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 [59]. 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 [60]. For this reason, the number of iterations that the model has to perform to simulate the same time lapse increases with the spatial resolution.

Table 3.1 shows that while ORCA2 has a default time step length of 5760 seconds, the usual time step lengths for higher resolutions are 3600, 900 and 300 seconds 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.

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	300 6	991.6M

Table 3.1:: 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.

3.3.2 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...) 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 [29]. Further studies considering the other modules are foreseen in the future.

3.3.3 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 3,056 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.

3.4 Analysis and results

This section sets out in 3.4.1 the analysis of NEMO, the case study, following the steps presented in 3.2. Additionally, in 3.4.2 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.

3.4.1 Analysis

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 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 Figure 3.2 and table 3.2 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.

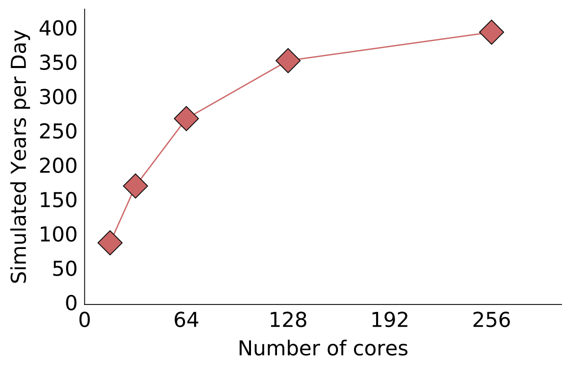

Figure 3.2: Throughput in Simulated Years per Day achieved by the NEMO model using an ORCA2-LIM3 configuration with different number of resources.

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%

Table 3.2:: 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.

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 figure 3.3 to have a clearer picture of how the time consumed by each function evolves.

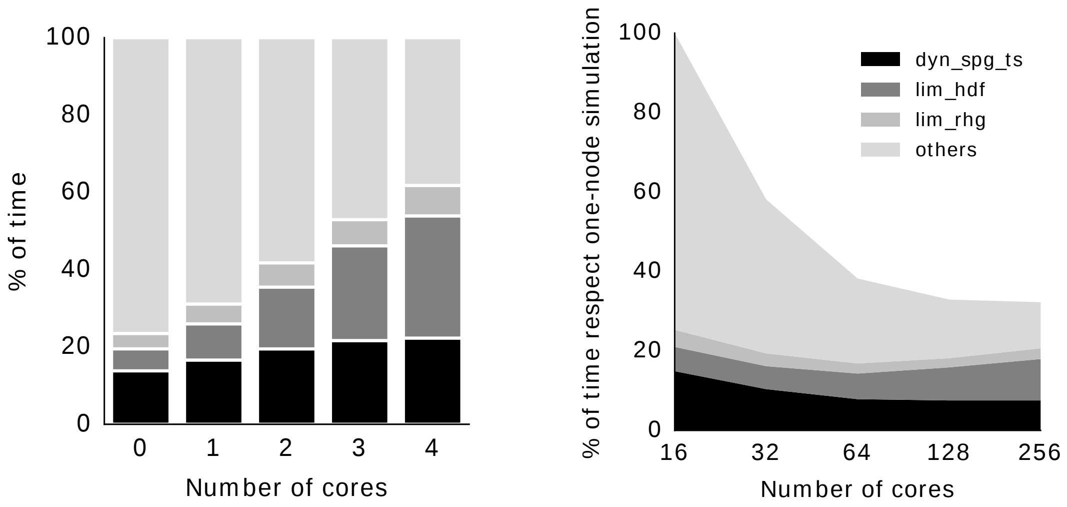

Figure 3.3: A) Percentage of time spent in sea surface pressure gradient (dyn_spg_ts), sea-ice horizontal diffusion (lim_hdf), sea-ice rheology (lim_rhg) and all the other routines (others) when using different number of cores.B)Time spent in sea surface pressure gradient (dyn_spg_ts), sea-ice horizontal 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.

Looking at the distribution of the time spent in each function (See Figure 3.4), 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 figure 3.3, the functions that are relevant because of their bad 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.

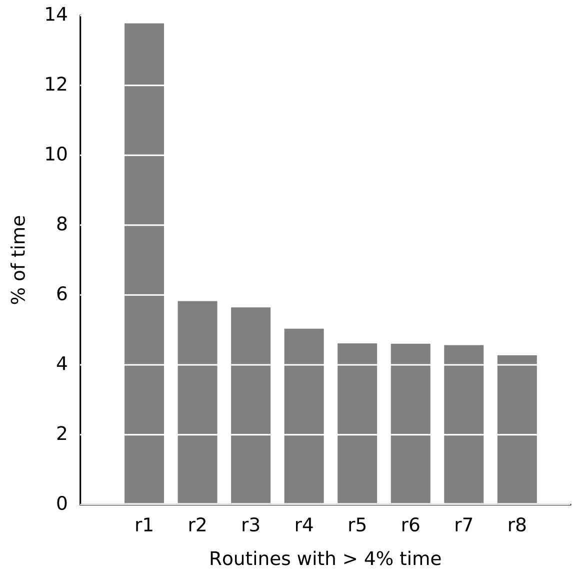

Figure 3.4: Percentage of time spent in the eight most costly routines for an ORCA2-LIM3 simulation using a single-node (16-cores) of Marenostrum III. The profile is very flat and the most costly routine (the surface pressure gradient) does not even reach a 14% of the total execution time.

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.

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 previously 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 3.2.1.

In the case of the **surface pressure gradient** routine, using the Paraver tool to visualize the MPI information collected through different experiments (Figure 3.5), 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.

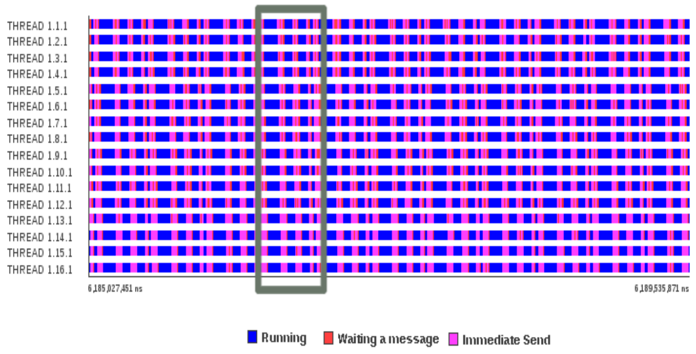

Figure 3.5: Paraver view of the sea surface pressure gradient routine. Inside the gray rectangle, it can be seen the pattern that it is repeated consisting on four computation phases (in blue) separated by communication phases (red and pink). The different lines in the y axis represent the different 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 milliseconds of simulation.

For the second routine identified, **sea-ice horizontal diffusion**, the previous step of the analysis showed a very bad scalability. Using Paraver (see Figure 3.6), 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.

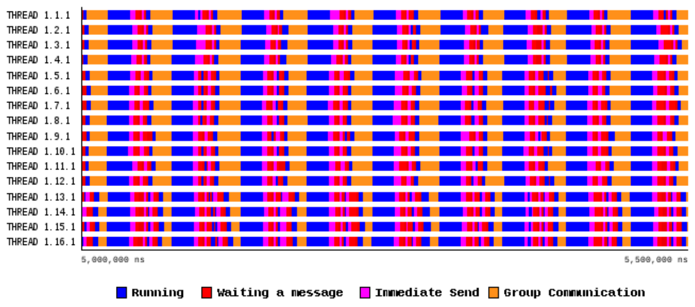

Figure 3.6: Paraver view of the sea-ice horizontal diffusion routine. It can be seen a iterative structure consisting in a computational phase (in blue), a border interchange (red and pink) a very short computational phase and a group communication (in orange).The different lines in the y axis represent 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 milliseconds of simulation.

The third one of the routines, **sea-ice rheology** (see Figure 3.7), 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.

3.4.2 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.

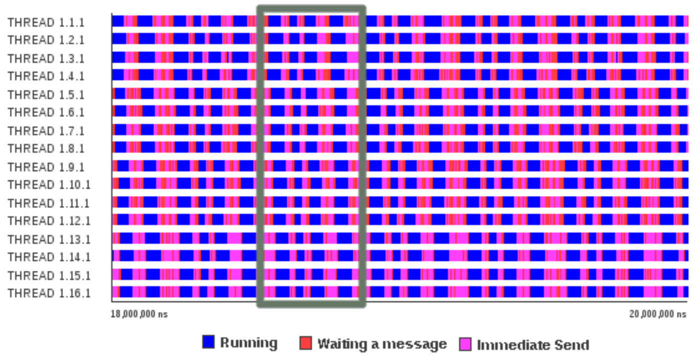

Figure 3.7: Paraver view of the sea-ice rheology routine. Inside the gray rectangle It can be seen the iterative structure consisting in four computational phases (in blue) separated by border interchanges (red and pink). The different lines in the y axis represent the different threads (16 in the single 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 milliseconds of simulation.

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 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. The Figure 7 shows that a 15% throughput improvement is achieved using 256 cores. In the same sense, the Figure 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 [61], 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 parallelisation 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 Figure 3.8. 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, figure 3.9 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.

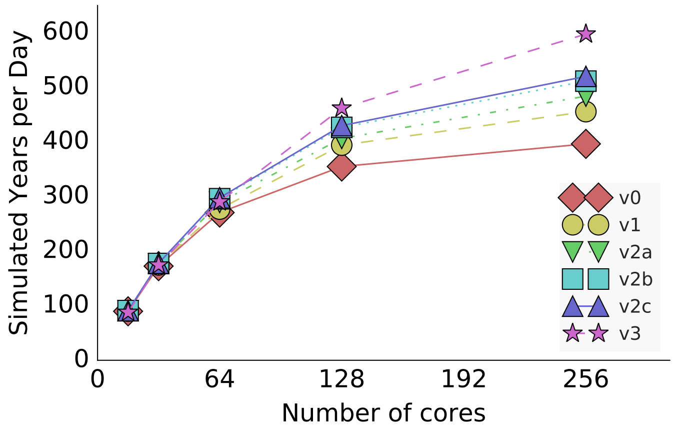

Figure 3.8: Model throughput in Simulated Years per Day using an ORCA2-LIM3 configuration for the different versions of the code with the different optimizations and for different number of cores.

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 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

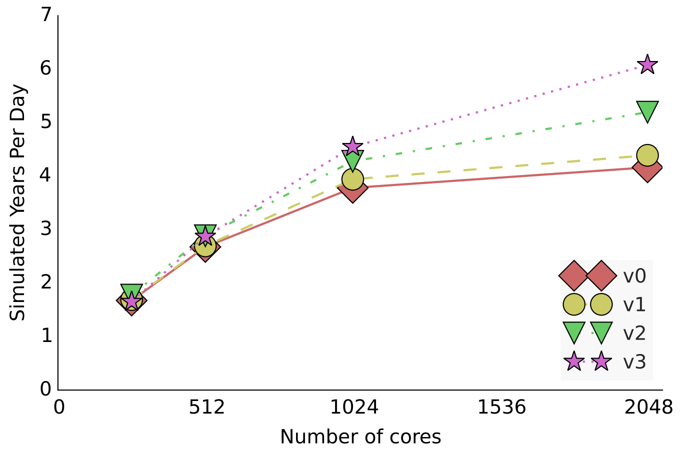

Figure 3.9: Model throughput in Simulated Years per Day using an high resolution ORCA025-LIM3 configuration for the different versions of the code with the different optimizations and for different number of cores.

seen in figures 3.8 and 3.9. 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.

3.5 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.

Chapter 4

Optimizing domain decomposition in an ocean model: the case of NEMO

This thesis chapter originally appeared in the literature as Oriol Tintó, Mario Acosta, Miguel Castrillo, Ana Cortés, Alícia Sanchez, Kim Serradell, Francisco J. Doblas-Reyes ***Optimizing domain decomposition in an ocean model: the case of NEMO****, Procedia Computer Science, 2017, Volume 108, Pages 776-785, ISSN 1877-0509,DOI:10.1016/j.procs.2017.05.257.*

Extended Abstract: Climate science, like other Earth Sciences, heavily rely on computer simulations. The computational models used to perform these simulations are in constant evolution and their growing complexity will imply an increased computational cost. Since the cost of using these state-of-the-art models is already huge, looking closely at the factors that are able to impact their computational performance is mandatory. In the case of the state-of-the-art ocean model NEMO (Nucleus for European Modelling of the Ocean), used in many projects around the world, not enough attention has been given to the domain decomposition. NEMO uses a domain decomposition method to exploit parallel computer systems, by splitting the ocean domain in parts that are computed separately, and to do so there are few factors that are relevant in order to chose the specific domain decomposition. Even that the ocean is represented in a three-dimensional grid, along the code there is a different treatment of the vertical and horizontal dimensions (parallel and perpendicular to gravity), for convenience, the approach used to partition the grid is to keep all the columns in the same domain, using a 2D pencil domain decomposition. The ocean domain is represented as a three-dimensional grid, but for its that in global domains covers the full surface of the Earth. However, the oceans only cover the 71% of the

surface, meaning that there will be points of this grid that actually correspond to a point placed over land.

In this work we show the impact that the selection of a particular domain decomposition can have on computational performance and how the proposed methodology substantially improves it.

4.1 Introduction

Climate change and the threat that it poses to the humankind and to the entire planet has become one of the main challenges for the society. For that reason, it is not coincidence that the understanding of the climate system is a subject of growing interest and a hot topic that has even moved to the center of political actuality. There are different tools available to scientists to study climate change, from which Earth System Models (ESMs) are the most powerful ones. ESMs are complex systems made up by one or more mathematical models describing different parts of the Earth System. These models simulate the atmosphere, ocean, sea-ice and the other components, usually involving the resolution of differential equations that describe the behaviour of these systems.

These models have been in constant evolution over the last decades and have considerably grown in complexity, including new phenomena or solving better old features. One of the factors that helped to represent better simulated processes is the increment of grid resolution, that is, the vertical and horizontal space where the equations are solved. For example, several works have proved that enhanced horizontal resolution can lead to significant changes in large-scale aspects of circulation as well as improvements in small-scale processes and extremes [62] [63]. This fact is consistent with other studies showing that an increment of resolution is useful to reduce biases [38]. In the ocean, some of the processes that are better represented at higher resolution are boundary currents, water exchange through narrow straits, coastal currents, upwelling, oceanic eddies [64] [65] [66], ENSO [67] [68] [69], and sea-ice drift and deformation [70] [71]. Nevertheless, the additional complexity has demanded an increase of the computational resources required to run these models, reducing the range of machines in which they can work to the domain of supercomputers. The consequence is that, although in theory the usage of the highest possible resolution for ESMs grids would benefit scientific performance of simulations, at the time of defining an experiment there is usually a tradeoff between scientific interest and computational cost. In other words, the computational requirements of an ESM determine both the amount of resources that are needed to perform an experiment and also the kind of experiments that can realistically be done. Having a computationally efficient model is not just a way to use the resources wisely, but a manner to allow better experiments. And even more, the possibility to keep increasing the complexity of these models will exist only if the way they can use the new architectures and technologies is improved.

Like any other software, in order to execute ESMs in supercomputers, they must first be parallelized. ESMs that have been for more than two decades in development had already completed this process in late eighties [72] [21], to start taking profit of multi-core systems when they were first appearing. They usually exploited domain decomposition strategies, in which the different processes execute the same code, performing the same work on different sub-domains. Which is the domain to divide in the case of ocean models? Since the Earth is not a regular sphere made of water, when discretizing the ocean, models have to deal with the fact that the planet Earth has emerged land areas and the ocean has different depths. If we try to describe the domain as a regular grid, we will have grid-points where there is no water, because they are located over the land or because they are located under the ocean bottom. A way to deal with this, consists in using a mask to determine which points of a regular grid belong to ocean or land.

Using this approach, when it is time to compute over the grid there are two main possibilities: the first one is to perform calculations over the whole domain and, afterwards, use the mask to omit the results over land-points and, the second one, consist in, before doing any computation, check if the point about to be computed corresponds to a sea-point or not. Between these two options, the former presents one main disadvantage and many advantages with respect to the latter. The disadvantage is that the model does some useless computation, and the advantages are that the regular pattern of the memory accesses and the lack of conditionals benefits the computational performance by allowing the exploitation of many modern processor features like pipelining and vectorization. Contrarily, the use of conditionals inside computation loops prevents vectorization and can cause pipeline bubbles that can seriously hurt performance. On the other hand, a complete different approach consists in using a non-regular grid in which only the ocean-points are defined. This approach could be a better option if taking into account aspects like memory usage (useless land points are not allocated in memory) and better load balance among subdomains. However, these advantages have a direct impact in the model discretization, adding complexity in the implementation of the equations solving. For this reason, this approach has not been widely explored.

Even though the "compute everything" choice has many advantages, it can lead to a ridiculous situation. Applying a regular domain decomposition can produce some subdomains where all the points correspond to land, and consequently, all the computation done in that specific subdomain is useless. Figure 4.1 shows this situation, with a particular case in which 12 % of the depicted subdomains do not contain any sea-point and, consequently, the work done at these subdomains is completely useless. Anyhow, a way to exploit the advantage of having a regular domain decomposition without suffering the penalty of considering only-land subdomains as computational elements, consists in removing these subdomains from the set of subdomains to compute. This simple strategy can save a significant part of the computational resources but it is not the only thing to take into consideration.

In this work, we propose a methodology to determine the best domain decomposition taking into account the portion of the domain that can be omitted and other aspects as the grid overlapping required for satisfying boundary conditions and the resources available.As a case study we have used the state-of-the-art ocean model NEMO[22] (Nucleus for European Modelling of the Ocean), that was the ocean component of many of the ESMs taking part in the last Coupled Model Intercomparison Project (CMIP) experiment for the Intergovernmental Panel on Climate Change and will be in 18 of the 37 ESMs participating in the next CMIP6. It has thousands of users and it is being developed by a broad number of scientists from many institutions, mainly in Europe. Since so many resources are invested in simulations involving the NEMO model, to squeeze at maximum its domain decomposition is a relevant point. In this state-of-the-art ocean model, it is possible to switch off the land-only processes with the aim of not wasting resources on it. However, the model itself does not take advantage of this capability and its activation requires a fine-tuning from the user.

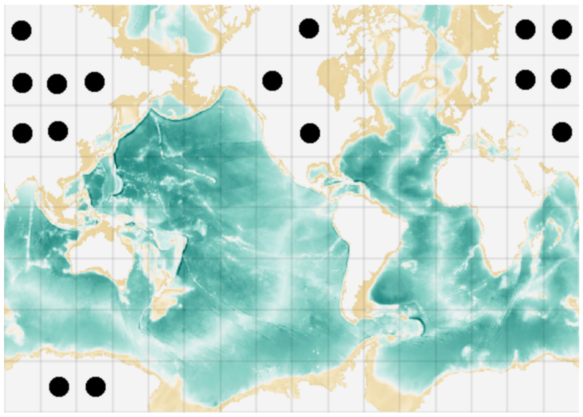

Figure 4.1: Domain decomposition of a tripolar grid of the ORCA family with a resolution of a quarter of degree into 128 subdomains (16 x 8). Subdomains marked with a black dot do not contain any ocean grid point.

The next section describes how the domain decomposition is currently done in NEMO and the proposed methodology to determine the best domain decomposition for the ocean model. The experimental results of the improvements in the NEMO performance and in ESMs in general are shown in section 3. Finally, section 4 states the main conclusions of this work

4.2 Methodology

The problem of finding an optimal domain decomposition can be defined as a multi-objective optimization problem since there are multiple factors to optimize that are conflicting and, for that reason, there is no unequivocal optimal solution. Even more, the resulting layout may have to fit into limited resources. To understand the problem that we want to solve, the section 4.2.1 explains how the domain decomposition is done in the NEMO model, in section 4.2.2 a method to find decompositions that adjust the user interests is presented.

4.2.1 Domain decomposition in NEMO

For global simulations, NEMO uses ORCA grids. They are a grid family for global models whose common characteristic is that they are tripolar, with the aim of avoiding numerical singularities at the north pole. The geographic south pole is conserved and from 80°S to 20°N the grids are like a regular Mercator-grid. By contrast, in the north hemisphere two poles are used, placing them over land, one in Canada and the other in Asia. From 20°N to the north-poles, the circles of the Mercator-grid are progressively distorted into ellipses. For the third-dimension of the grid, the earth is assumed to be spherical and the vertical dimension is considered to be perpendicular to the surface.

The resolution of these grids is not homogeneous, and the denomination of the different ORCA grids refer to their resolution at the latitude at its coarsest part, that is the equator i.e. ORCA2, ORCA1, ORCA025, ORCA12, etc. The number of points of the grid is $jpiglo \times jpjglo \times jpk$ where $jpiglo$ is the number of grid points in the longitudes direction, $jpjglo$ is the number of grid points in the latitudes direction and jpk is the number of vertical levels. The specific values of these variables depend on the resolution of the grid (See Table 4.1).

Name	jpiglo	jpjglo	jpk	Horizontal resolution (km)	TotalSize (Million points)
ORCA2	182	149	32	219.8	0.86
ORCA1	362	292	75	110.5	7.92
ORCA025	1442	1021	75	39.1	110.42
ORCA12	4322	3059	75	9.3	991.57

Table 4.1:: Parameters of some ORCA grids with different resolutions.

The model was parallelized and is able to run on both shared and distributed memory environments, using a 2D-Pencil domain decomposition method (Figure 3.1) that splits the global three-dimensional domain in two dimensions, keeping all the vertical levels of the global domain in the same subdomain. Each one of the subdomains is executed in a different processor core and it uses the Message Passing Interface message passing system to manage the necessary communications between subdomains. These include not only the corresponding part of the global domain but also the halos, that are used to satisfy the boundary conditions, containing information from the border points of the neighbour subdomains. We refer to these points that are represented twice as overlapping. Due to the overlapping, the sum of the size of the subdomains is bigger than the original domain.

The global domain of size $jpiglo \times jpjglo \times jpk$ is decomposed into $jpni \times jpnj$ subdomains of size $jpi \times jpj \times jpk$ where $jpi = \frac{jpiglo}{jpni} + 2 \cdot haloSize$ and $jpj = \frac{jpjglo}{jpnj} + 2 \cdot haloSize$. The user can specify the jpi and jpj himself or can leave these values empty and let the model find a particular decomposition.

As it was explained before, some of the subdomains can end-up containing only land points (See figure 4.1), so it is possible to eventually exclude these subdomains from the calculus. However, despite the model being able to exclude land-only subdomains, the way this option can be activated is not straightforward. To enable it, the user has to specify the values for $jpni$ and $jpnj$ and also the number of subdomains that contain at least one sea-point, such that $seaSubdomains < jpni \times jpnj$. Finally, at the time of execution $seaSubdomains$ must coincide with the number of launched processes.

4.2.2 About the proposed method

Since the domain decomposition is done to distribute the work among different processes, one of the main targets is to distribute the work as much as possible with the objective of reducing the computing resources required by each subdomain. Further dividing the domain will have two additional consequences, on one hand the proportion of land-only subdomains that can be omitted in computation will increase, but on the other hand the part of the domain that corresponds to overlapped areas will have more significance. These two factors determine the size of the computed domain, which is the sum of the sizes of all the subdomains.The objective is then to minimize at the same time the **subdomain size** and the **computed domain**, which depends on both the proportion of land-only subdomains and the overlapping.

To assess how good a specific decomposition is, regarding the resulting computed domain, we can use the ratio between the computed domain and the original domain size . This ratio, that we will call **gridFactor**, can be expressed as the product of the overlapping factor by the proportion of sea-subdomains. The **overlapping** factor is the proportion of the sum of all the subdomains to the size of the global domain, and the **proportion of sea-subdomains** is the ratio of the subdomains that contain at least one sea-point to the total number of subdomains.

$$overlappingFactor = \frac{subdomainSize \cdot totalSubdomains}{globalDomainSize} \tag{4.1}$$

$$seaSubdomainProportion = \frac{seaSubdomains}{totalSubdomains} = 1 - \frac{landSubdomains}{totalSubdomains} \tag{4.2}$$

The subdomain size depends not only on the total number of subdomains but on the specific decomposition, i.e. the number of divisions in each dimension (latitudes and longitudes). It will be then the number of points in the longitudes direction (*jpiglo*) divided by the number of divisions in that direction (*jpni*) plus the halo, multiplied by the number of points in the latitude direction (*jpjglo*) divided by the number of divisions in that direction (*jpnj*) plus the halo. For implementation specifics, the formula used in the model is the following.

$$\begin{aligned} seaSubdomainSize = &\left(\frac{jpiglo - 2 \cdot halo + (jpni - 1)}{jpni} + 2 \cdot halo\right) \cdot \\ &\left(\frac{jpjglo - 2 \cdot halo + (jpnj - 1)}{jpnj} + 2 \cdot halo\right) \end{aligned} \tag{4.3}$$

Then the **gridFactor** is:

$$gridFactor = seaSubdomainProportion \cdot overlappingFactor$$

If our intention was only to minimize the overall work, the **gridFactor** index would be enough but since we are also interested in minimizing the **subdomainSize** we can define a fitness function that takes into account both elements:

$$fitness = gridFactor^{\alpha} \cdot subdomainSize^{\beta}$$

where α and β are the exponents that we give to each factor. With $\alpha = 0$ and $\beta > 0$ we are considering better the decompositions that minimize the subdomain-size, with $\alpha > 0$ and $\beta = 0$ we are considering better the decompositions that minimize the computed domain.

To compute the fitness function of a specific decomposition, the only parameter that we can not know a priori is the number of land subdomains, that must be computed using the bathymetry information. To find how many of the subdomains are land-only, the process consists in building the subdomains and looking if there are any sea-points inside.

With this information we can compute the fitness of any decomposition and estimate which one is better for our interests. There is yet another problem that needs to be solved; while it is straightforward to find the fitness of a specific domain decomposition, answering the question of which decomposition has the best fitness with a limited number of resources is not trivial. Since we can omit the land subdomains, the number of subdomains will be reduced, and therefore decompositions that would not fit into a specific number of resources without removing these processes may fit if these are discarded. To find the decomposition that fits in the resources and has the best fitness value we can perform a search of all the possible decompositions, sort the resulting list by fitness and filter the ones that have more subdomains than available processes.

The process of finding the best decomposition will be:

- For each possible number of subdomains between 1 and the maximum number of available cores plus 45%, find all the possible factorizations of two elements using integers (for instance, 100 = 10 x 10 = 20 x 5 = 25 x 4 = 50 x 2 = 100 x 1). The value of 45% comes from the fact that approximately 30% of the surface points correspond to land, so combinations that have a number of subdomains bigger than 145% of the available resources will not fit even if it is possible to remove all the land points.
- For each case compute its fitness and the number of sea-subdomains.
- Sort the list by fitness.
- When it is time to run the model, remove from the list the combinations where the number of sea-subdomains is bigger than the number of available processes.
- The first combination of the list is the best option.

The whole process can be done once for each specific bathymetry and a ranking with the fitness of the different decompositions can be saved. In that manner, if the number of available resources changes, it is not necessary to repeat all the analysis.

4.3 Results

Results commented in this section come from simulations performed on the Marenostrum 3 supercomputer, hosted by the Barcelona Supercomputing Center. It has a peak performance of 1,1 Petaflops, with 48,896 Intel Sandy Bridge processors in 3,056 nodes. A The metric used to quantify the model throughput is Simulated Years per Day (*SYpD*) that is the number of model years that can be simulated in a period of one day of wall-clock time [73]. This metric is widely used in climate science because it is an important measure of the ability to perform research in a timely manner. A general consensus is that 5 years per day is the minimum acceptable useful simulation rate [74].

Case	jpni	jpnj	Computed Domains	Throughput
With land processes	64	32	2048	5.47 SYpD
Without land processes	64	32	1533	7.15 SYpD

Table 4.2:: Impact of land-process removal.

In table 4.2 we compared the throughput of two ORCA025LIM3 simulations using 2048 cores and the same domain decomposition (jpni=64,jpnj=32). In one case the land subdomains are included and the other avoids its computation. In the first case the model achieves a throughput of 5.47 *SYpD* and in the second, where the 515 subdomains that only contained land-points are removed, the achieved throughput is 7.15 *SYpD*. Summarizing, using 25% less resources the model goes 30% faster, resulting in an efficiency improvement of 74%.

The Figure 4.2 shows the ratio between the computed and global domain sizes. This figure shows that the NEMO algorithm to find a domain decomposition in most of the cases ends up using a really bad decomposition as a solution that increments hugely the size of the computed domain, and it is merely random if one specific number of subdomains leads to either a good or a bad decomposition. On the other hand, the method proposed in section 4.2.2 always maximizes the fitness function for any given number of subdomains. Since the method proposed evaluates all the possible decompositions that fit with a specific number of resources, it will always outperform the default NEMO decomposition.

The figure 4.3 shows the model throughput of real simulations using an ORCA025 grid with the NEMO' s default decomposition and with the proposed method, It can be seen how the proposed methodology impacts the performance of real simulations achieving a speed-up thanks to the reinvestment of the resources saved by land process removal. In the case of having up to 2048 processor cores available, the default decomposition chosen by NEMO is 64 x 32 domains, achieving a throughput of 5.47 SYpD, while with the proposed method the decomposition of 64 x 47 achieves 7.73 SYpD, 41% more throughput. With the double of resources (4096 processor cores) and being 64 x 64 the default decomposition, the measured throughput was 5.25 SYpD, which is even slower than the 2048 case. Nevertheless, using the proposed method the throughput was 8.81 SYpD using an 85 x 68 decomposition, which is

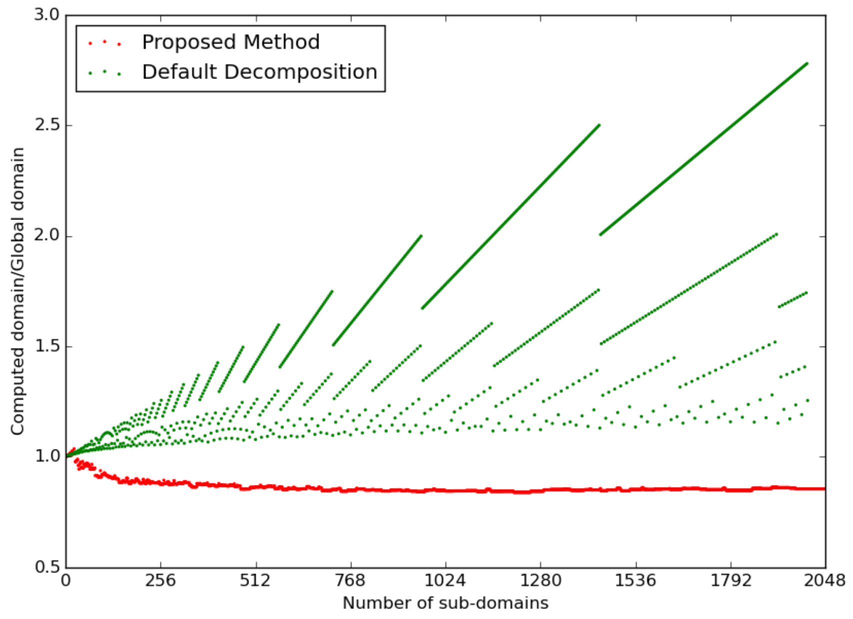

Figure 4.2: Ratio between the computed domain size and the global domain size for the domain decompositions used for each number of subdomains in a ORCA025 grid. In green the cases using the NEMO's default algorithm, that do not remove land-only subdomains, and in red the decompositions found using the method proposed, removing the land-only subdomains. The red data samples corresponds to domain decompositions found using the fitness function with $\alpha = 1$ and $\beta = 1$.

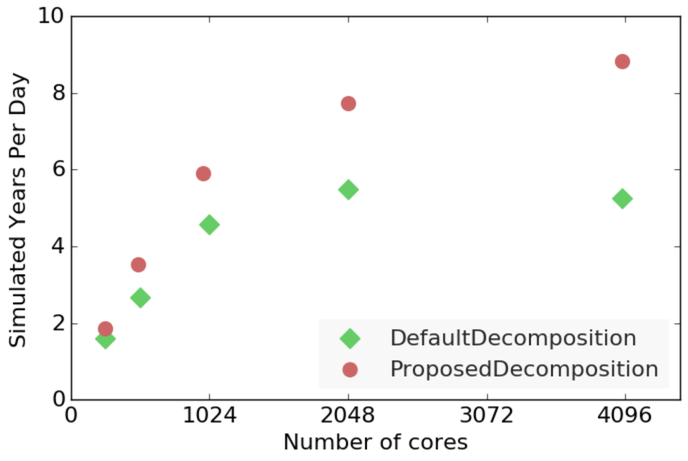

Figure 4.3: Model throughput using the model's default decomposition without land process removal and using the proposed domain decomposition with land process removal.

faster than the 2048 case.

4.4 Conclusions

Regular domain decomposition is sometimes the best option for Earth System computational models. However, results shown in this work prove that the impact of the domain decomposition and the removal of the land-processes have been alarmingly underestimated.

In the case of NEMO, the model evaluated and which is used and supported by a large community around the world, the default domain-decomposition given by the model' s algorithm can lead to decompositions that are quite far from the optimal because they do not minimize the overlap between the subdomains and do not maximize the possibility to remove land-only processes. From the measures performed in real simulations, shown at section 3, we can get two important conclusions. In first place, the fact that removing the land-only subdomains the model has better throughput (30% faster using 25% less resources) indicates that these processes were not only doing useless computation but also harming the overall performance, since they were increasing communication overhead due to their participation in collective communications. In second place, the proposed method outperforms the model default domain decomposition and it is useful to find the best domain decomposition for a specific number of available resources. As an example, we achieve 41% gain in throughput for ORCA025LIM3 simulations with 2048 available processor cores. We have shown that taking into consideration both the overlapping and the possibility to remove land-only sub-domains it is not only possible to reduce the resources used but also to achieve performance gains in throughput. Using this methodology we are able to exploit at the maximum the available resources, by speeding-up the model or by reducing the necessary resources to achieve the target throughput. Over all these facts, the most important conclusion is that the proper selection of a domain decomposition can not be ignored any longer.

Chapter 5

How to use mixed precision in ocean models: exploring a potential reduction of numerical precision in NEMO 4.0 and ROMS 3.6

This thesis chapter originally appeared in the literature as Oriol Tintó, Mario Acosta, Andrew M. Moore, Miguel Castrillo, Kim Serradell, Ana Cortés, Francisco J. Doblas-Reyes ***How to use mixed precision in ocean models: exploring a potential reduction of numerical precision in NEMO 4.0 and ROMS 3.6***, *Geoscientific Model Development, 2019, Volume 7, Pages 3135-3148, DOI:10.5194/gmd-12-3135-2019.*

Extended Abstract: Mixed-precision approaches can provide substantial speed-ups for both computing- and memory-bound codes with little effort. Most scientific codes have overengineered the numerical precision leading to a situation where models are using more resources than required without knowing where they are required and where they are not. Consequently, it is possible to improve computational performance by establishing a more appropriate choice of precision. The only input that is needed is a method to determine which real variables can be represented with fewer bits without affecting the accuracy of the results. This paper presents a novel method that enables modern and legacy codes to benefit from a reduction of precision of certain variables without sacrificing accuracy. It consists of a simple idea: we reduce the precision of a group of variables and measure how it affects the outputs. Then we

can evaluate the level of precision that they truly need. Modifying and recompiling the code for each case that has to be evaluated would require a prohibitive amount of effort. Instead, the method presented in this paper relies on the use of a tool called Reduced Precision Emulator (RPE) that can significantly streamline the process . Using the RPE and a list of parameters containing the precisions that will be used for each real variable in the code, it is possible within a single binary to emulate the effect on the outputs of a specific choice of precision. When we are able to emulate the effects of reduced precision, we can proceed with the design of the tests that will give us knowledge of the sensitivity of the model variables regarding their numerical precision. The number of possible combinations is prohibitively large and therefore impossible to explore. The alternative of performing a screening of the variables individually can give a certain insight about the required precision of variables, but on the other hand other complex interactions that involve several variables may remain hidden. Instead, we use a divide-and-conquer algorithm that identifies the parts that require high precision and establishes a set of variables that can. This method has been tested using two state-of-the-art ocean models, NEMO and ROMS, with very promising results. Obtaining this information is crucial to build an actual mixed precision version of the code in the next phase that will bring the promised performance benefits.

5.1 Introduction

Global warming and climate change are a great challenge for human kind, and given the social [10], economic [11] and environmental threat [6] that it poses, any effort to understand and fight them fall short. A better knowledge and greater capacity to forecast how the climate will evolve can be a game changing achievement, since it could help to justify ambitious policies that adapt for future scenarios [75]. The Earth System can be seen as an amalgamation of many parts: the atmosphere, the hydrosphere, the cryosphere, the land surface and the biosphere. All these elements are extremely rich in phenomena, and are open and inter-related. Fluxes of mass, heat and momentum interchange in ways that are virtually endless, some of which are poorly or not known. The magnitude and complexity of these systems make it difficult for scientists to observe and understand them fully. For this reason, the birth of computational science was a turning point, leading to the development of Earth System Models (ESMs) that allowed the execution of experiments that were impossible until then. ESMs, despite being incomplete, inaccurate and uncertain, have been a framework where possible to build upon knowledge, and have become crucial tools [76]. Since their inception, the capability to mimic the climate system has increased, and with it the capacity to perform useful forecasts [2]. The main developments that have led to this improvement in model skill are the enhancement of the physical parameterizations, the use of ensemble forecasts, the improvement of the model initialization and increases in resolution [2]. Most of these contribute to a higher computational cost [77]. For this reason, these developments are only possible with an increasing availability of computing power, a situation that will continue in the future. The motivation to make models efficient is twofold. Firstly, developments that are considered crucial to increase the skill of the models require more computational power. This is not just a matter of having a larger machine since some of the issues that emerge are not trivial and require additional developments [13]. Secondly, the huge investment in computational resources that is necessary to perform simulations with ESMs implies that investing time in optimizing them will be of value.

One research field that has gained momentum in recent years and that can improve model performance is the use of mixed precision algorithms. Until not so long ago, the speed of most computations was constrained by how fast a CPU could perform operations, with the speed of the memory being fast enough to provide more data than the processor could process. In addition CPUs were designed in a way that they could virtually perform operations at the same speed no matter whether they were operating with 32-bit or 64-bit floating point representations. Therefore, the only benefit of using less precision was to reduce the memory requirements and the computational performance was not so much a motivation. Mainly two factors have changed that scenario. First, the CPU speed increased at a faster rate than memory speed meaning that at some point many codes that were CPU-bound before would become memory-bound, with the memory-bandwidth being insufficient to feed all the data that the processor can process. Second, vector operations doubled the number of floating point operations per cycle that could be performed when the number of bits of the representation is halved. For this reason, the use of smaller representations can now provide performance benefits that justify the effort of optimizing the numerical precision [78], and while this is true with the actual hardware, the expected potential is even bigger in future architectures that will include not only 64-bit and 32-bit but also 16-bit arithmetic.

We are now in a situation where ESMs, as computer codes of other domains, need to use computational resources efficiently and where mixed-precision approaches emerge as a potential solution to help improve efficiency.

The main risk of reducing the precision is falling short, since using less precision than needed can lead to numerical errors that make model results inaccurate or simply wrong. The precision required depends on many factors, so what is needed is a method to identify which variables can effectively use less precision without compromising the quality of the simulations and which ones cannot. If the precision can be reduced, in many situations it is because the precision has been over engineered. One example is the precision used to represent the input data provided to ESMs. The precision of the physical observations is limited by the instruments used to collect them. In the case of sea surface temperature measurements from Earth orbiting satellites, this precision is of a few tenths of a degree. 64-bit representations are often used when 16-bit could potentially be enough.

Recent work has demonstrated the potential benefits that mixed-precision approaches can provide to many different kinds of codes, since it is possible to achieve substantial speed-ups for both computing- and memory-bound codes requiring little effort with respect to the code [78]. The spectrum of studies goes from explicit code manipulation in very specific algorithms to automatic modification of binaries of any kind of code. Some studies have focused on the use and development of mixed-precision algorithms to obtain performance benefits without compromising the accuracy of the results [78][79]. There are also several automatic mixed-precision exploration tools [80] [81] that have been mainly tested on small benchmarks, usually C++ codes. These kinds of studies inspired Earth Science groups working with ESMs that are willing to improve their computational performance to make bigger and more ambitious experiments possible [20][82][83][84][85]. Inspired by previous work, we propose a method that automatically explores the precision required for the real variables used in state-of-the-art ESMs.

The method emerged while trying to explore how to achieve simulations as similar as possible to the standard double precision simulations whilst reducing the precision of some of the variables used. Our work extends the research aforementioned to achieve mixed precision implementations for full-scale models. To do so we rely on the Reduced Precision Emulator (RPE) [86], that mimics the effects of using an arbitrary number of significand bits to represent the real variables in a code and measure the impact of a specific reduced precision configuration in the output produced by the model. Minimizing the user intervention by automating all the tedious intermediate processes allows an analysis of models that would have otherwise required too much manpower. Although the tool is very convenient for exploring the impact of the bits used to represent the mantissa of the floating point numbers, the effect of changing the number of bits devoted to represent the exponent is not explored.

To test the methodology, this work includes two case studies. These cases correspond to two different ocean models that are widely used worldwide: NEMO and ROMS. With these models we demonstrate how the methodology can be used with different applications, thus demonstrating its potential.

5.2 Method

In this section we will demonstrate how we can establish which real variables written in a Fortran code can effectively use less precision than the *de facto* 64-bits. The reader will find an explanation as to why and how we developed this method, with the specific steps of the methodology detailed below. The basic idea behind the method is to perform simulations with a Fortran model using a custom set of precisions and directly assess the outputs to see if the results are accurate enough. To do so we use the RPE tool, a Fortran library that allows us to simulate the result of a floating point operation given that the variables are using a specific number of significand bits. This can be integrated into an actual code to mimic the possible consequences of using reduced precision in certain variables. The advantage of the tool is the flexibility that it offers once implemented into the code, allowing us to easily test any given combination of precisions. The main drawback is the considerable overhead added to the simulations, increasing its cost.

The objective of the method is to find a set of precisions that minimizes the numerical resources used while keeping the accuracy of the results. A set of precisions is a specific combination of the precisions assigned to each variable, with 52^n being the number of possible sets, where n is the number of real variables used in the code and 52 the number of bits used to describe the significand in a double-precision representation. This number makes it prohibitively expensive to explore all combinations. For any real-world code, this holds true even when not considering all possible values between 0 and 52 but just considering double-precision (64-bits), single-precision (32-bits) and half-precision (16-bits), where the number of possible sets is still 3^n.

A feasible alternative is to perform a screening of the variables individually. The idea is to simply perform n simulations to observe what happens when all the variables are retained at high precision except the one that is being tested. While it is true that this approach can give an insight about the precision needed by a particular variable, it may not reveal issues regarding more complex interactions between more than one variable. Additionally, building a complete set of variables from the tests performed on individual variables is not trivial at all i.e. a set which consists of variables that, as individuals, can use lower precision may not behave as such when combined and produce inaccurate results.

Another alternative is a divide-and-conquer algorithm that identifies the sections of the code that cannot handle reduced precision and builds a complete set of variables that can. Starting from a variable set that contains all the real variables that we want to analyze, the approach consists of evaluating what happens when the set uses reduced precision. If the results become inaccurate, we proceed to split the set in two parts which are evaluated individually. The process is recursive and ideally a binary search is performed until the sets that are evaluated contain only a single variable. If a set containing a single variable ends up being inaccurate, this variable is kept in high precision in the preceding sets when they are re-evaluated. The advantage of this approach is that it is cheaper to find all the variables that can individually compromise the results, and in addition it is easier to rebuild a complete set assimilating the results of the simulations of subsets that gave an accurate result.

Nevertheless, there are some elements that prevent this approach from working properly. The non-linearity of most of the Earth science codes implies that the differences between two simulations performed using different numerical precisions will not be constant. In many cases two accurate subsets resulted in an inaccurate set when combined. To increase the confidence in the results, we propose to re-evaluate the sets whose results are accurate with different initial conditions. This method is similar to the ensemble simulation used in ESMs [87], which tries to assess the uncertainty of the simulation outcomes by taking into consideration the uncertainty in the model inputs. The method remains the same adding an extra simulation initialized with different initial conditions for the sets that show accurate results. According to our results, an ensemble of only two members was required to solve most of the issues related with combinations of accurate subsets resulting in inaccurate sets.

The steps of the methodology, which are discussed in the next subsections, consist of:

- Implementing the emulator into the code, completing all the necessary actions to obtain a code that uses the emulator in which it is possible to select the precision of each real variable through a list of parameters.
- Establishing a test that will determine if the results of a simulation are accurate enough.
- Performing a precision analysis by launching the necessary tests to obtain a set of variables that can effectively use reduced precision passing the accuracy test.

5.2.1 Implementing the emulator

The **RPE** is a Fortran library that allows the user to emulate floating point arithmetic using a specific number of significand bits. It has the capability to emulate the use of arbitrary reduced floating-point precision. Its development was motivated by the need to explore mixed-precision approaches in large numerical models that demand more computational power,with weather and climate fields in mind, making the tool highly suitable for our purpose. The emulator is open source, can be accessed through github, has documentation available and a reference paper [86] with more detailed information.

Although the use of the emulator facilitates the testing of different precision configurations without recompiling, in large codes like ESMs the implementation of the emulator can carry more work than expected. The length of the code, the large development time, the quantity of different developers and the lack of a rigid style guide can end up with a large number of exceptions that make it harder to fully automate the emulator implementation, requiring the user to solve the emerging issues.

Our implementation of the emulator has two different parts:

1. Replacing the declaration of the real variables with the custom type **rpe_var**.
2. Introducing a method of selecting the precision of the variables without requiring recompilation of the code.

Replace variable declarations

To use the emulator, the user has to replace the declarations of the real variables with the custom type defined in the emulator library [86]. Even though the idea is quite simple, the practical process in a complex and large state-of-the-art ESM can present several minor issues that can add up to a considerable amount of work. This is a list of some of the specific issues that can be found implementing the emulator:

- All the real variables that were initialized at declaration time need to be initialized providing a derived type variable instead of a real, which requires modification of all these declarations.
- When a hard-coded real is used as a routine argument where the routine is expecting an rpe_var variable, it is necessary to cast this hard-coded variable into a rpe_var type variable. (i.e. "call routine($var, \mathbf{1.0}, var2, \mathbf{0.1}$)" has to be "call routine(var, rpe_var($\mathbf{1.0}$), $var2$, rpe_var($\mathbf{0.1}$))")
- Although it is possible to adapt the RPE library to include intrinsic functions (i.e. max,min) that can use rpe_var variables as arguments, there is still a problem when mixing rpe variables and real variables. The problem can be overcome converting all the variables to the same type.
- When there is a call to an external library (i.e. netcdf, mpi), the arguments cannot be rpe_var and must be reals.
- Read and write statements that expect a real variable cannot deal with rpe_var variables.
- Several other minor issues (pointer assignations, type conversions, used modules,...)

In codes with hundreds/thousands of variables this task can represent months/years of work, and for this reason it is worthwhile to automate the whole process. When all the issues are solved the model should be able to compile and run.

Selecting the precision

To specify at runtime the precision of each individual variable, the method that we use is to create a new Fortran module which includes an array of integers containing the precision value of each one of the real variables of the model. The values of this array will be read and assigned at the beginning of the simulation. After implementing the emulator into a code, one should be able to launch simulations specifying individually the number of significand bits used for each variable and obtain outputs, having completed the most arduous part of the proposed methodology.

5.2.2 Designing the accuracy tests

Once we are able to launch simulations, we must define how we will verify the results, the kind of experiment that we want to perform, and define a True/False test to perform on the outputs.

To define a test to verify results, we must define a function that can be applied to simulation output and determine if the outputs are correct.

To give an example, consider a simulation whose only output is a single scalar value, then we can consider that a given test is accurate if the output and reference match to a certain number of significant figures. Using the value of π as a reference, we can consider that a given simulation is accurate enough if the difference between the reference and the value obtained is smaller than a given threshold, i.e. the results coincide up to a specific number of significant figures. For example, a result accurate to 6 significant figures would pass if the required condition was to coincide with 4 significant figures, but would fail if it had to coincide with 10.

5.2.3 Performing the analysis

We propose a recursive divide and conquer algorithm with few slight modifications. For a given set of variables, we generate a list of parameters that set these variables to use reduced precision and we launch a simulation. When the simulation is completed, we proceed to apply the accuracy test described in section 5.2.2. If the simulation with the specific set passes the test we consider it safe to reduce the precision of this set. If this is not the case, the purpose of the algorithm is to identify which part of the set is responsible for the inaccuracy: we proceed by sub-dividing the set and evaluating its parts separately until we have identified one of the variables that needs to preserve higher precision. The sets that yield inaccurate results require information from the subsets after these have been evaluated to be modified and reevaluated.

This initial approach has some drawbacks: One one hand, the effect of reducing the precision of a set of variables can not be directly deduced from what happens to its individual parts. The simplest case where this can happen is when we deal with two variables that appear in the same arithmetic operation: using the same logic as the actual processors, the emulator performs intermediate operations using the largest precision between the variables involved, so it may happen that having any of the two variables in higher precision will give an accurate enough result but not when both of the variables use reduced precision.

To illustrate this lets consider the following example: having two variables x and y, with $x = 2^6$ and $y = 2^{-6}$, if we compute the sum $z = x + y$ in real number arithmetic, the result is $z = 64.015625$. Both x and y can be perfectly represented using a 10-bit significand, but it is not the case for z that requires more numerical precision. Following the processor logic, if either x or y use a 52-bit significand, the computation of z will be done using 52 bits leading to a correct result, but if both variables are using 10 bits, the computation will be done using 10 bits and will yield a wrong result ($z = 2^6$).

On the other hand, numerical error in most algorithms has a stochastic component which, combined with the non-linearity of these kind of models, can mean that a specific set of variables give accurate results under certain conditions, but gives inaccurate results under different conditions.

For these two reasons we added an extra test into our workflow that consists in assessing the results with different initial conditions whenever the first evaluation result in a positive outcome. While we cannot be sure that performing only a single re-evaluation will be enough, it can be sufficient for the most sensitive cases.

Another slight modification that we added to the initial approach is the definition of stricter thresholds for smaller subsets, to prevent the possibility of errors accumulating in bigger sets.

The entire algorithm is described in the pseudo-code presented in Appendix *Put the annex with the algorithm* , that also contains the instructions that any given set has to follow in order to learn which elements of the set can use reduced precision.

1

5.3 Study cases

In this section we will present a proof of concept of the method using two state-of-the-art models, NEMO and ROMS. For each one of the two models we carried out two different experiments. With NEMO, we performed two analyses using different accuracy tests and both cases have as a target to find which variables can use single-precision (23-bit significand). With ROMS, we performed two analyses with the same accuracy tests but having two different reduced precision targets, single and half precision (23-bit and 10-bit significand). The section is divided in three parts, one part for each model and finally a discussion of the results.

5.3.1 NEMO

NEMO (Nucleus for European Modelling of the Ocean) is a state-of-the-art modelling framework of ocean related engines. The physical core engines for the different parts of the model are the OPA engine for the ocean, LIM for the sea-ice and TOP-PISCES for biogeochemistry [22][29][88]. The range of applications includes oceanographic research, operational oceanography, seasonal forecast and (paleo)climate studies.

Previous performance analyses of NEMO have shown that the most time-consuming routine was not even responsible for 20% of the total computation time ([33]). For this reason, any effort to improve the computational performace of the model cannottarget a single region of the code but something that has to be applied along all the sections.

As explained in section 5.1, previous publications have demonstrated the positive impact that the use of mixed precision approaches can have on the performance of scientific models. Previous experiences in reducing the working precision of NEMO from 64-bit to 32-bit demonstrated a significant change in the results (see Figure 5.1), indicating that blindly reducing the precision in the entire code was not an option.

To make the outcome of this work relevant for the modeling community, the analysis has been performed using the version 4.0b of the code that at the time of writing is the latest version available. The configuration used was an ocean-only simulation using the ORCA2 grid (about 220 km resolution near the equator) and the objective of the analysis is to identify which set of variables can effectively use 32-bit floating point representations instead of the 64-bit standard while keeping the difference with the reference below a chosen threshold.

Emulator implementation

We have developed a tool (see code availability section) that not only modifies the source code to implement the emulator solving all the issues mentioned in the methods section, but also creates a database with information about the sources, including its modules, routines, functions, variables and their relations. This database will have several uses afterwards, since it can be used to generate the list of precisions assigned to each variable and to process the results of the analysis afterwards.

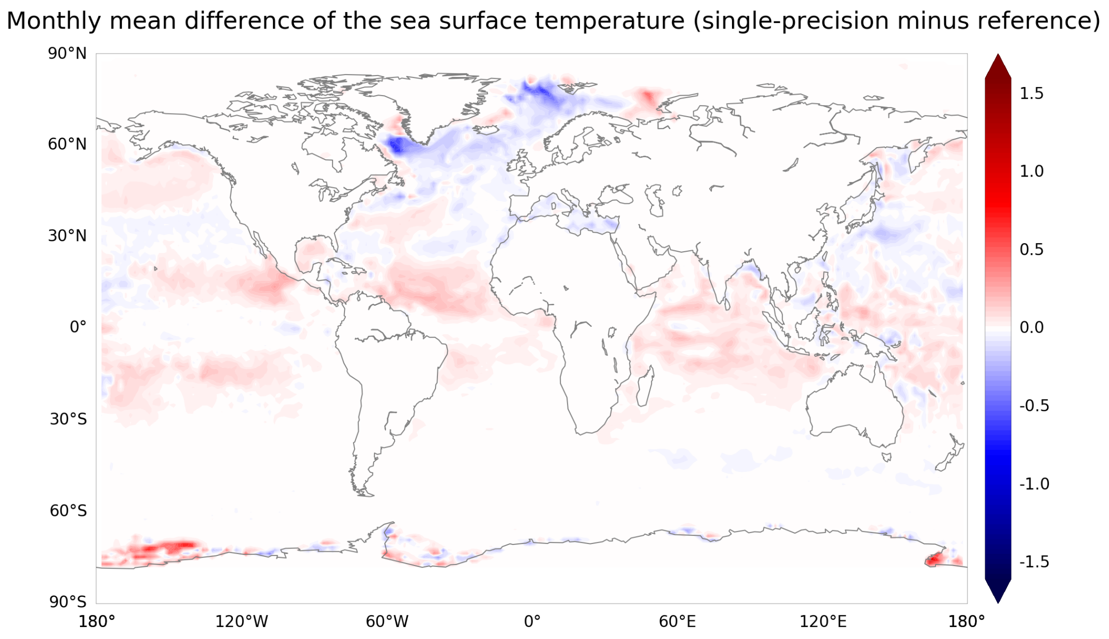
Monthly mean difference of the sea surface temperature (single-precision minus reference)
90°N
60°N
30°N
0°
30°S
60°S
90°S
180°
120°W
60°W
0°
60°E
120°E
180°
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5

Figure 5.1: Difference in sea surface temperature monthly mean (°C) for the first month of simulation between a NEMO 4.0 simulation performed emulating single precision arithmetic and a NEMO 4.0 simulation performed using double precision arithmetic. The monthly mean of the sea surface temperature for the first month of simulation shows localized biases exceeding 1.5 °C.

The code largely relies on MPI and netCDF. Since there is no special interest in analyzing these routines, a simple solution is to keep them unmodified. The selection of the source files that should not be modified requires user expertise. After that the tool handles all the necessary workarounds to ensure that the proper variable type is passed as routine arguments.

Designing the accuracy tests

Our approach was to define a metric to evaluate how similar two simulations are, and define a threshold above which we will consider simulations inaccurate.

The metric used to evaluate the similarity between two simulation outputs is the Root Mean Square Deviation divided by the Interquartile Range. It is computed for each time step and the maximum value is retained.

$$RMSD(t)) = \sqrt{\frac{\sum_{i=1}^{i_{max}} \sum_{j=1}^{j_{max}} \sum_{k=1}^{k_{max}} \left(ref_{i,j,k}(t) - x_{i,j,k}(t)\right)^2}{i_{max} \cdot j_{max} \cdot k_{max}}} \tag{5.1}$$

where i, j, k are the spatial axis indices, t is the time index, $ref_{i,j,k}(t)$ is the value of the reference simulation at a given point i, j, k and a given time t and $x_{i,j,k}(t)$ the value of simulation that is being evaluated at the same point i, j, k and a given time t.

$$IQR(t) = Q3(t) - Q1(t) \tag{5.2}$$

where *Q3* and *Q1* are the values of the third quartile and first quartile respectively, and t is the time index.

So the final metric will be the accuracy score:

$$AccuracyScore = \max_{\forall t} \left(\frac{RMSD(t)}{IQR(t)} \right) \tag{5.3}$$

The final accuracy test can thus be defined as:

$$AccuracyTest(AccuracyScore, threshold) = \begin{cases} \text{True} & \text{if } AccuracyScore < threshold \\ \text{False} & \text{if } AccuracyScore \geq threshold \end{cases} \tag{5.4}$$

To be able to use the test defined above we need a reference simulation and to establish the thresholds. To obtain the reference simulation, first we have to define what we want to compare and the kind of test. For this analysis, we are using as a reference a 64-bit real variables version of the code. The runs consisted of 10-day simulations that produced daily outputs of the 3D temperature field, the 3D salinity field, and the column integrated heat and salt content. Temperature and salinity were selected because these are the two active tracers that appear in the model equations. Reference simulations were launched for each different initial condition that was used later in the analysis.

In order to define some meaningful thresholds, we performed a simulation using a halved time-step to compute the accuracy score against the reference simulation, to use this value as a first guess to define our thresholds (see Table 5.1). To show how defining different thresholds leads to different results and to emphasize how with this method is possible to keep arbitrarily small output differences, two different thresholds were defined (see Table 5.1). The first of the two thresholds is defined to be one thousand times smaller than the differences obtained using a halved time-step. The second is defined to be only ten times smaller. We will refer these cases as the tight case and the loose case respectively.

Variable	Accuracy Score half-time step (AS)	Thresholds tight case ($AS \times 10^{-3}$)	Thresholds loose case ($AS \times 10^{-1}$)
Temperature (3D field)	1.49×10^{-3}	1.49×10^{-6}	1.49×10^{-4}
Salinity (3D field)	6.41×10^{-3}	6.41×10^{-6}	6.41×10^{-4}
Heat Content (2D field)	2.74×10^{-3}	2.74×10^{-6}	2.74×10^{-4}
Salt Content (2D field)	5.47×10^{-6}	5.47×10^{-9}	5.47×10^{-7}

Table 5.1:: Accuracy score of the simulation performed using a time-step shorter than the reference and the different thresholds used for the different variables in the tight and loose cases. The thresholds for the tight case are defined multiplying the accuracy score of the half-time step simulations by 10^-3 and the thresholds for the loose case are defined multiplying the accuracy score of the half-time step by 10^-1.

Executing the tests

To execute the tests, we implemented the rules described in the methodology section developing a python workflow manager capable of handling the dynamic workflow by creating the simulation scripts, launching them on a remote platform, then checking the status and the adequacy of the results.

To perform tests, the Marenostrum 4 supercomputer has been used (see table 5.2), with each individual simulation taking about 8 minutes 30 seconds on a single node.

During the implementation of the emulator, the declarations of more than 3500 real variables were replaced with emulator variables. These variables can be scalars or arrays of up to 5 dimensions. They can be global variables or just temporary variables used only in a single line of code inside a subroutine. In a large code like NEMO there are usually parts of the code that are either used or not depending on the specific parameters employed. For this reason even when more than 3500 variables were identified, using our specific configuration only 942 are used. The variables that are used are identified during the runtime. Given that considering the unused variables for the analysis would be useless and dangerous (as incorrect conclusions could be drawn about which precision is needed) those are simply omitted from the analysis. The initial set used to start the analysis will therefore be the set containing the 942 variables that are actually used during our specific case.

Discussion

Starting with the evaluation of the initial set containing all the variables, the results show that a global reduction of the precision changes the results beyond what is acceptable. After running the full analysis for both cases we can see that the results differ. On one hand the tight case shows that 652 variables (69.2%) could use single-precision and on the other hand in the loose case the obtained solution contained 902 variables (95.8%), keeping only 40 variables in higher precision. As expected, defining looser thresholds allows the use of lower-precision in a larger portion of the variables. Table 5.3 presents the results split by arrays' dimension. Also, the tighter case required more tests, with 1442 simulations required to arrive to the solution (consuming 204.3 node-hours), while the loose case required only 321 (consuming 45.5 node-hours).

The method ensures that the differences between the reduced-precision simulations performed with the final variable sets and the reference are below the determined thresholds. Although the accuracy test used for this example was not designed to ensure the conservation of global quantities, Figure 5.2 shows that the global heat and salt content of the simulations performed using the loose configuration resembles those of the reference simulation performed fully in double-precision, which was not the case for a simulation fully performed using single-precision.

The Figure 5.3 shows the expected impact in the memory usage when using the set obtained with the loose accuracy test. Since the results produced by both cases

Processor	2 sockets Intel Xeon Platinum 8160 CPU with 24 cores each @ 2.10GHz for a total of 48 cores per node
Memory	L1d 32K; L1i cache 32K; L2 cache 1024K; L3 cache 33792K 96 GB of main memory 1.880 GB/core
Network	100 Gbit/s Intel Omni-Path HFI Silicon 100 Series PCI-E adapter 10 Gbit Ethernet
Local Storage	200 GB local SSD available as temporary storage during jobs

Table 5.2:: Marenostrum 4 node specifications

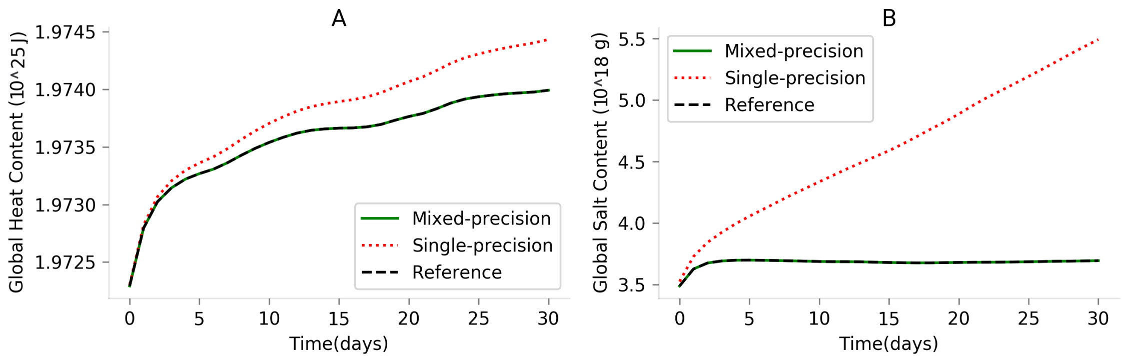

Figure 5.2: Daily averages of the global heat content (A) and the global salt content (B) for one month simulations using NEMO 4. The plot of the global salt content (B) has an offset of 4.731×10^{22} grams. The reference simulation was performed using double-precision, the single-precision simulation was performed using single-precision arithmetic for all the variables and the mixed-precision simulation was performed keeping 40 of the 942 model real variables in double precision and using single precision for the remaining 902 variables.

are so close to the reference simulation, Figure 5.3 only demonstrates the loose case since we anticipate that a mixed-precision implementation that is accepted by the scientific community would be more similar to the loose case.

5.3.2 ROMS

The Regional Ocean Model System (ROMS) is a free-surface, hydrostatic, primitive equation ocean model that uses stretched, terrain-following coordinates on the vertical and orthogonal curvilinear coordinates on the horizontal. It contains a variety of features including high-order advection schemes; accurate pressure gradient algorithms; several subgrid-scale parameterizations; atmospheric, oceanic, and benthic boundary layers, biological modules, radiation boundary conditions, and data assimilation [89].

The experiments performed with ROMS were done applying the primal form of incremental strong constraint 4D-Var (4DVAR) [90] . The configuration used is the U.S. west coast at 1/3 degree horizontal, referred to as WC13, a standard model test case. It has 30 km horizontal resolution, and 30 levels on the vertical [89]. This configuration was selected because 4DVAR ROMS has a large community of users, there is an easy-to-follow tutorial to set-up the configuration and involves linear models that make this an interesting choice to expand the results obtained in Section 5.3.1.

To perform this analysis, we set up the tutorial available online that performs a I4DVAR data assimilation cycle that spans the period 3-6 January, 2004. The ob-

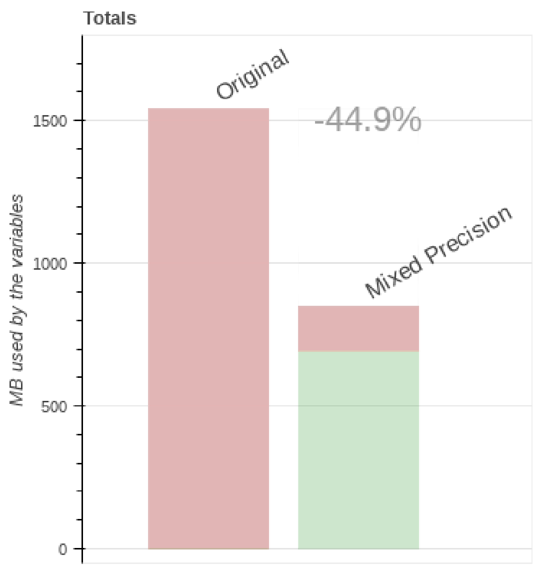

Figure 5.3: Estimation of the memory usage impact in an ORCA2 configuration using the set obtained with the loose accuracy test. The estimation has been performed using the dimensions of the variables and the size of the domain. In red (darker) we have the memory that is occupied by variables in double precision (100% in the original case), in light green the proportion of the memory occupied by variables in single precision, the difference between the original case and the loose case implies a potential 44.9% decrease in memory usage.

		Tight case		Loose case	
Dimension	Total variables	n	%	n	%
0	374	304	81.3	366	97.9
1	32	23	71.9	32	100.0
2	322	216	67.1	307	95.3
3	200	105	52.5	190	95.0
4	13	3	23.1	6	46.2
5	1	1	100.0	1	100.0
TOTAL	942	652	69.2	902	95.8

Table 5.3:: The table presents the total number of variables that were included in the analysis split by the dimension of the arrays (0 for scalars). Also, the number of variables that can safely use single-precision for the two cases discussed in section 5.3.1 (tight and loose) and the percentage of the total variables that this represents. Using tight constrains 69.2 % of the variables can use single-precision, although only 52.5 % of the 3D variables are responsible for the most memory usage. Using loose conditions, 95.8% of the variables can use single-precision, including 95% of the 3D variables.

servations assimilated into the model are satellite sea surface temperature, satellite sea surface height in the form of a gridded product from Aviso, and hydrographic observations of temperature and salinity collected from Argo floats and during the GLOBEC/LTOP and CalCOFI cruises off the coast of Oregon and southern California, respectively.

With the I4DVAR version of ROMS, most of the computational time is spent inside the data assimilation “inner loops”, within the adjoint and tangent linear sub-models (AD and TL respectively). To give some numbers, in a simulation with the WC13 configuration, 35% of the time is spent in the TL model and 50% in the AD model while the time spent in the nonlinear model is below 14%. It was felt that since the TL and AD models are used to compute an approximation gradient of the I4DVAR cost function, further approximations in the gradient resulting from lower precision will probably be not so detrimental leading to our starting hypothesis that the AD and TL models are better targets than the nonlinear model. However, for this hypothesis to be true the AD needs to keep being the transpose of the TL to within a chosen precision, otherwise it might prevent the convergence of the algorithm. For this reason, we are trying to minimize the precision used in these regions of the code. In this case, the target reduced precision is not limited to single-precision (23-bit significand) but also the half-precision is explored (10-bit significand).

It is important to remark that in our experiments the number of bits used to represent the exponent of the variables is 11 (the same number as used in double-precision), no matter which number of bits is used for the significand. In practice, it means that to ensure that it is possible to reduce the precision to half-precision we would need to ensure that the values do not exceed the range that can be represented with a smaller exponent. However, the results are still interesting because we can learn the potential use of reduced number of bits for the significand.

Emulator Implementation

One aspect that made this exercise different from that of NEMO in terms of implementing the emulator was that the interest was focused on specific regions of the code, i.e. the parts related to the tangent linear model and the adjoint model.

Designing the tests

When running a data assimilation experiment, the objective is to obtain a coherent state of the ocean that minimizes the difference between the model and the observations. Through different forward-backward iterations using the TL and AD models, the model should converge to a state where the cost function (function that describes the difference between the model state and the observations) is minimum. We can consider that a simulation is accurate enough if the model converges to the same result as the double-precision reference. Through the different iterations, the solution should converge to a minimum. To set a threshold for the accuracy of the simulations, we can look at the difference between the last two inner-loop iterations. In the reference simulation the value of this difference computed as a cost function defined in the model is is of 1.77×10^{-1} . Defining a threshold 10 times smaller (1.77×10^{-2}) ensures that the impact of reducing the precision will be smaller than changing the number of inner iterations performed.

Executing the tests

For ROMS, we use the same workflow manager developed for NEMO, simply using different templates for launching simulations and evaluating the results.

Discussion

As in the case with NEMO, we first performed the reference executions for all the different initial conditions, using 64-bit precision.

The parts selected for the analysis contain 1556 real variable declarations, of which 1144 are actually used in our case. Starting with this initial set and trying to use a

23-bit significand, the results showed that in fact all the variables can use reduced precision at once while still obtaining accurate results.

This suggests that this model and in particular this specific configuration are suitable for a more drastic reduction of the numerical precision. To test that, we proceeded with a new analysis using a 10-bit significand. The first simulation of the analysis crashed, not providing any output and showing that it is not possible to reduce the precision to 10-bit overall. Using the analysis to find where this precision can be used, the results show that a 80.7% of the variables can use 10-bit instead of 52. Looking to the effects produced by the variables that were not able to use 10-bit precision we could see that only a single variable was preventing the model from completing the simulations, even though many others were introducing too many numerical errors. In Table 5.4 the results are presented split by dimension.

		23-bits		10-bits	
Dimension	Total variables	n	%	n	%
0	326	326	100.0	277	85.0
1	82	82	100.0	71	86.6
2	450	450	100.0	385	85.6
3	173	173	100.0	135	78.0
4	78	78	100.0	44	56.4
5	31	31	100.0	9	29.0
6	4	4	100.0	2	50.0
40	2	2	100.0	2	100.0
TOTAL	1146	1146	100.0	925	80.7

Table 5.4:: The table presents the total number of variables that were included in the ROMS analysis split by the dimension of the arrays (0 for scalars). Also, the number of variables that can safely use reduced-precision for the two cases discussed in section 5.3.2 (single and half precision) and the percentage of the total variables that they represent. The single-precision case was trivial since all the variables considered can use single-precision while keeping the accuracy. For the half-precision case, 80.7% of the variables could use half-precision, although the proportion of variables that can use reduced precision becomes less as the dimension increases.

5.3.3 Common discussion

From the two exercises shown in section 5.3 we can draw several conclusions. The most important is that the method can provide a set of variables that can use reduced numerical precision preserving the accuracy of the results, as it was observed in the four cases explored (two different accuracy tests for NEMO and two target precisions for ROMS). The results in the four cases showed that the potential to reduce the numerical precision is considerable even in the most constrained cases (i.e. NEMO tight). In the NEMO case, the analysis covers the full model, while the ROMS case was focused only on a part of the model, demonstrating the versatility of the method. It has also been shown that it is possible to achieve results that are arbitrarily close to the reference, while requiring more exact results will have the cost of leaving more parts of the code in higher precision.

5.4 Conclusions

Previous works suggested that the generalized use of double precision for Earth Science modelling is a case of over-engineering and that for this reason adapting the computational models to use the smallest numerical precision that is essential would compensate in terms of improvement in performance. However, an improper reduction can lead to accuracy losses that may make the results unreliable. In this manuscript we presented a method that was designed to solve this problem by finding which variables can use a lower level of precision without degrading the accuracy of the results.

The method was thought and designed to be applied to computational ocean models coded in Fortran. It relies on the Reduced Precision Emulator tool that was created to help understand the effect of reduced precision within numerical simulations, it allows us to simulate the results that would be produced in case of performing the arithmetic operations in reduced precision, using a custom number of significant bits for each floating point variable. The proposed analysis algorithm finds which variables need to be kept in double precision and which ones can use reduced precision without impacting the accuracy of the results.

The method has been tested with two widely used state-of-the-art ocean models, NEMO and ROMS. The experiences with the two models pursued different objectives. With NEMO, the analysis covered all the routines and variables used within the ocean-only simulation, the target precision was 32-bits and we explored how the selection of a specific accuracy tests changes which variables can safely use reduced precision. However with ROMS, the analysis covered only the variables belonging to the adjoin and tangent linear models, using a single accuracy test and examining how the method can be used to discover the viability of using numerical precisions below 32-bits.

The results presented in this work allows us to draw some conclusions. It is shown that both models can use reduced precision for large portions of the code proving the feasibility of mixed precision approaches, and the method described is able to find these. The method can also provide a configuration that can be arbitrarily close to the double-precision reference, where the amount of variables that can use reduced-precision will depend on how strict the imposed conditions are.

Users might want to follow the method presented here to build a version of the model that can benefit from the reduction of the precision and can be used by the whole community without concerns.

5.A Search Algorithm

```
if STATUS is PENDING then
    submit job
    set STATUS to RUNNING
else if STATUS is RUNNING then
    Check job status in remote machine
    if SIMULATION was SUCCESSFUL then
        Assert results and set STATUS to ASSERTING
    else
        if the set can be divided then
            Create and submit sub-sets and set STATUS to SUSPENDED
        else
            Set STATUS to FAILED
        end if
    end if
else if STATUS is ASSERTING then
    Check assertion job status in remote machine
    if job status is COMPLETED then
        Check results:
        if results are successful then
            Set STATUS to SUCCESS
        else
            if  set can be divided then
                Create and submit sub-sets and set STATUS to SUSPENDED
            else
                Set STATUS to FAILED
            end if
        end if
    end if
else if STATUS is SUSPENDED then
    Check subsets STATUS
    if subsets are FAILED or SUCCESS] then
        if Both subsets FAILED then
            Set STATUS to FAILED
        else
            Integrate subset information and set STATUS to PENDING
        end if
    end if
end if
```

Algorithm 1: The analysis algorithm describes the actions that have to be taken in a given set that has a given status. Using that framework, the list of possible statuses is: **Pending**: the set is ready to be evaluated, **Running**: the set is being evaluated, **Asserting**: the set had a successful evaluation and it is ready to be reevaluated using a different configuration, **Suspended**: the set had an unsuccessful evaluation and now it is waiting for the results of its child subsets, **Failed**: the set had a unsuccessful evaluation and can not be further divided.

Chapter 6

Discriminating accurate results in nonlinear models

This thesis chapter originally appeared in the literature as Oriol Tintó, Mario Acosta, Miguel Castrillo, Stella V. Paronuzzi Ticco, Kim Serradell, Ana Cortés, Francisco J. Doblas-Reyes ***Discriminating accurate results in nonlinear models****, 2019 International Conference on High Performance Computing and Simulation (HPCS), Dublin, 2019*

Extended Abstract: Non-linear models are challenging when it is time to verify that a certain HPC optimization does not degrade the accuracy of a model. Any apparently insignificant change in the code, in the software stack, or in the HPC system used can prevent bit-to-bit reproducibility, which added to the intrinsic nonlinearities of the model can lead to differences in the results of the simulation. Being able to deduce whether the different results can be explained by the internal variability of the model can help to decide if a specific change is acceptable. This manuscript presents a method that consists in estimating the uncertainty of the model outputs by doing many almost-identical simulations slightly modifying the model inputs. The statistical information extracted from these simulations can be used to discern if the results of a given simulation are indistinguishable or instead there are significant differences. Two illustrative usage examples of the method are provided, the first one studying whether a Lorenz system model can use less numerical precision and the second one studying whether the state-of-the art ocean model NEMO can safely use certain compiler optimization flags.

6.1 Introduction

When trying to optimize a computer code to make it faster and/or more efficient, one requisite is to avoid degrading the accuracy of the results. In many cases this can be trivial to evaluate, but in the case of computational models that try to simulate nonlinear systems (i.e. Navier-Stokes equations of fluid dynamics widely used in weather and climate models) this is far from being trivial. In these cases, any slight perturbation in the input data, any apparently inoffensive change in the algorithms, a change in the software stack, the specific compilation flags used or the use of a different machine can easily lead to differences in the final results. Dealing with such situations is not easy and it is an open topic in computational science [91] [92]. In many cases, trying to achieve exactly the same results is possible [93] but difficult, expensive and, above all, possibly useless. In a computational model there are many potential sources of error (model inputs, numerical methods, compiler optimization . . .) and, as a consequence, the results produced by a simulation will have an inherent uncertainty. In problems that are highly conditioned by the initial conditions, such as weather prediction, minimizing the errors due to the numerical approximations or the computational implementation will have little or no effect on the accuracy if the model inputs still have uncertainties. In these cases, it is legitimate to ask ourselves if it is worth to invest resources on minimizing the model errors when this will have no impact on the accuracy of the results. Nevertheless, even accepting that any change will affect the outputs, it is still necessary to have the capacity to determine if a certain change produces differences that are within model variability of beyond. Although the literature has examples of other attempts to handle the same problem [94], this manuscript proposes a simple method that can be easily used in a variety of cases. It consists in measuring how an initial perturbation evolves in a series of simulations to obtain a reference that will be used to determine if a certain case under evaluation is within the variability of the model. To illustrate how the method can be used, two examples are provided. In the first of the two examples, we change the numerical precision used to simulate a Lorenz system and study whether the change in the precision impacts the quality of the results. In the second example, we compile the state-of-the art ocean model NEMO with different compiler optimization flags and evaluate whether the optimization applied degrades the quality of the results.

6.2 Method

In order to establish if a certain change in a computational model (a different numerical method, different compilation flags, a different platform,...) changes the results beyond what can be acceptable we have to start defining a test case. A test case usually will be composed by specific model inputs (initial conditions, simulation parameters, simulation length,...), the software (source code, compiler, software stack) and the hardware (a specific platform).

With all these elements settled, we compute the test case and define the outputs obtained as our reference outputs. Having the reference outputs, we can then evaluate the differences between any given simulation and our test case. To achieve this we use the root mean square error (RMSE), whose definition can be found in equation 6.1, generating a time-series of the error for each variable being evaluated.

$$RMSE(t) = \sqrt{\frac{\sum_{i=1}^{i_{max}} \sum_{j=1}^{j_{max}} ... \sum_{n=1}^{n_{max}} \left(ref_{i,j,...,n}(t) - x_{i,j,...,n}(t)\right)^2}{i_{max} \cdot j_{max} \cdot ... \cdot n_{max}}} \tag{6.1}$$

Where i,j, ..., n are the dimension indices.

At this point, we are only able to distinguish whether the results are exactly the same and the RMSE for all the evaluated variables is exactly 0 or if the results present some difference. In case of having a difference, we are still not able to explain if this difference is relevant or not.

In order to have something to compare with, we propose to produce a series of simulations that are almost identical to the reference case , but where the initial conditions have been slightly modified. This is widely known in Earth sciences as an ensemble of simulations. In these cases, the simulation parameters, the source-code, and everything but the initial conditions of the simulation must be exactly like the reference case. For each one of those simulations and for each output variable being evaluated we produce the RMSE time-series. At this point, by plotting the time-series we should be able to see how the RMSE of each case follows a different trajectory due to the non-linearity of the model.

With all this done, in order to evaluate the impact that a certain change has in the outputs, we just need to launch the simulation with the specific conditions that we want to test, compute the RMSE of each variable against the reference simulation and compare these RMSE time-series with the time-series of the ensemble simulations. If the time-series of the case being evaluated are similar or below the ensemble, we consider that the results are good enough, and if on the contrary the time-series show that the RMSE grows faster than the members of the ensemble, we consider that the results are not good. This is illustrated in Figure 6.1

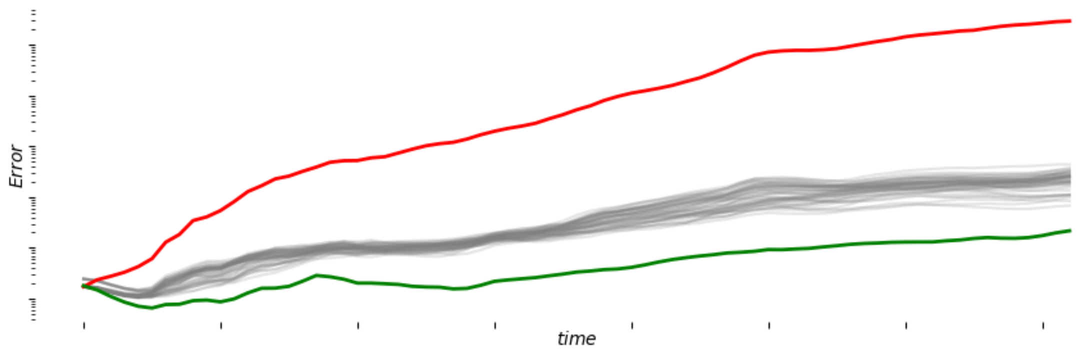

Figure 6.1: Illustrative case of time-series of the error measured on a given value for each ensemble simulation is computed (gray) which allows the reader to discriminate if the error of a given case being evaluated is below the model uncertainty (green) and therefore the model is virtually the same or above (red) and therefore the changes modified what the model is simulating.

6.3 Examples

6.3.1 Adjusting numerical precision for a Lorenz System simulation

A Lorenz System is a simple system of equations (see equation 6.2) that shows chaotic behaviour. This implies that that the slightest perturbation in the system (inputs, numerical methods, ...) would make that the results of two very similar simulations end up completely different.

$$\frac{\partial x}{\partial t} = \sigma(y - x), \frac{\partial y}{\partial t} = x(\rho - z) - y, \frac{\partial z}{\partial t} = x \cdot y + \beta z \tag{6.2}$$

To study if the precision can be reduced without degrading the quality of the results we apply the method explained in the previous section. We perform a simulation using the reference code in double precision with parameters $\sigma = 10$, $\rho = 28$ and $\beta = 3/8$, with the initial conditions being $x = y = z = 1$. After that, we launch one hundred simulations with the same code but adding a slight perturbation ($p \propto 10^{-4}$) to the initial conditions. In Figure 6.2 it can be seen how the system behaves chaotically and the members of the ensemble that have very similar initial conditions (grey lines) have very different trajectories after 15 seconds.

Using a reduced precision emulator[86] that allows us to simulate what would happen if the simulations were performed using floating-point representations with a specific number of significant bits, we performed tests with different cases and evaluated them against the ensemble. The results that can be seen in Figure 6.2.

In the top part of the figure it can be seen that using a 13-bit significant floating point representation (red line) the resulting simulation is significantly different from the reference simulation (dashed black line), far beyond the variability measured by the ensemble, while the simulation performed with a 23-bit significant floating point representation is closer to the reference than the ensemble members. In the bottom part of figure it can be seen that with a simple visual inspection of the two reduced-precision cases against the ensemble one can determine if the approximation degrades the results beyond model uncertainty (like the red case) or the error introduced is negligible compared to the model uncertainty due to other error sources (green case).

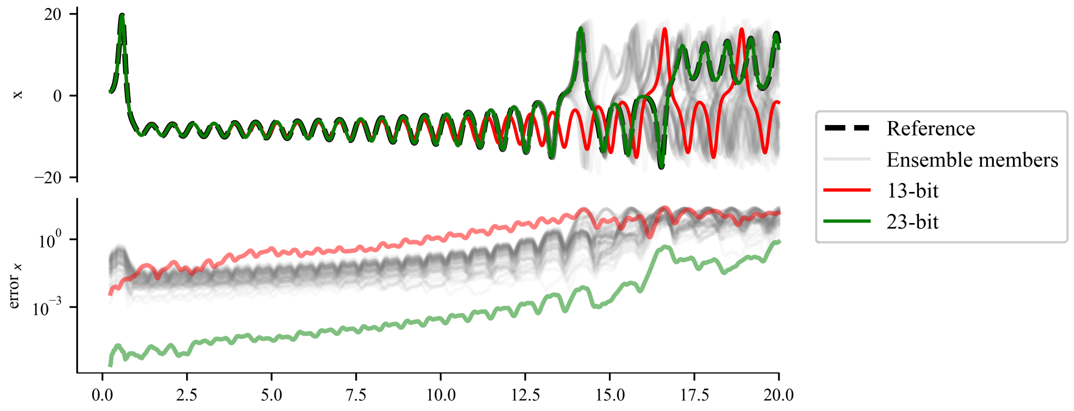

Figure 6.2: Numerical integration of the Lorenz system described in equation 6.2. In the top part of the figure, the dashed black line represents the reference solution, the gray lines represent members of the ensemble, the red line represents a case using 13-bits in the significant and the green line represents a case computed using 23-bits in the significant. In the bottom, the logarithm of the error in x between the reference and the ensemble members (gray), the 13-bit case (red) and the 23-bit case (green).

6.3.2 Trying different compiler flags in NEMO4

NEMO is a state-of-the-art ocean model and it is based on the Navier-Stokes nonlinear partial differential equations. In order to use the NEMO source-code to perform simulations on a supercomputer we have to rely on compilers that build the actual executable. Compilers can use approximations to obtain faster programs, and some of these approximations can affect the results of the floating point arithmetic. In order to determine if the usage of an specific compiler optimization degrades the accuracy of the model, we use the method explained in this manuscript. Our test case is a one year simulation with an ORCA2 grid (two degree horizontal resolution) with which we perform a one year simulation without compiler optimization. Afterwards we launch an ensemble of simulations with the same version of the executable but starting with perturbed initial conditions (the temperature field is modified with white noise), and measure how much the ensemble members differ from the reference in the many variables that the model can output. We use this information to evaluate the effects of using certain flags with the Intel compiler on the Marenostrum 4 supercomputer hosted by the Barcelona Supercomputing Center. Modern compilers have a very extensive list of possible optimization methods that can be activated or deactivated through a compilation flag. Understanding the specific action of each flag goes beyond the scope of this manuscript, which only aims to illustrate the potential usage of the method. To do so we tested the most usual optimization flags (O1, O2, O3 and xHost).

In the results shown in Figure 6.3 it can be seen that while it is true that compiler optimization leads to different results, with the flags **O1**, **O2** and **O3** the results are comparable to those generated without optimization (see Figure 6.3). However, the flag **xHost**, which enables the use architecture specific instructions, leads to results that clearly differ from the ensemble for the global sum of the sea surface height (see Figure 6.3 B).

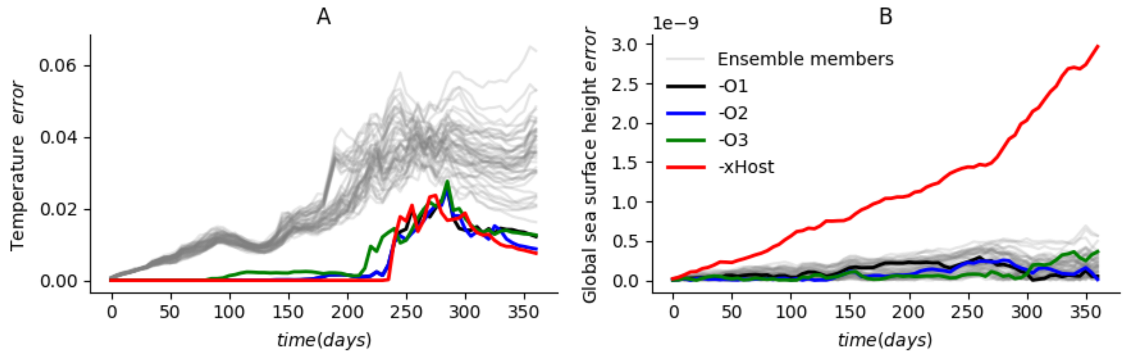

Figure 6.3: Lower is better. A) Root mean square error of the 3D temperature field. B) Error in the global sea surface height.

6.4 Discussion

The method presented in this manuscript allows us to discriminate with a fairly solid basis whether a model modification degrades the results or these remain within its intrinsic uncertainty. It has been successfully used in two very different exercises, proving its usefulness. The Lorenz system example illustrated how by simple visual inspection it can be seen if the results are degraded beyond model uncertainty or not. The NEMO example showed that while compiler optimization leds to different results, these results remained within model variability except for the use of -xHost optimization which caused the degradation of the results for the global sea surface height. While there is room for more theoretical considerations and improvements have to be made in order to generalize even more this method, it can be a valuable resource to validate results from non-linear models. It can be specially useful for computer engineers trying to optimize the computational performance of models with non-linear behaviour and who do not have a strong background on the scientific field.

Chapter 7

Conclusions and Open Lines

7.1 Conclusions

The main objective of the thesis presented here was to identify and tackle the computational challenges that ocean simulation faces in order to exploit modern and future High Performance Computing systems. Ocean models, even that if are cutting-edge scientific tools, rely on legacy codes that were thought to be executed in small clusters or even in serial machines. This situation has led to codes that present difficulties to scale preventing them from exploiting modern and future supercomputers. It was clear that understanding the computational behaviour of an application was necessary in order to improve its performance. This led to a performance analysis of NEMO, which highlighted the role of inter-process communication. To better understand this role, a methodology to dissect it and characterize communication-related issues was developed. The methodology was published in a peer-reviewed journal and led to a code optimization that achieved a 49% (46%) gain in throughput for low (high) resolution global grids. The fact that the methodology highlighted communication issues and led to valuable optimization proved that this kind of analyses can significantly help model developers to adapt their codes to be used more effectively with supercomputers.

The performance analysis also showed that the computational time was widely distributed in different routines, therefore needing optimization to affect all different regions of the model to have significance. One of the decisions that affect all the regions of the code is how you divide the computational domain in order to distribute the work among processors. In the second contribution we proved that the impact of the domain decomposition and the removal of the land-processes in NEMO have been alarmingly underestimated. In certain circumstances NEMO's default domain-decomposition led to a decomposition that was sub-optimal because it didn't minimize the overlapping nor maximize the possibility to remove land-only sub-domains. The results showed that taking into account these factors, it is not only possible to save resources but also it allows the model throughput to increase, achieving a 41% gain using NEMO3.6 with an ORCA025-LIM3 configuration running in the Marenostrum 3 supercomputer. Beyond these results, it became clear that the domain decomposition can not be ignored for any longer. It illustrated also that simple solutions can have a very positive impact in the usage of computational resources.

After the improvement in the understanding of the model achieved in the first part of the thesis, the optimization that allowed to improve the maximum throughput of the model, and trying to figure out potential optimization with significance along the whole code, the attention focused on mixed-precision algorithms. The main idea from mixed-precision algorithms is that a proper usage of numerical precision would allow computational performance to benefit without sacrificing accuracy. In ESM's, the fact that the input data that is fed to the models has a significant uncertainty raised the question about whether using 64-bit to represent real numbers was really needed. Further investigation confirmed that in fact the use of 64-bit precision for all the real variables was indeed unnecessary. However, an improper reduction of precision can introduce too much numerical error leading to accuracy losses. To avoid that, the work presented in the third contribution presented in this thesis introduced a method to find out the precision required for each one of the real variables of a code while fulfilling certain accuracy requirements. The method was not only tested in NEMO but also using the Regional Ocean Modelling System (ROMS), showing that in both models most of the variables could use less than 64-bit precision without problems. The method also proved to be useful independently of how we define whether a result is accurate or not, and that defining stricter thresholds leads to the requirement of keeping more variables in higher precision.

The precision analysis method proved to be a very useful tool to learn which variables of an ocean model can use less numerical precision, but it requires a way to discriminate whether the results are accurate or not. Although a simple question, it can be quite hard to find an answer. The butterfly effect in ill-conditioned problems represents a challenge when it is time to verify that the result of a simulation is correct. When numerically solving these kind of problems, any apparently insignificant change in how the problem is solved can lead to results that are not exactly the same. Being able to discriminate whether different results can be explained by the internal variability of the system itself or not is crucial, since it might help to decide whether a proposed change with the aim of increasing the computational performance can be introduced or not. In the fourth contribution this problem is tackled. In it, a method to discriminate accurate results is introduced and used in two illustrative cases. In the first case, it was used to establish if simulations of a Lorenz system performed using reduced precision arithmetic were accurate enough. In the second case, the method was used to evaluate if a set of compiler optimizations were modifying results beyond an acceptable threshold or not. Although the examples presented in the contribution are simple, the relevance of having a method to discriminate accurate results goes much beyond that. It will allow computer scientists to develop code optimization with the confidence of having a reliable method to actually test if the proposed changes degrade the model accuracy or not. It will also be used along with the method proposed in the third contribution to find which variables in NEMO can use reduced precision as it is a way to discriminate whether results are accurate or not, being as strict as necessary but also as flexible as possible.

In conclusion, the main objective of the thesis presented here was to identify and tackle the computational challenges, and the contributions achieved during its realization are a good step in that direction. The results of this thesis are very satisfactory and promising. The methodology used to analyze the model was generalized and can be used to analyze other scientific models. The methodologies proposed allowed to substantially improve the performance of NEMO. The improvements done regarding communication and domain decomposition have been adopted by the community, with the subsequent savings that will represent, being currently in production in different research centers. The method to analyze the numerical precision required led to very valuable information that will lead to a mixed-precision version of NEMO, positively impacting most NEMO users allowing a more efficient use of their computational resources. Moreover, most of the contributions have surpassed the original scope of the thesis and would be easily transferable to other computational models.

7.2 Future Work

The most important piece that is left after the conclusion of this thesis it to provide the community with a working version of the NEMO model which optimally uses numerical precision. In the work included in the fifth chapter of this thesis the most difficult part, knowing which variables can use less precision, is already solved. What is left to provide a proper version of the model working is to solve minor technical details in the code implementation. A work that is expected to be completed in the following months.

There are also other topics that can be further exploited: the main contributions of the thesis are not specific developments to improve the performance of the NEMO model but the methodologies developed to find and solve some of the computational problems that can exist in these kind of models. These methodologies have the potential to be very useful for other computational models, especially the one related with the analysis of the numerical precision required to run a model. Additionally, it would be interesting for the community to regularly repeat the kind of analyses that were proposed in order to better track and document how the computational performance of the model evolves, especially having in mind that supercomputers are renewed regularly and their changes can deeply affect the performance. It would be good also to extend the analysis method to other ocean models that are similar in order to gain insight about how they compare in terms of performance.

Outreach

During the realization of the doctoral thesis I had the opportunity to publish some of the results obtained and to participate in several outreach activities that helped me to obtain feedback and inform colleagues about my work. The activities included national and international scientific congresses where I could present my work via oral presentations or posters. In the following sections a list of the outreach activities can be found, with an additional part dedicated to the international research stay.

Peer-reviewed publications

- **Oriol Tintó**, Mario Acosta, Miguel Castrillo, Ana Cortés, Alícia Sanchez, Kim Serradell, Francisco J. Doblas-Reyes *Optimizing domain decomposition in an ocean model: the case of NEMO*, Procedia Computer Science, 2017, Volume 108, Pages 776-785, ISSN 1877-0509,DOI:10.1016/j.procs.2017.05.257. , Computer Science Conference Rating: A

- **Oriol Tintó Prims**, Miguel Castrillo, Mario C. Acosta, Oriol Mula-Valls, Alicia Sanchez Lorente, Kim Serradell, Ana Cortés, Francisco J. Doblas-Reyes, *Finding, analysing and solving MPI communication bottlenecks in Earth System models*, Journal of Computational Science, 2018, ISSN 1877-7503, DOI: 10.1016/j.jocs.2018.04.015. , IF:1.078 Q2

- **Oriol Tintó**, Mario Acosta, Andrew M. Moore, Miguel Castrillo, Kim Serradell, Ana Cortés, Francisco J. Doblas-Reyes *How to use mixed precision in ocean models: exploring a potential reduction of numerical precision in NEMO 4.0 and ROMS 3.6*, Geoscientific Model Development, 2019, Volume 7, Pages 3135-3148, DOI:10.5194/gmd-12-3135-2019, IF: 4,89 Q1

- **Oriol Tintó**, Mario Acosta, Miguel Castrillo, Stella V. Paronuzzi Ticco, Kim Serradell, Ana Cortés, Francisco J. Doblas-Reyes *Discriminating accurate results in nonlinear models*, 2019 International Conference on High Performance Computing and Simulation (HPCS), Dublin, 2019 , Computer Science Conference Rating: B

Other publications

- **Oriol Tintó Prims** , Miguel Castrillo, Kim Serradell , Oriol Mula-Valls, Ana Cortés Fité, Francisco-Javier Doblas Reyes, Jesús Labarta *Entendiendo el rendimiento aplicaciones cientificas de predicción climática.*, Jornadas de Paralelismo 2015, Córdoba

- **Oriol Tintó**, Miguel Castrillo, Kim Serradell, Oriol Mula-Valls, Francisco J. Doblas-Reyes, *Optimization of an ocean model using performance tools*, Barcelona Supercomputing Center - Earth Sciences Technical Memorandum, 2015

Presentations

- **Oriol Tintó**, Miguel Castrillo, Kim Serradell, Oriol Mula-Valls, Ana Cortés, Francisco J. Doblas-Reyes, *Understanding scientific application's performance using BSC tools*, International Conference on Computer Science, Iceland, 2015, Computer Science Conference Ranking: A

- Miguel Castrillo, **Oriol Tintó**, Harald Servat, G. S. Markomanolis, Kim Serradell *Applying clustering and folding techniques to study performance issues on the NEMO global ocean model*, HPC Knowladge Portal, Barcelona, 2015

- **Oriol Tintó Prims**, Mario Acosta, Miguel Castrillo, Ana Cortés, Alícia Sanchez, Kim Serradell and Francisco J. Doblas-Reyes *Optimizing domain decomposition in an ocean model: the case of NEMO*, International Conference on Computer Science, Switzerland, Year: 2017, Computer Science Conference Ranking: A

- **Oriol Tintó**, Miguel Castrillo,*Exploring the use of mixed precision in NEMO*, BSC Doctoral Symposium 2017, Barcelona, 2017

- **Oriol Tintó**, Miguel Castrillo, *Exploring the use of mixed precision in NEMO*, NEMO extended Developers Committee, Barcelona, 2017

- **Oriol Tintó**, Mario C. Acosta, Miguel Castrillo, Kim Serradell, Alícia Sanchez , Ana Cortés , Francisco J. Doblas-Reyes, *Towards an optimal use of numerical precision in Earth Science models: the case of NEMO*, Joint Laboratory on Exascale Computing, Barcelona, 2018

- **Oriol Tintó**, Mario C. Acosta, Miguel Castrillo, Kim Serradell, Alícia Sanchez , Ana Cortés , Francisco J. Doblas-Reyes, *Towards an optimal use of numerical precision in Earth Science models* , Oxford University, United Kingdom, 2018

- **Oriol Tintó**, Mario C. Acosta, Miguel Castrillo, Kim Serradell, Alícia Sanchez , Ana Cortés , Francisco J. Doblas-Reyes, *Towards an optimal use of numerical precision in Earth Science models*, European Centre for Medium-Range Weather Forecasts, United Kingdom, 2018

- **Oriol Tintó**, Mario C. Acosta, Miguel Castrillo, Kim Serradell, Ana Cortés , Francisco J. Doblas-Reyes *Adjusting numerical precision in NEMO*, ESCAPE2 VVUQ Workshop, France, 2018

Posters

- **Oriol Tintó**, Miguel Castrillo, Kim Serradell, Oriol Mula-Valls , Ana Cortés , Francisco J. Doblas-Reyes , *Understanding scientific application's performance*, BSC Doctoral Symposium 2015, Barcelona, 2015

- **Oriol Tintó**, Miguel Castrillo, Harald Servat, Germán Llort, Kim Serradell, Oriol Mula-Valls , Francisco J. Doblas-Reyes, *Optimization of an Ocean Model Using Performance Tools*, Supercomputing, United States, 2015

- **Oriol Tintó**, Miguel Castrillo, Kim Serradell, Oriol Mula-Valls , Ana Cortés , Francisco J. Doblas-Reyes, *Understanding scientific application's performance*, International HPC Summer School, Canada, 2015

- Authors: **Oriol Tintó**, Mario C. Acosta, Miguel Castrillo, Kim Serradell, Alícia Sanchez , Ana Cortés , Francisco J. Doblas-Reyes, *Towards an optimal use of numerical precision in Earth Science models: the case of NEMO*, European Geosciences Union General Assembly 2018 , Austria, 2018

International Secondment

I had the opportunity to perform a three month stay at the University of California - Santa Cruz, under the supervision of Dr. Andrew M. Moore. During this period we explored the possibility to use the methods and resources developed for NEMO in relation to the numerical precision required with a different ocean model, the Regional Ocean Modeling System. The results obtained showed that the Regional Ocean Modeling System can largely benefit from a reduction of the precision. These results have been part of a peer-reviewed publication that was written with the co-authorship of Dr. Andrew M. Moore, and that represent one of the chapters of this thesis.

Code contributions

The research conducted during this thesis led to code developments that have been included in NEMO and EC-Earth repositories.

References

[1] Peter Lynch. The origins of computer weather prediction and climate modeling. *Journal of Computational Physics*, 227(7):3431–3444, 2008.

[2] Peter Bauer, Alan Thorpe, and Gilbert Brunet. The quiet revolution of numerical weather prediction. *Nature*, 525(7567):47–55, 2015.

[3] Dudley B Chelton, Steven K Esbensen, Michael G Schlax, Nicolai Thum, Michael H Freilich, Frank J Wentz, Chelle L Gentemann, Michael J McPhaden, and Paul S Schopf. Observations of coupling between surface wind stress and sea surface temperature in the eastern tropical pacific. *Journal of Climate*, 14(7):1479–1498, 2001.

[4] Swen Jullien. Ocean response and feedback to tropical cyclones in the south pacific: processes and climatology. 11 2013.

[5] Woods Hole MA USA Raymond W. Schmitt — Emeritus, Woods Hole Oceanographic Institution. The ocean's role in climate. *Oceanography*, 31, June 2018.

[6] Céline Bellard, Cleo Bertelsmeier, Paul Leadley, Wilfried Thuiller, and Franck Courchamp. Impacts of climate change on the future of biodiversity. *Ecology Letters*, 15(4):365–377, apr 2012.

[7] Eric Chivian and Aaron Bernstein. *Sustaining life: how human health depends on biodiversity.* Oxford University Press, 2008.

[8] Fouad Ibrahim. Anthropogenic causes of desertification in western sudan. *GeoJournal*, 2(3), 1978.

[9] Naomi Oreskes. The scientific consensus on climate change. *Science*, 306(5702):1686–1686, 2004.

[10] Dominic R Kniveton, Christopher D Smith, and Richard Black. Emerging migration flows in a changing climate in dryland Africa. *Nature Climate Change*, 2(6):444–447, 2012.

[11] Gail Whiteman, Chris Hope, and Peter Wadhams. Climate science: Vast costs of arctic change. *Nature*, 499(7459):401, 2013.

[12] Jürgen Scheffran. Security risks of climate change: vulnerabilities, threats, conflicts and strategies. In *Coping with global environmental change, disasters and security*, pages 735–756. Springer, 2011.

[13] John M. Dennis and Richard D. Loft. *Refactoring Scientific Applications for Massive Parallelism*, pages 539–556. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[14] Robert H Dennard, Fritz H Gaensslen, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[15] A performance-based comparison of c/c++ compilers. `https://colfaxresearch.com/compiler-comparison/`. Accessed: 22-08-2019.

[16] Eric Winsberg. Computer simulations in science. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2019 edition, 2019.

[17] Tools@bsc.es. EXTRAE User Guide, 2015.

[18] Jesús Labarta, Judit Gimenez, E. Martínez, P. González, Harald Servat, Germán Llort, and Xavier Aguilar. Scalability of Tracing and Visualization Tools, 2005.

[19] Oriol Tintó Prims. Accelerating nemo: the road to exa-scale computing, 2014.

[20] Filip Váňa, Peter Düben, Simon Lang, Tim Palmer, Martin Leutbecher, Deborah Salmond, and Glenn Carver. Single Precision in Weather Forecasting Models: An Evaluation with the IFS. *Monthly Weather Review*, 145(2):495–502, 2017.

[21] Gurvan Madec, Pascale Delecluse, Maurice Imbard, and Claire Levy. OPA 8 . 1 Ocean General Circulation Model reference manual. *Notes du Pôle de Modélisation, Institut Pierre Simon Laplace*, (11):97, 1998.

[22] Gurvan Madec. *NEMO ocean engine.* Number 27 in 1288-1619. Note du Pôle de modélisation, Institut Pierre-Simon Laplace (IPSL), France, No 27, ISSN No 1288-1619, 2015.

[23] Stephen M. Griffies, Claus Böning, Frank O. Bryan, Eric P. Chassignet, Rüdiger Gerdes, Hiroyasu Hasumi, Anthony Hirst, Anne-Marie Treguier, and David Webb. Developments in ocean climate modelling. *Ocean Modelling*, 2(3):123 – 192, 2000.

[24] Alistair Adcroft and Jean-Michel Campin. Rescaled height coordinates for accurate representation of free-surface flows in ocean circulation models. *Ocean Modelling*, 7(3-4):269–284, 2004.

[25] A Arakawa and F Mesinger. Numerical methods used in atmospheric models. *Global Atmospheric Research Program*, 1:1–42, 1976.

[26] Bram Van Leer. Towards the ultimate conservative difference scheme. v. a second-order sequel to godunov's method. *Journal of computational Physics*, 32(1):101–136, 1979.

[27] Brian P Leonard. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Computer methods in applied mechanics and engineering*, 19(1):59–98, 1979.

[28] LIM team. LIM Website - www.climate.be/repomodx/lim, 2019.

[29] C. Rousset, M. Vancoppenolle, G. Madec, T. Fichefet, S. Flavoni, A. Barthélemy, R. Benshila, J. Chanut, C. Levy, S. Masson, and F. Vivier. The louvain-la-neuve sea ice model lim3.6: global and regional capabilities. *Geoscientific Model Development*, 8(10):2991–3005, 2015.

[30] Martin Vancoppenolle, Thierry Fichefet, Hugues Goosse, Sylvain Bouillon, Gurvan Madec, and Miguel Angel Morales Maqueda. Simulating the mass balance and salinity of Arctic and Antarctic sea ice. 1. Model description and validation. *Ocean Modelling*, 27(1-2):33–53, 2009.

[31] E. C. Hunke and J. K. Dukowicz. An elastic–viscous–plastic model for sea ice dynamics. *Journal of Physical Oceanography*, 27(9):1849–1867, 1997.

[32] Michael J Prather. Numerical advection by conservation of second-order moments. *Journal of Geophysical Research: Atmospheres*, 91(D6):6671–6681, 1986.

[33] O. Tintó Prims, M. Castrillo, M.C. Acosta, O. Mula-Valls, A. Sanchez Lorente, K. Serradell, A. Cortés, and F.J. Doblas-Reyes. Finding, analysing and solving MPI communication bottlenecks in Earth System models. *Journal of Computational Science*, 2018.

[34] Hao-Qiang Jin, Michael Frumkin, and Jerry Yan. The openmp implementation of nas parallel benchmarks and its performance. *NAS Technical Report NAS-99-011*, 1999.

[35] Tom Henderson, J Middlecoff, J Rosinski, M Govett, and P Madden. Experience applying fortran gpu compilers to numerical weather prediction. In *2011 Symposium on Application Accelerators in High-Performance Computing*, pages 34–41. IEEE, 2011.

[36] Oliver Fuhrer, Carlos Osuna, Xavier Lapillonne, Tobias Gysi, Ben Cumming, Mauro Bianco, Andrea Arteaga, and Thomas Christoph Schulthess. Towards a performance portable, architecture agnostic implementation strategy for weather and climate models. *Supercomputing frontiers and innovations*, 1(1):45–62, 2014.

[37] Burkhard Rockel, Andreas Will, and Andreas Hense. The regional climate model COSMO-CLM (CCLM). *Meteorologische Zeitschrift*, 17(4):347–348, 2008.

[38] Chloe Prodhomme, Lauriane Batté, François Massonnet, Paolo Davini, Omar Bellprat, Virginie Guemas, and Francisco J. Doblas-Reyes. Benefits of Increasing the Model Resolution for the Seasonal Forecast Quality in EC-Earth. *Journal of Climate*, 29(24):9141–9162, 12 2016.

[39] Tomas Wilhelmsson. *Parallelization of the HIROMB ocean model.* PhD thesis, Numerisk analys och datalogi, 2002.

[40] Jeffrey Lazo. Benefits of investing in weather forecasting research: An application to supercomputing. *Yuejiang Academic Journal*, 2:18–39, 2010.

[41] Ping Wang, Y. Tony Song, Yi Chao, and Hongchun Zhang. Parallel Computation of the Regional Ocean Modeling System. *International Journal of High Performance Computing Applications*, 19(4):375, 2005.

[42] Berk Hess, Carsten Kutzner, David Van Der Spoel, and Erik Lindahl. GRGMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *Journal of Chemical Theory and Computation*, 4(3):435–447, 2008.

[43] NAOMI H. NAIK, VIJAY K. NAIK, and MICHEL NICOULES. PARALLELIZATION OF A CLASS OF IMPLICIT FINITE DIFFERENCE SCHEMES IN COMPUTATIONAL FLUID DYNAMICS. *International Journal of High Speed Computing*, 05(01):1–50, 3 1993.

[44] P D Meyer, A J Valocchi, S F Ashby, and 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 Resources Res.*, 25(6):1440–1446, 1989.

[45] David Hutchison and John C Mitchell. Lecture Notes in Computer Science. 2012.

[46] P. Amestoy, I. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer Methods in Applied Mechanics and Engineering*, 184(2-4):501–520, 2000.

[47] Michele Benzi. Preconditioning Techniques for Large Linear Systems: A Survey. *Journal of Computational Physics*, 182(2):418–477, 2002.

[48] Mardochée Magolu monga Made and Henk A. van der Vorst. Paric: A family of parallel incomplete cholesky preconditioners. In Marian Bubak, Hamideh Afsarmanesh, Bob Hertzberger, and Roy Williams, editors, *High Performance Computing and Networking*, pages 89–98, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[49] Richard L Naff and John D Wilson. A comparison of preconditioning techniques for parallelized pcg solvers for the cell-centered finite-difference problem. In *XVI International Conference on Computational Methods in Water Resources*, pages 18–22, 2006.

[50] Keita Teranishi and Padma Raghavan. A Hybrid Parallel Preconditioner Using Incomplete Cholesky Factorization and Sparse Approximate Inversion. In *Domain Decomposition Methods in Science and Engineering XVI*, pages 755–762. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[51] V Eijkhout. Beware Of Unperturbed Modified Incomplete Factorizations. *Iterative Methods in Linear Algebra*, pages 583–591, 1991.

[52] Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John Kubiatowicz, Nelson Morgan, David Patterson, Koushik Sen, John Wawrzynek, David Wessel, and Katherine Yelick. A View of the Parallel Computing Landscape. *Commun. ACM*, 52(10):56–67, 2009.

[53] P. Hanappe, A. Beurivé, F. Laguzet, L. Steels, N. Bellouin, O. Boucher, Y. H. Yamazaki, T. Aina, and M. Allen. FAMOUS, faster: using parallel computing techniques to accelerate the FAMOUS/HadCM3 climate model with a focus on the radiative transfer algorithm. *Geoscientific Model Development Discussions*, 4:1273–1303, 2011.

[54] Joussaume S Lawrence B Mitchell J, Budich R and Marotzke J. Infrastructure Strategy European Earth System Modelling Community 2012-2022. *ENES Report Series 1*, page 33p, 2012.

[55] NEMO development team. NEMO Website - www.nemo-ocean.eu, 2019.

[56] Fiona JL Reid. Nemo on hector–a dcse project. *Report from the dCSE project, EPCC and University of Edinburgh, UK*, 2009.

[57] I. Epicoco, S. Mocavero, F. Macchia, M. Vichi, T. Lovato, S. Masina, and G. Aloisio. Performance and results of the high-resolution biogeochemical model PELAGOS025 within NEMO. *Geoscientific Model Development Discussions*, 8(12):10585–10625, 2015.

[58] Jay Fenlason and Richard Stallman. Gnu gprof. *GNU Binutils. Available online: http://www. gnu. org/software/binutils (accessed on 21 April 2018)*, 1988.

[59] Stephen M Griffies and Alistair J Adcroft. Formulating the equations of ocean models. *Ocean Modeling in an Eddying Regime*, 177:281–317, 2008.

[60] Alexander F. Shchepetkin and James C. McWilliams. The regional oceanic modeling system (ROMS): A split-explicit, free-surface, topography-following-coordinate oceanic model. *Ocean Modelling*, 9(4):347–404, 2005.

[61] J Crank and P Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Mathematical Proceedings of the Cambridge Philosophical Society*, 43(01):50–67, 1947.

[62] Reindert J Haarsma, Malcolm J Roberts, Pier Luigi Vidale, A. Catherine, Alessio Bellucci, Qing Bao, Ping Chang, Susanna Corti, Neven S. Fu??kar, Virginie Guemas, Jost Von Hardenberg, Wilco Hazeleger, Chihiro Kodama, Torben Koenigk, L. Ruby Leung, Jian Lu, Jing Jia Luo, Jiafu Mao, Matthew S. Mizielinski, Ryo Mizuta, Paulo Nobre, Masaki Satoh, Enrico Scoccimarro, Tido Semmler, Justin Small, and Jin Song Von Storch. High Resolution Model Intercomparison Project (HighResMIP v1.0) for CMIP6. *Geoscientific Model Development*, 9(11):4185–4208, 2016.

[63] Mario Acosta, Mancia Anguita, F. Javier Fernández-Baldomero, Cintia L. Ramón, S. Geoffrey Schladow, and Francisco J. Rueda. Evaluation of a nested-grid implementation for 3D finite-difference semi-implicit hydrodynamic models. *Environmental Modelling and Software*, 64:241–262, 2015.

[64] Takashi T. Sakamoto, Yoshiki Komuro, Teruyuki Nishimura, Masayoshi Ishii, Hiroaki Tatebe, Hideo Shiogama, Akira Hasegawa, Takahiro Toyoda, Masato Mori, Tatsuo Suzuki, Yukiko Imada, Toru Nozawa, Kumiko Takata, Takashi

Mochizuki, Koji Ogochi, Seita Emori, Hiroyasu Hasumi, and Masahide Kimoto. MIROC4hˆ—ˆmdash;A New High-Resolution Atmosphere-Ocean Coupled General Circulation Model. *Journal of the Meteorological Society of Japan*, 90(3):325–359, 2012.

[65] Thomas L. Delworth, Anthony Rosati, Whit Anderson, Alistair J. Adcroft, V. Balaji, Rusty Benson, Keith Dixon, Stephen M. Griffies, Hyun Chul Lee, Ronald C. Pacanowski, Gabriel A. Vecchi, Andrew T. Wittenberg, Fanrong Zeng, and Rong Zhang. Simulated climate and climate change in the GFDL CM2.5 high-resolution coupled climate model. *Journal of Climate*, 25(8):2755–2781, 2012.

[66] R. Justin Small, Enrique Curchitser, Katherine Hedstrom, Brian Kauffman, and William G. Large. The Benguela upwelling system: Quantifying the sensitivity to resolution and coastal wind representation in a global climate model. *Journal of Climate*, 28(23):9409–9432, 2015.

[67] Yukio Masumoto, H Sasaki, T Kagimoto, N Komori, a Ishida, Y Sasai, T Miyama, T Motoi, Humio Mitsudera, and K Takahashi. A fifty-year eddy-resolving simulation of the world ocean: Preliminary outcomes of OFES (OGCM for the Earth Simulator). *Journal of the Earth Simulator*, 1(April):35–56, 2004.

[68] Richard D Smith, Mathew E Maltrud, Frank O Bryan, and Matthew W Hecht. Numerical Simulation of the North Atlantic Ocean at 1 / 10 °. *Journal of Physical Oceanography*, 30:1532–1561, 2000.

[69] T. Rackow, H. F. Goessling, T. Jung, D. Sidorenko, T. Semmler, D. Barbi, and D. Handorf. Towards multi-resolution global climate modeling with ECHAM6-FESOM. Part II: climate variability. *Climate Dynamics*, 2016.

[70] Yuxia Zhang, Wieslaw Maslowski, and Albert J. Semtner. Impact of mesoscale ocean currents on sea ice in high-resolution Arctic ice and ocean simulations. *Journal of Geophysical Research*, 104(C8):18409, 1999.

[71] Peter R. Gent, Stephen G. Yeager, Richard B. Neale, Samuel Levis, and David A. Bailey. Improvements in a half degree atmosphere/land version of the CCSM. *Climate Dynamics*, 34(6):819–833, 2010.

[72] Carlos R. Mechoso, Chung-Chun Ma, John D. Farrara, Joseph A. Spahr, and Reagan Moore. Parallelization and Distribution of a Coupled Atmosphere–Ocean General Circulation Model, 1993.

[73] Venkatramani Balaji, Eric Maisonnave, Niki Zadeh, Bryan N. Lawrence, Joachim Biercamp, Uwe Fladrich, Giovanni Aloisio, Rusty Benson, Arnaud Caubel, Jeffrey Durachta, Marie-Alice Foujols, Grenville Lister, Silvia Mocavero, Seth Underwood, and Garrett Wright. CPMIP: measurements of real computational performance of Earth system models in CMIP6. *Geoscientific Model Development*, 10(1):19–34, 2017.

[74] John M Dennis, Mariana Vertenstein, and Anthony P Craig. Performance Evaluation of Ultra-High-Resolution Climate Simulations. In *10th LCI International Conference on High-Performance Clustered Computing*, 2009.

[75] Naomi Oreskes, David A Stainforth, and Leonard A Smith. Adaptation to Global Warming: Do Climate Models Tell Us What We Need to Know? *Philosophy of Science*, 77(5):1012–1028, dec 2010.

[76] Gabriele Hegerl and Francis Zwiers. Use of models in detection and attribution of climate change. *Wiley Interdisciplinary Reviews: Climate Change*, 2011.

[77] D.A. Randall, R.A. Wood, S. Bony, R. Colman, T. Fichefet, J. Fyfe, V. Kattsov, A. Pitman, J. Shukla, J. Srinivasan, R.J. Stouffer, A. Sumi, and K.E. Taylor. Climate Models and Their Evaluation. In *IPCC, 2007: Climate Change 2007: the physical science basis. contribution of Working Group I to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change*, volume 323, pages 589–662. Cambridge University Press, 2007.

[78] Marc Baboulin, Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Julie Langou, Julien Langou, Piotr Luszczek, and Stanimire Tomov. Accelerating scientific computations with mixed precision algorithms. *Computer Physics Communications*, 180(12):2526–2533, 12 2009.

[79] Azzam Haidar, Stanimire Tomov, Jack Dongarra, and Nicholas J. Higham. Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18 (Dallas, TX), pages 47:1–47:11, Piscataway, NJ, USA, 2018. IEEE Press.

[80] Stef Graillat, Fabienne Jézéquel, Romain Picot, François Févotte, and Bruno Lathuiliere. Promise: floating-point precision tuning with stochastic arithmetic. In *Proceedings of the 17th International Symposium on Scientific Computing, Computer Arithmetics and Verified Numerics (SCAN)*, pages 98–99, 2016.

[81] Michael O. Lam, Bronis R. De Supinksi, Matthew P. Legendre, and Jeffrey K. Hollingsworth. Automatically adapting programs for mixed-precision floating-point computation. *Proceedings - 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, SCC 2012*, pages 1423–1424, 2012.

[82] Peter D. Düben, Hugh McNamara, and T. N. Palmer. The use of imprecise processing to improve accuracy in weather & climate prediction. *Journal of Computational Physics*, 271:2–18, 2014.

[83] Peter D. Düben, Aneesh Subramanian, Andrew Dawson, and T. N. Palmer. A study of reduced numerical precision to make superparameterization more competitive using a hardware emulator in the OpenIFS model. *Journal of Advances in Modeling Earth Systems*, 9(1):566–584, 2017.

[84] Peter D. Düben, Aneesh Subramanian, Andrew Dawson, and T. N. Palmer. A study of reduced numerical precision to make superparameterization more competitive using a hardware emulator in the openifs model. *Journal of Advances in Modeling Earth Systems*, 9(1):566–584, 2017.

[85] Tobias Thornes. Can reducing precision improve accuracy in weather and climate models? *Weather*, 71(6):147–150, 2016.

[86] A. Dawson and P. D. Düben. rpe v5: an emulator for reduced floating-point precision in large numerical simulations. *Geoscientific Model Development*, 10(6):2221–2230, 2017.

[87] T.N. Palmer, G.J. Shutts, R. Hagedorn, F.J. Doblas-Reyes, T. Jung, and M. Leutbecher. Representing model uncertainty in wheather and climate prediction. *Annual Review of Earth and Planetary Sciences*, 33(1):163–193, may 2005.

[88] O. Aumont, C. Ethé, A. Tagliabue, L. Bopp, and M. Gehlen. PISCES-v2: An ocean biogeochemical model for carbon and ecosystem studies. *Geoscientific Model Development*, 2015.

[89] http://www.myroms.org. Roms webpage. `https://www.myroms.org`, 2018. [Online; accessed 18-October-2018].

[90] Andrew M. Moore, Hernan Arango, Gregoire Broquet, Brian S. Powell, Anthony Weaver, and Javier Zavala-Garay. The Regional Ocean Modeling System (ROMS) 4-dimensional variational data assimilation systems Part I - System overview and formulation. *Progress in Oceanography*, 91:34–49, 2011.

[91] Christian Collberg and Todd A. Proebsting. Repeatability in computer systems research. *Commun. ACM*, 59(3):62–69, February 2016.

[92] Peter Ivie and Douglas Thain. Reproducibility in scientific computing. *ACM Comput. Surv.*, 51(3):63:1–63:36, July 2018.

[93] R. Li, L. Liu, G. Yang, C. Zhang, and B. Wang. Bitwise identical compiling setup: prospective for reproducibility and reliability of earth system modeling. *Geoscientific Model Development*, 9(2):731–748, 2016.

[94] J. S. Targett, P. Düben, and W. Luk. Validating optimisations for chaotic simulations. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4, Sep. 2017.