Universitat Autònoma de Barcelona

Master Thesis

Accelerating NEMO: Towards Exascale Climate Simulation

A thesis submitted by Oriol Tintó Prims
in fulfilment of the requirements for the degree of
Master's in Mathematicall modelling for Science and Engineering

in the

Department of Computer Architecture and Operating Systems

developed at

Climate Forecasting Unit, Catalan Institute of Climate Sciences (IC3)

September 2014

Universitat Autònoma de Barcelona

Master Thesis

Accelerating NEMO: Towards Exascale Climate Simulation

Author:	*Advisor:*	*Co-advisor:*
Oriol Tintó Prims	Dr. Ana Cortés	Oriol Mula Valls

Author signature:	*Advisor signature:*	*Co-advisor signature:*

Abstract

The great potential and quick evolution of the High Performance Computing (HPC) encouraged the necessary research to adapt the current climate models so they will be able to be executed with efficiency in supercomputers that are being developed right now. For this work, tests about the current scalability of the NEMO oceanic model were performed and the the future problems that would emerge in the eventual future exa-scale executions are discussed. Some possible ideas to face these issues are introduced and , lastly, a new communication pattern to improve the current one is proposed and evaluated.

Resum

El gran potencial i la ràpida evolució de la Computació d'Altes Prestacions (HPC) han impulsat la recerca necessària per a adaptar els models climàtics existents amb la finalitat de poder-los executar amb eficiència en els super ordinadors que s'estan desenvolupant. Per a aquest treball s'han dut a terme mesures del rendiment per determinar quins són els problemes actuals d'escalabilitat del model climàtic oceànic NEMO i es discuteix quins problemes més poden emergir en un futur en què eventualment s'executi el model en milions de processadors. Per acabar, també es presenta una proposta de millora del patró de comunicacions.

Resumen

El gran potencial y la rápida evolución de la Computación de Altas Prestaciones (HPC) han impulsado la investigación necesaria para adaptar los modelos climáticos existentes con la finalidad de poder ejecutarlos con eficiéncia en los super computadores que se estan desarrollando. Para este trabajo se han llevado a cabo mediciones del rendimiento para determinar quales son los problemas actuales que limitan la escalabilidad del modelo climático oceánico NEMO y se discute qué problemas más pueden emerger en un futuro en que eventualmente se ejecute el modelo en millones de procesadores. Para terminar, también se presenta una propuesta de mejora del patrón de comunicaciones.

Acknowledgements

Thanks to everyone that helped me to complete this thesis. Especially Oriol Mula, Anna Cortés and Lola Fígols for their invaluable help.

Contents

List of Figures

Chapter 1

Introduction

The subject of the current thesis is the study of the scalability of the **NEMO**[1] model considering the future possibility of performing executions on computers that are capable to achieve one exaFLOP[2]. The computation with this capacity has been referred as **exascale** computing.

By "scalable" ,the general meaning is the ability to use additional computational resources effectively to solve increasingly larger problems. Many factors contribute to scalability, including the architecture of the parallel computer and the parallel implementation of the algorithm. However, one important issue is often overlooked: the scalability of the algorithm itself. Here, scalability is a description of how the total computational work requirements grow with problem size, which can be discussed independently of the computing platform.

The model in which the tests have been performed is the NEMO model, but some of the conclusions can be extrapolated to other models and HPC applications.

The **NEMO** model is an oceanic general circulation model (OGCM) developed in the European community [3].

A general circulation model (GCM), a type of climate model, is a mathematical model of the general circulation of a planetary atmosphere or ocean. It is based on the Navier–Stokes equations on a rotating sphere with thermodynamic terms for various energy sources (radiation, latent heat). These equations are the basis for complex computer programs commonly used for simulating the atmosphere or ocean of the Earth.

[1] Nucleus for European Modelling of the Ocean

[2] One exaFLOPS is a thousand petaFLOPS or a quintillion, 10^{18}, floating point operations per second.

[3] The main centers involved in the development are CNRS (France), Mercator-Ocean (France) , NERC (UK), UKMO (UK), CMCC, (Italy) and INGV(Italy).

Atmospheric and oceanic GCMs (AGCM and OGCM) are key components of global climate models along with sea ice and land-surface components. GCMs are widely applied for weather forecasting, understanding the climate, and projecting climate change.

Although it is possible to execute this model in HPC platforms with a large number of processors, the achieved speed-up with the current model and the efficiency are far to be good and it would be worth the effort to improve the performance of the model on HPC's. In this thesis, the current and future problems that need to be faced in order to reach good speed-up and efficiency are studied. For the 3.6 and older versions of NEMO the technical problems are limiting the model scalability more than the method-induced problems but when focusing in the future exa-scale executions, scalable numerical algorithms are also required.

Chapter 2

Brief introduction to NEMO

The Nucleus for European Modelling of the Ocean (NEMO) is a framework of ocean related engines, namely OPA[1] for the ocean dynamics and thermodynamics, LIM[2] for the sea-ice dynamics and thermodynamics, TOP[3] for the biogeochemistry both transport (TRP) and sources minus sinks (LOBSTER, PISCES)[4]. It is intended to be a flexible tool for studying the ocean and its interactions with the other components of the earth climate system (atmosphere, sea-ice, biogeochemical tracers, ...) over a wide range of space and time scales.

The ocean engine of NEMO is a primitive equation model adapted to regional and global ocean circulation problems. Prognostic variables are the three-dimensional velocity field, a linear or non-linear sea surface height, the temperature and the salinity. In the horizontal direction, the model uses a curvilinear orthogonal grid and in the vertical direction, a full or partial step z-coordinate, or s-coordinate, or a mixture of the two. The distribution of variables is a three-dimensional Arakawa C-type grid. Various physical choices are available to describe ocean physics, including TKE [5] , GLS [6] and KPP vertical physics [7] . Within NEMO, the ocean is interfaced with a sea-ice model (LIM v2 and v3), passive tracer and biogeochemical models (TOP) and, via the OASIS coupler, with several atmospheric general circulation models. It also support two-way grid embedding via the AGRIF software.

This model has been used for a wide range of applications, both regional or global, as a forced ocean model and as a model coupled with the atmosphere.

[1] OPA = Océan PArallélisé
[2] LIM= Louvain)la-neuve Ice Model
[3] TOP = Tracer in the Ocean Paradigm
[4] Both LOBSTER and PISCES are not acronyms just name
[5] Turbulent Kinetic Energy
[6] Generalized Least Squares
[7] K-Profile Parametrization

Here can be found a very brief summary of the basic equations solved by the model and the methodology used to discretise the problem to move from continuous equations to a discrete space where is posible to solve numerically.

This brief introduction to NEMO and the sections below are from the **reference documentation**[1] of the NEMO model. Here some of the basic elements of NEMO are explained.For a most extended knowladge , the reference documentation is recomended.

Following the same order as the one used in the reference documentation of NEMO , this summary is organised in as follows:

- Model Basics
- Domain Discretisation
- Boundary Conditions
- Physical Parametrisations

2.1 Model Basics

The model basics, *i.e.* the equations and their assumptions, the vertical coordinates used, and the subgrid scale physics. This part deals with the continuous equations of the model (primitive equations, with potential temperature, salinity and an equation of state). The equations are written in a curvilinear coordinate system, with a choice of vertical coordinates (z or s, with the rescaled height coordinate formulation z^*, or s^*). Momentum equations are formulated in the vector invariant form or in the flux form. Dimensional units in the meter, kilogram, second (MKS) international system are used throughout.

2.1.1 Primitive Equations

Doing some approximations, the vector invariant form of the primitive equations in the (**i**,**j**,**k**) vector system provides the following six equations (namely the momentum balance, the hydrostatic equilibrium, the incompressibility equation, the heat and salt conservation equations and an equation of state):

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

$$\frac{\partial p}{\partial z} = -\rho\, g \tag{2.1b}$$

$$\nabla \cdot \mathbf{U} = 0 \tag{2.1c}$$

$$\frac{\partial T}{\partial t} = -\nabla \cdot (T\ \mathbf{U}) + D^T + F^T \tag{2.1d}$$

$$\frac{\partial S}{\partial t} = -\nabla \cdot (S\ \mathbf{U}) + D^S + F^S \tag{2.1e}$$

$$\rho = \rho\,(T, S, p) \tag{2.1f}$$

where ∇ is the generalised derivative vector operator in $(\mathbf{i}, \mathbf{j}, \mathbf{k})$ directions, t is the time, z is the vertical coordinate, ρ is the *in situ* density , ρ_o is a reference density, p the pressure, $f = 2\mathbf{\Omega} \cdot \mathbf{k}$ is the Coriolis acceleration (where $\mathbf{\Omega}$ is the Earth's angular velocity vector), and g is the gravitational acceleration. $\mathbf{D}^{\mathbf{U}}$, D^T and D^S are the parameterisations of small-scale physics for momentum, temperature and salinity, and $\mathbf{F}^{\mathbf{U}}$, F^T and F^S surface forcing terms.

2.1.2 The Horizontal Pressure Gradient

The total pressure at a given depth z is composed of a surface pressure p_s at a reference geopotential surface ($z = 0$) and a hydrostatic pressure p_h such that: $p(i, j, k, t) = p_s(i, j, t) + p_h(i, j, k, t)$. The latter is computed by integrating (2.1b), assuming that pressure in decibars can be approximated by depth in meters in (2.1f). The hydrostatic pressure is then given by:

$$p_h\,(i, j, z, t) = \int_{\varsigma=z}^{\varsigma=0} g\ \rho\,(T, S, \varsigma)\ d\varsigma \tag{2.2}$$

2.2 Domain Discretisation

Once we know the basic equations that we need to solve, the next step is to deal with the discrete equations.

- We talk about the time domain. The model time stepping environment is a three level scheme in which the tendency terms of the equations are evaluated either centered in time, or forward, or backward depending of the nature of the term.
- The other domain that we need to talk about is the space domain. The model is discretised on a staggered grid (Arakawa C grid) with masking of land areas. Vertical discretisation used depends on both how the bottom topography is represented and whether the free surface is linear or not. Full step or partial step

z-coordinate or s- (terrain-following) coordinate is used with linear free surface (level position are then fixed in time). In non-linear free surface, the corresponding rescaled height coordinate formulation (z^* or s^*) is used (the level position then vary in time as a function of the sea surface heigh).

- Two other things that the model need to discretise are the prognostic equations for the active tracers and the momentum. Explicit, split-explicit and filtered free surface formulations are implemented. A number of numerical schemes are available for momentum advection, for the computation of the pressure gradients, as well as for the advection of tracers (second or higher order advection schemes, including positive ones).

2.3 Domain Decomposition

Domain decomposition methods solve a boundary value problem by splitting it into smaller boundary value problems on subdomains and iterating to coordinate the solution between adjacent subdomains.

When we have a discrete domain defined, we can determine how to distribute the domain in order to be able to parallelise the model.

2.4 Boundary Conditions

- We also need to describe the Surface boundary conditions. Those can be implemented as prescribed fluxes, or bulk formulations for the surface fluxes (wind stress, heat, freshwater). The model allows penetration of solar radiation There is an optional geothermal heating at the ocean bottom. Within the NEMO system the ocean model is interactively coupled with a sea ice model (LIM) and with biogeochemistry models (PISCES, LOBSTER). Interactive coupling to Atmospheric models is possible via the OASIS coupler. Two-way nesting is also available through an interface to the AGRIF package (Adaptative Grid Refinement in FORTRAN) . The interface code for coupling to an alternative sea ice model (CICE) has now been upgraded so that it works for both global and regional domains, although AGRIF is still not available.

- Other model characteristics are the lateral boundary conditions (chapter LBC). Global configurations of the model make use of the ORCA tripolar grid, with special north fold boundary condition. Free-slip or no-slip boundary conditions are

allowed at land boundaries. Closed basin geometries as well as periodic domains and open boundary conditions are possible.

2.5 Physical Parametrisations

- Finally , the physical parameterisations are described. The model includes an implicit treatment of vertical viscosity and diffusivity. The lateral Laplacian and biharmonic viscosity and diffusion can be rotated following a geopotential or neutral direction. There is an optional eddy induced velocity with a space and time variable coefficient. The model has vertical harmonic viscosity and diffusion with a space and time variable coefficient

2.6 Namelists

The namelist allows to input variables (character, logical, real and integer) into the code. There is one namelist file for each component of NEMO (dynamics, sea-ice, biogeochemistry...) containing all the FOTRAN namelists needed. The implementation in NEMO uses a two step process. For each FORTRAN namelist, two files are read:

1. A reference namelist (in *CONFIG/SHARED/namelist_ref*) is read first. This file contains all the namelist variables which are initialised to default values
2. A configuration namelist (in *CONFIG/CFG_NAME/EXP00/namelist_cfg*) is read aferwards. This file contains only the namelist variables which are changed from default values, and overwrites those.

A template can be found in *NEMO/OPA_SRC/module.example* The effective namelist, taken in account during the run, is stored at execution time in an output_namelist_dyn (or _ice or _top) file.

Model outputs management and specific online diagnostics are described in chapters DIA. The diagnostics includes the output of all the tendencies of the momentum and tracers equations, the output of tracers tendencies averaged over the time evolving mixed layer, the output of the tendencies of the barotropic vorticity equation, the computation of on-line floats trajectories... Chapter OBS describes a tool which reads in observation

files (profile temperature and salinity, sea surface temperature, sea level anomaly and sea ice concentration) and calculates an interpolated model equivalent value at the observation location and nearest model timestep. Originally developed of data assimilation, it is a fantastic tool for model and data comparison. Chapter ASM describes how increments produced by data assimilation may be applied to the model equations. Finally, Chapter CFG provides a brief introduction to the pre-defined model configurations (water column model, ORCA and GYRE families of configurations).

The model is implemented in FORTRAN 90, with preprocessing (C-pre-processor). It runs under UNIX. It is optimized for vector computers and parallelised by domain decomposition with MPI. All input and output is done in NetCDF (Network Common Data Format) with a optional direct access format for output. To ensure the clarity and readability of the code it is necessary to follow coding rules. The coding rules for OPA include conventions for naming variables, with different starting letters for different types of variables (real, integer, parameter...). Those rules are briefly presented in Appendix Apdx D and a more complete document is available on the NEMO web site.

The model is organized with a high internal modularity based on physics. For example, each trend (*i.e.*, a term in the Right Hand Side of the prognostic equation) for momentum and tracers is computed in a dedicated module. To make it easier for the user to find his way around the code, the module names follow a three-letter rule. For example, **traldf** is a module related to the TRAcers equation, computing the Lateral DiFfussion. Furthermore, modules are organized in a few directories that correspond to their category, as indicated by the first three letters of their name (Tab. Tab chap).

The manual mirrors the organization of the model. After the presentation of the continuous equations (Chapter PE), the following chapters refer to specific terms of the equations each associated with a group of modules (Tab. Tab chap).

TABLE 2.1: Organization of Chapters mimicking the one of the model directories.

Chapter STP	-	model time STePping environment
Chapter DOM	DOM	model DOMain
Chapter TRA	TRA	TRAcer equations (potential temperature and salinity)
Chapter DYN	DYN	DYNamic equations (momentum)
Chapter SBC	SBC	Surface Boundary Conditions
Chapter LBC	LBC	Lateral Boundary Conditions (also OBC and BDY)
Chapter LDF	LDF	Lateral DiFfusion (parameterisations)
Chapter ZDF	ZDF	vertical (Z) DiFfusion (parameterisations)
Chapter DIA	DIA	I/O and DIAgnostics (also IOM, FLO and TRD)
Chapter OBS	OBS	OBServation and model comparison
Chapter ASM	ASM	ASsiMilation increment
Chapter MISC	SOL	Miscellaneous topics (including solvers)
Chapter CFG	-	predefined configurations (including C1D)

Chapter 3

How does NEMO Work?

In this chapter it will be explained how some parts of the NEMO model are implemented so that it can be better understood what is causing the problems on the scalability of NEMO.

3.1 Domain Definition

As NEMO is a global model, the domain must cover all the earth surface. What is necessary to be defined is the grid configuration. The NEMO model can work with different grid configurations.

ORCA is the generic name given to a set of global ocean configurations. Its specificity lies on the horizontal curvilinear mesh used to overcome the North Pole singularity found for geographical meshes.

The ORCA reference grid is ORCA2 and its resolution approximately 2 degrees but it differs depending on the geographical zone of the earth. The figures 3.1, 3.2 and 3.3 show how is the tripolar mesh of ORCA2. The dimensions of the mesh ORCA 2 are 182 x 149 for a total of 27118 points.

Apart from the ORCA2 there are also other ORCA configurations with different resolutions, but these configurations are not included in the NEMO reference configurations. The grid sizes of some of the available ORCA configurations are listed on the table 3.1.

The amount of points determines the computational cost, the need for scale the model and also the efficiency that the scaling can reach. Higher resolutions implies more computational cost, requires more resources and more processors to be able to perform

name	npi	npj	points
ORCA2	182	149	27118
ORCA1	362	292	105704
ORCA05	722	511	368942
ORCA025	1440	1021	1470240

TABLE 3.1: Number of rows, columns and total number of horizontal grid points for the different resolutions of the ORCA grid.

simulations in a acceptable time. Since there are more computation for each communication and the overlapping is smaller,the efficiency can be better.

There are also other configurations apart from the ORCA configuration. The other kind of configuration that are also included in the reference configurations are the GYRE configurations.

Their characteristics are listed below:

1. Idealized configuration representing double gyres in the North hemisphere,
2. Beta-plane with a regular grid spacing at 1 degree horizontal resolution,
3. 31 vertical levels,
4. Forced with analytical heat, freshwater and wind-stress fields

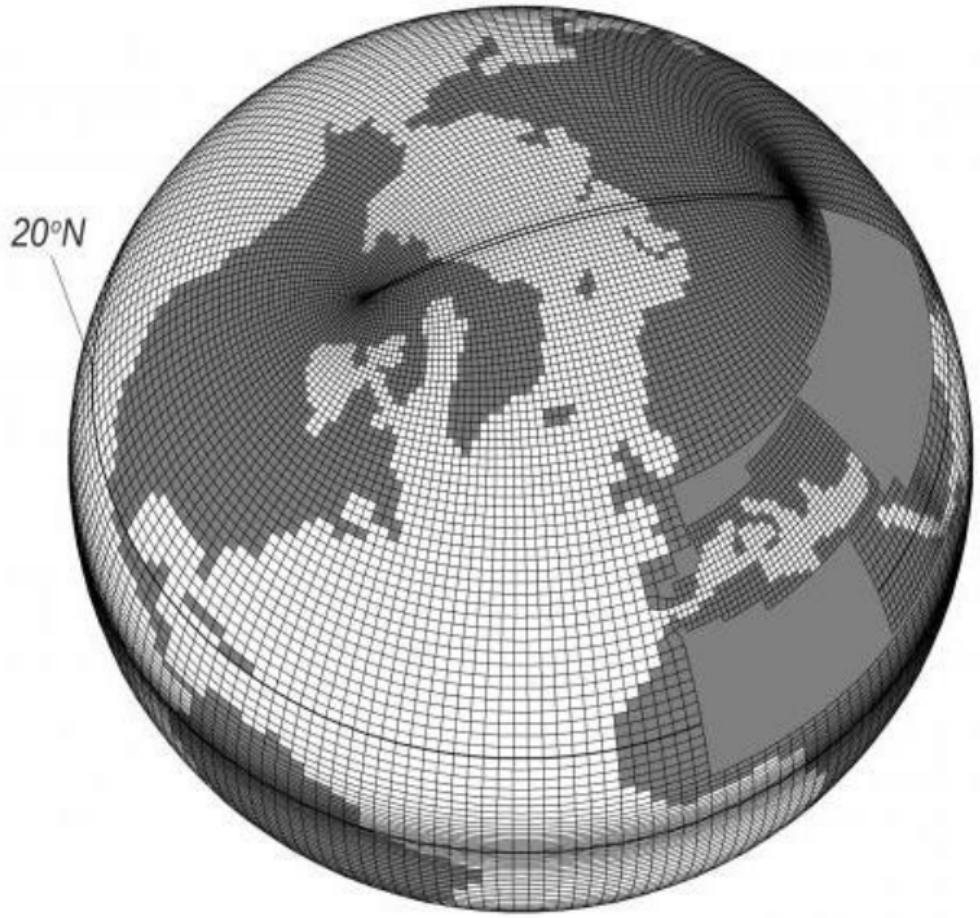

FIGURE 3.1: ORCA2 grid representation.

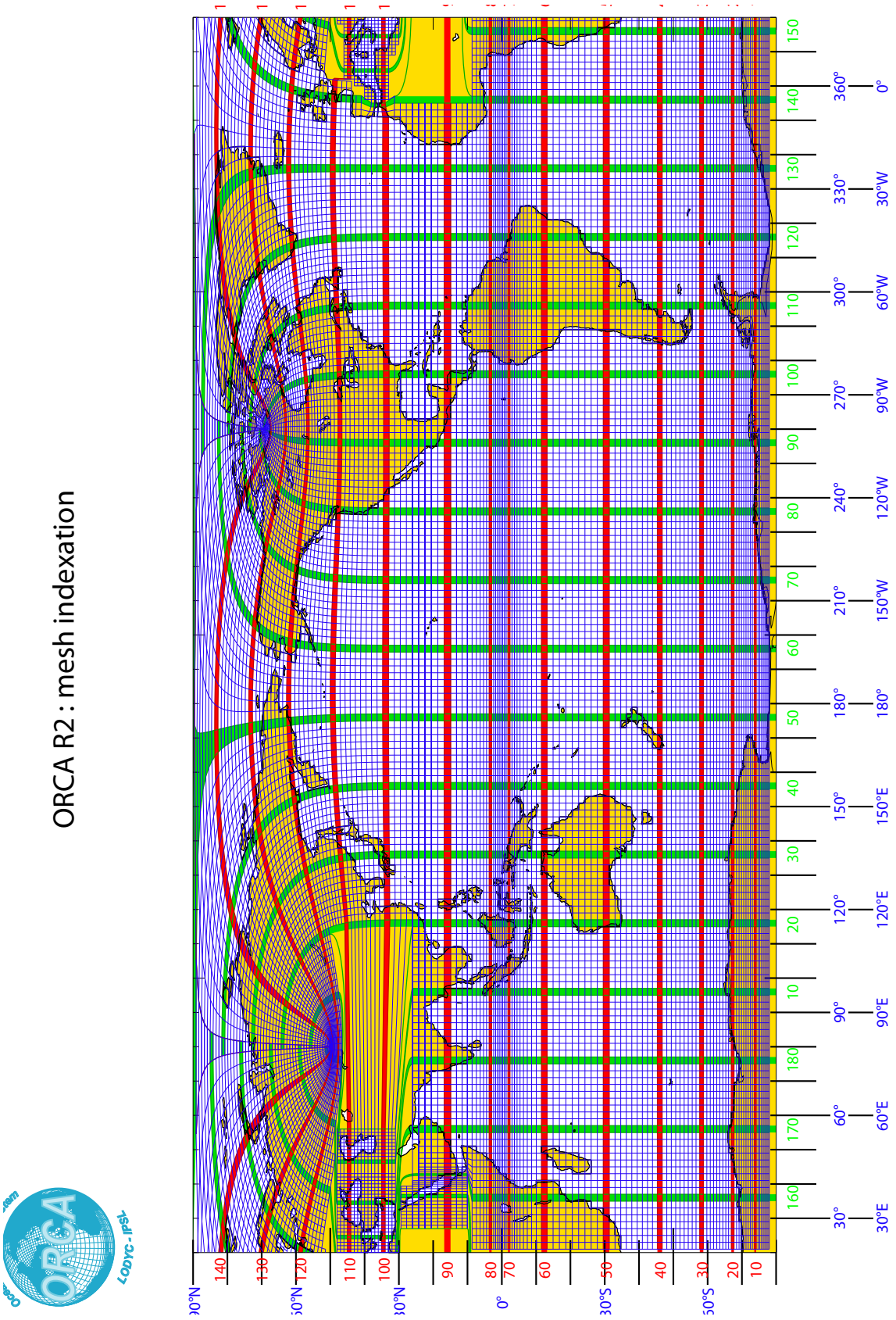

FIGURE 3.2: ORCA2 grid indexation.

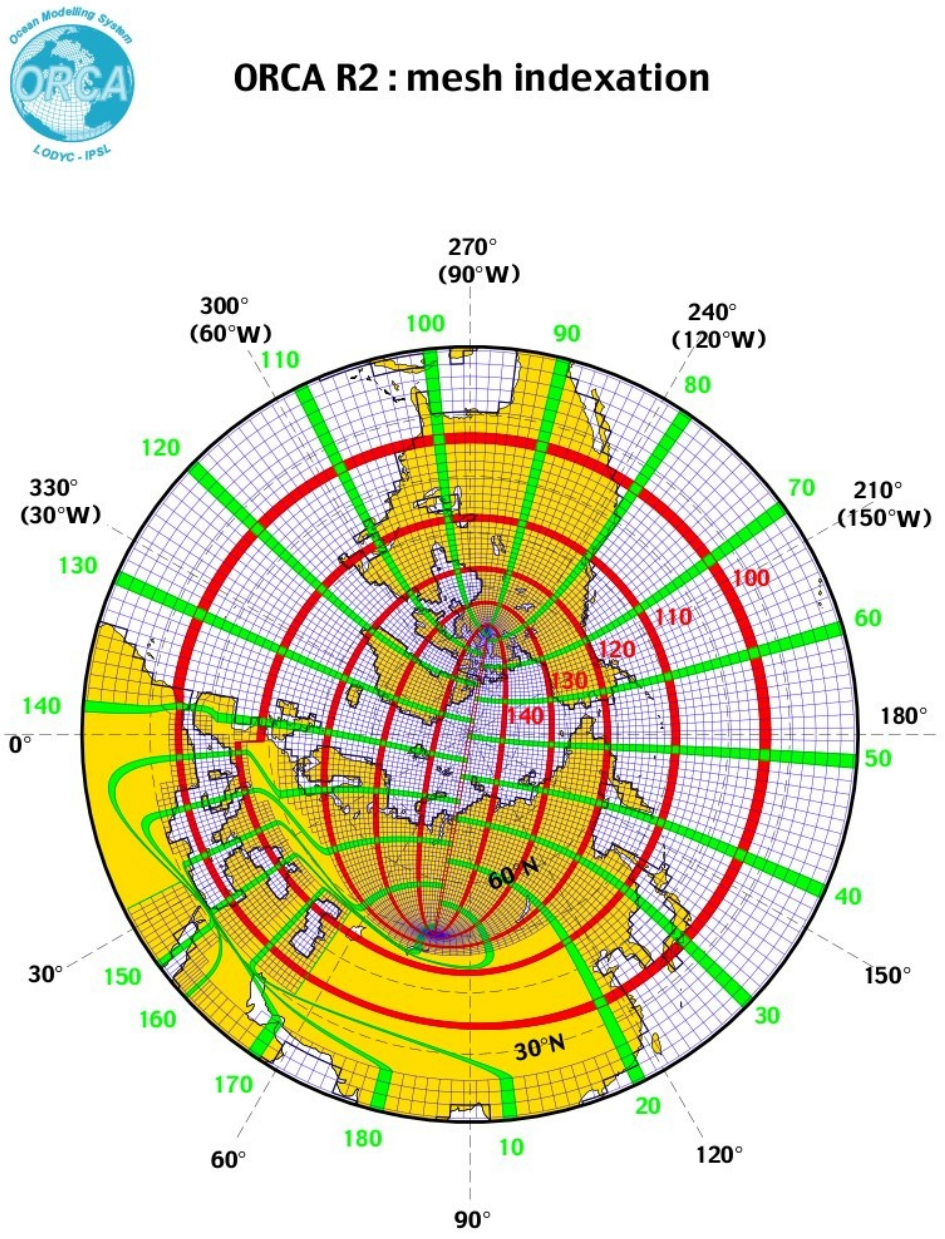

FIGURE 3.3: North pole view of ORCA2 grid indexation.

3.2 Domain Decomposition

To parallelize the computation of the discrete equations, the program uses a domain decomposition method to split it into different sub-domains. Each processor has its own local memory and computes the model equation over a sub-domain of the whole model domain. The domain decomposition methods solve a boundary value problem by splitting it into smaller boundary value problems on sub-domains and iterating to coordinate the solution between adjacent sub-domains. The sub-domain boundary conditions are specified through communications between processors which are organized by explicit statements (message passing method).

The parallelization strategy is defined by the physical characteristics of the ocean model. Second order finite difference schemes lead to local discrete operators that depend at the very most on one neighbouring point. That means it is necessary an overlapping between the sub-domains to solve the equations.

Each sub-domain computes its own surface and bottom boundary conditions and has a side wall overlapping interface which defines the lateral boundary conditions for computations in the inner sub-domain. The overlapping area consists of the two rows at each edge of the sub-domain. After a computation, a communication phase starts: each processor sends to its neighbouring processors the update values of the points corresponding to the interior overlapping area to its neighbouring sub-domain (i.e. the innermost of the two overlapping rows).

Due to the overlapping , the shape factor becomes important. The current decomposition is made in rectangular sub-domains and when the the sub-domains are squares, the overlapping becomes minimum (Figure 3.4).

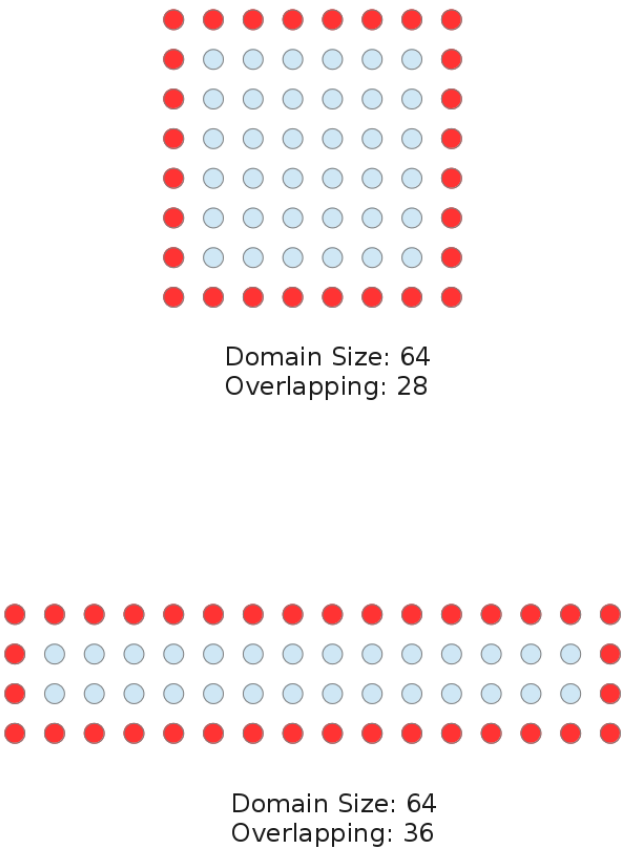

FIGURE 3.4: Comparison of the overlapping size with two different shapes.

In the current code, it is not necessary to specify the number of sub-domains in each dimension since it is decided automatically to choose the best decomposition (the decomposition with the best shape-factor, i.e. lowest overlapping).

3.3 Communications

All the different modules in the model use the same communication routine.

The first routine used to parallelise the NEMO dates back to 1994, and the same routine with several improvements is still used in the current code.

While the firsts versions of the code worked with blocking communications in both send and receive subroutines, the current communication routine allows the user to select the kind of communications that is to be used to send messages (Send, Isend, Bsend) but it is still using blocking communications to receive the messages.

The communication routine is coded using the following pattern:

1. Send a message to the west and east neighbour
2. Wait for a message from the west neighbour.
3. Receive the message from the west.
4. Wait for a message from the east neighbour.
5. Receive the message from the east.
6. Wait until all sent messages are received by the neighbours (east - west).
7. Send a message to the north and south neighbour
8. Wait for a message from the north neighbour.
9. Receive the message from the north.
10. Wait for a message from the south neighbour.
11. Receive the message from the south.
12. Wait until all sent messages are received by the neighbours (north - south)

In chapter 4 the problems that emerge from this communication paradigm are discussed and in chapter 5 it is analysed if its possible to improve the current pattern with some easy-to-apply changes. Several improvements are proposed and tested and the results show that it would be interesting to apply it into the NEMO code.

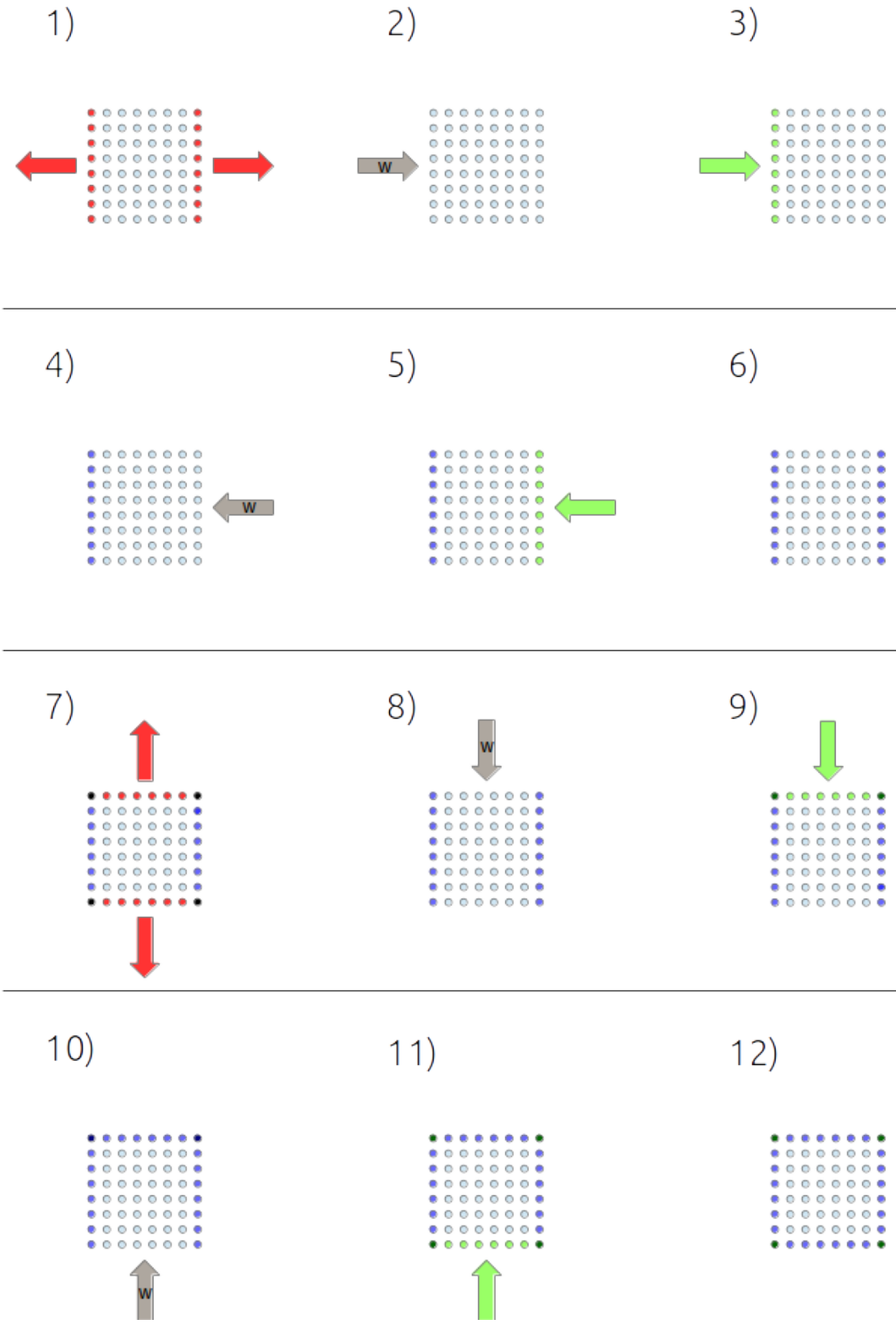

FIGURE 3.5: Current NEMO communication pattern

Chapter 4

Nemo Profiling

Sometimes when thinking about scalability problems, the ideas that emerge are only technical related issues and usually a very important one is overlooked: the scalability of the algorithms.

Although in the current implementation of NEMO those elements are not the main problem (there are also technical problems with more weight), they could also be the bottleneck for the scalability in the future and it is important to have all this elements in mind.

Usually both technical and methodological problems are mixed, and one example are the **communications**.

The method used by the model requires very frequent communications between processes. The communication itself is not necessarily a problem if the network system is fast enough, but there is a problem with the need of synchronisation.

The need for synchronisation forces the processes to wait each other every time that the model need to communicate.

Even without parallel computing, the execution time for a specific routine is not always the same since there are multiple factors that can affect the performance of the computation. The use of shared resources can cause a further increase of those differences. And even if the computation time of each process was always the same for all of them, the need for synchrony produces that the use of the shared resources is not distributed in time, *i.e.* all the processes use some specific resource at the same time. It causes that the hardware resources needed to achieve a good performance become much more than they could be if the use of the resources was not so simultaneous.

In this chapter some of the problems found on the current NEMO are discussed and some ideas to improve these problems are proposed[1] In the chapter 5 a deeper analysis of the technical solutions proposed improve communication is introduced.

4.1 Method Limitations

The main limitations now existing, caused by the methods applied to solve the differential equations, are the overlapping of the sub-domains due to the discretisation of the domain and the use of synchronous communication.

When the problems that we want to solve come from the methods chosen, what should be done instead of trying to modify the current code is to think in **alternative methods.**

To decide when it is worth to change the current algorithms for another one, the weakest and the strongest points of each one must be evaluated , given that computational needs and communication needs could be opposed. As an example of this, in the section 4.1.2 the loss of efficiency due to using the extra-halo strategy for the solver is evaluated.

4.1.1 Overlapping

The overlapping caused by the domain discretisation among different processes implies a loss of efficiency. The impact of this factor on the scalability depends on the configuration resolution. With a specific number of processors, the bigger the domain is, the smaller the impact of the overlapping , since the proportion between the overlapped points with the non-overlapped points becomes smaller .

The problem of the loss of efficiency due to the overlapping becomes relevant when the performance of simulations of low resolution with a very low wall-clock time is required. For higher resolutions, the impact of the overlapping it is still not so relevant.

Here is an approximate effect of the overlapping in the theoretical limits of efficiency and speed up.

Assuming the shape of the sub-domains are close to a square, the sub-domain size could be modelled as :

$$SD_S = \frac{G_S}{n} + 4 \cdot \sqrt{\frac{G_S}{n}} \tag{4.1}$$

Where SD_S is the Sub Domain Size, G_S is the original grid size and n the number of sub-domains.

[1] The proposed ideas to improve the program's performance have not been tested on the NEMO code.

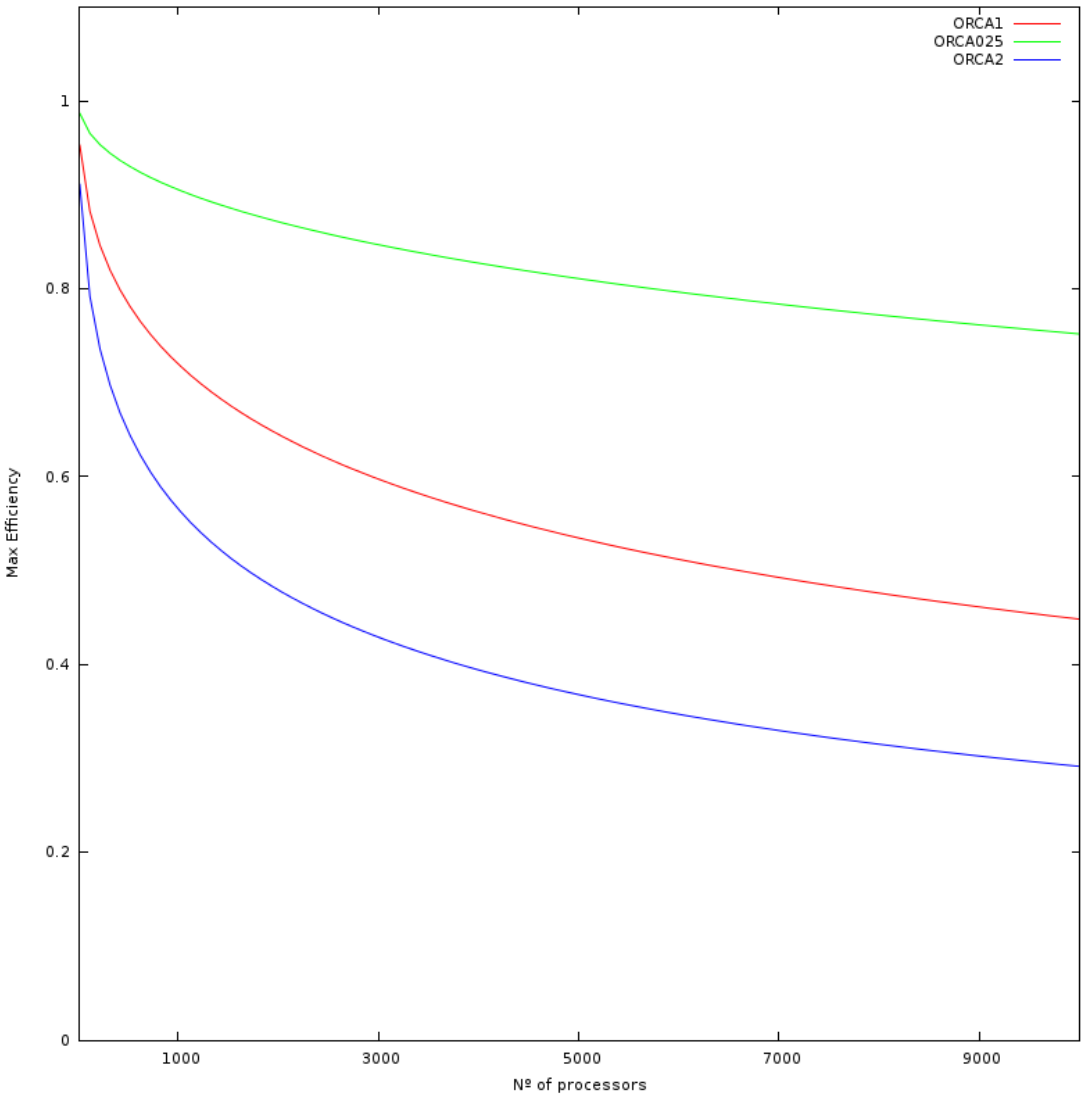

FIGURE 4.1: The effect of the overlapping on the limit of efficiency

The figures 4.1 and 4.2 are graphical representations derived from of the equation 4.1 for three different resolutions. (See the table 3.1)

The figure 4.1 shows the efficiency constraining due to the overlapping when the number of processes grows and the figure 4.2 shows the maximum speed-up achievable.

The only way to solve the overlapping problem is changing the current algorithm to solve the model equations for another method that does not need overlapping of the sub-domains. There are some recent papers talking about these possible methods, so it would be of great interest to take them into account. [4]

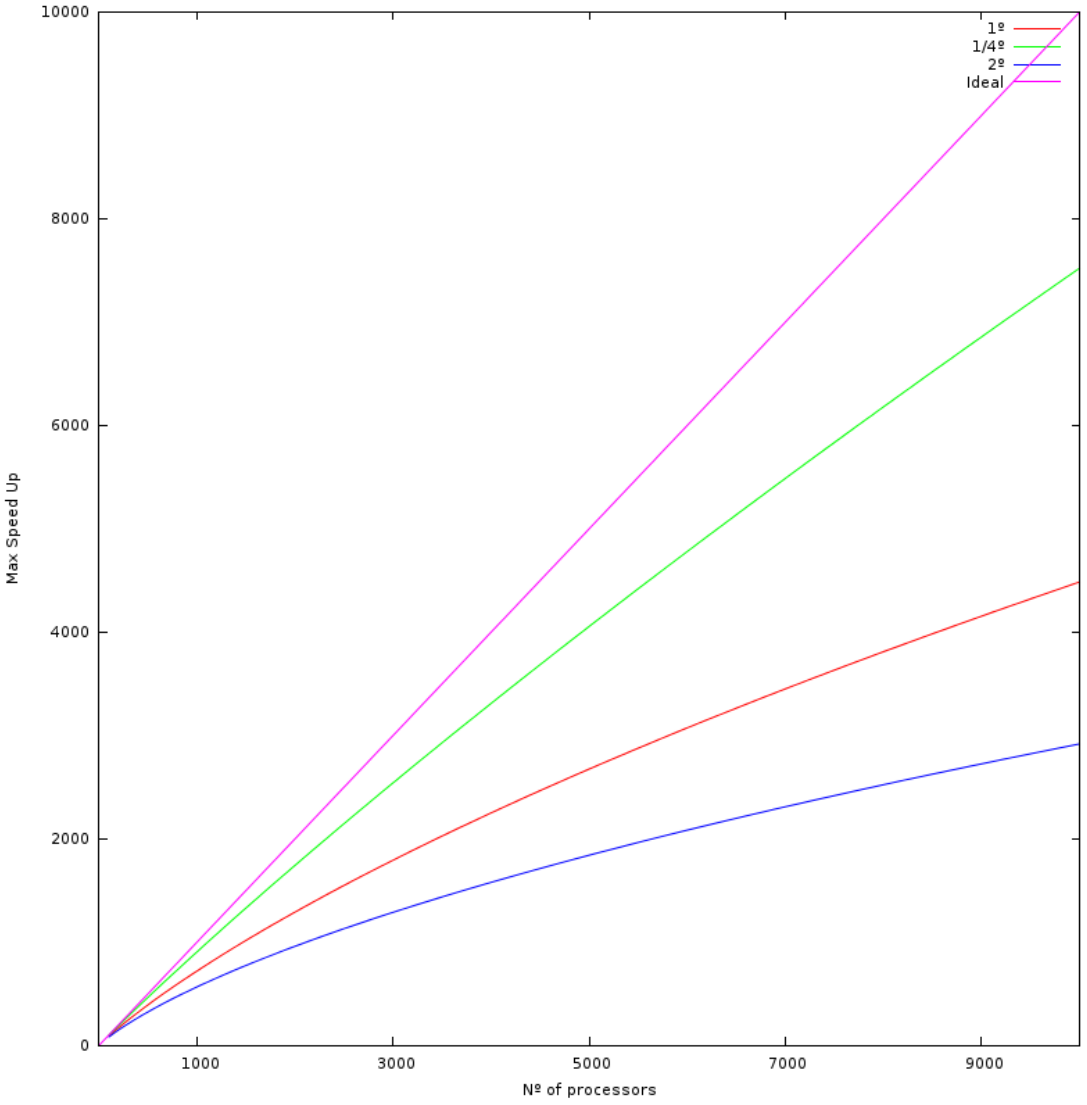

FIGURE 4.2: The effect of the overlapping on the limit of the speed-up

4.1.2 Communication and Synchronization

As said on Chapter 3.3, the routine that performs the communications dates from 1994. Some improvements have been done since the original code was written but the core of the routine still remains. When the routine was conceived the objective was to be able to run the model with few processors instead of a single one, but the possibility to run the model into several thousands or millions of processors was not on the horizon.

The question of what would happen if they tried to run this model using thousands or millions of processors was never made and the problems that would emerge from it were not studied when the routine was developed.

Here follow some proposals of the answers to the expounded questions: There is a direct consequence of the method used and the need of synchronisation. The most important aspect that come from the communication system is the **need of synchrony**. The need of synchrony produces that all the processes wait for each other in every single step.

Since the time spent in computation in each processor is not exactly defined (it varies from one process to another due to some run conditions), the time spent in every step between communications for all the processes will be equal to the slowest process.

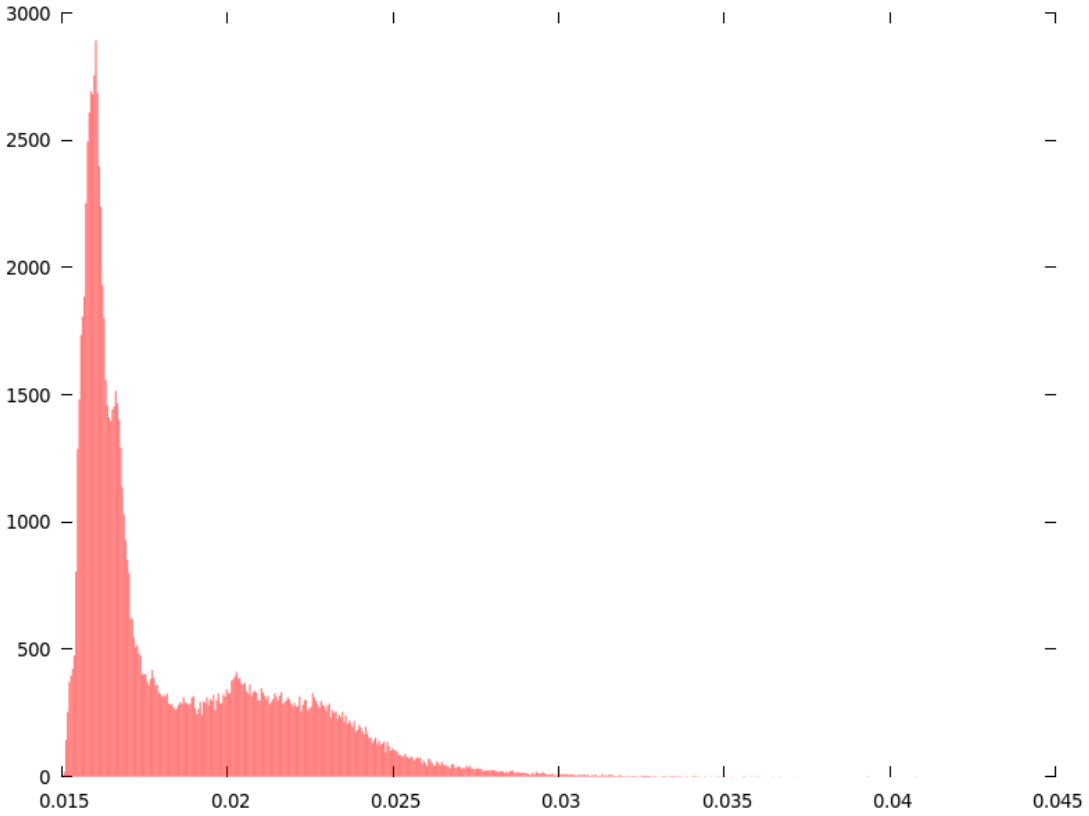

FIGURE 4.3: Example of the time distribution of the different executions of the same routine.The x axis show the time in seconds.

Even without technical issues like saturation of the shared resources, this phenomena will decrease the performance.

The impact that the number of processes has in this effect will be explained straightaway with two examples. This will illustrate the effect of increasing the number of processes when synchrony is needed.

In the first example, we have a dice, and we are interested in the expected value of the maximum when we throw the dice n times.

The expected value of the maximum when we throw the dice only once is the same as the expected value of the dice value. But when we throw the dice more times, the expected value of the maximum increase.

We can calculate the expected value by computing the value of the maximum for all the possible permutations.

$$\langle max_n \rangle = \sum_{i=1}^{c^n} \frac{max(X_{i,1}...X_{i,n})}{c^n} \tag{4.2}$$

where c is the number of possible values of the dice and n the times that we throw the dice.

With a classic dice with 6 faces, the evolution of the expected value of the maximum when increasing the number of throws is the following:

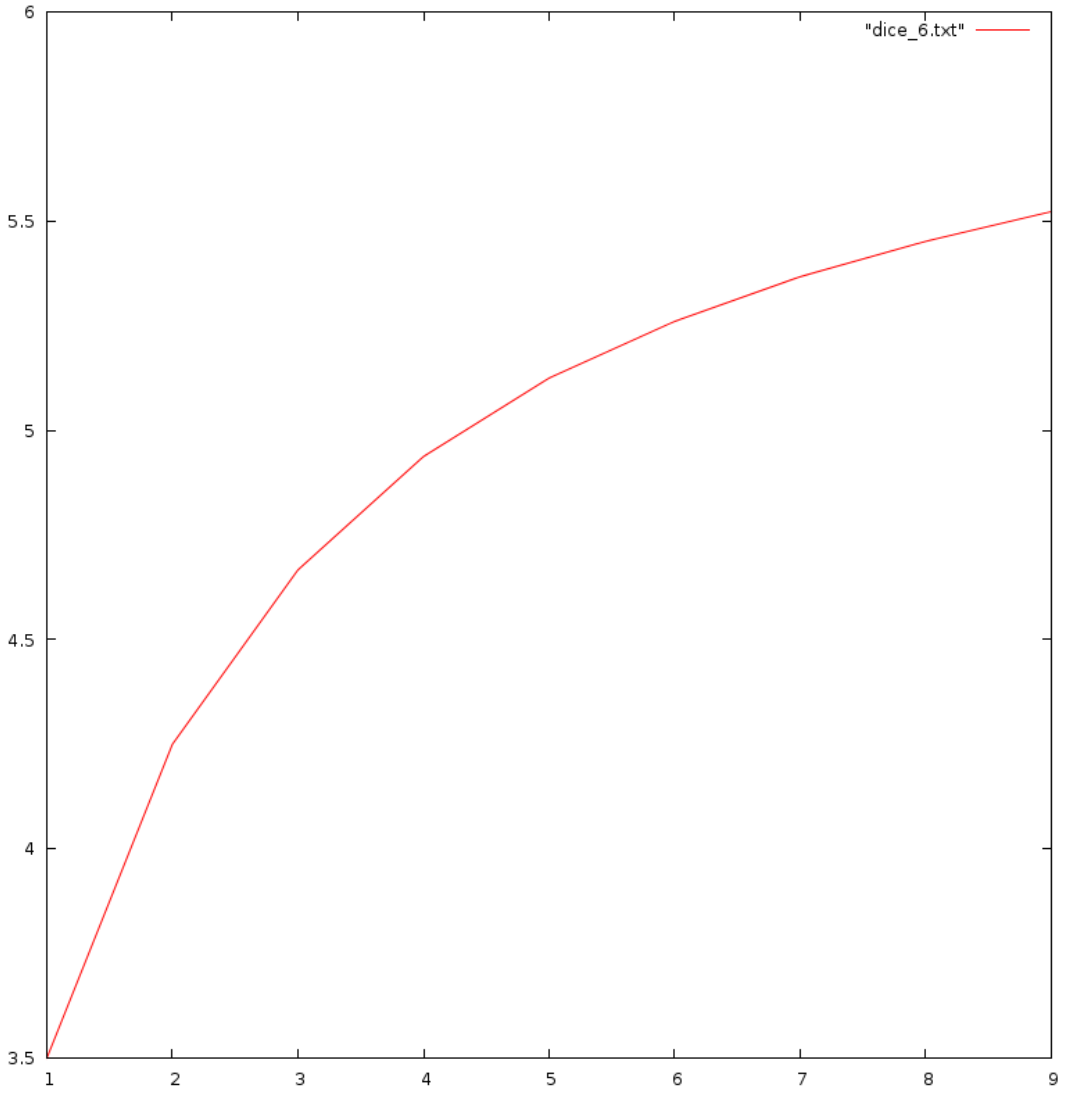

FIGURE 4.4: Expected value for the maximum corresponding to n throws of a dice. Those values are calculated analytically from the formula 4.2

In the second example, consider random numbers corresponding to a normal distribution instead of a dice. The evolution here is similar:

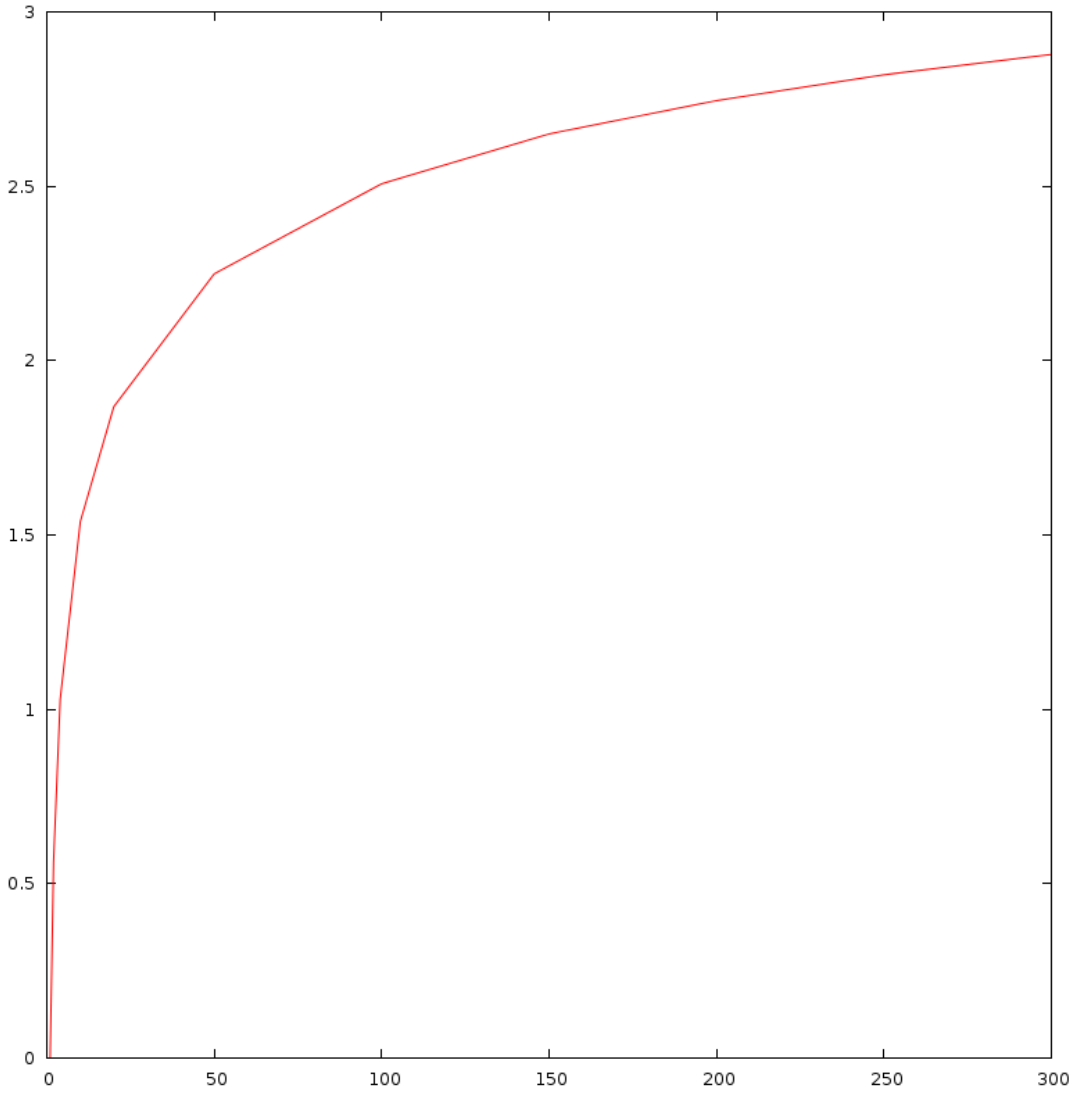

FIGURE 4.5: Expected value for the maximum of n random numbers that follow a normal distribution.Those values correspond to computer simulations. With a confidence interval of 99.9% the confidence interval error is less than 0.02 .

How are these two examples related to the effects of synchronisation is explained below.

If the computation time for a specific routine follows a certain distribution, and there is only one process or if synchrony is not needed, when we execute this routine n times, the total time will be close to

$$time = n \cdot \mu \tag{4.3}$$

where μ is the mean value of the computation time for that routine.

But if we include here the need of synchronisation, the behaviour will be similar to the previous examples and the total time of the execution would evolve like the expected

value of the maximum when you have a random number a certain number of times. We could make an analogy between the maximum value of a set of dice throws with the maximum value of the execution times for different processors.

The real distribution is neither a normal distribution and of course nor the one of a dice. Then the behaviour of the real system will be different from those two examples but still quite similar.

A real case in which there is a need of synchronisation in every small step will be more complex, since the time distribution of the processes will not be a normal distribution. This effect could be increased considering that the increment of the needs of communication can also increase the standard deviation of the time spent on communication.

It could seem that this effect is not so relevant as a logarithmic grow may be acceptable, but the explanation on this section only shows the evolution of the **lower limit** of the wall-clock time in an ideal case. In the real computer model, the time distribution of some of the routines is highly dependent on the run conditions and the use of shared resources, and this produces an even bigger increase of the wall-clock time.

In the NEMO code, one of the hypothesis is that the current communication pattern leads to cascade events that could slow down some of the processes and because the effect explained in this section increase the global wall-clock time.

Another problem that emerge of synchronisation is that all the processes are using the same specific shared resource approximately at the same time.

To solve these problems that emerge from communication and the need of synchrony, first it must be analysed if the synchrony is necessary or if other methods that allow asynchronous communications can be adopted. There are some algorithms currently used that need synchronous communications. The most important algorithm used that require synchronization is the **solver**, as the algorithm iterates until the overlapped points converge on all the processes.

If a change from the current method to an asynchronous one was not possible, the clearer approach to reduce the synchronisation effects would be to reduce the amount of communications.

When reducing the number of communications implies an increment of the computation cost, it would be necessary to evaluate if this increase is worthy in order to improve the scalability of the code.

There are many possibilities to decrease the number of communications. In the current NEMO code it is possible to use an **extra-halo** method that allow the solver to decrease

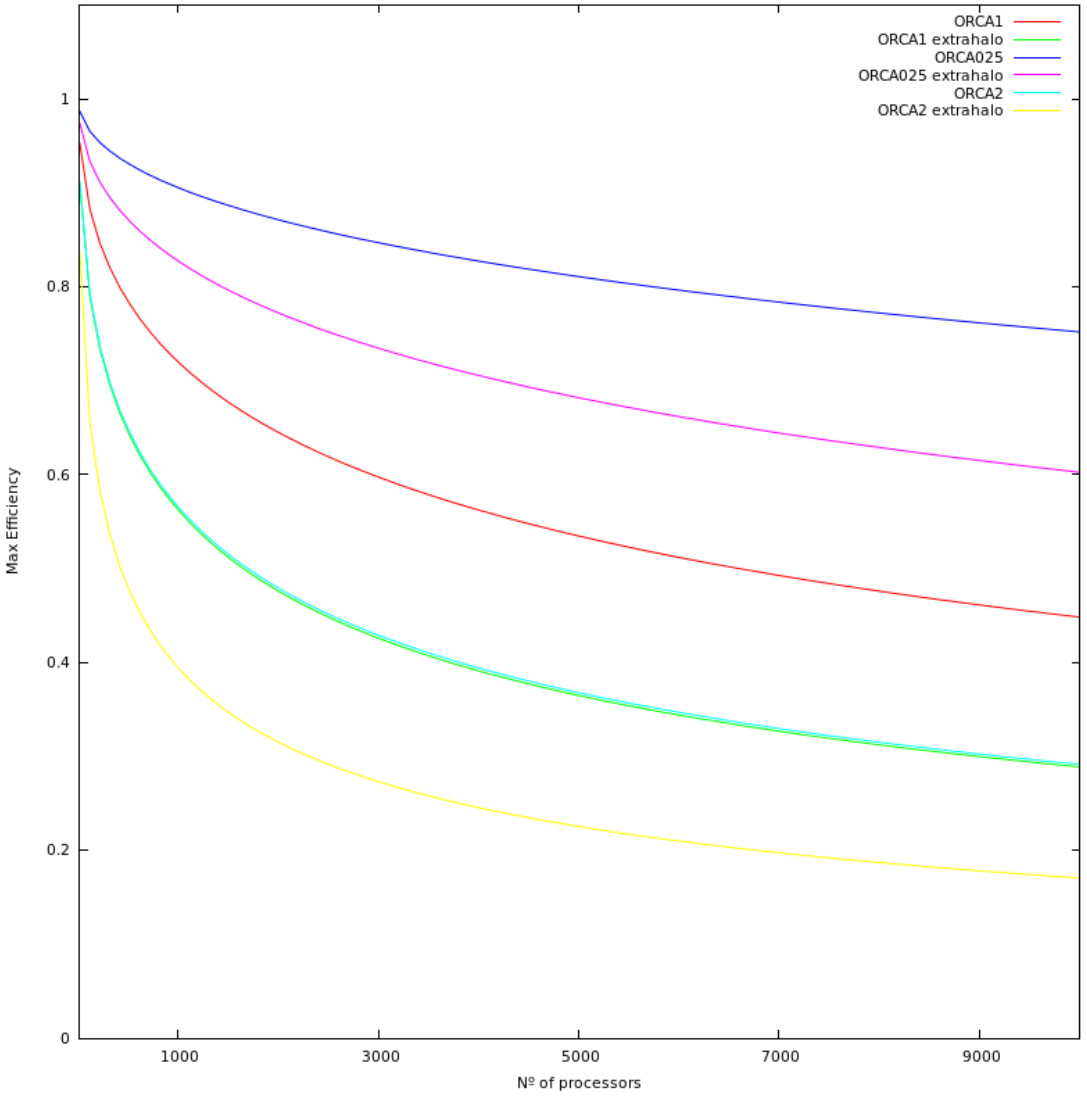

FIGURE 4.6: Comparison of limit of efficiency for the classical overlapping and extra-halo.

the number of communications between the processors by adding an extra-row of shared points. The effect of this extra-halo on the maximum speed-up and efficiency can be easily modelled.

As we can see on the figures 4.1 and 4.2 the maximum efficiency and speed-up are even more constrained because of overlapping.

This kind of approximation can improve the performance of the model when it is constrained by the communications and need of synchrony, but if the goal is the possibility to perform exa-scale executions, discussions about if the loss of efficiency is worth must be done. Also it is important to discuss if the model can keep working with synchronous

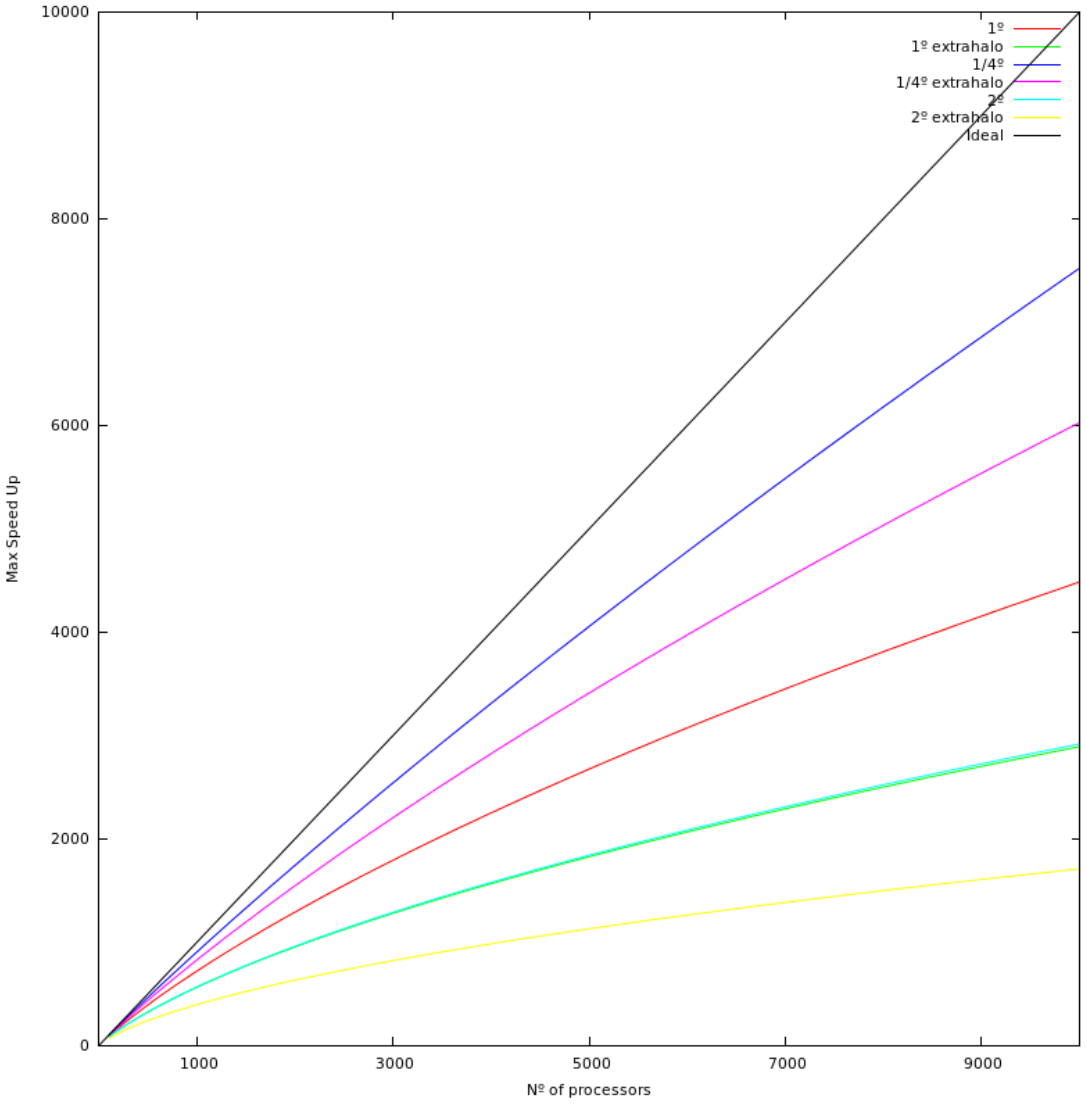

FIGURE 4.7: Comparison of limit of the speed-up for the classical overlapping and extra-halo

communications or if it is better to make the effort to find alternatives to allow asynchronous communications.

There are also other possibilities to decrease the number of communications. The **Split-Explicit free surface** algorithm [1] for the solver have shown that the communication requirements decrease (and in consequence the synchrony requirements) in comparison to the **filtered free surface** algorithm[1].

4.2 Technical Limitations

The main technical limitations in the scalability of the NEMO model are saturation of the shared resources (Essentially the network bandwidth and file system access) and the work unbalance.

As we said before, it is important to know that due to the synchronisation issue, is not necessary to have a global increase on the time spent in one particular routine to have a global growth in the global time. *i.e.* the delay of one process is enough to delay the whole model.

As the results of the profiling performed on NEMO show, when increasing the number of processors, there are two types of routines that increase their times much more than desirable. The routines related with I/O and the routines with a big amount of communications.

What is important part about the increment of time spent in communications is that almost all the time is spent in waiting, not in transferring data, and this shows that the problem may not be caused by the limits of the available bandwidth, but by the problems related to synchrony and the differences of time spent performing calculations in different processors.

Since the synchrony problems came from the method used, to improve the scalability of the model we must focus on the origin of differences between the execution times on different processors.

These differences proceed from two main origins:

- **Work Unbalance**
- **Running Conditions**

and as said before, the reason that these differences are really a problem is:

- **Communication and Synchronization**

These problems must be solved separately.

4.2.1 Work Unbalance

When we talk about **work unbalance** we talk about the differences on the amount of computation that the different processes have to do.

In the NEMO model most of this work unbalance comes from the nature of the problem to solve, the LIM[2] module and its differences on the work load in different regions of the earth. For the Sea-Ice , most of the work is placed near the earth poles, while the tropical zones do not have any work to perform.

To achieve a good efficiency it is necessary to reduce the work unbalance and avoid the waiting times, otherwise the resources will not be taken full profit of.

To solve this specific problem into NEMO we have two proposals, none of them tested.

The first approach is to uncouple the sea-ice module from the NEMO code and couple it later using OASIS. It would allow to run specific processors to run the sea-ice model, and the number of this processors could be adjusted to avoid the work unbalance.

The second possible improvement is modify the geometry of the sub-domains to create smaller sub-domains where the sea-ice module has more work to do and bigger sub-domains with no sea-ice work.

This second approach is not possible with the current synchronous communications due to the impossibility to find concurrency between different modules. *i.e.* it is not possible to compensate bigger workloads on one module in another module because then we will find waiting time due to work unbalance in both modules that will make the performance worse.

4.2.2 Running Conditions

The use of shared resources could cause that the same task performed by two different processors or in two different moments takes different times to be completed.

When talking here about shared resources it is referred to both the platforms shared with other users that could interfere with our job, and to the shared resources with the other processes (memory access, communication network,...).

In the current implementation of NEMO, several differences were found in the times spent by the processors in some of the routines, caused by run conditions. These routines relate to both file access and communication.

[2]The LIM module is the sea-ice module of NEMO

4.2.2.1 Input/Output

The problems about I/O were the main bottleneck in the scalability of NEMO until the new approach of a dedicated processes for writing/reading files was implemented. The system XIOS[3][3] implemented in the lasts versions of NEMO allows the model to write outputs at any frequency without compromising the performance.

In this point (July 2014) the system does not allow a parallel approach to access files and this is still a bottleneck in some routines. The massive access to shared file system delays some processes and the synchronisation produces a global slowdown.

The server approach (*i.e.* to have processors dedicated to perform the input/output operations) has proved to be a good option at generating outputs. It is also expected to be a good solution to solve the problems with the file access to read files.

4.2.2.2 Communication and Synchronization

The main problem about the communications is that when simulations with a large number of processors are performed , the time spent in computations becomes smaller and smaller but the time spent in communications grows , and then this time spent in communications becomes an important proportion of the total time of the simulation. This reduces the efficiency, since communication time is not a valuable work. In the current NEMO version, the time spent in communication plus the time spent in waiting is bigger than the time spent on useful computation for simulations with more than 144 processors.

As explained before, when the number of processors is increased , the waiting times produced by the need of the synchronisation also increase. Since this is not related with the interchange of data itself , it can not be expected to decrease the time involved in communication and synchronisation.

To minimize the synchronisation issues, the goal must be to maintain the time spent in every step between communications in each process as equal as possible in all of them. To do it , it is necessary to take care of the use of shared resources.

One of these shared resources is the network system. The use of synchronous communication causes that the communication occurs in short pulses, and this makes that the bandwidth becomes more loaded than desirable.

As an example of this, in one sample execution of the NEMO model running on 1152 processors, it transfers about $7.5 \cdot 10^{10}$ bytes in a 5-simulated-days run in about 255

[3]XML Input Output Server tool developed on the IPSL institute in France

seconds. This give us a rate of $2.94 \cdot 10^8$ $\frac{bytes}{second}$, much smaller than the bandwidth capacity of our network system, that is approximately $4 \cdot 10^{10}$ $\frac{bytes}{second}$. But what really happens is that all communications are executed in synchrony, in a very narrow time-space , and that causes that the network throughput becomes much bigger , which could cause the model to saturate the total bandwidth of the system.

In this section it will only be discussed the possible improvements on the current communications, considering that it can not be changed to an asynchronous paradigm .

The current implementation of the communication system could be causing unnecessary cascades of events, accentuating even further the fact that the communication occurs in short pulses.

This, later on, could be also causing important differences in the time spent by different processes to complete every communication step and therefore incrementing the synchronisation problems.

Technically what can be done to improve communication is to minimize these cascade events and make the communication process as distributed as possible.

In order to achieve this, three different improvements to the current pattern have been suggested and a test-application has been developed to compare the performance of the current pattern with the alternatives proposed to see if the suggested changes to the communication pattern could improve the model performance.

As proved in chapter 5 , the current communication's paradigm implemented on NEMO is far from being optimal and an improvement could be interesting. In this chapter, a study of different communication patterns is introduced and evaluated.

The interesting conclusions of the communication's study are the following:

1. The current communication pattern is far from being optimal.
2. It would be easy to implement a better solution on the NEMO code.

In the chapter 5 we have explained and tested the improvements proposed and the results support the those affirmations.

Chapter 5

Proposal of Communication Improvement

In this chapter the tests performed with a test-program created to emulate the NEMO communicate requirements will be explained in order to be able to evaluate if the current communication pattern can be improved.

In addition to the current pattern, three alternative communication patterns have been suggested and have been tested it. To perform these experiments with a different numbers of processes theMare Nostrum 3 superomputer was used. Its characteristics are available at the appendix A.

5.1 Current Pattern

In the current version of NEMO, when the model calls the mpp-lib routine to interchange data with their neighbours in any routine. The pattern used is explained on the section 3.3. (Figure 3.5)

Since the receiving functions are blocking, the processes cannot receive the message from West before the message from East. The same happens when the processes communicate in the north-south direction.

The processes call this function when they finish some computation. It send messages to East and West, waits for messages from East, and when the message is received it waits for the message from the West. When this point is reached , almost every process is waiting until the last process has finished . When the last process finishes the computation , it triggers a cascade of events from all the processes that were waiting. It also provokes that all the communications of the second round (north south) start at the same time. Therefore all the processes send their messages to the north-south communication in a very short period of time and this could be producing a few moments of high traffic that could be avoided. This algorithm is far from being optimal. Here some ideas to improve it will be proposed.

One element that can be relevant is that in the current pattern there is a transmission of the corner values to the diagonal neighbours. In some routines this could be important but in the solver it is not important since all the points corresponding to overlapped areas must converge to the same values. For this reason, it should be studied if the changes applied to the communication routines could be applied to all the routines or not. In the second case, it would be still interesting to apply these changes to the solver as it is the most communication consuming routine.

5.2 Possible Improvements to the Communication Pattern

We propose three possible improvements to the current rountine. The goal of these improvements is to minimize the time spent on communication without changing how the model developers use the routine.

5.2.1 Couple E-W with N-S communication

This possible improvement just consists in coupling the E-W and N-S communication. It would help to avoid cascade events and therefore avoid a load excessively the network system.

1. Every process sends a message to the all its neighbours.
2. Waits for a message from the west neighbour.
3. Receives the message from the west.
4. Waits for a message from the east neighbour.
5. Receives the message from the east.
6. Waits for a message from the north neighbour.
7. Receives the message from the north.
8. Waits for a message from the south neighbour.
9. Receives the message from the south.
10. Waits until all the messages have been successfully delivered.

This change in the algorithm avoids the intermediate sending that could be causing problems.

5.2.2 First In First Out

Another possible improvement could be trying to allow the program to process the received messages in the same order as they arrive. By processing the already received messages, even more cascades of events than in the previous algorithm are avoided.

This algorithm also sends the messages to all the neighbours at the beginning of the routine.

1. Every process sends a message to the all their neighbours.
2. Polls all the neighbours to see if there is a message.
3. Receives the first message to get in.
4. Polls the remaining three neighbours for messages.

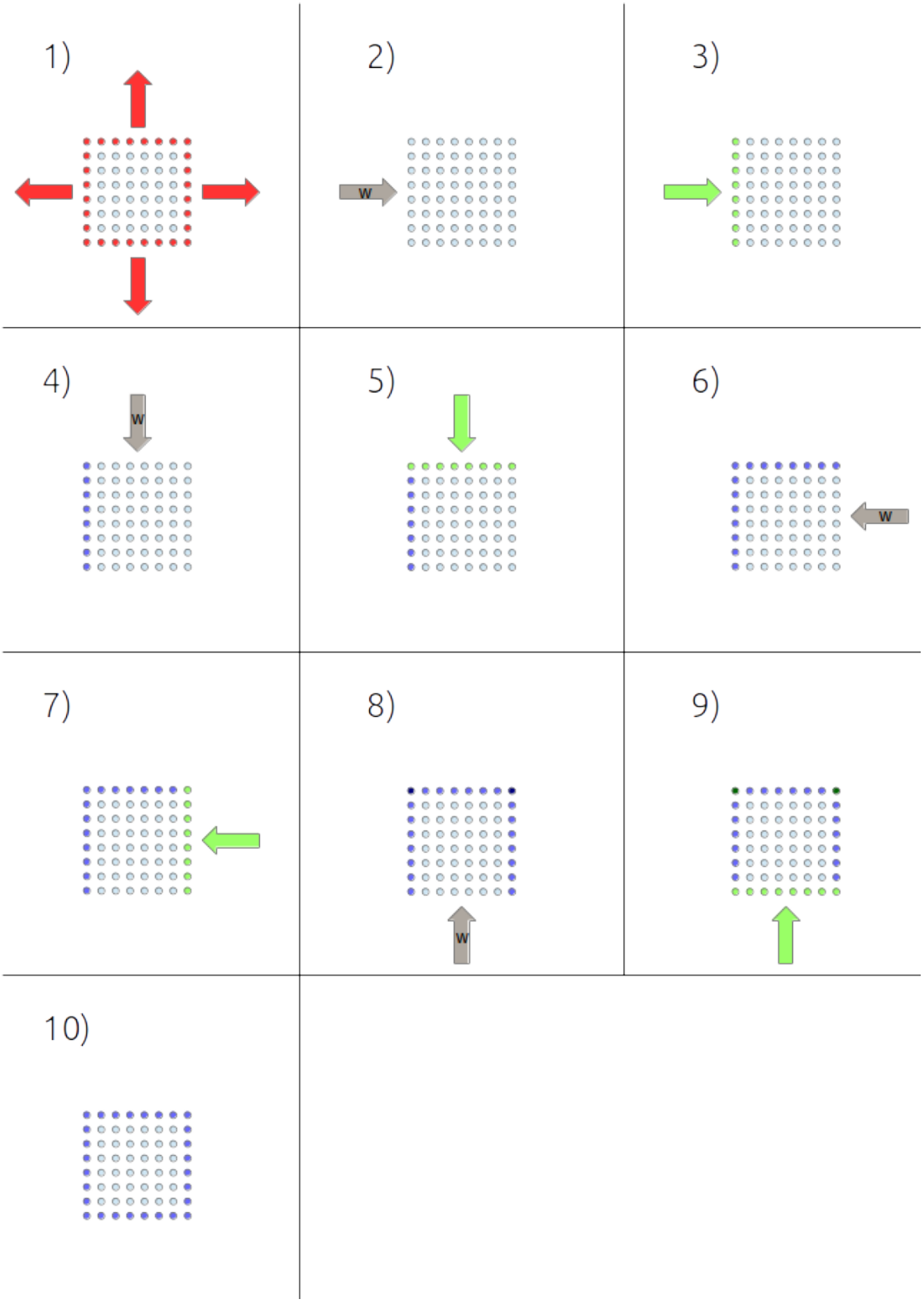

FIGURE 5.1: Coupled E-W with N-S communication pattern.

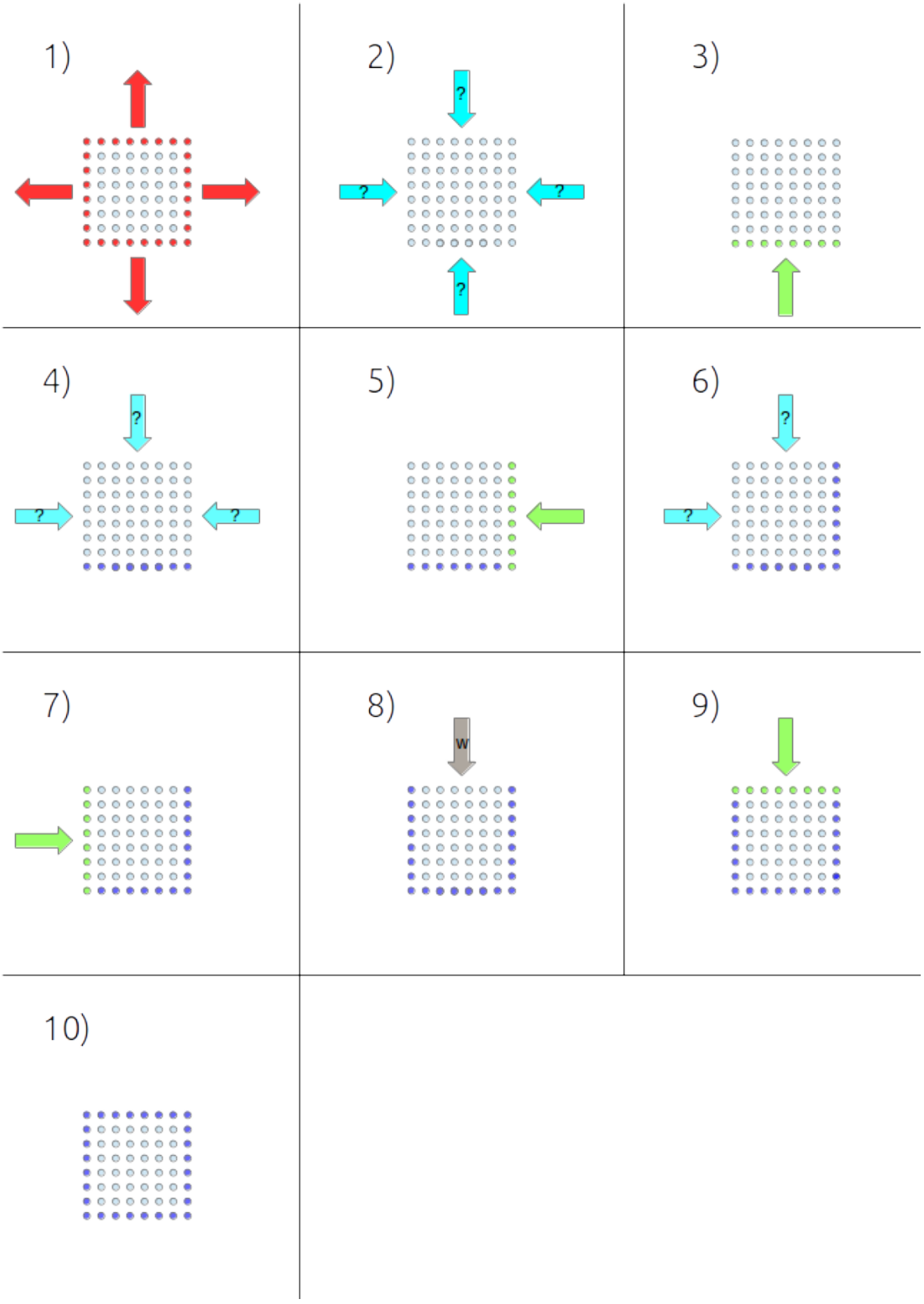

FIGURE 5.2: First In First Out communication pattern.

5. Receives the first message to get in.
6. Polls the remaining two neighbours for messages.
7. Receives the first message to get in.
8. Waits for the last message.
9. Receives the last message.
10. Waits until all the messages have been successfully delivered.

The figure 5.2 is the representation of this algorithm. In this figure the order of the receptions is arbitrary, it will be determined for the order of reception of the messages. This algorithm also avoids cascades of events and moreover to spend time waiting while it is possible to process messages that are already available.

5.2.3 Non-blocking receive function.

Another simpler approach that should solve some of the problems that the current pattern has, is simply to use only non-blocking communications both for sending and receiving.

The pattern is the following:

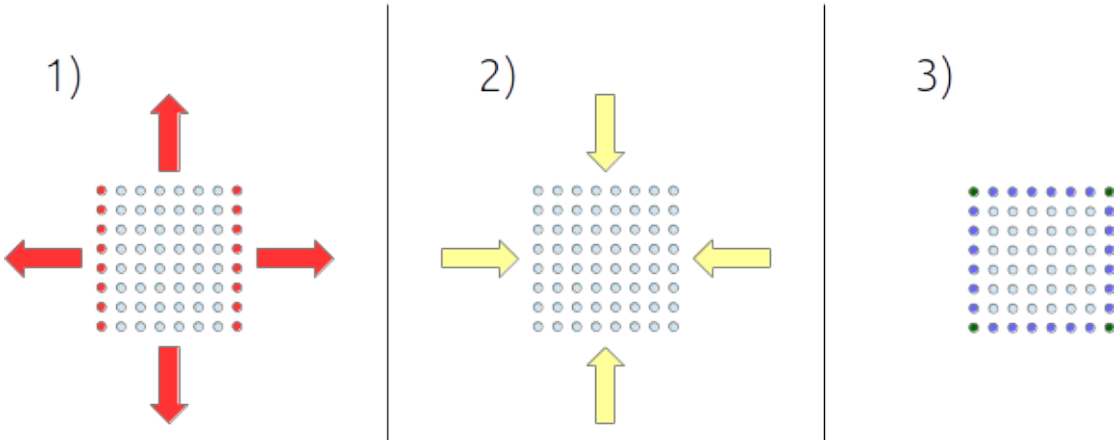

FIGURE 5.3: Non-blocking receive function communication pattern.

1. Every process sends a message to the all their neighbours.
2. Every process starts the non-blocking receiving routine MPI_Irecv for all the neighbours.
3. Uses Wait_all routine to wait for all communications to end.

We could expect to have better results with this approach but the results show that in some cases it is slower that the other approaches.

As we do not know exactly how this MPI functions work, we do not know the causes of this results.

5.3 Results

The results corresponding to this section are from a test program developed in C language to simulate the same comunication scheme that occurs in the NEMO model. In order to test and compare the performance of the four different algorithms some experiments were done using the four algorithms for a different number of processors.

In the first row of tests the non-blocking receive function algorithm was not tested. Later on we also tested the non-blocking receive function algorithm and considered that the plot of the time's distribution would give more information than the simple mean and deviation, since the distribution plays a relevant role on the behaviour of the system.

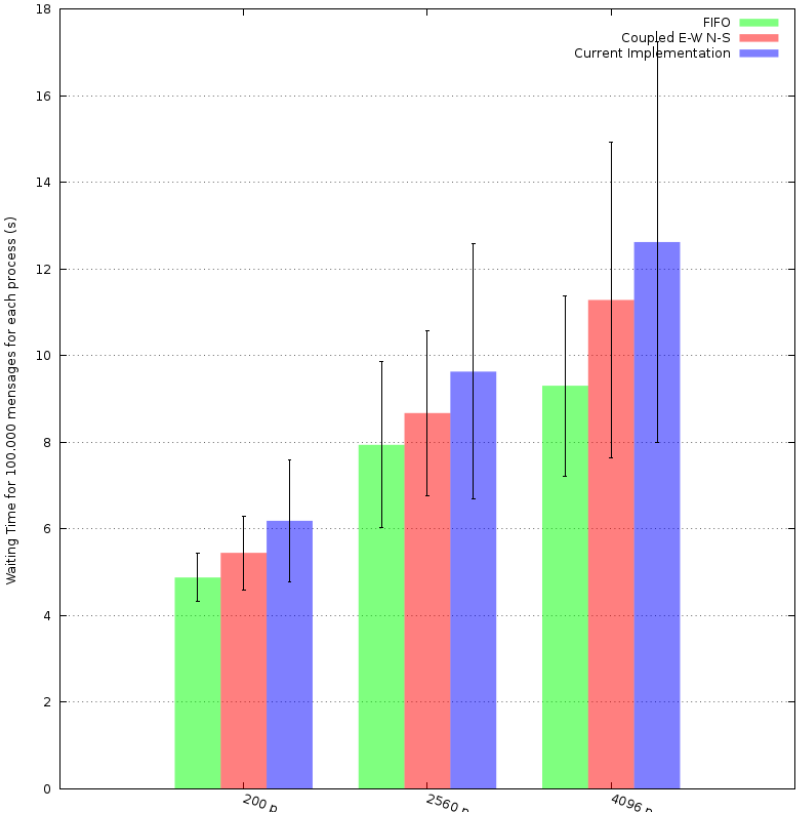

FIGURE 5.4: Evolution of waiting time for different patterns.

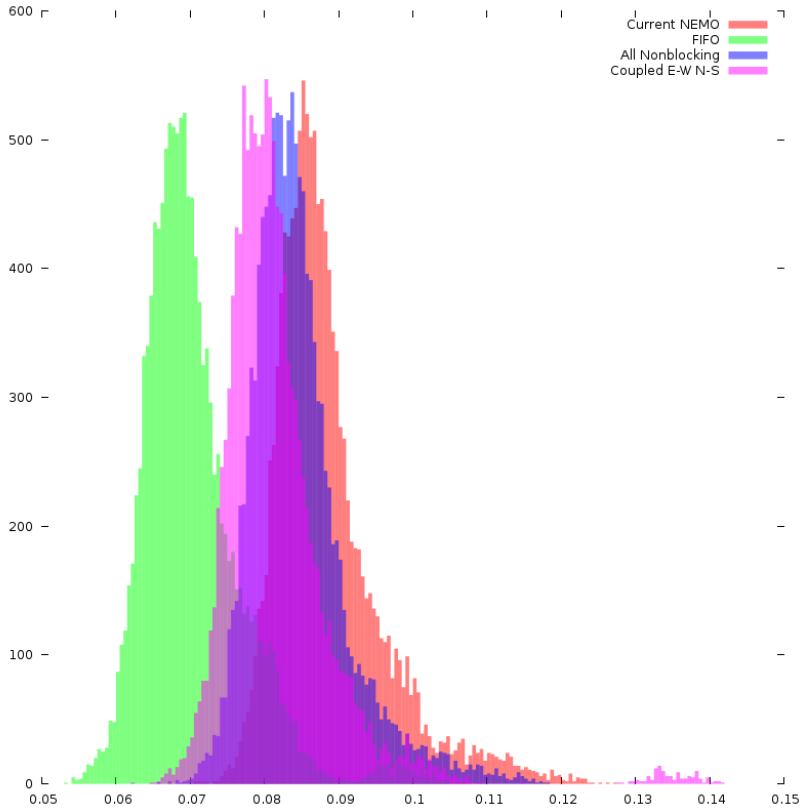

FIGURE 5.5: Waiting time distributions for a run with 128 processors

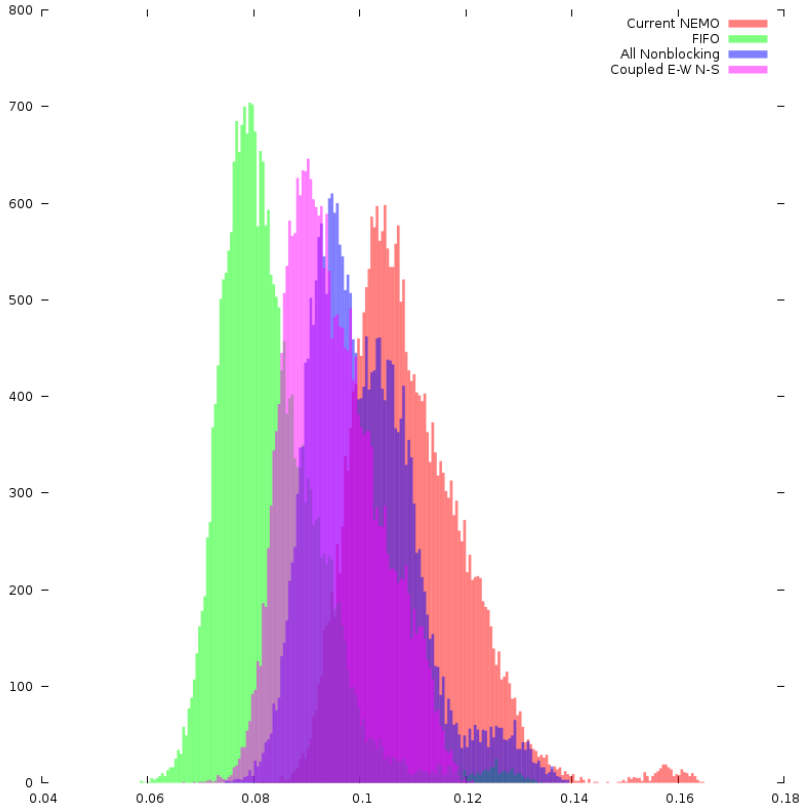

FIGURE 5.6: Waiting time distributions for a run with 256 processors

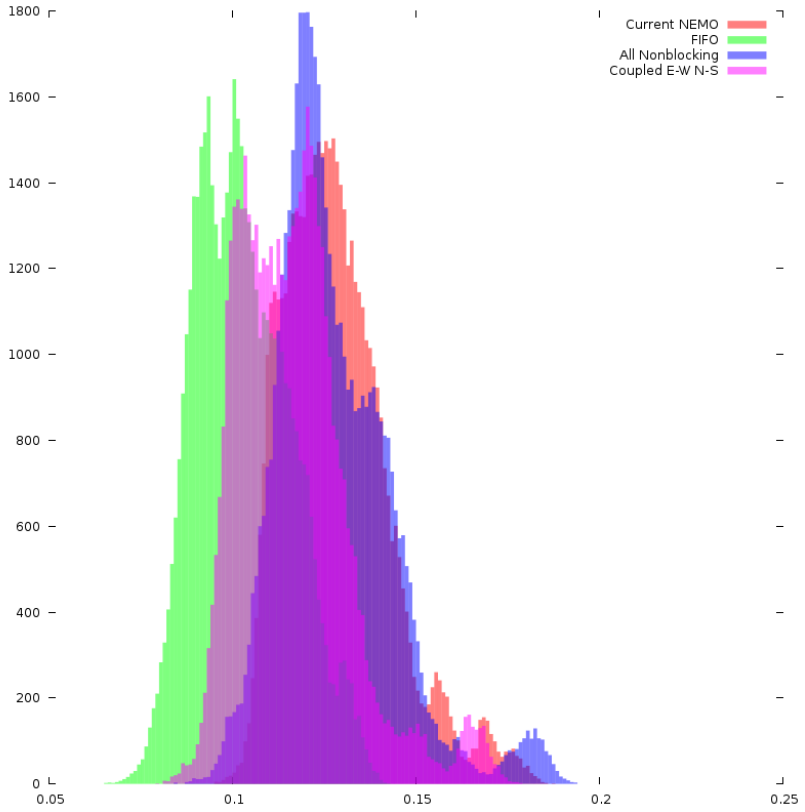

FIGURE 5.7: Waiting time distributions for a run with 512 processors

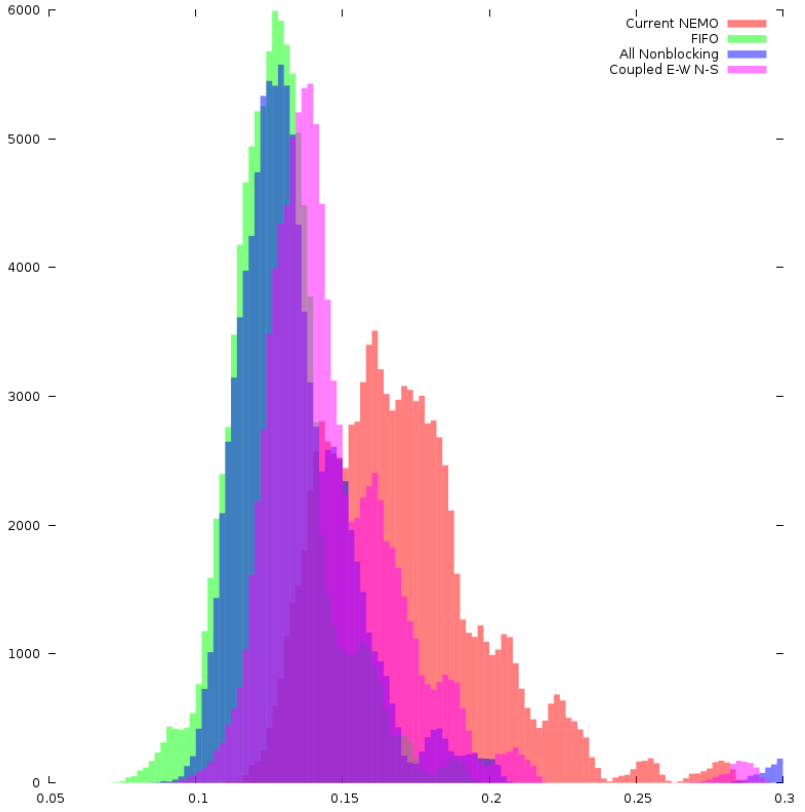

FIGURE 5.8: Waiting time distributions for a run with 1024 processors

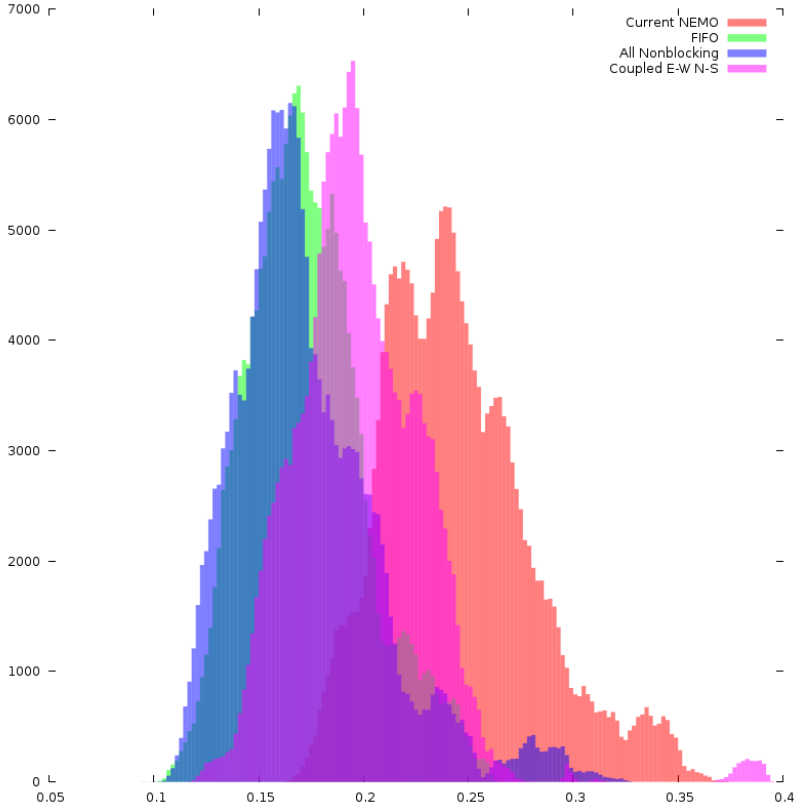

FIGURE 5.9: Waiting time distributions for a run with 2048 processors

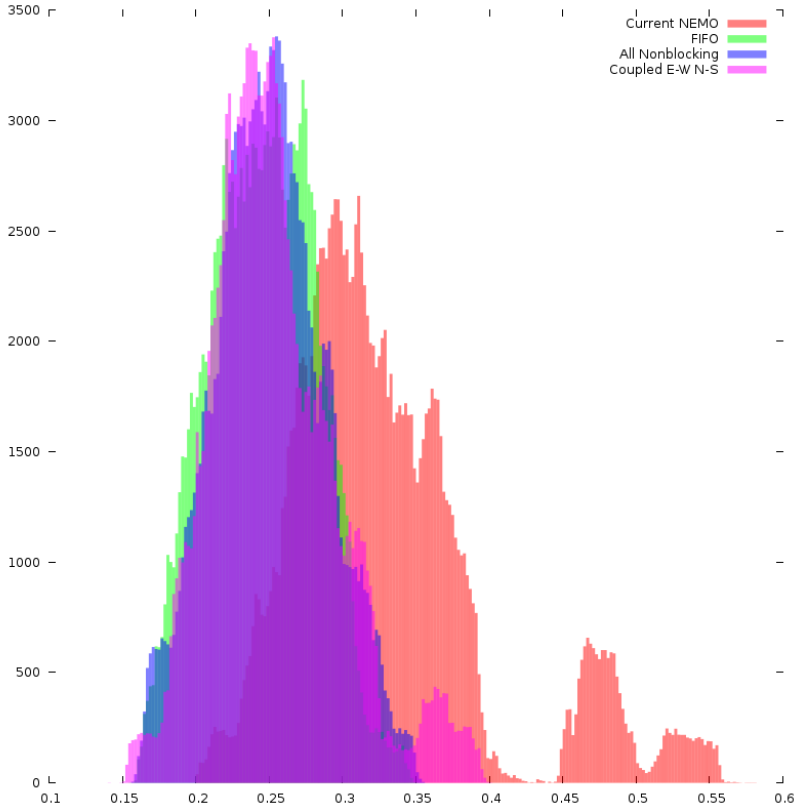

FIGURE 5.10: Waiting time distributions for a run with 4096 processors

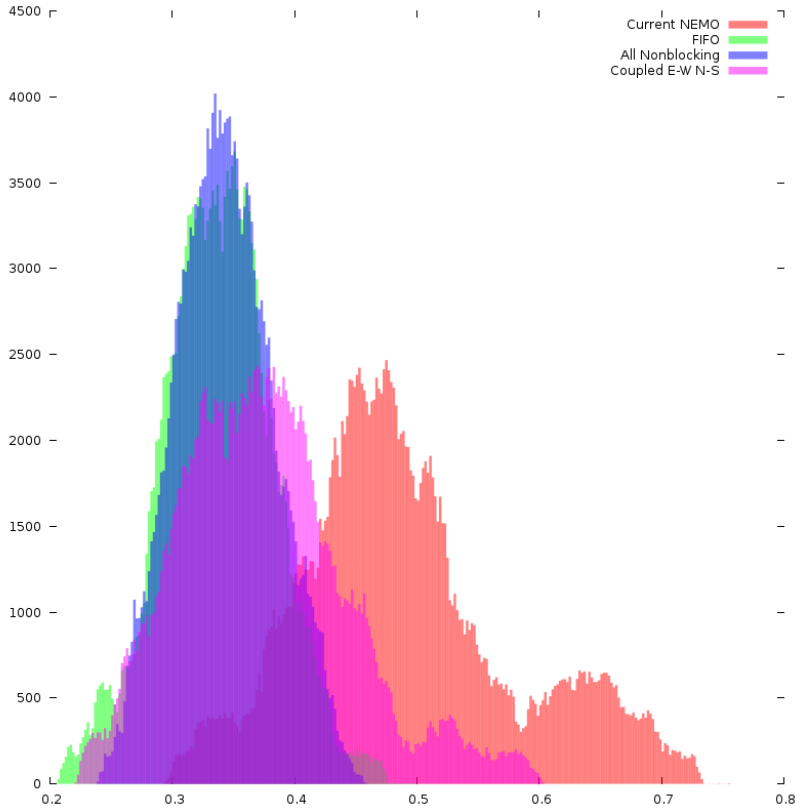

FIGURE 5.11: Waiting time distributions for a run with 8192 processors

The following graphics show the time spent waiting for each processor to receive the messages of its neighbours. In the first plot (Figure 5.4, the number of messages is 100.000 and the size of the messages corresponds to 1000 byte. This quantities are similar to the number of messages of 5 simulated days (120 time step) of NEMO. On the distribution plots, the number of messages is 2.000 and the size of the messages is also 250 floats. The rest of the figures (Figures 5.5, 5.6, 5.7, 5.8, 5.9, 5.10 and 5.11) are the histograms of the simulations of 2.000 messages with a size of 1000 bytes.

Chapter 6

Conclusions and Open Lines

In this chapter the conclusions of the thesis are briefly exposed and the main Open Lines are enumerated.

6.1 Conclusions

To face the possibility of executing the NEMO model with the modern HPC's with thousands or even millions of processors, some changes can not be done by just adapting and modifying the old code and it is necessary to rethink some important elements.

This thesis show that it is important for further developements to discuss two possibilities. If it is sustainable to maintain only one synchronous communication routine, used by all the model routines, or if it is possible to allow asynchronous communication, even though only on some parts of the model. It is also necessary to be aware of the improvements done in algorithms for massive parallel applications as the problems of NEMO probably are shared with other applications.

This work also show that the current technical limitations of the model can be reduced to improve the efficiency with some hundreds or even few thousands of processors. The profilings show that some of these limitations are the file access and how the communications are taking place.

As it is explained in this thesis, the need of synchronous communication itself is a big issue in the current NEMO application. Moreover, how the communication routine is implemented also degrades the performance, as it requires even more synchrony than the demanded by the algorithms. To solve this, a few improvements in the communication pattern were studied.

The results of this study are promising and effort should be done in order to apply the changes into the NEMO code.

6.2 Open Lines

The main Open Line that follow this thesis is the application of a different communication pattern into the NEMO code. Some people involved in NEMO's developement have shown interest in the results of this thesis and have encouraged the inclusion of these changes into the code. As it is shown in the last document delivered from the NEMO community, the *"NEMO's development strategy"* released on June 2014, the efficiency and performance of the NEMO model is really a matter of interest for developers and users.

Each of the elements identified as a possible bottlenecks of the model can be more deeply studied.

Appendix A

HPC Platforms

During this thesis, all the tests and executions were performed in two different HPC's:

1. Ithaca
2. MareNostrum3

A.1 Ithaca

The Ithaca HPC is a small cluster located at the Catalan Institute for Climate Sciences (IC3) and it's characteristics are the following:

A.1.1 Hardware

- **Nodes and cores:** 48 homogeneous nodes (384 cores)
- **Motherboard:** Sun Blade X6270 servers
- **Processor:** Dual Quad-Core Intel Xeon 5570(2.93 GHz)
- **Main memory:** 8 GB per node
- **Interconnect:** Infiniband (IB) and Gigabit Ethernet
- **Scratch storage:** 7 TB of Sun storage J4200
- **Permanent storage:** 48 TB of Sun FIRE X4500 network-attached storage
- **Additional permanent storage:** 97 TB of Sun StorageTek SL500

A.1.2 Software

- **Operating System:** SUSE Linux Enterprise Server 11 (x86_64)
- **Queue Scheduler:** OGS/GE 2011.11
- **Compiler(s):** GNU v4.3.4, Intel v13.0.1
- **MPI(s):** OpenMPI v1.6, IntelMPI v4.0.3

A.2 MareNostrum3

The MareNostrum3 HPC is a supercomputer located at the BSC center at UPC north campus in Barcelona. (IC3) and it's characteristics are the following:

A.2.1 Hardware

- **Nodes and cores:** 36 supernodes iDataPlex with 84 nodes each one. Each node with 2 8-core proccessor card (48384 processors into 3024 nodes)
- **Motherboard:** IBM dx360 M4
- **Processor:**Intel SandyBridge-EP E5-2670 (2.6 GHz)
- **Main memory:** 32 GB per node, 2GB per processor
- **Interconnect:** Infiniband (IB) and Gigabit Ethernet

A.2.2 Software

- **Operating System:** Linux - SuSe Distribution 11 SP2
- **Queue Scheduler:** LSF v9.1
- **Compiler(s):** GNU v4.3.4, Intel v13.0.1
- **MPI(s):** OpenMPI v1.5.4, IntelMPI v4.1.0

Bibliography

[1] Madec G. 2008: "NEMO ocean engine". Note du Pole de modélisation, Institut Pierre-Simon Laplace (IPSL), France, No 27 ISSN No 1288-1619

[2] Vancoppenolle M., Bouillon S., Fichefet T., Goosse H., Lecomte O., Morales Maqueda M.A., and Madec G., 2012 :" LIM The Louvain-la-Neuve sea Ice Model". Note du Pole de modélisation, Institut Pierre-Simon Laplace (IPSL), France, No 31 ISSN No 1288-1619.

[3] XML IO Server: forge.ipsl.jussieu.fr/ioserver/

[4] Ismael Herrera , Luis M. de la Cruz and Alberto Rosas-Medina 2014: "Nonoverlapping discretization methods for partial differential equations"

[5] Fiona Reid 2011 "The NEMO Ocean Modelling Code: A Case Study.“ 2010. Paper presented at Cray User Groups (CUG) 2010, Edinburgh, United Kingdom

[6] Italo Epicoco, Silvia Mocavero, Giovanni Aloisio 2011: "NEMO-MED: Extra-halo performance model“ , CMCC Research Paper RP0099,

[7] Italo Epicoco, Silvia Mocavero, Giovanni Aloisio 2011: "NEMO-MED: Optimization and improvement of scalability“ , CMCC Research Paper RP0096

[8] Barcelona Supercomputing Centre 2009: "Marenostrum user's guide“ Technical report, BSC-CNS

[9] Luisa D'Amore, Vania Boccia, Luisa Carracciuolo, Almerico Murli. 2013: "Driving NEMO towards Exascale: introduction of a new software layer in the NEMO stack software“ , CMCC Research Papers RP0190

[10] Mocavero S., Epicoco I., Aloisio G. 2011 : "The Nemo Oceanic Model: Improvement of Scalability on MareNostrum“ , Science and Supercomputing in Europe, research highlights 2010

[11] Epicoco I., Mocavero S., Aloisio G. 2011: "NEMO Oceanic Model Optimization“, Science and Supercomputing in Europe, S. Monfardini - Research highlights 2010

[12] Epicoco I., Mocavero S., Aloisio G. 2011: "The NEMO Oceanic Model: Computational Performance Analysis and Optimization" , Proceedings of the 2011 IEEE International Conference on High Performance Computing and Communications