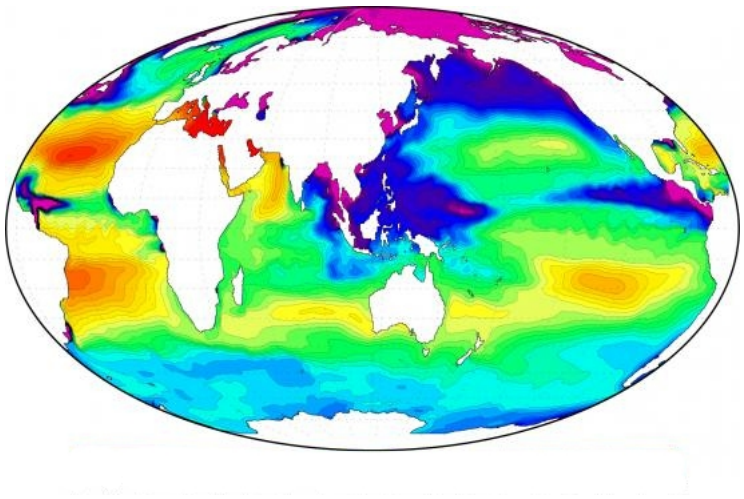

Polytech'Nice-Sophia

Final Term Project

# Handling a 3D oceanographic simulation program

*Authors:*
Denis Brun
Aymeric Loche

*Supervisor:*
Dr. Pierre Dreyfuss



February 3, 2011

# Contents

# List of Figures

## Acknowledgements

We would like to thank M. Dreyfuss for helping us during this three-week project and providing us with useful information.

## Introduction

### What is NEMO ?

NEMO (Nucleus for European Modelling of the Ocean) is a state-of-the-art modeling framework for oceanographic research, operational oceanography seasonal forecast and climate studies.
NEMO includes 4 major components:

- the blue ocean (ocean dynamics, NEMO-OPA)
- the white ocean (sea-ice, NEMO-LIM)
- the green ocean (biogeochemistry, NEMO-TOP) ;
- the adaptative mesh refinement software (AGRIF) ;
- some reference configurations allowing to set-up and validate the applications
- a set of scripts and tools (including pre- and post-processing) to use the system

NEMO is used by a large community: 240 projects in 27 countries (14 in Europe, 13 elsewhere) and is available under the CeCILL license (public license), the evolution and reliability of the code are organised and controlled by a European Consortium between CNRS, Mercator-Ocean (ocean forecasters), UKMO (UK Meteorological Office) and NERC (Natural Environment Research Council)

### What is the purpose of this project ?

The goal here is to handle such a big code, overcome difficulties due to our specific architecture, run the code, and finally display the result of the computation.

# Chapter 1

# Code installation

First of all, setting up such a code is not easy, NEMO aims to be the european flagship engine for ocean modelling, this is a huge open project with thousands of lines. As always, when it comes to open source projects, the code is not made to run on an single computer architecture, NEMO can be installed either on supercomputers or on a desktop PC depending on the purpose of your computation (that is to say how powerful your central processing unit has to be).

## 1.1 Compilation

Installing the code on our computers was a difficult part. In fact, we tried to compile a NEMO version which had been modified and was not fitted to our LINUX distribution. Besides depending on the compiler we used, some code modifications had to be done in order that the program could be compiled. Besides, it affects the installation of the netCDF libraries.

### 1.1.1 Compilers

A compiler is a computer program (or set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language, often having a binary form known as object code). The most common reason for wanting to transform source code is to create an executable program. source : wikipedia

#### The Gfortran compiler

Gfortran is the GNU fortran compiler, this the free native compiler on Ubuntu distributions, that is the one we used in the first place. It has been developed by a community of developers that give their free time to contribute to the project, this is not a professional tool. This compiler is based on C compiler which means it is not optimized.

### The Intel Fortran Compiler

There are also other fortran compilers such as ifort from Intel which is a proprietary compiler but free for a non commercial use. This compiler is developed by a world-class team of engineers so the compilation is efficient, furthermore the compiler include the Intel Math Kernel Library that provides a high-level optimization for this scientific kind of code.

### Comparison test

To compare the efficiency of the two compilers run for two different versions of the program, to do so, we had to have 2 versions of the netCDF library each compiled with the two different compilers, we also needed two versions of the OPA engine, generated with the two compilers. Here are the result of the test, the opa engine runtime functions of timesteps and compilers :

| | Gfortran | iFort |
|---|---|---|
| Timesteps | Time(sec) | Time(sec) |
| 50 | 3,4 | 1,5 |
| 100 | 6,7 | 3,3 |
| 200 | 13,4 | 5,23 |
| 400 | 26,7 | 10,6 |
| 800 | 54,1 | 21,3 |
| 1600 | 107 | 43,1 |
| 3200 | 215 | 86,5 |
| 6400 | 427 | 179 |

These runs have been made for the same configuration on the same computer.

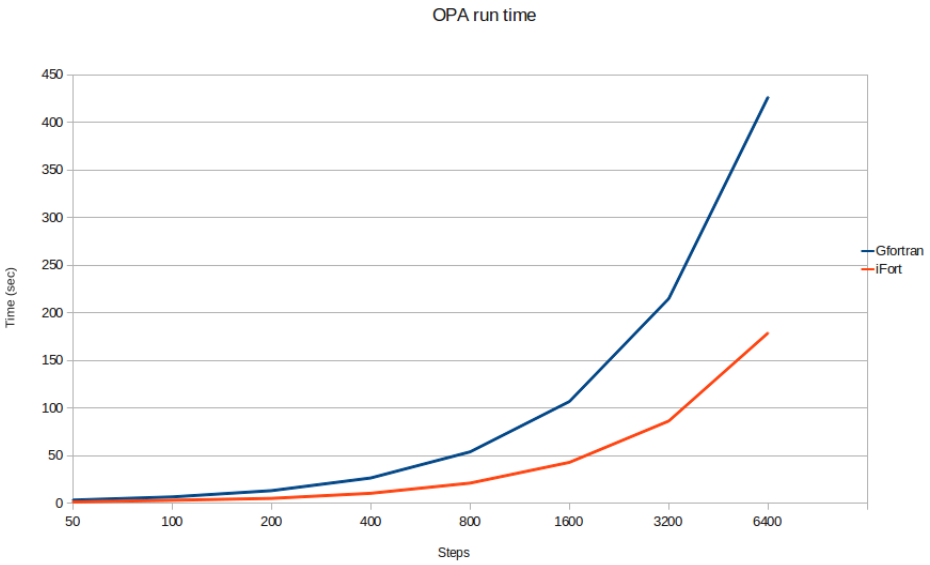



Figure 1.1: Runtime functions of timesteps and compilers

The Intel compiler makes the program run twice as fast as the GNU fortran compiler, and maybe more with a huge number of steps.

### 1.1.2 NEMO Revisions

NEMO is a project in constant development, from one day to another the code may have changed a bit, that is why sometimes with the same computer configuration the code might not compile, execute...

## 1.2 NEMO Overview

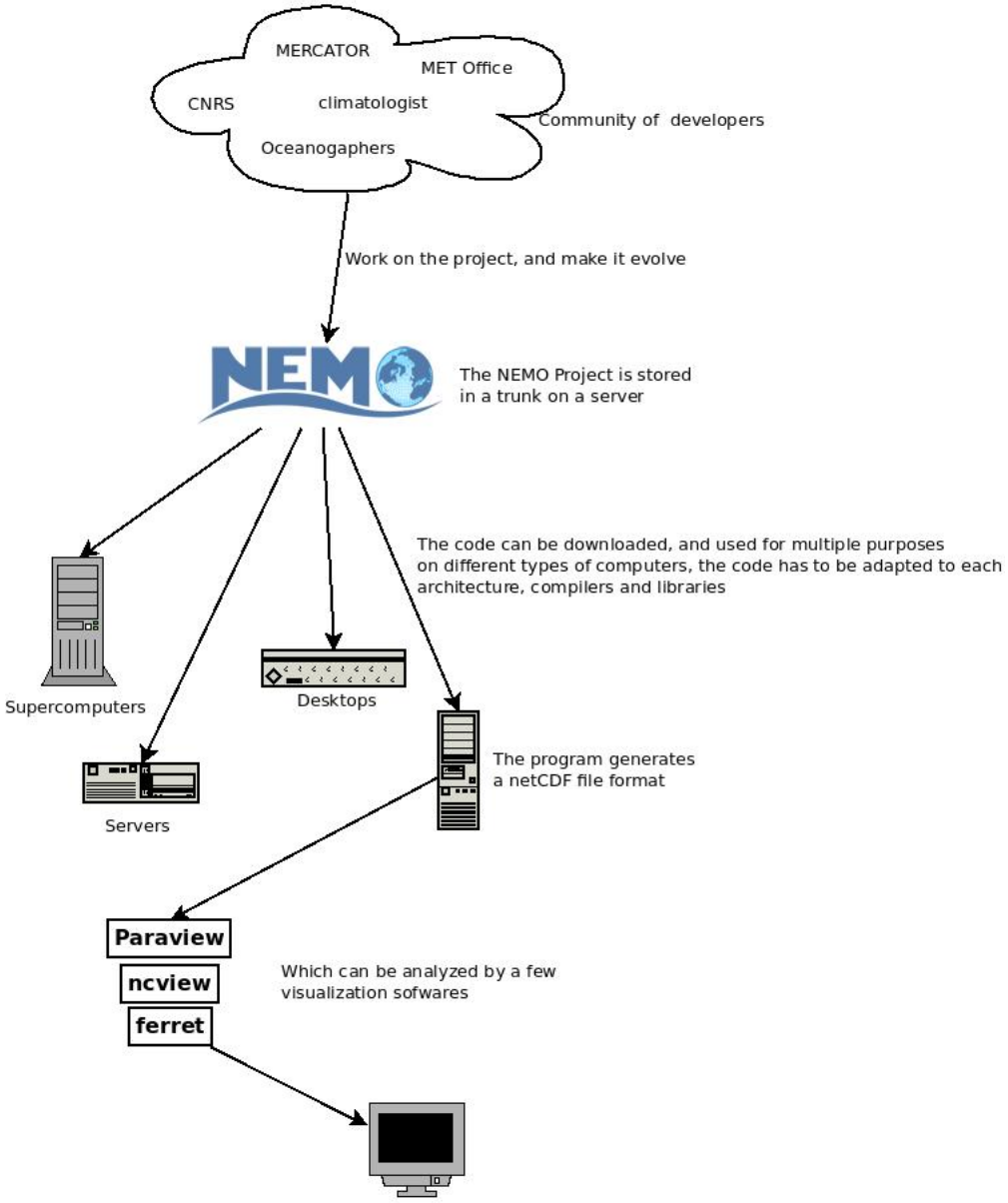



Figure 1.2: NEMO project overview

# Chapter 2

# Mathematical aspects

## 2.1 Presentation of the primitive equations

Before coding, it is important for us to have a model. This modelling comes from physical and phenomenal laws which best described the nature and its behaviours. Here is the question of interpreting the primitive equations. The studied variables are the temperature $T$, the salinity $S$, the velocity $U$ in 3 dimensions and the water height $\eta$.

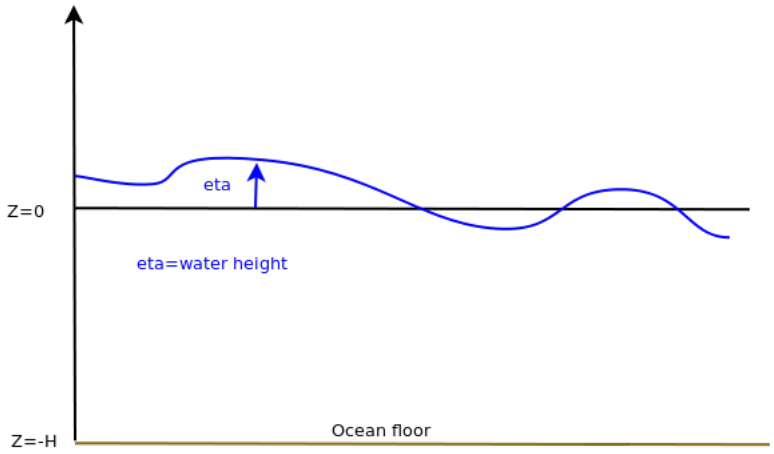



Figure 2.1: Simplified scheme of the ocean floor

$H = H(x, y)$ represents the ocean floor profile (topography) and $\eta = \eta(x, y)$ the water height variation compared with the reference surface $z = 0$.

In order to simplify the primitive equations, some hypothesis can be made. To begin we can neglect the ocean's depth with respect to the radius of the Earth. Besides we will consider the gravity parallel to the radius of

the Earth. NEMO uses the following primitive equations:

$$\frac{\partial T}{\partial t} = -\nabla \cdot (TU) + D^T + F^T$$

$$\frac{\partial S}{\partial t} = -\nabla \cdot (SU) + D^S + F^S$$

$$\rho = \rho(T, S, p)$$

$$\frac{\partial U_h}{\partial t} = -\left[(\nabla \wedge U) \wedge U + \frac{1}{2}\nabla(|U|^2)\right]_h - f \cdot U_h - \frac{1}{\rho_0}\nabla_h p + D^U + F^U$$

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

$$\nabla \cdot U = 0$$

$$\frac{\partial \eta}{\partial t} = -div_h((H + \eta)\overline{U_h})$$

where:

- $U_h = U_h(x, y, z)$ is the horizontal velocity (2-component velocity vector)
- $\overline{U_h} = \overline{U_h}(x, y)$ is the horizontal velocity $U_h$ vertically averaged
- $\rho$ for water density

## 2.2 GYRE configuration

We can see the studied domain compared with the Earth. It is a North-Atlantic part situated on a middle latitude:

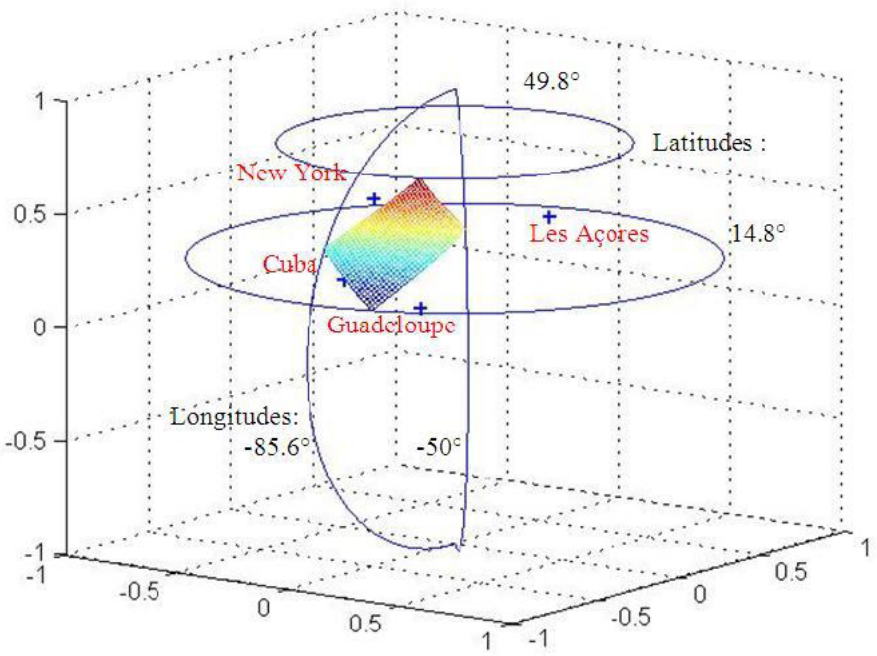



Figure 2.2: GYRE domain

Here a 3D representation by Paraview visualisation software.

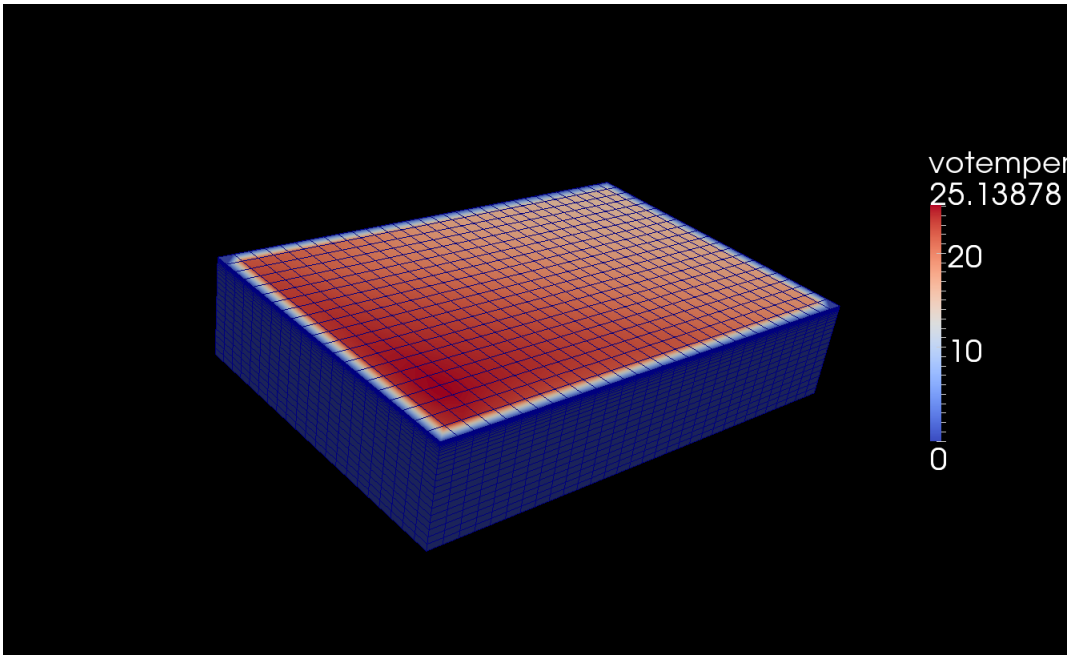



Figure 2.3: GYRE domain in 3D representation

The maximum depth is 4451.26$m$ and is divided into 31 layers. Here we can see the different water heights considered in our configuration:

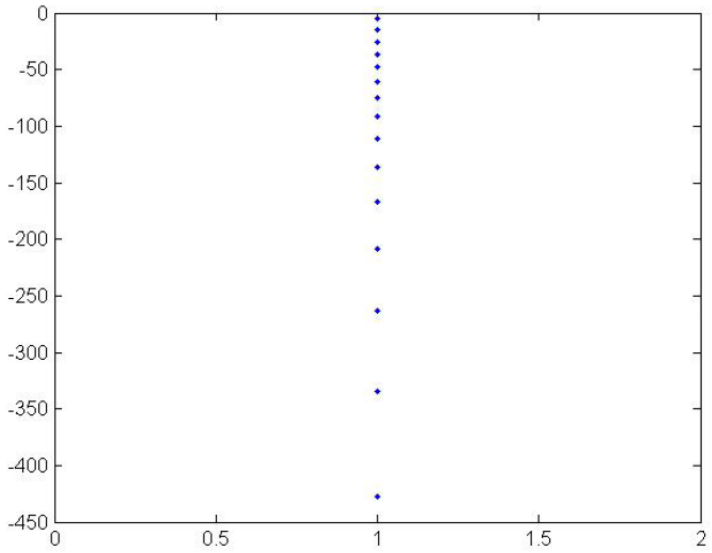



Figure 2.4: Representation of the first 15 layers, depth: 0 to -500m

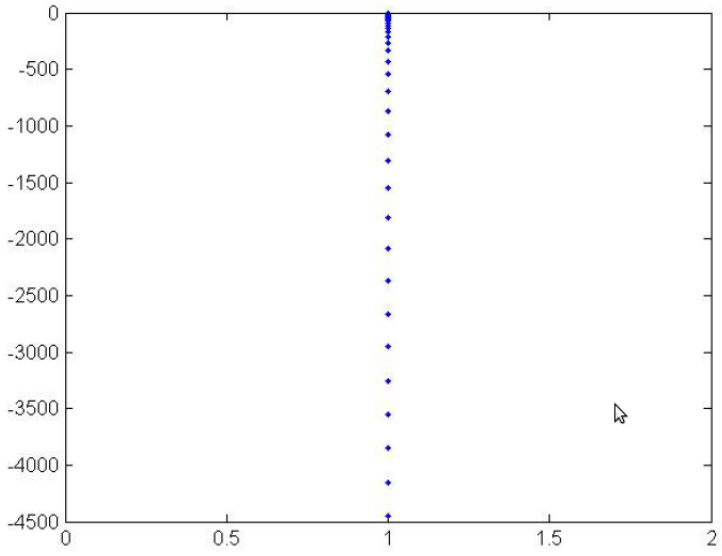



Figure 2.5: Representation of all the vertical layers

### 2.2.1 Coordinates system definition

In NEMO code several $z$ coordinates are conceivable but only one is chosen. To complete the vertical mesh, 3 parameters are set:

- the bathymetry in meter
- the number of levels on $z$ coordinate (31)
- the transformation coefficients of the domain, for example from a spherical domain to a flat domain

All variables and their value are written in a "namelist" file.

### 2.2.2 Mesh configuration

In NEMO code, curvilinear coordinates $(i, j, k)$ are used and permit us to be inside a spherical mesh.

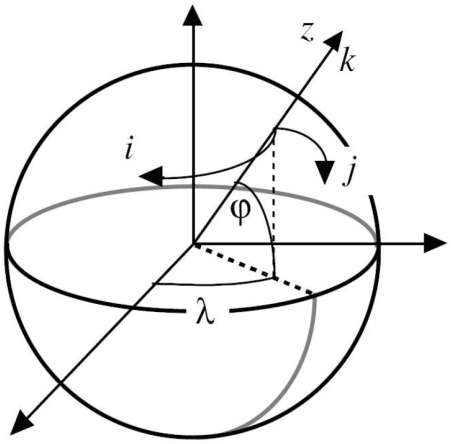



Figure 2.6: System of geographical coordinates

It is defined by the following variables:

- $\phi \in [\frac{-\pi}{2}; \frac{\pi}{2}]$ gives us the latitude
- $\lambda \in [-\pi; \pi]$ gives us the longitude
- $z = a + z(k)$ where $a$ is the radius of the Earth and $z(k)$ represents the water height.

## 2.3 Boundary conditions

We chose "closed" boundaries that means fluxes through the lateral boundary were null all around the surface we studied.

Surface conditions: windy conditions were set and were applied on the surface. The flux on $U_h$ is prescrited as:

$$\frac{\partial U_h}{\partial z} = \underbrace{\frac{1}{\rho_0}\begin{pmatrix}\tau_u\\ \tau_v\end{pmatrix}}_{\text{known}}$$

Ocean floor: a flat ocean floor is modelled (GYRE configuration) [$ntopo = 0$ variable in the namelist].
Friction condition is determined as:

$$\frac{\partial U_h}{\partial z} + kU_h = 0$$

# Chapter 3

# Data visualisation

NEMO provides a specific type of files with the result of the computation in it (NetCDF format). The goal here is to visualise this set of data.

## 3.1 The NetCDF files

NetCDF (Network Common Data Form) is a set of software libraries and data formats that support the creation, access, and sharing of scientific data. The project homepage is hosted by the Unidata program at the University Corporation for Atmospheric Research(UCAR). The format is an open standard. It is commonly used in climatology, meteorology and oceanography applications.

## 3.2 Visualisation sofwares

### 3.2.1 Ncview

Ncview is a visual browser for NetCDF format files. This program is a simple, fast, a very handy tool for visualising fields in a NetCDF file.

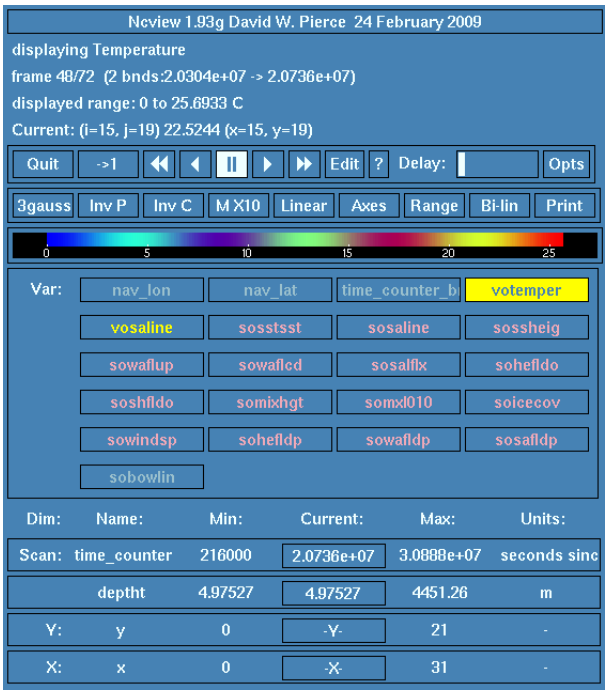



Figure 3.1: The Ncview panel for temperature reading

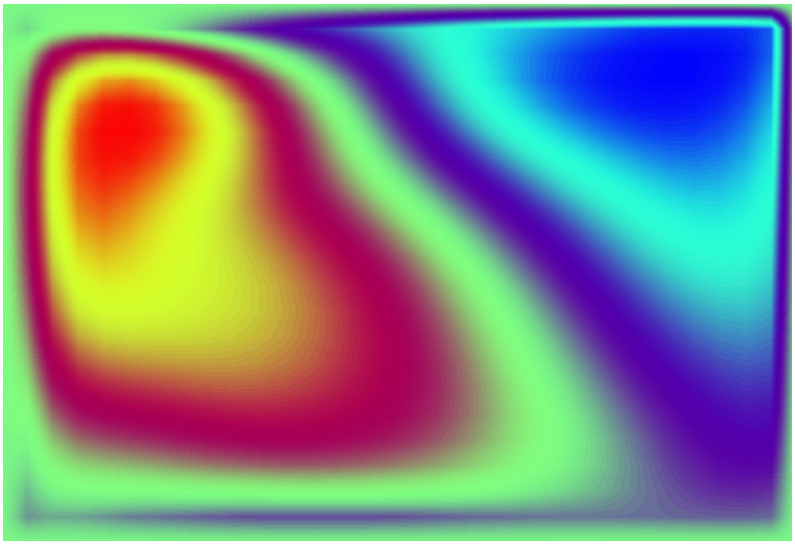


Figure 3.2: Visualisation of the water height by Ncview

### 3.2.2 Ferret

Ferret is an interactive computer visualisation and analysis environment designed to meet the needs of oceanographers and meteorologists analysing large and complex gridded data sets. This software permits us to read NetCDF files which contain all the results on the different variables.

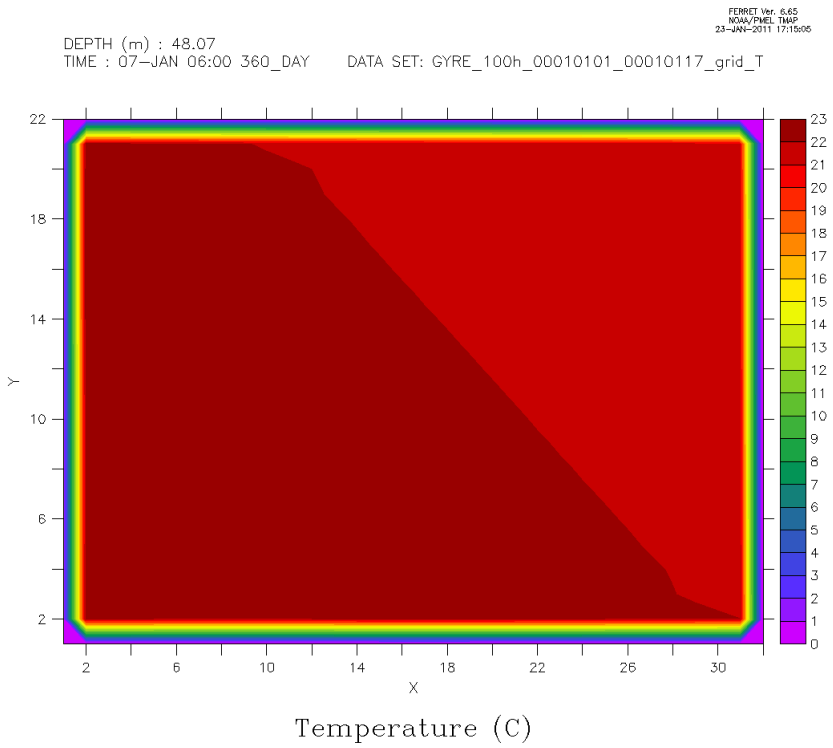



Figure 3.3: Visualisation of the temperature by Ferret

### 3.2.3 Paraview

ParaView is an open-source, multi-platform data analysis and visualisation application. The data exploration can be done interactively in 3D or programmatically. ParaView was developed to analyze extremely large data sets using distributed memory computing resources. It can be run on supercomputers to analyse data sets of terascale as well as on laptops for smaller data.

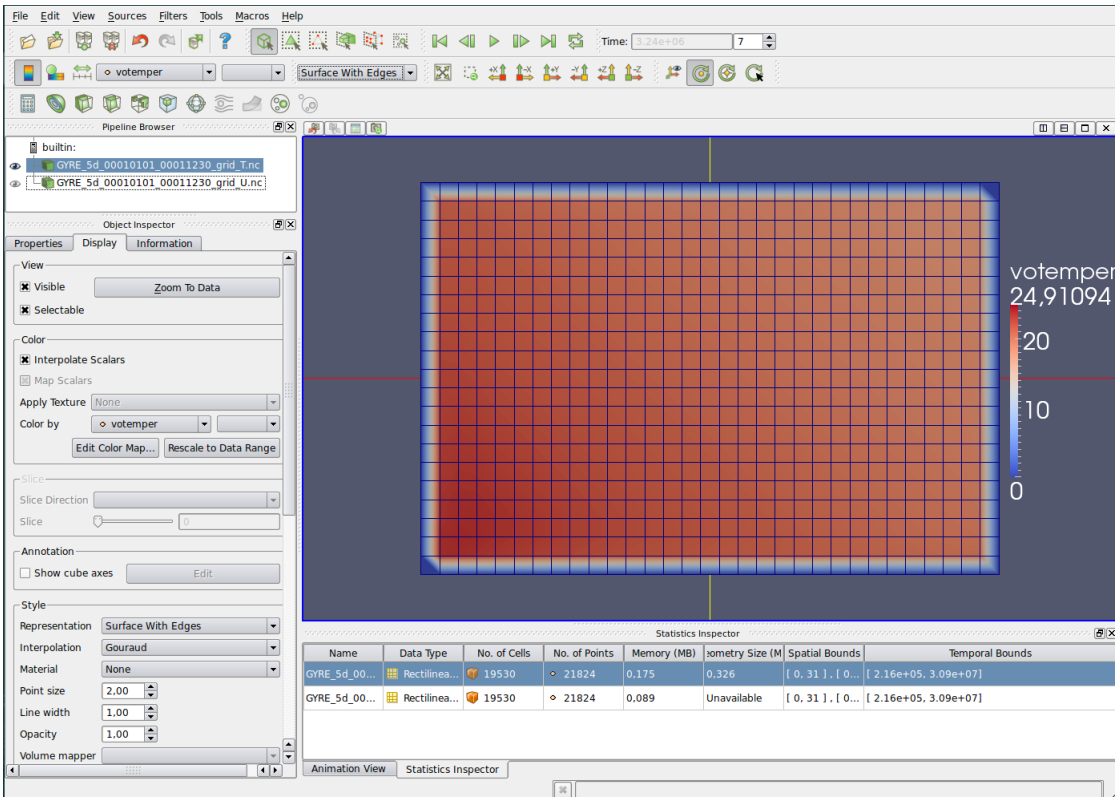



Figure 3.4: Paraview, software overview

Paraview includes interesting features such as the slice tool, we were able to display the temperature for different slices of the domain, this tool is very comprehensive, there are default slices (orthogonal to the axes) as well as customizable cuts.

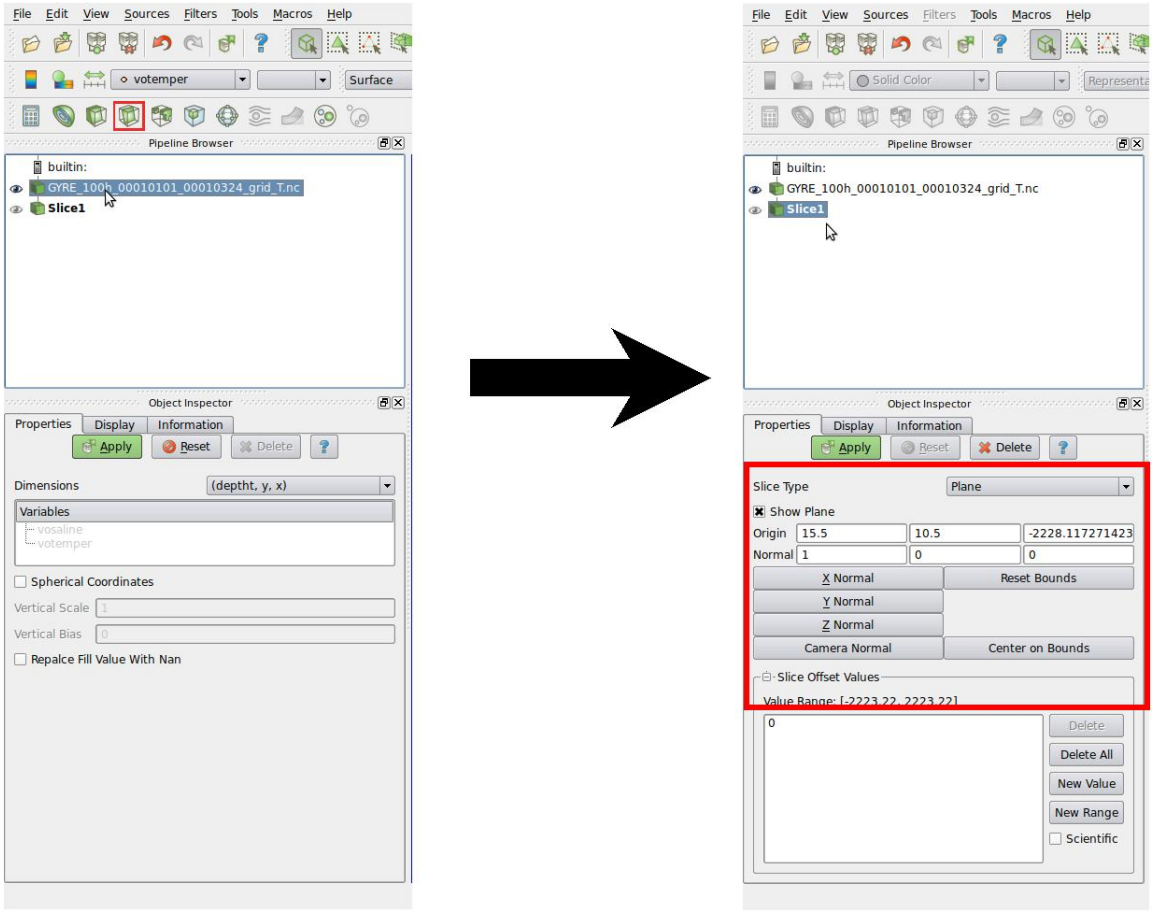



Figure 3.5: Paraview slice tool

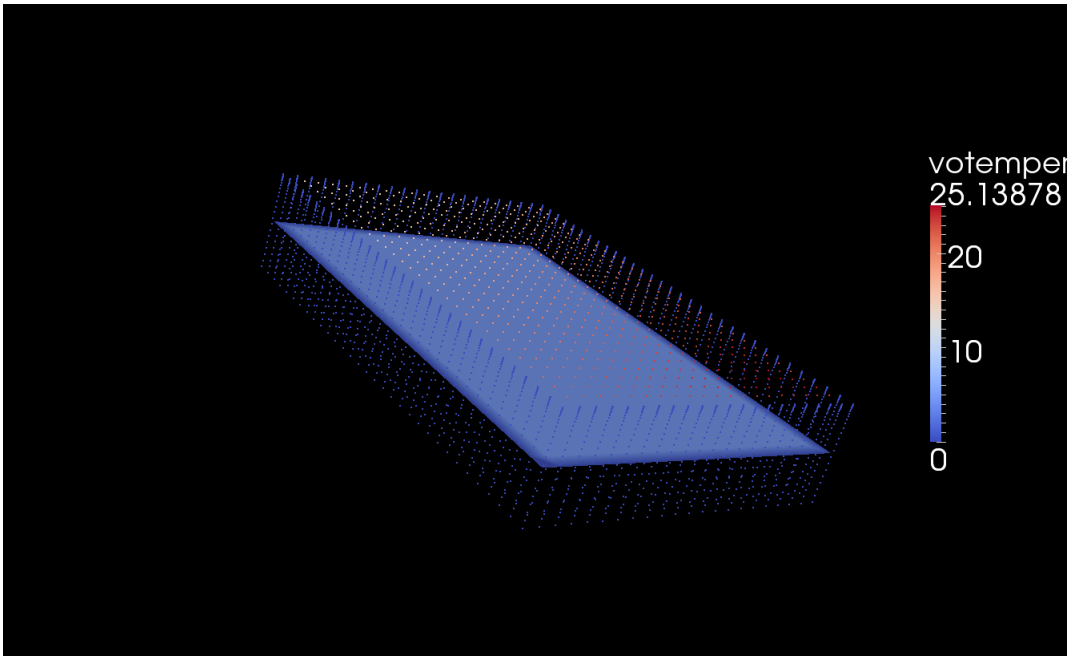



Figure 3.6: An horizontal slice showing the temperature $\approx$ 2250 meters under the ocean' surface

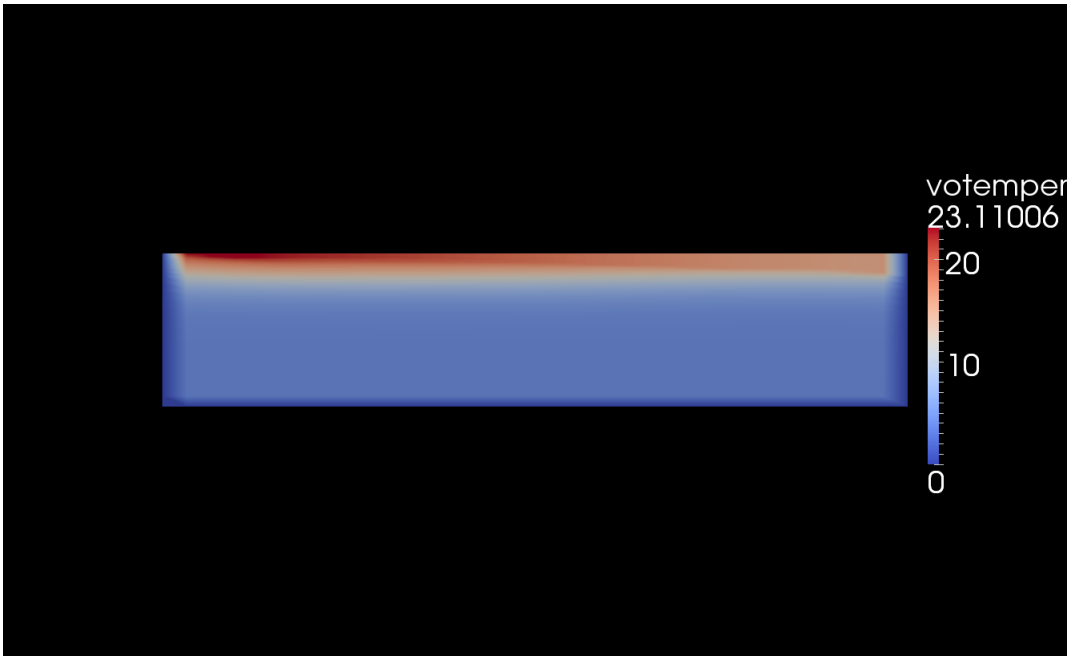



Figure 3.7: Slice from the surface to 4500 meters showing the temperature

# Chapter 4

# Appendix : NEMO full-setup

Here is an example of a complete installation of NEMO on UBUNTU 10.04 32 bits, with the Intel Fortran Compiler.

## 4.1 Installing the Intel Fortran compliler

1. First things first, install the prerequisite software packages:

   ```
   sudo apt-get install rpm build-essential
   ```

2. Register at Intel's site for a non-commercial version of the Intel Fortran compiler.

   ```
   http://software.intel.com/en-us/articles/non-commercial-software-download/
   ```

   Intel's site will send you an email that starts with the following:

   ```
   Thank you for registering the Intel(R) Fortran Compiler for Linux*.

   SAVE THIS SERIAL NUMBER
   Your serial number for this registration is...
   ```

3. Download the compiler archive.

4. Extract the compiler archive :

   ```
   tar xvzf l_fcompxe_*.tgz
   ```

5. Run the installer

   ```
   cd l_fcompxe_*
   sudo ./install.sh
   ```

6. Press Enter to accept the default install directory for the compiler as well as default options...

7. You have to edit /.bashrc

   ```
   gedit ~/.bashrc
   ```

8. add the following lines (that should matches you architecture)

   ```
   PATH="/opt/intel/compilerpro-12.0.1.107/bin:$PATH"
   export PATH
   LD_LIBRARY_PATH="/opt/intel/compilerpro-12.0.1.107/compiler/lib:$LD_LIBRARY_PATH"
   export LD_LIBRARY_PATH
   source /opt/intel/compilerpro-12.0.1.107/bin/ifortvars.sh ia32
   LD_LIBRARY_PATH="/opt/intel/compilerpro-12.0.1.107/compiler/lib/ia32:$LD_LIBRARY_PATH"
   export LD_LIBRARY_PATH
   ```

9. Create a symbolic link

   ```
   sudo ln -s /opt/intel/compilerpro-12.0.1.107/compiler/lib/ia32/ifort /bin
   ```

That's it for the Intel Fortran compiler

## 4.2 The netCDF library :

We have to make sure that the library files are compiled with the same compiler we've just installed on our computer.

1. Download the latest version of the library (in our example : netcdf-4.1.1.tar.gz)

2. Extract the files

3. build like this:

   ```
   ./configure --prefix=~/usr/local/netcdf4 --disable-netcdf-4
   make check install
   ```

   If you get the message that netCDF installed correctly, then you are done !

## 4.3 NEMO installation and compilation

1. We had to register on the NEMO website : www.nemo-ocean.eu to get a password and an username.

2. Insall ksh

   ```
   sudo apt-get install ksh
   ```

3. Create the following alias :

   ```
   alias svn_ano='svn co http://forge.ipsl.jussieu.fr/igcmg/svn/modipsl/trunk modipsl'
   ```

4. Create the following workspace directory :

   ```
   mkdir NEMO/TRY
   ```

5. Extract modipsl

   ```
   svn_ano
   ```

6. Extract NEMO

   ```
   cd modipsl/util
   ./model NEMO
   ```

7. Make GYRE configuration

   ```
   ../modeles/UTIL/fait_config GYRE
   ```

8. Edit *TRY/modipsl/util/AA_make.gdef* for target lxiv8, that should looks like this afterward :

   ```
   #-Q- lxiv8    #- Global definitions for Intel compiler (ifort) at LSCE
   #-Q- lxiv8    M_K = make
   #-Q- lxiv8    P_C = cpp
   #-Q- lxiv8    P_O = -P -C $(P_P)
   #-Q- lxiv8    F_C = mpif90 -f90=ifort -c -cpp
   #-Q- lxiv8    #-D- MD    F_D = -g
   #-Q- lxiv8    #-D- MN    F_D =
   #-Q- lxiv8    #-P- I4R4  F_P = -i4
   #-Q- lxiv8    #-P- I4R8  F_P = -i4 -r8
   #-Q- lxiv8    #-P- I8R8  F_P = -i8 -r8
   #-Q- lxiv8    F_O = -O $(F_D) $(F_P) -I$(MODDIR) -module $(MODDIR)
   #-Q- lxiv8    F_L = mpif90 -f90=ifort
   ```

```
#-Q- lxiv8    M_M = 0
#-Q- lxiv8    L_X = 0
#-Q- lxiv8    L_O = -Vaxlib
#-Q- lxiv8    A_C = ar -r
#-Q- lxiv8    A_G = ar -x
#-Q- lxiv8    C_C = cc -c
#-Q- lxiv8    C_O =
#-Q- lxiv8    C_L = cc
#-Q- lxiv8    #-
#-Q- lxiv8    NCDF_INC = /usr/local/lib/netcdf4/include
#-Q- lxiv8    NCDF_LIB = -L/usr/local/lib/netcdf4/lib -lnetcdf
#-Q- lxiv8    #-
```

9. We had to remove the *key_iomput* from */TRY/modipsl/config/GYRE/scripts/BB_make.ldef*

10. Create a file mpi.f90 in modipsl/lib, and then add the following content in it :

```
MODULE mpi
INCLUDE 'mpif.h'
END MODULE mpi
```

11. Compile it with :

```
mpif90 -f90=ifort -c mpi.f90
```

12. Then

```
./ins_make -t lxiv8
```

13. Compile NEMO

```
cd ../config/GYRE
make
```

## 4.4 Run NEMO

1. You have to run NEMO from /TRY/modipsl/config/GYRE/EXP00

2. To run NEMO :

```
../../../modipsl/bin/opa
```

# Conclusion

Despite the lack of time, we succeded in installing and using properly an important code. It took a lot of patience and reading to understand how the primitive equations model such an intricate system.

# References


- *La méthode BFN, une nouvelle méthode d'assimilation de données pour le code NEMO*, rapport de stage, Baptiste CHOUZET.
- Ubuntu documentation, *http://doc.ubuntu-fr.org/fortran*
- NEMO website project, *http://www.nemo-ocean.eu/*
- netCDF documentation *http://www.unidata.ucar.edu/software/netcdf/*
- Paraview documentation, *http://www.paraview.org/paraview/help/documentation.html*
- Wikipedia