UNIVERSITY OF COPENHAGEN

MSc in Physics

The Need for Speed

An Investigation of bottlenecks towards optimization of EC-Earth3-HR (on CRAY XC50)

Nga Ying Lo

Supervised by Jens Hesselbjerg Christensen, Shuting Yang,

Cosmin Eugene Oancea and Marianne Sloth Madsen

Handed in: 05/09/2022

Nga Ying Lo

The Need for Speed: An Investigation of bottlenecks towards optimization of EC-Earth3-HR (on CRAY XC50)

MSc in Physics, Handed in: 05/09/2022

Internal Supervisors: Jens Hesselbjerg Christensen (primary, KU-NBI) and Cosmin Eugene Oancea (KU-DIKU)

External Supervisors: Shuting Yang (DMI) and Marianne Sloth Madsen (DMI)

University of Copenhagen

Faculty of Science

Niels Bohr Institute

Blegdamsvej 17

2100 Copenhagen

Preface

This master thesis project was carried out in collaboration between the Niels Bohr Institute (NBI) and the Department of Computer Science (DIKU) at the University of Copenhagen, and the Danish Meteorological Institute (DMI). The following thesis report was submitted for the degree of Master of Science in Physics with a specialization in Computational Physics at the University of Copenhagen in September 2022.

Course: Physics Thesis, 60 ECTS (NFYK10020E)

Internal supervisors: Jens Hesselbjerg Christensen (primary, NBI) and Cosmin Eugene Oancea (DIKU)
External Supervisors: Shuting Yang (DMI) and Marianne Sloth Madsen (DMI)
Censor: Peter Aakjær

Acknowledgements

First and foremost, I would like to thank all of my supervisors for the continuous supports and guidance they have shown me throughout this thesis project. Their encouraging words and insights are much appreciated every time I have doubts on my progress and capabilities.

I also owe thanks to Jacob Weismann Poulsen and Stephen Reiner at DMI, Philippe Le Sager at the Royal Dutch Meteorological Institute, and Miguel Castrillo at the Barcelona Supercomputing Center, who provided help with any technical difficulties I have encountered with the EC-Earth3 model.

Thank you to my family, friends and the scientists at DMI's Climate Department for the emotional and moral support. A special gratitude to my friends Dana Sarah Ludemann, Johanne Dorothea Hauser and Emily Madeleine Wilbur for all that you have done for me.

Last but not least, an immense thank you to Christine Schøtt Hvidberg (NBI) for her suggestion of this thesis collaboration with DMI.

Abstract

Earth system models are state-of-the-art tools that climate physicists use to understand climate feedbacks, attribute changes to specific drivers, and make projections for the future. With more fine resolution, models would be able to account for fine scale transient processes that are crucial for improving results' fidelity. The drawback is decrease in the model's performance efficiency due to requirement of long computation to extract signals of climate change. Thus, optimization of these models is most necessary to study climate evolution. Currently, the global high-resolution coupled climate model EC-Earth3-HR can simulate about 1.31 years of climate evolution per day on the high performance computing platform (CRAY XC50) at the Danish Meteorological Institute. Meanwhile the standard-resolution of this model's efficiency is 10.07 simulated years per day. In order to improve efficiency of EC-Earth3-HR, scalability and performance analysis are performed to diagnose the bottlenecks of the model. And, various optimization methods are considered in order to address these bottlenecks. With the tools available on CRAY XC50, the optimal load balance was not obtained, due to incompatibility between performance analysis tool with the ocean component of EC-Earth3-HR. From scalability analysis, the ocean component shows to be performing worse compared to the atmosphere component. The least optimal subroutine within the ocean component is identified to be one handling the computation of sea-ice dynamics. Specifically, the computation for sea-ice rheology and velocities using the EVP framework. Potential optimization via vectorization is then identified by the Intel compiler v18.0.0 on the HPC platform. Following these results, a vectorization method is designed specifically to optimize the computation for shear strain rates. Implementation of the vectorization method is reserved for future work.

Contents

List of variables

VARIABLES RELATED TO NEMO OCEAN MODEL

T	Center point of Arakawa C grid
f	Corner points of Arakawa C grid
(u,v,w)	Velocity points on Arakawa C grid
u	Velocity component in the i-direction
v	Velocity component in the j-direction
w	Velocity component in the k-direction
$jpkdta$	Number of grid points in the k-direction of the global domain (Discretized levels of ocean's depth)
$jpiglo$	Number of grid points in the i-direction of the global domain
$jpjglo$	Number of grid points in the j-direction of the global domain
jpi	Number of grid points in the i-axis of the local domain
jpj	Number of grid points in the j-axis of the local domain
jpk	Number of grid points in the k-axis of the local domain ($jpk = jpkdta$) in local domains
$(nimpp, njmpp)$	Global position of a local domain's (1,1) grid point
$jpreci$, $jprecj$	Number of rows and columns to be exchanged
$jpni$, $jpnj$	Number of processors dedicated along the i- and j-axes
T_l	Local domain element
T_g	Global domain element

SEA-ICE DYNAMICS VARIABLES

m	Sea-ice mass per unit area
A	Sea-ice concentration
τ_a	Air-ice stress
τ_w	Ocean-ice stress
$-mf(\boldsymbol{k} \times \boldsymbol{u})$	Coriolis force

$-mg\nabla\eta$	Pressure force due to horizontal sea surface tilt
σ	Sea-ice internal stress tensor
$\nabla \cdot \sigma$	Sea-ice internal forces arising in response to deformation
σ_{11}, σ_{22}, σ_{12}	Components of σ
σ_1	Compressive strength: $\sigma_{11} + \sigma_{22}$
σ_2	$\sigma_{11} - \sigma_{22}$
D_D, D_T, D_S	Divergence, horizontal tension and shearing strain rates
ξ_1, ξ_2	Generalized orthogonal coordinates
h_1, h_2	Associated scale factors of the above
e	Ratio of principal axes of elliptical yield curve
Δ	Measure of deformation rate (Invariant)
P	Sea-ice compressive strength
σ_s	Shearing stress of sea-ice: $\sqrt{\sigma_2^2 + 4\sigma_{12}^2}$
h	Sea-ice thickness per unit area
P	Empirical constants of sea-ice compressive strength
C	Empirical constants of sea ice compressive strength
T	Time scale controlling the damping rate of elastic waves
$\sigma_{*,t}$	Time derivative of σ_*
F_1, F_2	Sea-ice internal stress force components w.r.t. to the generalized coordinates

DISCRETIZED VARIABLES FOR SEA-ICE DYNAMICS

i, j	Grid point indices
Δt	Sea-ice time step
$\{\cdot\}^k_{(i,j)}$	Variable $\{\cdot\}$ at grid point (i, j) at kth time step
u	Sea-ice velocity at **u** points
v	Sea-ice velocity at **v** points
$\tau_{a,1or2}$	Air-ice stress at generalized coordinate ξ_1 or ξ_2
c_D	Ice-ocean drag coefficient
ρ_0	Reference density of seawater
u_o	Surface oceanic current defined to be (u_o, v_o)
δ	0 for odd iterations and 1 for even iterations
$\{\cdot\}_{u,v}$	Variable $\{\cdot\}$ defined on or interpolated onto **u** and **v** points

Introduction 1

This thesis aims to profile the performance of the atmosphere-ocean global circulation core model in the Earth system model EC-Earth3, in high resolution configuration, on the Cray machine CRAY XC50 at the Danish Meteorological Institute (DMI). Using available tools on the high performance computing (HPC) platform, areas of potential optimization within the model will be investigated for possible implementation in the future, which may lead to better simulation efficiency.

1.1 What is an Earth System Model?

To better our understanding of how Earth's climate system evolves over time, all physical and biogeochemistry processes must be taken into account. An Earth system model (ESM) is a state-of-the-art tool which tries to simulate these relevant aspects of Earth ("World Climate Research Programme Strategic Plan 2019-2028" 2019). It provides climate physicists the necessary tools to understand climate feedback, attribute changes to specific drivers, and make projections for the future (*IPCC Sixth Assessment Report* 2021, Eyring *et al.*, 2016, Döscher *et al.*, 2021). All of these, in turn, support international development policies such as the Paris Accord, which calls for limiting rise of global temperature to "well below 2°C" and pursuing "the effort to limit the increase to 1.5°C" (UNFCCC, 2015).

The core of an ESM model is the atmosphere-ocean global circulation model (AOGCM), which simulate the dynamic aspects of atmospheric and oceanic processes. They are modeled using the hydrostatic primitive equations (Temam and Zaine, 2005). These equations are then solved numerically on supercomputers, in which the equations are discretized spatially and temporally.

In general, ESMs are known for being one of the most computationally intensive scientific challenges (Flato, 2011). Extremely long integration is needed to extract signals of climate change because the underlying climate system is physically characterized by sensitive dependence and natural stochastic variability (Balaji *et al.*, 2017). Thus, an experiment's simulation period has to be long enough to extract these crucial signals. As the resolution becomes more fine, the global map will be discretized into more number of grid points with smaller time step, which leads to increase in a model's execution time.

More fine spatial and temporal resolution leads to greater fidelity in a model's results. When simulating at finer granulated resolution, small-scale transient processes that play a crucial role in energy and momentum transfer can be better accounted for. In the atmosphere, fine resolution provides answers to climate sensitivity questions by giving better representation of probability distribution associated with climatology of certain weather regimes (Haarsma *et al.*, 2016, Dawson *et al.*, 2012). In the ocean, increase in resolution is particularly crucial towards improvement in prediction skill by resolving fine scale features of boundary currents and simulated mesoscale eddies[1] (Hewitt *et al.*, 2017). In coarse resolution simulation, these processes are not resolved and need to be parameterized. Hence, the fidelity of a climate model's results is highly dependent on the resolution scale at which simulations are performed.

The computing resources consumed by ESMs can be categorized into three groups. Complexity (to account for all feedback internal to the climate system) and resolution (to capture small scale and transient processes) are already discussed. The last one is the ensemble size to sample uncertainty across the chaotic nonlinear dynamics that underlie climate systems. The performance of an ESM would then have to be measured in regards to these three categories which means that traditional measurement of computing power such as FLOPS (floating point operations per second) would be insufficient (Balaji, 2015). Balaji *et al.* (2017) proposed a new metric system for measuring ESMs' performance called the computational performance model intercomparison project (CPMIP) metrics, designed to address issues specifically related to ESMs.[2]

Following this metric system, performance efficiencies of ESMs such as EC-Earth3 have been studied (Döscher *et al.*, 2021) and used for further opti-

[1]Appears as swirls of fluid. Transient and small scale processes that are important to be accounted for in nature as they are responsible for momentum and energy transfer in the climate system. Due to the chaotic nature of turbulence, it is difficult to resolve eddies of all spatial scales in a numerical model.

[2]A complete list of these issues can also be found in the same paper, Balaji *et al.* (2017).

mization. Haarsma *et al.* (2020) investigated the scalability and bottlenecks of the high resolution configuration of EC-Earth3, which will be referred to as EC-Earth3-HR for the rest of the thesis, on the MareNostrum4 computer at the Barcelona Supercomputing Center (BSC). Specifically, they studied the scalability of EC-Earth3-HR AOGCM core model. A few optimizations they have implemented are dedicated to enhance data interpolation and exchange between the atmosphere and ocean components. Others include determining optimal load balance, finding the optimal domain decomposition, and programming I/O in ocean model to run in parallel with the experiment.

However, these implemented optimizations are not specific to the HPC hardware, but general features of EC-Earth3-HR that will allow it to run on some type of parallelization regardless of the HPC hardware. To exploit the full capability of EC-Earth3-HR on DMI's Cray system, its performance has to be further analyzed in order to tailor optimization methods that are appropriate for this specific HPC platform.

With these motivations introduced, an investigation towards optimization of EC-Earth3-HR AOGCM configuration and its component models on DMI's CRAY XC50 is carried out in this thesis project.

1.2 Overview

Chapter 2 gives the rationale for this explorative thesis to investigate potential aspects of optimizing EC-Earth3-HR on DMI's HPC. General optimization that are implemented on EC-Earth3-HR by Haarsma *et al.* (2020) and Tintó Prims *et al.* (2019) are summarized to give inspiration to the investigative approach this thesis takes on. The research questions and respective hypotheses are then stated.

Chapter 3 presents the methodolgy of this investigation. Concepts relevant for performing scalability and performance analysis are described. Data and loop dependencies that must be considered for implementation of low level optimization methods, such as vectorization, are also presented.

Chapter 4 describes the EC-Earth3 model. Description of each model configuration: AOGCM and its component models (atmosphere and ocean) are given along with details of available resolution configuration. This is followed by an in-depth presentation of the EVP framework used to solve for sea-ice rheology and velocities in the sea-ice model. It also contains a succinct description of the parallelism employed within the ocean model.

The detailed experiment designed to investigate potential optimization approach is given in Chapter 5. Here, the models' simulation parameters used for each simulation/experiment are also described.

Chapter 6 presents the results from the investigation of each model's scalability and profiling of the model's efficiency. Results which require further investigation are discussed in Chapter 7. These include: minimum amount of processors required for EC-Earth3-HR AOGCM to complete a simulation successfully; inconclusive results on optimal load balance due to incompatibility between performance analysis tool CrayPat and the ocean model; profiling of the sea-ice subroutine `limrhg`; and, tackling the possibly false "dependency" detected by the Intel compiler v18.0.0 on CRAY XC50.

Chapter 8 summarizes the findings of this thesis and recap the importance of optimizing Earth system models to achieve satisfactory efficiency. This is followed by suggestions of implementing vectorization method, studied in this thesis, to test for speedup in future work.

Chapter 9 is the appendix where Fortran90 script of the subroutine `limrhg` is presented along with the shortened version of the optimization report returned by the compiler. Additional tables and plots can also be found here.

2 Rationale & Research Questions

In this chapter, a revisit of Haarsma *et al.* (2020)'s work will be described to introduce the method that they have used for general optimization of EC-Earth3-HR, which will inspire my investigation aiming to optimize the model's efficiency on the current HPC platform at the Danish Meoterological Institute (DMI). In turn, this will introduce the research questions to be answered in this thesis. A brief background on the environment of DMI's HPC Cray machine is also given in between to understand the capacity of this supercomputer.

In general, EC-Earth3-HR has been optimized with respect to scalability, performance, data storage and post-processing by Haarsma *et al.* (2020). These implementation, however, is not specific to the HPC platform and does not utilize the advantages that a specific platform's hardware offers. Haarsma *et al.* (2020) began the investigation to optimal load balance by studying the scalability of EC-Earth3-HR AOGCM and its components on the HPC platform at the Barcelona Supercomputing Center. They have found that the ocean component is performing less efficiently than the atmosphere component. Bottlenecks were also diagnosed by using performance analysis tools that were developed at the Computer Science Department of the Barcelona Supercomputing Center (Tintó Prims *et al.*, 2019). Some of the bottlenecks are Meassage Passing Interface (MPI) communication, expensive cost of atmosphere output and non-optimized domain decomposition in NEMO, among others.(Haarsma *et al.*, 2020). General optimizations addressing these issues were then implemented and tested on other HPC platforms (Tintó Prims *et al.*, 2019, Haarsma *et al.*, 2020).

Motivated by the work of Tintó Prims *et al.* (2019) and Haarsma *et al.* (2020), this explorative thesis aims to investigate potential optimization of EC-Earth3-HR AOGCM on the HPC platform CRAY XC50 at DMI via the following approach:

- determining the optimal load balance by studying the scalability of the model and its components, and

- using performance analysis tool available on the Cray machine to diagnose bottlenecks within the model

On DMI's HPC platform, two resolution configurations are available for AOGCM simulations. In the first configuration, referred to as the standard resolution configuration, the atmosphere and ocean components have a resolution of ~ 80 km and $1.0°$, respectively. In the second option, atmosphere and ocean components have a resolution of ~ 40 km and $0.25°$, respectively.[1] This is referred as the high resolution configuration or EC-Earth3-HR to be concise.

All experiments are executed on the supercomputer CRAY XC50. It consists of two identical clusters, one of which is available for scientific research and development. One cluster has 152 nodes dedicated for high performance computing jobs. One node has 36 computer processing unit (CPU), specifically the Intel(R) Xeon(R) CPU E5-2695, and 64 GB of shared memory. Each CPU has four level of cache which totals to 46,400 kilobytes.

With all that stated, the following questions are addressed in thesis project by making deductions from EC-Earth3-HR simulations on CRAY XC50:

- What is the optimal load balance for EC-Earth3-HR AOGCM? Between the atmosphere and ocean component, which is performing worse?

- What are the bottlenecks in EC-Earth3-HR AOGCM and the least optimal component?

- What are some low level optimizations that can be safely implemented, without affecting currently implemented interaction between components that may lead to failed or inaccurate experiments?

Hypotheses
Given that this was already determined in Haarsma *et al.* (2020), the ocean component will observe to be performing worse than the atmosphere component in EC-Earth3-HR AOGCM via scalability analysis. The atmosphere

[1]More information about the resolution configuration is available in Chapter 4.

component should perform relatively better given that it is adopted from a weather prediction model used, maintained, and governed by the intergovernmental organization European Center for Medium-Range Weather Forecast (ECMWF); whereas the ocean model is developed and governed by a consortium of five institutes[2].

For EC-Earth3-HR AOGCM, overhead from communications between processors should contribute significantly to the model's decrease in efficiency. As the resolution increases, so does the number of grid points that span across the mesh that makes up the global map. This would require more computer resources for intensive calculation to be carried out on additional number of grid points. Using more processors will require more communications for data exchanges. This general bottleneck may also be applied to the ocean model.

Another bottleneck specific toward ocean model may stem from subroutines dedicated to handling boundary conditions. As the number of processors increase, the global domain will be divided into higher number of local domains. These local domains might not be optimal for handling boundary conditions. For instance, grid points representing a river mouth could possibly be separated into different local domains, which will require additional MPI communication and execution time to handle these cumbersome aspects of an Earth system model.

Last but not least, intensive computation is always a factor to be considered in poor efficiency. With limited knowledge on Cray XC50 at DMI[3] and layout of EC-Earth3-HR, it is conjectured that intensive computation would also contribute significantly to the ocean model's execution. An attainable approach that can potentially optimize the subroutine's computation is using low level optimization such as vectorization.

The methodology and experiment set-up designed to answer these research questions are detailed in Chapter 3 and 5.

[2]Centro Euro-Mediterraneo Sui Cambiamenti Climatici (CMCC), Centre National De La Recherche Scientifique (CNRS-INSU), Mercator Ocean International, Met Office, Natural Environmental Research Council (NERC-NOC)

[3]Detailed understanding of the HPC platform is outside of the scope of this thesis

Methodology

3

The brief methodology described in this chapter is inspired by the one proposed in Tintó Prims *et al.* (2019). It begins by studying the scalability of EC-Earth3-HR AOGCM and its components to understand how fast they can perform with different number of resources. A widely used metric, in the CPMIP metric system for Earth system model (Balaji *et al.*, 2017), is the Simulated Years per Day (SYPD). This metric indicates the number of simulation years that can be completed in one day of wall clock time. To identify the code regions that do not scale properly with increasing number of resources, scalabilities in terms of CPU and wall clock time are also studied for the subroutines of a model. Other bottlenecks such as overhead from MPI communication is determined via performance tool available on DMI's Cray HPC system. Once the problematic code regions are identified, potential low level optimization is investigated by the compiler. According to results of this investigation, different optimization methods are considered after data and loop dependency analyses are carried out.

Below, the general concepts of several components that are necessary in the methodology are introduced. These include scalability analysis in terms of wall clock and CPU time; various dependencies that would prevent low level optimization such as vectorization from being implemented; and available tools on DMI's CRAY XC50 for optimization and performance analysis.

3.1 Scalability analysis

Ideally, the speedup S of a model scales linearly with the number of processors that it is given. For example, the model's execution time reduces by a factor of two when one doubles the amount of computer resources assigned to the model. It can be easily computed with the following expression

$$S = \frac{t_s}{t_N} \tag{3.1}$$

where t_s is the measured model's execution time when it's running sequentially (using one processor) and t_N is measured execution time when it's running in parallel with N processors. In practice, the number of processors and speedup does not scale linearly due to built-up of overhead from communications between processors and some necessary synchronization between model's components.

The execution time of a model can also be measured in terms of wall clock and CPU time. Wall clock time is the time measured between the beginning of a process's execution and at any point of the execution. For instance, one can measure the wall clock time of a specific computation within the model and the wall clock of the model. The wall clock time includes the time when processors are idle while the model is running. CPU time, on the other hand, measures the amount of time a process is actively running, not the time when a process is suspended. Both wall clock and CPU time are measured in seconds.

In this thesis, the scalability of the model is studied in terms of SYPD and wall clock time. Scalabilities of a model's subroutines are measured in both wall clock and CPU time. The wall clock time is measured using `MPI_Wtime()` from the Message Parallel Interface (MPI) available on the Cray system, and the CPU time is measured using the Fortran90 function `CPU_TIME()`.

Instead of measuring speedup or scalability with respect to execution time from running the model sequentially, it is measured with respect to the optimal load balance of EC-Earth3 in the standard resolution configuration on CRAY XC50. Since the models in standard resolution have an optimal performance using N processors, then the high resolution configuration should require at least N processors to carry out simulations of the same computation but with a bigger problem size due to finer granulated resolution.

3.2 Parallelization of loops via vectorization

Two kinds of parallelism can be implemented on a program: task parallelism and data parallelism. The first allows the different operations within a program to perform simultaneously. These operations do not necessarily have to work on different pieces of the same data set. The latter allows the same set of operations to perform simultaneously on different pieces of the same data set.

A low level optimization of data parallelism is vectorization. Vectorization is the process of rewriting a loop so that multiple pieces of data in an array are processed simultaneously instead of processing one element at a time. Today, high performance computing (HPC) systems have compilers that can implement auto-vectorization of loops using compiling options such as `-guide-vec[=<level>`, provided by the Intel compiler (Intel (2017)). This auto-vectorization instructs the compiler to analyze loops in the code with operations that can be executed in parallel and transforms them into vector operations. The compiler can also return reports pinpoint where optimization is applicable or not in the loops. This is done by using the compiling option `-qopt-report=<level> -qopt-report-phase=<option>`. These information are extremely helpful in identifying potential potential methods that can be implemented for optimizing performance of a subroutine.

3.2.1 Dependencies

To implement vectorization, one must identify dependencies in operations on data that prevent such optimization. Below are brief introduction to several types of such dependencies.

FLOW dependency: When an instruction depends on the results from the previous iteration, this leads to a "read-after-write" (RAW) dependency between the current and previous variables. This also shows loop-carried dependency in which the current iteration result depends on the previous iteration result.

```
DO i = 1, N
    b[i] = b[i-1] + c[i]
END DO
```

ANTI dependency: When an instruction depends on a result that is later updated in the next iteration, this leads to a "write-after-read" (WAR) dependency between the current and future variables.

```
DO i = 1, N
    a[i] = a[i+1]
END DO
```

OUTPUT dependency: Also known as "write-after-write" (WAW) dependency, in which data at the same memory address is written to twice by two instructions, one after another.

```
DO i = 1, N
    d[i] = i
    d[i+1] = 3
END DO
```

Control dependency: These can typically be introduced using an if-statement, which can disrupt the flow of an iteration. Using the example in OUTPUT dependency, a write-after-write dependency can also be introduced using an if-statement that specifies a condition for `d[i]` to be written to again.

Cross iteration dependency: Given a set of instructions where array `A` has to be computed in a (nested-)loop first in order to compute array `C` in another (nested-)loop. Then a dependency is introduced between the two loops, known as cross iteration dependency.

3.3 Performance analysis tool: CrayPat

Performance tool such as the Cray Performance Measurement and Analysis Tool (CrayPat) is a powerful software, available on CRAY XC50, that enables users to analyze how well a parallelized program is performing on a Cray supercomputer, and how to optimize it further (Hewlett-Packard, n.d.).

It details information on timing and performance of individual application procedures. From these information, one can determine the top time consuming routines, load balance across processes and threads, parallel overhead, etc. (ECMWF, n.d.). It also directly incorporates information from the hardware performance counters available on the Intel Xeon processors (ECMWF, 2015), the same processors used in DMI's Cray machine.

By instrumenting CrayPat with EC-Earth3-HR AOGCM, one can then analyze the current performance of the model and identify areas of bottlenecks. For example, if parallel overhead is reported to be the cause of the model's unsatisfactory efficiency, then one should investigate how to optimize the current methods implemented for communication between processors.

Details on how the software can be instrumented to programs are not provided in this thesis as it is well-documented in other manuals such as ECMWF (2015), Hewlett-Packard (n.d.) and ECMWF (n.d.).

4 Climate Model

EC-Earth is a global coupled climate model (Hazeleger *et al.*, 2012) that has been developed by a consortium of 27 European research institutes. It combines and couples existing models that describe the atmosphere, ocean, sea ice, land surface, dynamic vegetation, atmospheric composition, ocean biogeochemistry and the Greenland ice sheet, to give a state-of-the-art tool for simulating climate evolution. EC-Earth3, the third version of EC-Earth, has been developed in preparation for the Coupled Model Intercomparison Project phase 6 (CMIP6) (Eyring *et al.*, 2016). The specific version of EC-Earth3 that is studied in this thesis is the published EC-Earth3-v3.3.3.2 high resolution configuration model.

4.1 Model Configuration Option

In this thesis, performance of three model configurations are investigated: the atmosphere-coupled global circulation model (AOGCM), the atmosphere only model, and the ocean only model. The atmosphere and ocean components of the climate model are simulated using the European Centre of Medium-Range Weather Forecasts' Integrated Forecasting System (IFS) and the Nucleus for European Modelling of the Ocean (NEMO), respectively. For the rest of the thesis, these model configurations will simply be referred to as model.

In both standalone models, feedbacks from the missing atmosphere or ocean component are supplied by forcing data sets following the CMIP6 protocol (Eyring *et al.* (2016)). A full description of all model configurations and forcing data are presented in Döscher *et al.*, 2021.

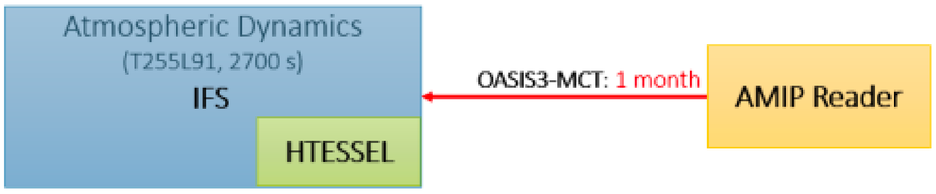


Figure 4.1: The IFS-standalone configuration in EC-Earth3 taken from Ludemann (2022). Within the IFS component, the standard resolution and accompanying time step are shown in blue parentheses. The supplement of oceanic observational data from AMIP to IFS through OASIS3-MCT is delineated by the red arrow.

4.1.1 Atmospheric (IFS) only

The atmospheric component of EC-Earth3 is adopted from IFS CY36R4, part of ECMWF's operational seasonal forecast system Since IFS is designed for the purpose of numerical weather forecast, many modifications (e.g. conservation of mass and energy, sea-ice albedo, time stepping scheme, etc.) have to be made for running long climate simulations or simulations under different climate conditions. It includes HTESSEL, a submodule that handles energy and moisture exchange between land and atmosphere.

In the IFS-only model, feedback that would have been otherwise simulated by the ocean model is replaced by the interface Atmospheric Model Intercomparison Project (AMIP) reader. Forcing data sets of monthly sea surface temperature and sea ice concentration fields from CMIP6 are taken and interpolated into daily fields by AMIP reader (Ludemann, 2022). The results are then sent to IFS for simulation via the OASIS3-MCT coupling library (Craig *et al.*, 2017).

Many resolution configurations are available from ECMWF's IFS, and three options are adopted into EC-Earth3 as listed in Table (4.1). As per to principle in an atmospheric model, these are spectral resolutions with a linear reduced Gaussian grid. The corresponding global horizontal and vertical resolutions are also given for each three configuration.

Resolution	T159L62	T255L91	T511L91
Horizontal (km)	~ 125	~ 80	~ 40
Vertical (levels)	62	91	91
no. of horizontal grid points	35,718	88,838	348,528
Total no. of grid points	2,214,516	8,084,258	31,716,048

Table 4.1: IFS spectral resolution options available on EC-Earth3. Each resolution corresponds to a grid cell that has a specific land area coverage and number of vertical levels that the atmosphere is divided into.

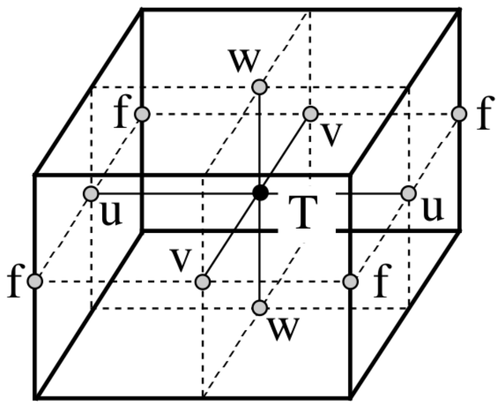


Figure 4.2: Typical variable distribution on an Arakawa C-type grid. **T** indicates where scalar variables are defined. This includes temperature, salinity, density and pressure. **(u,v,w)** indicates the velocity points. Finally, **f** indicates both the relative and planetary vorticities.

4.1.2 Ocean (NEMO) only

The ocean component of EC-Earth3 is modeled using NEMO3.6, developed by consortium of five institutes: Centro Euro-Mediterraneo Sui Cambiamenti Climatici (CMCC), Centre National De La Recherche Scientifique (CNRS-INSU), Mercator Ocean International, Met Office and the Natural Environmental Research Council (NERC-NOC).

It is a framework consisting of three engines: the Océan PArallélisé (OPA), the Louvain-La-Nueve3.6 (LIM3.6), and PISCES. Whereas the first two engines models the dynamics and thermodynamics of ocean and sea ice, the third accounts for the ocean biogeochemistry. These engines exchange data directly via shared data structures. Several combinations of these three engines are available for the EC-Earth3 ocean configuration. In this thesis, we focus on the standard configuration of NEMO which simulates using the ocean and sea-ice engines only.

NEMO simulates the dynamics and thermodynamics of the ocean by solving the primitive hydrostatic equations of ocean circulation using the classic, centered second-order finite difference approximation. The distribution of variables is given by a 3D Arakawa-C-type grid (Arakawa and Lamb, 1977). Whereas scalar variables are defined at the center of each 3D cell, vector variables are defined in the center of each face of the cells. At the center of each vertical edge is defined the relative and planetary vorticities (Madec and NEMO team, 2016). This arrangement is shown in Figure 4.2.

Resolution	ORCA1L75	ORCA025L75
Horizontal (°)	1	0.25
Vertical ($jpkdta$)	75 levels	75 levels
$jpiglo$	362	1,442
$jpjglo$	292	1,050
$jpiglo \times jpjglo$	105,704	1,514,100
$jpiglo \times jpjglo \times jpkdta$	7,927,800	113,557,500

Table 4.2: NEMO resolution configurations available in EC-Earth3. Each grid cell in NEMO's global ocean tripolar grid can cover either $1 \times 1°^2$ or $0.25 \times 0.25°^2$ area. The depth of the ocean is divided into 75 levels for both configuration. The number of grid points along the three axes are given by $jpiglo$ (i-axis), $jpjglo$ (j-axis) and $jpkdta$ (k-axis). These variables also represent the number of grid points spanning across the global domain.

Unlike IFS-only where the ocean's forcing data sets are supplied by OASIS3-MCT, atmospheric forcing data sets are handled by NEMO's internal modules. The forcing data fields, not necessarily in model grid type, are then interpolated onto NEMO model grid via either bicubic or bilinear interpolation, depending on the variables.

At the sea ice-ocean interface, heat, salinity, fresh water and momentum are exchanged and directly evaluated within the NEMO model. At this boundary, the sea surface temperature is constrained to be at the freezing point, and sea ice salinity is restricted to $\sim 4 - 6psu$ compared to the ocean's salinity of $\sim 34psu$. The boundary conditions at this interface are updated every (ocean/sea ice) time step to compute for the heat, salt, momentum and freshwater fluxes. Simultaneously, ocean surface stresses due to sea ice is also re-evaluated within a sea ice module.

In EC-Earth3, the NEMO-standalone configuration has two resolution configurations available, where the ocean and sea ice time steps are the same in each of them. From Table (4.2), the options, with corresponding horizontal and

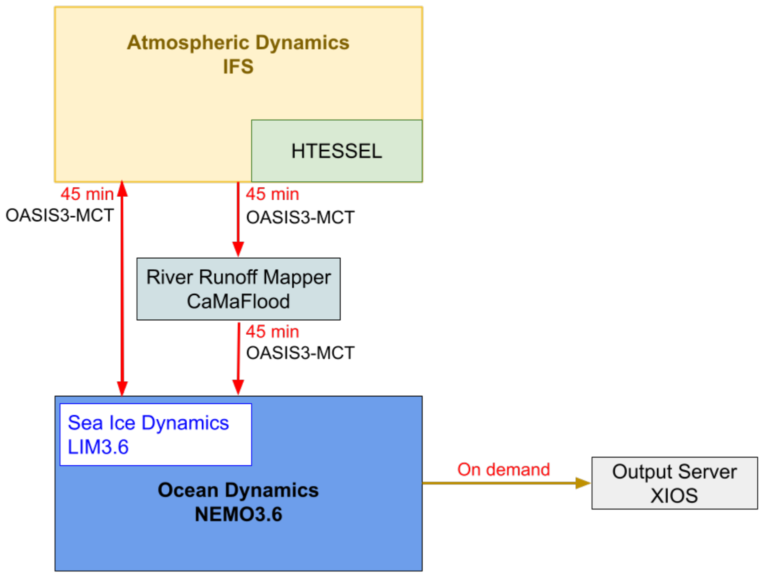


Figure 4.3: The IFS-NEMO coupled model configuration in EC-Earth3. This basic atmospheric-ocean model also includes the HTESSEL module and sea ice engine. Exchanges between land and ocean are handled by the runoff mapper.

vertical resolutions, are given in terms of degrees in a tripolar grid (Madec and Imbard (1996)) and levels, respectively.

Lastly, the input/output (I/O) of data in NEMO is handled by XML Input/Ouput Server (XIOS), developed by Yann Meurdesoif from IPSL. It is an asynchronous Message Passing Interface (MPI) I/O server designed for handling climate data files. In EC-Earth3 models involving the ocean, XIOS can be configured to run as a "detached" server (an external executable with some number of CPUs assigned to it) or as an "attached" library of the NEMO model (Madec and NEMO team, 2016) On CRAY XC50, XIOS is generally used as a detached external server to optimize I/O in simulations.

4.1.3 Atmosphere-ocean global circulation model (AOGCM)

The AOGCM model is the standard configuration offered by EC-Earth3 and the two components, IFS and NEMO, are coupled via the OASIS3-MCT coupling library. This coupling library ensures conservation of momentum, energy, evaporation and precipitation fluxes along with conservative remapping in this atmosphere-ocean environment (Döscher *et al.*, 2021). A configuration of the coupled model is presented in Figure (4.3).

The atmosphere-ocean interface is defined such that the ocean provides state variables to the atmospheric model whilst the atmosphere sends fluxes to the ocean. The fluxes sent from the atmosphere are computed following the formulations provided in the IFS CY36R1 documentation (ECMWF, 2010). Exchanges between the atmosphere and ocean occur every 10,800 seconds for low resolution and every 2700 seconds for both standard and high resolution configuration.

Meanwhile, exchange of freshwater runoff from land to ocean is handled by a runoff mapper that interpolate runoff and calving, from the atmosphere and surface model HTESSEL, onto drainage basins on a mapper grid and distributed along the ocean coastal points (Döscher *et al.*, 2021).[1]

From the available resolutions in IFS-standalone and NEMO-standalone models, three resolution configurations are created for coupled model simulations, as shown in Table (4.3). The time steps of each model (and sea ice sub-model) for the three different configurations are also listed along with the coupling frequency, which denotes how often data are exchanged between the atmosphere and ocean. For a low resolution configuration, data are exchanged every second IFS time step or every third NEMO time step. In the standard resolution configuration, data are exchanged every (IFS/NEMO/LIM3.6) time step. Finally, data are exchanged every third time step in the high resolution configuration.

4.2 Sea ice component: LIM3.6

As mentioned in the previous section, LIM3.6 is an integrated engine in NEMO that models the dynamics and thermodynamics of sea ice. It is based on the Artic Ice Dynamics Joint Experiment framework which accounts for variation in ice thickness using a distribution function; conservation of horizontal

[1]For details of variables and fluxes exchanged between the different components in the coupled model, refer to Tables 3 and 4 in Döscher *et al.* (2021).

Resolution option	Low resolution	Standard resolution	High resolution
IFS	T159L62	T255L91	T511L91
NEMO	ORCA1L75	ORCA1L75	ORCA025L75
IFS time step (s)	3,600	2,700	900
NEMO time step (s)	2,700	2,700	900
LIM3.6 time step (s)	2,700	2,700	900
Coupling frequency (s)	10,800	2,700	2,700

Table 4.3: Resolution configurations available on EC-Earth3 with corresponding time step. For standard and high resolution, the time steps are the same among all components. The couple frequency tells the model how often data is exchanged between the atmosphere, ocean and sea-ice components. At low resolution, data is exchanged every third NEMO time step. At standard resolution, data is exchanged every time step. At high resolution, data is also expressed every third time step.

momentum; energy-conservation halo-thermodynamics; and ice rheology by assuming that it's an elastic-viscous plastic (EVP) (Rousset *et al.* (2015)).

The rest of this section is dedicated to describe the EVP framework for calculating sea-ice deformation term $\Delta \cdot \sigma$. This algorithm is found to be a signficant run-time contributor in the least scalable subroutine of NEMO[2]

4.2.1 Sea Ice Dynamics

Like the ocean engine in NEMO, LIM3.6 uses an Arakawa C grid for variable distribution and models the following horizontal momentum equation for sea ice dynamics:

$$m\frac{\partial \boldsymbol{u}}{\partial t} = A(\tau_a + \tau_w) - mf(\boldsymbol{k} \times \boldsymbol{u}) - mg\nabla\eta + \nabla \cdot \sigma \tag{4.1}$$

where m is the ice mass per unit area, A is sea-ice concentration, τ_a and τ_w are the air-ice and ocean-ice stresses, $-mf(\boldsymbol{k} \times \boldsymbol{u})$ is the Coriolis force, $-mg\nabla\eta$ is the pressure force due to horizontal sea surface tilt, and $\nabla \cdot \sigma$ is the sea-ice internal forces arising in response to deformation. By assuming sea ice to be an EVP material, this equation can be solved explicitly in time in Arakawa C-grid. This regularizes the original viscous plastic (VP) formulation (Hibler, 1979) that is solved implicitly and assumes that sea ice's resistance to deformation depends on its instantaneous state of motion and several large-scale scalar properties such as ice thickness and lead fractional area (Bouillon *et al.*, 2009).

[2]To be discussed from results presented in Chapter 6.

The assumption of either EVP or VP leads to different formulations for the internal forces $\nabla \cdot \sigma$ (the last term in Eq. (4.1)). In the next section, the key elements of the EVP framework is highlighted. Descriptions of the general framework of either formulation are presented in Hunke and Dukowicz (2002) and Hunke and Lipscomb (2006).

4.2.2 EVP framework

To begin with the calculation of the sea ice internal forces, the internal stress tensor σ can be further decomposed into σ_{11}, σ_{22} and σ_{12}, and defined by the following:

$$\begin{aligned} \sigma_1 &= \sigma_{11} + \sigma_{22} \\ \sigma_2 &= \sigma_{11} - \sigma_{22} \\ D_D &= \frac{1}{h_1 h_2}\left(\frac{\partial}{\partial \xi_1}(h_2 u) + \frac{\partial}{\partial \xi_2}(h_1 v)\right) \\ D_T &= \frac{1}{h_1 h_2}(h_2^2 \frac{\partial}{\partial \xi_1}(u/h_2) - h_1^2 \frac{\partial}{\partial \xi_2}(v/h_1)) \\ D_S &= \frac{1}{h_1 h_2}(h_2^2 \frac{\partial}{\partial \xi_1}(v/h_2) + h_1^2 \frac{\partial}{\partial \xi_2}(u/h_1)) \end{aligned} \tag{4.2}$$

where D_D, D_T and D_S are divergence, horizontal tension and shearing strain rates, respectively; ξ_1 and ξ_2 are generalized orthogonal coordinates; and h_1 and h_2 are the associated scale factors. The internal stress tensor components can then by rewritten in these terms to be:

$$\begin{aligned} \sigma_1 &= \left(\frac{D_D}{\Delta} - 1\right) P \\ \sigma_2 &= \left(\frac{D_T}{e^2 \Delta}\right) P \\ \sigma_{12} &= \left(\frac{D_S}{2e^2 \Delta}\right) P \end{aligned} \tag{4.3}$$

where P is the ice compressive strength, e is the ratio of principal axes of the elliptical yield curve, and the invariant Δ is a measure of the deformation rate defined as followed:

$$\Delta = \sqrt{D_D^2 + \frac{1}{e^2}\left(D_T^2 + D_S^2\right)} \tag{4.4}$$

Whereas σ_1 represents the compressive stress, σ_2 and σ_{12} together give the shearing stress σ_s such that $\sigma_s = \sqrt{\sigma_2^2 + 4\sigma_{12}^2}$. The rheology of sea ice deformation is thus given by these terms and expressed in the form of an elliptical yield curve

$$1 = \left(\frac{\sigma_1}{P} + 1\right)^2 + e^2 \left(\frac{\sigma_s}{P}\right)^2 \tag{4.5}$$

The ice compressive strength P is empirically related to the ice thickness per unit area h and ice concentration A by the relation $P = \boldsymbol{P} h e^{-C(1-A)}$. ($\boldsymbol{P}$ and C are empirical constants.)

As Δ approaches zero, regularization is needed for numerical solution and the method proposed by Hunke and Dukowicz (1997) gives the following EVP formulation which introduces time dependence and an artificial elastic term:

$$\begin{aligned} 2T\sigma_{1,t} + \sigma_1 &= \left(\frac{D_D}{\Delta} - 1\right) P \\ \frac{2T}{e^2}\sigma_{2,t} + \sigma_2 &= \frac{D_T}{e^2\Delta} P \\ \frac{2T}{e^2}\sigma_{12,t} + \sigma_{12} &= \frac{D_S}{2e^2\Delta} P \end{aligned} \tag{4.6}$$

In this formulation, T is a time scale that controls damping rate of elastic waves. Based on this framework, the components of the internal forces $\nabla \cdot \sigma$ are (Hunke and Dukowicz, 2002):

$$\begin{aligned} 2F_1 &= \frac{1}{h_1}\frac{\partial \sigma_1}{\partial \xi_1} + \frac{1}{h_1 h_2^2}\frac{\partial (h_2^2 \sigma_2)}{\partial \xi_1} + \frac{2}{h_1^2 h_2}\frac{\partial (h_1^2 \sigma_{12})}{\partial \xi_2} \\ 2F_2 &= \frac{1}{h_2}\frac{\partial \sigma_1}{\partial \xi_2} - \frac{1}{h_1^2 h_2}\frac{\partial (h_1^2 \sigma_2}{\partial \xi_2} + \frac{2}{h_1 h_2^2}\frac{\partial (h_2^2 \sigma_{12}}{\partial \xi_1} \end{aligned} \tag{4.7}$$

4.2.3 EVP framework: discretized

A type of centered finite difference method is used for the sea ice dynamics with a twist of predictor-corrector scheme. Starting from a solution at time t, an intermediate solution is determined at time $t + \Delta t/2$. Then, the solution at the next full time step $t + \Delta t$ is determined using the internal stress's non-linear terms and ice-ocean stress terms centered at $t + \Delta t/2$. This scheme is iterated n times with a sub-cycling time step $\Delta t/n$ to improve accuracy of results.

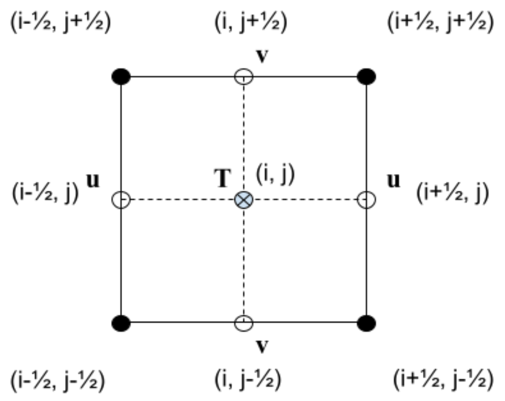


Figure 4.4: 2D Arakawa C grid taken from the middle plane of the 3D grid cell in Figure (4.2). The indices of each points on the grid cell represented by (i, j). The center of the cell is denoted by $\otimes$ where the components D_D, D_T, σ_1 and σ_2 are located, whilst D_S and σ_{12} are located at the corners represented by solid-filled circles. Lastly, deformation rate Δ is calculated at both the center and corners.

An example of Arakawa C grid is already given in Figure. (4.2). Consider now, only the middle plane where the variables **T** and **(u,v)** reside, as shown in Figure (4.4). Whilst the components D_S and σ_{12} are defined at the corners, D_D, D_T, σ_1 and σ_2 are all defined at the cell centers with **T**. The internal stress force components F_1 and F_2 are defined on the **u** and **v** points, respectively, which are indicated by the open circles. Since the deformation rate Δ (in Eq. (4.3)) is needed to determine components of internal stress tensor σ, and it is determined from the three strain rates ($D_{D,T,S}$) as defined in Eq. (4.4), this means that strain rates not defined on other grid points have to be interpolated from cell centers to the corners or vice-versa.

Here, we begin the presentation for the discretized versions of the above continuous expressions. For details, please refer to Hunke and Dukowicz (2002). Defining grid elements $e_1 = h_1\Delta\xi_1$ and $e_2 = h_2\Delta\xi_2$, such that $\Delta\xi_1$ and $\Delta\xi_2$ are the spatial steps in the two orthogonal directions, the components of the strain rates introduced in Eq. (4.2) are discretized and given by the following:

$$\begin{aligned} e_{1(i,j)}e_{2(i,j)}D_{D(i,j)} &= e_{2(i+1/2,j)}u_{(i+1/2,j)} - e_{2(i-1/2,j)}u_{(i-1/2,j)} \\ &+ e_{1(i,j+1/2)}v_{(1,j+1/2)} - e_{1(i,j-1/2)}v_{(i,j-1/2)} \end{aligned} \tag{4.8}$$

$$\begin{aligned} e_{1(i,j)}e_{2(i,j)}D_{T(i,j)} &= e^2_{2(i,j)}\left(\frac{u_{(i+1/2,j)}}{e_{2(i+1/2,j)}} - \frac{u_{(i-1/2,j)}}{e_{2(i-1/2,j)}}\right) \\ &- e^2_{1(i,j)}\left(\frac{v_{(i,j+1/2)}}{e_{1(i,j+1/2)}} - \frac{v_{(i,j-1/2)}}{e_{1(i,j-1/2)}}\right) \end{aligned} \tag{4.9}$$

$$\begin{aligned} &e_{1(i+1/2,j+1/2)}e_{2(i+1/2,j+1/2)}D_{S(i+1/2,j+1/2)} = \\ &\qquad e^2_{1(i+1/2),j+1/2}\left(\frac{u_{(i+1/2,j+1)}}{e_{1(i+1/2,j+1)}} - \frac{u_{i+1/2,j}}{e_{1(i+1/2,j)}}\right) \\ &\qquad + e^2_{2(i+1/2,j+1/2)}\left(\frac{v_{(i+1,j+1/2)}}{e_{2(i+1,j+1/2)}} - \frac{v_{(i,j+1/2)}}{e_{2(i,j+1/2)}}\right) \end{aligned} \tag{4.10}$$

Next, the variables' interpolation onto cell centers and corners are performed. The interpolation expressions can be found in in Bouillon et. al. (2009).

The discretization of Eq.(4.6) follows as thus (Hunke and Lipscomb, 2006):

$$\begin{aligned} 2T\left(\frac{\sigma_1^{k+1} - \sigma_1^k}{\Delta t}\right) + \sigma_1^{k+1} &= \left(\frac{D_D^k}{\Delta^k} - 1\right)P \\ \frac{2T}{e^2}\left(\frac{\sigma_2^{k+1} - \sigma_2^k}{\Delta t}\right) + \sigma_2^{k+1} &= \left(\frac{D_T^k}{e^2\Delta^k}\right)P \\ \frac{2T}{e^2}\left(\frac{\sigma_{12}^{k+1} - \sigma_{12}^k}{\Delta t}\right) + \sigma_{12}^{k+1} &= \left(\frac{D_S^k}{2e^2\Delta^k}\right)P \end{aligned} \tag{4.11}$$

where Δt is the dynamic time step. Here, the superscripts k denotes variables evaluated at times $k\Delta t$ and $k+1$ for $(k+1)\Delta t$. P is the compressive strength and it is updated every thermodynamic and ice transport time step.

Once discretized, the internal stress force expressions in Eq. (4.7) become

$$\begin{aligned} &2e_{1(i+1/2,j)}e_{2(i+1/2,j)}F_{1(i+1/2,j)} = \\ &e_{2(i+1/2,j)}(\sigma_{1(i+1,j)} - \sigma_{1(i,j)}) + \frac{1}{e_{2(i+1/2,j)}}(e^2_{2(i+1,j)}\sigma_{2(i+1,j)} - e^2_{2(i,j)}\sigma_{2(i,j)}) \\ &+ \frac{2}{e_{1(i+1/2,j)}}(e^2_{1(i+1/2,j+1/2)}\sigma_{12(i+1/2,j+1/2)} - e^2_{1(i+1/2,j-1/2)}\sigma_{12(i+1/2,j-1/2)}) \end{aligned} \tag{4.12}$$

$$\begin{aligned} &2e_{1(i,j+1/2)}e_{2(i,j+1/2)}F_{2(i,j+1/2)} = \\ &e_{1(i,j+1/2)}(\sigma_{1(i,j+1)} - \sigma_{1(i,j)}) - \frac{1}{e_{1(i,j+1/2)}}(e^2_{1(i,j+1)}\sigma_{2(i,j+1)} - e^2_{1(i,j)}\sigma_{2(i,j)}) \\ &+ \frac{2}{e_{2(i,j+1/2)}}(e^2_{2(i+1/2,j+1/2)}\sigma_{12(i+1/2,j+1/2)} - e^2_{2(i-1/2,j+1/2)}\sigma_{12(i-1/2,j+1/2)}) \end{aligned} \tag{4.13}$$

Finally, the discretized momentum equation is given by:

$$m_u \left(\frac{u^{k+1} - u^k}{\Delta t} \right) = F_1^{k+1} + A_u \left[\tau_{\sigma 1} + c_D \rho_0 \left| \mathbf{u}_0 - \mathbf{u}^k \right|_u (u_0 - u^{k+1}) \right] + m_u f_u v_u^{k+\delta} - \frac{m_u g}{h_1} \frac{\partial \eta}{\partial \xi_1} \tag{4.14}$$

$$m_v \left(\frac{v^{k+1} - v^k}{\Delta t} \right) = F_2^{k+1} + A_v \left[\tau_{\sigma 2} + c_D \rho_0 \left| \mathbf{u}_0 - \mathbf{u}^k \right|_v (v_0 - v^{k+1}) \right] + m_v f_v u_v^{k+1-\delta} - \frac{m_v g}{h_2} \frac{\partial \eta}{\partial \xi_2} \tag{4.15}$$

where the subscripts u and v denote a variable defined on or interpolated onto the corresponding u and v points, c_D is the ice-ocean drag coefficient, ρ_0 is the reference seawater density, $\mathbf{u}_0 \equiv (u_0, v_0)$ is the surface oceanic current, and δ is either 0 for odd iterations or 1 for even iterations.

The order in which Eq. (4.14) and (4.15) are solved depends on δ. For $\delta = 0$, odd iterations, Eq. (4.14) is solved first. Variable u^{k+1} is then interpolated onto v points and used to solve Eq. (4.15). For $\delta + 1$, even iterations, Eq. (4.15) is solved first and the updated value of v is interpolated onto u points to compute for the Coriolis term $m_u f_u v_u^{k+\delta}$ in Eq. (4.14). This is equivalent to solving the Coriolis term semi-implicitly, which means that this term would require the simultaneous solution of both Eq. (4.14) and (4.15).

4.2.4 Implementation of EVP framework in *limrhg*

A complete script of `limrhg` is available in Appendix (Chapter 9). Here, a brief overview of what is being done in the EVP section is presented.

The EVP algorithm officially begins at L365 and ends at L603 where the entire process is iterated `nn_nevp`120 times to achieve some degree of accuracy and convergence in our solution. At the beginning of every iteration, a "convergence test" is available by setting `ln_ctl=.TRUE.`. Next, the divergence, tension and shearing strain rates defined in Eq. (4.2) are being computed at their designated grid points, as explained in Section 4.2.3. These terms are then used to compute for the sea ice internal stress tensor components σ_1, σ_2 and σ_{12}, which contribute to the sea ice internal stress forces (F_1 and F_2) calculation. This is followed by the interpolation of sea ice velocities (u, v) (defined on U and V points, respectively) onto V and U grid points in preparation for the computation of sea ice velocities (u^{k+1}, v^{k+1}) after one time step at time $(k+1)\Delta t$.

4.3 Implemented parallelism on EC-Earth3

NEMO is built such that data are exchanged between the ocean and sea-ice models via shared data structures. This inspired EC-Earth3 to be implemented centering around the same idea of data exchange by following a multi-executable MPMD (mutliple programs, multiple data) approach. In the AOGCM model, IFS and NEMO run concurrently on the supercomputer and data exchanges are handled by MPI communications from Message Passing Interface (MPI). The atmosphere and ocean models are each given a number of processors, and then each model's global domain is divided into local domains which are then assigned to the processors. In the next section, we detailed the procedure of domain decomposition in the ocean model NEMO.

4.3.1 Domain decomposition in NEMO

The computational domain in NEMO, of course, covers the global ocean, and the total size is given by the parameters $jpiglo$, $jpjglo$ and $jpkdta$. These denote the number of grid/mesh points that span across the i- and j- axes horizontally and k-axis vertically. Domain decomposition is used for massively parallel processing (mpp) by splitting the global domain horizontally whilst preserving the number of grid points vertically. Each processor solves the ocean dynamic primitive equations over its local domain, and compute its own surface and bottom boundary conditions. The local domain boundary conditions are applied via communications between processors that specify how data at the boundaries are handled.

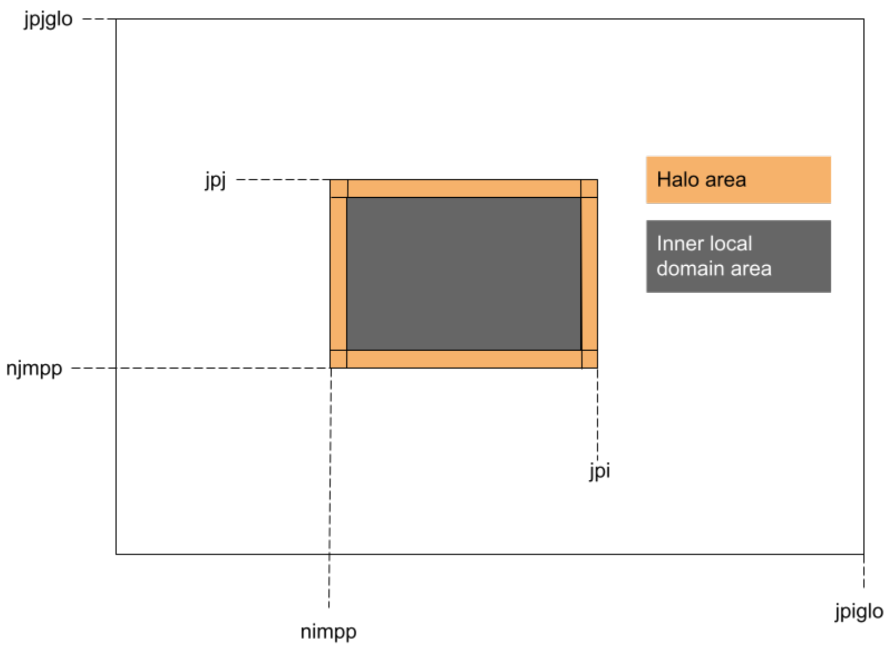


Figure 4.5: Positioning of local domain with respect to the global domain when OpenMPI is used.

The local domain size is given by $jpi \times jpj \times jpk$, with $jpk = jpjdta = 75$, whilst the other two are determined following these expressions:

$$
\begin{aligned}
jpi &= (jpiglo - 2jpreci + (jpni - 1))/jpni + 2jpreci \\
jpj &= (jpjglo - 2jprecj + (jpnj - 1))/jpnj + 2jprecj
\end{aligned}
\tag{4.16}
$$

where $jpni$, $jpnj$ are the number of processors dedicated along the i- and j-axes, and $jprei$, $jprecj$ are number of rows and columns to be exchanged (usually set to 1). The number of processors allocated for the axes are determined for a number of processors $jpnij$ most often equal to $jpni \times jpnj$.

By defining $(nimpp, njmpp)$ to be the global position corresponding to a local domain's grid point $(1, 1)$, we can use the following expression to relate elements in a local domain T_l to their position in the global domain T_g:

$$T_g(i + nimpp - 1, j + njmpp - 1, k) = T_l(i, j, k) \tag{4.17}$$

with $1 \leq i \leq jpi$, $1 \leq j \leq jpj$, and $1 \leq k \leq jpk$. An example of what this would look like is shown in Figure (4.5). At each edge of a local domain,

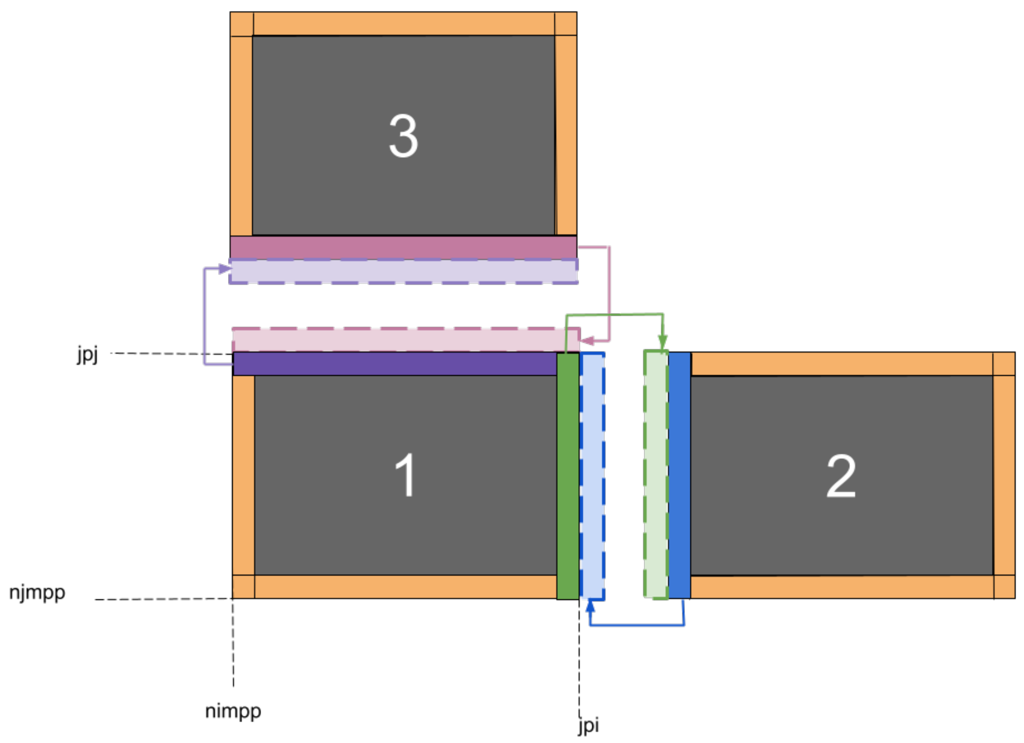


Figure 4.6: Data exchanges between neighboring processors in NEMO to update values at each edges of the local domain.

there is a row/column of data, labeled as "halo area", that is to be exchanged with neighboring processors. Meanwhile, data in the gray "inner local domain area" is not exchanged with exchanged. This halo area is exchanged with neighboring processor so that values at the edges of each local domain can be updated. An example of what this would look like is shown in 4.6. From processor 1, its top row of data (indigo) is exchanged with processor 3 and stored in an array called "ghost zone" so that the processor can continue computations in order to update the values in the bottom row of processor 3 (maroon). The same is done vice versa from processor 3 to processor 1. Exchanges of column data between processor 1 and 2 also follow the same principle.

Experiment Design 5

As a means to study the scalability of EC-Earth3-HR AOGCM on CRAY XC50 and investigate subroutines for potential optimization, simulations using the model configurations described in Chapter 4.1 are set up. Each configuration is run in both standard and high resolution with the same experiment start date on 01/01/1990 and runs for 10 simulation days with no results output. The initial model states of the simulations are the default set-up for EC-Earth3 on CRAY XC50. The ocean states are taken from a long spin-up run using a present day forcing, while the state of the atmosphere is from the ECMWF reanalysis data (ERA-Interim). Simulations are performed 5x or 10x in an attempt to achieve some degree of statistical average to account for the systematic variation within the supercomputer.

Following the methodology presented in Chapter 3, below details the investigation of EC-Earth3-HR AOGCM and its components' performance on CRAY XC50.

5.1 Determining optimal load balance

Before scalability analysis is performed, a comparison of the models' performance efficiency is carried out between the standard and high rseolutoin configuration.

Comparing wall clock run-time & SYPD between EC-Earth-HR and stanard resolution configuration.
The optimal load balance, or optimal number of processors for each model configuration, in the standard resolution is already determined. In the standard resolution configuration, IFS is given 252 CPUs whilst NEMO is given 144 CPUs. The sum of these then gives the optimal number of processors for AOGCM. Using the same numbers of CPUs for all three model configurations in

high resolution, the average wall clock run-times and SYPD are measured get an idea of how EC-Earth3's performance efficiency changes with resolution.

Measuring scalability of IFS-only, NEMO-only & AOGCM in high resolution. The speedup of teh models are determined with respect to the optimal load balance of the standard resolution configuration. Multiples of 252 and 144 CPUs are assigned to the IFS- and NEMO component model. SYPD and wall clock run-times are then measured from these standalone simulation.

The total number of grid points in each component has to be taken into consideration when allocating CPUs for AOGCM simulation. These numbers are presented in Table 4.1 and 4.2. Whilst IFS has a total of 31,716,048 grid points at a resolution of T511L91, NEMO has 113,557,500 grid points at resolution ORCA025L75; that's a factor of ~ 3.6 more than IFS. This allows us to make an initial conservative guess that the number of processors NEMO needs in AOGCM at high resolution might be a factor of 3 more than that of IFS. Trials of running EC-Earth3-HR AOGCM begin by assigning 252 CPUs to IFS and 756 CPUs to NEMO. The ratio of CPUs for AOGCM in high resolution is then adjusted with more experiments performed.

Performance profiling using CRAYPAT
The general idea of CrayPat is already presented in Chapter 3 and instructions on how to instrument it to executable files are given by the following manuals ECMWF (2015), ECMWF, n.d. and Hewlett-Packard, n.d. However, it is important to note this one feature of CrayPat that may have caused incompatibility with the NEMO model (discussed in Chapter 6.1.1. The way CrayPat is instrumented to the model is by rebuilding the model's executable file and instrumenting itself on every object file and linked libraries during a "second compilation". This method can become problematic if intermediate files or linked libraries generated during the model's first compilation is later deleted. Then, CrayPat cannot be instrumented to the model following the simple instruction in the manuals.

This is indeed the case for the NEMO model. During re-compilation with CrayPat, linked libraries in a temporary directory are reported to be missing when it was only the temporary directory that was deleted. Two methods of how to bypass this compilation error are detailed in the jupyter notebook `NEMO-XIOS-CrayPat-error.ipynb`[1]. Thus, CrayPat was instrumented to NEMO successfully.

5.2 Studying scalability of NEMO subroutines

Identifying least scalable subroutine.
From the scalability investigation, the model configuration observed to be less optimized is chosen for further examination to identify the least scalable subroutine within the model. From the results in Chapter 6, NEMO is found to be the less optimized component within EC-Earth3-HR. Using NEMO's built-in timer module `timing` (Benshila, 2001), the wall clock and CPU time of each subroutine during a simulation are determined. NEMO experiments are then repeated using [72, 144, 216, 288, 360, 432, 504, 576][2] CPUs to study the scalability of each subroutines.

A linear least-squares regression (LLSR) curve is fitted to each subroutine's scalability data. Subroutines with a LLSR slope less than 0.5 (slope of the theoretical speedup) are plotted against the cumulative time percentages to identify the least scalable subroutine that also consumes a substantial percentage of NEMO-only simulation time. Results are presented in Chapter 6 and the subroutine `limdyn`, designed to initialize the process of solving for the sea-ice velocities, is identified to be the least scalable subroutine. In which, another subroutine `limrhg`, dedicated to compute for the sea-ice rheology and sea-ice velocities using the EVP framework[3], is determined to be the main contributor to `limdyn`'s run-time.

Profiling run-time of the identified subroutine.
Another timer module `tasks_timer`[4], written in Fortran90, is created to determine how much time `limrhg` spends on various tasks such as defining and

[1]Available on `https://github.com/ylo0803/KU_thesis_2022`
[2]Subroutines' scalabilities are studied using a smaller range of computer resources in order to prevent long waiting times for available resources. There was ongoing supercomputer maintenance which limited access to the supercomputer.
[3]Presented in Chapter 4.2.2
[4]Available on `https://github.com/ylo0803/KU_thesis_2022`. The structure and code of this timer is written inspired by NEMO's built-in module *timing*

initializing variables, MPI communications, solving dynamical equation in iterations, etc.. Six aspects are accessed within `limrhg`:

- creation of sea-ice "masks" at the corners of Arakawa C grid to represent sea-ice location on ocean grid
- computation for wind/ocean stress and Coriolis force terms in the momentum equation (Eq. (4.1))
- computation for sea-ice internal stress term and solving for the sea-ice velocities using the EVP framework
- recompute invariant Δ of the strain rate tensor
- getting diagnostics on sea-ice variables
- MPI communications

Each aspect is measured in CPU and wall clock time.

Identifying potential area for vectorization optimization.
Using the Intel Fortran compiling option `-qopt-report=2 -qopt-report-phase=vec`, a report of potential optimization is returned. It documents the areas in the code where vectorization is employed and not employed, along with reasons as to why vectorization is not recommended at certain areas. It also provides information such as detected data dependency, and suggested other tools that may be used for optimizing the subroutine's performance.

A shortened version of the complete report on `limrhg.f90` is available in Appendix 9.3, which contains only relevant information on the EVP algorithm and details on all non-optimizable loops within the script. A level 5 report is also obtained for further investigation into some "assumed OUTPUT dependency" that the compiler has reported.[5]

[5]Available on `https://github.com/ylo0803/KU_thesis_2022`

Results

6

The results of this thesis project are presented in three main sections. The first is dedicated to results from attempting to determining the optimal load balance for EC-Earth3-HR AOGCM model. This includes a brief comparison of each model's (AOGCM and its components IFS & NEMO) run-time efficiency between standard and high resolution configuration. This is followed by scalability analysis of the components' standalone model and results from instrumenting CrayPat on both the IFS- and NEMO-only models.

Next, the scalability analysis of NEMO-only high-resolution model's subroutines are presented. Additional results from measuring run-time in a specific sea-ice subroutine that is identified to be contributing the most to NEMO-only high-resolution model run-time.

Finally, results from vectorization optimization reports returned by the Intel compiler are presented with further discussion in Chapter 7.

6.1 Optimal load balance for EC-Earth3-HR AOGCM

The following results are obtained from samples of five 10-day simulation performed in each model configuration: IFS-only, NEMO-only and AOGCM, as described at the beginning of Chapter 5.

6.1.1 Comparison of models' efficiency at high and standard resolution

First, a trivial comparison of execution efficiency between the two resolution configuration is performed for EC-Earth3 AOGCM and its components'

Resolution	Metric (Average)	Model configuration		
		IFS-only	NEMO-only	AOGCM
Standard (T255L91, ORCA1L75)	SYPD	12.33±0.60	19.80±0.45	10.07±0.49
	Run-time (s)	192±10	120±3	235±11
High (T511l91, ORCA025L75)	SYPD	1.43±0.03	0.44±0.01	–
	Run-time (s)	1654±41	5363±125	–

Table 6.1: SYPD and run-time comparison between standard and high resolution configuration for each model using the same number of CPUs. The averages are computed over samples of five 10-day simulation performed using each model. Resolution of IFS and NEMO are given in parentheses under the first column. The uncertainties are given by the one-standard deviation value.

standalone models. Comparing the wall clock run time and measured SYPD in either resolution configuration would shed light on how resolution scales with EC-Earth3's efficiency on CRAY XC50. Each model is assigned the same number of CPUs at both resolution configurations: 252 for atmosphere and 144 for ocean. A sample of five simulations over a 10-day period is gathered for each model and the averages are reported in Table 6.1 along with their respective 1-standard deviation uncertainties.

The exact relationship between resolution and run-time efficiency is model dependent. The relationship is even harder to decipher for state-of-the-art tools such as ESM, since it involves complexity of running several components in parallel and uses MPI to enhance parallelism. However, we can still get a sense of how SYPD and wall clock run-time scale with increasing resolution via simulations.

For IFS-only, the number of grid points increases by a factor of ~ 4 from 8,084,258 to 31,716,048 when the resolution becomes more fine from ~ 80 km to ~ 40 km. Both SYPD and wall clock run-time changes by a factor of ~ 8.6, with SYPD decreasing and run-time increasing. It may appear as if the number of grid points quadruples while run-time efficiency decreases by a factor of 8, and that implies a linear relationship between the two elements. But this should not be assumed to be the case from a small sample of two measurements.

For NEMO-only, the number of grid points increases by a factor of ~ 14.3 from 7,927,800 at 1° resolution to 113,557,500 at 0.25° resolution. Both SYPD and run-time changes by a factor of ~ 45.0. The exact relationship between resolution and model's run-time efficiency cannot be determined for NEMO-only in EC-Earth3 on CRAY XC50 for the same reason stated for IFS-only.

Nevertheless, measuring the SYPD and run-time of these component models in high resolution using the same number of computer resources as they do in standard resolution gives an intuition of how well each model performs on CRAY XC50. In terms of SYPD, following the CPMIP metrics guidelines for ESM (Balaji *et al.*, 2017), the NEMO component seems to be performing worse compared to IFS as it cannot even simulate half a year (0.44 years = 5.28 months) of climate evolution per day[1]. Meanwhile, the IFS component can simulate almost 1.5 years (1.43 SYPD) of climate evolution per day.

As for AOGCM, SYPD and wall clock run-time are not measurable in high resolution configuration using the same number of processors in standard resolution due to insufficient computer resources. Thus, this leads to the investigation of the minimum number of CPUs needed for EC-Earth3-HR AOGCM to complete successfully, which will be discussed next.

6.1.2 Load balance: scalability & performance analysis

The SYPD scalability results of high resolution IFS- and NEMO-only models, with respect to optimal load balance of AOGCM in standard resolution, are plotted in Figure 6.1. The ideal scalability of each model is presented as a dashed linear curve in blue or orange. IFS-only model's ideal scalability is given by the equation $f(x) = 5.67\text{e}{-3}x$ whilst the NEMO-only model's is given by $f(x) = 3.06\text{e}{-3}x$.[2]

From Figure 6.1, IFS-only appears to be scaling worse with increasing number of resources compared to NEMO-only. Looking at the ideal scalability of each component model, however, shows that ideal NEMO is still performing worse than ideal IFS. This can be explained by the result that NEMO has an ideal scalability slope of $3.06\text{e}{-3}$, which is less than IFS' $5.67\text{e}{-3}$. As the number of

[1]Real time

[2]The slope of each ideal scalability curve is computed by taking the average SYPD reported in the first row of Table 6.2 and divide it by the corresponding number of CPUs.

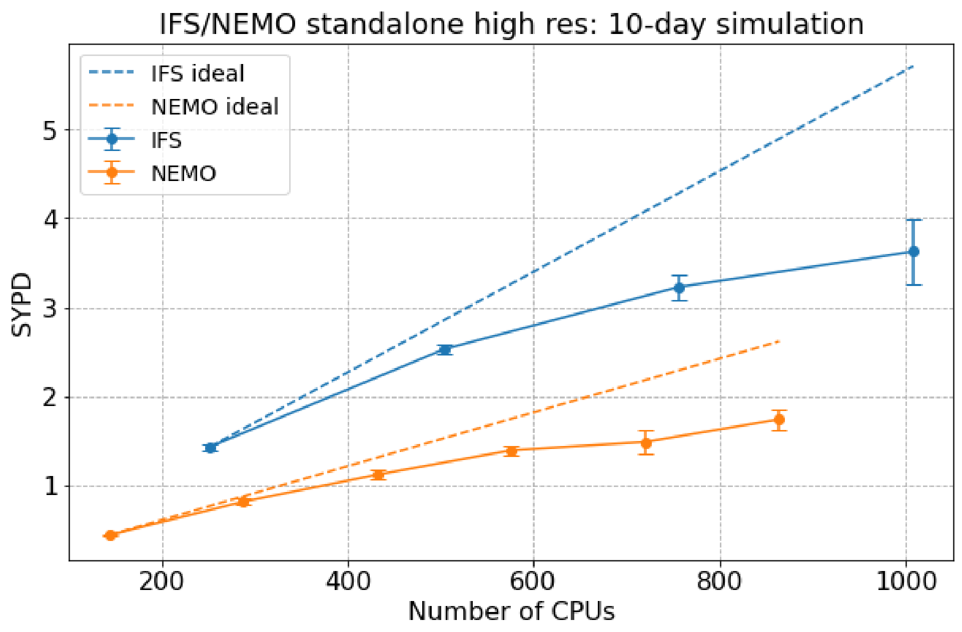


Figure 6.1: IFS-only (blue) and NEMO-only (orange) scalabilities in EC-Earth3-HR on CRAY XC50. The scalabilities are measured in terms of SYPD. The average SYPD and the corresponding 1-standard deviation uncertainties are attained from samples of five 10-day simulations performed using different numbers of CPUs. The exact numbers are presented in Table 6.2

CPUs increases to 800, an increase in SYPD difference between NEMO and IFS is observed. This is a significant increase compared to the SYPD difference at around 250 CPUs. This shows that, in high resolution configuration, the NEMO component is still performing worse than IFS in terms of run-time efficiency even though it is more scalable than IFS on the CRAY XC50 platform.

Scalabilities in terms of wall clock time is also studied and the results (Figure 9.1 and 9.2) are given in the Appendix (Chapter 9). These results show that NEMO-only is actually scaling more efficiently in terms of wall clock time. In accordance with CPMIP metrics, this thesis continues with its investigation in the scalability of NEMO's subroutines based on the results of SYPD scalability. Further investigation into this discrepancy between SYPD and wall clock scalabilities of each component should be revisited.

The scalability of EC-Earth3-HR AOGCM on CRAY XC50 is inconclusive due to its demand for large amount of computer resources. Further discussion on determining the minimum number of processors needed for the simulation to complete is presented in Chapter 7

IFS no. of CPUs	Average SYPD
252	1.43±0.03
504	2.53±0.06
756	3.23±0.14
1008	3.626±0.37

NEMO no. of CPUs	Average SYPD
144	0.44±0.01
288	0.81±0.03
432	1.12±0.05
576	1.39±0.06
720	1.48±0.14
864	1.74±0.11

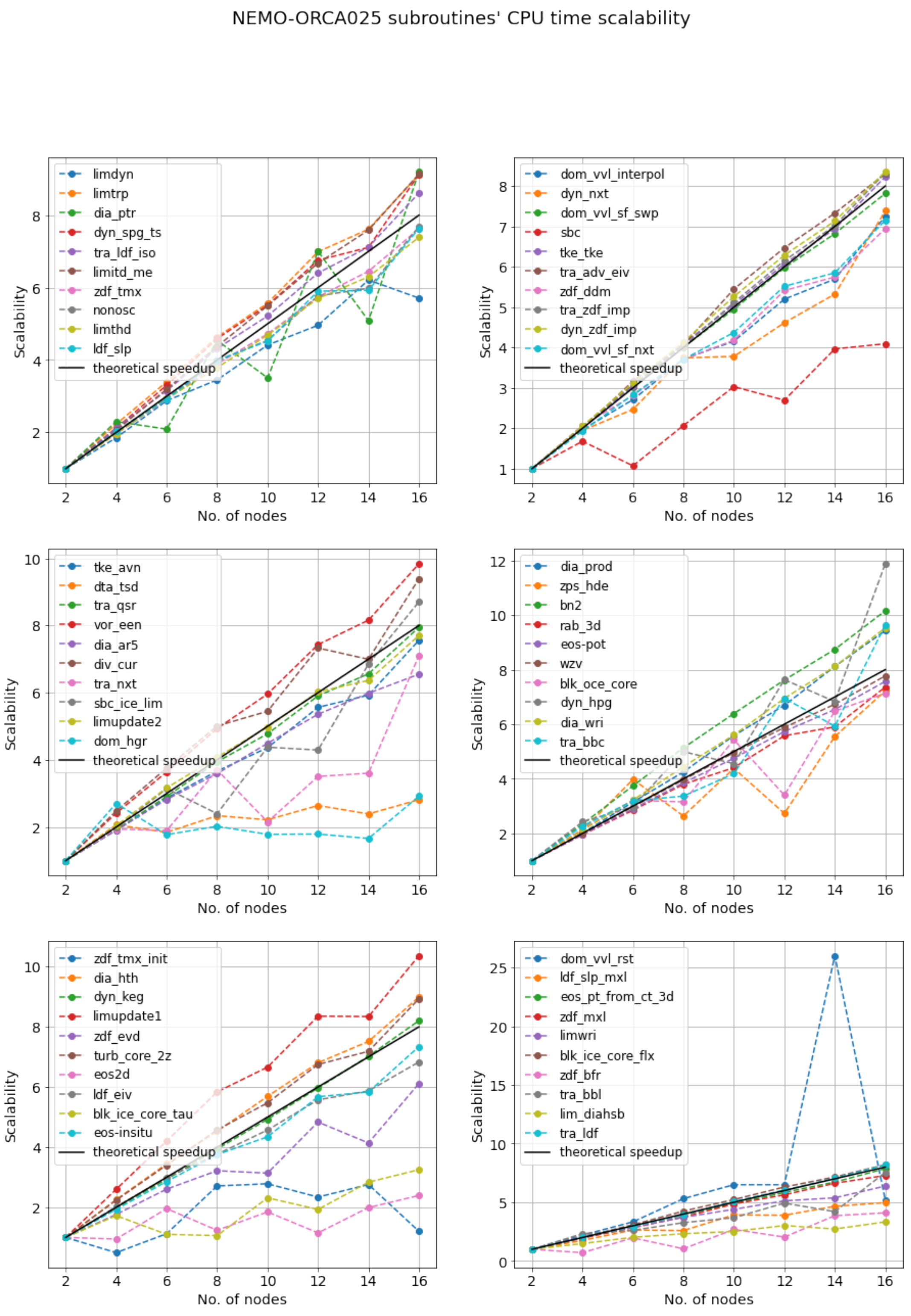
Table 6.2: Average SYPD and 1-standard deviation uncertainties data plotted in Figure 6.1.

Performance analysis of EC-Earth3-HR AOGCM was also attempted by benchmarking the components (IFS & NEMO) using the CrayPat software available on CRAY XC50. IFS-only simulation with CrayPat instrumented is performed and completed successfully. Simulations of NEMO-only with CrayPat instrumented, however, failed. An example of the returned error messages are shown below in the standard output file `np01.out.001` of a NEMO-only simulation:

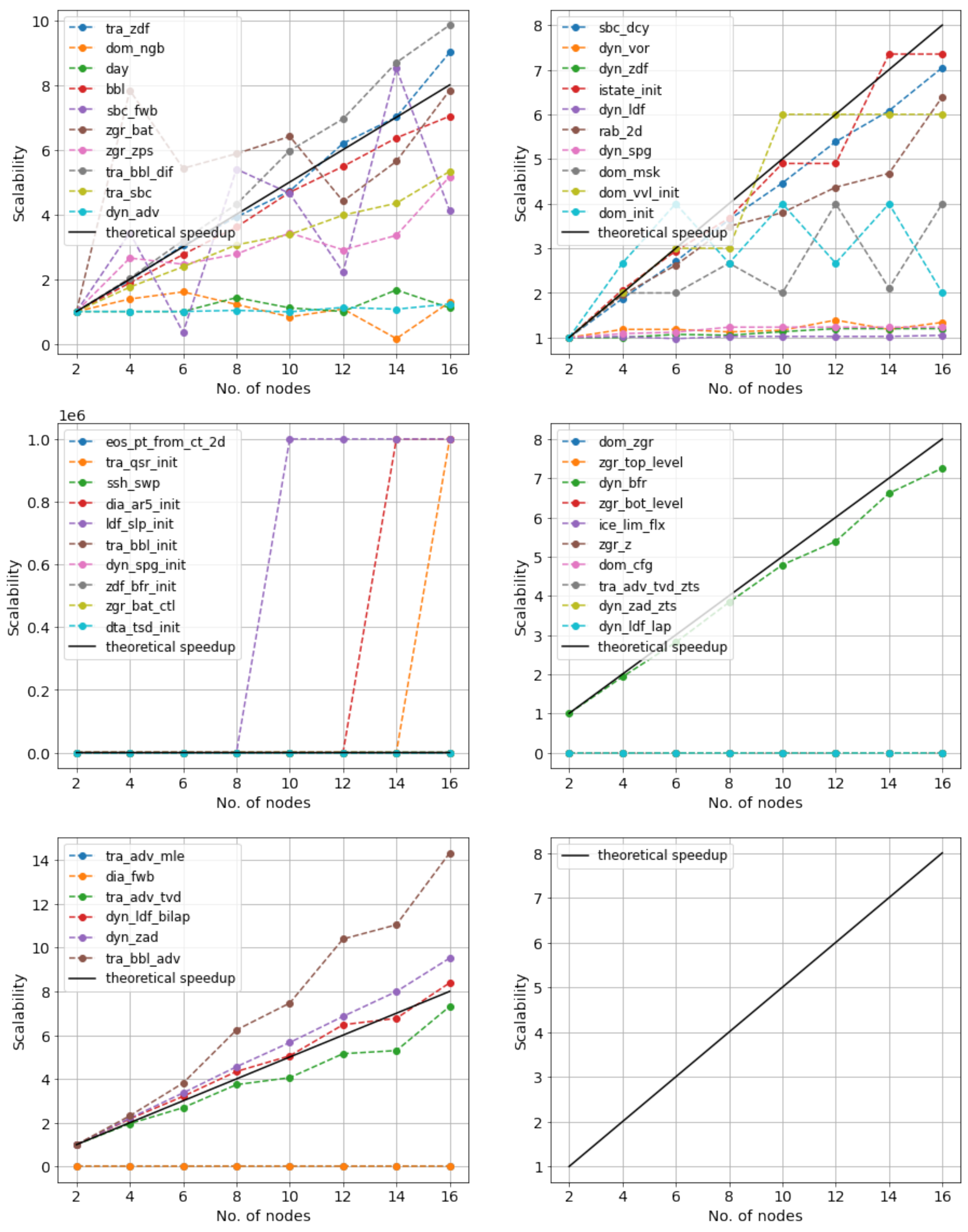
```
*II* nemo original domain decomposition (not using ELPiN)
/usr/bin/time -p aprun -n 1 ./xios_server.exe+pat : -n 144 ./nemo.exe+pat
CrayPat/X:  Version 7.0.0 Revision 5c29ce2  12/11/17 15:26:24
Rank 0 [Mon Nov 15 10:55:17 2021] [c0-0c1s4n0] Fatal error in ↩
    PMPI_Group_translate_ranks: Invalid rank, error stack:
PMPI_Group_translate_ranks(220): MPI_Group_translate_ranks(group=0x88000001, n=1,↩
     ranks1=0x7fffffff60ec, group=0x88000000, ranks2=0x7fffffff60fc) failed
PMPI_Group_translate_ranks(191): Invalid rank has value 10652570 but must be ↩
    nonnegative and less than 1
forrtl: error (76): Abort trap signal

Stack trace terminated abnormally.
_pmiu_daemon(SIGCHLD): [NID 00080] [c0-0c1s4n0] [Mon Nov 15 10:55:17 2021] PE ↩
    RANK 0 exit signal Aborted
[NID 00080] 2021-11-15 10:55:17 Apid 35427172: initiated application termination
Application 35427172 exit codes: 134
Application 35427172 exit signals: Killed
Application 35427172 resources: utime ~1s, stime ~9s, Rss ~47520, inblocks ~0, ↩
    outblocks ~64880
real 5.42
user 0.99
sys 0.20
```

From the above message, a fatal error in `PMPI_Group_translate_ranks` is detected in Rank 0 with an additional message of "invalid rank" reported. This led to the exit of the process and the simulation is killed.

Since NEMO is unable to run successfully with CrayPat, performance analysis of AOGCM is incomplete and the optimal load balance between the IFS and NEMO components cannot be determined with the CrayPat software on the HPC platform CRAY XC50.

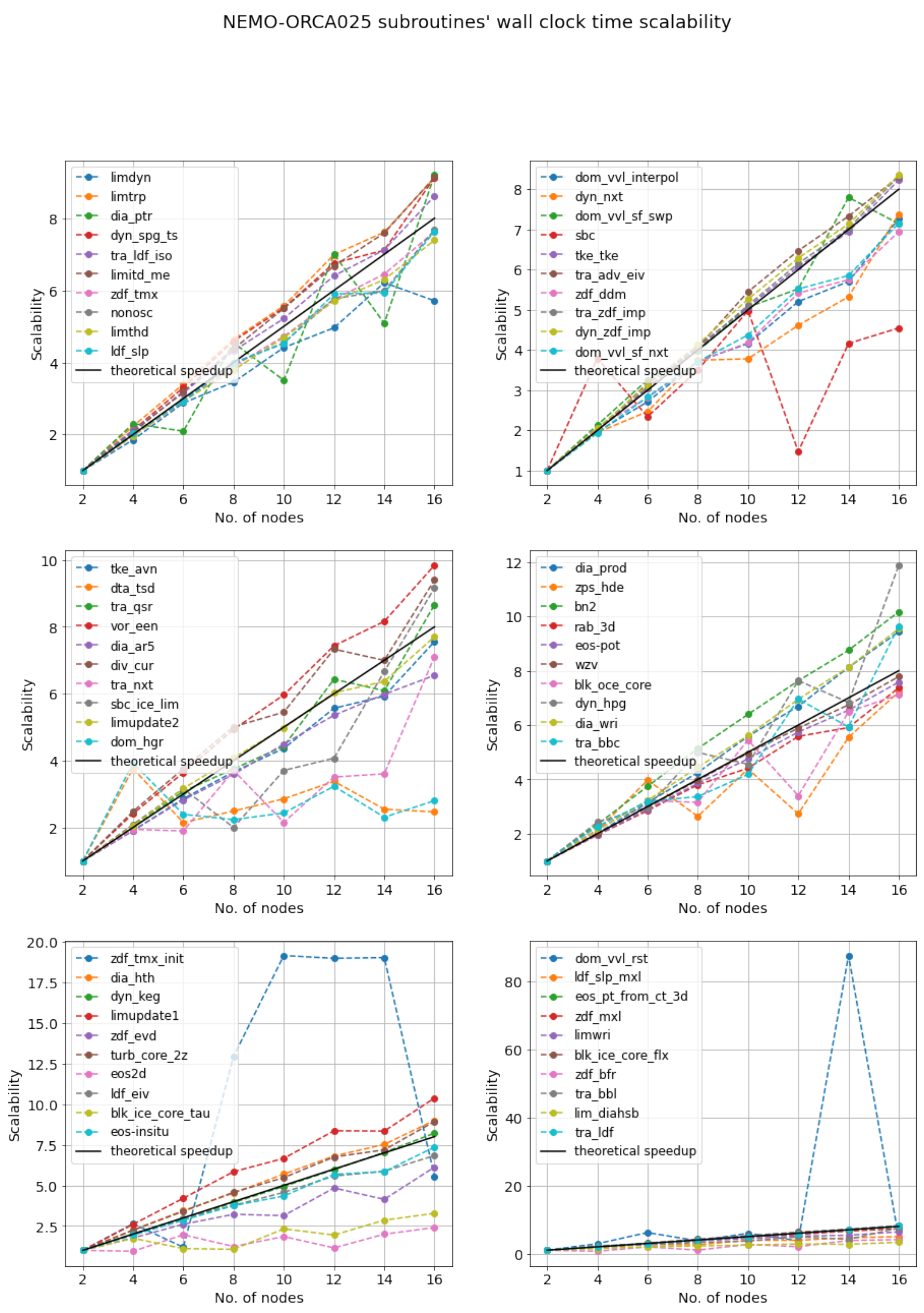


(a) Top 60 subroutines in NEMO-only high-resolution model that consumes the most CPU time during a 10-day simulation. The top 10 subroutines with the highest CPU time are presented in the upper-left plot with *limdyn* consuming the most CPU time and *ldf_slp* consuming the least CPU time out of the top 10.

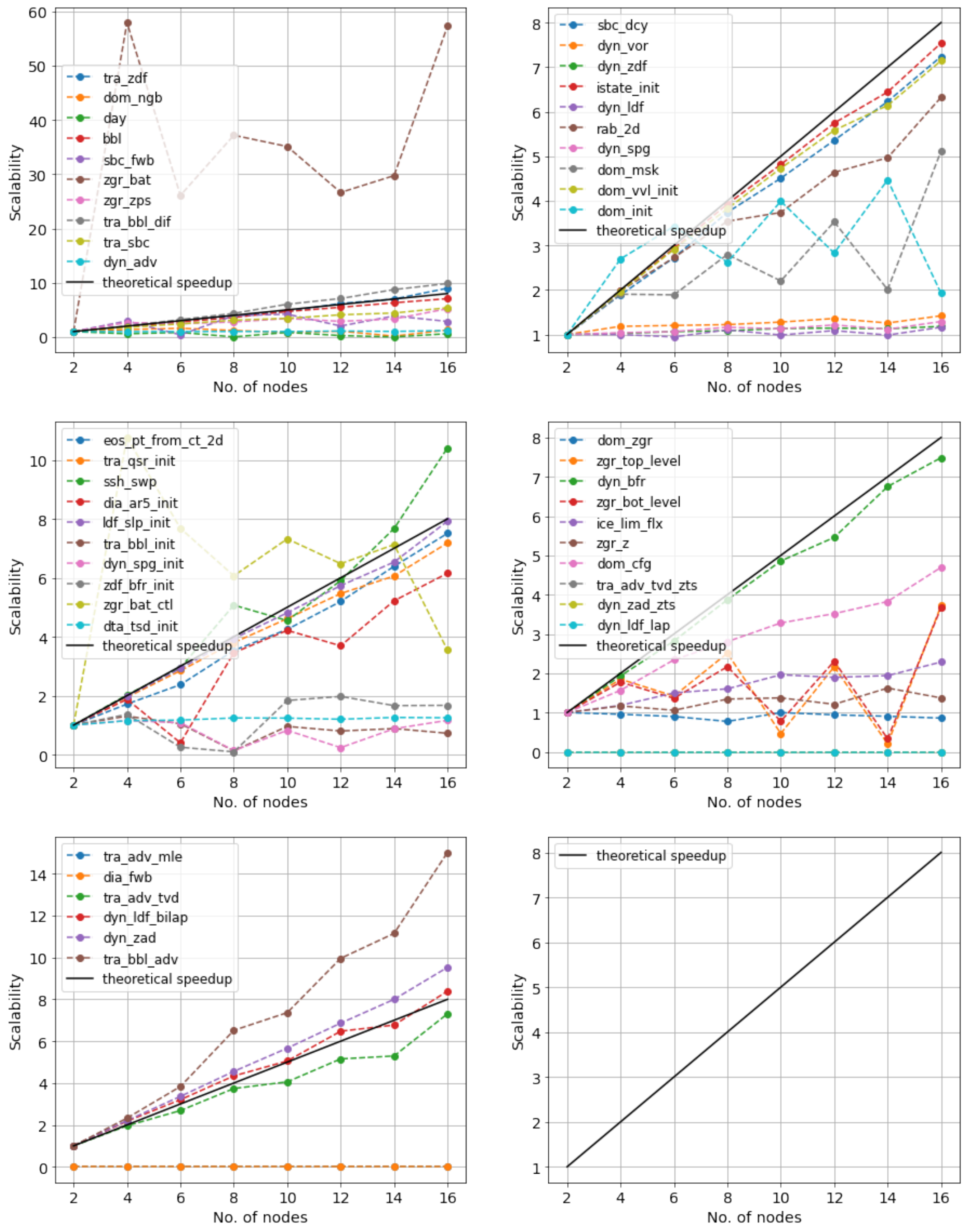


(b) CPU scalabilites of the remaining subroutines in NEMO model.

Figure 6.2: CPU scalability of all subroutines in NEMO-only model at resolution ORCA025. The subroutines are plotted in order of decreasing CPU time. Figure 6.2a shows the top 60 subroutines and 6.2b shows the remaining subroutines. Please read the plot in the following order: upper-left, upper right, middle left, middle right, lower-left and lower-right. The subroutines in each legend are also listed in order of decreasing CPU time. An example is given in the caption of Figure 6.2a.



(a) Top 60 subroutines in NEMO-only high-resolution model that consumes the most wall clock time during a 10-day simulation. The top 10 subroutines with the highest wall clock time are presented in the upper-left plot with *limdyn* consuming the most wall clock time and *ldf_slp* consuming the least wall clock time out of the top 10.



(b) Wall clock scalabilites of the remaining subroutines in NEMO model.

Figure 6.3: Wall clock scalability of all subroutines in NEMO-only model at resolution ORCA025. The subroutines are plotted in order of decreasing wall clock time. Figure 6.3a shows the top 60 subroutines and 6.3b shows the remaining subroutines. Please read the plot in the following order: upper-left, upper right, middle left, middle right, lower-left and lower-right. The subroutines in each legend are also listed in order of decreasing CPU time. An example is given in the caption of Figure 6.3a.

6.2 Scalability analysis of subroutines in NEMO-only high-resolution model

The following results are obtained from samples of ten 10-day NEMO-only high-resolution simulation. The average CPU and wall clock time for each subroutine is calculated and, in turn, used to compute for the average scalability or speedup.

6.2.1 Least scalable subroutine *limdyn*

Using NEMO's built-in timer module `timing.f90`, the CPU and wall clock time consumption of each subroutine are determined for simulations assigned with [72, 144, 216, 288, 360, 432, 504, 576] CPUs. The results of all NEMO subroutines' scalabilities in CPU and wall clock time are plotted in Figure 6.2 and 6.3. In each subplot of Figure 6.2 and 6.3, scalabilites of ten subroutines are plotted. Subroutines in different subplots sharing the same notation in the legends are completely unrelated[3].

The upper-left plot in Figure 6.2a and 6.3a shows the top ten subroutines that are most time-consuming, with `limdyn` ranked the highest and `ldf_slp` ranked the lowest among the top ten. The scalability of `limdyn` is represented by the dark blue dashed curve, which lays below the ideal speedup represented by the black solid curve. A few subroutines with odd scalabilites are shown in the middle-left plot of Figure 6.2b. A scalability of 10^6 is measured for subroutines `ldf_slp_init` (using 10 nodes and more), `dia_ar5_init` (using 14 nodes and more) and `tra_qsr_init` (using 16 nodes). This is because the CPU time measured for these subroutines using certain number of computer nodes are significantly larger than the measured CPU time using only 2 nodes.[4] Thus, the computed scalabilities of 10^6 for these subroutines are inconsequential.

Linear least-squares regression lines are fitted to all subroutines' scalabilities data[5] and those with slopes less than 0.5, the ideal scalability slope, are plotted

[3]The sharing of same color and line style is simply for facilitating the plotting process of so many subroutines in the NEMO model.

[4]$\lim_{a\to 0} \frac{b}{a} = \infty$ where b is a constant.

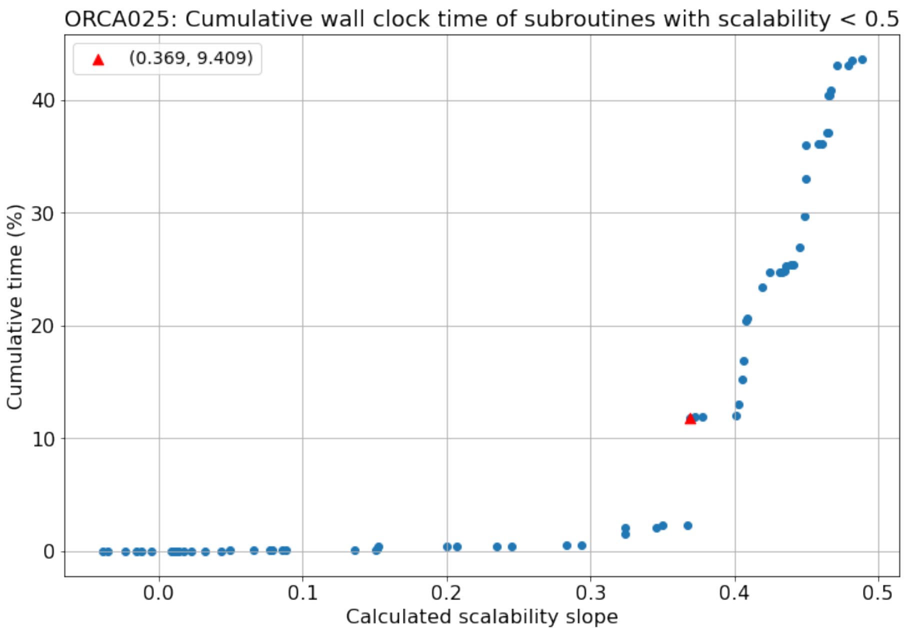
[5]Using the *linregress* function from the python library *scipy.stats*.

in Figure 6.4. The cumulative time percentage of these subroutines are plotted against the computed slopes. In both figures, the red triangle represents the biggest increase in cumulative time and it corresponds to the subroutine `limdyn` with a slope of 0.396. Using 144 CPUs, `limdyn` takes up 9.69% and 9.41% of the total NEMO-only high-resolution 10-day simulation CPU and wall clock time, respectively.

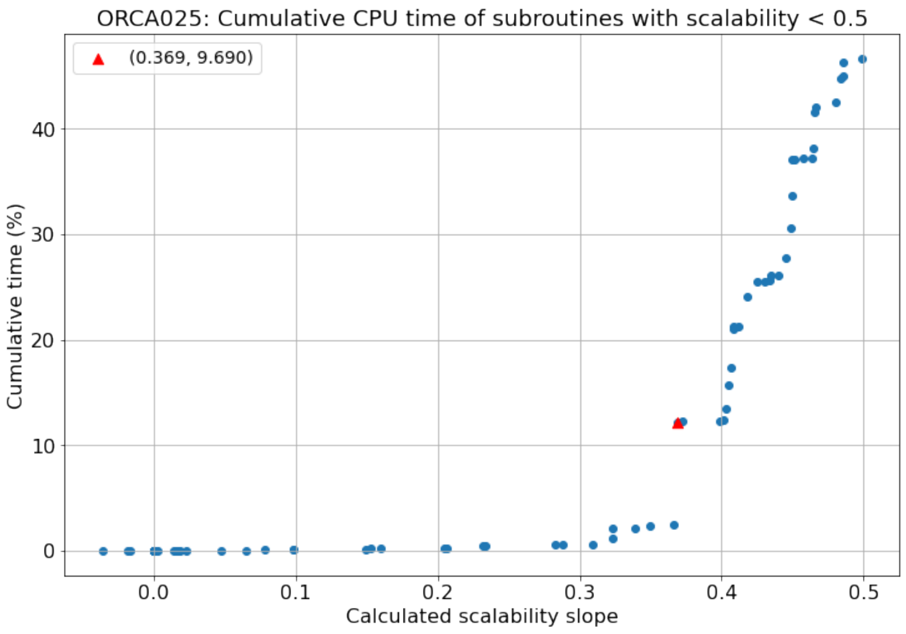
Further inspection into the code of `limdyn` reveals that an important subroutine of LIM3.6 related to sea-ice dynamics is not evaluated by NEMO's built-in timer module. This is the `limrhg.f90` subroutine which is dedicated to compute for the sea-ice rheology and solving for the sea-ice velocities using the EVP framework described in Chapter 4.2.2. Applying the built-in module to this subroutine to measure its CPU and wall clock time, the CPU and wall clock time percentages for `limrhg` are measured to be 9.69% and 9.41% Meanwhile, a drastic decrease of 9.66% and 9.38% in `limdyn`'s CPU and wall clock time percentage is observed. Since the built-in module timer is designed to exclude child-subroutine's run-time from parent-subroutine's run-time, this demonstrates that `limrhg` contributes significantly to `limdyn`'s run-time. And, this result leads to the profiling of run-time consumption within `limrhg` in the next section. [6]

Thus, `limdyn` is identified as the least scalable subroutine in NEMO-only high-resolution model with `limrhg` being the actual time-consuming factor. Next, the results of investigating CPU and wall clock execution time of various tasks performed in `limrhg`. And, further details on how `limdyn` is identified are discussed in Chapter 7.2.

[6]The built-in timer module is designed such that the simulation time (CPU or wall clock) of a child-subroutine is not included in simulation time of the parent-subroutine.



(a) Cumulative wall clock time percentage of all subroutines with scalability slope less than 0.5. The red triangle indicates the subroutine *limdyn* has a slope of 0.369 and takes up 9.409% of the simulation wall clock time.



(b) Cumulative CPU time percentage of all subroutines with scalability slope less than 0.5. The red triangle indicates the subroutine *limdyn* has a slope of 0.369 and takes up 9.690% of the simulation CPU time.

Figure 6.4: A linear least-squares regression line is fitted to the scalability data of each subroutine in the NEMO-only model at resolution ORCA025. Subroutines with a slope less than 0.5 are plotted above on the x-axis. The time percentage each corresponding subroutine takes of the simulation are measured and the cumulative time percentages are plotted on the y-axis. The red triangle in both legends indicates the subroutine *limdyn*.

6.2.2 Run-time consumption within *limrhg*

Applying the second timer module `tasks_timer` to `limrhg`, the CPU and wall clock time of tasks in the following categories[7] are measured:

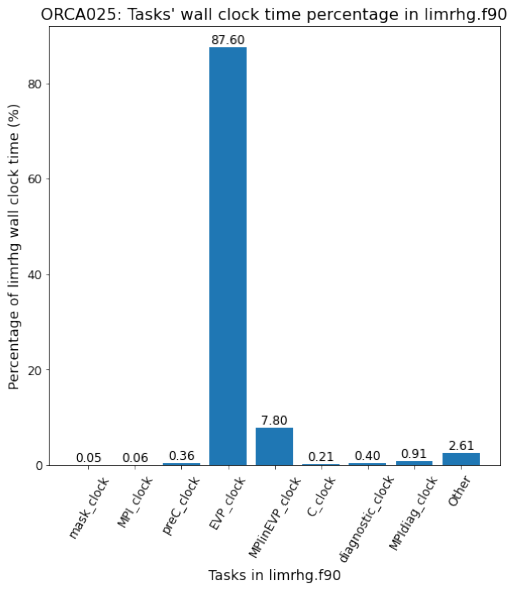
- **mask**: creation of sea-ice "masks" at the corners of Arakawa C grid to represent sea-ice location on ocean grid
- **preC**: computation for wind/ocean stress and Coriolis force terms in the momentum equation (Eq. (4.1))
- **EVP**: computation for sea-ice internal stress term and solving for the sea-ice velocities in 120 iterations
- **C**: recompute invariant Δ of the strain rate tensor
- **diagnostic**: getting diagnostics of sea-ice variables after EVP iteration
- **MPI, MPIinEVP, MPIdiag**: MPI communications (split into three parts)
- **Other**: other tasks (defining variables, initialization of arrays, arrays allocation and deallocation, etc.)

The results are presented in Figure 6.5 in terms of percentages of the subroutine `limrhg`'s run-time (CPU and wall clock) that is reported from `timing` module. CPU time percentages are plotted in Figure 6.5b and wall clock time percentages are plotted in Figure 6.5a.[8]. The percentages are labeled at the top of each bar. The time percentages labeled **EVP** is corrected such that the overhead from timing MPI communications within the EVP iteration, and I/O for these measurements, is not included. The time percentages of carrying out these MPI communications within the EVP iterations, however, is included in the timer percentages of the EVP iterations.

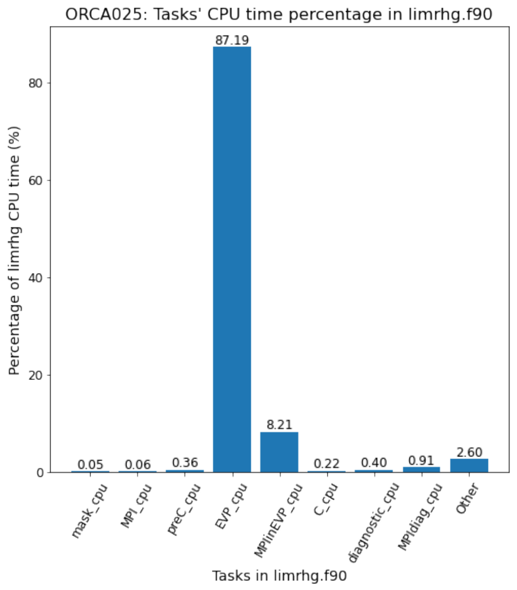
In both bar plots, the iteration for computing the sea-ice internal stress term and sea-ice velocities, using the EVP framework, is shown to be the most time

[7]The categories given the same labels as the ones in the timer *tasks_timer*

[8]Note that the timing of MPI communications is split into three sub-categories for easy application of *tasks_timer* module



(a) Average percentage of wall clock time each group of tasks takes in *limrhg.f90*.



(b) Average percentage of CPU time each group of tasks takes in *limrhg.f90*.

Figure 6.5: Measured wall clock and CPU time of all tasks performed within the subroutine *limrhg.f90*. The tasks are grouped into seven categories with MPI communications having three subcategories. Average percentage of wall clock and CPU time of each category in the subroutine. The averages are computed from a sample of ten 10-day simulations with 144 CPUs assigned to NEMO-only ORCA025.

consuming task in `limrhg` with 87.60% in wall clock time and 87.19% in CPU time. The second highest-ranked task is MPI communications called within this process, with a wall clock time percentage of 7.80% and CPU time percentage of 8.21%. From high to low percentages, the rest of the measured categories are ordered as thus: other tasks with 2.61% wall clock time and 2.60% CPU time; MPI communications called for exchanging calculated diagnostic of sea-ice variables with 0.91% for both wall clock and CPU time; calculating diagnostic of sea-ice variables with 0.40% for both wall clock and CPU time; computation for external stress and force with 0.36% for both wall clock and CPU time; re-computation of strain rate tensor invariant with 0.21% wall clock time and 0.22% CPU time; MPI communication of sea-ice masks with 0.06% wall clock and CPU time; and, creation of sea-ice masks with 0.05% wall clock and CPU time.

The result of computation for sea-ice internal stress and sea-ice velocities being the most time-consuming is expected since these tasks are iterated 120 times within the subroutine. In addition, Bouillon *et al.* (2009) also reported that this framework is known to be computationally expensive. Among the MPI communication categories, the same can be argued for **MPIinEVP** which represents MPI communications called within this 120-iteration process.

6.3 Vectorization optimization report on *limrhg*

A vectorization optimization report on `limrhg` is returned by the compiler to diagnose potential vectorization implementation in the subroutine. A shortened version of this report is available in the Appendix 9.3. From the results presented in Section 6.2.2 where the EVP framework (used to solve for the sea-ice internal stress tensor and velocities) is measured to be the most time-consuming tasks in `limrhg`, analysis of potential vectoriztion optimization within this process is chosen to be the focus of this section. Identified optimizable and non-optimizable loops related to the EVP framework will be presented here.

6.3.1 Optimizable loops in EVP framework

Implemented vectorization in single do-loop are reported on L369, L370, L596. The first two lines concern with saving the sea-ice velocities of a previous time steps to `zu_ice` and `zv_ice`. However, this instruction is only carried out `ln_ctl=.TRUE.`. Under an actual global climate experiment, this option is usually set to `.FALSE.`; thus, we can disregard any further investigation into this. The vectorization on L596 instructs the differences between sea ice velocities at the previous and current time steps to be determined. Like the instructions at L369 and L370, this can also be disregarded since it is carried out under the conditional statement at L594 with `ln_ctl=.FALSE.`.

First vector dependence found within the EVP algorithm is reported at L376 where the inner do-loop for calculating the shear strain rates at the corners of Arakawa C grid. An OUTPUT dependency is assumed between the shear strain rate `zds` and the u-component of the sea ice velocity `u_ice`.

Another assumed OUTPUT dependence is reported at the inner do-loop on L422 where the terms P/Δ and σ_{12} (both from Eq. (4.3)) at the corners of Arakawa C grid are calculated. A clear dependence of the latter on the first is shown in L428 where the stress rate is being computed. Furthermore, the calculation for P/Δ at the corners `zp_delf` on L425 depends on the same term at T points on the Arakawa C grid, which must be computed beforehand (as it does at L411 in the previous nested do-loops). The outer do-loop at L421, however, may be further optimized using SIMD directive.

This SIMD directive is also suggested by the report on L598 when determining the maximum value of array `zresm`. Since this instruction is carried out under the if-statement with `ln_ctl` on L594, we may disregard further investigation here also for `ln_ctl=.FALSE.`.

Vectorization that is already implemented within interfaces `lbc_lnk` for MPI communication on L385 and L419 is also reported. A quick investigation into building of `lbc_lnk` interface and the subroutines used within reveals the complexity of how MPI communications are carried out in NEMO. Any potential optimization in regards to MPI communication will have to be further investigated by studying the scalability of MPI communication within `limrhg`.

6.3.2 Non-optimizable loops in EVP framework

As for the non-optimizable loops found within the implementation of EVP algorithm, they can be explained by either compilation constraints or prevention of outer loop vectorization due to inner loop throttling.

The first reasoning is used for nested loops ending on the following lines:

- L384: begins at L375 to calculate for the shear strain rates at grid's corners.

- L418: begins at L387 to calculate for the D_S^2, D_D, D_T, Δ, P/Δ, σ_1 and σ_2 at T points.

- L463: begins at L435 to calculate for the sea-ice internal stress forces, and the interpolated velocities `v_iceU` and `u_iceV` at **u** and **v** points, respectively.

- L494: begins at L470 to calculate for sea-ice velocity `v_ice` first in even iterations

- L524: begins at L500 to calculate for sea-ice velocity `u_ice` after in even iterations

- L556: begins at L532 to calculate for sea-ice velocity `v_ice` first in odd iterations.

- L586: begins at L562 to calculate for sea-ice velocity `u_ice` after in odd iterations.

The compilation constraints are also reported to be the cause of non-optimizable loops ended on L371 and L597. Theses are disregarded, however, because they are only carried out under the `IF` statement with `ln_ctl`. For `ln_ctl=.FALSE.` in NEMO, these loops are not performed.

The second reasoning is reported for non-optimizable procedures related to `lbc_lnk` interfaces. Specifically, it was reported at the position where the arrays to be exchanges are stated at L385,24 and L419,24, as shown below:

```
! Calculate shear at F points (corners of Arakawa C grid)
CALL lbc_lnk( zds, 'F', 1. )     ! (L385,24)
! ...
! Calculation of variables at T points (center of Arakawa C grid
! ...
CALL lbc_lnk( zp_delt, 'T', 1. ) ! (L419,24)
```

where 24 indicates the position of arrays `zds` and `zp_delt`. Further investigation in this might be beneficial for considering potential optimization in MPI aspects of the NEMO model.

Discussion 7

Further discussion on some of the results, shown in Chapter 6, is presented here. First, discussion of results from attempting to determine optimal load balance using scalability and performance analysis of EC-Earth3-HR AOGCM is presented. This is followed by details on the process of identifying `limdyn` as the least optimal subroutine in NEMO. After that, a brief recap of profiling run-time within `limrhg`, the main execution time contributor to `limdyn`, is mentioned before a small anomaly in the time measurement of MPI communication within EVP framework is addressed. Finally, a detailed discussion on potential vectorization in computation for the shear strain rates is presented.

7.1 Optimal load balance for AOGCM in EC-Earth3-HR

Recall from Chapter 6.1.2 that the execution time of the coupled model in high resolution configuration cannot be compared with that in the standard resolution configuration because of lack of computer resources. Here, trial simulations are carried out to determine the least amount of computer resources EC-Earth3-HR AOGCM needs to complete successfully. Afterward, we will dive into the cause of failed ocean simulation when the model is instrumented with the performance analysis tool CrayPat.

7.1.1 CPU resources for EC-Earth3-HR AOGCM

With a more refined map, there exist a higher number of grid points which demands for more computation to be carried out in order for a simulation to complete. More computation means more operation to be performed and, in turn, demands for more CPUs for sufficient memory space and computer power. Following the detailed approach described in Chapter 5.1, the minimum computer resources needed for EC-Earth3-HR AOGCM to run successfully on CRAY XC50 is determined. Therefore, this must be taken into consideration when CPU resources are being allocated to the components of AOGCM.

A minimum of 1263 CPUs are required for an EC-Earth3-HR AOGCM simulation to complete successfully. Of the 1263 CPUs, 1008 CPUs are allocated for NEMO component, 252 for IFS component, and the remainder is dedicated to I/O server and the runoff mapper to handle exchanges of freshwater. This remains the case with I/O turned off for simulation. In order to analyze the scalability of AOGCM, measurements from samples of simulation performing with multiples of 1008 and 252 CPUs are required. This was a challenging task to acheive during times when resources on CRAY XC50 were in high demands and constant maintenance was carried out on the supercomputer platform. Thus, scalability analysis of EC-Earth3-HR AOGCM is inconclusive. Nevertheless, the minimum computer power for running EC-Earth3-HR AOGCM is determined.

Using this set of CPU resources, EC-Earth3-HR AOGCM's performance efficiency is measured over a sample of five 10-day simulations. It is measured to have an average 1.31 ± 0.05 SYPD and an average wall clock time of 1801 ± 63 seconds.

7.1.2 CrayPat Instrumentation on NEMO3.6

As mentioned previously, the performance analysis of AOGCM in high resolution on CRAY XC50 was unsuccessful due to failed simulations from the NEMO component when it is instrumented with CrayPat. An example of the error message is shown in Chapter 6.1.2 where a fatal error is detected in `PMPI_Group_translate_ranks`. More simulations were performed to locate the source of this error by using different combination of NEMO and XIOS with and without CrayPat instrumented, and varying the number of CPUs allocated to either executable. It is concluded that the error stems from some unidentified incompatibility between CrayPat and the NEMO model, particularly the XIOS external server for I/O. Both of the executable files have to be instrumented in order for CrayPat to profile the entire ocean model's performance. When more processors are assigned to XIOS a similar error message is returned as stated below:

```
*II* nemo original domain decomposition (not using ELPiN)
/usr/bin/time -p aprun -n 2 ./xios_server.exe+pat : -n 144 ./nemo.exe+pat
CrayPat/X:  Version 7.0.0 Revision 5c29ce2  12/11/17 15:26:24
Rank 1 [Mon Nov 15 12:16:10 2021] [c0-0c0s15n1] Fatal error in ↩
    PMPI_Group_translate_ranks: Invalid rank, error stack:
PMPI_Group_translate_ranks(220): MPI_Group_translate_ranks(group=0x88000001, n=1,↩
     ranks1=0x7fffffff61bc, group=0x88000000, ranks2=0x7fffffff61cc) failed
PMPI_Group_translate_ranks(191): Invalid rank has value 32767 but must be ↩
    nonnegative and less than 2
Rank 0 [Mon Nov 15 12:16:10 2021] [c0-0c0s15n1] Fatal error in ↩
    PMPI_Group_translate_ranks: Invalid rank, error stack:
PMPI_Group_translate_ranks(220): MPI_Group_translate_ranks(group=0x88000001, n=1,↩
     ranks1=0x7fffffff616c, group=0x88000000, ranks2=0x7fffffff617c) failed
PMPI_Group_translate_ranks(191): Invalid rank has value 10652570 but must be ↩
    nonnegative and less than 2
forrtl: error (76): Abort trap signal

Stack trace terminated abnormally.
forrtl: error (76): Abort trap signal

Stack trace terminated abnormally.
_pmiu_daemon(SIGCHLD): [NID 00061] [c0-0c0s15n1] [Mon Nov 15 12:16:11 2021] PE ↩
    RANK 0 exit signal Aborted
[NID 00061] 2021-11-15 12:16:11 Apid 35430693: initiated application termination
Application 35430693 exit codes: 134
Application 35430693 exit signals: Killed
Application 35430693 resources: utime ~0s, stime ~9s, Rss ~46768, inblocks ~0, ↩
    outblocks ~16
real 4.84
user 1.02
sys 0.12
```

Comparing this error message to the one presented in Chapter 6.1.2, the additional message beginning with `Rank 1` is returned when two CPUs are assigned to XIOS instead of one. Again, the message states that there is a fatal error in `PMPI_Group_translate_ranks`. An invalid rank is also present when the ranks of processes in a MPI group is translated to another using the MPI function. It is suspected that a mistake is developed in the numbering of ranks in different groups, and a rank is assigned a number significantly bigger than the number of CPUs allocated for XIOS.

Due to the MPI complication caused by CrayPat instrumented to NEMO and XIOS, performance analysis of AOGCM is incomplete and the optimal load balance for AOGCM in EC-Earth3-HR on CRAY XC50 cannot be determined using CrayPat.

7.2 Scalability analysis of NEMO subroutines

From Figure 6.2 and 6.3, NEMO subroutines with less-than-ideal scalabilities are clearly identified as they lay below the ideal speedup represented by the black solid curves. Some of these subroutines, however, take up less than 1% of the model simulation time. This must be taken into consideration in order to find the least optimal subroutine where further optimization can be carried out and potentially lead to a substantial impact on NEMO simulation time.

To do so, linear least-squares regression (LLSR) are fitted to the scalabilities data to determine the corresponding slopes for each subroutine. Those with a LLSR slope less than the ideal scalability slope of 0.5 are candidates for identifying the least optimal subroutine. Using the measured CPU and wall clock time percentages of a simulation using 4 nodes, provided by the NEMO built-in timer, the cumulative percentages of the corresponding subroutines are computed. By analyzing the cumulative timer percentage against the LLSR slope, the least optimal subroutine that also takes up a significant percentage of the NEMO model simulation can be identified. As shown in Figure 6.4, we see a significant increase in cumulative time percentage represented by the red triangle. This corresponds to the subroutine `limdyn` which initializes the process for solving the sea-ice momentum equation. It has a LLSR slope of 0.396, less than the ideal slope 0.5, and takes up almost 10% of the a NEMO simulation time. Thus, `limdyn` is identified to be the least optimal subroutine.

7.3 Profiled time consumption within *limrhg*

The detail profiling of `limrhg` is performed using the timer module `tasks_timer`. Tasks within `limrhg` were grouped in the seven categories listed in Chapter 6.2.2. From the results presented in Figure 6.5, the EVP iterations is measured to be the most time-consuming task of `limrhg.f90` as it takes up 87.60% of the subroutine's wall clock time and 87.19% of the subroutine's CPU time.

These measurements also include the time spent on MPI communications (**MPIinEVP**) within EVP iterations. From the measurements of **EVP** and **MPI-inEVP** in CPU and wall clock time, as shown in Table 9.1, proportion of MPI communication in the EVP framework can be computed. It is determined that MPI communications take up only about 9.42% and 8.90% of the EVP iterations' CPU and wall clock time. This means majority of the subroutine `limrhg` run-time is still spent on the performing intensive computation within the EVP framework. Moreover, MPI communication does not contribute significantly to the subroutine's total execution time; the total wall clock time of all MPI processes within `limrhg` amounts to only 8.77% of the subroutine's total wall clock run-time.

A difference of 0.41% in the percentages of CPU and wall clock time of MPI processes within the EVP iterations is noted. With respect to the CPU and wall clock run-time of `limrhg`, which are both approximately 324.0 s[1], 0.41% corresponds to about 1.0 s. A 1-second difference between CPU time and wall-clock time is tolerable and does not reflect on any obvious and alarming timing issues within the model.

From this basic profiling of `limrhg`, it is concluded that the iterative EVP framework used to compute sea-ice rheology and velocities, is the most time-consuming task within the subroutine. In turn, it also contributes to the computational cost of `limdyn`, the least optimal subroutine in the NEMO model.

7.4 Potential optimization in EVP implementation

Summarized results from the vectorization optimization report in Chapter 6.3 details information about loops that are already vectorized, potential loops for vectorization and non-optimizable loops. Here, an analysis one of the reported optimizable loops, as listed in Chapter 6.3 is discussed. Specifically, we will focus on the nested do-loops for calculating the shear strain rates.

[1]Measured by NEMO's built-in timer module.

7.4.1 Analysis of "assumed OUTPUT dependency" in optimizable loops

An OUTPUT dependency was assumed at L376, under the nested loops starting at L375, and specifically between arrays zds and u_ice. Whilst zds is an array and a pointer declared in limrhg, u_ice is an array declared in ice, a module consisting of all diagnostics variables relevant to the sea-ice model LIM3.6. Recall that an OUTPUT dependency means that a variable at a memory address is being written to more than once in (nested) loops. This means that the compiler interprets zds and u_ice to have the same memory address, and data at this memory address can be accessed via either variables. This is known as aliasing, and the two arrays would be aliased arrays.

This would, indeed, be the case if zds, a pointer, is associated with u_ice in the subroutine limrhg.f90 via the following:

```
zds => u_ice
u_ice = zds
```

In the first statement, the array/pointer zds is associated to the target array u_ice. In the second statement, the values of u_ice is changed to be the same as the target that zds "points" to, which is u_ice itself . These statements were not executed in any subroutines within the sea-ice model LIM3.6, an indication that aliasing memory between the two was not initiated via these statements.

Another cause of aliasing memory between the two arrays could stem from the method used to allocate memory for these arrays. For the array u_ice, it is allocated in the module ice using the fundamental Fortran90 function ALLOCATE(). Array/pointer zds is allocated using an interface wrk_alloc consisting of several subroutines that are dedicated to allocate work space for arrays of different dimension. For 2D arrays like zds, they are handled by subroutines wrk_alloc_2dr and wrk_alloc_2di, which in return uses other subroutines (wrk_alloc_xd and wrk_allocbase) to specify multiple conditions before an array is allocated. An investigation into these subroutines would be challenging as the details of the subroutines' methods are not documented in

Madec and NEMO team (2016), which is a manual intended for users of the NEMO model.

A trivial approach to test if aliasing memory between the arrays is resulted due to some complexity of `wrk_alloc` is to use another fundamental Fortran90 function `ALLOCATE()` to allocate space for `zds` instead. Since the dimension of the array `zds` is known[2], the complication of using the interface `wrk_alloc` can be bypassed with following changes in the subroutine `limrhg`:

```
! Keep everything else the same
! Using ALLOCATE() instead for zds
! Take out zds from the wrk_alloc() on L202 and add the following line
ALLOCATE( zds(jpi, jpj) )

...

! Take out zds from wrk_dealloc() on L879 and add the following line
DEALLOCATE( zds )
```

The subroutine is then compiled again using the same compilation flags mentioned in Chapter 5 to generate a new vectorization optimization report. If `wrk_alloc` is the cause for aliased arrays, then replacing it with `ALLOCATE()` should not result in an "assumed OUTPUT dependency". This is not the case however when the new report also returns an "assumed OUTPUT dependency":

```
   LOOP BEGIN at /home/ngyilo/ec_earth3/xio-parlib/sources/nemo-3.6/CONFIG/↩
       ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(410,13)
       remark #15344: loop was not vectorized: vector dependence prevents ↩
           vectorization. First dependence is shown below. Use level 5 report ↩
           for details
       remark #15346: vector dependence: assumed OUTPUT dependence between call:↩
           for_emit_diagnostic (413:33) and call:for_emit_diagnostic (413:16)
   LOOP END
```

The compiler's detection of this dependency would require further investigation.

This "assumed OUTPUT dependency" can be bypassed using Intel's directive `IVDEP` to instruct the compiler to ignore assumed vector dependencies in

[2]From the subroutine in which *limrhg* is called, the dimension of the array is set to be the dimension of the local domains (ji, jj).

a particular loop. This can be done because, from my investigation and understanding of the code, no executable statements are written that would obviously result in aliasing memory between zds and u_ice. Based on this reasoning, the IVDEP directive is used such as follows:

```
...
! --- divergence, tension & shear (Appendix B of Hunke & Dukowicz, 2002) --- !
   DO jj = k_j1, k_jpj-1          ! loops start at 1 since there is no boundary ↩
       condition (lbc_lnk) at i=1 and j=1 for F points
       !DIR$ IVDEP                          ! IGNORE ASSUMED DEPENDENCY
       DO ji = 1, jpim1
! shear at F points
          zds(ji,jj) = ( ( u_ice(ji,jj+1) * r1_e1u(ji,jj+1) - u_ice(ji,jj) * ↩
              r1_e1u(ji,jj) ) * e1f(ji,jj) * e1f(ji,jj)   &
          &         + ( v_ice(ji+1,jj) * r1_e2v(ji+1,jj) - v_ice(ji,jj) * ↩
              r1_e2v(ji,jj) ) * e2f(ji,jj) * e2f(ji,jj)   &
          &         ) * r1_e12f(ji,jj) * zfmask(ji,jj)
       END DO
   END DO
```

The subroutine limrhg.f90 is compiled again and the "assumed OUTPUT dependency" is no longer returned in the vectorization optimization report. However, it is replaced by the following remark:

```
   remark #15527: loop was not vectorized: function call to for_emit_diagnostic ↩
       cannot be vectorized   [ /home/ngyilo/ec_earth3/nocray-compile-v3.3.3.2/↩
       sources/nemo-3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.↩
       f90(380,33) ]
```

The Intel library function for_emit_diagnostic is raised when an error is detected by tests in the code during compilation, such as array bounds violation and unassociated pointers[3] (Intel (2016), University (2022)). Additionally, potential vectorization optimization might not be reported if there are too many manipulation in indexing (e.g. a[i+1] where a is an array) (University (2022)).

Studying the code of the nested do-loops again from L375 to L384, two pointers/arrays are involved and also not associated to a target. They are the array/pointer zds and array/pointer zfmask, of which the latter is the sea-ice mask at the corners of Arakawa C grid. For these pointers are simply used memory allocation in Fortran90. Furthermore, manipulation in indexing

[3] In Fortran, pointers are associated to a target variable via the pointer association operator "=>". An example was given at the beginning of this section 7.4.

is also used where the computation for zds often requires data of variables, such as u_ice and v_ice, at the next jj and ji indices (u_ice[ji,jj+1] and v_ice[ji+1,jj]). These reasoning could explain why the message,

```
function call to for_emit_diagnostic cannot be vectorized
```

, is returned when the compiler is trying to test the code for vectorization optimization.

7.4.2 Potential vectorization for computing shear strain rates

Based on on these information presented in the last section, it is reasoned that the returned message may be insubstantial. Thus, vectorization may be applied in the nested do-loops where the shear strain rate is computed (from L375 to L384).

Looking into the operation of this nested do-loops, one can see that the computation can be separated into two parts using simple algebraic distribution. The first part of the computation requires data at indices (ji,jj) and (ji,jj+1). Meanwhile, the second part requires data at indices (ji+1,jj) and (ji+1,jj). We can thus, split the nested do-loops into two single loops in which they both loop over one index only. The first part, which uses array u_ice, will loop over the jj index only since all other variables in the computation access the same index ji as the do-loop instructed[4]. Similarly, the second part, which uses array v_ice, will loop over the ji index only with the same reasoning previously stated for the index jj of u_ice.

Hence, the nested do-loops for calculating the shear strain rate at the corners of an Arakawa C grid can be vectorized as shown below:

```
REAL(wp), POINTER, DIMENSION(:,:) ::   zds_i   ! shear, iterating over i
REAL(wp), POINTER, DIMENSION(:,:) ::   zds_j   ! shear, iterating over j

! Allocate memory for temporary arrays zds_i, zds_j to have the same dimension as↩
    zds
```

[4]During iteration $ji = 1$, data of u_ice(ji=1,jj=*)

```
! Calculate using column vectorization since it never depends on j+1
DO ji = 1, jpim1
    zds_i(ji,:) = ( ( v_ice(ji+1,:) * r1_e2v(ji+1,:) - v_ice(ji,:) * r1_e2v(ji,:)↩
         ) * e2f(ji,:) * e2f(ji,:) ) * r1_e12f(ji,:) * zfmask(ji,:)
END DO

! Calculate using row vectorization since it never depends on i+1
DO jj = k_j1, k_jpj-1
    zds_j(:,jj) = ( ( u_ice(:,jj+1) * r1_e1u(:,jj+1) - u_ice(:,jj) * r1_e1u(:,jj)↩
         ) * e1f(:,jj) * e1f(:,jj) ) * r1_e12f(:,jj) * zfmask(:,jj)
END DO

! zds = (zds_i + zds_j)
zds = zds_i + zds_j

! Deallocate these arrays later when they're not needed anymore
```

Other areas where "assumed OUTPUT dependency" are reported by the compiler may also be vectorized if the code semantics imply no variable dependency, and similar trivial messages, such as `for_emit_diagnostic`, are returned.

8 Conclusion & Outlook

In this thesis, the optimal load balance of AOGCM in EC-Earth3-HR and potential optimization of the NEMO component of DMI's HPC platform CRAY XC50 are investigated.

Determining the optimal load balance is crucial to optimizing AOGCM's performance by guaranteeing that both the atmosphere and ocean components are given the optimal number of processors that can give the most reduction in execution time of the coupled model. Of the two components, the ocean model NEMO is chosen to be the focus of this study based on the assumption that it's less robust than the atmosphere model IFS, which is developed, maintained and used by the intergovernmental organization ECMWF. Optimizing ocean model in high resolution configuration is important to study climate evolution because refined resolution can resolve fine scale features and benefit coupled model predictions (Hewitt *et al.*, 2017).

Attempts to determine the optimal load balance of EC-Earth3-HR AOGCM were made by studying the scalability of each component, and using software tool CrayPat to profile the model's performance. From scalability analysis, NEMO is concluded to be performing worse than IFS in EC-Earth3-HR. Performance analysis of AOGCM is incomplete due to incompatibility of CrayPat with NEMO's external server XIOS that is dedicated for I/O purposes.

Scalability analysis was also carried out for all of NEMO's subroutines to determine which is the least scalable. `limdyn` is identified to be the least scalable subroutine with `limrhg.f90` found to be contributing most to `limdyn`'s execution time. Furthermore, the EVP implementation used to solve for sea-ice rheology and velocities is measured to be the most time-consuming task within `limrhg.f90`. Nested do-loops of computation for variables such as shear strain rates may be vectorized and lead to potential reduction in NEMO's run-time.

8.1 Outlook

Recall that an interesting result was mentioned when comparing SYPD and wall clock scalabilities of the IFS- and NEMO-only models. Whilst we see that NEMO is performing less efficiently in terms of SYPD, it shows to be the opposite in terms of wall clock time (Figure 9.1 and 9.2). This "discrepancy" between the two metrics is of most interest and should be further analyzed in furture work.

Another "assumed OUTPUT dependency" was detected by the vectorization report where the terms P/Δ (`zp_delf`) and σ_{12} (`zs12`) from Eq. (4.7 are calculated to determine the components of internal stress tensor σ. These computation directly follows that of `zds`, which was discussed that it may have a false detection of "OUTPUT dependency". For this follow-up computation, data and loop dependency analyses must be carried out in the same fashion as that for array/pointer `zds` before one may consider any potential vectorization implementation.

Other vectorization implementation suggested by the report include using SIMD directive for some outer do-loop, iterating over the `jj` index, such as the one for computing `zs12`. As for most of the non-optimizable loops reported in Chapter 6.3, further investigation can easily be carried out by compiling `limrhg` with different option which will allow the compiler to test for more loops that can be potentially optimized.

Overall, the next step to this thesis project would be to design and implement different vectorization methods within `limrhg` following the results from the optimization report. Ocean simulations in the high resolution configuration can then be carried out to observe if there is an increase in efficiency and scalability for the subroutines `limdyn` and `limrhg`.

Appendix 9

9.1 Additional plots

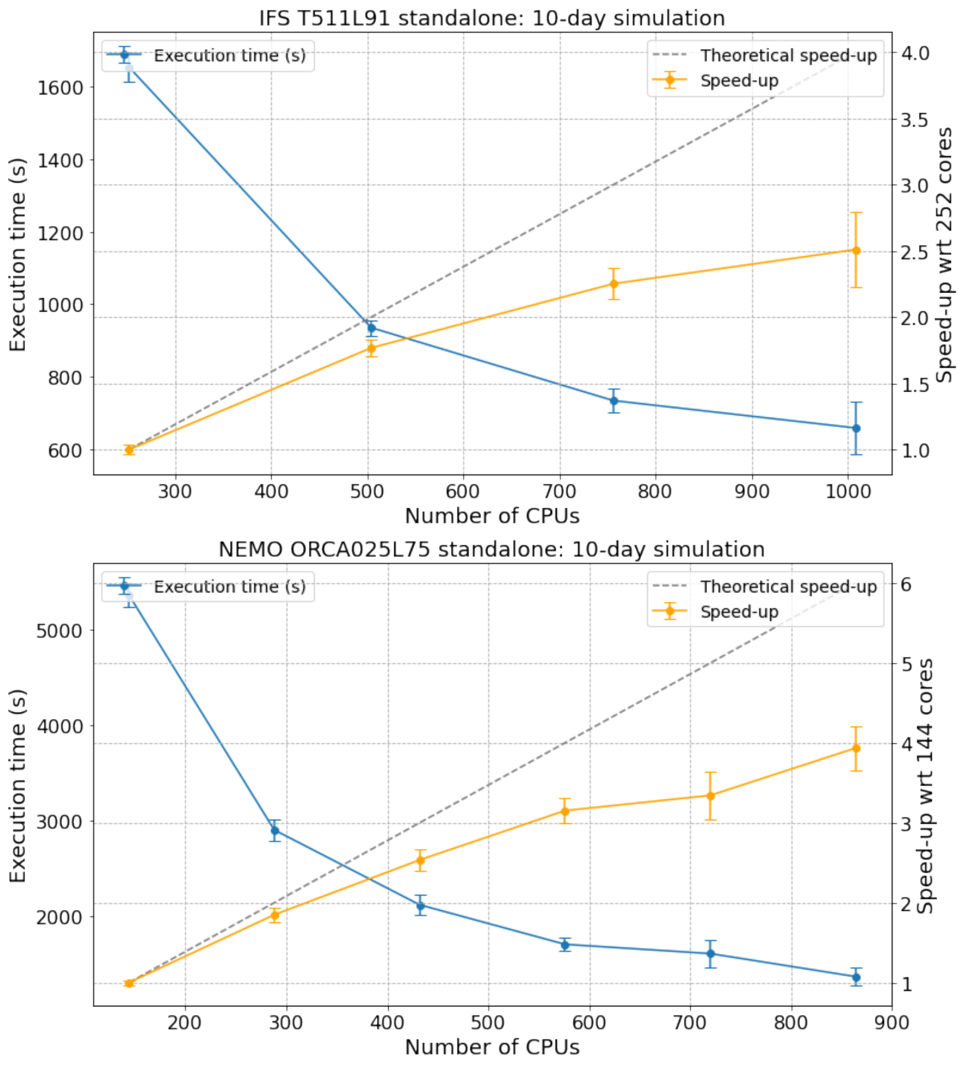


Figure 9.1: Wall clock time scalability of IFS- and NEMO-only simulation. The plotted scalabilities are averages computered from a sample of five 10-day simulation. The scalabilities are measured with respect to execution time using 252 (IFs-only) and 144 CPUs (NEMO-only). In terms of wall clock time, NEMO-only is observed to be scaling more efficiently than IFs-only.

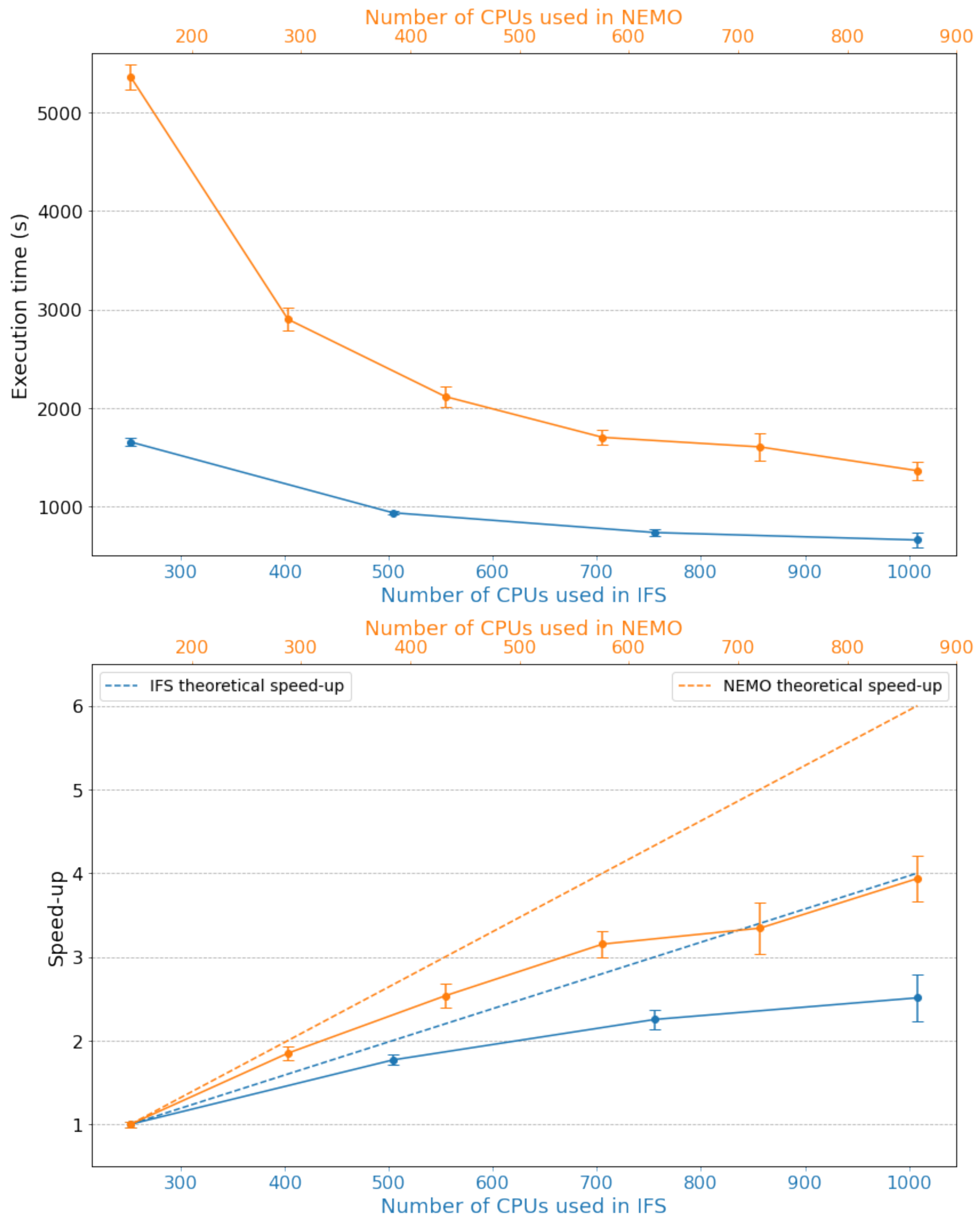


Figure 9.2: Comparing wall clock time scalability of IFs- and NEMO-only on the same plot. Measurements of IFS are labeled in blue whilst those of NEMO are labeled in orange. In these plots, it is clear that NEMO is observed to be scaling better than IFS in terms of wall clock time.

9.2 Additional tables

Tasks	Wall clock time (%)	CPU time (%)
Create sea-ice masks	4.81542098e-02	4.99141235e-02
MPI communication	5.80963600e-02	6.40500088e-02
Compute for τ_a & τ_w	3.58018069e-01	3.59027628e-01
Iterating EVP algorithm	8.76040085e+01	8.71896374e+01
MPI communications w/in EVP	7.79607150e+00	8.21338259e+00
Recompute Δ	2.13091231e-01	2.16830343e-01
Gather diagnostics	3.97022167e-01	3.97641081e-01
MPI communication in gathering diagnostics	9.11134942e-01	9.09126754e-01
Others	2.61440306e+00	2.60039003e+00

Table 9.1: Measured average CPU and wall clock time percentage of each group of tasks in *limrhg.f90*. The averages are calculated over a sample of ten 10-day NEMO-standalone simulations in high resolution configuration. This is the data set plotted in Figure 6.5 in Chapter 6.2.2.

Section	slope (m)	intercept (b)	slope error	4n-elap.time(%)
limdyn	0.3688807349618543	0.48903444041116106	0.03027120872756227	9.409
tra_adv_tvd	0.4080740308847504	0.2266793898645334	0.02919325364059388	3.586
zdf_tmx	0.46531866978775144	0.10026616418017742	0.0066527520371656435	3.311
limthd	0.45006027675451366	0.17030791225563746	0.007490085906125674	3.286
nonosc	0.45010645655457293	0.19183318973212682	0.021014948677265367	3.024
ldf_slp	0.4490010087653073	0.19903620163766167	0.023531409192176472	2.760
dom_vvl_interpol	0.41941330879197974	0.19154888449681362	0.01869101557649952	2.707
dom_vvl_sf_swp	0.4714823536934848	0.24252790351512665	0.0391814530599681	2.274
dyn_nxt	0.40514140735128396	0.13673471828551298	0.036182971595338904	2.234
zdf_ddm	0.4065562755962135	0.3259698984250523	0.014904315953617922	1.613
tke_avn	0.44556266320605575	0.09495907171661155	0.01826976086315192	1.599
dom_vvl_sf_nxt	0.424928889320912	0.2220436528925851	0.013512790423633736	1.395
dia_ar5	0.40296892641711857	0.3360388285730229	0.011458632831667438	1.014
limupdate2	0.4645228239033432	0.23938535906989333	0.016130969611116708	0.954
tra_nxt	0.3236263326876192	0.20180989703675012	0.08546381720991837	0.954
zdf_evd	0.3237108070397244	0.4404740074509155	0.03972942578817702	0.569
rab_3d	0.4357741997117707	0.1869512553447743	0.016006236528277516	0.447
eos-pot	0.46695587063196653	0.08906753383161092	0.0032293135661365926	0.438
wzv	0.4816009630550083	0.08052152610617291	0.003760542624464019	0.412
sbc	0.152210046573115	1.8551005087562613	0.10452036373068674	0.255
zps_hde	0.349736434041953	0.5529365278758616	0.08726650278464375	0.197
tra_bbl	0.37238117718370173	0.34333901113574505	0.061294876100033154	0.200
ldf_eiv	0.4088206381806604	0.39911396367583274	0.015071237765119717	0.182
eos-insitu	0.4351906434301559	0.1803059608753168	0.01874209769926132	0.136
eos_pt_from_ct_3d	0.48835922940851634	0.006235964362717894	0.004665854374558751	0.120
ldf_slp_mxl	0.2834943544400375	0.6351040105752506	0.022414262194953384	0.110
zdf_mxl	0.4582138744712392	0.12245465911561038	0.00822355246016425	0.102
limwri	0.3677612125958831	0.5726508610167111	0.017658279145124106	0.058
dta_tsd	0.049042318918080516	2.1392309881614406	0.06621948276865824	0.062
blk_oce_core	0.40079647945532376	0.3897722458419661	0.0712909104979446	0.056
lim_diahsb	0.1514005890240254	0.9068832367188191	0.014716864927607093	0.050
dom_hgr	0.04297883300990102	2.1542013013858368	0.06843426595060499	0.043
eos2d	0.07878263983275065	0.8584107020654218	0.03248113312389073	0.020
bbl	0.4391087020369736	0.14606135826219724	0.006772638434543726	0.010
zdf_bfr	0.2344704173505147	0.06897524429694357	0.04978750233593961	0.011
dyn_bfr	0.4665728818919912	0.07294142625372224	0.009515103197088462	0.010
sbc_dcy	0.440710415601966	0.11891534818876481	0.0046454491800585	0.010
blk_ice_core_tau	0.15055849925524473	0.5597914463601898	0.036656546901904744	0.010
tra_sbc	0.29371372679898855	0.5580137478049525	0.012147999169492052	0.009
sbc_fwb	0.1357708356061614	1.43613228478576	0.10346802092884534	0.000
rab_2d	0.34602622306343206	0.4994850425276973	0.02036092125409238	0.002
dom_ngb	-0.03542077992514939	1.3883649909829991	0.03341477014141735	0.000
istate_init	0.46081296017327056	0.1500789313509463	0.006848357821904733	0.000
zgr_zps	0.20671470050341728	1.1037043441990475	0.04926464530921187	0.001
eos_pt_from_ct_2d	0.4649208822506542	-0.18907090176311359	0.01651322865661005	0.000
dyn_adv	0.01030327415510345	0.9470008085166807	0.003715844532913308	0.000
dom_msk	0.2003058294252502	0.7524131997481383	0.06706867360323954	0.000
dyn_vor	0.022811336048011506	1.0344698758223898	0.004792837616722159	0.000
dyn_zdf	0.012397115914752708	0.9831604185678158	0.0013570720804919085	0.000
dom_vvl_init	0.43337633027277095	0.2613757595541366	0.009910851675815359	0.000
dyn_ldf	0.008214649664105374	0.9598989132259559	0.005208306313965356	0.000
dom_init	0.08874859801097083	2.0704427585346803	0.08501030568455553	0.000
dyn_spg	0.01688624115297397	0.974611870975622	0.003968583349821347	0.000
tra_qsr_init	0.43169771364744314	0.22886100734187131	0.008787492015827505	0.000
day	-0.03915145970369992	0.8745638632800291	0.027670429315816748	0.000
tra_bbl_init	-0.023545881049454532	1.0670138011373116	0.026893528362743883	0.000
ldf_slp_init	0.47938349306163663	0.04016145134858551	0.009945839084364073	0.000
dia_ar5_init	0.3776630776755514	-0.14801371341273528	0.06762360166871728	0.000
zgr_bat_ctl	-0.016052984707233677	6.3910819968827886	0.24235972336085862	0.000
zdf_bfr_init	0.07802812619625149	0.5289137324888178	0.050834058227277636	0.000
dyn_spg_init	-0.012500346988890096	0.9211499632814194	0.033274721006477485	0.000
zgr_top_level	0.0654741006863068	1.0772909792083132	0.09222939038625987	0.000
ice_lim_flx	0.08531922373404187	0.9034484519347666	0.009180277840116064	0.000
zgr_bot_level	0.07697877920436491	0.9815535537918522	0.0821038228173318	0.000
dta_tsd_init	0.014115282949239213	1.0613994815526113	0.004221910366257674	0.000
dom_zgr	-0.00527104369403528	0.9644433966879677	0.005920212427468085	0.000
zgr_z	0.031969508163829014	0.9812462268378234	0.010525627576497555	0.000
dom_cfg	0.24503958414832777	0.673873594822318	0.014924571294851745	0.000

Table 9.2: NEMO subroutines with wall clock scalability slope of less than 0.5. The subroutines are ordered from high to low wall clock time percentage each subroutine takes up of the a NEMO-ORCA025 standalone simulation. Data provided here is plotted in Figure 6.4a.

Section	slope (m)	intercept (b)	slope error	4n-cputime(%)
limdyn	0.3688635401887801	0.48907296418317436	0.030269566195719275	9.690
tra_adv_tvd	0.40813076310406915	0.2264688018902583	0.029203156718795087	3.693
zdf_tmx	0.4652968445997492	0.1003423289950689	0.006654714620784975	3.411
limthd	0.45002750083581783	0.1703958462342161	0.00748870385108864	3.384
nonosc	0.44998747520836785	0.19224090609883326	0.02100476811994398	3.118
ldf_slp	0.4490685959209471	0.1987252811540623	0.023526554818372424	2.844
dom_vvl_interpol	0.41844570762605704	0.19505276837287866	0.018579847185964057	2.789
dom_vvl_sf_swp	0.48426020663049113	0.0970649147772642	0.0038404692080961766	2.341
dyn_nxt	0.4050705950858999	0.13704839126544544	0.03616161994874835	2.302
zdf_ddm	0.4064827400745136	0.3261935692731983	0.014917686323673742	1.659
tke_avn	0.4455325324834473	0.09496917446928776	0.0182796040090466	1.646
dom_vvl_sf_nxt	0.42476127563138855	0.22259069776417695	0.013509670463814669	1.435
tra_qsr	0.4860271194153389	-0.003809544773058171	0.010190091179242456	1.306
dia_ar5	0.40310736608072695	0.33554742601180276	0.011462554721737631	1.043
limupdate2	0.46440435574860683	0.23968192161502788	0.01613252108614412	0.983
tra_nxt	0.32350294184965545	0.20240972762165743	0.0854272366778598	0.982
sbc_ice_lim	0.49850825862874565	-0.3931184381720323	0.07102949761092796	0.440
zdf_evd	0.3233011014514409	0.44217508349252377	0.03969465129882691	0.588
rab_3d	0.43517836160724516	0.189052609891474	0.015981247300462735	0.458
eos-pot	0.46616298172852355	0.09165673728360524	0.0031603831061132984	0.451
wzv	0.48078515059049337	0.08328693290234046	0.0038308494117456535	0.425
sbc	0.23170232662883944	0.36502775752778405	0.03335097816348135	0.262
zps_hde	0.34997617374774903	0.552049773591746	0.08737602092828141	0.205
tra_bbl	0.3726294436268529	0.3428895905746816	0.061361302829702084	0.208
ldf_eiv	0.4085146418806407	0.4004042113068964	0.01507208014539027	0.186
eos-insitu	0.43429967293053257	0.1843507252955945	0.01878361319649657	0.139
eos_pt_from_ct_3d	0.4856741471489371	0.017384342092753968	0.004233876915474066	0.124
ldf_slp_mxl	0.2825401970019426	0.6381676472938733	0.022282589975981506	0.117
zdf_mxl	0.4575105003785958	0.12290107022667751	0.007976189824759533	0.108
limwri	0.3661363816108928	0.5772492545592018	0.01756621162433836	0.060
dta_tsd	0.098797209884286	1.275537744267317	0.023664657161216004	0.063
blk_oce_core	0.40130527343188804	0.38427437890949845	0.07102362601163008	0.057
lim_diahsb	0.15296987288535113	0.9244020013433958	0.017424202062268256	0.050
dom_hgr	0.047883253290406186	1.5257487762392499	0.04665279872355748	0.043
zdf_tmx_init	0.098023867864261	0.9195013116641073	0.067230205145723	0.020
eos2d	0.0784282874114051	0.8601672679878029	0.03229214238326283	0.020
bbl	0.43967706380395843	0.14495556135752263	0.00727235588289295	0.012
zdf_bfr	0.2340693626082678	0.06939176471236363	0.04975071082565362	0.011
dyn_bfr	0.45165720952992383	0.13740060769307139	0.009576285349054577	0.010
sbc_dcy	0.430024596786778	0.15360558645766442	0.004181744199008306	0.010
blk_ice_core_tau	0.149190927883186	0.5605627841395806	0.03618657018127851	0.010
tra_sbc	0.28793138526432904	0.5616981846664197	0.012592332183397769	0.002
sbc_fwb	0.30942802868146396	0.9239698431504535	0.1775484313872046	0.000
rab_2d	0.33864412602909094	0.48984544834853505	0.0266359382782776	0.001
dom_ngb	-0.03603319547055498	1.3960515708185808	0.03342285944027699	0.000
istate_init	0.4640052816901408	0.0971302816901396	0.0352138280319078	0.000
zgr_zps	0.20638872230130487	1.1096823341750577	0.047990460286614356	0.001
eos_pt_from_ct_2d	0.41167371553884713	0.060154593213800323	0.032501075811424016	0.000
dyn_adv	0.013790524249015041	0.9324851606824917	0.004306009860964851	0.000
dom_msk	0.15987886382623223	1.0325814536340854	0.05793060612614594	0.000
dyn_vor	0.018099877250278073	1.0336883619310184	0.006954586999981552	0.000
dyn_zdf	0.017054512967053792	0.9535263360557726	0.0021892484877831814	0.000
dom_vvl_init	0.398809523809524	0.5357142857142847	0.06420114955302948	0.000
dyn_ldf	0.0030083507762079234	0.9919839089481949	0.0013712354510533303	0.000
dom_init	0.06547619047619054	2.285714285714285	0.08618970884513286	0.000
dyn_spg	0.016042780748663072	1.0276292335115862	0.003825370231619843	0.000
zgr_bat	0.20446593337218338	3.7123171620046627	0.16147339364487548	0.000
day	0.02258125472411185	0.9614512471655329	0.01863872180000812	0.000
tra_bbl_init	-0.0185806108801517	1.059982428968653	0.028337845756879025	0.000
zgr_bat_ctl	-0.01668719211822665	7.214408866995075	0.30077792707318624	0.000
zdf_bfr_init	0.0	0.0	0.0	0.000
dyn_spg_init	0.0	0.0	0.0	0.000
zgr_top_level	0.0	0.0	0.0	0.000
ice_lim_flx	0.0	0.0	0.0	0.000
zgr_bot_level	0.0	0.0	0.0	0.000
dta_tsd_init	0.0	0.0	0.0	0.000
dom_zgr	0.0	0.0	0.0	0.000
zgr_z	0.0	0.0	0.0	0.000
dom_cfg	0.0	0.0	0.0	0.000

Table 9.3: NEMO subroutines with CPU time scalability slope of less than 0.5. The subroutines are ordered from high to low CPU time percentage each subroutine takes up of the a NEMO-ORCA025 standalone simulation. Data provided here is plotted in Figure 6.4b.

9.3 Vectorization optimization report: *limrhg.optrpt* (shortened)

Here is the shortened vectorization optimization report presented in Chapter 6.3.1 and discussed in Chapter 7.4. Only information related to the EVP framework in `limrhg.f90` and details on all detected non-optimizable loops are presented here.

```
Intel(R) Advisor can now assist with vectorization and show optimization
  report messages with your source code.
See "https://software.intel.com/en-us/intel-advisor-xe" for details.


Begin optimization report for: LIMRHG::LIM_RHG

    Report from: Vector optimizations [vec]


LOOP BEGIN at /home/ngyilo/ec_earth3/nocray-compile-v3.3.3.2/sources/nemo-3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(369,16)
<Peeled loop for vectorization, Multiversioned v1>
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray-compile-v3.3.3.2/sources/nemo-3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(369,16)
<Multiversioned v1>
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray-compile-v3.3.3.2/sources/nemo-3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(369,16)
<Remainder loop for vectorization, Multiversioned v1>
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray-compile-v3.3.3.2/sources/nemo-3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(369,16)
<Peeled loop for vectorization, Multiversioned v2>
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray-compile-v3.3.3.2/sources/nemo-3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(369,16)
<Multiversioned v2>
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray-compile-v3.3.3.2/sources/nemo-3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(369,16)
<Remainder loop for vectorization, Multiversioned v2>
LOOP END
LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo −3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(370,16)
<Peeled loop for vectorization, Multiversioned v1>
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo −3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(370,16)
<Multiversioned v1>
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo −3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(370,16)
<Remainder loop for vectorization, Multiversioned v1>
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo −3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(370,16)
<Peeled loop for vectorization, Multiversioned v2>
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo −3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(370,16)
<Multiversioned v2>
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo −3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(370,16)
<Remainder loop for vectorization, Multiversioned v2>
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo −3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(376,13)
   remark #15344: loop was not vectorized: vector dependence prevents ↩
       vectorization. First dependence is shown below. Use level 5 report for ↩
       details
   remark #15346: vector dependence: assumed OUTPUT dependence between call:↩
       for_emit_diagnostic (379:33) and call:for_emit_diagnostic (379:16)
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo −3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(385,15)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo −3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(385,15)
   <Peeled loop for vectorization, Multiversioned v1>
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo −3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(385,15)
   <Multiversioned v1>
      remark #15300: LOOP WAS VECTORIZED
   LOOP END
   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(385,15)
   <Remainder loop for vectorization, Multiversioned v1>
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(385,15)
   <Peeled loop for vectorization, Multiversioned v2>
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(385,15)
   <Multiversioned v2>
      remark #15300: LOOP WAS VECTORIZED
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(385,15)
   <Remainder loop for vectorization, Multiversioned v2>
   LOOP END
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(419,15)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(419,15)
   <Peeled loop for vectorization, Multiversioned v1>
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(419,15)
   <Multiversioned v1>
      remark #15300: LOOP WAS VECTORIZED
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(419,15)
   <Remainder loop for vectorization, Multiversioned v1>
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(419,15)
   <Peeled loop for vectorization, Multiversioned v2>
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(419,15)
   <Multiversioned v2>
      remark #15300: LOOP WAS VECTORIZED
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(419,15)
   <Remainder loop for vectorization, Multiversioned v2>
   LOOP END
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(421,10)
   remark #15541: outer loop was not auto−vectorized: consider using SIMD ↩
       directive

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(422,13)
      remark #15344: loop was not vectorized: vector dependence prevents ↩
          vectorization. First dependence is shown below. Use level 5 report for ↩
          details
      remark #15346: vector dependence: assumed OUTPUT dependence between call:↩
          for_emit_diagnostic (425:38) and call:for_emit_diagnostic (428:16)
   LOOP END
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(596,16)
<Multiversioned v1>
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(596,16)
<Remainder loop for vectorization, Multiversioned v1>
   remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(596,16)
<Remainder loop for vectorization, Multiversioned v1>
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(596,16)
<Multiversioned v2>
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(596,16)
<Remainder loop for vectorization, Multiversioned v2>
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(596,16)
<Multiversioned v1>
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(596,16)
<Remainder loop for vectorization, Multiversioned v1>
LOOP END
LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(596,16)
<Multiversioned v2>
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(596,16)
<Remainder loop for vectorization, Multiversioned v2>
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(598,21)
   remark #15541: outer loop was not auto−vectorized: consider using SIMD ↩
       directive

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(598,21)
      remark #15331: loop was not vectorized: precise FP model implied by the ↩
          command line or a directive prevents vectorization. Consider using fast↩
           FP model
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(598,21)
   <Remainder>
   LOOP END
LOOP END



Non−optimizable loops:


LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(222,21)
   remark #15536: loop was not vectorized: inner loop throttling prevents ↩
       vectorization of this outer loop. Refer to inner loop message for more ↩
       details.   [ /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo↩
       −3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(222,21) ]

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(222,21)
      remark #15523: loop was not vectorized: loop control variable ? was found, ↩
          but loop iteration count cannot be computed before executing the loop
   LOOP END
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(249,21)
   remark #15536: loop was not vectorized: inner loop throttling prevents ↩
       vectorization of this outer loop. Refer to inner loop message for more ↩
       details.   [ /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo↩
       −3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(249,21) ]

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(249,21)
      remark #15523: loop was not vectorized: loop control variable ? was found, ↩
          but loop iteration count cannot be computed before executing the loop
   LOOP END
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(356,7)
   remark #15532: loop was not vectorized: compile time constraints prevent loop ↩
       optimization. Consider using −O3.

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(355,10)
      remark #15532: loop was not vectorized: compile time constraints prevent ↩
          loop optimization. Consider using −O3.
   LOOP END
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(358,21)
   remark #15536: loop was not vectorized: inner loop throttling prevents ↩
       vectorization of this outer loop. Refer to inner loop message for more ↩
       details.   [ /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo↩
       −3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(358,21) ]

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(358,21)
      remark #15523: loop was not vectorized: loop control variable ? was found, ↩
          but loop iteration count cannot be computed before executing the loop
   LOOP END
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(603,7)
   remark #15543: loop was not vectorized: loop with function call not considered↩
        an optimization candidate.

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(371,13)
      remark #15532: loop was not vectorized: compile time constraints prevent ↩
          loop optimization. Consider using −O3.
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(384,10)
      remark #15532: loop was not vectorized: compile time constraints prevent ↩
          loop optimization. Consider using −O3.
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(385,24)
      remark #15536: loop was not vectorized: inner loop throttling prevents ↩
          vectorization of this outer loop. Refer to inner loop message for more ↩
          details.   [ /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/↩
          nemo−3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90↩
          (385,24) ]
      LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo↩
          −3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90↩
          (385,24)
         remark #15523: loop was not vectorized: loop control variable ? was ↩
             found, but loop iteration count cannot be computed before executing ↩
             the loop
      LOOP END
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(418,10)
      remark #15532: loop was not vectorized: compile time constraints prevent ↩
          loop optimization. Consider using −O3.

      LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo↩
          −3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90↩
          (417,13)
         remark #15532: loop was not vectorized: compile time constraints prevent↩
              loop optimization. Consider using −O3.
      LOOP END
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(419,24)
      remark #15536: loop was not vectorized: inner loop throttling prevents ↩
          vectorization of this outer loop. Refer to inner loop message for more ↩
          details.   [ /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/↩
          nemo−3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90↩
          (419,24) ]

      LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo↩
          −3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90↩
          (419,24)
         remark #15523: loop was not vectorized: loop control variable ? was ↩
             found, but loop iteration count cannot be computed before executing ↩
             the loop
      LOOP END
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(463,10)
      remark #15532: loop was not vectorized: compile time constraints prevent ↩
          loop optimization. Consider using −O3.

      LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo↩
          −3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90↩
          (462,13)
         remark #15532: loop was not vectorized: compile time constraints prevent↩
              loop optimization. Consider using −O3.
      LOOP END
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(494,13)
      remark #15532: loop was not vectorized: compile time constraints prevent ↩
          loop optimization. Consider using −O3.
      LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo↩
          −3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90↩
          (493,16)
         remark #15532: loop was not vectorized: compile time constraints prevent↩
              loop optimization. Consider using −O3.
      LOOP END
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(524,13)
      remark #15532: loop was not vectorized: compile time constraints prevent ↩
          loop optimization. Consider using −O3.

      LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo↩
          −3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90↩
          (523,16)
         remark #15532: loop was not vectorized: compile time constraints prevent↩
              loop optimization. Consider using −O3.
      LOOP END
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(556,13)
      remark #15532: loop was not vectorized: compile time constraints prevent ↩
          loop optimization. Consider using −O3.

      LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo↩
          −3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90↩
          (555,16)
         remark #15532: loop was not vectorized: compile time constraints prevent↩
              loop optimization. Consider using −O3.
      LOOP END
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(586,13)
      remark #15532: loop was not vectorized: compile time constraints prevent ↩
          loop optimization. Consider using −O3.

      LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo↩
          −3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90↩
          (585,16)
         remark #15532: loop was not vectorized: compile time constraints prevent↩
              loop optimization. Consider using −O3.
      LOOP END
   LOOP END

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(597,13)
      remark #15532: loop was not vectorized: compile time constraints prevent ↩
          loop optimization. Consider using −O3.
   LOOP END
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(618,7)
```

```
   remark #15532: loop was not vectorized: compile time constraints prevent loop ↩
       optimization. Consider using −O3.
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(619,21)
   remark #15536: loop was not vectorized: inner loop throttling prevents ↩
       vectorization of this outer loop. Refer to inner loop message for more ↩
       details.   [ /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo↩
       −3.6/CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(619,21) ]

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(619,21)
      remark #15523: loop was not vectorized: loop control variable ? was found, ↩
          but loop iteration count cannot be computed before executing the loop
   LOOP END
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(649,7)
   remark #15532: loop was not vectorized: compile time constraints prevent loop ↩
       optimization. Consider using −O3.

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(648,10)
      remark #15532: loop was not vectorized: compile time constraints prevent ↩
          loop optimization. Consider using −O3.
   LOOP END
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(703,7)
   remark #15532: loop was not vectorized: compile time constraints prevent loop ↩
       optimization. Consider using −O3.

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(702,10)
      remark #15532: loop was not vectorized: compile time constraints prevent ↩
          loop optimization. Consider using −O3.
   LOOP END
LOOP END

LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
    CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(774,13)
   remark #15543: loop was not vectorized: loop with function call not considered↩
        an optimization candidate.

   LOOP BEGIN at /home/ngyilo/ec_earth3/nocray−compile−v3.3.3.2/sources/nemo−3.6/↩
       CONFIG/ORCA025L75_LIM3_standalone/BLD/ppsrc/nemo/limrhg.f90(773,16)
      remark #15543: loop was not vectorized: loop with function call not ↩
          considered an optimization candidate.
   LOOP END
LOOP END
Optimization 'PRE' reduced: function size or variable count limit exceeded: use −↩
    override_limits / −Qoverride_limits to override
===========================================================================
```

9.4 Sea ice velocity subroutine: *limrhg.f90*

Full script of the subroutine `limrhg.f90` in NEMO3.6 used to compute for the sea-ice rheology and velocities using the Elastic Viscous Plastic (EVP) framework (Bouillon *et al.*, 2009).

```fortran
MODULE limrhg
!=====================================================================
!!                     ***  MODULE  limrhg  ***
!!   Ice rheology : sea ice rheology
!=====================================================================
!! History :   -   !  2007-03  (M.A. Morales Maqueda, S. Bouillon) Original code
!!            3.0  !  2008-03  (M. Vancoppenolle) LIM3
!!             -   !  2008-11  (M. Vancoppenolle, S. Bouillon, Y. Aksenov) add surface tilt in ice rheolohy
!!            3.3  !  2009-05  (G.Garric) addition of the lim2_evp cas
!!            3.4  !  2011-01  (A. Porter)  dynamical allocation
!!            3.5  !  2012-08  (R. Benshila)  AGRIF
!!            3.6  !  2016-06  (C. Rousset) Rewriting (conserves energy)
!---------------------------------------------------------------------

```

```
!↩
   !----------------------------------------------------------------------
!!   'key_lim3'               OR ↩
                    LIM-3 sea-ice model
!!   'key_lim2' AND NOT 'key_lim2_vp'           EVP↩
    LIM-2 sea-ice model
!↩
   !----------------------------------------------------------------------
!!   lim_rhg       : computes ice velocities
!↩
   !----------------------------------------------------------------------
   USE phycst          ! Physical constant
   USE oce      , ONLY :  snwice_mass, snwice_mass_b
   USE par_oce         ! Ocean parameters
   USE dom_oce         ! Ocean domain
   USE sbc_oce         ! Surface boundary condition: ↩
      ocean fields
   USE sbc_ice         ! Surface boundary condition: ↩
      ice fields

   USE ice             ! LIM-3: ice variables
   USE dom_ice         ! LIM-3: ice domain
   USE limitd_me       ! LIM-3:

   USE lbclnk          ! Lateral Boundary Condition /↩
       MPP link
   USE lib_mpp         ! MPP library
   USE wrk_nemo        ! work arrays
   USE in_out_manager  ! I/O manager
   USE prtctl          ! Print control
   USE iom
   USE lib_fortran     ! Fortran utilities (allows no↩
       signed zero when 'key_nosignedzero' defined)


   IMPLICIT NONE
   PRIVATE

   PUBLIC   lim_rhg         ! routine called by ↩
      lim_dyn (or lim_dyn_2)

!! * Substitutions
!↩
   !------------------------------------------------------------------

!!                   ***  vectopt_loop_substitute  ↩
   ***
!↩
   !------------------------------------------------------------------

!! ** purpose :   substitute the inner loop start/↩
   end indices with CPP macro
!!                allow unrolling of do-loop (useful↩
    with vector processors)
!↩
   !------------------------------------------------------------------

!↩
   !------------------------------------------------------------------

!! NEMO/OPA 3.7 , NEMO Consortium (2014)
!! $Id: vectopt_loop_substitute.h90 4990 2014-12-15 ↩
   16:42:49Z timgraham $
!! Software governed by the CeCILL licence (NEMOGCM/↩
   NEMO_CeCILL.txt)
!↩
   !------------------------------------------------------------------
```

```
!!----------------------------------------------------------------------
!! NEMO/LIM3 4.0 , UCL - NEMO Consortium (2011)
!! $Id: limrhg.F90 8285 2017-07-06 06:40:51Z vancop $
!! Software governed by the CeCILL licence     (NEMOGCM/NEMO_CeCILL.txt)
!!----------------------------------------------------------------------
CONTAINS

   SUBROUTINE lim_rhg( k_j1, k_jpj )
!!----------------------------------------------------------------------
!!                 ***  SUBROUTINE lim_rhg  ***
!!                          EVP-C-grid
!!
!! ** purpose : determines sea ice drift from wind stress, ice-ocean
!!  stress and sea-surface slope. Ice-ice interaction is described by
!!  a non-linear elasto-viscous-plastic (EVP) law including shear
!!  strength and a bulk rheology (Hunke and Dukowicz, 2002).
!!
!!  The points in the C-grid look like this, dear reader
!!
!!                              (ji,jj)
!!                                 |
!!                                 |
!!                      (ji-1,jj)  |  (ji,jj)
!!                             ---------
!!                            |         |
!!                            | (ji,jj) |------(ji,jj)
!!                            |         |
!!                             ---------
!!                     (ji-1,jj-1)     (ji,jj-1)
!!
!! ** Inputs  : - wind forcing (stress), oceanic currents
!!                ice total volume (vt_i) per unit area
!!                snow total volume (vt_s) per unit area
!!
!! ** Action  : - compute u_ice, v_ice : the components of the
!!                sea-ice velocity vector
!!              - compute delta_i, shear_i, divu_i, which are inputs
!!                of the ice thickness distribution
!!
!! ** Steps   : 1) Compute ice snow mass, ice strength
!!              2) Compute wind, oceanic stresses, mass terms and
!!                 coriolis terms of the momentum equation
!!              3) Solve the momentum equation (iterative procedure)
!!              4) Recompute invariants of the strain rate tensor
!!                 which are inputs of the ITD, store stress
!!                 for the next time step
!!              5) Control prints of residual (convergence)
!!                 and charge ellipse.
!!                 The user should make sure that the parameters
!!                 nn_nevp, elastic time scale and rn_creepl maintain stress state
!!                 on the charge ellipse for plastic flow
!!                 e.g. in the Canadian Archipelago
!!
!! ** Notes    : Boundary condition for ice is chosen no-slip
!!               but can be adjusted with param rn_shlat
!!
!! References : Hunke and Dukowicz, JPO97
!!               Bouillon et al., Ocean Modelling 2009
!!------------------------------------------------------------------
      INTEGER, INTENT(in) ::   k_j1    ! southern j-index for ice computation
      INTEGER, INTENT(in) ::   k_jpj   ! northern j-index for ice computation
!!
      INTEGER ::   ji, jj    ! dummy loop indices
      INTEGER ::   jter      ! local integers
      CHARACTER (len=50) ::   charout

      REAL(wp) ::   zdtevp, z1_dtevp                              !   time step for subcycling
      REAL(wp) ::   ecc2, z1_ecc2                                   ! square of yield ellipse eccenticity
      REAL(wp) ::   zbeta, zalph1, z1_alph1, zalph2, z1_alph2               ! alpha and beta from Bouillon 2009 and 2013
      REAL(wp) ::   zm1, zm2, zm3, zmassU, zmassV             ! ice/snow mass
      REAL(wp) ::   zdelta, zp_delf, zds2, zdt, zdt2, zdiv, zdiv2            ! temporary scalars
      REAL(wp) ::   zTau0, zTauE                                             ! temporary scalars

      REAL(wp) ::   zsig1, zsig2                                             ! internal ice stress
      REAL(wp) ::   zresm                                                    ! Maximal error on ice velocity
      REAL(wp) ::   zintb, zintn                                             ! dummy argument
      REAL(wp) ::   zfac_x, zfac_y

      REAL(wp), POINTER, DIMENSION(:,:) ::   zpresh                          ! temporary array for ice strength
      REAL(wp), POINTER, DIMENSION(:,:) ::   z1_e1t0, z1_e2t0                ! scale factors
      REAL(wp), POINTER, DIMENSION(:,:) ::   zp_delt                         ! P/delta at T points
!
      REAL(wp), POINTER, DIMENSION(:,:) ::   zaU   , zaV                     ! ice fraction on U/V points
      REAL(wp), POINTER, DIMENSION(:,:) ::   zmU_t, zmV_t                    ! ice/snow mass/dt on U/V points
      REAL(wp), POINTER, DIMENSION(:,:) ::   zmf                             ! coriolis parameter at T points
      REAL(wp), POINTER, DIMENSION(:,:) ::   zTauU_ia , ztauV_ia            ! ice-atm. stress at U-V points
      REAL(wp), POINTER, DIMENSION(:,:) ::   zspgU , zspgV                  ! surface pressure gradient at U/V points
      REAL(wp), POINTER, DIMENSION(:,:) ::   v_oceU, u_oceV, v_iceU, u_iceV  ! ocean/ice u/v component on V/U points
      REAL(wp), POINTER, DIMENSION(:,:) ::   zfU   , zfV                    ! internal stresses

      REAL(wp), POINTER, DIMENSION(:,:) ::   zds                            ! shear
      REAL(wp), POINTER, DIMENSION(:,:) ::   zs1, zs2, zs12                 ! stress tensor components
      REAL(wp), POINTER, DIMENSION(:,:) ::   zu_ice, zv_ice, zresr          ! check convergence
      REAL(wp), POINTER, DIMENSION(:,:) ::   zpice                          ! array used for the calculation of ice surface slope:
!    ocean surface (ssh_m) if ice is not embedded
!    ice top surface if ice is embedded
      REAL(wp), POINTER, DIMENSION(:,:) ::   zCorx, zCory                   ! Coriolis stress array
      REAL(wp), POINTER, DIMENSION(:,:) ::   ztaux_oi, ztauy_oi             ! Ocean-to-ice stress array

      REAL(wp), POINTER, DIMENSION(:,:) ::   zswitchU, zswitchV             ! dummy arrays
```

```fortran
      REAL(wp), POINTER, DIMENSION(:,:) ::   zmaskU, zmaskV            ! mask for ice presence
      REAL(wp), POINTER, DIMENSION(:,:) ::   zfmask, zwf               ! mask at F points for the ice

      REAL(wp), POINTER, DIMENSION(:,:) ::   zdiag_xmtrp_ice   ! X-component of ice mass transport  (kg/s)
      REAL(wp), POINTER, DIMENSION(:,:) ::   zdiag_ymtrp_ice   ! Y-component of ice mass transport  (kg/s)
      REAL(wp), POINTER, DIMENSION(:,:) ::   zdiag_xmtrp_snw   ! X-component of snow mass transport (kg/s)
      REAL(wp), POINTER, DIMENSION(:,:) ::   zdiag_ymtrp_snw   ! Y-component of snow mass transport (kg/s)
      REAL(wp), POINTER, DIMENSION(:,:) ::   zdiag_xatrp       ! X-component of area transport (m2/s)
      REAL(wp), POINTER, DIMENSION(:,:) ::   zdiag_yatrp       ! Y-component of area transport (m2/s)
      REAL(wp), POINTER, DIMENSION(:,:) ::   zdiag_utau_oi     ! X-direction ocean-ice stress
      REAL(wp), POINTER, DIMENSION(:,:) ::   zdiag_vtau_oi     ! Y-direction ocean-ice stress
      REAL(wp), POINTER, DIMENSION(:,:) ::   zdiag_dssh_dx     ! X-direction sea-surface tilt term (N/m2)
      REAL(wp), POINTER, DIMENSION(:,:) ::   zdiag_dssh_dy     ! X-direction sea-surface tilt term (N/m2)
```

```
   REAL(wp), POINTER, DIMENSION(:,:) ::   ↩
      zdiag_corstrx                     ! X-↩
      direction coriolis stress (N/m2)
   REAL(wp), POINTER, DIMENSION(:,:) ::   ↩
      zdiag_corstry                     ! Y-↩
      direction coriolis stress (N/m2)
   REAL(wp), POINTER, DIMENSION(:,:) ::   ↩
      zdiag_intstrx                     ! X-↩
      direction internal stress (N/m2)
   REAL(wp), POINTER, DIMENSION(:,:) ::   ↩
      zdiag_intstry                     ! Y-↩
      direction internal stress (N/m2)
   REAL(wp), POINTER, DIMENSION(:,:) ::   ↩
      zdiag_sig1                        ! Average ↩
      normal stress in sea ice
   REAL(wp), POINTER, DIMENSION(:,:) ::   ↩
      zdiag_sig2                        ! Maximum ↩
      shear stress in sea ice

   REAL(wp), POINTER, DIMENSION(:,:) ::   zswi, ↩
      zmiss                       ! Switch & ↩
      missing value array

   REAL(wp), PARAMETER               ::   zepsi  ↩
      = 1.0e-20_wp               ! tolerance ↩
      parameter
   REAL(wp), PARAMETER               ::   zmmin  ↩
      = 1._wp                    ! ice mass (kg/m2)↩
       below which ice velocity equals ocean ↩
      velocity
   REAL(wp), PARAMETER               ::   zshlat ↩
      = 2._wp                    ! boundary ↩
      condition for sea-ice velocity (2=no slip ;↩
       0=free slip)
   REAL(wp), PARAMETER               ::   ↩
      zmiss_val = 1.0e+20               ! missing ↩
      value for outputs

```

```fortran
!!---------------------------------------------------------

      CALL wrk_alloc( jpi,jpj, zpresh, z1_e1t0, z1_e2t0, zp_delt )
      CALL wrk_alloc( jpi,jpj, zaU, zaV, zmU_t, zmV_t, zmf, zTauU_ia, ztauV_ia )
      CALL wrk_alloc( jpi,jpj, zspgU, zspgV, v_oceU, u_oceV, v_iceU, u_iceV, zfU, zfV )
      CALL wrk_alloc( jpi,jpj, zds, zs1, zs2, zs12, zu_ice, zv_ice, zresr, zpice )
      CALL wrk_alloc( jpi,jpj, zswitchU, zswitchV, zmaskU, zmaskV, zfmask, zwf )
      CALL wrk_alloc( jpi,jpj, zCorx, zCory)
      CALL wrk_alloc( jpi,jpj, ztaux_oi, ztauy_oi)

      CALL wrk_alloc( jpi,jpj, zdiag_xmtrp_ice, zdiag_ymtrp_ice )
      CALL wrk_alloc( jpi,jpj, zdiag_xmtrp_snw, zdiag_ymtrp_snw )
      CALL wrk_alloc( jpi,jpj, zdiag_xatrp     , zdiag_yatrp     )
      CALL wrk_alloc( jpi,jpj, zdiag_utau_oi   , zdiag_vtau_oi   )
      CALL wrk_alloc( jpi,jpj, zdiag_dssh_dx   , zdiag_dssh_dy   )
      CALL wrk_alloc( jpi,jpj, zdiag_corstrx   , zdiag_corstry   )
      CALL wrk_alloc( jpi,jpj, zdiag_intstrx   , zdiag_intstry   )
      CALL wrk_alloc( jpi,jpj, zdiag_sig1      , zdiag_sig2      )
      CALL wrk_alloc( jpi,jpj, zswi            , zmiss           )



```

```
!
!------------------------------------------------------------------
! 0) mask at F points for the ice (on the whole domain, not only k_j1,k_jpj)
!------------------------------------------------------------------
! ocean/land mask
      DO jj = 1, jpjm1
         DO ji = 1, jpim1      ! NO vector opt.
            zfmask(ji,jj) = tmask(ji,jj,1) * tmask(ji+1,jj,1) * tmask(ji,jj+1,1) * tmask(ji+1,jj+1,1)
         END DO
      END DO
      CALL lbc_lnk( zfmask, 'F', 1._wp )

! Lateral boundary conditions on velocity (modify zfmask)
      zwf(:,:) = zfmask(:,:)
      DO jj = 2, jpjm1
         DO ji = 2, jpim1   ! vector opt.
            IF( zfmask(ji,jj) == 0._wp ) THEN
               zfmask(ji,jj) = zshlat * MIN( 1._wp , MAX( zwf(ji+1,jj), zwf(ji,jj+1), zwf(ji-1,jj), zwf(ji,jj-1) ) )
            ENDIF
         END DO
      END DO
      DO jj = 2, jpjm1
         IF( zfmask(1,jj) == 0._wp ) THEN
            zfmask(1  ,jj) = zshlat * MIN( 1._wp , MAX( zwf(2,jj), zwf(1,jj+1), zwf(1,jj-1) ) )
         ENDIF
         IF( zfmask(jpi,jj) == 0._wp ) THEN
```

```
            zfmask(jpi,jj) = zshlat * MIN( 1._wp , MAX( zwf(jpi,jj+1), zwf(jpim1,jj), zwf(jpi,jj-1) ) )
         ENDIF
      END DO
      DO ji = 2, jpim1
         IF( zfmask(ji,1) == 0._wp ) THEN
            zfmask(ji,1  ) = zshlat * MIN( 1._wp , MAX( zwf(ji+1,1), zwf(ji,2), zwf(ji-1,1) ) )
         ENDIF
         IF( zfmask(ji,jpj) == 0._wp ) THEN
            zfmask(ji,jpj) = zshlat * MIN( 1._wp , MAX( zwf(ji+1,jpj), zwf(ji-1,jpj), zwf(ji,jpjm1) ) )
         ENDIF
      END DO
      CALL lbc_lnk( zfmask, 'F', 1._wp )

!--------------------------------------------------------------
! 1) define some variables and initialize arrays
!--------------------------------------------------------------
! ecc2: square of yield ellipse eccenticrity
      ecc2    = rn_ecc * rn_ecc
      z1_ecc2 = 1._wp / ecc2

! Time step for subcycling
      zdtevp    = rdt_ice / REAL( nn_nevp )
      z1_dtevp = 1._wp / zdtevp

! alpha parameters (Bouillon 2009)

      zalph1 = ( 2._wp * rn_relast * rdt_ice ) * z1_dtevp

      zalph2 = zalph1 * z1_ecc2

      z1_alph1 = 1._wp / ( zalph1 + 1._wp )
      z1_alph2 = 1._wp / ( zalph2 + 1._wp )

! Initialise stress tensor
      zs1 (:,:) = stress1_i (:,:)
      zs2 (:,:) = stress2_i (:,:)
      zs12(:,:) = stress12_i(:,:)

! Ice strength

      CALL lim_itd_me_icestrength( nn_icestr )
      zpresh(:,:) = tmask(:,:,1) *  strength(:,:)


! scale factors
      DO jj = k_j1+1, k_jpj-1
         DO ji = 2, jpim1
            z1_e1t0(ji,jj) = 1._wp / ( e1t(ji+1,jj  ↩
              ) + e1t(ji,jj  ) )
            z1_e2t0(ji,jj) = 1._wp / ( e2t(ji  ,jj↩
              +1) + e2t(ji,jj  ) )
         END DO
      END DO

!
!↩
  -------------------------------------------------------------------
! 2) Wind / ocean stress, mass terms, coriolis terms
!↩
  -------------------------------------------------------------------

```

```
      IF( nn_ice_embd == 2 ) THEN             !== embedded sea ice: compute representative ice top surface ==!
!
! average interpolation coeff as used in dynspg = (1/nn_fsbc) * {SUM[n/nn_fsbc], n=0,nn_fsbc-1}
!                                               = (1/nn_fsbc)^2 * {SUM[n], n=0,nn_fsbc-1}
         zintn = REAL( nn_fsbc - 1 ) / REAL( nn_fsbc ) * 0.5_wp
!
! average interpolation coeff as used in dynspg = (1/nn_fsbc) * {SUM[1-n/nn_fsbc], n=0,nn_fsbc-1}
!                                               = (1/nn_fsbc)^2 * (nn_fsbc^2 - {SUM[n], n=0,nn_fsbc-1})
         zintb = REAL( nn_fsbc + 1 ) / REAL( nn_fsbc ) * 0.5_wp
!
         zpice(:,:) = ssh_m(:,:) + ( zintn * snwice_mass(:,:) + zintb * snwice_mass_b(:,:) ) * r1_rau0
!
      ELSE                                    !== non-embedded sea ice: use ocean surface for slope calculation ==!
         zpice(:,:) = ssh_m(:,:)
      ENDIF

      DO jj = k_j1+1, k_jpj-1
         DO ji = 2, jpim1

! ice fraction at U-V points
            zaU(ji,jj) = 0.5_wp * ( at_i(ji,jj) * e12t(ji,jj) + at_i(ji+1,jj) * e12t(ji+1,jj) ) * r1_e12u(ji,jj) * umask(ji,jj,1)
         zaV(ji,jj) = 0.5_wp * ( at_i(ji,jj) * e12t(ji,jj) + at_i(ji,jj+1) * e12t(ji,jj+1) ) * r1_e12v(ji,jj) * vmask(ji,jj,1)

! Ice/snow mass at U-V points
         zm1 = ( rhosn * vt_s(ji  ,jj  ) + rhoic * vt_i(ji  ,jj  ) )
         zm2 = ( rhosn * vt_s(ji+1,jj  ) + rhoic * vt_i(ji+1,jj  ) )
         zm3 = ( rhosn * vt_s(ji  ,jj+1) + rhoic * vt_i(ji  ,jj+1) )
         zmassU = 0.5_wp * ( zm1 * e12t(ji,jj) + zm2 * e12t(ji+1,jj) ) * r1_e12u(ji,jj) * umask(ji,jj,1)
         zmassV = 0.5_wp * ( zm1 * e12t(ji,jj) + zm3 * e12t(ji,jj+1) ) * r1_e12v(ji,jj) * vmask(ji,jj,1)

! Ocean currents at U-V points
         v_oceU(ji,jj)   = 0.5_wp * ( ( v_oce(ji  ,jj) + v_oce(ji  ,jj-1) ) * e1t(ji+1,jj)    &
            &                       + ( v_oce(ji+1,jj) + v_oce(ji+1,jj-1) ) * e1t(ji  ,jj) ) * z1_e1t0(ji,jj) * umask(ji,jj,1)

         u_oceV(ji,jj)   = 0.5_wp * ( ( u_oce(ji,jj  ) + u_oce(ji-1,jj  ) ) * e2t(ji,jj+1)    &
            &                       + ( u_oce(ji,jj+1) + u_oce(ji-1,jj+1) ) * e2t(ji,jj  ) ) * z1_e2t0(ji,jj) * vmask(ji,jj,1)

! Coriolis at T points (m*f)
         zmf(ji,jj)      = zm1 * fcor(ji,jj)
```

```fortran

! m/dt
            zmU_t(ji,jj)    = zmassU * z1_dtevp
            zmV_t(ji,jj)    = zmassV * z1_dtevp

! Drag ice-atm.
            zTauU_ia(ji,jj) = zaU(ji,jj) * utau_ice(ji,jj)
            zTauV_ia(ji,jj) = zaV(ji,jj) * vtau_ice(ji,jj)

! Surface pressure gradient (- m*g*GRAD(ssh)) at U-V points
            zspgU(ji,jj)    = - zmassU * grav * ( zpice(ji+1,jj) - zpice(ji,jj) ) * r1_e1u(ji,jj)
            zspgV(ji,jj)    = - zmassV * grav * ( zpice(ji,jj+1) - zpice(ji,jj) ) * r1_e2v(ji,jj)

! masks
            zmaskU(ji,jj) = 1._wp - MAX( 0._wp, SIGN( 1._wp, -zmassU ) )  ! 0 if no ice
            zmaskV(ji,jj) = 1._wp - MAX( 0._wp, SIGN( 1._wp, -zmassV ) )  ! 0 if no ice

! switches
            zswitchU(ji,jj) = MAX( 0._wp, SIGN( 1._wp, zmassU - zmmin ) ) ! 0 if ice mass < zmmin
            zswitchV(ji,jj) = MAX( 0._wp, SIGN( 1._wp, zmassV - zmmin ) ) ! 0 if ice mass < zmmin

         END DO
      END DO

      CALL lbc_lnk( zmf, 'T', 1. )
```

```
!
!------------------------------------------------------------
! 3) Solution of the momentum equation, iterative procedure
!------------------------------------------------------------
!
!                                                            !-----------------------!
      DO jter = 1 , nn_nevp                                  !    loop over jter    !
!                                                            !-----------------------!
         IF(ln_ctl) THEN   ! Convergence test
            DO jj = k_j1 , k_jpj-1
               zu_ice(:,jj) = u_ice(:,jj) ! velocity at previous time step
               zv_ice(:,jj) = v_ice(:,jj)
            END DO
         ENDIF

! --- divergence, tension & shear (Appendix B of Hunke & Dukowicz, 2002) --- !
         DO jj = k_j1, k_jpj-1          ! loops start at 1 since there is no boundary condition (lbc_lnk) at i=1 and j=1 for F points
            DO ji = 1, jpim1

! shear at F points
               zds(ji,jj) = ( ( u_ice(ji,jj+1) * r1_e1u(ji,jj+1) - u_ice(ji,jj) * r1_e1u(ji,jj) ) * e1f(ji,jj) * e1f(ji,jj)   &
```

```fortran
                  &          + ( v_ice(ji+1,jj) * r1_e2v(ji+1,jj) - v_ice(ji,jj) * r1_e2v(ji,jj) ) * e2f(ji,jj) * e2f(ji,jj)   &
                  &          ) * r1_e12f(ji,jj) * zfmask(ji,jj)

            END DO
         END DO
         CALL lbc_lnk( zds, 'F', 1. )

         DO jj = k_j1+1, k_jpj-1
            DO ji = 2, jpim1 ! no vector loop

! shear**2 at T points (doc eq. A16)
               zds2 = ( zds(ji,jj  ) * zds(ji,jj  ) * e12f(ji,jj  ) + zds(ji-1,jj  ) * zds(ji-1,jj  ) * e12f(ji-1,jj  )   &
                  &   + zds(ji,jj-1) * zds(ji,jj-1) * e12f(ji,jj-1) + zds(ji-1,jj-1) * zds(ji-1,jj-1) * e12f(ji-1,jj-1)   &
                  &   ) * 0.25_wp * r1_e12t(ji,jj)

! divergence at T points
               zdiv   = ( e2u(ji,jj) * u_ice(ji,jj) - e2u(ji-1,jj) * u_ice(ji-1,jj)   &
                  &    + e1v(ji,jj) * v_ice(ji,jj) - e1v(ji,jj-1) * v_ice(ji,jj-1)   &
                  &    ) * r1_e12t(ji,jj)
               zdiv2 = zdiv * zdiv

! tension at T points
               zdt   = ( ( u_ice(ji,jj) * r1_e2u(ji,jj) - u_ice(ji-1,jj) * r1_e2u(ji
                  -1,jj) ) * e2t(ji,jj) * e2t(ji,jj) &
                  &   - ( v_ice(ji,jj) * r1_e1v(ji,jj) - v_ice(ji,jj-1) * r1_e1v(ji,jj-1) ) * e1t(ji,jj) * e1t(ji,jj)   &
                  &   ) * r1_e12t(ji,jj)
               zdt2 = zdt * zdt

! delta at T points
               zdelta = SQRT( zdiv2 + ( zdt2 + zds2 ) * usecc2 )

! P/delta at T points
               zp_delt(ji,jj) = zpresh(ji,jj) / ( zdelta + rn_creepl )

! stress at T points
               zs1(ji,jj) = ( zs1(ji,jj) * zalph1 + zp_delt(ji,jj) * ( zdiv - zdelta ) ) * z1_alph1
               zs2(ji,jj) = ( zs2(ji,jj) * zalph2 + zp_delt(ji,jj) * ( zdt * z1_ecc2 ) ) * z1_alph2

            END DO
         END DO
         CALL lbc_lnk( zp_delt, 'T', 1. )

         DO jj = k_j1, k_jpj-1
            DO ji = 1, jpim1

! P/delta at F points
               zp_delf = 0.25_wp * ( zp_delt(ji,jj) + zp_delt(ji+1,jj) + zp_delt(ji,jj+1) + zp_delt(ji+1,jj+1) )

! stress at F points
```

```
               zs12(ji,jj)= ( zs12(ji,jj) * zalph2 + zp_delf * ( zds(ji,jj) * z1_ecc2 ) * 0.5_wp ) * z1_alph2

            END DO
         END DO
         CALL lbc_lnk_multi( zs1, 'T', 1., zs2, 'T', 1., zs12, 'F', 1. )

! --- Ice internal stresses (Appendix C of Hunke and Dukowicz, 2002) --- !
         DO jj = k_j1+1, k_jpj-1
            DO ji = 2, jpim1

! U points
               zfU(ji,jj) = 0.5_wp * ( ( zs1(ji+1,jj) - zs1(ji,jj) ) * e2u(ji,jj)                                                   &
                  &                  + ( zs2(ji+1,jj) * e2t(ji+1,jj) * e2t(ji+1,jj) - zs2(ji,jj) * e2t(ji,jj) * e2t(ji,jj)    &
                  &                    ) * r1_e2u(ji,jj)                                                                    &
                  &                  + ( zs12(ji,jj) * e1f(ji,jj) * e1f(ji,jj) - zs12(ji,jj-1) * e1f(ji,jj-1) * e1f(ji,jj-1)  &
                  &                    ) * 2._wp * r1_e1u(ji,jj)                                                            &
                  &                  ) * r1_e12u(ji,jj)

```

```
! V points
            zfV(ji,jj) = 0.5_wp * ( ( zs1(ji,jj+1) - zs1(ji,jj) ) * e1v(ji,jj) &
               &                  - ( zs2(ji,jj+1) * e1t(ji,jj+1) * e1t(ji,jj+1) - zs2(ji,jj) * e1t(ji,jj) * e1t(ji,jj)    &
               &                    ) * r1_e1v(ji,jj) &
               &                  + ( zs12(ji,jj) * e2f(ji,jj) * e2f(ji,jj) - zs12(ji-1,jj) * e2f(ji-1,jj) * e2f(ji-1,jj)  &
               &                    ) * 2._wp * r1_e2v(ji,jj) &
               &                  ) * r1_e12v(ji,jj)

! u_ice at V point
            u_iceV(ji,jj) = 0.5_wp * ( ( u_ice(ji,jj  ) + u_ice(ji-1,jj  ) ) * e2t(ji,jj+1)      &
               &                     + ( u_ice(ji,jj+1) + u_ice(ji-1,jj+1) ) * e2t(ji,jj  ) ) * z1_e2t0(ji,jj) * vmask(ji,jj,1)

! v_ice at U point
            v_iceU(ji,jj) = 0.5_wp * ( ( v_ice(ji,jj) + v_ice(ji  ,jj-1) ) * e1t(ji+1,jj)      &
                 &                       + ( v_ice(ji+1,jj) + v_ice(ji+1,jj-1) ) * e1t(ji  ,jj) ) * z1_e1t0(ji,jj) * umask(ji,jj,1)

           END DO
        END DO
!
! --- Computation of ice velocity --- !
!  Bouillon et al. 2013 (eq 47-48) => unstable unless alpha, beta are chosen wisely and large nn_nevp
!  Bouillon et al. 2009 (eq 34-35) => stable
        IF( MOD(jter,2) .EQ. 0 ) THEN ! even iterations

           DO jj = k_j1+1, k_jpj-1
              DO ji = 2, jpim1

! tau_io/(v_oce - v_ice)
                 zTau0 = zaV(ji,jj) * rhoco * SQRT( ( v_ice (ji,jj) - v_oce (ji,jj) ) * ( v_ice (ji,jj) - v_oce (ji,jj) )  &
                    &                             +( u_iceV(ji,jj) - u_oceV(ji,jj) ) * ( u_iceV(ji,jj) - u_oceV(ji,jj) ) )

! Ocean-to-Ice stress
                 ztauy_oi(ji,jj) = zTau0 * ( v_oce(ji,jj) - v_ice(ji,jj) )

! Coriolis at V-points (energy conserving formulation)
                 zCory(ji,jj)  = - 0.25_wp * r1_e2v(ji,jj) *  &
```

```
                     &     ( zmf(ji,jj  ) * ( e2u(ji,jj  ) * u_ice(ji,jj  ) + e2u(ji-1,jj  ) * u_ice(ji-1,jj  ) ) )   &
                     &     + zmf(ji,jj+1) * ( e2u(ji,jj+1) * u_ice(ji,jj+1) + e2u(ji-1,jj+1) * u_ice(ji-1,jj+1) ) )

! Sum of external forces (explicit solution) = F + tau_ia + Coriolis + spg + tau_io
                  zTauE = zfV(ji,jj) + zTauV_ia(ji,jj) + zCory(ji,jj) + zspgV(ji,jj) + ztauy_oi(ji,jj)

! ice velocity using implicit formulation (cf Madec doc & Bouillon 2009)
                  v_ice(ji,jj) = ( ( zmV_t(ji,jj) * v_ice(ji,jj) + zTauE + zTau0 * v_ice(ji,jj)  &  ! F + tau_ia + Coriolis + spg + tau_io(only ocean part)
                     &             ) / MAX( zepsi, zmV_t(ji,jj) + zTau0 ) * zswitchV(ji,jj)     &  ! m/dt + tau_io(only ice part)
                     &             + v_oce(ji,jj) * ( 1._wp - zswitchV(ji,jj) )                 &  ! v_ice = v_oce if mass < zmmin
                     &           ) * zmaskV(ji,jj)
               END DO
            END DO
            CALL lbc_lnk( v_ice, 'V', -1. )




```

```fortran
         DO jj = k_j1+1, k_jpj-1
            DO ji = 2, jpim1

! tau_io/(u_oce - u_ice)
               zTau0 = zaU(ji,jj) * rhoco * SQRT( ( u_ice (ji,jj) - u_oce (ji,jj) ) * ( u_ice (ji,jj) - u_oce (ji,jj) )  &
                  &                              + ( v_iceU(ji,jj) - v_oceU(ji,jj) ) * ( v_iceU(ji,jj) - v_oceU(ji,jj) ) )

! Ocean-to-Ice stress
               ztaux_oi(ji,jj) = zTau0 * ( u_oce(ji,jj) - u_ice(ji,jj) )

! Coriolis at U-points (energy conserving formulation)
               zCorx(ji,jj)  =   0.25_wp * r1_e1u(ji,jj) *  &
                  &    ( zmf(ji  ,jj) * ( e1v(ji  ,jj) * v_ice(ji  ,jj) + e1v(ji  ,jj-1) * v_ice(ji  ,jj-1) )  &
                  &    + zmf(ji+1,jj) * ( e1v(ji+1,jj) * v_ice(ji+1,jj) + e1v(ji+1,jj-1) * v_ice(ji+1,jj-1) ) )

! Sum of external forces (explicit solution) = F + tau_ia + Coriolis + spg + tau_io
               zTauE = zfU(ji,jj) + zTauU_ia(ji,jj) + zCorx(ji,jj) + zspgU(ji,jj) + ztaux_oi(ji,jj)

! ice velocity using implicit formulation (cf Madec doc & Bouillon 2009)
```

```
                  u_ice(ji,jj) = ( ( zmU_t(ji,jj) * u_ice(ji,jj) + zTauE + zTauO * u_ice(ji,jj)  &  ! F + tau_ia + Coriolis + spg + tau_io(only ocean part)
                     &              ) / MAX( zepsi, zmU_t(ji,jj) + zTauO ) * zswitchU(ji,jj)      &  ! m/dt + tau_io(only ice part)
                     &              + u_oce(ji,jj) * ( 1._wp - zswitchU(ji,jj) )                   &  ! v_ice = v_oce if mass < zmmin
                     &            ) * zmaskU(ji,jj)
               END DO
            END DO
            CALL lbc_lnk( u_ice, 'U', -1. )




         ELSE ! odd iterations

            DO jj = k_j1+1, k_jpj-1
               DO ji = 2, jpim1

! tau_io/(u_oce - u_ice)
                  zTauO = zaU(ji,jj) * rhoco * SQRT( ( u_ice (ji,jj) - u_oce (ji,jj) ) * ( u_ice (ji,jj) - u_oce (ji,jj) )  &
                     &                                             + ( v_iceU(ji,jj) - v_oceU(ji,jj) ) * ( v_iceU(ji,jj) - v_oceU(ji,jj) ) )

! Ocean-to-Ice stress
```

```fortran
                  ztaux_oi(ji,jj) = zTau0 * ( u_oce(ji,jj) - u_ice(ji,jj) )

! Coriolis at U-points (energy conserving formulation)
                  zCorx(ji,jj)   =   0.25_wp * r1_e1u(ji,jj) *  &
                     &    ( zmf(ji  ,jj) * ( e1v(ji  ,jj) * v_ice(ji  ,jj) + e1v(ji  ,jj-1) * v_ice(ji  ,jj-1) )  &
                     &    + zmf(ji+1,jj) * ( e1v(ji+1,jj) * v_ice(ji+1,jj) + e1v(ji+1,jj-1) * v_ice(ji+1,jj-1) ) )

! Sum of external forces (explicit solution) = F + tau_ia + Coriolis + spg + tau_io
                  zTauE = zfU(ji,jj) + zTauU_ia(ji,jj) + zCorx(ji,jj) + zspgU(ji,jj) + ztaux_oi(ji,jj)

! ice velocity using implicit formulation (cf Madec doc & Bouillon 2009)
                  u_ice(ji,jj) = ( ( zmU_t(ji,jj) * u_ice(ji,jj) + zTauE + zTau0 * u_ice(ji,jj)  & ! F + tau_ia + Coriolis + spg + tau_io(only ocean part)
                     &             ) / MAX( zepsi, zmU_t(ji,jj) + zTau0 ) * zswitchU(ji,jj)      & ! m/dt + tau_io(only ice part)
                     &             + u_oce(ji,jj) * ( 1._wp - zswitchU(ji,jj) )                  & ! v_ice = v_oce if mass < zmmin
                     &           ) * zmaskU(ji,jj)
                  END DO
               END DO
               CALL lbc_lnk( u_ice, 'U', -1. )




              DO jj = k_j1+1, k_jpj-1
                  DO ji = 2, jpim1

! tau_io/(v_oce - v_ice)
                     zTau0 = zaV(ji,jj) * rhoco * SQRT( ( v_ice (ji,jj) - v_oce (ji,jj) ) * ( v_ice (ji,jj) - v_oce (ji,jj) )  &
                        &                                    + ( u_iceV(ji,jj) - u_oceV(ji,jj) ) * ( u_iceV(ji,jj) - u_oceV(ji,jj) ) )

! Ocean-to-Ice stress
                     ztauy_oi(ji,jj) = zTau0 * ( v_oce(ji,jj) - v_ice(ji,jj) )

! Coriolis at V-points (energy conserving formulation)
                     zCory(ji,jj)  = - 0.25_wp * r1_e2v(ji,jj) *  &
                        &    ( zmf(ji,jj  ) * ( e2u(ji,jj  ) * u_ice(ji,jj  ) + e2u(ji-1,jj  ) * u_ice(ji-1,jj  ) )  &
                        &    + zmf(ji,jj+1) * ( e2u(ji,jj+1) * u_ice(ji,jj+1) + e2u(ji-1,jj+1) * u_ice(ji-1,jj+1) ) )

```

```
! Sum of external forces (explicit solution) = F + tau_ia + Coriolis + spg + tau_io
                  zTauE = zfV(ji,jj) + zTauV_ia(ji,jj) + zCory(ji,jj) + zspgV(ji,jj) + ztauy_oi(ji,jj)

! ice velocity using implicit formulation (cf Madec doc & Bouillon 2009)
                  v_ice(ji,jj) = ( ( zmV_t(ji,jj) * v_ice(ji,jj) + zTauE + zTau0 * v_ice(ji,jj)  &  ! F + tau_ia + Coriolis + spg + tau_io(only ocean part)
                     &              ) / MAX( zepsi, zmV_t(ji,jj) + zTau0 ) * zswitchV(ji,jj)      &  ! m/dt + tau_io(only ice part)
                     &              + v_oce(ji,jj) * ( 1._wp - zswitchV(ji,jj) )                   &  ! v_ice = v_oce if mass < zmmin
                     &            ) * zmaskV(ji,jj)
               END DO
            END DO
            CALL lbc_lnk( v_ice, 'V', -1. )




         ENDIF

         IF(ln_ctl) THEN   ! Convergence test
            DO jj = k_j1+1, k_jpj-1
               zresr(:,jj) = MAX( ABS( u_ice(:,jj) - zu_ice(:,jj) ), ABS( v_ice(:,jj) - zv_ice(:,jj) ) )
            END DO
```

```
               zresm = MAXVAL( zresr( 1:jpi, k_j1+1:k_jpj-1 ) )
               IF( lk_mpp )   CALL mpp_max( zresm )   ! max over the global domain
            ENDIF
!
!                                                        ! ==================== !
         END DO                                          ! end loop over jter   !
!                                                        ! ==================== !
!
!-----------------------------------------------------------------------------------
! 4) Recompute delta, shear and div (inputs for mechanical redistribution)
!-----------------------------------------------------------------------------------

         DO jj = k_j1, k_jpj-1
            DO ji = 1, jpim1

! shear at F points
               zds(ji,jj) = ( ( u_ice(ji,jj+1) * r1_e1u(ji,jj+1) - u_ice(ji,jj) * r1_e1u(ji,jj) ) * e1f(ji,jj) * e1f(ji,jj)   &
                  &          + ( v_ice(ji+1,jj) * r1_e2v(ji+1,jj) - v_ice(ji,jj) * r1_e2v(ji,jj) ) * e2f(ji,jj) * e2f(ji,jj)   &
                  &          ) * r1_e12f(ji,jj) * zfmask(ji,jj)

            END DO
         END DO
```

```
      CALL lbc_lnk( zds, 'F', 1. )

      DO jj = k_j1+1, k_jpj-1
         DO ji = 2, jpim1 ! no vector loop

! tension**2 at T points
            zdt  = ( ( u_ice(ji,jj) * r1_e2u(ji,jj) - u_ice(ji-1,jj) * r1_e2u(ji-1,jj) ) * e2t(ji,jj) * e2t(ji,jj)   &
               &   - ( v_ice(ji,jj) * r1_e1v(ji,jj) - v_ice(ji,jj-1) * r1_e1v(ji,jj-1) ) * e1t(ji,jj) * e1t(ji,jj)   &
               &   ) * r1_e12t(ji,jj)
            zdt2 = zdt * zdt

! shear**2 at T points (doc eq. A16)
            zds2 = ( zds(ji,jj  ) * zds(ji,jj  ) * e12f(ji,jj  ) + zds(ji-1,jj  ) * zds(ji-1,jj  ) * e12f(ji-1,jj  )  &
               &   + zds(ji,jj-1) * zds(ji,jj-1) * e12f(ji,jj-1) + zds(ji-1,jj-1) * zds(ji-1,jj-1) * e12f(ji-1,jj-1)  &
               &   ) * 0.25_wp * r1_e12t(ji,jj)

! shear at T points
            shear_i(ji,jj) = SQRT( zdt2 + zds2 )

! divergence at T points
            divu_i(ji,jj) = ( e2u(ji,jj) * u_ice(ji,jj) - e2u(ji-1,jj) * u_ice(ji-1,jj)   &
               &             + e1v(ji,jj) * v_ice(ji,jj) - e1v(ji,jj-1) * v_ice(ji,jj-1)   &
               &             ) * r1_e12t(ji,jj)

! delta at T points
```

```
            zdelta          = SQRT( divu_i(ji,jj) * ↩
               divu_i(ji,jj) + ( zdt2 + zds2 ) * ↩
               usecc2 )
            rswitch         = 1._wp - MAX( 0._wp, ↩
               SIGN( 1._wp, -zdelta ) ) ! 0 if delta↩
               =0
            delta_i(ji,jj) = zdelta + rn_creepl * ↩
               rswitch

         END DO
      END DO
      CALL lbc_lnk_multi( shear_i, 'T', 1., divu_i, ↩
         'T', 1., delta_i, 'T', 1. )

! --- Store the stress tensor for the next time step↩
    --- !
      stress1_i (:,:) = zs1 (:,:)
      stress2_i (:,:) = zs2 (:,:)
      stress12_i(:,:) = zs12(:,:)

!↩
   ------------------------------------------------------------------------

! 5) SIMIP diagnostics
!↩
   ------------------------------------------------------------------------


      DO jj = 1, jpj
         DO ji = 1, jpi
            zswi(ji,jj)  = MAX( 0._wp , SIGN( 1._wp ↩
               , at_i(ji,jj) - epsi06 ) ) ! 1 if ice↩
               , 0 if no ice
         END DO
      END DO

      zmiss(:,:)          = zmiss_val * ( 1. - zswi↩
         (:,:) )
```

```fortran

      DO jj = k_j1+1, k_jpj-1
         DO ji = 2, jpim1

! Stress tensor invariants (normal and shear stress N/m)
            zdiag_sig1(ji,jj) = ( zs1(ji,jj) + zs2(ji,jj) ) * zswi(ji,jj)                                   ! normal stress
            zdiag_sig2(ji,jj) = SQRT( ( zs1(ji,jj) - zs2(ji,jj) )**2 + 4*zs12(ji,jj)**2 ) * zswi(ji,jj)    ! shear stress

! Stress terms of the momentum equation (N/m2)
            zdiag_dssh_dx(ji,jj) = zspgU(ji,jj) * zswi(ji,jj)     ! sea surface slope stress term
            zdiag_dssh_dy(ji,jj) = zspgV(ji,jj) * zswi(ji,jj)

            zdiag_corstrx(ji,jj) = zCorx(ji,jj) * zswi(ji,jj)     ! Coriolis stress term
            zdiag_corstry(ji,jj) = zCory(ji,jj) * zswi(ji,jj)

            zdiag_intstrx(ji,jj) = zfU(ji,jj)   * zswi(ji,jj)     ! internal stress term
            zdiag_intstry(ji,jj) = zfV(ji,jj)   * zswi(ji,jj)

            zdiag_utau_oi(ji,jj) = ztaux_oi(ji,jj) * zswi(ji,jj)  ! oceanic stress
            zdiag_vtau_oi(ji,jj) = ztauy_oi(ji,jj) * zswi(ji,jj)

```

```
! 2D ice mass, snow mass, area transport arrays (X, Y)
             zfac_x = 0.5 * u_ice(ji,jj) * e2u(ji,jj) * zswi(ji,jj)
             zfac_y = 0.5 * v_ice(ji,jj) * e1v(ji,jj) * zswi(ji,jj)

             zdiag_xmtrp_ice(ji,jj) = rhoic * zfac_x * ( vt_i(ji+1,jj) + vt_i(ji,jj) ) ! ice mass transport, X-component (kg/s)
             zdiag_ymtrp_ice(ji,jj) = rhoic * zfac_y * ( vt_i(ji,jj+1) + vt_i(ji,jj) ) !        ''           Y-   ''

             zdiag_xmtrp_snw(ji,jj) = rhosn * zfac_x * ( vt_s(ji+1,jj) + vt_s(ji,jj) ) ! snow mass transport, X-component
             zdiag_ymtrp_snw(ji,jj) = rhosn * zfac_y * ( vt_s(ji,jj+1) + vt_s(ji,jj) ) !         ''          Y-   ''

             zdiag_xatrp(ji,jj)     = zfac_x        * ( at_i(ji+1,jj) + at_i(ji,jj) ) ! area transport,      X-component (m2/s)
             zdiag_yatrp(ji,jj)     = zfac_y        * ( at_i(ji,jj+1) + at_i(ji,jj) ) !        ''             Y-   ''

         END DO
      END DO

      CALL lbc_lnk_multi(   zdiag_sig1    , 'T',  1., zdiag_sig2    , 'T',  1.,    &
                   &           zdiag_dssh_dx, 'U', -1., zdiag_dssh_dy, 'V', -1.,    &
```

```
                    &          zdiag_corstrx, 'U', -1., zdiag_corstry, 'V', -1.,    &
                    &          zdiag_intstrx, 'U', -1., zdiag_intstry, 'V', -1.     )

      CALL lbc_lnk_multi(   zdiag_utau_oi, 'U', -1., zdiag_vtau_oi, 'V', -1.     )

      CALL lbc_lnk_multi(   zdiag_xmtrp_ice, 'U', -1., zdiag_xmtrp_snw, 'U', -1., &
                    &          zdiag_xatrp     , 'U', -1., zdiag_ymtrp_ice, 'V', -1., &
                    &          zdiag_ymtrp_snw, 'V', -1., zdiag_yatrp     , 'V', -1. )

      IF  ( iom_use( "xmtrpice" ) ) CALL iom_put( "xmtrpice"     ,  zdiag_xmtrp_ice(:,:)      )                    ! X-component of sea-ice mass transport (kg/s)
      IF  ( iom_use( "ymtrpice" ) ) CALL iom_put( "ymtrpice"     ,  zdiag_ymtrp_ice(:,:)      )                    ! Y-component of sea-ice mass transport

      IF  ( iom_use( "xmtrpsnw" ) ) CALL iom_put( "xmtrpsnw"     ,  zdiag_xmtrp_snw(:,:)      )                    ! X-component of snow mass transport (kg/s)
      IF  ( iom_use( "ymtrpsnw" ) ) CALL iom_put( "ymtrpsnw"     ,  zdiag_ymtrp_snw(:,:)      )                    ! Y-component of snow mass transport

      IF  ( iom_use( "xatrp"    ) ) CALL iom_put( "xatrp"        ,  zdiag_xatrp(:,:)
   )                             ! X-component ↩
   of ice area transport
IF  ( iom_use( "yatrp"    ) ) CALL iom_put( "↩
   yatrp"          ,  zdiag_yatrp(:,:)          ↩
   )                             ! Y-component ↩
   of ice area transport

IF  ( iom_use( "utau_ice" ) ) CALL iom_put( "↩
   utau_ice"      ,  utau_ice(:,:)      * zswi↩
   (:,:) + zmiss(:,:) )            ! Wind stress↩
    term in force balance (x)
IF  ( iom_use( "vtau_ice" ) ) CALL iom_put( "↩
   vtau_ice"      ,  vtau_ice(:,:)      * zswi↩
   (:,:) + zmiss(:,:) )            ! Wind stress↩
    term in force balance (y)

IF  ( iom_use( "utau_oi"  ) ) CALL iom_put( "↩
   utau_oi"      ,   zdiag_utau_oi(:,:) * zswi↩
   (:,:) + zmiss(:,:) )            ! Ocean ↩
   stress term in force balance (x)
IF  ( iom_use( "vtau_oi"  ) ) CALL iom_put( "↩
   vtau_oi"      ,   zdiag_vtau_oi(:,:) * zswi↩
   (:,:) + zmiss(:,:) )            ! Ocean ↩
   stress term in force balance (y)

IF  ( iom_use( "dssh_dx"  ) ) CALL iom_put( "↩
   dssh_dx"      ,   zdiag_dssh_dx(:,:) * zswi↩
   (:,:) + zmiss(:,:) )            ! Sea-surface↩
    tilt term in force balance (x)
IF  ( iom_use( "dssh_dy"  ) ) CALL iom_put( "↩
   dssh_dy"      ,   zdiag_dssh_dy(:,:) * zswi↩
   (:,:) + zmiss(:,:) )            ! Sea-surface↩
    tilt term in force balance (y)

IF  ( iom_use( "corstrx"  ) ) CALL iom_put( "↩
   corstrx"      ,   zdiag_corstrx(:,:) * zswi↩
   (:,:) + zmiss(:,:) )            ! Coriolis ↩
   force term in force balance (x)
```

```
      IF  ( iom_use( "corstry"  ) ) CALL iom_put( "corstry"     ,   zdiag_corstry(:,:) * zswi(:,:) + zmiss(:,:) )          ! Coriolis force term in force balance (y)

      IF  ( iom_use( "intstrx"  ) ) CALL iom_put( "intstrx"     ,   zdiag_intstrx(:,:) * zswi(:,:) + zmiss(:,:) )          ! Internal force term in force balance (x)
      IF  ( iom_use( "intstry"  ) ) CALL iom_put( "intstry"     ,   zdiag_intstry(:,:) * zswi(:,:) + zmiss(:,:) )          ! Internal force term in force balance (y)

      IF  ( iom_use( "normstr"  ) ) CALL iom_put( "normstr"     ,   zdiag_sig1(:,:)    * zswi(:,:) + zmiss(:,:) )          ! Normal stress
      IF  ( iom_use( "sheastr"  ) ) CALL iom_put( "sheastr"     ,   zdiag_sig2(:,:)    * zswi(:,:) + zmiss(:,:) )          ! Shear stress

!
!--------------------------------------------------------------
! 6) Control prints of residual and charge ellipse
!--------------------------------------------------------------
!
! print the residual for convergence
      IF(ln_ctl) THEN
         WRITE(charout,FMT="('lim_rhg  : res =',D23.16, ' iter =',I4)") zresm, jter
         CALL prt_ctl_info(charout)
```

```fortran
         CALL prt_ctl(tab2d_1=u_ice, clinfo1=' lim_rhg  : u_ice :', tab2d_2=v_ice, clinfo2=' v_ice :')
      ENDIF

! print charge ellipse
! This can be desactivated once the user is sure that the stress state
! lie on the charge ellipse. See Bouillon et al. 08 for more details
      IF(ln_ctl) THEN
         CALL prt_ctl_info('lim_rhg  : numit  :',ivar1=numit)
         CALL prt_ctl_info('lim_rhg  : nwrite :',ivar1=nwrite)
         CALL prt_ctl_info('lim_rhg  : MOD    :',ivar1=MOD(numit,nwrite))
         IF( MOD(numit,nwrite) .EQ. 0 ) THEN
            WRITE(charout,FMT="('lim_rhg  :', I4, I6, I1, I1, A10)") 1000, numit, 0, 0, ' ch. ell. '
            CALL prt_ctl_info(charout)
            DO jj = k_j1+1, k_jpj-1
               DO ji = 2, jpim1
                  IF (zpresh(ji,jj) > 1.0) THEN
                     zsig1 = ( zs1(ji,jj) + (zs2(ji,jj)**2 + 4*zs12(ji,jj)**2 )**0.5 ) / ( 2*zpresh(ji,jj) )
                     zsig2 = ( zs1(ji,jj) - (zs2(ji,jj)**2 + 4*zs12(ji,jj)**2 )**0.5 ) / ( 2*zpresh(ji,jj) )
                     WRITE(charout,FMT="('lim_rhg  :', I4, I4, D23.16, D23.16, D23.16, D23.16, A10)")
                     CALL prt_ctl_info(charout)
                  ENDIF
               END DO
            END DO
            WRITE(charout,FMT="('lim_rhg  :', I4, I6, I1, I1, A10)") 2000, numit, 0, 0, ' ch. ell. '
            CALL prt_ctl_info(charout)
         ENDIF
      ENDIF

!
      CALL wrk_dealloc( jpi,jpj, zpresh, z1_e1t0, z1_e2t0, zp_delt )
      CALL wrk_dealloc( jpi,jpj, zaU, zaV, zmU_t, zmV_t, zmf, zTauU_ia, ztauV_ia )
      CALL wrk_dealloc( jpi,jpj, zspgU, zspgV, v_oceU, u_oceV, v_iceU, u_iceV, zfU, zfV )
      CALL wrk_dealloc( jpi,jpj, zds, zs1, zs2, zs12, zu_ice, zv_ice, zresr, zpice )
      CALL wrk_dealloc( jpi,jpj, zswitchU, zswitchV, zmaskU, zmaskV, zfmask, zwf )
      CALL wrk_dealloc( jpi,jpj, zCorx, zCory )
      CALL wrk_dealloc( jpi,jpj, ztaux_oi, ztauy_oi )

      CALL wrk_dealloc( jpi,jpj, zdiag_xmtrp_ice, zdiag_ymtrp_ice )
      CALL wrk_dealloc( jpi,jpj, zdiag_xmtrp_snw, zdiag_ymtrp_snw )
      CALL wrk_dealloc( jpi,jpj, zdiag_xatrp    , zdiag_yatrp     )
      CALL wrk_dealloc( jpi,jpj, zdiag_utau_oi  , zdiag_vtau_oi   )
      CALL wrk_dealloc( jpi,jpj, zdiag_dssh_dx  , zdiag_dssh_dy   )
      CALL wrk_dealloc( jpi,jpj, zdiag_corstrx  , zdiag_corstry   )
      CALL wrk_dealloc( jpi,jpj, zdiag_intstrx  , zdiag_intstry   )
      CALL wrk_dealloc( jpi,jpj, zdiag_sig1      , zdiag_sig2       )
      CALL wrk_dealloc( jpi,jpj, zswi            , zmiss            )

   END SUBROUTINE lim_rhg



!!====================================================================================================
END MODULE limrhg
```

Bibliography

10

Arakawa, Akio and Vivian R. Lamb (1977). “Computational Design of the Basic Dynamical Processes of the UCLA General Circulation Model”. In: *General Circulation Models of the Atmosphere*. Ed. by Julius Chang. Vol. 17. Methods in Computational Physics: Advances in Research and Applications. Elsevier, pp. 173–265.

Balaji, V. (2015). “Climate Computing: The State of Play”. In: *Computing in Science & Engineering* 17.6, pp. 9–13.

Balaji, V., E. Maisonnave, N. Zadeh, *et al.* (2017). “CPMIP: measurements of real computational performance of Earth system models in CMIP6”. In: *Geoscientific Model Development* 10.1, pp. 19–34.

Benshila, R. (2001). *NEMO limrhg.f90: module timing*. Version 4.0.

Bouillon, Sylvain, Miguel Ángel Morales Maqueda, Vincent Legat, and Thierry Fichefet (2009). “An elastic–viscous–plastic sea ice model formulated on Arakawa B and C grids”. In: *Ocean Modelling* 27.3, pp. 174–184.

Craig, A., S. Valcke, and L. Coquart (2017). “Development and performance of a new version of the OASIS coupler, OASIS3-MCT_3.0”. In: *Geoscientific Model Development* 10.9, pp. 3297–3308.

Dawson, A., T. N. Palmer, and S. Corti (2012). “Simulating regime structures in weather and climate prediction models”. In: *Geophysical Research Letters* 39.21. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2012GL053284.

Döscher, R., M. Acosta, A. Alessandri, *et al.* (2021). “The EC-Earth3 Earth system model for the Coupled Model Intercomparison Project 6”. In: *Geoscientific Model Development* 15.7, pp. 2973–3020.

ECMWF (2010). “IFS Documentation CY36R1 - Part III: Dynamics and Numerical Procedures”. In: *IFS Documentation CY36R1*. IFS Documentation 3. Operational implementation 26 January 2010. ECMWF.

ECMWF (2015). *Using CrayPat and Apprentice2: A Step-by-step guide*. https://confluence.ecmwf.int/download/attachments/46600240/CrayPAT_Introduction_VH1.pdf?api=v2. Accessed: 2022-09-01.

ECMWF (n.d.). *15 Performance Analysis with Craypat*. `https://confluence.ecmwf.int/download/attachments/46600240/15_Performance_Analysis_with_Craypat.pdf?api=v2`.

Eyring, V., S. Bony, G. A. Meehl, C. A. Senior, B. Stevens, R. J. Stouffer, and K. E. Taylor (2016). "Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organization". In: *Geoscientific Model Development* 9.5, pp. 1937–1958.

Flato, Gregory M. (2011). "Earth system models: an overview". In: *WIREs Climate Change* 2.6, pp. 783–800. eprint: `https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/wcc.148`.

Haarsma, R., M. Acosta, R. Bakhshi, *et al.* (2020). "HighResMIP versions of EC-Earth: EC-Earth3P and EC-Earth3P-HR – description, model computational performance and basic validation". In: *Geoscientific Model Development* 13.8, pp. 3507–3527.

Haarsma, R. J., M. J. Roberts, P. L. Vidale, *et al.* (2016). "High Resolution Model Intercomparison Project (HighResMIP v1.0) for CMIP6". In: *Geoscientific Model Development* 9.11, pp. 4185–4208.

Hazeleger, W., X. Wang, C. Severijns, *et al.* (Dec. 2012). "EC-Earth V2.2: description and validation of a new seamless earth system prediction model". In: *Climate Dynamics* 39.11, pp. 2611–2629.

Hewitt, Helene T., Michael J. Bell, Eric P. Chassignet, Arnaud Czaja, David Ferreira, Stephen M. Griffies, Pat Hyder, Julie L. McClean, Adrian L. New, and Malcolm J. Roberts (2017). "Will high-resolution global ocean models benefit coupled predictions on short-range to climate timescales?" In: *Ocean Modelling* 120, pp. 120–136.

Hewlett-Packard (n.d.). *Cray Performance Measurement and Analysis Tools User Guide 6.4.4 S-2376*. `https://support.hpe.com/hpesc/public/docDisplay?docLocale=en_US&docId=a00113914en_us&page=About_the_Cray_Performance_Measurement_and_Analysis_Tools_User_Guide.html`. Accessed: 2022-08-29.

Hibler, W. D. (1979). "A Dynamic Thermodynamic Sea Ice Model". In: *Journal of Physical Oceanography* 9.4, pp. 815–846.

Hunke, E. C. and J. K. Dukowicz (1997). "An Elastic–Viscous–Plastic Model for Sea Ice Dynamics". In: *Journal of Physical Oceanography* 27.9, pp. 1849–1867.

Hunke, E. C. and W. H. Lipscomb (2006). "CICE: the Los Alamos Sea Ice Model Documentation and Software User's Manual, Tech". In: Report LA-CC-98-16.

Hunke, Elizabeth C. and John K. Dukowicz (2002). “The elastic-viscous-plastic sea ice dynamics model in general orthogonal curvilinear coordinates on a sphere: Incorporation of metric terms”. eng. In: *Monthly weather review* 130.7, pp. 1848–1865.

Intel (2016). *for_emit_diagnostic in backtrace*. `https://community.intel.com/t5/Intel-Fortran-Compiler/for-emit-diagnostic-in-backtrace/td-p/1077188`. Accessed: 2016-09-16.

Intel (2017). *Intel(R) Fortran Compiler Help v18.0.0 20170811*.

IPCC Sixth Assessment Report (2021).

Ludemann, D. S. (2022). “Impact of snow albedo parameterization over the Greenland ice sheet on the simulated climate of EC-Earth3”. In: Unpublished master’s thesis., p. 73.

Madec, G. and M. Imbard (1996). “A global ocean mesh to overcome the North Pole singularity”. In: *Climate Dynamics* 12, pp. 381–388.

Madec, G. and the NEMO team (Jan. 2016). *NEMO ocean engine*. Version 3.6 stable. Note du Pôle de modélisation de l’Institut Pierre-Simon Laplace No 27. NEMO team. 402 pp.

Rousset, C., M. Vancoppenolle, G. Madec, *et al.* (2015). “The Louvain-La-Neuve sea ice model LIM3.6: global and regional capabilities”. In: *Geoscientific Model Development* 8.10, pp. 2991–3005.

Temam, Roger and Mohammed Zaine (2005). *Handbook of Mathematical Fluid Dynamics, Chapter 6 – Some Mathematical Problems in Geophysical Fluid Dynamics*. Ed. by S. Friedlander and D. Serre. Vol. 3. North-Holland, pp. 535–658.

Tintó Prims, Oriol, Miguel Castrillo, Mario C. Acosta, Oriol Mula-Valls, Alicia Sanchez Lorente, Kim Serradell, Ana Cortés, and Francisco J. Doblas-Reyes (2019). “Finding, analysing and solving MPI communication bottlenecks in Earth System models”. In: *Journal of Computational Science* 36, p. 100864.

UNFCCC (2015). *Paris Agreement*.

University, Cornel (2022). *Vectorization: Pointer Aliasing*. `https://cvw.cac.cornell.edu/vector/coding_aliasing`.

“World Climate Research Programme Strategic Plan 2019-2028” (2019). In: *WCRP*.