Calcul Haute Performance et optimisation de simulations océanographique multi-échelles

Gaston Irrmann

▶ **To cite this version:**

Gaston Irrmann. Calcul Haute Performance et optimisation de simulations océanographique multi-échelles. Océan, Atmosphère. Sorbonne Université, 2024. Français. NNT : 2024SORUS270 . tel-04815140

HAL Id: tel-04815140

https://theses.hal.science/tel-04815140v1

Submitted on 2 Dec 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE DE DOCTORAT DE SORBONNE UNIVERSITE

Ecole Doctorale : 129 - Sciences de l'Environnement d'Ile de France

Spécialité : Océanographie Physique

réalisée au

Laboratoire d'Océanographie et du Climat : Expérimentations et Approches Numériques

et dans le

Groupe BULL SAS

présentée par

Gaston IRRMANN

pour obtenir le grade de

Docteur de Sorbonne Université

Titre de la thèse

Calcul Haute Performance et optimisation de simulations océanographique multi-échelles

Soutenance prévue devant le jury composé de :

Rapporteur	Martin Schreiber	Université Grenoble Alpes
Rapporteur	Thomas Dubos	IP Paris
Examinateur	Francis Codron	LOCEAN, Paris
Examinateur	Christophe Calvin	CEA Saclay
Examinateur	Dimitri Lecas	IDRIS - CNRS
Directeur de thèse	Gurvan Madec	LOCEAN, Paris
CoDirecteur de thèse	Sébastien Masson	LOCEAN, Paris
Encadrant de thèse	Erwan Raffin	CEPP, Eviden
Invité	David Guibert	CEPP, Eviden

Avant-Propos

Cette étude a été menée avec le soutien du groupe BULL SAS et de l'Association Nationale Recherche Technologie (Convention CIFRE n° 2020/0518).

Remerciement

L'existence des présents travaux doit beaucoup aux personnes qui m'ont encadré, conseillé et motivé tout au long de cette thèse. En premier lieu, Sébastien Masson qui m'a introduit au monde de la recherche et à ses méthodes, m'a soutenu même dans les débogages les plus ardus et à qui je dois les rencontres avec la plupart des autres contributeurs de ce travail. Je remercie Gurvan Madec pour m'avoir pris en thèse et avoir guidé ces travaux grâce à ses connaissances pointues qui m'ont permis de me familiariser avec le code NEMO. Les conseils apportés par Erwan Raffin ont été d'une aide précieuse. L'expertise technique de David Guibert et son accompagnement ont été indispensables pour mener à bien les développements mis en œuvre dans ce travail.

Pour chaque axe principal de recherche et de développement exploré, la participation d'experts du domaine a été centrale pour affiner notre compréhension et apporter des connaissances primordiales. Marc Sergent a apporté son importante expérience sur la librairie MPI et ses fonctionnalités les plus avancées sans laquelle de nombreux développements auraient été impossibles. Je remercie Jesús Labarta d'avoir été un interlocuteur passionnant sur les sujets de l'analyse des performances et les schémas de communication, mais aussi pour ses explications approfondies sur les outils Paraver et Extrae. Laurent Debreu et Jérôme Chanut ont apporté une aide précieuse pour notre compréhension du code AGRIF. Rachid Benshila a apporté ses conseils aux moments clé de cette thèse.

Je remercie enfin les équipes de Atos et NEMO pour le cadre de travail chaleureux qu'elles m'ont apporté, pour les conversations enrichissantes et les nombreux moments festifs. Ce travail doit beaucoup à la bonne humeur apportée par les doctorants du LOCEAN et les discussions enrichissantes avec Sara Sergi.

Résumé

Ce travail a pour objet l'optimisation des performances du modèle de circulation générale de l'océan, de banquise et de bio-géochimie NEMO (Nucleus for European Modelling of the Ocean) et en particulier dans ses applications multi-échelles qui reposent sur l'utilisation de la librairie AGRIF (Adaptive Grid Refinement in Fortran). L'analyse se concentre dans un premier temps sur les performances de NEMO sans utilisation de AGRIF. Des configurations adaptées sont mises en place, des outils de profilage sont développés, et une méthodologie d'évaluation des performances est proposée. Cette analyse révèle des obstacles majeurs aux performances de NEMO, notamment le nombre élevé de communications MPI (Message Passing Interface) et des ralentissements à différents niveaux de l'exécution. Des stratégies sont élaborées pour réduire les communications MPI et atténuer les ralentissements. L'une de ces stratégies consiste en une modification du schéma de communication pour limiter l'impact des ralentissements. Une analyse approfondie du schéma de communication est présentée, comprenant le développement de schémas alternatifs utilisant des fonctionnalités MPI avancées. Le schéma par défaut optimisé offre les meilleures performances, sauf dans certaines portions du code. Enfin, l'étude se tourne vers AGRIF pour lequel des configurations adaptées sont développées. Une analyse met en évidence les points d'optimisation potentiels. Des optimisations sont conçues pour réduire le temps d'exécution des simulations, elles apportent des gains en performances importants.

Mots clés : Calcul de Haute Performance, Simulation océanographique multi-échelles, Communications MPI

Abstract

The aim of this work is to optimize the performance of the NEMO (Nucleus for European Modelling of the Ocean") model of ocean general circulation, pack ice and bio-geochemistry, in particular in its multi-scale applications based on the use of the AGRIF (Adaptive Grid Refinement in Fortran) library. The analysis focuses initially on the performance of NEMO without the use of AGRIF. Adapted configurations are set up, profiling tools are developed, and methodology for performance evaluation is proposed. This analysis reveals major obstacles to NEMO's performance, notably the high number of MPI (Message Passing Interface) communications and slowdowns at various execution levels. Strategies are developed to reduce unnecessary MPI communications and mitigate slowdowns. One such strategy involves modifying the communication scheme to limit the impact of slowdowns. An in-depth analysis of the communication scheme is presented, including the development of alternative schemes using advanced MPI features. The optimized default scheme offers the best performance, except in certain portions of the code. Finally, the study turns to AGRIF, for which adapted configurations are developed. An analysis highlights potential optimization points. Optimizations are designed to reduce simulation execution time, bringing significant performance gains.

Keywords : High-Performance Computing, multi-scale ocean simulation, MPI communications.

Table des matières

Avant-Propos 3

Remerciement 5

Résumé 7

Abstract 9

Table des matières 11

Glossaire 13

I Introduction 17

1 NEMO et Calcul Haute Performance 18

II Cadre et outils dédiés à l'optimisation de NEMO 27

1 Stratégie d'optimisation de NEMO 28

2 Configurations 29

3 Profilage de NEMO 36

4 Évaluation des performances 46

5 Conclusion et perspectives 55

III Freins aux performances dans NEMO 57

1 Nombre de communications 58

2 Ralentissements de l'exécution 62

3 Spécificité du calcul du gradient de pression de surface 69

4 Conclusion et perspectives 71

IV Optimisation des schémas de communication **73**

1 Le fonctionnement de la librairie MPI 74

2 Schémas de communication dans NEMO 79

3 Performances des schémas de communication 103

4 Conclusion et perspectives 113

V Optimisation de simulations avec zooms emboîtés **117**

1 Le logiciel AGRIF 118

2 Profilage de AGRIF 126

3 Optimisation des communications MPI dans AGRIF 136

4 Optimisation des routines d'interpolation 140

5 Conclusion et perspectives 159

VI Discussion **163**

A Annexe : Réduction du nombre de communications **169**

1 Frontières ouvertes 170

2 Calcul du gradient de pression de surface 177

B Annexe : Algorithme de synchronisation des horloges **181**

1 Synchronisation par barrière MPI globale 182

2 Synchronisation par barrière MPI 2 à 2 183

3 Synchronisation par chronologie des communications 186

4 Synchronisation par aller-retour 188

Références **195**

Glossaire

AGRIF *Adaptive Grid Refinement In Fortran* est une librairie qui permet un raffinement de maillage dans un modèle multidimensionnel à différences finies écrit en langage Fortran.

Benchmark Banc d'essai en français : test permettant de mesurer les performances, souvent la rapidité d'exécution, d'un code.

Cache Une mémoire cache stocke temporairement un nombre limité de données pour les rendre accessible plus rapidement.

CPU *Central Processing Unit* ou unité centrale de calcul en français, est un composant qui exécute les instructions machine des programmes informatiques. Il est composé d'un certain nombre de cœur de calcul..

Cœur Composant d'un CPU qui, sans vectorisation, ne peut exécuter qu'une seule instruction à la fois.

Discrétisation Transposition des équations du problème définies sur un espace continu vers un espace discret.

GIEC Le *Groupe d'experts intergouvernemental sur l'évolution du climat* est un organisme scientifique intergouvernemental qui évalue les causes, les impacts et les solutions du changement climatique sur la base de publications scientifiques et techniques.

GPU *Graphics Processing Unit* ou processeur graphique en français est une unité de calcul conçue pour présenter une grande efficacité sur certains types de calculs, en particulier les calculs sur les images.

HPC *High Performance Computing* ou Calcul Haute Performance en français : discipline consistant à utiliser de manière aussi efficace que possible de toutes les ressources de calcul à disposition.

Hyperthreading Exécution sur un seul cœur de plusieurs processus.

ICE Module de glace de mer de NEMO.

InterConnect Réseau de connexion entre nœuds d'un supercalculateur.

MPI *Message Passing Interface*, norme permettant l'échange de messages, appelés communications MPI, entre CPU. Elle est implémentée dans plusieurs librairies dont la intelMPI ou la openMPI.

namelist Fonctionnalité du Fortran qui permet de lire ou écrire un groupe de variables de manière simple et flexible. NEMO utilise ce type de fichiers pour définir un grand nombre de ses paramètres d'entrée afin de préciser la configuration à exécuter.

NEMO *Nucleus for European Modelling of the Ocean*, est une plateforme numérique dédiée aux activités de recherche et aux services de prévision dans les sciences de l'océan et du climat, développé par un consortium européen.

Nœud Un nœud de calcul regroupe plusieurs CPUs.

OpenMP *Open Multi-Processing* est une librairie permettant de répartir la charge de calcul sur plusieurs processus partageant un même cache mémoire.

profilage Le profilage est une analyse précise des performances de l'exécution d'un code informatique. Il existe des outils de profilage, appelés profileurs, par exemple Paraver-Extrae, APS ou Gprof.

RAM *Random-Access Memory* ou mémoire vive en français est un type de mémoire utilisé pour stocker des données utilisées par un programme.

Scalabilité Capacité d'un code à maintenir ses performances quand le nombre de processus, sur lequel il est exécuté, augmente.

Stencil Agencement sur une grille des points utilisés pour une approximations numériques.

Superordinateur Superordinateur (ou supercalculateur) est un ordinateur conçu pour atteindre les plus hautes performances possibles au regard de la technologie disponible au moment de sa conception.

Switch Un switch, ou commutateur réseau, est un équipement qui relie plusieurs segments (câbles ou fibres) dans un réseau informatique.

TOP-PISCES Module de bio-géochimie de NEMO.

Vectorisation Transformation du code par le compilateur qui permet le calcul simultané d'opération effectuées par un unique cœur sur des données différentes mais en utilisant des opérations identiques.

XIOS Librairie permettant à des applications utilisant MPI de gérer efficacement les lectures et écritures de fichiers au cours de l'exécution.

CHAPITRE

I

Introduction

1 NEMO et Calcul Haute Performance

1.1 NEMO, un modèle de simulation océanographique

Il n'est plus nécessaire de justifier l'importance de la recherche climatique pour nos sociétés (Masson-Delmotte et al., 2018). Les études climatiques explorent un système complexe régi par une grande variété d'interactions allant des processus physiques aux processus bio-géochimiques dans l'océan, l'atmosphère, la surface terrestre et la cryosphère. La modélisation numérique est un outil essentiel dans la recherche sur le climat, qui complète les observations éparses dans le temps et l'espace. Les expériences numériques sont un moyen unique de tester des hypothèses, d'étudier les processus en jeu, de quantifier leurs impacts sur le climat et sa variabilité et, enfin, de réaliser des prévisions sur le changement climatique (Intergovernmental Panel on Climate Change (IPCC), 2023).

L'océan contient 97% de l'eau libre, c'est à dire non contenue dans des roches (Fieux, 2020). Du fait de sa masse et de la capacité thermique élevée de l'eau, il agit comme un réservoir de chaleur pour le système océan-atmosphère. De plus, l'océan réagit lentement aux variations de l'atmosphère et joue, de ce fait, un rôle important dans l'évolution du climat. Dans cette perspective, mes travaux se concentrent sur le modèle "NEMO" (Nucleus for European Modelling of the Ocean, Madec et al. (2023)), un cadre pour les activités de recherche et les services de prévision dans les sciences de l'océan et du climat, développé par un consortium européen. NEMO permet de modéliser l'océan sous ses différents aspects (dynamique, bio-géochimique et glace de mer). Les applications s'étendent des projections climatiques du GIEC aux prévisions météorologiques et saisonnières en passant par la recherche académique sur des sujets extrêmement variés comme par exemple la paléoclimatologie. En Europe, il est le modèle le plus utilisé pour des simulations à l'échelle globale ainsi que régionale. C'est aussi le modèle d'océan le plus utilisé, à égalité avec le modèle américan MOM, dans les modèles de climat sur lesquels repose une partie du rapport du groupe I du GIEC ("Annex II : Models", 2023).

Pour mener à bien la modélisation des masses d'eau dans l'océan, NEMO résout les équations de Navier Stokes avec un certain nombre d'hypothèses (hypothèse de couche mince, hypothèse d'incompressibilité, la forme de la terre approximée par une sphère...) ainsi que plusieurs équations de conservations (de la température et du sel). Les interactions avec l'atmosphère, les continents et la glace font de même appel à des équations modélisant ces interactions. Les dimensions temporelles et spatiales du problème sont discrétisées : l'axe temporel est découpé en segments et les trois dimensions spatiales (horizontales et verticale) sont discrétisées sur une grille, les différentes équations sont ensuite exprimées sur cet espace discrétisé.

Certaines simulations NEMO utilisent des grilles de discrétisation comportant un grand nombre de points pour permettre une résolution suffisamment précise et spatialement fine sur un domaine de simulation pouvant aller d'une taille caractéristique d'une centaine de kilomètre à la totalité de la planète. Plus le nombre de points de grille est important plus la quantité de calcul à effectuer est importante et donc le temps de calcul de la simulation élevé pour les utilisateurs du code. D'autre part, l'étendue de la plage temporelle sur laquelle certaines simulations doivent être calculées, en particulier pour des simulations climatiques ou paléoclimatiques, rajoute à la charge de calcul. Pour que NEMO reste un code utilisé par une vaste communauté, le code doit pouvoir être exécuté efficacement sur les architectures des derniers supercalculateurs.

1.2 Évolution des supercalculateurs

L'architecture d'un Central Processing Unit (CPU) est présentée sur la figure I.1 qui montre un exemple d'un CPU comportant 24 cœurs de calcul (indiqués *core*), les caches L1 et L2 sont uniquement accessibles par les cœurs auxquels ils sont associés. Les mémoires du cache L3 et de la mémoire RAM sont accessibles par tous les cœurs du CPU. Plus un cache est proche du cœur plus il est accessible rapidement par ce cœur, mais moins sa capacité mémoire est élevée. Un nœud de calcul regroupe plusieurs CPUs, souvent 2 mais possiblement plus, et met à disposition des CPUs des composants permettant par exemple l'écriture de fichiers ou des transferts de données entre nœuds.

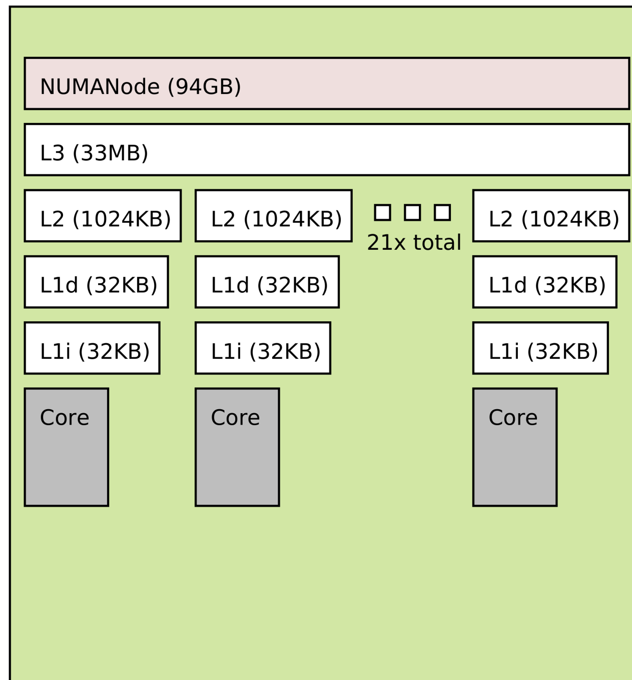


FIGURE I.1 – Architecture d'un CPU Skylake du supercalculateur Irene, les rectangles blancs sont des caches de mémoires (ou d'instructions pour les L1i), le rectangle rose représente la mémoire RAM et les rectangles gris les cœurs de calculs. Les caches L3 et la mémoire RAM sont communs pour tous les cœurs du CPU.

On distingue ainsi deux types d'architectures de mémoire, la mémoire partagée et la mémoire distribuée. La mémoire partagée se caractérise par des unités de calculs qui effectuent leurs accès mémoire en accédant à un même matériel (au cache L1 ou L2 pour des cœurs au sein d'un même CPU Skylake sur Irene). Contrairement à la mémoire distribuée, où les unités de calcul accèdent à un matériel différent, une communication doit être effectuée pour qu'une unité de calcul obtienne des données sur du matériel qui ne lui est pas directement accessible. Les supercalculateurs actuels sont d'une taille telle qu'il est plus efficace de scinder le matériel de stockage de la mémoire. Autrement dit, les supercalculateurs utilisent une architecture au moins partiellement en mémoire partagée.

La nouvelle génération de supercalculateurs affiche des capacités de calcul exaflopiques, c'est à dire capables de réaliser 10^{18} opérations en virgule flottante à double précision par seconde. Le supercalculateur *Frontier*, classé numéro 1 de la liste TOP 500 de juin 2023 (June 2023 | TOP500, 2023), a atteint une performance de 1.194 exaflops par secondes ($EFlop/s$) pour une utilisation de la librairie d'algèbre linéaire *Linpack*.

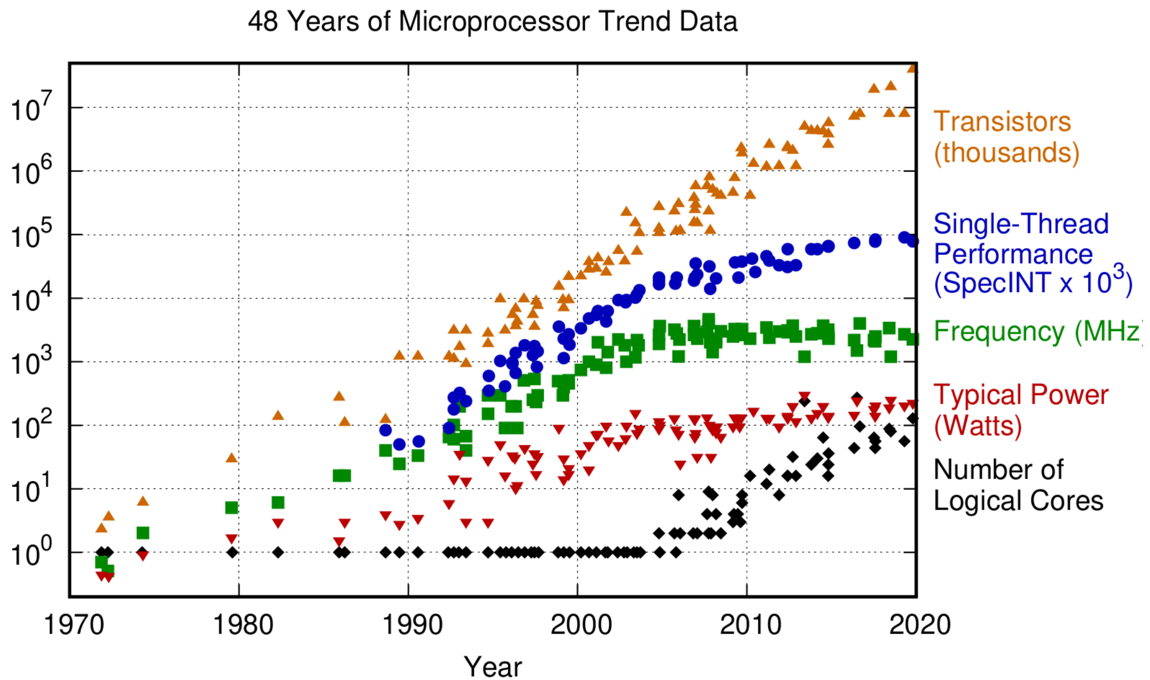


FIGURE I.2 – Données jusqu'à l'année 2010 collectées et tracées par M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond et C. Batten. Nouvelle figure et données collectées pour la période 2010-2019 par K. Rupp

La figure I.2, tirée de Rupp (2020), représente l'évolution des performances des supercalculateurs depuis 1970. Le nuage de points indiqué *Transistors* représente le nombre de transistors par cœur. Les performances sur un seul processus (*Single-Thread Performance*) sont évaluées sur le benchmark *SPECint*.

Le nuage de points indiqué *Number of Logical Cores* correspond au nombre de cœurs logiques dans un nœud. Les cœurs logiques se différencient des cœurs physiques qui correspondent à un matériel pouvant effectuer des opérations. Certains cœurs physiques ont parfois la capacité d'alterner et d'effectuer des opérations commandées par 2 processus associés au même cœur physique. On dit alors que chaque cœur physique a deux cœurs logiques et utiliser plusieurs processus par cœur physique s'appelle faire de l'*hyperthreading*. Le nombre de cœurs logiques par cœurs physiques est typiquement de 1 ou de 2, le nuage de points *Number of Logical Cores* n'est donc pas strictement équivalent au nombre de cœurs physiques mais reflète tout de même son évolution.

Les performances sur un seul processus augmentent continuellement, malgré un ralentissement à partir du milieu des années 2000. Ce profil est principalement expliqué par l'évolution de la fréquence de l'horloge. D'après Etiemble (2018), la consommation énergétique est proportionnelle à la fréquence d'horloge. Pour des raisons écologiques, mais bien sûr surtout pour des raisons économiques, la fréquence n'est plus systématiquement augmentée. La stratégie adoptée par les constructeurs de supercalculateurs pour continuer d'augmenter les performances a donc changé vers le milieu des années 2000. Pour compenser la lente augmentation des performances sur un seul processus, le nombre de cœurs par nœud est augmenté. Cela permet d'augmenter encore les performances au niveau du nœud. La tendance à augmenter le nombre de cœurs sur les supercalculateurs peut se traduire par une augmentation du nombre de cœurs par nœud de mémoire partagée, du nombre total de nœuds de la machine, ou des deux. Pour tirer toutes les performances rendues disponibles par la nouvelle génération de nœuds, NEMO doit être adapté pour une exécution sur un grand nombre de cœurs.

La quasi-totalité des codes ne parviennent pas à tirer parti du matériel des supercalculateurs aussi bien que la librairie *Linpack*. De plus, la complexité croissante des supercalculateurs fait qu'il est de plus en plus difficile pour les développeurs d'atteindre les performances attendues. En particulier, la scalabilité, c'est à dire la capacité d'un code à maintenir ses performances quand le nombre de processus augmente, est un enjeu majeur, car les modèles devront atteindre de bonnes performances sur des centaines de milliers, voire des millions de processus (Etiemble, 2018).

En plus de la contrainte matérielle, l'intérêt de la communauté pour une meilleure représentation des phénomènes à échelle spatiale fine pousse à une augmentation de la résolution spatiale, ce qui augmente le besoin en ressources de calculs et la nécessité d'un code informatique utilisant ces ressources de manière efficace.

1.3 Adapter le code NEMO aux supercalculateurs

La performance numérique des modèles climatiques est essentielle et doit être maintenue au meilleur niveau possible afin de minimiser le temps de résolution et ainsi l'énergie nécessaire à la résolution. En effet, pour une exécution sur un matériel identique, la consommation énergétique est largement proportionnelle à la durée de résolution de la simulation. Même si le supercalculateur n'effectue pas de calcul ou d'accès à la mémoire, sa consommation énergétique reste sensiblement identique. Réduire le temps de résolution d'une manière ou d'une autre réduit donc la consommation énergétique. De plus, comme dans tous les domaines, consommation énergétique est synonyme d'émission carbone.

Les modèles doivent évoluer en permanence afin de s'adapter aux nouvelles machines et exploiter au mieux leurs capacités. En l'occurrence, les machines actuelles tirent performances d'une architecture constituée d'une multitude de cœurs. Pour être compatible avec une exécution utilisant plusieurs cœurs, NEMO doit répartir les calculs de la résolution des équations sur l'ensemble de ces processus.

Il existe différentes librairies qui permettent aux développeurs de gérer facilement les liens entre les processus. OpenMP (*Open Multi-Processing*) est l'une de ces librairies. Elle ne peut répartir la charge de calcul sur plusieurs processus que si ces processus partagent un même cache mémoire. Cette librairie est donc utilisable pour des processus liés à des cœurs situés sur un même CPU, mais pas pour des processus localisés sur des nœuds ou CPU différents (voir Figure I.1). OpenMP n'est donc pas suffisant pour une exécution d'un code distribué sur plusieurs nœuds.

NEMO doit donc utiliser une autre librairie. La librairie *MPI (Message Passing Interface)* fournit des instructions pour transférer des données d'un processus à un autre, que les processus soient localisés sur un même CPU ou non. Les développeurs doivent alors prévoir la répartition de la charge de calcul et des données sur les processus ainsi que l'utilisation d'instructions MPI pour transférer des données lorsque cela est nécessaire. C'est cette stratégie qui a été choisie pour exécuter NEMO sur plusieurs processus.

Une nouvelle possibilité est envisagée, c'est l'utilisation conjointe de la librairie OpenMP entre les processus d'un même CPU et la librairie MPI pour les communications entre CPU. Si cette implémentation voit le jour, elle permettrait à NEMO de bénéficier des avantages de OpenMP tout en gardant le code compatible pour une exécution sur plusieurs CPU.

Dans NEMO, la répartition de la charge de calcul est effectuée en découpant le domaine de la simulation sur les dimensions horizontales. La grille est divisée horizontalement en rectangles comportant un nombre de points de grille aussi proches que possible les uns des autres. Les données associées aux variables physiques sont distribuées sur les processus. Chaque processus effectue des calculs et stocke en mémoire uniquement les données associées à la partie qui lui est assignée. On appelle sous-domaine MPI les parties du domaine issues de ce découpage, ils sont représentés par les rectangles colorés sur la figure I.3 à gauche.

La résolution des équations discrétisées sur la grille nécessite régulièrement des calculs faisant intervenir des données associées à des points adjacents, par exemple pour le calcul des dérivées. Pour que les calculs puissent être réalisés pour tous les points du sous-domaine, les processus doivent avoir accès aux valeurs des points voisins, y compris pour les points situés sur les bords de la décomposition. Pour répondre à ce besoin, ces valeurs doivent être accessibles directement et donc stockées directement dans la mémoire associée au processus. Le sous-domaine MPI est donc étendu d'un ou de plusieurs points de grille dans les dimensions horizontales. Cette extension est appelée le halo et recouvre des points dont les sous-domaines voisins ont la charge de calcul. Sur la figure I.3 à droite, le sous-domaine 4 est, par exemple, étendu d'une bande bleu clair en bas qui correspond à des données dont le processus associé au sous-domaine 1 a la charge. Par la suite, on utilise les directions cardinales pour désigner les positions des sous-domaines les uns par rapport aux autres. On dira ainsi que le sous-domaine 1 est au "sud" du sous-domaine 4.

Les calculs nécessitant les valeurs des points voisins ne peuvent pas être effectués sur les halos. À chaque fois qu'un tel calcul est réalisé sur le sous-domaine MPI, les valeurs des halos doivent être rapatriées par une communication avec le processus qui a effectué le calcul pour

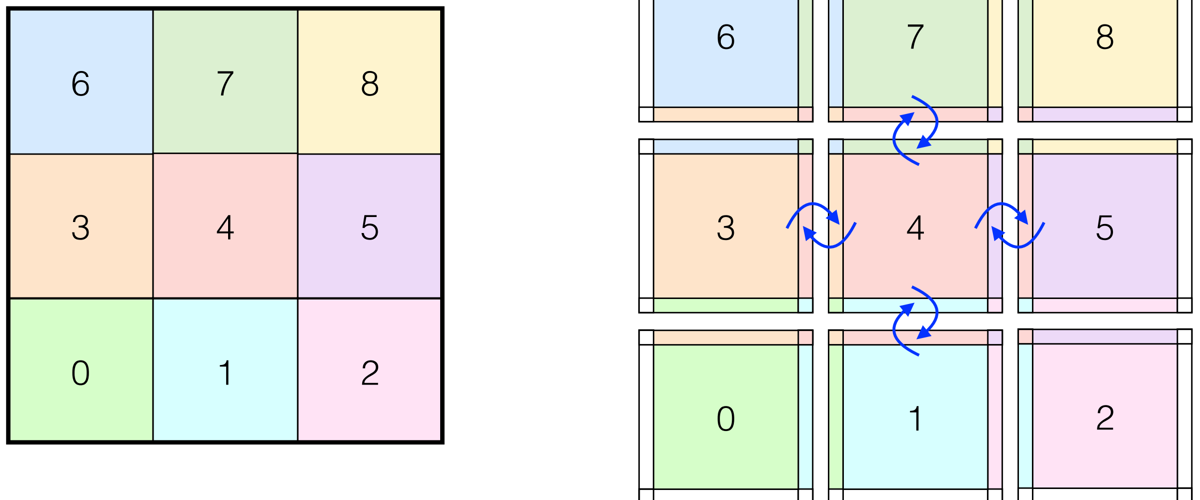


FIGURE I.3 – Exemple d'une décomposition horizontale d'un domaine global en 9 sous-domaines MPI. La ligne épaisse sur le panneau de gauche délimite le domaine global (sans halos) et les lignes fines délimitent les sous-domaines. À droite, les communications du sous-domaine 4 avec ses voisins sont représentées par les flèches bleues.

ces points. Ces échanges de mémoire entre processus sont effectués à l'aide de la librairie MPI. Un échange de données entre processus symbolisé par une flèche bleu du sous-domaine 1 vers le halo du sous-domaine 4 vient mettre à jour les données du halo après modification par des calculs uniquement réalisables par le processus 1. Il en va de même pour les voisins 3, 5 et 7 du sous-domaine 4. Pour les voisins dans les directions diagonales $(0, 2, 6, 8)$ le sous-domaine est aussi étendu pour inclure des données de ces sous-domaines, mais on verra au chapitre IV que l'échange de données avec les voisins diagonaux peux se faire de manière directe ou indirecte.

Les communications entre les processus sont coûteuses en temps de calcul et doivent donc être minimisées afin de maintenir la restitution temporelle des expériences numériques à son meilleur niveau. Nous nous attendons à ce que le besoin d'une telle réduction persiste à l'avenir.

Le chapitre II a pour objectif de mettre en place un dispositif d'évaluation précis et transparent de diagnostic des performances de NEMO.

Au chapitre III le dispositif mis en place fait apparaître les principales entraves limitant les performances du code.

Le chapitre IV explore une variété de schémas de communication visant à alléger les entraves mises en évidence au chapitre précédent et à la réduction des coûts des communications MPI.

Enfin, le chapitre V examine et optimise une stratégie de réduction du coût en calcul : le raffinement de maillage par zooms emboîtés.

CHAPITRE

II

Cadre et outils dédiés à l’optimisation de NEMO

1 Stratégie d'optimisation de NEMO

1.1 Axes d'optimisation de NEMO

L'amélioration constante de la physique des modèles, d'une part, et l'évolution de la technologie des superordinateurs, d'autre part, exigent que les performances de calcul des modèles soient continuellement revues et améliorées.

Les axes de développement visant l'augmentation des performances de NEMO sont multiples. Ils doivent couvrir tous les aspects du calcul : les lectures et écritures de fichiers, l'utilisation de la mémoire, la vectorisation des boucles, les communications ou encore la capacité d'exécution sur GPU.

Des développements sont en cours et certains ont été réalisés sur les lectures et écritures à travers des optimisations sur XIOS, l'outil dédié à cette tâche, sur la mémoire au moyen d'une réécriture en profondeur de NEMO pour permettre une meilleure utilisation des caches, mais aussi sur le portage sur GPU via PSYCLONE, un outil de réécriture automatique de code.

Chacun des axes de développement doit faire l'objet d'études et d'optimisations pour améliorer au mieux les performances. En effet, si chaque point est amélioré sauf un, ce dernier point devient en comparaison très limitant. Les communications deviennent donc un point critique à optimiser.

1.2 Phases de développement d'une optimisation

C'est un travail délicat que d'optimiser un modèle comme NEMO, qui est utilisé par une large communauté dont les profils vont du doctorant à l'expert en physique et modélisation du climat ou en océanographie opérationnelle. Ce travail doit permettre d'améliorer les performances tout en préservant l'accessibilité du code pour les climatologues qui l'utilisent et le développent mais qui ne sont pas nécessairement des experts en informatique. Ce travail d'optimisation s'inscrit dans ce cadre et nous avons réuni, dans cette étude, des auteurs aux profils très complémentaires :

océanographes, développeurs NEMO, ingénieurs spécialisés dans la modélisation climatique et les simulations frontières et ingénieurs purement HPC. Nous avons ainsi pu bénéficier de leurs conseils et de leur assistance dans leur domaine d'expertise.

Le mode opératoire que j'ai appliqué pour réaliser différentes optimisations comporte les étapes suivantes :

— Analyser le code et trouver les goulots d'étranglements de performances à l'origine de limitations de performance du code.

— Comprendre la raison précise de la faible performance des goulots d'étranglements identifiés.

— Concevoir et implémenter une optimisation visant à limiter l'impact des goulots d'étranglements, éclairé par l'analyse de l'étape précédente.

— Mesurer les performances du code avant et après l'optimisation pour confirmer que l'optimisation améliore effectivement les performances.

— Confirmer que les résultats physiques sont inchangés avant et après l'optimisation.

Mener à bien toutes ces étapes demande un certain nombre d'outils et de techniques que je présente dans la suite de ce chapitre. Mesurer le temps d'exécution et les différentes analyses du code demande d'exécuter le code sur une configuration donnée. La partie 2 discute la configuration qu'il convient d'utiliser pour les analyses de NEMO. La partie 4 expose ensuite différentes manières de mesurer et d'évaluer le temps d'exécution du code. Je présente les méthodes et outils d'analyse des performances dans la partie 3.

2 Configurations

Pour optimiser NEMO, il est indispensable de passer par une phase d'analyse des performances d'exécution du code et donc de choisir une configuration sur laquelle l'exécuter. Le choix de cette configuration a un fort impact sur les performances mesurées et analysées. Les configurations peuvent par exemple inclure la modélisation de la glace (avec le module ICE) ou de

la bio-géochimie (avec le module TOP-PISCES), changer considérablement le nombre de points de grille, utiliser des schémas numériques plus ou moins complexes et coûteux ou encore activer des fonctionnalités comme le calcul de la marée ou la présence de frontières ouvertes.

Mes travaux visent à l'amélioration des performances en priorité dans un certain contexte qui est celui de la nouvelle génération de supercalculateurs. Les configurations utilisées cherchent à mettre en avant les points bloquants pour adapter NEMO à ces architectures, notamment le nombre de points de grille dans les sous-domaines que nous appellerons la taille du sous-domaine. De plus, les configurations seront simplifiées le plus possible pour permettre une analyse plus probante en isolant les facteurs intervenant sur les performances.

2.1 La configuration BENCH

Pour nos configurations, on se base sur la configuration BENCH présentée dans Irrmann et al. (2022). Cette configuration a été développée dans le but de fournir à la communauté des développeurs de NEMO une configuration dédiée à l'étude des performances.

L'exploration des performances numériques du code nécessite une configuration appropriée. Elle doit garder l'utilisation du code proche du mode de production, c'est-à-dire proche des configurations variées utilisées par la communauté pour permettre une représentation fidèle des différentes limitations rencontrées par les utilisateurs.

NEMO fournit différentes configurations basées sur un modèle océanique de base : par exemple, des grilles globales (famille ORCA) ou régionales, avec différentes résolutions spatiales verticales et horizontales. Différentes composantes peuvent en outre être ajoutées au noyau dynamique de l'océan : la glace de mer (c'est-à-dire ICE) et/ou la bio-géochimie (c'est-à-dire TOP-PISCES). Une analyse complète des performances doit donc être capable d'examiner les routines de chaque module afin de délivrer un message pertinent sur les performances de calcul du modèle NEMO à l'ensemble de la communauté des utilisateurs. D'autre part, comme l'a souligné l'ancien consortium RAPS (Mozdzynski, 2012), la réalisation d'un tel exercice d'ana-

lyse comparative doit rester simple, car il est souvent effectué par des personnes n'ayant qu'une connaissance de base de NEMO ou de l'océanographie physique (par exemple, les experts en calcul haute performance et les vendeurs de matériel).

Nous détaillons ici la configuration NEMO spécifiquement développée dans Irrmann et al. (2022) pour simplifier les futures activités d'évaluation des performances en répondant à la double contrainte de (1) être léger et trivial à utiliser et (2) permettre de tester n'importe quelle configuration NEMO (taille, modèle de périodicité, composants, paramétrisations physiques...). A l'opposé du concept de dwarf (Müller et al., 2019), cette configuration englobe toute la complexité du modèle et permet de répondre aux questions actuelles de la communauté. La configuration BENCH a été utilisée dans Maisonnave et Masson (2019) pour évaluer les performances de la configuration globale (famille ORCA).

2.1.1 BENCH : description générale

Cette nouvelle configuration, appelée BENCH, est disponible dans le répertoire *tests/BENCH* de la distribution NEMO.

BENCH est basé sur un bassin rectangulaire à fond plat, ce qui permet de s'affranchir de tout fichier de configuration en entrée et donne la possibilité de définir les dimensions du domaine simplement via des paramètres de namelist.

Les conditions initiales et les champs de forçage ne correspondent à aucune réalité physique et ont été spécifiquement conçus (1) pour assurer un maximum de robustesse et (2) pour faciliter la manipulation de BENCH car ils ne nécessitent aucun fichier d'entrée. En conséquence, les résultats de BENCH n'ont aucune signification d'un point de vue physique et ne devraient être utilisés qu'à des fins d'évaluation des performances. La température et la salinité du modèle sont presque constantes partout avec une légère stratification qui maintient la stabilité verticale du modèle. Nous ajoutons à chaque point de chaque niveau horizontal une perturbation suffisamment petite pour maintenir la solution très stable tout en laissant les processus d'ajustement océanique

se produire pour maintenir la quantité de calculs associée à un niveau réaliste. Cette perturbation garantit également que chaque point du domaine a une valeur initiale unique de température et de salinité, ce qui facilite la détection de problèmes MPI potentiels.

Afin de simplifier au maximum son utilisation, la configuration BENCH ne nécessite aucun fichier d'entrée. En outre, l'absence de fichiers d'entrée (ou de sortie) empêche toute perturbation de la mesure de performance par l'accès au disque.

2.1.2 Flexibilité de BENCH

Tout schéma numérique et paramétrage NEMO peut être utilisé dans BENCH. Une sélection de paramètres de namelist est fournie avec BENCH pour conférer les propriétés numériques correspondant aux configurations globales les plus courantes, les grilles ORCA. De même, les modules ICE de glace de mer et TOP-PISCES peuvent être activés ou désactivés en choisissant les paramètres appropriés de la namelist et les clés CPP lors de la compilation.

Toutes les conditions fermées ou périodiques peuvent être utilisées dans BENCH et spécifiées par un paramètre de la namelist (nn_perio). L'utilisateur peut choisir entre des limites fermées, des conditions périodiques Est-Ouest, des conditions bi-périodiques ($nn_perio = 7$). Notez que les conditions bi-périodiques garantissent que tous les sous-domaines MPI ont le même nombre de sous-domaines voisins s'il n'y a pas de points terrestres. C'est un moyen pratique de réduire le déséquilibre de la charge de calcul associé à un sous-domaine. La spécificité des conditions périodiques dans les grilles globales ORCA a un impact important sur les performances.

Dans NEMO, les calculs sont presque toujours effectués sur chaque point de grille qu'il soit un point terrestre ou océan. Un masque est appliqué suite aux calculs pour prendre en compte les points de grille terrestres s'il y en a. Par conséquent, la quantité de calculs est la même avec ou sans bathymétrie, il s'ensuit que les performances de calcul de BENCH et d'une configuration réaliste sont extrêmement proches.

La stabilité de la configuration BENCH permet d'effectuer des expérimentations innovantes pour estimer facilement l'impact d'optimisations potentielles. BENCH peut en effet fonctionner pendant un grand nombre de pas de temps même si les appels aux communications MPI sont désactivées. Par exemple, il est possible d'examiner la scalabilité du code sans aucune communication en ajoutant simplement une commande `RETURN` au début de la routine de communication NEMO. Alternativement, il est possible d'ajouter un test `IF` avec un modulo pour effectuer ou non cette commande `RETURN`. Une telle expérience donne une idée des performances du code si l'on peut réduire les communications d'un facteur N. Les résultats physiques sotn incorrects, mais l'exécution se poursuit grâce à la stabilité de la configuration. On peut ainsi tester les avantages potentiels de nombreuses idées avant de dédier beaucoup de temps à leur implémentation.

La possibilité d'activer et de désactiver la plupart des modules et routines de NEMO sans passer par une phase d'implémentation permet une évaluation et un diagnostic faciles des différents constituants de NEMO.

2.2 Configuration BENCH_OCE et son utilisation

La configuration BENCH est adaptée dans cette sous-section pour analyser plus simplement les points qui nous intéressent. On analyse les performances de NEMO pour un grand nombre de cœurs. Dans un tel cas de figure, les sous-domaines MPI tendent à contenir un petit nombre de points de grille. L'impact de la mémoire sur les performances du code est moins important car les tableaux gérés par un cœurs sont de petite taille, on choisit donc de ne pas inclure le module de bio-géochimie (TOP-PISCES) qui a principalement un impact sur les performances en raison de l'impact mémoire engendré par l'utilisation de nombreux tableaux 3D propre au module.

Le module de gestion de la glace, ICE, est lui aussi désactivé. Ce module introduit moins de tableaux que pour la bio-géochimie. Les algorithmes qui interviennent dans ce module ont des caractéristiques du point de vue performances similaires à celles des algorithmes du cœur dynamique de NEMO. Par exemple, l'algorithme de calcul de rhéologie de la glace comporte une boucle avec beaucoup d'itérations où plusieurs communications échangent des faibles vo-

lumes de données. Cet algorithme effectue des communications à haute fréquence, c'est-à-dire des communications entrecoupées de courtes phases de calcul. Ces caractéristiques le rendent semblable à l'algorithme de calcul du gradient de pression de surface qui comporte aussi des communications hautes fréquences sur des faibles volumes de données. La désactivation du module ICE ne crée donc pas d'angle mort dans notre analyse. De plus, les optimisations ayant trait au calcul du gradient de pression de surface pourront être transposées au calcul de la rhéologie. Se passer du module de gestion de la glace permet de simplifier l'objet de notre étude sans le modifier substantiellement.

Les entrées et sorties de fichiers sont un angle d'analyse important pour des simulations à grand nombre de cœurs. Le code NEMO prévoit une écriture des variables physique par chacun des processus MPI. Cet accès au système de fichier par un grand nombre de processus pose rapidement des problèmes et ralentit fortement l'exécution de la simulation. La librairie XIOS a été développée dans le but de résoudre ce problème (Meurdesoif, 2018). Cette librairie est couramment utilisée pour des simulations NEMO disposant d'un grand nombre de cœurs. Les enjeux de performances liés aux entrées et aux sorties sont alors propres à la librairie XIOS et à son interaction avec NEMO. Des optimisations concernant la librairie XIOS sont en effet importantes mais font appel à des expertises légèrement différentes que les nôtres, ce ne sera donc pas directement l'objet de notre étude.

Une configuration créée à partir de BENCH et excluant ces 3 modules peut être générée simplement en utilisant la commande `makenemo`. Soit BENCH_OCE le nom de cette configuration,

```
./makenemo -m <fcm> -a BENCH -n BENCH_OCE -d "OCE" del_key "key_si3 key_top"
```

est la commande permettant de créer et de compiler cette configuration. `<fcm>` doit être remplacé par la partie centrale du nom du fichier `.fcm`. Par exemple, si le nom du fichier `.fcm` dans le répertoire `arch/` donnant les informations de compilation correspondant au supercalculateur est `arch-X64_supercomputer.fcm`, alors `<fcm>` doit être remplacé par `X64_supercomputer`.

Nos tests n'utilisent aucun fichier d'entrée définissant la grille ou les forçages. Les conditions limites choisies sont les conditions bi-périodiques ($nn_perio = 7$) pour que chaque sous-domaine MPI communique avec le même nombre de voisins. Les conditions bi-périodiques imposent que les sous-domaines situés sur le bord du domaine communiquent avec des sous-domaines situés à l'opposé du domaine. Par exemple, un sous-domaine situé à la limite "sud" est voisin avec des sous-domaines situés à la limite "nord" du fait des conditions bi-périodiques. De plus, la configuration ne contenant aucun point terre, aucune communication entre deux voisins ne peut être supprimée sous prétexte qu'elle n'échange que des points terre. Les sous-domaines MPI ayant tous le même nombre de voisins, les communications sont gérées de manière identique pour tous et le domaine global ne comporte pas de frontière. Il n'y a donc pas de processus MPI qui aurait pour charge supplémentaire la gestion de la frontière sur son sous-domaine. Ces particularités permettent un équilibrage de charge entre processus MPI théoriquement parfait puisque chaque processus exécute exactement les mêmes instructions à chaque pas de temps.

Résumons les caractéristiques de la configuration BENCH_OCE et des conditions dans lesquelles nous l'utilisons pour l'évaluation des performances :

- Petits sous-domaines MPI (10 par 10 ou 30 par 30 points dans les dimensions horizontales)
- Pas de bio-géochimie (TOP-PISCES)
- Pas de glace (ICE)
- Pas d'entrées/sorties (XIOS)
- Conditions bi-périodiques
- Pas de points terre

BENCH_OCE ainsi construite est une configuration issue de BENCH simplifiée pour une analyse de simulations à grand nombre de cœurs qui reste pertinente et représentative d'un grand nombre de configurations de production utilisant NEMO.

2.3 Configuration TSUNAMI

On verra par la suite qu'une routine dans la configuration BENCH_OCE est particulièrement coûteuse en temps de calcul, c'est la routine *dynspg_ts* (voir section 3 du chapitre III), qui calcule le gradient de pression de surface. Le calcul du gradient de pression de surface a pour particularité de ne manier que des tableaux en deux dimensions ce qui limite fortement les ralentissements liés à la mémoire. En revanche, cette routine de calcul comporte un grand nombre de communications MPI. Du fait des spécificités de cette routine par rapport au reste du code, il est intéressant d'en estimer ses performances.

Cette routine est isolée dans la configuration TSUNAMI où seul le gradient de pression est calculé. Les autres caractéristiques de cette configuration sont identiques à la configuration BENCH_OCE. Résumons les caractéristiques de la configuration TSUNAMI et des conditions dans lesquelles nous l'utilisons pour l'évaluation des performances :

— Petits sous-domaines MPI (10 par 10 ou 30 par 30 points dans les dimensions horizontales)
— Uniquement le calcul du gradient de pression de surface
— Conditions bi-périodiques
— Pas de points terre

3 Profilage de NEMO

Le profilage est une analyse précise des performances de l'exécution d'un code informatique. C'est une étape fondamentale dans le processus d'optimisation d'un code qui permet à la fois d'identifier les goulots d'étranglements aux performances et de comprendre quel type de limitation entre en jeu (gestion de la mémoire, communication, charge de calcul, écriture ou lecture de fichier). Il faut cependant être conscient que plus le profilage est précis, plus il impacte la simulation en elle-même. Les profileurs les plus puissants sont des outils qui peuvent être lourds et complexes à utiliser (par exemple Paraver-Extrae). Certains des profileurs sont spécifiques à des

compilateurs et/ou architectures matérielles et ne sont donc pas toujours disponibles sur toutes les machines (par exemple Vtune, Intel Advisor). Ils ne présentent donc pas que des avantages en comparaison aux méthodes simples et légères présentées précédemment. Les deux types d'approche sont complémentaires et doivent être utilisées conjointement et à bon escient.

3.1 Les enjeux du profilage

Les opérations permettant le profilage du code étant effectuées sur le même matériel que l'exécution du code lui-même, l'analyse du code modifie les performances qui sont l'objet même de l'analyse. Mesurer une partie du code peut ralentir son exécution et détériorer artificiellement ses performances par rapport aux autres parties. Le profilage peut ainsi produire une vue déformée du code. Le coût en temps d'exécution qu'ajoute le profilage est donc un facteur important à prendre en compte.

Les données issues d'un profilage peuvent être produites de plusieurs manières.

Une approche est d'interrompre le déroulement du code à intervalles réguliers pour permettre au profilage de récupérer l'état de l'exécution (voir figure II.1 en bas). On accède ainsi à une estimation du temps passé dans les routines qui composent le code mais aussi une estimation de certaines métriques propres au matériel informatique. La possibilité de changer l'intervalle auquel est effectué le profilage permet, en le diminuant d'augmenter la précision des estimations produites, et en l'augmentant, de diminuer l'impact du profilage sur les performances. Une valeur adéquate de ce paramètre doit donc être trouvée. L'un des intérêts de ce profilage est la possibilité d'avoir un impact non seulement faible sur les performances, mais surtout un impact équivalent sur toutes les parties du code ce qui permet d'avoir une vue peu déformée du code. Le fait que ce profilage nécessite peu de mémoire participe au faible impact de la méthode. Il peut cependant exister une déformation pour des évènements très courts. Lorsque l'échantillonnage tombe sur un évènement court tout l'intervalle lui est attribué ce qui peut ne pas être représentatif de son temps d'exécution (Hardware Event-based Sampling Collection, s. d.). Par ailleurs, le problème

de la précision de la mesure peut être amélioré en augmentant le temps de la simulation analysée en ajustant le nombre de pas de temps. Par la suite ce type de profilage sera désigné sous le terme de profilage par échantillonnage.

Une autre possibilité est de mesurer les instants de début et de fin de certains évènements choisis pour déterminer leur durée et possiblement la chronologie des évènements (voir figure II.1 en haut). Cette manière de faire permet de mesurer précisément chacun des évènements au fil de la simulation et pas uniquement la valeur moyenne. Suivant les évènements mesurés, l'impact sur les performances peut être conséquent. De plus, un grand nombre d'évènements courts demande d'utiliser un grand nombre de mesure des instants de début et de fin. Ces mesures prenant du temps d'exécution, le poids moyen des évènements courts est surévalué. Ce type de profilage est appelé profilage par instrumentation.

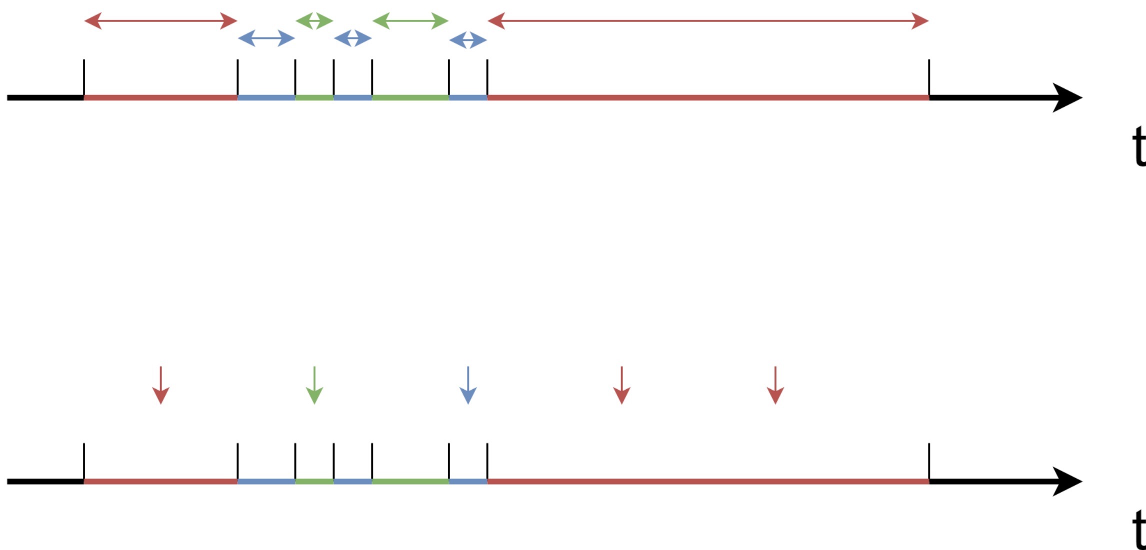


Figure II.1 – Schéma des deux types de mesures pour un profilage. En haut le profilage par instrumentation et en bas par échantillonnage. Les axes en gras indiqués t sont les axes temporels coloriés en différentes sections représentant des phases de simulation différentes. Les flèches colorées correspondent aux instructions de profilage.

Typiquement, les évènements mesurés sont relatifs au code exécuté (nombre d'appel et temps d'exécution des routines composant le code), à des évènements liés au matériel informatique (défaut de cache, cycle d'horloge, nombres d'instructions) ou au réseau de communication (taille et nombre des messages communiqués).

3.2 Utilisation d'outils de profilage

Les techniques de profilage exposées ci-dessus peuvent être mises en œuvre dans des outils de profilage à disposition des développeurs. Un certain nombre de ces outils sont utilisés dans la suite de ce travail. Résumons rapidement leurs caractéristiques.

3.2.1 Gprof

Gprof est un outil de GNU et est donc largement disponible et compatible avec de nombreux compilateurs. C'est un profileur par échantillonnage très simple d'utilisation. L'intervalle utilisé pour collecter les données est réglable par l'utilisateur. Le profileur fournit aussi des sorties adaptées pour la visualisation des résultats. Les informations collectées sont les temps d'exécution et le nombre d'appels aux routines qui composent le code, les librairies utilisées ne sont pas profilées. Gprof est utilisé dans ce travail pour avoir une idée, peu déformée par le profilage, du poids de chaque routine dans le temps d'exécution.

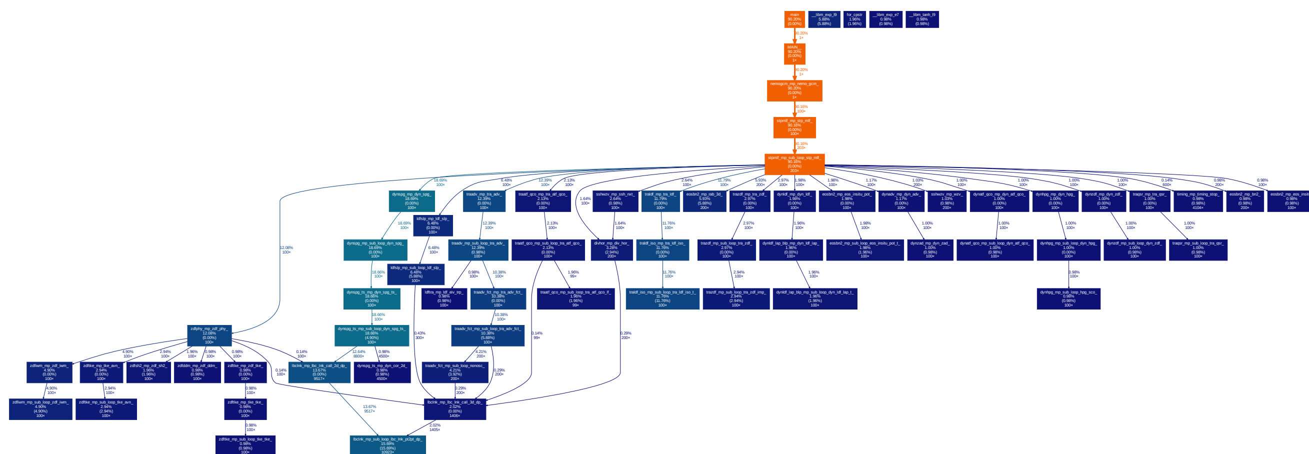


FIGURE II.2 – Profilage gprof d'une exécution d'une simulation NEMO BENCH_OCE sur un nœud Skylake avec des sous-domaines MPI de 10 points par 10 points.

La figure II.2 montre le profilage, effectué grâce au logiciel gprof, des principales routines de NEMO pour une exécution sur des petits sous-domaines MPI (10 points par 10 points). La figure II.3 consiste en un agrandissement de ce même profilage sur les routines du calcul du gradient de pression. Chaque case représente une routine, les cases sont coloriées suivant le pourcentage du temps d'exécution de cette routine par rapport au temps d'exécution total, clair pour les temps

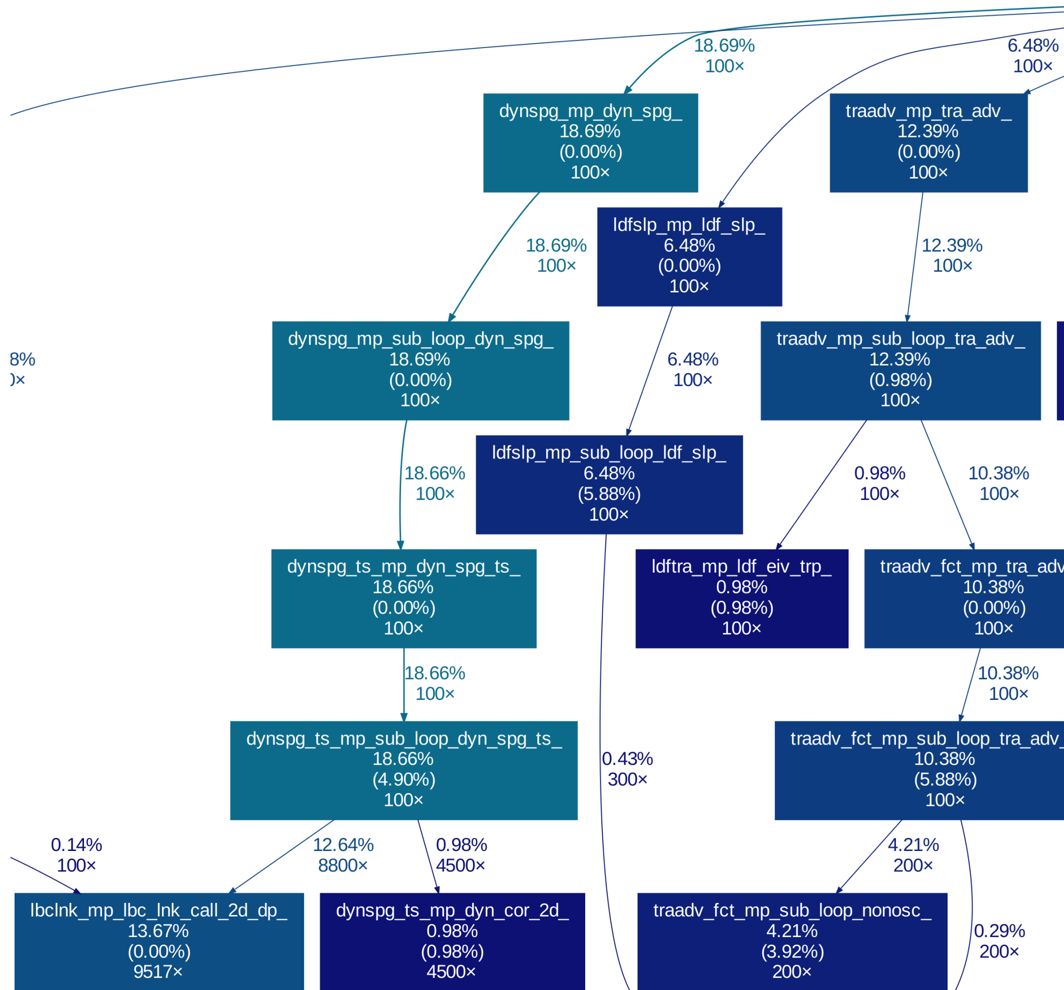


FIGURE II.3 – Profilage gprof d'une exécution d'une simulation NEMO BENCH_OCE sur un nœud Skylake avec des sous-domaines MPI de 10 points par 10 points. Zoom sur les routines du calcul du gradient de pression (*dynspg*).

élevés et sombre pour les faibles. Ce pourcentage est aussi indiqué par le premier chiffre dans la case. Le pourcentage indiqué ensuite entre parenthèse est le temps passé dans cette routine excluant le temps passé dans les sous-routines appelées. Les flèches entre cases représentent les appels d'une routine à une autre avec le nombre d'appels suivi d'un x. Le pourcentage du temps d'exécution de la routine appelée par rapport au temps d'exécution total de la simulation résultant des appels symbolisés par la flèche est aussi indiqué.

Ce profilage élémentaire montre que la plupart des routines principales de NEMO ont des temps d'exécution similaires d'environ 1% à 5% à l'exception de trois routines aux alentours de 12%, mais aussi de la routine de calcul du gradient de pression *dynspg_mp_dyn_spg_* à presque 19% du temps d'exécution total. La plus grande partie du temps d'exécution du cal-

cul du gradient de pression de surface est due aux communications MPI avec la routine (*lbclnk_mp_lbc_lnk_call_2d_dp_*). Notons que le profilage de la figure II.2 a été effectué après l'optimisation présentée dans la section 1 du chapitre III qui est plus amplement détaillée à l'annexe A à la section 2. De plus, cette routine de calcul fait l'objet d'une analyse dans la section 3 du chapitre III

3.2.2 APS

Le profileur APS, pour *Application Performance Snapshot*, est un profileur de *Intel* qui est compatible avec les compilateurs *Intel*. Il utilise une méthode d'échantillonnage pour collecter des métriques sur le temps passé dans MPI, efficacité de l'accès à la mémoire et vectorisation. Toutes ces métriques sont moyennées sur la totalité de l'exécution du code, aucune donnée sur la chronologie de l'exécution est disponible. La phase d'initialisation n'est pas exclue de ce profilage et peut avoir des caractéristiques différentes du reste du code que l'on veut profiler. Le temps de simulation doit être suffisamment long pour que la phase d'initialisation ait un impact limité sur les performances. APS est utile pour avoir une vue très rapide des performances mémoire, vectorisation et MPI.

Le profilage affiché figure II.4 présente les possibilités offertes par APS. Les communications MPI sont désignées comme étant le principal facteur limitant des performances dans ces conditions.

3.2.3 Extrae et Paraver

Extrae et Paraver sont deux outils complémentaires développés par le Barcelona Supercomputing Center disponibles sur une grande variété de plateformes. Extrae ajoute des sondes dans l'exécutable pour extraire des données de performances et Paraver est un outil de visualisation et d'analyse des données fournies par Extrae.

Extrae utilise diverses méthodes pour recueillir les données, à la fois par échantillonnage et par instrumentation. Des données très diverses sont disponibles et placées sur un axe temporel ce qui permet, à l'aide de Paraver, de visualiser la chronologie des évènements, mais aussi

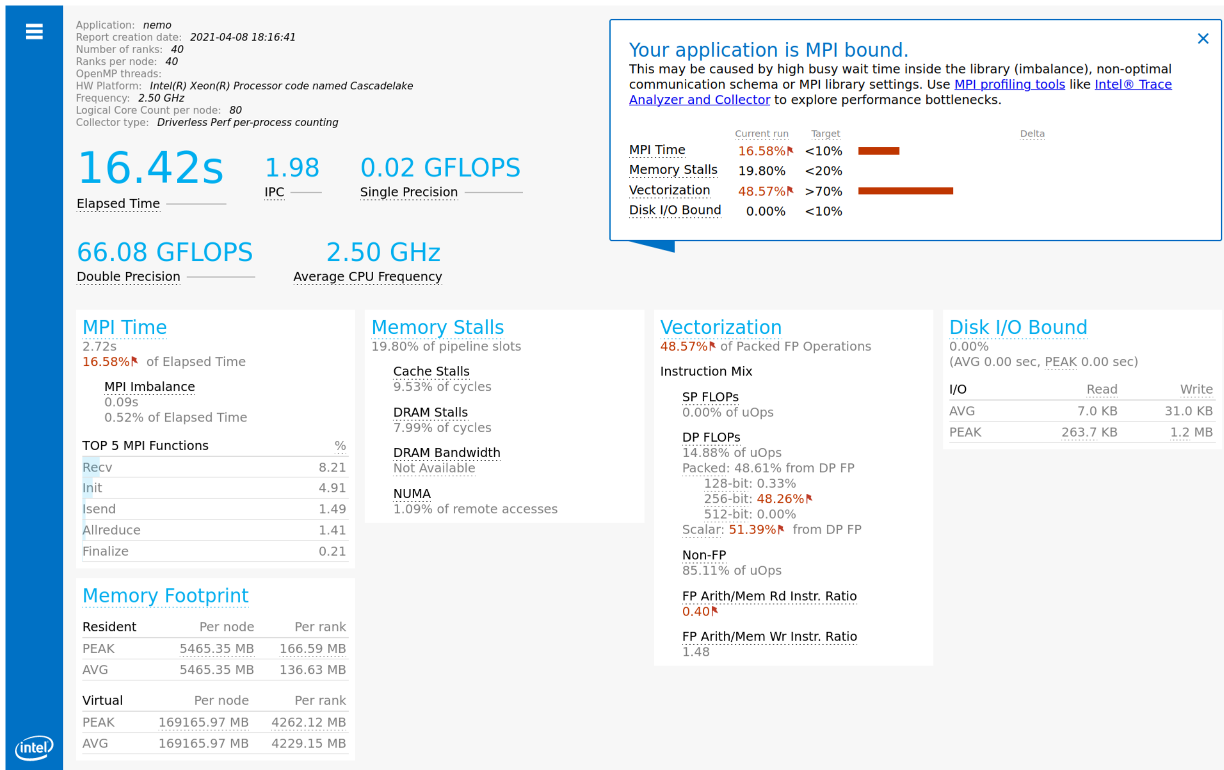


FIGURE II.4 – Profilage APS d'une exécution d'une simulation NEMO BENCH_OCE sur un nœud Cascadelake avec des sous-domaines MPI de 10 points par 10 points.

de calculer les moyennes des données recueillies par échantillonnage sur une section temporelle déterminée par l'utilisateur. Des analyses statistiques poussées sont aussi disponibles ainsi qu'une grande variété de graphiques. Par ailleurs, Paraver offre des possibilités pour comparer des codes ou estimer les performances dans des hypothèses qui ne peuvent pas être réalisées ou difficilement. Par exemple, Paraver peut reproduire le profil qu'aurait l'exécution si elle avait lieu sur un supercalculateur avec un InterConnect, c'est-à-dire un réseau de connexion entre nœuds, plus rapide voire infiniment rapide.

Extrae et Paraver sont des outils beaucoup plus puissants, complets et complexes qui permettent une vision approfondie du code profilé.

L'inconvénient de ces profilages est que les sondes sont largement placées automatiquement et potentiellement en grand nombre ce qui impacte l'exécution du code. De plus, on peut vouloir mesurer des choses légèrement différentes que ce qui est mesuré par défaut et parvenir à l'analyse visée.

3.3 Profilage spécifique à NEMO

Pour limiter au plus l'impact du profilage sur le code et examiner un aspect donné du code, il est possible d'implémenter un profilage adapté à une analyse spécifique de NEMO. De plus, réaliser soit-même l'implémentation d'un profilage permet de maîtriser tous ses aspects et de savoir comment son utilisation impacte l'exécution et ainsi d'être certain des phénomènes analysés.

C'est ce que j'ai réalisé pour étudier un phénomène présenté à la section 2.3. Le profilage est effectué à plus fine échelle temporelle pour chacun des processus de la simulation. C'est un profilage par instrumentation réalisé en plaçant directement des directives dans le code pour mesurer le temps d'exécution des différentes sections. Chaque processus mesure le temps à l'horloge lorsqu'il entre et sort de la routine de communication, ce qui permet de mesurer le temps passé dans la routine de communication et dans les routines de calcul entre les communications. Le profilage peut être encore précisé en mesurant chacune des sections de la routine de communication.

Une série de précautions est prise pour garantir la fiabilité des données générées. Premièrement, la précision des mesures est évaluée. Deuxièmement, les données générées par ces mesures sont traitées pour limiter l'impact sur les performances de la simulation en tenant compte de trois aspects traités ci-dessous : la fréquence de sortie, la simultanéité des fichiers et le volume de données. Enfin, plusieurs algorithmes sont étudiés pour que les mesures prises sur différents cœurs et sur différents nœuds soient cohérentes entre elles et puissent être comparées.

3.3.1 Précision de la mesure du temps

Jusqu'à présent c'est la routine *MPI_Wtime* qui était utilisée dans NEMO à la fois pour estimer les performances, mais aussi pour réaliser des profilages.

Cette fonction n'est pas assez précise pour réaliser des profilages au niveau des communications. La fonction *MPI_Wtick* donne la résolution de *MPI_Wtime* qui est d'environ $10^{-3}s$, ce qui n'est pas suffisant pour des évènements qui peuvent être plus court. De plus, *MPI_Wtime* a un défaut majeur : les processus liés à des cœurs sur le même nœud n'auront pas d'horloge syn-

chronisée. En effet *MPI_Wtime* indique le temps écoulé depuis un moment arbitraire propre à ce processus. Selon la documentation, la variable *MPI_WTIME_IS_GLOBAL* indique si les horloges sont synchronisées, mais elle semble peu fiable au mieux.

La fonction *system_clock* est choisie pour remplacer *MPI_Wtime*. Selon la documentation, la précision de *system_clock* est de l'ordre de la fréquence de l'horloge qui est typiquement de quelques Ghz, la précision est donc inférieure à la nanoseconde. J'ai mesuré la durée moyenne d'exécution de *system_clock* en appelant à la suite un grand nombre de fois la routine. En pratique, la durée d'exécution de *system_clock* est donc de l'ordre de 10^{-8}, le temps mesuré est un instant quelconque pris dans l'intervalle de cette exécution. Par conséquent, la précision réelle de *system_clock* est de l'ordre de 10^{-8}. De plus, contrairement à *MPI_Wtime* la fonction *system_clock* possède la caractéristique de partager une horloge sur un nœud.

La mesure fournie par *system_clock* correspond au temps écoulé depuis le 1er janvier 1970, une partie de la précision est utilisée pour des informations largement inutiles et la partie utile peut être perdue. Pour conserver cette précision le résultat est conservé sous la forme d'un *integer(8)* car le convertir directement en un double résulterait en une énorme perte de précision. Lors du post-traitement des données, nous soustrayons le nombre de secondes retourné par le premier appel du processus 0 à *system_clock* à toutes les données avant de les convertir en double.

3.3.2 Limiter le surcoût de la mesure

Le nombre de mesures doit être limité au cours de la simulation, sinon on prend le risque de trop ralentir l'exécution. De plus, le nombre de mesures doit être le même entre deux versions du code pour pouvoir être comparées, sans quoi la version avec le plus de mesures se trouvera ralentie par l'instrumentation. J'ai rencontré ce problème lors de l'évaluation des performances des schémas de communication (voir chapitre IV section 3.1).

Par ailleurs, l'écriture de ces données dans un fichier de sortie a, elle aussi, un coût en temps d'exécution. Pour limiter cet effet les données ne sont pas écrites à chaque pas de temps, elles ne peuvent pas non plus être écrites en une seule fois à la fin de la simulation sinon la quantité de données en mémoire pourrait devenir trop importante et perturber le déroulement de l'exécution.

Un paramètre est ajouté dans la namelist pour définir la fréquence d'écriture des mesures offrant le meilleur compromis entre le surcoût lié à l'écriture ou au stockage des données. Chaque cœur stocke et écrit, dans un fichier qui lui est propre, les mesures qu'il récolte durant son exécution. Cela permet d'éviter qu'un grand nombre de processus écrivent dans le même fichier et entrent en concurrence les uns avec les autres. Pour limiter la quantité de données écrites, les données restent en mémoire sous la forme d'*integer* (nombres entiers) et les fichiers sont écrits en binaire au format NetCDF ce qui facilite les analyses à suivre.

Notons que malgré toutes ces précautions les performances du système de fichier sur lequel va s'effectuer l'écriture peuvent entrer en jeu, il faut notamment prendre garde à écrire les fichiers dans un système de fichier adapté. Par exemple un système de fichier peut être approprié pour un stockage à longue ou moyenne durée mais pas adapté à une utilisation intensive.

On compare le temps d'exécution d'un pas de temps médian pour une simulation avec et sans les mesures de profilage. Sur les configurations TSUNAMI et BENCH_OCE avec 10 nœuds AMD Epyc 7763 (64 cœurs par nœuds), les différences de temps d'exécution du pas de temps médian entre différentes expériences sont de $\approx 1\%$. Cette variation est bien supérieure à la différence du pas de temps médian entre les exécutions avec et sans mesures de $\approx 0.2\%$. Sur ce test rudimentaire les mesures de profilage n'ont pas d'impact sur les performances.

Une fois les données écrites, elles peuvent être directement utilisées pour calculer les temps d'exécutions des sections délimitées par les mesures. Pour comparer les instants où sont pris les mesures entre les cœurs il faut traiter les mesures pour les resynchroniser.

3.3.3 Algorithmes de synchronisation de l'horloge des nœuds

On a vu que la fonction *system_clock* partage la même horloge pour tous les processus associés à un nœud, c'est-à-dire que deux processus qui extraient le temps à l'horloge du nœud au même instant auront la même valeur. Ce n'est pas le cas pour des nœuds différents qui ont des horloges différentes.

Si l'on veut analyser l'évolution temporelle des processus sur des simulations utilisant plusieurs nœuds, alors on doit disposer d'un algorithme permettant de mesurer le décalage d'horloge de différents nœuds. Ce décalage est mesuré au cours de l'exécution ou lors de l'analyse, puis il est retiré des timings sauvegardés afin de synchroniser a posteriori les horloges des nœuds. La mesure du décalage doit être suffisamment précise pour permettre une étude des phénomènes que l'on souhaite analyser. Les communications dans NEMO pouvant être complétées dans un intervalle de temps de l'ordre de $10^{-5}s$, il faut que la mesure de ce décalage soit au moins aussi précise.

J'ai testé et développé différents algorithmes visant à cette synchronisation, ce travail est présenté à l'annexe B. L'un d'eux permet d'obtenir une estimation du décalage d'horloge de l'ordre de $10^{-6}s$ suffisante pour une analyse des phénomènes de désynchronisation des processus principalement à des fins de compréhension. Toutefois, cette précision ne permet pas d'utiliser cet outil pour quantifier rigoureusement la désynchronisation et servir à évaluer les performances du code, par exemple pour différents schémas de communication.

Ce profilage s'ajoute aux outils à notre disposition pour étudier le code sous différents points de vue. Gprof permet de quantifier le poids des principales routines du code, APS d'étudier les caractéristiques sur l'ensemble de l'exécution y compris relativement à l'utilisation du matériel et Extrae et Paraver d'examiner le code sous de nombreux aspects. Le profilage "à la main" est utilisé pour inspecter une partie du code spécifique, les communications dans notre cas, avec un impact sur l'exécution minime. Par ailleurs, cette étude nous a permis de corriger les problèmes du profilage par défaut de NEMO qui ralentissait l'exécution et empêchait la bonne comparaison des performances des schémas de communications.

4 Évaluation des performances

L'évaluation des performances est une composante essentielle du processus d'optimisation d'un code. D'une part, il convient d'évaluer les performances d'un code avant et après une optimisation pour s'assurer qu'elle apporte bien les améliorations attendues. D'autre part, nous

verrons par la suite qu'une évaluation fine des performances apporte une connaissance plus ou moins approfondie du code qui permet d'élaborer des solutions aux problèmes de performance rencontrés.

Évaluer les performances semble a priori quelque chose de trivial, mais en réalité, plusieurs problèmes se présentent :

— Quelle grandeur est réellement pertinente à mesurer ? La durée totale de l'exécution ou la durée moyenne de l'exécution des pas de temps ? Faut-il exclure de la mesure les phases d'initialisation et de finalisation ?

— Quelle est la variabilité associée à la mesure des performances ? Quelles sont les sources potentielles de cette variabilité ? Comment limiter au maximum cette variabilité afin de pouvoir comparer les performances mesurées en toute confiance ?

4.1 Temps total de simulation

Une manière usuelle d'évaluer les performances est de mesurer le temps total de la simulation. En fonction de la technique de mesure choisie ce temps représente des choses différentes.

Une première possibilité est d'utiliser la fonction Bash `time` qui mesure le temps d'exécution du script appelé à sa suite. Si `srun ./nemo` est la commande Bash qui lance la simulation NEMO, `time srun ./nemo` est la commande qui permet d'obtenir le temps passé dans la simulation NEMO. Plus précisément la commande `time` renvoie trois mesures du temps d'exécution *Real*, *User* et *Sys*. C'est la première mesure, *Real*, qui convient pour l'évaluation des performances. *Real* mesure le temps entre le début et la fin de l'exécution de la commande.

Cette première possibilité a pour particularité d'inclure dans le temps d'exécution le temps d'initialisation et de finalisation de la simulation. Dans notre cas cela pose un problème. En effet, l'initialisation comprend plusieurs phases qui peuvent être très variables en temps de calcul et compliquées et inutiles à analyser. L'initialisation est principalement constituée de la lecture des fichiers d'entrées (namelist et éventuels fichiers de configuration), de l'allocation en mémoire des principales variables et de l'initialisation des communications MPI. Comme l'initialisation et la

finalisation sont des parties négligeables en temps de calcul dans des simulations de production qui comportent un grand nombre de pas de temps, ce n'est pas sur ces parties que les efforts d'optimisations se concentrent. Les simulations qui visent à évaluer les performances comportant un nombre de pas de temps limité, l'impact des phases d'initialisation et de finalisation sont amplifiées et doivent donc être exclues.

Une évaluation fine des performances ne peut donc se contenter d'une simple utilisation de la commande `time`. Il est plus convenable de mesurer la durée entre le début du premier pas de temps et la fin du dernier pas de temps, en excluant ainsi les phases d'initialisation et de finalisation. L'implémentation consiste simplement à placer des commandes de mesure de l'horloge de la machine à ces deux endroits, la différence entre les valeurs obtenues donne le temps passé dans la section ainsi délimitée.

Le problème de cette méthode est qu'elle permet d'obtenir une unique mesure derrière laquelle se cache une grande variabilité qui n'est pas évaluée.

4.2 Variabilité au sein d'une simulation

Le calcul de l'ensemble des pas de temps, c'est-à-dire la simulation privée de l'initialisation et de la finalisation est caractérisé par une certaine variabilité du temps passé à calculer de chaque pas de temps de la simulation même si, théoriquement, ce temps devrait être parfaitement constant. Une manière d'observer et possiblement de filtrer cette variabilité est de mesurer le temps passé à calculer chaque pas de temps de la simulation.

La figure II.5 montre le temps de calcul pour chaque pas de temps d'une simulation. Un certain nombre de pas de temps ont un temps d'exécution 2 à 10 fois plus grand qu'un pas de temps moyen. Les ralentissements sur ces pas de temps "lents" sont si importants que la moyenne de cette série temporelle est plus 7% élevée que la médiane. Cet effet augmente avec le nombre de cœur, comme illustré sur la figure III.3 où les performances calculées avec la médiane deviennent $\approx 75\%$ plus efficaces que les moyennes. Comme les mêmes instructions sont

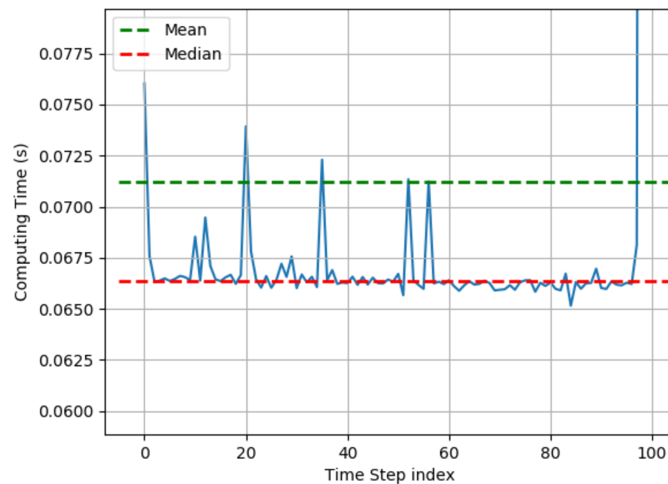


FIGURE II.5 – Temps de calcul (en secondes) par pas de temps pour une simulation TSUNAMI sur 10 nœuds AMD Epyc 7763. La ligne pointillée verte (resp. rouge) indique le temps de calcul moyen (resp. médian) d'un pas de temps.

exécutées à chaque pas de temps, ces ralentissements sont probablement la conséquence d'instabilités du super calculateur ou de préemptions des unités de calcul pour effectuer certaines tâches du système.

Ces instabilités arrivent de manière aléatoire et ne sont pas causées par le code NEMO, il est donc naturel de vouloir éliminer l'impact qu'elles ont sur l'exécution. Les instabilités se traduisent par des variations des performances, par exemple à l'échelle du pas de temps comme on l'observe sur la figure II.5. Une manière de limiter l'effet de ces instabilités aléatoires, sur l'estimation des performances du modèle, est donc de filtrer la variabilité au niveau du pas de temps. En prenant la moyenne des pas de temps, on retrouve la mesure du temps de la simulation privé de l'initialisation et de la finalisation divisé par le nombre de pas de temps. En revanche, un outil statistique aussi simple que la médiane permet de filtrer les pas de temps qui ont subi des instabilités et sont donc lents à l'exécution. On s'affranchit ainsi d'une partie aléatoire dans la mesure des performances.

L'utilisation de la médiane a toutefois pour conséquence de masquer une grande partie du comportement de l'exécution. La figure II.6 présente certaines exécutions avec des caractéristiques notables. Les temps d'exécution par pas de temps pour les simulations sur 1 nœud sont visiblement regroupés autour de deux valeurs. Pour le nœud AMD, au début de l'exécution le

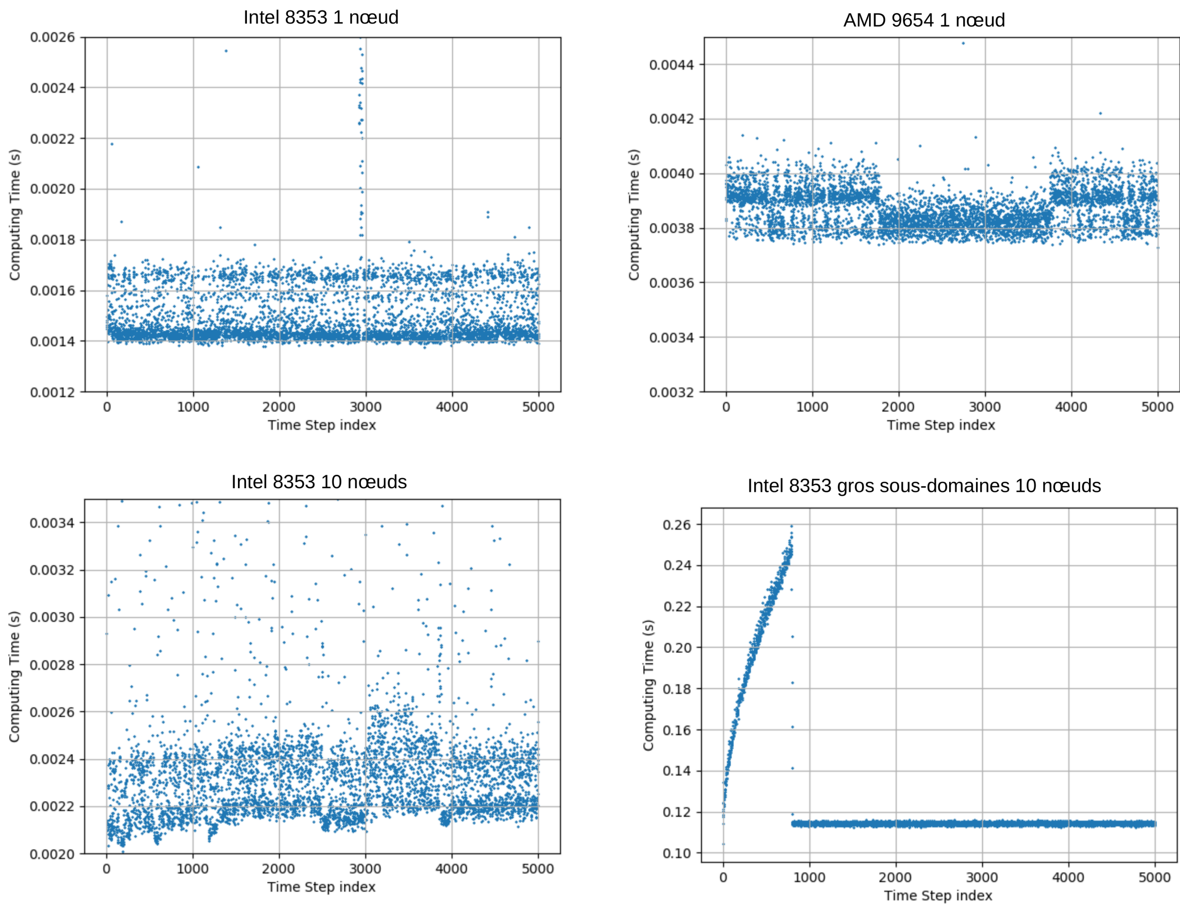


FIGURE II.6 – Temps de calcul (en secondes) par pas de temps pour un florilège d'exécution de la configuration TSUNAMI. Les sous-domaines ont tous une taille de 10 points par 10 points sauf pour l'exécution indiquée "gros sous-domaines" avec 100 sur 100 points.

temps de calcul d'un pas de temps est centré autour de la valeur haute, puis basse et à nouveau haute. Pour le nœud Intel, ce phénomène n'est pas cloisonné temporellement, les pas de temps plus lents ou plus rapides à l'exécution s'alternent à haute fréquence. Ce comportement semble être toujours présent sur 10 nœuds bien que plus bruité. On observe aussi des variations plus fortes, comme sur la simulation en bas à droite où le temps d'exécution du pas de temps augmente progressivement jusqu'à doubler, puis de redescendre brutalement. Ces phénomènes ne sont pas exceptionnels mais sont au contraire la règle.

Contrairement à la moyenne qui garde une trace de tous ces éléments, la médiane filtre toute cette complexité qui constitue la réalité d'une exécution. Une mesure des performances utilisant la médiane peut être pénalisante pour certaines versions du code développées pour améliorer les performances dans des situations de ralentissement, comme c'est le cas pour certains schémas

de communication développés au chapitre IV. En effet, il est possible que les pas de temps plus lents soient liés à des ralentissements, les atouts de versions du code peuvent donc être masqués par l'utilisation de la médiane.

Par ailleurs, l'appel à une routine de mesure de l'heure est peu coûteuse en temps de calcul et n'intervient que deux fois par pas de temps. Ces techniques d'évaluation des performances n'ont donc quasiment pas d'impact sur les performances elles-mêmes.

4.3 Variabilité entre simulations

On a vu que calculer soit la moyenne soit la médiane permet de masquer la variabilité au sein d'une simulation, mais il subsiste une variabilité entre différentes exécutions d'une simulation dans les mêmes conditions. Tout d'abord, deux exécutions d'un même code sur une même configuration et avec les mêmes paramètres d'entrées sur un matériel identique peuvent donner des performances largement différentes. Ce phénomène peut partiellement s'expliquer par certains facteurs, par exemple, des nœuds différents peuvent être utilisés par le supercalculateur pour ces deux exécutions. Ces nœuds sont potentiellement reliés de manières différentes par le réseau d'InterConnect et ainsi occasionner des temps de communication plus longs pour certaines exécutions. Le réseau d'InterConnect lui aussi peut être en cours d'utilisation par d'autres utilisateurs et donc présenter une rapidité variable.

Il convient de prendre certaines précautions pour garantir que les résultats de la mesure des performances soient utilisables. Dans le cadre d'une estimation de la pertinence d'une optimisation, plusieurs versions du code doivent être comparées. Ces versions doivent être exécutées dans un même job, c'est-à-dire dans le même environnement et directement à la suite l'une de l'autre, de manière à exécuter les différents tests dans des conditions d'occupation de la machine les plus proches possible, limitant ainsi une source de variabilité dans les mesures.

Même lorsque cette technique est utilisée, il est toujours possible d'avoir des variations de performances dues à des paramètres extérieurs au code lui-même (préemptions, charge de la machine...). Ce phénomène peut, par exemple, apparaître lorsqu'un même code est exécuté plusieurs fois en boucle de manière à obtenir plusieurs valeurs moyennes ou médianes.

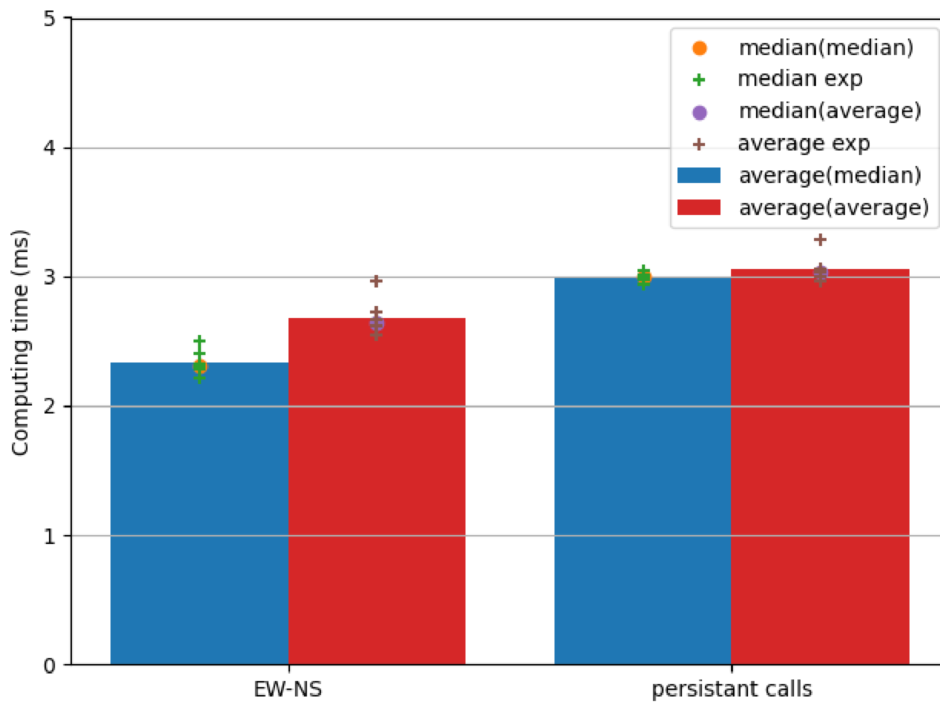


FIGURE II.7 – Temps de calcul (en secondes) par pas de temps pour différentes versions de NEMO. Les croix vertes (resp. rouge) sont les médianes (resp. moyenne) du temps de calcul des pas de temps de chaque exécution du code. Les ronds jaunes (resp. mauves) sont les valeurs médianes des médianes (resp. moyennes) sur une exécution. De même, les barres bleu (resp. rouge) sont les valeurs moyennes des médianes (resp. moyennes).

La figure II.7 illustre ce problème en comparant les performances obtenues sur deux versions du code exécutées 5 fois. Dans certains cas, les 5 répétitions de la simulation donnent des résultats très proches concernant la médiane ou même la moyenne (faible étalement des croix vertes et rouges). Les simulations *EW-NS* montrent une nette différence entre les valeurs moyennes et médianes (barres bleues/rouges) qui est caractéristique d'une forte variabilité de la durée d'un pas de temps au sein d'une même simulation. On constante aussi un étalement important des 5 valeurs de médiane et de moyenne ce qui montre que les résultats de chaque simulation diffèrent significativement entre eux et que l'utilisation de la médiane et de simulations réalisées dans le

même job n'a, dans ce cas, pas suffi à gommer les sources de variabilité externes au code. Dans ce cas, il semble plus prudent de faire à nouveau tourner la totalité des simulations si l'on veut quantifier précisément la différence de performance entre les deux versions de NEMO.

4.4 Méthodologie d'évaluation des performances

La méthodologie utilisée pour l'évaluation des performances vise à s'affranchir de la variabilité pour pouvoir comparer des versions du code et affirmer si une version réduit ou non le temps de calcul.

Une attention particulière doit être apporté dans le choix des conditions d'exécution. L'évaluation des performances doit par exemple se faire sans un profilage simultané qui modifierait les performances du code. En pratique, les instructions du profilage par défaut de NEMO au niveau des appels au schéma de communication sont désactivées. Les seules mesures de l'horloge restantes servent au calcul du temps d'exécution des pas de temps.

Au sein d'un même job les différentes versions de NEMO sont exécutées les unes après les autres (voir l'algorithme 1) avec un certain nombre de répétition N. N doit être suffisamment grand pour s'assurer que les performances des exécutions ne soient pas dues à la variabilité du supercalculateur. Prendre $N = 5$ permet d'ordinaire cela dans une certaine mesure.

Les différentes versions du code peuvent être concrètement des exécutables différents ou le même exécutable utilisé avec des fichiers de namelist différents. Les exécutions ont lieu en alternant les versions du code de manière à ce que les conditions soient les plus équivalentes possibles entre les versions. En effet, le supercalculateur peut être temporairement ralenti et serait sinon susceptible de diminuer uniquement les performances d'une seule des versions.

Algorithm 1 Exécution pour l'évaluation des performances

for $irep \leftarrow 1$ to N **do**
    for `Versions de NEMO` **do**
        `Exécution de la version de NEMO`
    end for
end for

Une fois l'exécution effectuée, la médiane du temps de calcul des pas de temps pour chaque exécution du code est calculée. Si pour une même version du code la médiane a une grande variabilité, alors l'ensemble des simulations doit être recommencé. Sinon, la médiane de la médiane des temps de calculs peut être calculée et utilisée pour comparer les versions du code entre elles.

Notons que cette méthode permet d'obtenir des résultats moins sensibles à la variabilité du temps d'exécution ce qui est primordial pour pouvoir comparer différentes versions du code. Cependant c'est au prix de ne pas prendre en compte les pas de temps les plus lents ce qui ne reflète donc pas précisément le temps de restitution de la simulation. Or c'est cette dernière métrique qui est réellement pertinente à optimiser, mais elle n'est malheureusement pas utilisable en pratique pour quantifier les performances du code et non la variabilité de la machine. De plus, cette méthode d'évaluation des performances peut limiter les performances particulièrement de certaines versions du code.

Les résultats entre deux exécutions peuvent très fortement changer, mais malgré toutes nos précautions, les rapports entre les performances des différentes versions du code montrent aussi des variations significatives. Il est possible de renouveler ce protocole d'évaluation des performances à répétition pour prendre en compte plusieurs états du supercalculateur et pour réduire l'incertitude liée à la variabilité, mais cela demande d'utiliser beaucoup de ressources de calcul. Par ailleurs, même s'il était possible d'évaluer les performances de manière robuste et fiable ce ne serait que dans une situation bien précise, notamment sur un seul supercalculateur avec une configuration et un nombre de nœuds donnés.

En fin de compte, aucune stratégie ne semble pouvoir quantifier correctement les performances du modèle. Même si la méthode que nous utilisons n'est pas complètement satisfaisante, elle nous apparaît comme la plus aboutie, du moins pour la comparaison de différentes versions du code.

5 Conclusion et perspectives

L'approche choisie pour l'optimisation de NEMO est exposée. L'environnement sur lequel les différentes étapes de l'optimisation seront effectuées est ensuite détaillé, en particulier les configurations BENCH_OCE et TSUNAMI sélectionnées et construites pour leur stabilité et leur simplicité d'utilisation. Les outils nécessaires aux étapes de profilage et à l'évaluation des performances sont présentés. Premièrement, les principaux outils de profilage utilisés par la suite sont examinés et un profilage spécifique à NEMO est développé. Deuxièmement, les enjeux de l'évaluation des performances sont expliqués ainsi que la méthodologie retenue.

Idéalement, une évaluation des performances réaliserait une estimation fiable et consistante du temps de simulation total, mais ce n'est pas possible. À la place, pour garantir une certaine fiabilité de la mesure on estime le temps d'exécution médian d'un pas de temps de la simulation.

Pour tenter de capturer aussi les performances liées à une certaine variabilité on pourrait estimer les capacités du code à limiter l'impact des instabilités du supercalculateur. Le travail sur le profilage effectué dans la section 3.3 peut être utilisé pour estimer ce dernier point. Dans le chapitre III à la section 2.3 il sera montré que des ralentissements peuvent avoir lieu sur un processus de la simulation. Un profilage adapté de ce phénomène permettrait aussi d'évaluer comment ce ralentissement est propagé selon les versions du code utilisées. Cette dernière évaluation complétée par le calcul du temps d'exécution du pas de temps médian se substituerait alors à une mesure du temps total de simulation.

À la suite du travail de ce chapitre sur le profilage, le profilage par défaut de NEMO a été complètement réécrit. Il sera intégré dans la future version de référence de NEMO.

CHAPITRE

III

Freins aux performances dans NEMO

1 Nombre de communications

Nous l'avons évoqué précédemment (voir figure II.4), les communications sont un point limitant les performances lorsqu'une simulation NEMO est exécutée sur un grand nombre de cœurs. Un moyen évident de réduire le temps passé dans les communications est de réduire le nombre de communications effectuées par le code. Pour que cela soit possible sans modification des résultats physiques et donc sans modification des calculs, seules les communications dont les résultats ne sont pas utilisés peuvent être supprimées. En effet, il est possible que le code contienne des communications devenues obsolètes, en raison de l'évolution du code, voire non essentielles, ajoutées par des développeurs jugeant peut-être que la communication ne porterait pas préjudice aux performances.

Alternativement, plusieurs communications peuvent être regroupées en une seule. Comme le montre Tintó et al. (2019), l'efficacité des communications MPI dans NEMO n'est pas limitée par le volume de données à transférer entre les processus, mais par le nombre de communications en lui-même. En effet, dans NEMO seul le bord des sous-domaines MPI est échangé ce qui représente peu de données, par contre, la procédure exécutée pour chaque communication a un coût assez élevé. Regrouper les communications qui ne peuvent pas être supprimées est donc une bonne stratégie pour améliorer les performances de NEMO. Notre objectif est donc de trouver les parties du code qui effectuent le plus de communications, puis de déterminer comment réduire ce nombre, soit en supprimant les communications inutiles, soit en regroupant les communications indispensables. Notons que suite au développement de Tintó et al. (2019), une interface générique Fortran a été ajoutée à NEMO qui rend le regroupement de communications multiples extrêmement facile. Elle est présentée dans la section 2.1 dans le chapitre IV. Précédemment, les communications actualisant les valeurs des halos ne pouvaient être effectuées que sur un seul tableau à la fois. Ces développements ont permis de réaliser des communications sur plusieurs tableaux à la fois et ainsi de regrouper les instructions de communication, bien que le volume de données échangé reste le même.

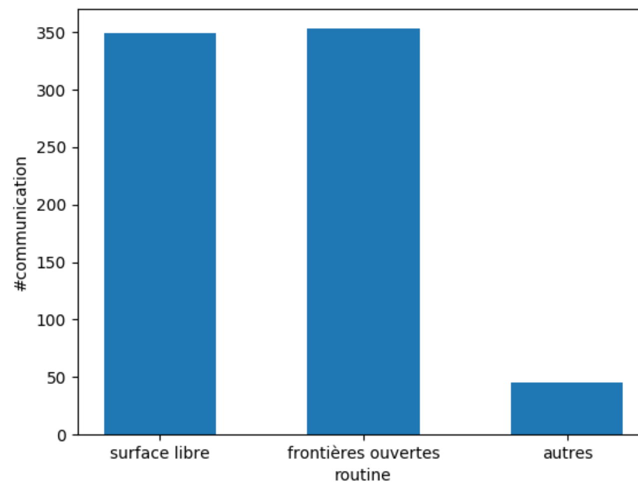


FIGURE III.1 – Nombre de communications par pas de temps pour différentes routines dans une simulation avec frontières ouvertes.

Dans la configuration que nous testons (BENCH sans périodicité, frontières ouvertes et sans glace de mer), près de 90 % des communications se font dans deux routines : le calcul du gradient de pression de surface (44 %) et les conditions aux frontières ouvertes (45 %) avec environ 300 appels à chaque pas de temps. Les optimisations suivantes se concentreront donc sur ces deux parties du code. Notez que dans les deux cas, les communications impliquées transfèrent un volume très limité de données (d'un scalaire unique à un tableau à une dimension), ce qui justifie encore plus la stratégie proposée par Tintó et al. (2019). Le nombre exact de communications par pas de temps peut changer suivant les paramètres fournis en namelist. Le reste des routines ne comporte qu'un faible nombre de communications.

Nous avons réalisé deux optimisations utilisant ces techniques pour réduire le nombre d'appels aux routines de communications. La première qui élimine le besoin de communication dans le traitement des frontières ouvertes et la seconde qui réduit de $\frac{1}{3}$ les communications dans le calcul du gradient de pression de surface en les regroupant quand cela est possible. Ces optimisations sont décrites en détail dans l'annexe A ainsi que dans la publication Irrmann et al. (2022). Le nombre de communications restant par pas de temps est indiqué figure III.2.

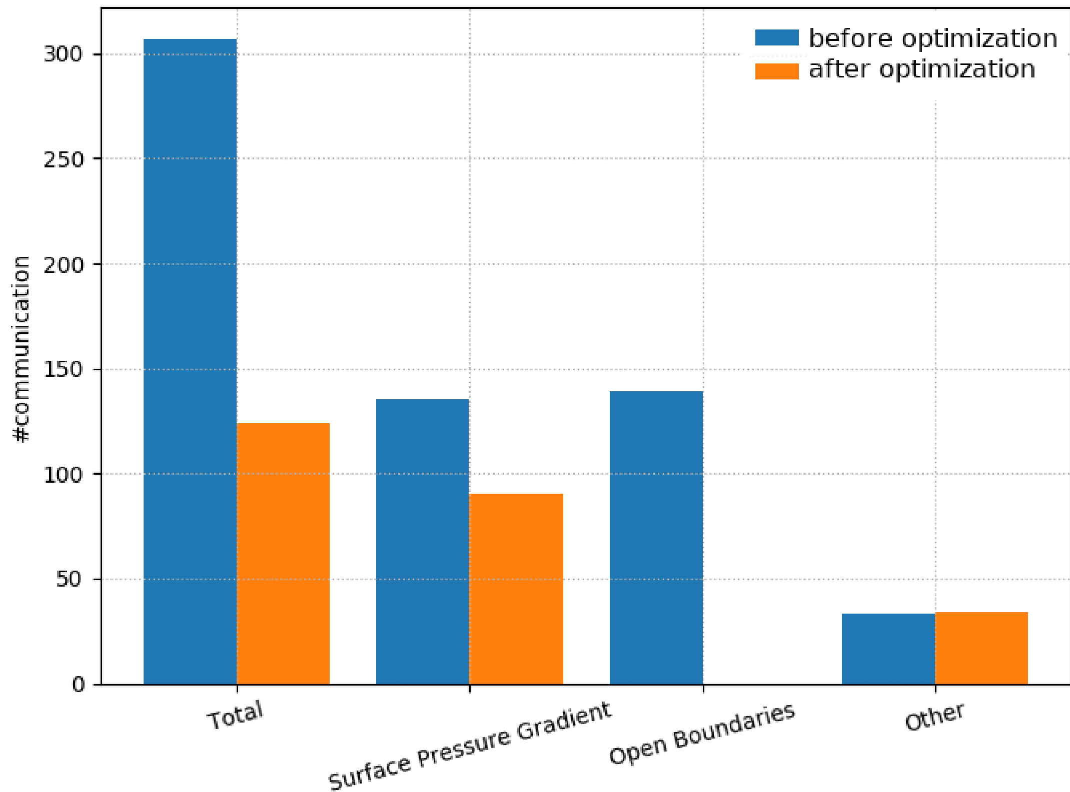


FIGURE III.2 – Comparaison entre le nombre de communications nécessaires à chaque pas de temps dans NEMO avant (en bleu) et après (en orange) les optimisations pour une configuration avec des frontières ouvertes rectilignes et sans glace.

Dans ce cas test, qui est représentatif de la très grande majorité des utilisations, les frontières ouvertes sont droites et situées le long du bord du domaine. Les communications liées aux frontières ouvertes ont donc été complètement supprimées et les communications liées au gradient de pression de surface sont réduites d'un tiers. Par conséquent, dans cette configuration, le nombre total de communications par pas de temps a été réduit d'environ 60 % (Figure III.2) c'est à dire d'environ 300 communications par pas de temps à 125.

La figure III.3 montre l'amélioration, apportée par nos optimisations, de la scalabilité de NEMO, c'est-à-dire sa capacité à conserver ses performances lorsqu'un nombre de plus en plus grand d'unités de calcul est utilisé dans son exécution. Dans cette figure, la performance du modèle est quantifiée par le nombre d'années qui peuvent être simulées pendant une période de 24 heures (années simulées par jour ou Simulated Year Per Day, SYPD). Pour un petit nombre de cœurs, les optimisations n'ont pas d'effet notable car le temps consacré aux communications est très faible. Cependant, à mesure que le nombre de cœurs augmente, chaque sous-domaine

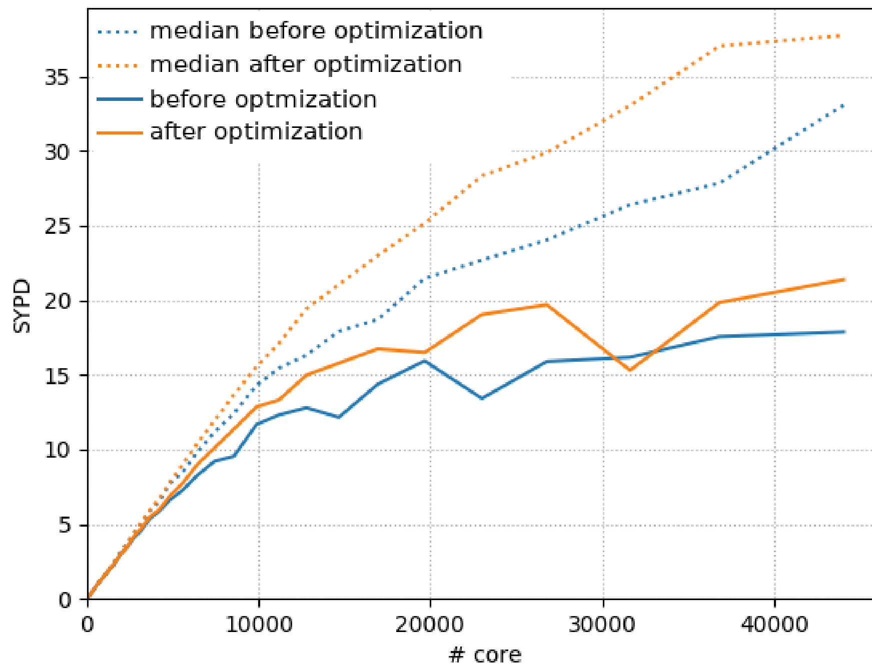


FIGURE III.3 – Graphique du passage à l'échelle des performances en scalabilité forte : années simulées par jour (SYPD) en fonction du nombre de cœurs utilisés dans une simulation de l'Atlantique Ouest de taille fixe.

MPI devient plus petit, la charge de calcul diminue et la charge de communication devient prédominante. Ici, les optimisations apportent de nettes améliorations : le nombre d'années simulées par jour est plus élevé dans la version optimisée du code (les courbes orange sont au-dessus des courbes bleues correspondantes). Les courbes de scalabilité construites en considérant la moyenne de la totalité des pas de temps (lignes continues) sont néanmoins assez bruitées et l'amélioration n'est pas aussi bonne qu'attendu : 20% au mieux avec même une valeur négative autour de $30,000$ cores.

En filtrant les pas de temps aberrants à l'aide de la médiane, on obtient des résultats bien meilleurs et plus robustes. Les courbes de scalabilité qui en résultent (lignes en pointillés) sont moins bruitées. De plus, si l'on excepte le dernier point, la différence entre les deux lignes en pointillés croît régulièrement au fur et à mesure que l'on utilise plus de cœurs. L'impact de nos optimisations est donc plus important pour un plus grand nombre de cœurs. Le nombre de SYPD est, par exemple, augmenté de 35% pour $37,600$ cœurs lors de l'utilisation de la version optimisée.

On peut également noter que la version optimisée exécutée sur $23,000$ cœurs simule le même nombre de SYPD que l'ancienne version sur $37,600$ cœurs, ce qui représente une réduction de l'utilisation des ressources de près de 40%.

Les différences entre les résultats utilisant la médiane et la moyenne soulignent l'impact significatif qu'ont les instabilités observées sur la figure II.5 sur la durée de calcul de chaque pas de temps qui, en théorie, devrait être parfaitement constante.

On verra dans la section 2.3 que les cœurs peuvent présenter des ralentissements. Si on suppose que chaque cœur a des chances similaires de souffrir d'instabilités ou d'entrer en préemption, les cœurs lents sont plus fréquents lorsque le nombre de cœurs est élevé. Comme les communications tendent à synchroniser les cœurs, un seul cœur lent ralentit l'ensemble de l'exécution. La médiane élimine les pas de temps au cours desquels la préemption ou de grandes instabilités se produisent. Elle indique les performances que nous pourrions obtenir sur une machine "parfaite" qui ne présenterait aucune "anomalie" pendant l'exécution du code. Une telle machine n'existe malheureusement pas. La tendance à l'architecture de nouvelles machines de plus en plus complexes et hétérogènes suggère que les "anomalies" de performance pendant la résolution du modèle peuvent se produire de plus en plus souvent et devenir une caractéristique courante. Nos résultats mettent en évidence cette nouvelle contrainte qui limite déjà une partie significative de nos gains d'optimisation (de 40% à 20%). Ce phénomène est étudié dans la suite de cette partie.

2 Ralentissements de l'exécution

Les instabilités dans les simulations sont un point limitant les performances des optimisations, mais aussi les performances de n'importe quelle simulation.

Suivant les détails de l'environnement informatique dans lequel la simulation est exécutée, les pas de temps les plus lents de la simulation peuvent être de 10 à 1000 fois plus lents à l'exécution que le pas de temps médian.

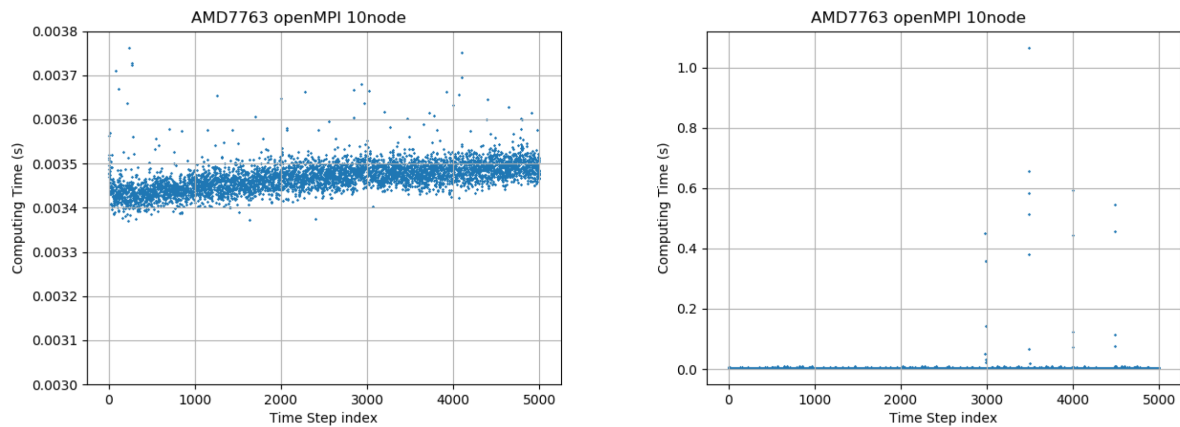


FIGURE III.4 – Temps d'exécution par pas de temps, les deux figures correspondent à la même exécution de la simulation avec une échelle de temps d'exécution différente, la figure de gauche est centrée sur le temps d'exécution du pas de temps médian.

2.1 Ralentissements extrêmes sur un pas de temps

Il arrive sur certaines simulations qu'un ou plusieurs pas de temps soient ralentis au point d'être 1000 fois plus lents, la figure III.4 donne un exemple d'une telle exécution. La quasi-totalité des temps d'exécutions sont proches de la médiane sauf un faible nombre dont le temps d'exécution est environ 100 fois plus lent que le pas de temps médian et jusqu'à 300 fois plus lent pour le pas de temps le plus lent à l'exécution.

Du fait de la rareté de ces pas de temps ralentis on pourrait penser qu'ils ne sont responsables que d'une faible partie du temps de simulation total. En réalité les temps d'exécution de ces pas de temps sont si élevés que pour la simulation représentée figure III.4 les 1% pas de temps les plus lents sont à l'origine de 25% du temps de simulation. Leur importance est donc considérable.

Précisons ici que ce phénomène de ralentissement extrême n'est pas le résultat de l'état à un instant précis du supercalculateur mais persiste tant que l'environnement et les paramètres de la simulation sont identiques. Les facteurs qui semblent importants pour que de tels ralentissements se produisent sont la taille des sous-domaines (suffisamment petits, environ 10 par 10) et le nombre de nœuds (suffisamment grand).

Une analyse des cartes réseau des nœuds de la simulation laisse penser que l'origine de ce phénomène est la congestion des communications sur le réseau d'InterConnect. Lorsque les sous-domaines sont petits, les communications s'effectuent à très haute fréquence car les phases de calculs sont plus courtes.

Les données ne peuvent pas être réceptionnées par un nœud ou un *switch*, c'est-à-dire un embranchement de l'InterConnect, si leur espace mémoire dédié aux réceptions est saturé. Pour que la transmission soit complétée quel que soit l'état de la mémoire sur le supercalculateur, soit la transmission est mise en attente tant que l'espace mémoire pour la réception n'est pas disponible, soit les données sont réenvoyées jusqu'à réception (Gran et al., 2012). Dans les deux cas, des opérations supplémentaires sont nécessaires pour mener à bien la transmission et l'espace mémoire est occupé sur le matériel effectuant l'envoi tant que la réception n'est pas réalisable. La saturation de la mémoire dédiée aux réceptions se propage ainsi de proche en proche et forme un phénomène de congestion. Gran et al. (2012) assure que cela cause une baisse extrême des performances.

C'est ce phénomène qui serait à l'origine des pas de temps 100 voire 1000 fois plus lents que la médiane.

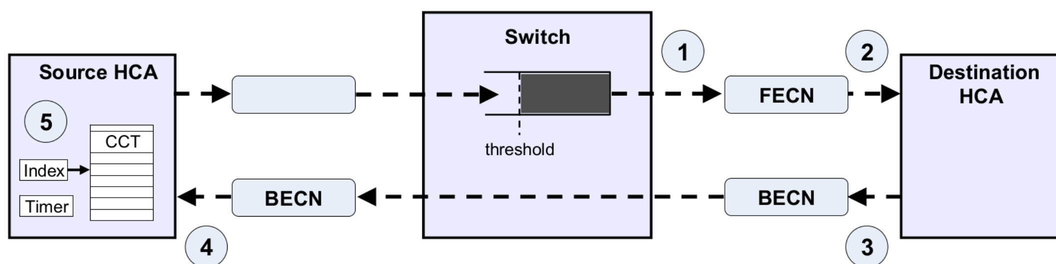


FIGURE III.5 – Architecture de la gestion de la congestion sur un réseau d'InterConnect.

Le réseau d'InterConnect utilise sa propre méthode visant à détecter et limiter la congestion voire éliminer le phénomène entièrement. La figure III.5, issue de Gusat et al. (2005), présente un exemple de contrôle de la congestion sur un switch situé sur le chemin d'un transfert d'une source vers une destination et expose les trois temps de la technique utilisée. Premièrement, la congestion est détectée par le dépassement d'un seuil (*threshold* sur la figure), le destinataire est ensuite informé de la congestion par un message (FECN) et enfin la source par un autre message

(BECN) qui réduit son flux de message envoyés. La réduction résultante a une durée limitée dans le temps. Le contrôle de la congestion prend donc la forme d'une boucle de rétroaction ajustée par une série de paramètres : le seuil de détection, la réduction du flux et la durée de cette réduction (Crupnicoff et al., s. d.).

Ces caractéristiques correspondent aux particularités du phénomène de ralentissement que l'on observe. Une fois la congestion maîtrisée, la durée limitée de la rétroaction explique le fait que les performances reviennent à la normale au pas de temps suivant. De plus, le caractère visiblement aléatoire de l'occurrence des ralentissements peut s'expliquer par le fait que l'état des *switchs* de la machine n'est pas contrôlable par un utilisateur, les *switchs* étant sollicités par les codes d'une multitude d'utilisateurs.

Les paramètres par défaut de ce contrôle de la congestion ne sont visiblement pas adaptés à notre utilisation de l'InterConnect. Pour une efficacité optimale, ces paramètres peuvent être modifiés (Gusat et al., 2005) pour prendre en compte les spécificités du code, le nombre de nœuds utilisés et leur disposition dans le réseau d'InterConnect. Ce travail demande une connaissance pointue de la technologie d'interconnexion ainsi que les droits administrateurs des supercalculateurs utilisés, or, ces deux éléments me font défaut. De plus, dans l'optique d'une optimisation pérenne, il n'est pas envisageable que d'autres utilisateurs de NEMO modifient ces paramètres pour chaque exécution. Une autre solution a donc dû être élaborée.

Cette solution repose sur l'idée de limiter en amont, au niveau des nœuds, la charge mise sur le réseau d'InterConnect pour la garder à un niveau que le contrôle de la congestion parvient à maîtriser. Pour les librairies MPI qui reposent sur *UCX* (Shamis et al., 2015), des variables d'environnement sont disponibles pour donner des directives à la carte réseau concernant la manière dont les communications sont gérées. D'après la documentation en ligne NVIDIA (Unified Communication - X Framework Library, s. d.) la variable d'environnement `UCX_RC_TX_NUM_GET_BYTES` limite la quantité de données transférée simultanément et `UCX_RC_MAX_GET_ZCOPY` limite la taille des messages transférés. L'utilisation simultanée de ces deux paramètres permet de limiter le nombre de transferts gérés à un même instant par la carte

réseau. La congestion est alors régulée à sa source. Les lignes de code suivantes exécutées avant le lancement de NEMO permettent de limiter la congestion durant la simulation :

```
export UCX_RC_TX_NUM_GET_BYTES=10240
export UCX_RC_MAX_GET_ZCOPY=64
```

Les ralentissements extrêmes sur certains pas de temps sont éliminés par cette nouvelle gestion des transferts et la réduction des performances de certaines simulations sont rétablies au niveau habituel. On en déduit de plus que les ralentissements extrêmes au niveau du pas de temps étaient une conséquence de la congestion sur l'InterConnect.

2.2 Légers ralentissements sur un pas de temps

Même lorsque les pas de temps extrêmement lents sont éliminés par les variables d'environnement, il reste des pas de temps significativement plus lents, au maximum 10 fois plus lents. La figure III.4 donne un exemple d'une telle exécution.

Nous avons essayé une variété d'environnements et de conditions d'exécution sans que la présence de ces pas de temps s'en trouve significativement altérée. Changer de librairie MPI, entre openMPI et intelMPI, n'a pas d'influence et changer naïvement la version de la librairie MPI sans modifier aussi l'ensemble de l'environnement peut être fortement nuisible aux performances.

De même, une grande variété de matériel, nœud, InterConnect etc, a été testé sans que le problème soit résolu. Nous avons analysé le profil du temps d'exécution des pas de temps sur différents types de nœuds ainsi que sur différents supercalculateurs. La durée exacte du temps d'exécution pour chaque pas de temps varie en effet suivant le matériel utilisé, mais des pas de temps significativement plus lents se détachent toujours du pas de temps médian.

2.3 Ralentissement sur un cœur

La figure III.6 montre l'ensemble des processus MPI d'une simulation, à chaque processus MPI correspond un petit rectangle de la figure organisé selon le découpage en sous-domaines MPI. Rappelons que les sous-domaines ne contenant que des points terre ne sont pas simulés et

FIGURE III.6 – Temps de calcul en secondes. Chaque petit rectangle non blanc représente un sous-domaine MPI de la simulation. Le temps de calcul d'un tronçon délimité par la fin d'un appel à la routine de communication d'une part et le début de l'appel suivant d'autre part est affiché. Ce tronçon est sélectionné au cours de l'exécution d'une simulation de l'Atlantique Ouest utilisant 3851 cœurs.

les rectangles correspondants sont coloriés en blanc sur la figure. Comme à chaque processus MPI correspond un cœur de calcul, chaque rectangle représente le temps de calcul d'une portion de code choisie par un cœur. La figure III.6 montre une section donnée de l'exécution de la simulation durant laquelle un unique cœur présente un temps de calcul anormalement long. Les autres cœurs affichent en comparaison des temps de calcul beaucoup plus faibles en cohérence avec la faible quantité de calcul de ce tronçon de code. Tout comme pour les pas de temps plus lents, les librairies utilisées ainsi que le type de nœud ou le supercalculateur ne semblent pas avoir d'impact sur l'occurrence du ralentissement du cœur. Par ailleurs, le cœur causant le ralentissement n'étant pas toujours le même au cours de la simulation, le matériel ne semble pas devoir être mis en cause.

L'une des potentielles raisons des ralentissements au niveau d'un cœur est une interruption de l'exécution de NEMO par le système d'exploitation pour mener d'autre tâches qui doivent être effectuées. Ce phénomène est appelé préemption. À notre connaissance il n'existe pas de manière d'empêcher le système d'exploitation de préempter l'exécution de NEMO et s'il en existait un, il ne serait pas certain que l'utiliser soit une bonne idée, les tâches effectuées par le système d'exploitation devant certainement être réalisées pour de bonnes raisons.

Une potentielle solution pour éliminer les ralentissements au niveau des cœurs, que nous n'avons pas encore testée, serait de faire du dépeuplement. C'est-à-dire de réserver pour le calcul de la simulation NEMO plus de cœurs qu'il n'y aura de tâche MPI, par exemple un cœur de

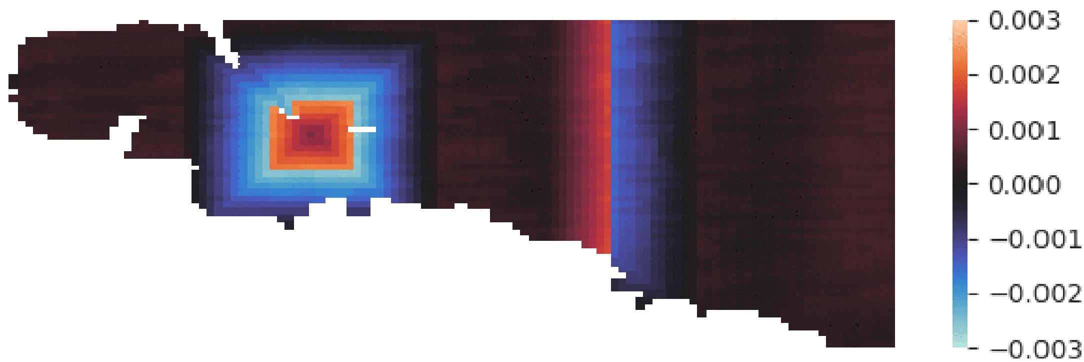


FIGURE III.7 – Retard par rapport à la progression moyenne en secondes. Les valeurs positives indiquent les sous-domaines MPI qui arrivent au début de la routine de communication plus tard que la moyenne sur les sous-domaines.

plus par nœud. Cela se traduit par certains cœurs qui ne sont pas utilisés pour la simulation et qui sont donc inactifs, par exemple le cœur 0 de chaque nœud. L'idée est que ces cœurs seraient alors disponibles pour effectuer des tâches pour le système d'exploitation, ce qui permettrait de supprimer l'impact sur le déroulement de la simulation.

Ce ralentissement a lieu sur un seul cœur, mais impacte l'ensemble des cœurs de la simulation. Les communications dans NEMO mettent en relation chaque processus MPI exclusivement avec ses voisins immédiats et propagent le ralentissement d'un cœur à ses voisins. Comme il n'y a pas de communications globales la propagation n'est jamais étendue en une seule étape à l'ensemble des cœurs de la simulation, mais se fait uniquement de proche en proche. La figure III.7 montre cette propagation. Elle correspond à la même simulation que la figure III.6, 5 communications après le moment où le ralentissement est représenté. Cette carte est rendue possible par le travail réalisé dans la section 3.3.

Étant donné que le ralentissement du cœur est propagé par le schéma de communication, ce schéma peut être modifié pour limiter cette propagation et mitiger l'impact du ralentissement sur le temps de calcul du pas de temps et en définitive réduire le temps de restitution des simulations. Des pistes de solution sont développées dans le chapitre IV.

Une variété de ralentissements a été mesurée sur les simulations NEMO au niveau du pas de temps avec différentes amplitudes ainsi qu'à l'échelle des cœurs. Ces phénomènes à différentes échelles sont potentiellement liés. Le ralentissement sur les cœurs est responsable, partiellement

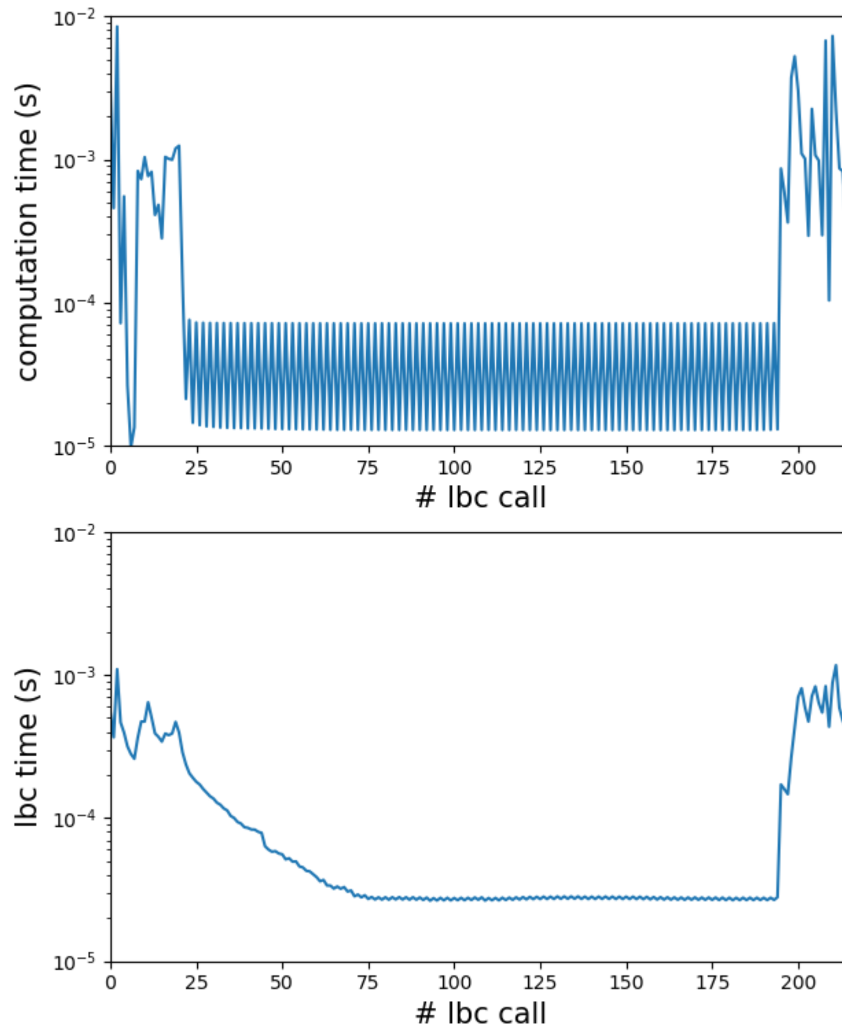


FIGURE III.8 – Temps d'exécution en secondes pour différentes sections du code au cours d'un pas de temps de la simulation. En haut le temps passé entre 2 appels à la routine de communication et en bas le temps passé dans la routine de communication. Les valeurs sont moyennées sur l'ensemble des cœurs.

ou totalement, du ralentissement au niveau du pas de temps. Cette éventuelle causalité n'a pas encore été étudiée. Si ce n'est pas le cas, l'origine du ralentissement sur les pas de temps reste à élucider.

3 Spécificité du calcul du gradient de pression de surface

On a vu sur la figure II.2 que la routine responsable d'une part importante du temps d'exécution de la simulation du fait de ses appels à la routine de communication est la routine du calcul du gradient de pression de surface.

La figure III.8 montre la structure d'un pas de temps de NEMO avec le temps d'exécution des phases de calculs et de communications. Le graphique du temps de calcul présente deux parties.

La partie située au milieu du pas de temps (phases de calcul entre les communications 22 à 190) correspond au calcul du terme de gradient de pression de surface à l'aide d'une boucle sur un sous-pas de temps. Comme il y a deux communications par sous-pas de temps (suite à nos optimisations à la section 1) précédées de deux phases de calcul différentes, cette partie présente une structure en dents de scie. Elle se caractérise par des séquences de calcul courtes et des communications à haute fréquence sur des quantités limitées de données.

L'autre partie, au début et à la fin du pas de temps, correspond au reste des calculs de NEMO. Elle est caractérisée par des séquences de calcul plus longues entre chaque phase de communication.

Le profil de communication comporte deux parties correspondantes : à l'extérieur des calculs du gradient de pression de surface, le temps de communication est assez élevé et à l'intérieur, il est plus faible, avec une phase de transition entre les deux. Deux facteurs expliquent la différence de temps de communication. Premièrement, plus le volume de données échangées est important, plus la communication est longue car le débit est limité dans le transfert et parce que les étapes de copies des donnés croissent en conséquence. Le volume de données échangées dans le calcul du terme de gradient de pression de surface est plus petit que le volume échangé dans la plupart des autres communications, les communications sont donc plus courtes. Deuxièmement, l'asynchronisme entre les cœurs ralentit les communications. Les cœurs peuvent se désynchroniser en raison d'une préemption ou simplement parce qu'ils ont une vitesse de calcul légèrement différente. Les cœurs se désynchronisent d'autant plus que la séquence de calcul est longue, et la probabilité qu'un cœur soit préempté est aussi plus importante. Il est probable que ce facteur soit dominant.

La transition entre les deux parties est également symptomatique des effets de l'asynchronisme. L'asynchronisme entre les cœurs reste élevé lorsqu'on entre dans la partie de NEMO consacrée à la communication à haute fréquence, mais il diminue progressivement jusqu'à ce que les cœurs soient aussi synchronisés que possible. Un plateau est alors atteint.

Le calcul du gradient de pression de surface a des caractéristiques différentes du reste du code qui ont des impacts sur les performances des communications. Les problématiques de l'optimisation des communications sont donc légèrement différentes suivant que ce soit cette partie qui est exécutée ou non. Les configurations TSUNAMI et BENCH_OCE permettent donc d'évaluer les schémas de communications au sein du calcul du gradient de pression (pour TSUNAMI) ou pour l'ensemble du code (pour BENCH_OCE).

4 Conclusion et perspectives

Les principaux freins aux performances de NEMO dans le cadre que l'on s'est fixé dans le chapitre II ont été présentés. Le nombre d'appel aux routines de communications est mis en évidence comme facteur limitant des performances, bien que cet élément ne soit pas aussi important que ce que je pensais initialement. Cet écart est dû aux pas de temps plus lents à l'exécution de la simulation. Ces particularités de l'exécution ont différentes causes qui ne peuvent probablement pas toutes être supprimées, par exemple, le ralentissement au niveau des cœurs. Une stratégie pour aborder ce problème n'est pas de le supprimer, mais de limiter son impact en modifiant le schéma de communication. Ce schéma devra être adapté aux particularités de la partie du code dans laquelle il est utilisé, notamment au calcul du gradient de pression.

Notons que les spécificités du calcul du gradient de pression ne sont pas uniques à cette partie mais se retrouvent aussi dans les routines de calcul de la rhéologie de la glace. En effet, ces routines sont aussi caractérisées par un grand nombre de communications sur un faible volume de données entrecoupées de courts calculs. De plus, la stratégie de réduction du nombre de communications mise en œuvre dans l'annexe A à la section 2 pourrait aussi être appliquée aux calculs de la glace.

Un certain travail reste à effectuer pour comprendre pleinement les ralentissements, par exemple l'importance qu'ont les ralentissements au niveau des cœurs sur les ralentissements des pas de temps.

CHAPITRE

IV

Optimisation des schémas de communication

1 Le fonctionnement de la librairie MPI

Les communications entre sous-domaines sont effectuées par la librairie *Message Passing Interface, (MPI)*. Une optimisation des communications dans NEMO s'appuie nécessairement sur une compréhension fine de cette librairie.

Un standard de communication unique a été développé (Dongarra et al., 1994) pour que chaque développeur puisse utiliser les mêmes directives au niveau du code pour initier des communications sur n'importe quel supercalculateur, quel que soit le matériel reliant les CPUs ou la pile logiciel accompagnant ce matériel. Ce standard assure une portabilité des codes qui n'ont ainsi pas besoin d'être adaptés spécifiquement à chaque supercalculateur. Nommé *Message Passing Interface, (MPI)*, ce standard est implémenté dans plusieurs librairies, par exemple *intelMPI* ou *openMPI*, qui mettent à disposition une grande variété de fonctions permettant de réaliser des communications de différentes manières.

1.0.1 Communications bloquantes et non-bloquantes

MPI propose plusieurs modes de communications, on présente ici les modes des communications point à point, c'est-à-dire des communications n'impliquant que deux processus à la fois au contraire des communications globales qui impliquent un ensemble de processus, souvent tous les processus de la simulation.

Deux grands modes de communications MPI points à points existent : les communications bloquantes et non-bloquantes. Ces modes font références au protocole d'envoi et de réception de messages. Lorsqu'un processus effectue des instructions d'envoi bloquantes, il reste dans l'instruction jusqu'à ce qu'elle soit complétée, c'est-à-dire jusqu'à ce que le processus ciblé exécute l'instruction complémentaire. C'est le cas des instructions sur la figure IV.1, le processus indiqué *Proc 1* reste dans l'instruction de réception tant que le message venant du processus

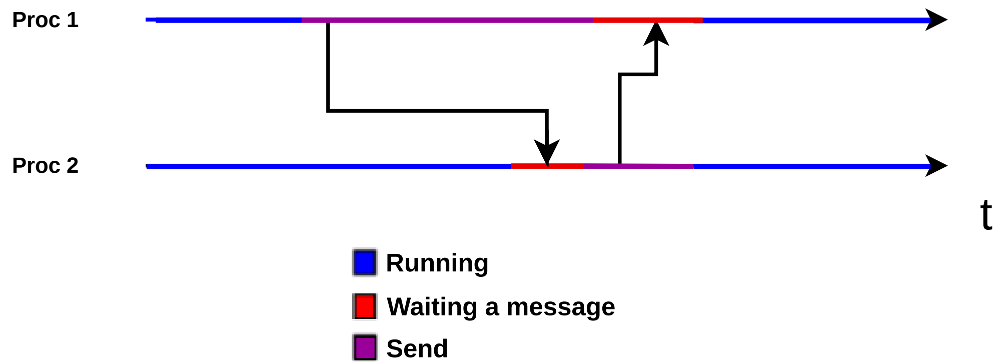


FIGURE IV.1 – Schéma d'une communication bloquante. Chaque ligne horizontale est un axe temporel représentant l'état d'un processus, en bleu s'il effectue des calculs, rouge s'il attend ou reçoit un message MPI (*MPI_Recv*) et en violet les instructions d'envoi (*MPI_Send*). Les flèches représentent la transmission de données d'un processus à un autre.

Proc 2 n'est pas réceptionné et à fortiori que le *Proc 2* n'a pas effectué son instruction d'envoi. Les communications bloquantes doivent donc impérativement s'effectuer séquentiellement (l'une à la suite de l'autre), chaque envoi/réception devant être en phase avec une réception/un envoi.

Sur la figure IV.1 le *Proc 2* termine ses instructions de calcul avant le *Proc 1*, cela peut être dû à un mauvais équilibre de charge, mais cela n'est pas censé arriver sur nos configurations TSUNAMI et BENCH_OCE. En revanche, le phénomène de ralentissement sur un cœur (voir chapitre III section 2.3) peut être à l'origine de cette situation.

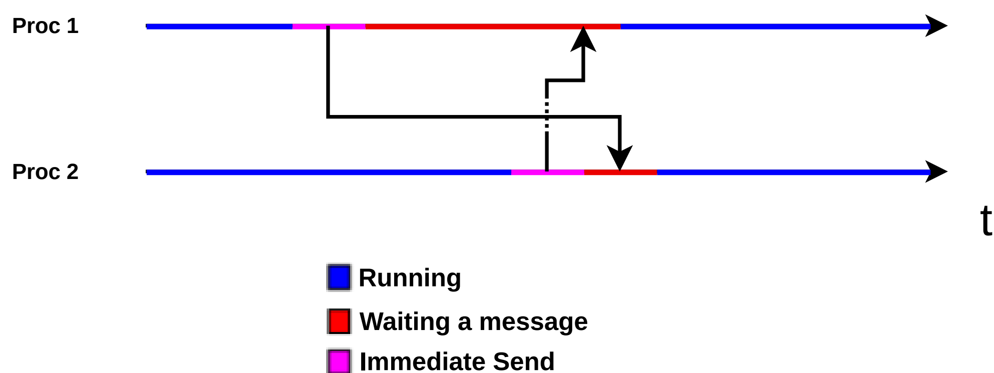


FIGURE IV.2 – Schéma d'une communication non bloquante. Chaque ligne horizontale est un axe temporel représentant l'état d'un processus, en bleu s'il effectue des calculs, rouge s'il reçoit un message MPI (*MPI_Recv*) et en rose les instructions d'envoi (*MPI_Isend*). Les flèches représentent la transmission de données d'un processus à un autre.

Les communications non-bloquantes fonctionnent différemment : l'instruction d'envoi se termine dès lors que le message est envoyé sans qu'il ait besoin d'être réceptionné. De même, une instruction de réception non-bloquante ne fait que préparer une réception et quitte l'instruction sans attendre que les données soient disponibles par le processus. C'est le cas des instructions d'envoi sur la figure IV.3. Ce type de communication requière, dans presque tous les cas (voir section 2.8), l'utilisation d'instructions bloquantes (par exemple *MPI_Wait*). Elles permettent de s'assurer que les données ont bel et bien été envoyées et qu'elles peuvent donc être modifiées ou désallouées, mais aussi pour s'assurer que les données ont été reçues et qu'elles peuvent donc être utilisées, par exemple dans un calcul.

Les instructions MPI non bloquantes présentent l'avantage de permettre au processus d'effectuer d'autres opérations sans rester bloqué. On peut typiquement lancer une série d'envoi/réception en parallèle sans avoir à se soucier que chaque séquence d'envoi/réception soit parfaitement séquencé.

1.1 Recouvrement du temps de communication

Plusieurs facteurs sont centraux pour caractériser les types de schéma de communications et expliquer leurs performances.

Tout d'abord, il est possible de tirer avantage de l'utilisation des instructions de communications non-bloquantes pour réaliser du recouvrement. Le recouvrement pour un schéma de communication désigne la capacité du schéma à réaliser des opérations en attendant que d'autres opérations soient disponibles à l'exécution. La figure IV.3 montre un exemple de communication avec recouvrement. Des calculs qui seraient normalement exécutés après la fin du schéma de communication (comme sur la figure IV.2) le sont ici avant l'instruction bloquante de réception. Cela permet de prendre de l'avance sur les calculs qui suivront tout en permettant à des processus potentiellement en retard de rattraper ce retard. En comparant les schémas des figures IV.2 et IV.3, on remarque que le schéma avec recouvrement passe moins de temps dans les communications. On a dans ce cas un recouvrement des communications par le temps de calcul.

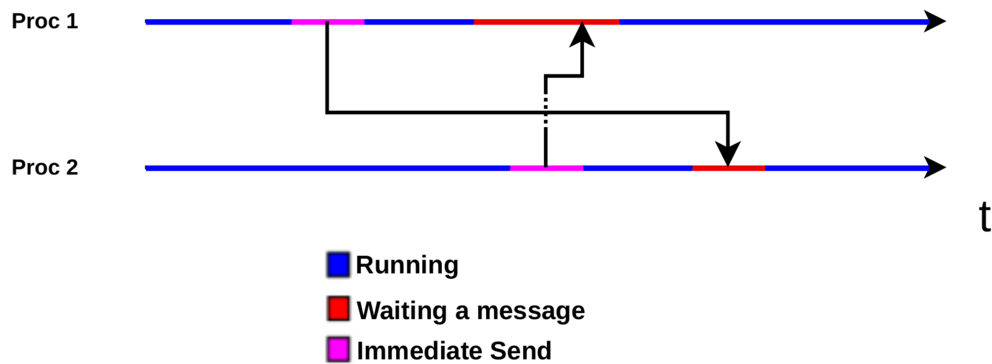


FIGURE IV.3 – Schéma d'une communication avec recouvrement. Chaque ligne horizontale est un axe temporel représentant l'état d'un processus, en bleu s'il effectue des calculs, rouge s'il reçoit un message MPI (*MPI_Recv*) et en rose les instructions d'envoi (*MPI_Isend*). Les flèches représentent la transmission de données d'un processus à un autre.

Les instructions bloquantes font obstacle aux possibilités de recouvrement du schéma. Puisque le processus reste en attente dans une instruction bloquante il n'y a pas de possibilité de recouvrement pour cette instruction. Au contraire, lorsque des instructions non bloquantes sont utilisées, on offre la possibilité d'avoir du recouvrement du temps de communication par toutes les instructions placées avant une phase de synchronisation bloquante (par exemple *MPI_Wait*).

Notons que ce phénomène de recouvrement n'a lieu que parce que les processus engagés dans une communication ne sont pas tous aussi avancés dans l'exécution et que l'un doit attendre l'autre. Si tous les processus étaient synchronisés leur temps de communication serait limité par la durée des opérations de la librairie MPI et du transfert dans le réseau d'InterConnect. On sait que cette vue idéalisée de l'exécution d'un code n'est pas conforme à la réalité.

Premièrement, il existe souvent un déséquilibre de charge entre les processus de la simulation, par exemple parce que la taille des sous-domaines n'est pas rigoureusement identique. Ce déséquilibre cause un retard des processus ayant la plus importante charge de travail. Cependant, ce n'est pas dans ce cas que le recouvrement améliorera les performances. En effet, si les mêmes processus sont toujours les plus lents, alors ce sont eux qui limiteront le temps de restitution de la simulation. Le fait que les autres processus effectuent des opérations pendant leur temps d'attente est sans importance.

Ensuite et surtout, on a vu dans le chapitre III que les processus pouvaient être sujet à des ralentissements se manifestant de manière aléatoire en apparence et concernent des processus potentiellement différents à chaque occurrence. Si un processus subit temporairement un ralentissement, le recouvrement permettra que le retard engendré se propage de manière atténuée, les prochains processus qui subiront des ralentissements seront alors plus avancés et cela limitera l'ampleur du ralentissement qui sera généré. Dans cette nouvelle perspective, le recouvrement permet effectivement d'améliorer les performances des codes.

1.2 Surcoût des instructions

Un autre facteur à prendre en compte lors de la composition d'un schéma de communication est le surcoût des instructions MPI. C'est le temps d'exécution des instructions de la librairie MPI, non pas le temps lié au transfert du message dans le réseau de communication, mais à l'exécution d'opérations internes à la librairie MPI. On a vu que les instructions non bloquantes demandent généralement des instructions supplémentaires pour garantir leur bonne tenue, typiquement *MPI_Wait*. Ces instructions supplémentaires vont avoir un coût qui peut être non négligeable. Certaines stratégies de réduction du surcoût peuvent ainsi s'opposer à l'utilisation d'instructions bloquantes.

Pour avoir un schéma de communication efficace, il convient de limiter les instructions bloquantes ainsi que le surcoût en réduisant le nombre d'appels à la librairie MPI et en augmentant les possibilités de recouvrement du schéma. Certains de ces éléments pouvant entrer en conflit, les développeurs doivent donc trouver un équilibre permettant de mener le plus efficacement possible une communication donnée en prenant en compte toutes ses spécificités.

2 Schémas de communication dans NEMO

2.1 Organisation et utilisation des schémas de communication

L'implémentation des routines de communications dans NEMO permet de regrouper des communications sur plusieurs tableaux en une seule communication et ainsi de limiter les instructions MPI. Tous les appels à des communications (sauf pour quelques communications particulières, dans le cas des icebergs par exemple) font appel à une unique routine : *lbc_lnk* qui peut prendre en arguments plusieurs tableaux de même dimension.

La routine de communication *lbc_lnk* est organisée en plusieurs étapes :

1. Regrouper les tableaux en arguments dans un unique tableau comprenant une dimension supplémentaire.
2. Sélectionner le schéma de communication adapté pour la suite de la communication.
3. La communication en tant que telle suivant le schéma choisi. C'est cette étape sur laquelle se concentre les développements présentés dans ce chapitre.

Avant les optimisations présentées dans ce chapitre, NEMO comportait deux schémas de communications, le schéma *EW-NS* et le schéma *neighborhood collective*. Ces schémas ainsi que ceux que nous avons développés sont présentés dans la suite de ce chapitre.

Les schémas de communication développés dans cette thèse ne peuvent pas tous être utilisés dans toutes les parties du code (tableau IV.1). En effet, certaines implémentations s'appuient sur des caractéristiques spécifiques à certaines communications et demandent parfois des ajustements de la routine de calcul qui appelle la routine de communication. J'ai choisi la routine du calcul du gradient de pression de surface pour ces schémas parce qu'elle réalise un grand nombre de communications (voir chapitre III) et est relativement concise et facile à modifier. Les schémas qui ne sont pas indiqués comme utilisables partout sont tous d'ores et déjà utilisables pour les communications du calcul du gradient de pression de surface et pourraient être étendues aux autres parties du code si cela s'avère approprié.

Schéma de communication	Utilisable partout
EW-NS	Oui
Waitall	Oui
Async	Non
Neighborhood collective	Oui
Persistent calls	Non
RMA	Non

TABLE IV.1 – Tableau récapitulatif des possibilités d'utilisation des schémas de communications.

Pour utiliser les schémas de communication uniquement disponibles pour le calcul du gradient de pression de surface, il faut le combiner avec un autre schéma de communication utilisable sur l'ensemble du code pour mener à bien le reste des communications. Une première modification du code permet cette utilisation. Un paramètre de namelist est ajouté pour spécifier si un schéma de communication différent doit être utilisé dans les routines de calcul du gradient de pression. La deuxième étape de *lbc_lnk* qui se charge de sélectionner le schéma de communication est modifié pour prendre en compte si la routine est appelée depuis le calcul du gradient de pression ou non.

Cette structure permet de changer facilement de schéma de communication entre différentes exécutions du code sans repasser par une étape de recompilation mais par un simple changement de namelist. On peut ainsi comparer facilement les routines de communications. De plus, il est facile d'ajouter des routines de communications pour tester des nouvelles implémentations et évaluer leurs performances. Par ailleurs, cette organisation des routines permet de regrouper toute la complexité des routines de communications dans une seule routine ce qui rend le code plus lisible et maintenable.

2.2 Structure des schémas de communication

Les schémas de communications dans NEMO sont tous organisés suivant une structure similaire.

On a vu dans le chapitre I que les halos des tableaux doivent être mis à jour par les communications. Les instructions MPI qui effectuent ces communications demandent que les données soient contiguës en mémoire. Or ce n'est pas le cas pour les données de certains halos d'un

tableau et, *a fortiori*, pour les données de halos issus de multiples tableaux. Les schémas de communication dans NEMO font donc appel à des tableaux utilisés spécifiquement pour contenir les données à envoyer ou à recevoir et cela de manière contiguë. On appelle ces tableaux des buffers. Les instructions d'envoi sont donc précédées d'une copie des données vers les buffers d'envoi et les instructions de réception sont suivies d'une copie des buffers vers les tableaux en arguments de la routine de communication.

MPI propose certaines fonctionnalités pour transférer des données qui ne sont pas contiguës en mémoire, les *MPI_Datatype*. Cette possibilité a été testée mais ne semble pas apporter des gains en performances dans le cas de NEMO. L'utilisation de buffers est donc conservée.

Il faut noter que, même si un processus manipule toujours un sous-domaine de même dimension, les différentes communications effectuées au cours d'un pas de temps gèrent des buffers de taille différentes. Cela est dû au fait que différentes communications peuvent prendre en argument un nombre très variable de tableaux, tableaux qui peuvent être de différentes dimension.

2.3 *EW-NS*

Le schéma utilisé systématiquement jusqu'à la version 3.6 de NEMO est le schéma *East-West North-South, (EW-NS)*.

```
Remplir Buffers d'Envoi Est-Ouest
CALL MPI_ISEND ⇨
CALL MPI_RECV ⇦
Mise à jour du domaine avec les valeurs reçues


Remplir Buffers d'Envoi Nord-Sud
CALL MPI_ISEND ⇨
CALL MPI_RECV ⇦
Mise à jour du domaine avec les valeurs reçues

CALL MPI_WAITALL(send)
```

FIGURE IV.4 – Pseudo-code du schéma de communication *EW-NS*.

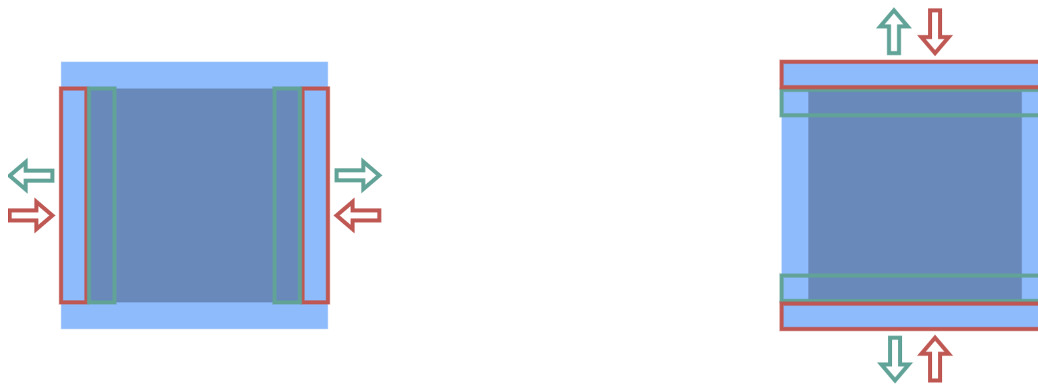


FIGURE IV.5 – Schéma explicatif du schéma de communication *EW-NS*. Le même sous-domaine MPI associé à un processus est représenté deux fois, à gauche le processus effectue la première étape du schéma et à droite la deuxième. L'intérieur du sous-domaine est en bleu foncé et les halos en bleu clair, les points dont les valeurs sont envoyées sont encadrés en vert et les points mis à jour par une réception sont encadrés en rouge.

La figure IV.4 affiche le pseudo code du schéma de communication *EW-NS* et la figure IV.5 en fournit un schéma explicatif. Il est composé de deux parties. La première s'occupe des échanges avec les sous-domaines voisins à "l'est" et à "l'ouest" et la seconde avec les sous-domaines voisins au "nord" et au "sud".

Chaque partie effectue tout d'abord une copie des valeurs du tableau vers les buffers qui seront utilisés pour les instructions d'envoi. Deux instructions d'envoi non bloquantes et deux réceptions bloquantes sont ensuite exécutées, chacune utilisant son propre buffer. Chaque partie se termine par la mise à jour du domaine avec les valeurs reçues par une copie des buffers de réception vers le tableau. Une fois les deux parties complétées, l'instruction bloquante *MPI_Waitall* se charge de s'assurer que les instructions d'envoi non-bloquantes se sont bien terminées.

Ce schéma de communication permet donc de mettre à jour les coins du sous-domaine tout en ne communiquant qu'avec quatre voisins, ce qui est rendu possible par l'utilisation d'instructions MPI bloquantes. Dans une vue idéalisée du code où tous les processus sont synchronisés, c'est un avantage considérable.

Les instructions bloquantes et la mise à jour des tableaux à la fin des communications "est" et "ouest" garantissent que les coins des halos sont pris en charge. En effet, le coin "nord-est" à l'intérieur d'un sous-domaine de référence (indiqué V_1 sur le schéma IV.6) sera envoyé dans un premier temps dans le halo du sous-domaine voisin "est", via la première communication suivie

de la copie du buffer de réception (encadré en rouge) dans le tableau. Ce sous-domaine copie ensuite les données de la totalité de la bande intérieure "nord" de son sous-domaine vers le buffer d'envoi (en vert). Comme la valeur V_1 est déjà écrite dans le halo "ouest" du fait de la phase bloquante de la réception, elle peut être copiée dans le buffer d'envoi puis envoyée au voisin "nord". L'envoi de l'intégralité de cette bande vient mettre à jour l'intégralité du halo "sud" (en rouge) du voisin "nord-est" du sous-domaine de référence. Le coin "sud-ouest" de ce voisin est donc mis à jour par la valeur V_1 du coin intérieur "nord-est" du sous-domaine de référence. Il en va de même pour les valeurs des autres coins. Ces valeurs sont mises à jour sans avoir à effectuer une communication directe entre le processus sur lequel se trouve la valeur et celui qui cherche à mettre à jour les valeurs des coins de ces halos.

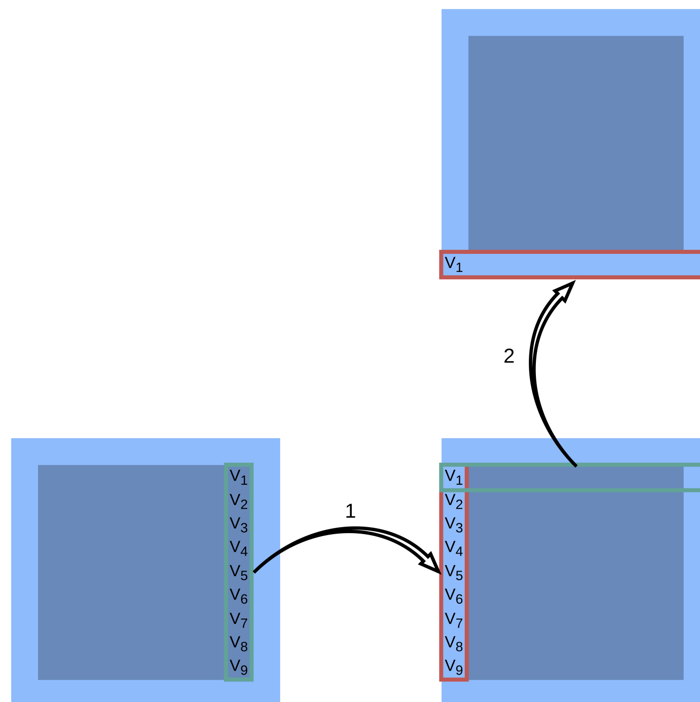


FIGURE IV.6 – Schéma de la transmission d'une valeur d'un coin nord-est d'un sous-domaine MPI vers le coin (sud-ouest) du sous-domaine voisin situé au nord-est dans une communication utilisant le schéma de communication *EW-NS*. La valeur de ce coin est indiquée V_1, les buffers d'envois sont mis en évidence en vert et les buffers de réceptions en rouge. Les flèches noires symbolisent les communications.

Le schéma présente néanmoins l'inconvénient de nécessiter plusieurs passages bloquants. Les instructions *MPI_ISend* sont non-bloquantes, à l'inverse des instructions *MPI_Recv* qui le sont. La routine de communication *EW-NS* contient donc quatre instructions bloquantes de réception se suivant deux par deux et une cinquième avant la désallocation des buffers d'envoi.

Prenons le point de vue d'un processus associé à un sous-domaine quelconque, lorsque le processus du sous-domaine situé du côté "est" est en retard, le processus ne peut pas recevoir les données de la communication avec ce voisin. Un incident similaire se produit lorsque le processus du sous-domaine "ouest" est en retard. Le processus ne peut pas commencer les instructions de copie ni commencer à envoyer vers les sous-domaines "nord" et "sud".

2.4 Tentatives de réduction de la rigidité du schéma

Les optimisations apportées dans cette section visent à la réduction de la rigidité du schéma de communication par la suppression des instructions bloquantes ou par la limitation de leur impact en permettant le recouvrement du temps de communication. Dans un premier temps les phases bloquantes liées à la gestion des buffers sont étudiées et les instructions d'allocation de mémoire sont minimisées. Une multitude de schémas de communication est ensuite développée et analysée.

2.4.1 Réorganisation initiale de la routine de communication

En préambule d'optimisations changeant profondément la structure du schéma de communication, une réorganisation de la routine de communication est effectuée. Elle vise à regrouper au maximum les instructions redondantes et à déplacer certaines sections pour permettre du recouvrement.

La version initiale du schéma de communication *EW-NS* utilisait un tableau différent pour chaque buffer correspondant à un voisin avec lequel des messages étaient échangés. Pour un sous-domaine avec 4 voisins (à l'"est", à l'"ouest", au "nord" et au "sud") cela revient à manipuler 8 tableaux différents et donc de recourir à autant d'instructions d'allocation et de désallocation à chaque appel de la routine de communication. Cette multitude de tableaux est remplacée dans notre version par deux tableaux de taille suffisante pour contenir les données de tous les buf-

fers. Lors des transferts MPI c'est simplement la partie adéquate du tableau qui sera utilisée. La routine de communication ne manipule désormais plus que 2 tableaux de buffer. Les instructions d'allocation et de désallocations sont ainsi regroupées.

Similairement, les instructions d'attentes *MPI_Wait* sont regroupées à chaque fois que cela est possible sans modification du schéma en une seule instruction *MPI_Waitall*. Par exemple, à la fin du schéma une série d'appels à *MPI_Wait* étaient jusqu'alors utilisés pour s'assurer que les instructions d'envoi *MPI_Isend* soient complétées. Les appels à *MPI_Wait* n'étaient entrecoupés d'aucune autre instruction ou calcul permettant du recouvrement et sont donc regroupés en un unique appel à *MPI_Waitall* sans que cela nécessite d'autres ajustements.

Par ailleurs, les routines de communication ont une étape qui n'est pas représentée sur les schémas dans ce travail et n'est pas exécutée du fait des configurations utilisées. Dans certaines configurations, pour les sous-domaines au bord du domaine de la simulation, les valeurs des halos vers l'extérieur du domaine peuvent être fixées à une valeur constante ou a une valeur prise en un point de ce même sous-domaine. Dans les deux cas, aucune donnée d'un autre sous-domaine de la simulation n'est utilisée et un transfert MPI est inutile. En revanche, c'est la routine de communication qui se charge de fixer ces valeurs lors d'une étape dédiée. Ce travail, jusqu'à présent effectué en tout début de routine, est déplacé de manière à permettre du recouvrement dans les schémas où cela est possible, par exemple en le déplaçant juste avant les instructions bloquantes. Cette optimisation ne concerne que certains processus de la simulation selon la configuration. Elle sera cependant sans effet dans nos configurations où tous les halos sont remplis par des échanges MPI, les performances apportées ne sont donc pas quantifiées.

De plus, certains échanges MPI inutiles sont supprimés. Lors de l'initialisation de NEMO une procédure de détection des échanges MPI ne contenant que des valeurs issues de points terre est effectuée. En effet, dans certaines simulations, deux sous-domaines voisins peuvent n'avoir que des points terre à leur interface, par exemple dans le cas de sous-domaines situés d'un côté et de l'autre d'une péninsule. Les processus associés à ces sous-domaines ne seront donc pas supprimés de l'exécution car ils comportent des points océans, mais un échange MPI entre eux

est inutile. Cette détection est faite en contrôlant si la totalité d'un halo est composé de points terre, auquel cas la réception de données du voisin est supprimée. Inversement, si la totalité des points utilisés pour mettre à jour les valeurs des points de halos du voisin sont des points terre, l'envoi est supprimé. De même, cette optimisation ne concerne que certains processus de certaines simulations et n'aura pas d'impact dans nos configurations où aucun point terre n'est défini. Les performances apportées ne sont pas non plus quantifiées.

2.4.2 Gestion des buffers d'envoi et de réception

Il est possible de déplacer les parties bloquantes liées à la désallocation des buffers d'envoi. En effet, l'instruction bloquante sur ces buffers doit être effectuée avant l'utilisation de ces mêmes variables lors du prochain appel à la routine de communication. On peut donc déplacer les instructions bloquantes de la fin de la routine de communication vers le début de cette routine comme présenté sur le pseudo-code figure IV.7. Ce code commence en effet par une instruction bloquante *MPI_Waitall* sur les buffers d'envoi. Ce n'est pas le cas de l'instruction *MPI_Waitall* sur les buffers de réception qui doit être effectuée avant la mise à jour du tableau.

```
CALL MPI_WAITALL(send)
DEALLOCATE(Buffers d'Envoi)
ALLOCATE( Buffers d'Envoi et Réception )

Reste du schéma de communication

CALL MPI_WAITALL(recv)
Mise à jour du domaine avec les valeurs reçues
DEALLOCATE(Buffers de Réception)
```

FIGURE IV.7 – Pseudo-code d'une version préliminaire de la gestion des buffers.

On voit sur la figure IV.7 que les phases de désallocation et d'allocation des buffers se suivent. Si le buffer désalloué est suffisamment grand pour convenir au nouveau passage dans la routine de communication, il est possible de ne pas le désallouer et de le réutiliser. Il est possible de faire de même pour les buffers d'envoi. On évite alors les étapes d'allocation et de désallocation, cette possibilité est présentée figure IV.8.

```
CALL MPI_WAITALL(send)
IF( Buffers d'Envoi trop petit ) THEN
    DEALLOCATE( Buffers d'Envoi )
    ALLOCATE( Buffers d'Envoi )
ENDIF
IF( Buffers de Réception trop petit ) THEN
    DEALLOCATE( Buffers de Réception )
    ALLOCATE( Buffers de Réception )
ENDIF

Reste du schéma de communication

CALL MPI_WAITALL(recv)
Mise à jour du domaine avec les valeurs reçues
```

FIGURE IV.8 – Pseudo-code d'une version préliminaire de la gestion des buffers.

Ces deux implémentations impliquent qu'au moins un buffer reste alloué en mémoire en dehors de la routine de communication, ce qui, finalement, peut être un inconvenient pour les performances. De plus, dans la seconde implémentation, le buffer est rapidement fixé à sa taille maximale. Pour pallier ce problème on peut faire ce décalage et la suppression des phases d'allocation et de désallocation uniquement lorsque les buffers sont suffisamment "petits" et n'ont donc, a priori, pas d'impact significatif sur les performances liées à la mémoire. On définit arbitrairement la taille maximale à la taille d'un tableau à deux dimensions sur le processus, c'est-à-dire $iszmax = jpi \times jpj$.

On obtient alors le code présenté figure IV.9 qui déplace le point bloquant sur les buffers d'envoi et réutilise les buffers que si les données en jeu ne sont pas trop importantes. Si les buffers requièrent trop de mémoire, on libère cet espace quitte à recourir à plus d'instructions d'allocations et de désallocations. L'implémentation consiste simplement en une fusion des deux précédentes, elle est compatible avec les schémas de communications développés par la suite à l'exception des RMA et des *persistent calls*.

```
IF( ALLOCATED( Buffers d'Envoi ) ) THEN
    CALL MPI_WAITALL(send)
    IF( Buffers d'Envoi trop petit ) DEALLOCATE( Buffers d'Envoi )
ENDIF
IF( .NOT. ALLOCATED( Buffers d'Envoi ) ) ALLOCATE( Buffers d'Envoi )
IF( ALLOCATED( Buffers de Réception ) ) THEN
    IF( Buffers de Réception trop petit ) DEALLOCATE( Buffers de Réception )
ENDIF
IF( .NOT. ALLOCATED( Buffers de Réception ) ) ALLOCATE( Buffers de Réception )

Reste du schéma de communication

CALL MPI_WAITALL(recv)
Mise à jour du domaine avec les valeurs reçues
IF( SIZE(Buffers d'Envoi) > iszmax ) THEN
    CALL MPI_WAITALL(send)
    DEALLOCATE( Buffers d'Envoi )
ENDIF
IF( SIZE(Buffers de Réception) > iszmax ) DEALLOCATE( Buffers de Réception )
```

FIGURE IV.9 – Pseudo-code de la gestion des buffers.

2.4.3 Condition préalable à la réduction de la rigidité des schémas

Une manière de réduire la rigidité du schéma de communication se fonde sur l'idée de briser l'ordre imposé par le schéma *EW-NS*. Or, c'est justement cet ordre imposé qui garantit que les valeurs des coins sont mises à jour en communiquant avec seulement 4 sous-domaines voisins. Cette démarche requière de passer à des schémas de communications faisant intervenir les 8 voisins pour communiquer directement les valeurs des coins avec les sous-domaines voisins situés dans les directions diagonales.

Les sections des tableaux échangés entre processus MPI doivent être modifiés en conséquence. La figure IV.10 montre ces secteurs pour l'envoi en vert et la réception en rouge. En ce qui concerne la réception (en rouge), les rectangles sont reçus en provenance des quatre voisins de chaque côté et les carrés reçus des voisins en diagonale. Le même principe est appliqué pour les envois (en vert) avec la particularité que les carrés chevauchent les rectangles. Ainsi, les données situées, par exemple, en bas à gauche, font à la fois partie du carré qui sera envoyé au voisin "sud-ouest", et aux 2 rectangles qui seront envoyés aux voisins "sud" et "ouest".

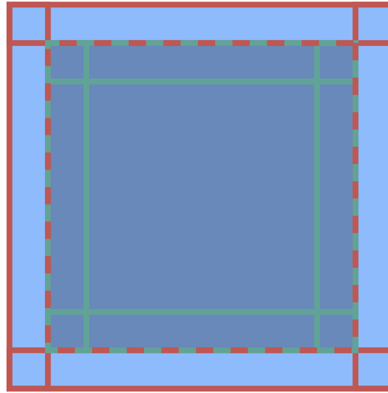


FIGURE IV.10 – Les points dont les valeurs sont envoyées sont encadrés en vert et les points mis à jour par une réception sont encadrés en rouge. Les lignes délimitant à la fois les rectangles d'envoi et de réception sont en pointillés rouge et vert. Chaque section encadrée communique avec un unique voisin, le carré rouge en bas à droite par exemple reçoit des données depuis le sous-domaine situé au "sud-est".

La réduction de la rigidité permet de réaliser des opérations en attendant que les communications des voisins les plus tardifs soient disponibles, ces opérations peuvent simplement être des opérations de réception et de copie de buffer. J'espère ainsi réduire le temps passé dans la routine de communication et de limiter l'impact d'un ralentissement d'un processus de la simulation décrit au chapitre III section 2.3.

2.4.4 Schéma de communication utilisant *MPI_Iprobe*

Pour réaliser les opérations de réception dès qu'elles sont disponibles une routine MPI renvoyant l'état d'avancement d'un échange peut être employé. J'ai dans un premier temps utilisé la routine *MPI_Iprobe* qui demande en entrée l'identifiant du transfert MPI, la routine renvoie une variable logique à la valeur $True$ si le message peut être reçu et $False$ sinon indiquant que les données ne sont pas encore arrivées. Les instructions qui sont détectées comme pouvant être reçues le sont puis les données sont copiées du buffer vers le tableau (voir figure IV.11). Ce processus est itéré jusqu'à ce que toutes les réceptions soient effectuées.

Ce schéma permet de réaliser les instructions de réception et de copie dès qu'elles peuvent être effectuées, sans ordre prédéfini, et ainsi prendre de l'avance sur des opérations alors que toutes les communications ne sont pas encore réceptionnées. On a ainsi un recouvrement de certaines communications par des instructions de réception et de copie.

```
Remplir Buffers d'Envoi
CALL MPI_ISEND ⇨
WHILE(Réception pas terminée)
    CALL MPI_Iprobe(recv)
    IF(recv disponible) THEN
        CALL MPI_RECV(recv) ⇦
        Mise à jour du domaine avec les valeurs reçues
    ENDIF
END WHILE
```

FIGURE IV.11 – Pseudo-code du schéma de communication utilisant *MPI_Iprobe*.

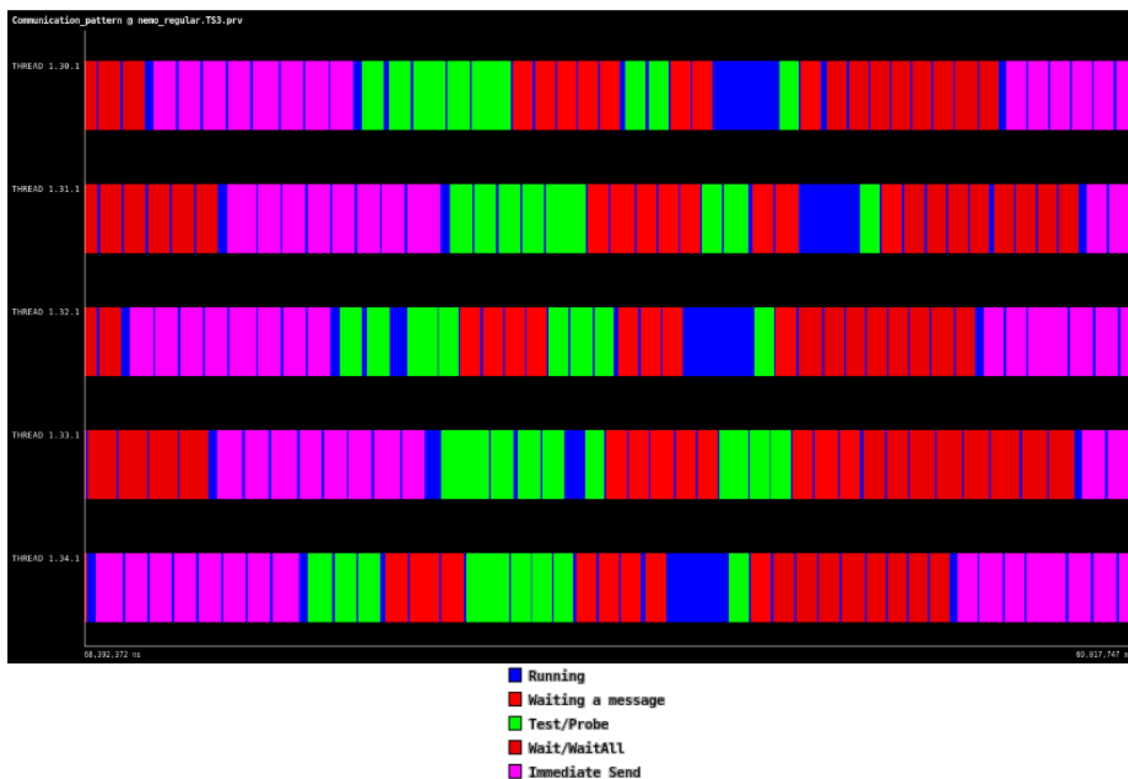


FIGURE IV.12 – Aperçu d'un profilage extrae et paraver du schéma de communication utilisant *MPI_Iprobe*. Chaque ligne horizontale représente l'état d'un processus au fil du temps, en bleu s'il effectue des calculs, rouge clair s'il reçoit un message MPI (*MPI_Recv*), en vert les instructions *MPI_Iprobe*, rouge foncé s'il attend un message MPI (*MPI_Wait* ou *MPI_Waitall*) et en rose les instructions d'envoi (*MPI_Isend*).

La figure IV.12 montre que les instructions *MPI_Iprobe* permettent de réaliser certaines opérations *MPI_Recv* en premier, celles correspondant à des messages directement disponibles à la réception, qui suivent directement la première salve d'instructions *MPI_Iprobe*. Les messages restants seront reçus par les instructions *MPI_Iprobe* et *MPI_Recv* suivantes. Les processus de la figure IV.12 reçoivent les messages en deux ou trois salves. Tous les processus, sauf le deuxième en partant du bas, ont une phase indiquée comme une phase de calcul assez longue

précédent les dernières instructions *MPI_Iprobe* et *MPI_Recv*. Cette phase correspond aux nombreux passages dans la boucle réalisant des appels à *MPI_Iprobe* jusqu'à ce que des messages puissent être réceptionnés. Les instructions de réceptions et de copies ne sont donc pas suffisantes pour recouvrir le temps de communication des communications les plus lentes. Cependant, même si le recouvrement n'est pas total, cette flexibilité dans l'ordre des réceptions peut être bénéfique. En effet, si la réception qui demande le plus de temps devait obligatoirement être effectuée en premier, elle prendrait tout aussi longtemps, mais toutes les autres réceptions et copies devraient encore être réalisées au lieu de passer à la suite. La dernière instruction d'attente indiquée en rouge est l'instruction *MPI_Waitall* sur la totalité des *MPI_Isend*.

Le schéma présenté ici a deux caractéristiques qui entravent ses performances. Premièrement, il nécessite beaucoup d'instructions MPI, au minimum deux pour chaque réception, ce qui ralenti considérablement l'exécution. Deuxièmement, la boucle d'appels à *MPI_Iprobe* est problématique en ce qu'elle exécute parfois de très nombreuses instructions alors qu'il suffirait d'attendre.

Le profilage extrae et paraver de la figure IV.12 tend à exagérer le temps de calcul des phases courtes et donc le coût lié à l'utilisation de très nombreuses instructions telles que *MPI_Iprobe*. Un test sur le temps d'exécution total nous permet d'éliminer le surcoût lié à extrae et paraver et montre que ce schéma a des performances moins bonnes que le schéma de référence (*EW-NS*). J'ai donc cherché à l'améliorer.

2.4.5 Schéma de communication utilisant *MPI_Waitany*

La routine *MPI_Iprobe* peut être remplacée par *MPI_Waitany* pour limiter le nombre d'instructions et éliminer la boucle d'appel à *MPI_Iprobe* (voir figure IV.13). La routine *MPI_Waitany* attend jusqu'à ce que n'importe quelle communication soit disponible à la réception. Les itérations inutiles de la boucle d'appel présentent avec *MPI_Iprobe* sont éliminées parce que cette routine implémente elle-même la phase d'attente. *MPI_Waitany* réduit le nombre d'instructions car un seul appel permet de détecter si une communication de n'importe quel processus voisin est disponible.

```
Remplir Buffers d'Envoi
CALL MPI_ISEND ⇨
CALL MPI_IRECV ⇦
WHILE(Réception pas terminée)
    CALL MPI_Waitany(recv)
    Mise à jour du domaine avec les valeurs reçues
END WHILE
```

Figure IV.13 – Pseudo code du schéma de communication utilisant *MPI_Waitany*

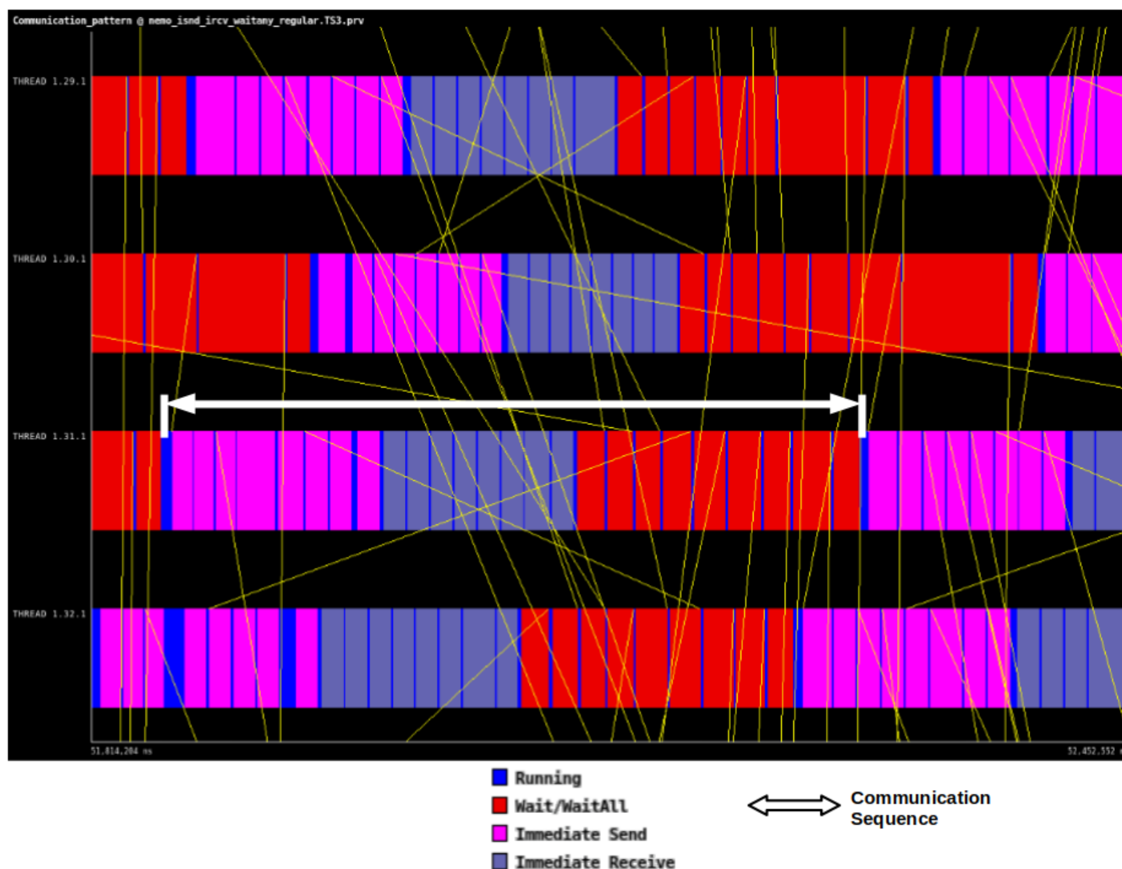


Figure IV.14 – Aperçu d'un profilage extrae et paraver du schéma de communication utilisant *MPI_Waitany*. Chaque ligne horizontale représente l'état d'un processus au fil du temps, en bleu s'il effectue des calculs, rouge foncé s'il attend un message MPI (*MPI_Waitany* ou *MPI_Waitall*), en rose les instructions d'envoi (*MPI_Isend*) et en bleu pastel celle de réception (*MPI_Irecv*). La flèche blanche indique l'étendue d'une séquence de communication.

La figure IV.14 montre un profilage du schéma de communication. Les instructions *MPI_Waitany* permettent d'éviter des passages dans une boucle, le processus attend en effet au sein de *MPI_Waitany*. Ici encore la dernière instruction d'attente indiquée en rouge est l'instruction *MPI_Waitall* sur la totalité des *MPI_Isend*.

L'utilisation de *MPI_Waitany* permet de conserver le recouvrement des communications par les instructions de réception et de copie tout en réduisant le coût du grand nombre d'appel à *MPI_Iprobe*. Le nombre d'appels, même réduit, à *MPI_Waitany* suffit à causer une perte de performance et rendre ce schéma de communication finalement peu efficace.

2.4.6 Schéma de communication non-bloquant

Pour limiter au mieux le nombre d'appels aux routines MPI, on peut s'assurer qu'une seule instruction MPI est en charge d'attendre que tous les messages soient reçus. L'instruction MPI utilisée est *MPI_Waitall* qui attend jusqu'à ce que toutes les instructions MPI données en arguments soient complétées. Le pseudo code auquel on parvient est présenté figure IV.15. On n'utilise toujours que des communications non-bloquantes mais on perd alors la possibilité de recouvrir le temps de communication par le temps de copie car la copie des buffers de réception ne peut être réalisée qu'une fois que *MPI_Waitall* assure que toutes les réceptions ont été faites.

```
Remplir Buffers d'Envoi
CALL MPI_ISEND ⇨
CALL MPI_IRECV ⇦
CALL MPI_WAITALL(recv)
Mise à jour du domaine avec les valeurs reçues
```

FIGURE IV.15 – Pseudo code du schéma de communication non bloquant utilisant *MPI_Waitall*.

La figure IV.16 montre un profilage utilisant les outils extrae et paraver. Le fait d'avoir une seule instruction assurant que toutes les réceptions ont bien été effectuées permet de réduire le surcoût dû au nombre d'appels à des instructions MPI. La somme du temps passée dans les instructions *MPI_Waitany* ou *MPI_Iprobe* est supérieure au temps passé dans l'unique instruction *MPI_Waitall*. Comme l'explique la section 3 du chapitre II le temps passé dans plusieurs segments profilés par extrae peut être surestimé par rapport au profilage d'un unique segment. Pour être certain des performances de ces 3 schémas de communication, on peut mesurer le temps de calcul d'un pas de temps moyen sans le poids de l'instrumentation d'un profilage.

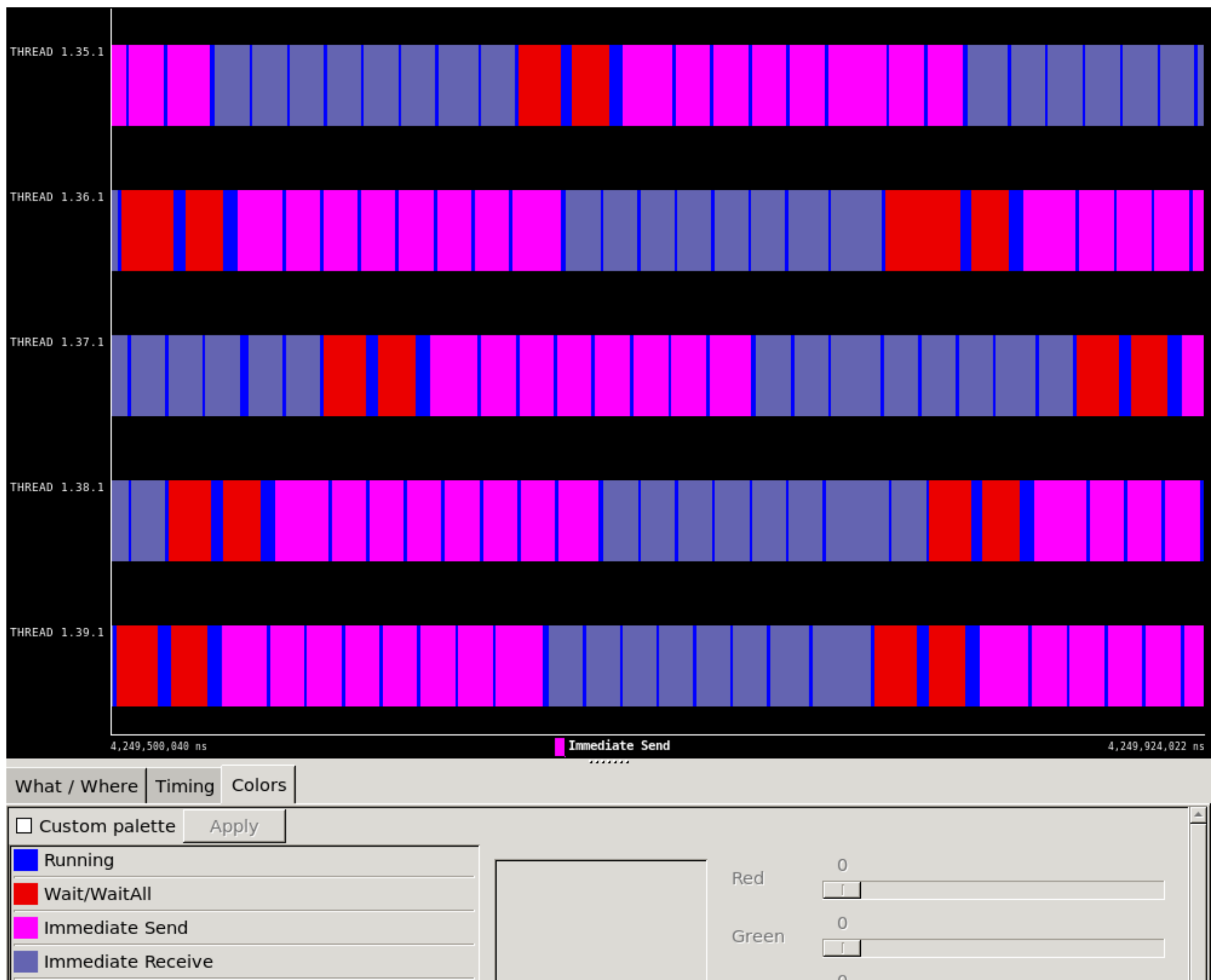


FIGURE IV.16 – Aperçu d'un profilage extrae et paraver du schéma de communication utilisant *MPI_Waitall*. Chaque ligne horizontale représente l'état d'un processus au fil du temps, en bleu s'il effectue des calculs, rouge foncé s'il attend un message MPI (*MPI_Waitall*), en rose les instructions d'envoi (*MPI_Isend*) et en bleu pastel celle de réception (*MPI_Irecv*).

Nos tentatives de réduire la rigidité du schéma de communication pour bénéficier d'un recouvrement des communication MPI par le temps d'exécution se heurtent au surcoût des instructions MPI qui permettent de maximiser ce recouvrement. Le schéma le plus efficace est finalement le schéma utilisant l'instruction *MPI_Waitall*, c'est-à-dire celui qui permet le moins de recouvrement.

2.4.7 Comparaison de ces schémas

Les schémas développés jusqu'à présent sont comparés sur la figure IV.17. Les résultats sont mesurés sur la configuration TSUNAMI avec des sous-domaines de 30 sur 30 points pour 19200 cœurs Skylake du supercalculateur Irene. Le schéma indiqué *référence* correspond à la version du schéma de communication avant nos modifications, les "*premières améliorations*" font

référence aux modifications de la section 2.4.1 et *EW-NS* est le schéma auquel on parvient après ces modifications et les changements relatifs à la gestion des buffers de la section 2.4.2. Les schémas *Iprobe* et *Waitany* sont ceux construits à partir des instructions *MPI_Iprobe* et *MPI_Waitany*. Enfin, le schéma "*non bloquant*" est le schéma construit à partir de *MPI_Waitall*.

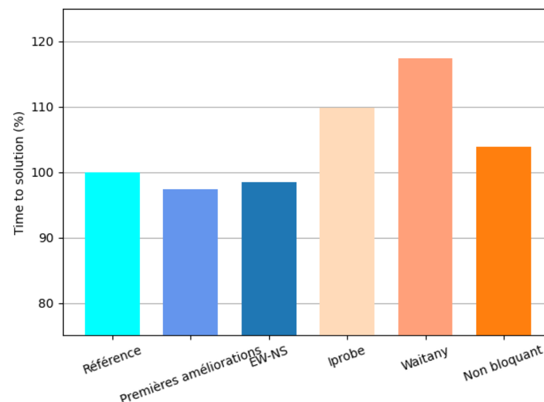


FIGURE IV.17 – Performances des schémas de communications en temps de calcul médian des pas de temps. Les schémas de communication sont exprimés en pourcentages du temps de calcul du schéma de *référence*.

Tout d'abord, les améliorations de la section 2.4.1 apportent effectivement un gain dans les performances avec environ 3% de réduction du temps d'exécution. La nouvelle gestion des buffers qui sépare le schéma "*premières améliorations*" du schéma *EW-NS* ne semble pas être bénéfique pour les performances. Notons tout de même qu'une mesure des performances via la moyenne des pas de temps indique que le schéma *EW-NS* est le plus rapide.

Le schéma *EW-NS* est conservé comme schéma de référence dans la suite de ce travail et est ajouté dans la version de référence de NEMO.

Les schémas de communication développés pour permettre un recouvrement du temps de communication (*Iprobe*, *Waitany* et "*non bloquant*") n'ont pas de très bonnes performances et sont plus lents que le schéma de référence. Le schéma "*non bloquant*" présente les meilleures performances des trois, les deux autres schémas sont donc abandonnés dans les développements qui suivent.

2.5 Recouvrement communication-calcul

Les schémas développés jusqu'à présent pour concurrencer le schéma *EW-NS* n'ont pas fourni les performances escomptées. On explore à présent des fonctionnalités et stratégies MPI plus avancées.

Une première stratégie est le recouvrement du temps de communication par des calculs. Contrairement à la section 2.4 le recouvrement visé n'est pas le recouvrement des communications par des instructions propres à la routine de calcul, mais le recouvrement des communications par des phases de calcul hors de la routine de communication.

```
Calcul sur les bords
Remplir Buffers d'Envoi
CALL MPI_ISEND ⇨
CALL MPI_IRECV ⇦
Calcul sur l'interieur
CALL MPI_WAITALL(recv)
Mise à jour du domaine avec les valeurs reçues
```

FIGURE IV.18 – Pseudo code du schéma de communication avec recouvrement communication-calcul.

FIGURE IV.19 – Schéma explicatif du schéma de communication avec recouvrement communication-calcul. Les trois étapes sont représentées de gauche à droite, premièrement, à gauche, le calcul sur les zones hachurées dont les différentes sections encadrées en vert sont ensuite envoyées aux voisins. Deuxièmement, au milieu, le calcul sur le reste de l'intérieur du domaine, sur la zone hachurée. Troisièmement, à droite, la phase bloquante de la réception des messages.

La figure IV.18 montre une implémentation simplifiée d'un schéma de communication effectuant un recouvrement des communications par des calculs. La routine commence par le calcul sur le bord du domaine uniquement de manière à pouvoir envoyer les données le plus tôt possible. Les instructions de réception non bloquantes sont exécutées à la suite. Les calculs sur le reste du domaine, délimité par la zone hachurée figure IV.19 au milieu, permettent au processus

d'avancer ses calculs en attendant que les messages de ses voisins puissent être réceptionnés. Cette étape constitue le recouvrement. Une fois ces calculs terminés, l'instruction *MPI_Waitall* s'assure que la réception a bien eu lieu, puis le tableau est mis à jour sur ses halos.

Pour que l'implémentation de ce schéma de communication soit possible, il faut fournir en argument de la routine de communication une routine de calcul qui servira au recouvrement. La routine de calcul doit prendre en argument les indices délimitant la partie du tableau sur laquelle doit s'effectuer les calculs pour pouvoir calculer séparément sur le bord du domaine et l'intérieur du domaine.

Du fait de l'implémentation de NEMO, des routines sont rarement disponibles telles quelles pour être fournies en argument de la routine de communication. Pour que cela soit possible, il faut trouver des calculs qui n'aient pas de dépendances avec le résultat de la routine de communication. Dans notre implémentation, on utilise des calculs effectués directement après la routine de communication mais qui ne dépendent pas de son résultat. Le code de NEMO est modifié de manière à envelopper ces calculs dans une routine qui pourra être utilisée dans la routine de communication. Pour cela, les limites des boucles de calcul doivent être remplacées par les arguments d'entrée de la routine. Le code doit donc être modifié à chaque fois que la routine de communication est appelée, une omniprésence de ces modifications peut rendre le code difficile à lire. Dans nos implémentations, nous nous sommes limités à la routine de calcul du gradient de pression de surface, dans un second temps les mêmes stratégies pourraient être étendues à d'autres parties du code. Par exemple, à l'algorithme du calcul de la rhéologie (déformation) de la glace qui comporte des caractéristiques similaires : une boucle d'une centaine d'itérations avec peu de calculs et des communications.

2.6 Neighborhood collective

La librairie MPI fournit la possibilité de réaliser automatiquement la totalité du schéma de communication en une seule instruction MPI, c'est les *neighborhood collectives*. La figure IV.20 montre que le schéma de communication se résume alors au remplissage des buffers, d'une instruction MPI (*mpi_neighbor_alltoallv*) et de la mise à jour du domaine.

Une phase d'initialisation permet d'initialiser un graphe en indiquant à la librairie MPI via l'instruction *MPI_Dist_graph_create_adjacent*. Cette routine prend en argument la liste des processus avec lesquels des communications doivent être effectuées et prépare ainsi les communications à venir en renvoyant un objet MPI "*Communicator*". Lors de l'exécution de la routine de communication, ce communicateur est utilisé en argument de la routine *mpi_neighbor_alltoallv* qui réalise l'ensemble de la communication préparée, envoi, réception et attente compris.

```
Remplir Buffers d'Envoi
CALL MPI_neighbor_alltoallv( Communicator )
Mise à jour du domaine avec les valeurs reçues
```

FIGURE IV.20 – Pseudo code du schéma de communication *neighborhood collectives*.

Ce schéma a plusieurs avantages, premièrement celui de demander un unique appel à une instruction de la routine MPI et donc potentiellement de limiter le coût par rapport à d'autres schémas de communication utilisant un plus grand nombre d'instructions. Deuxièmement, la phase d'initialisation permet probablement d'y déplacer certaines procédures et ainsi de ne pas les exécuter à chaque appel de la routine de communication. Troisièmement, les informations indiquées à la librairie MPI peuvent permettre une optimisation de la communication interne à la librairie. Cela peut présenter l'avantage de laisser la tâche de l'optimisation à librairie elle-même, ce qui réduit la complexité du code. De plus, on laisse aux développeurs de la librairie MPI, plus au fait des possibilités d'optimisation, la tâche de l'optimisation. On bénéficie ainsi des nouvelles optimisations sans modification du code mais simplement par une mise à jour de la librairie MPI.

L'inconvénient majeur est que, du fait de l'unique instruction MPI utilisée, il est difficile d'avoir du recouvrement avec ce schéma de communication. Il est possible que les collectives neighborhoods permettent un recouvrement des communications avec des instructions MPI, mais il n'est pas possible d'utiliser la copie des buffers dans les tableaux ou des routines de calcul comme dans la section 2.5.

2.7 *Persistent calls*

Une autre fonctionnalité de la librairie MPI permet de limiter le travail que doit effectuer la librairie via les *persistent calls* qui réutilisent les *request* MPI et n'ont ainsi pas besoin d'exécuter la partie des instructions qui produisent le *request*. On espère ainsi réduire le surcoût lié à l'utilisation de la librairie MPI, le temps passé dans la communication du fait du transfert dans l'InterConnect et de l'asynchronisme entre processus n'est à priori pas modifié.

Les *request* MPI visent un processus donné et sont liées à un tableau, dans le cas des communications dans NEMO à un buffer alloué spécialement, et donc à la taille de ce buffer. Les *persistent calls* sont donc adaptés à un ensemble de communications échangeant un volume de données identique ou très semblable, par exemple dans le calcul du gradient de pression de surface.

Lors de l'initialisation de la simulation, on alloue des buffers dédiés pour chaque envoi ou réception effectués par les *persistent calls*. Le calcul du gradient de pression de surface a deux communications par itération de la boucle de calcul qui communiquent des messages de tailles différentes. Un choix évident serait d'allouer le double de buffers et d'initialiser le double d'envois et de réceptions, mais cela ferait que MPI aurait un nombre important de *request* à gérer, le danger étant de rajouter un certain surcoût justement là où on essaye de le limiter. Par ailleurs, la taille des buffers étant petite dans le calcul du gradient de pression de surface, elle n'est pas un problème pour la transmission dans le réseau d'InterConnect. On choisit donc d'avoir des buffers associés à une seule communication, c'est-à-dire seulement huit (en comptant les voisins diagonales). La taille des buffers est choisie pour être compatible avec la communication

demandant le transfert du plus de données. Pour l'autre communication, la totalité du buffer est échangée mais seulement une partie contient des informations pertinentes et fait l'objet de copies depuis et vers les tableaux en argument de la routine de communication.

```
Remplir Buffers d'Envoi
CALL MPI_Startall(requests)
CALL MPI_Waitall( requests)
Mise à jour du domaine avec les valeurs reçues
```

FIGURE IV.21 – Pseudo code du schéma de communication *persistent calls*.

Les *persistent calls* permettent d'initialiser différents types d'instructions de communications. Dans un premier temps la routine de communication avec *persistent calls* utilise des communications non bloquantes de manières similaires au schéma *MPI_Waitall*. Elles sont initialisées avec les instructions *MPI_Send_Init* et *MPI_Recv_Init* en renseignant les processus en jeux, les buffers utilisés et la taille des messages envoyés, ces routines renvoient ensuite des *request* MPI.

Dans la routine de communication, un unique appel à la routine *MPI_Startall* lance les instructions d'envoi et de réception non bloquantes (voir figure IV.21). Elle remplace les appels aux routines *MPI_Isend* et *MPI_Irecv*, donc potentiellement 16 instructions. Une seconde instruction *MPI_Waitall* sert à vérifier que toutes les instructions d'envoi et de réception sont terminées avant la copie des buffers dans les tableaux et la fin de la routine de communication.

Les *persistent calls* préparent et regroupent des instructions MPI classiques, contrairement aux collectives neighborhoods où plus de latitude est accordée à la librairie MPI pour déterminer la meilleure manière d'effectuer la communication. De plus, les *persistent calls* utilisent 2 appels à des instructions MPI, soit une de plus que les *neighborhood collectives*. En revanche, une plus grande partie des instructions MPI est déplacée à l'initialisation. Malgré leurs similitudes, ces deux schémas de communication peuvent donc mener à des performances différentes.

Le schéma de communication implémenté ici est une version élémentaire de ce que propose les *persistent calls*. Cette technique peut être combinée avec d'autres stratégies vues précédemment. Des futures implémentations peuvent inclure des recouvrements avec des routines de calcul comme dans la section 2.5 ou des changements du schéma de communication.

2.8 *Remote Memory Access* (RMA)

Une dernière possibilité pour limiter le temps de communication est d'utiliser des communications RMA. Ces communications consistent en un accès direct, en écriture ou en lecture, dans l'espace mémoire d'un autre processus. Ces communications sont censément plus rapides parce que la communication ne nécessite que des instructions de la part d'un seul des processus impliqués.

Les données ne peuvent être écrites ou lues que dans certains espaces mémoire spécifiques désignés lors de l'initialisation, les *windows* RMA. Elles peuvent être définies directement sur les tableaux sur lesquels sont effectués les calculs sans passer par un buffer. Cette manière de faire est plus cohérente avec l'esprit des RMA de limiter les opérations encadrant les communications et ainsi de permettre un plus grand asynchronisme.

Cela a cependant plusieurs inconvénients. Premièrement il faut multiplier le nombre de window RMA utilisées, une pour chaque tableau sur lequel on effectue ce type de communication. Deuxièmement, puisque les *windows* sont définis sur les tableaux, les instructions de communications doivent être effectuées pour chaque tableau au lieu de pouvoir les regrouper comme c'est le cas en utilisant un buffer avec la routine *lbc_lnk*. Il devient nécessaire de multiplier les instructions MPI ce qui est en contradiction avec la logique de réduction du temps passé dans des instructions MPI. Troisièmement, l'implémentation d'une telle stratégie est compliquée, ne peut pas s'appuyer sur les routines de communications déjà existantes et demande de modifier plus de parties du code, y compris dans les parties non dédiées aux communications, rendant ainsi le code peu lisible et moins maintenable.

C'est pour toutes ces raisons que l'on choisit de définir les *windows* RMA sur les buffers de réception des communications qui doivent donc être alloués à une taille fixe pendant l'initialisation. Encore une fois, on choisit une taille de buffer compatible avec les communications du calcul du gradient de pression de surface. Le schéma RMA n'est pour l'instant opérationnel que pour ces communications.

```
Remplir Buffers d'Envoi
CALL MPI_WIN_FENCE
CALL MPI_PUT      ⇨
CALL MPI_WIN_FENCE
Mise à jour du domaine avec les valeurs reçues
```

Figure IV.22 – Pseudo code du schéma de communication RMA.

```
Calcul sur les bords
Remplir Buffers d'Envoi
CALL MPI_WIN_FENCE
CALL MPI_PUT      ⇨
Calcul sur l'interieur
CALL MPI_WIN_FENCE
Mise à jour du domaine avec les valeurs reçues
CALL MPI_WAITALL(send)
```

Figure IV.23 – Pseudo code du schéma de communication RMA avec recouvrement.

Étant donné que les instructions RMA d'écriture et de lecture sont par nature non bloquantes, la personne développant le code doit prendre elle-même des précautions pour garantir que les données sont mises à jour avant d'être utilisées. Pour le moment le schéma de communication RMA utilise des instructions *MPI_WIN_FENCE* qui délimitent des sections durant lesquelles l'accès aux *windows* RMA d'autres processus est possible. Elles se comportent comme des *mpi_barrier* qui synchronisent l'ensemble des processus et sont donc fortement nuisibles aux performances. Cette implémentation ne saurait donc être considérée comme une version terminée, mais plutôt comme une étape vers une implémentation performante.

Cela nous amène au schéma de communication présenté figure IV.22. Les deux instructions *MPI_WIN_FENCE* encadrent les instructions d'écriture chez tous les voisins : *MPI_PUT*. Une autre version de ce schéma incluant un recouvrement des communications avec des calculs est exposée figure IV.23. Le code qui en découle consiste en une fusion des RMA et du schéma présenté à la section 2.5, il nécessite lui aussi de spécifier une routine de calcul en argument de la routine de communication. Les calculs sur les bords du domaine de la routine de calcul sont effectués en premier de manière à envoyer les données au plus tôt avec l'instruction *MPI_PUT*. Avant le second appel à la routine bloquante *MPI_WIN_FENCE* le processus réalise le reste des calculs, c'est-à-dire sur l'intérieur du domaine.

Une piste d'optimisation de ce schéma serait de se passer des instructions *MPI_WIN_FENCE* à chaque communication. La section RMA délimitée par ces instructions pourrait être élargie pour englober l'ensemble de la simulation ou au moins la routine du calcul du gradient de pression de surface. Des communications RMA dites notifiées doivent alors être utilisées pour s'assurer de l'actualisation des valeurs du halo par d'autres processus. Elles reposent sur le principe suivant : les données habituelles sont transmises par les instructions RMA classiques mais accompagnées d'une donnée balise supplémentaire qui indique que cette transmission même a bien été effectuée. Un processus n'a alors qu'à vérifier la valeur de cette donnée balise qui certifie que la communication a eu lieu et que le calcul peut se poursuivre. Les RMA notifiées sont implémentées dans une librairie MPI développée par Atos, la *bullopenMPI*.

3 Performances des schémas de communication

3.1 Impact du profilage historique sur les communications

Avant nos travaux sur le profilage et l'évaluation des performances de NEMO expliqués au chapitre II, il existait déjà un profilage, activé par défaut en même temps que la mesure du temps d'exécution des pas de temps. Ce profilage est utilisé dans le but de fournir une estimation du temps passé dans les communications et plus précisément le temps passé dans les instructions MPI. C'est donc un profilage qui s'intéresse à des phénomènes de courte durée et demande plusieurs instructions de mesure de l'horloge par appel à la routine de communication. Il requière donc un nombre conséquent de mesures qui sont de plus réalisées par l'instruction *MPI_Wtime* qui est coûteuse.

Ce profilage a donc un impact significatif sur les performances, mais le problème principal est que le poids du profilage est très variable selon le schéma de communication utilisé. En effet, ce profilage concerne chaque appel à la librairie MPI et on a vu que le nombre d'appels aux instructions MPI peut fortement varier en fonction du schéma utilisé et du nombre de mesures de

l'horloge. Les performances de certains schémas seront légèrement pénalisées par le profilage et d'autres le seront fortement. Il est donc impossible de comparer de manière équitable les schémas de communication avec ce profilage.

L'activation par défaut du profilage dans les versions du code avant les travaux présentés au chapitre II s'explique par le fait que le coût des mesures de l'horloge était inconnu et était supposé être insignifiant. Mes premières évaluations des performances des schémas de communications ont été faites dans ce contexte, elles ne sont donc pas exploitables et m'ont amené à des conclusions erronées.

Le profilage historique de NEMO rend impossible une bonne évaluation des performances, particulièrement sur les schémas de communication. C'est une des raisons d'être du travail présenté au chapitre II qui permet la mise en place d'un cadre permettant l'obtention des résultats présentés ici.

3.2 Conditions d'exécution

Toutes les exécutions effectuées pour évaluer les performances utilisent la totalité des nœuds, c'est-à-dire que l'on utilise autant de processus MPI que de cœurs physiques disponibles sur les nœuds utilisées. Les nœuds sont donc utilisés dans leur totalité sans que d'autres utilisateurs du supercalculateur aient la possibilité d'utiliser les cœurs inactifs, ce qui pourrait perturber les tests, notamment à cause de l'utilisation de la mémoire. De plus, il n'y a ni dépeuplement ni *hyperthreading* dans nos expériences. L'utilisation de l'*hyperthreading* est absurde lorsque l'on considère les limitations de NEMO concernant la mémoire. Le dépeuplement pourrait être exploré et changerait certainement les performances du code, mais il serait surprenant que cela change la comparaison entre les schémas de communication, c'est-à-dire qu'une exécution avec et sans dépeuplement produirait des performances relatives semblables entre les schémas.

J'ai testé les performances des différentes versions du code dans de très multiples conditions, en variant le matériel et les librairies MPI utilisées. La variété de matériel comprend plusieurs supercalculateurs (Irene et Spartan), une grande variété de nœuds (AMD Rome et Intel Skylake

sur Irene et les nœuds AMD Epyc 7763 et 9654 et Intel Xeon Platinuim sur Spartan) et deux InterConnects (InfiniBand et BXI). Les librairies MPI testées sont les librairies inteMPI, openMPI et la bullopenMPI développée par Atos à partir de la librairie openMPI.

Les performances changent effectivement en fonction du matériel utilisé, certains nœuds de calculs exécutent par exemple des calculs plus rapidement que d'autres ou ont plus de cœurs par nœuds. Comme on utilise tous les cœurs et que l'on fixe la taille des sous-domaines MPI, utiliser des nœuds avec plus de cœurs revient à faire des simulations avec des domaines globaux plus grands que pour des simulations sur autant de nœuds mais avec moins de cœurs par nœuds. Les performances varient donc naturellement en fonction des nœuds.

De plus, le profil du temps d'exécution à chaque pas de temps a ses particularités sur chaque nœud de calcul et pour chaque librairie MPI. La valeur médiane peut évoluer curieusement au cours de l'exécution et la dispersion autour de cette valeur peut être plus ou moins élevée selon la situation. Lorsque l'on regarde les performances sur l'ensemble de l'exécution de NEMO, ces caractéristiques modifient même les valeurs moyennes et la dispersion. Comme ces profils se produisent pour tous les schémas de communications, on espère que notre méthodologie d'évaluation des performances permet de s'affranchir de ce problème. Une analyse complète de ces phénomènes pourrait être menée avec une bonne maîtrise du matériel et de la pile logicielle utilisée, mais cette analyse est hors du cadre du présent travail qui vise l'optimisation du code NEMO plutôt que l'étude précise du matériel et des librairies.

Les performances peuvent donc être évaluées dans un but de comparaison dans n'importe quelle condition tant que celle-ci permet une exécution acceptable de NEMO.

3.3 Résultats : configuration TSUNAMI

Les schémas de communication comparés ici sont : le schéma *EW-NS*, le schéma non bloquant (indiqué *Non bloquant*) utilisant l'instruction *MPI_Waitall*, le schéma avec recouvrement communication-calcul (indiqué *Recouvrement calcul*), les *neighborhood collectives* et les *persistent calls*. La gestion des buffers suit l'organisation présentée à la section 2.4.2. Les perfor-

mances des autres schémas développés dans le reste de ce chapitre ne sont pas évaluées. Le schéma RMA est encore en développement et la comparaison entre les schémas de la section 2.4 a déjà été effectuée et le schéma *MPI_Waitall* s'en est dégagé.

La configuration TSUNAMI est utilisée pour pouvoir comparer tous les schémas de communications entre eux, qu'ils soient uniquement utilisables dans le calcul du gradient de pression de surface ou dans l'ensemble du code.

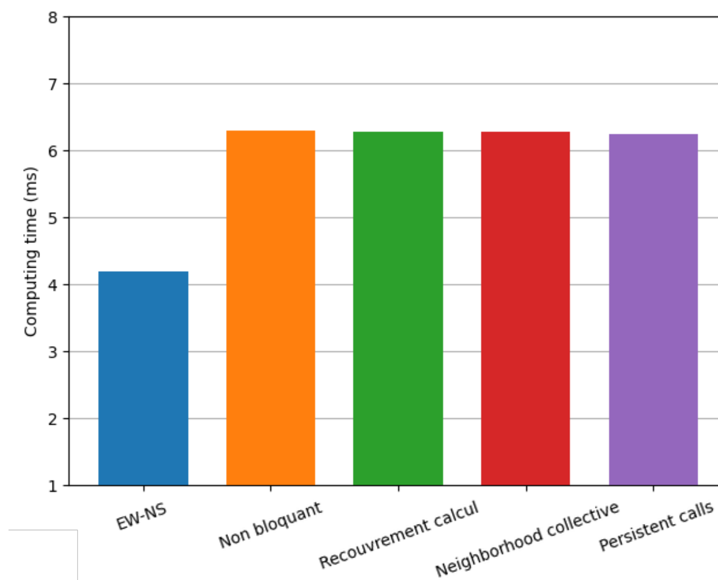


FIGURE IV.24 – Performances des schémas de communications en temps de calcul médian des pas de temps. Unité : millisecondes. Ce test a été réalisé sur 10 nœuds AMD Epyc 7763 du supercalculateur Spartan et utilise la librairie openMPI.

La figure IV.24 compare les performances des schémas de communication. Le schéma de communication bloquant *EW-NS* a de loin les meilleures performances pour cette expérience, les autres schémas de communication ayant des temps de calculs similaires et plus lents d'environ 45%. Le détail des performances peut varier en fonction des conditions d'exécution, mais la figure IV.25 prouve que dans tous les cas le schéma *EW-NS* est le plus rapide. Les valeurs des barres correspondent à la moyenne des valeurs obtenues sur une variété de conditions (dont une est affichée figure IV.24). Les schémas sont en moyenne $\approx 30\%$ plus lents que le schéma de référence et restent toujours au-dessus de la ligne des 100%, *EW-NS* est donc toujours le schéma de communication le plus rapide.

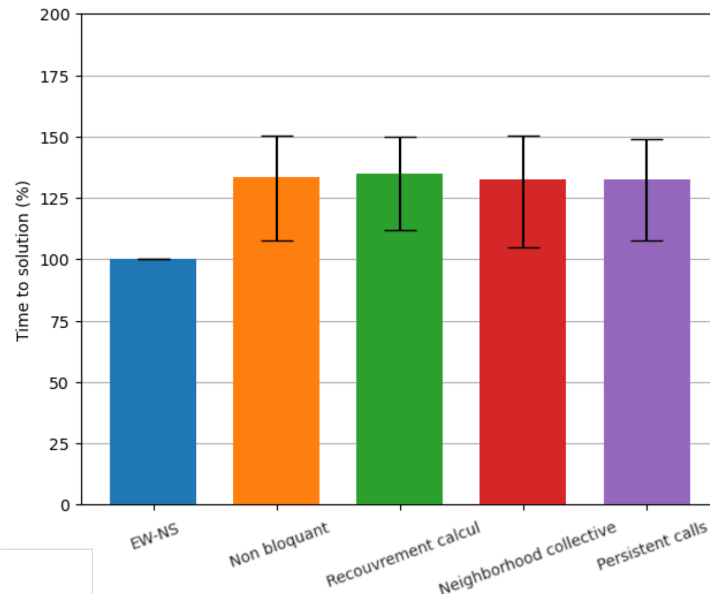


FIGURE IV.25 – Performances des schémas de communications en temps de calcul médian des pas de temps. Les schémas de communication sont exprimés en pourcentages du temps de calcul du schéma *EW-NS* qui est pris comme référence. Les barres correspondent aux performances de chaque schéma de communication moyenné sur un ensemble des conditions et les traits noirs représentent les valeurs minimales et maximales atteintes.

Les autres schémas montrent des temps de résolution très similaires et il est difficile de les comparer les uns aux autres, en raison de leur grande variabilité. En effet, les barres de minimum et maximum englobent largement les variations de la moyenne. Le schéma de communication avec recouvrement semble cependant avoir les pires performances, bien que cela soit uniquement d'une très faible marge. Ceci peut s'expliquer par le fait que les calculs qui servent au recouvrement ne peuvent pas être effectués de manière efficace. Par ailleurs, il est possible que le recouvrement n'ait pas réellement lieu. D'après certains experts MPI les instructions de communication non bloquantes sont généralement déclenchées que une fois que l'instruction bloquante correspondante est exécutée. En pratique, cela revient à dire que les communications ont lieu au cours de l'exécution de l'instruction *MPI_Waitall*. Le recouvrement est donc largement irréalisable pour ce type de schéma n'utilisant pas d'instructions *RMA*.

Les bonnes performances du schéma de communication *EW-NS* sont explicables premièrement du fait que seulement 4 voisins sont en jeux au lieu de 8 pour tous les autres schémas. Dans la configuration TSUNAMI, nous avons vu que les processus sont largement synchronisés. De plus, l'utilisation de la médiane masque les pas de temps avec un temps d'exécution plus

long certainement caractérisées par un plus fort asynchronisme. Les stratégies d'absorption de l'asynchronisme n'ont que peu d'intérêt dans ce cadre et le fait d'avoir 8 voisins au lieu de 4 devient une contrainte trop importante. Le schéma *EW-NS* occasionne une charge plus faible sur l'InterConnect et sur la carte réseau qui gère moins d'instructions.

3.4 Schémas de communication sans diagonales

Pour comparer plus équitablement les schémas de communications utilisant 8 voisins au schéma *EW-NS* qui n'en emploie que 4, il est possible de modifier les schémas pour qu'ils n'échangent plus avec les sous-domaines situés dans les directions diagonales.

Un certain nombre de communications dans le code NEMO pourraient être effectuées sans mises à jour des valeurs dans les coins des halos. Une telle expérience n'a donc pas qu'une valeur théorique. En effet, les schémas évaluent couramment les différences dans chaque dimension indépendamment des autres dimensions. Par exemple, sur la figure IV.26, les points en diagonal du point à mettre à jour ne sont pas utilisés, seulement des points dans les dimensions zonales ou méridionales. Finalement, dans ce cas, les valeurs des coins n'ont pas besoin d'être échangées. Dans le futur, ce type d'implémentation pourrait être utilisé dans les parties du code appropriées.

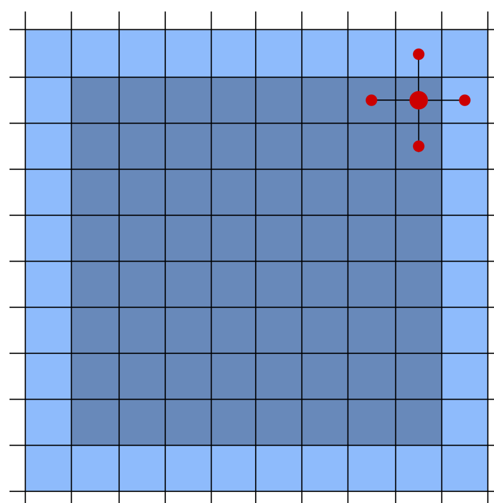


FIGURE IV.26 – Schéma décrivant certains calculs de NEMO. Les points illustrés en rouge sont ceux dont les valeurs sont utilisées dans un calcul pour mettre à jour le point central large en rouge.

Ce test est réalisable sur la configuration BENCH grâce à sa stabilité. Les valeurs dans les coins des halos ne sont pas mises à jour via une communication MPI. Les calculs effectués sont physiquement incorrect mais l'exécution, dans la configuration BENCH, se poursuit. Ceci permet de se faire une idée du gain potentiel de cette optimisation avant son déploiement effectif. La configuration TSUNAMI est elle aussi suffisamment stable pour réaliser ce test. On présente ici les résultats obtenus sur la configuration TSUNAMI pour estimer facilement les performances des schémas de communications directement reflétés par le temps de calcul de la simulation.

L'implémentation réalisée pour effectuer ce test consiste déjà en une première étape vers une version opérationnelle. On utilise pour cela un argument de type variable logique optionnel en entrée de la routine de communication qui désactive les échanges avec les voisins diagonaux. De plus, on ajoute dans la namelist un argument ln_4only qui enlève les échanges avec sous-domaines à la diagonale de toutes les communications. Il est alors possible soit de sélectionner, par des modifications minimes du code, quels appels à la routine de communication n'utiliseront pas d'échanges diagonaux, soit de supprimer tous les échanges diagonaux par simple modification de namelist. Le présent test se fonde sur cette dernière possibilité, mais une implémentation robuste s'appuie sur la modification du code au cas par cas, là où cette optimisation est effectivement réalisable.

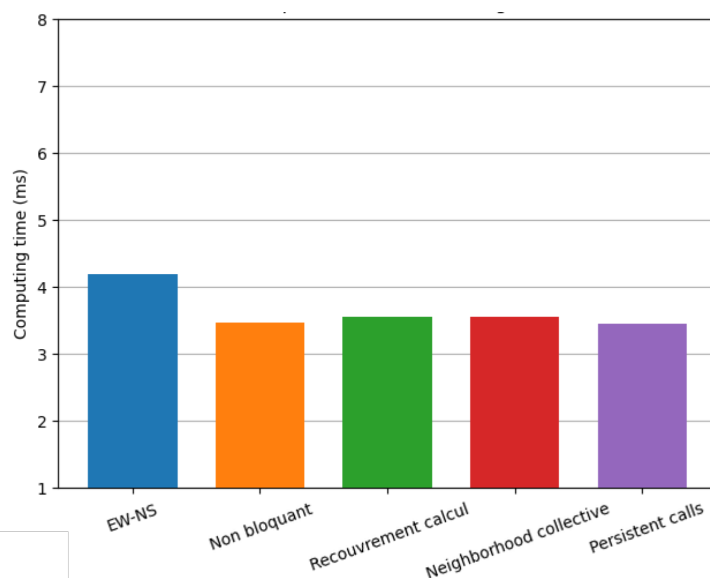


FIGURE IV.27 – Performances des schémas de communications sans diagonales. Temps de calcul médian des pas de temps en millisecondes. Ce test a été réalisé sur 10 nœuds AMD Epyc 7763 du supercalculateur Spartan et utilise la librairie openMPI.

La figure IV.27 montre les performances des schémas de communications sur 10 nœuds AMD Epyc 7763 avec la librairie openMPI. Lorsque les transferts avec les sous-domaines situés à la diagonale ne sont pas effectués, c'est le schéma *EW-NS* qui a les pires performances. Le temps de calcul de ce schéma est sensiblement le même qu'à la section précédente parce que ce schéma ne fait pas de transfert directement avec ses voisins diagonaux. Ce sont tous les autres schémas, impliquant à la base 8 voisins, qui ont été améliorés.

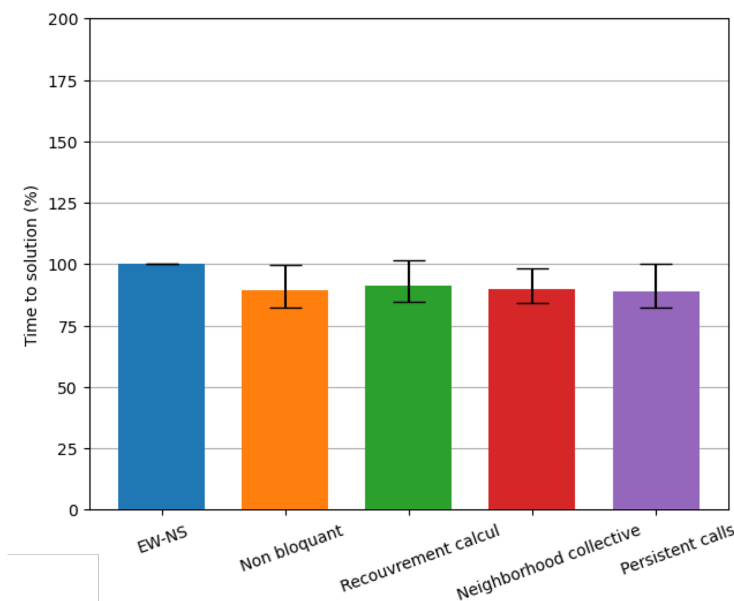


FIGURE IV.28 – Performances des schémas de communications sans transfert avec les sous-domaines diagonaux. Temps de calcul médian des pas de temps. Les schémas de communication sont exprimés en pourcentages du temps de calcul du schéma *EW-NS* qui est pris comme référence. Les barres correspondent aux performances de chaque schéma de communication moyennées sur un ensemble de conditions et les traits noirs représentent les valeurs minimales et maximales atteintes.

Cette tendance est confirmée par la figure IV.28, les pires temps de calculs des schémas sont au niveau du schéma *EW-NS*. En moyenne, les schémas ajoutés au code NEMO rendent la totalité de l'exécution 10% plus rapide que le schéma de référence et jusqu'à 18% dans certains cas. Pour une exécution sans transfert diagonal, nos développements améliorent fortement les performances de NEMO.

De plus, certaines communications n'ont besoin de mettre à jour que certains coins des halos, mais pas tous les 4. On peut ainsi envisager une implémentation des schémas de communications réalisant des mises à jour uniquement des coins désignés par un argument d'entrée optionnel.

En effet, les calculs réalisés dans cette configuration se limitent presque exclusivement au calcul du gradient de pression de surface. Certains de ces calculs sont analysés dans la section 2 de l'annexe A. Les seuls autres calculs du gradient de pression de surface faisant appel aux diagonales concernent le calcul du terme de vorticité qui peut être fait via l'utilisation de plusieurs schémas détaillés dans Madec et al. (2023). Ces schémas font tous appel à des valeurs des halos aux coins sud-est et nord-ouest, soit de tableaux de vitesses barotropes, soit de tableaux de flux barotropes qui en découlent. Ainsi, le calcul de gradient de pression de surface requière à chaque sous pas de temps deux communications échangeant avec 6 sous-domaines voisins au lieu de 8.

3.5 Résultats : configuration BENCH_OCE

Les mêmes schémas que pour la configuration TSUNAMI sont comparés sur la configuration BENCH_OCE. Rappelons que les schémas avec recouvrement et *persistent calls* sont opérationnels uniquement sur les communications du calcul du gradient de pression de surface, ils doivent donc être accompagnés d'un autre schéma pour les communications du reste du code. Ici, j'ai choisi le schéma non bloquant pour ces communications. Les performances des schémas avec recouvrement et *persistent calls* doivent donc être analysées avec cet élément en tête.

Tout comme pour les performances sur la configuration TSUNAMI, le schéma le plus efficace reste le schéma *EW-NS* (voir figure IV.29). Les autres schémas ont des résultats assez semblables les uns par rapport aux autres avec encore une fois des barres de minimum et de maximum largement plus importantes que les variations des valeurs moyennes entre schémas de communications.

L'écart entre le schéma *EW-NS* et les autres schémas est ici faible avec des valeurs moyennes différentes de seulement 1 ou 2% du temps d'exécution du pas de temps au lieu de $\approx 10\%$ pour la configuration TSUNAMI. D'une part cela s'explique par le fait que la configuration TSUNAMI passe plus de temps dans les communications que la configuration BENCH_OCE. L'impact des changements sur les communications est donc d'autant plus faible dans BENCH_OCE. D'autre

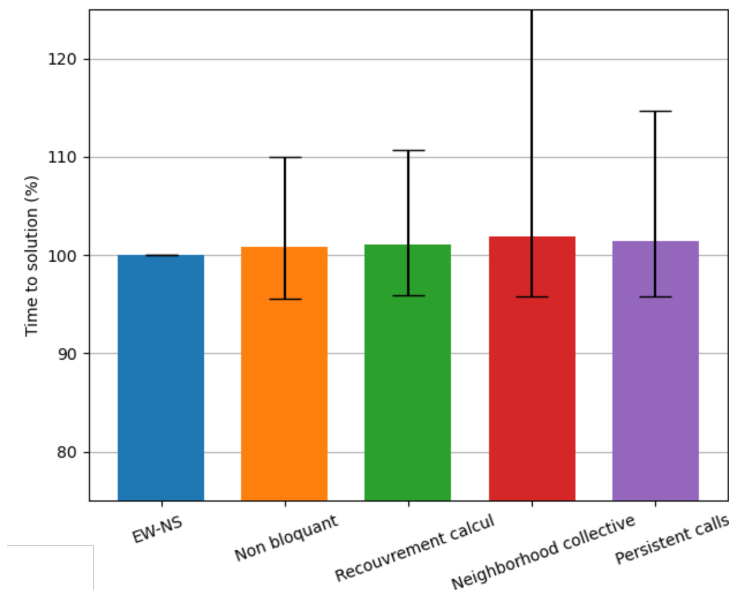


FIGURE IV.29 – Performances des schémas de communications avec transfert avec les sous-domaines diagonaux. Temps de calcul médian des pas de temps en millisecondes sur la configuration BENCH_OCE. La valeur maximale atteinte par le schéma *neighborhood collective* est de 132%.

part, contrairement à TSUNAMI, BENCH_OCE a un asynchronisme entre les processus de la simulation en dehors des calculs du gradient de pression de surface mais aussi dans les premières communications de ces calculs. Les stratégies d'absorption de l'asynchronisme prennent alors tout leur sens et, dans certains cas, les performances des schémas de communication dépassent celles du schéma de référence *EW-NS*.

Les performances sont aussi évaluées sans échange avec les sous-domaines situés en diagonal figure IV.30. Les schémas de communications développés sont plus rapides que le schéma de référence d'environ 3%. Sur la configuration TSUNAMI, ils étaient 10% meilleurs. Cet écart entre les configurations peut encore s'expliquer par le poids plus faible qu'ont les communications dans la configuration BENCH_OCE.

Les différences de performances entre les schémas de communication sont moins spectaculaires sur la configuration BENCH_OCE, mais montrent à nouveau que les schémas développés améliorent les performances par rapport au schéma de référence *EW-NS*. Cette configuration permet aussi de faire apparaître les gains des stratégies de réduction de l'impact de l'asynchronisme.

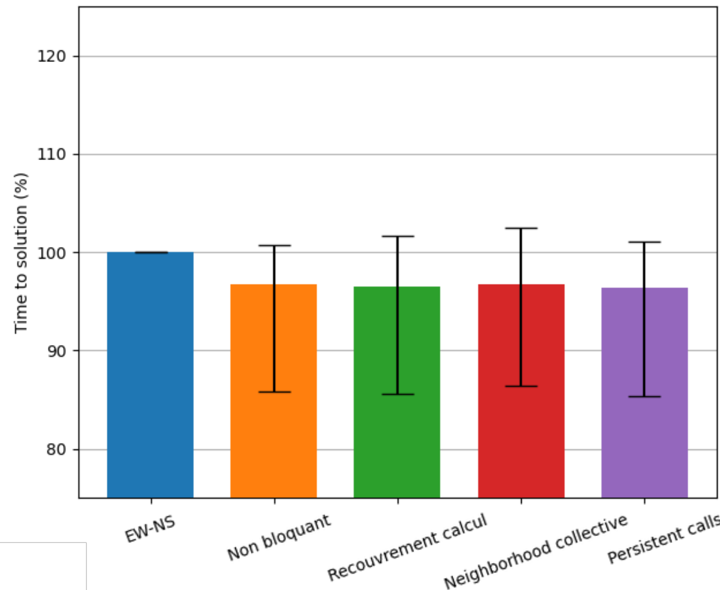


FIGURE IV.30 – Performances des schémas de communication sans transfert avec les sous-domaines diagonaux. Temps de calcul médian des pas de temps en millisecondes sur la configuration BENCH_OCE.

4 Conclusion et perspectives

Une grande variété de schémas de communication ainsi que de multiples optimisations de ces schémas ont été développées en suivant plusieurs impératifs : limiter le surcoût des instructions MPI et de limiter l'impact des phases bloquantes des schémas en permettant du recouvrement. Une première comparaison des performances permet de sélectionner un schéma de communication plus pertinent, le schéma non bloquant utilisant l'instruction *MPI_Waitall*. Ce schéma est ensuite comparé avec une sélection de schémas utilisant des fonctionnalités et des techniques MPI avancées, un schéma avec recouvrement communication-calcul, les *persistent calls*, les *neighborhood collective*s en prenant comme référence le schéma de référence de NEMO, le schéma *EW-NS*. Nos expériences montrent que *EW-NS* a les meilleures performances sauf lorsque les échanges avec les sous-domaines aux diagonales sont supprimés, auquel cas tous les autres schémas réduisent le temps d'exécution de l'ensemble de la configuration TSUNAMI de $\approx 10\%$ et $\approx 3\%$ pour BENCH_OCE. On notera que la variabilité des performances du code sur les machines utilisées, ne nous a pas permis de mesurer si l'utilisation de fonctionnalités MPI avancées apportent effectivement un gain en performances ou non.

Rappelons que, outre la variabilité des performances, la méthode utilisée pour mesurer les performances via un recours à la médiane filtre les pas de temps anormalement lents, or les schémas développés dans cette thèse sont censés aider à limiter l'impact de ces ralentissements. Les moments où certains schémas sont censés être les plus performants sont ainsi laissés de côté. Malheureusement, prendre en compte les pas de temps ralentis dans la mesure des performances revient en définitive à comparer l'occurrence et l'ampleur de ralentissements aléatoires entre plusieurs exécutions.

Pour bénéficier des performances apportées par les nouveaux schémas de communication, il faudrait activer dès que possible des options, déjà existantes dans NEMO, qui désactivent les échanges avec les diagonales lors de d'appel aux communications. De plus, par défaut dans NEMO, les communications sont toujours effectuées dans les quatre directions alors qu'il serait possible, dans certains cas, de ne les faire que dans 2 directions ce qui pourrait aider à gagner en performance.

Des pistes d'optimisation des schémas de communication développés ont été dégagées. Certaines ont déjà été exposées, pour les communications RMA les échanges utilisant des RMA notifiées peuvent être mis en place, pour les *persistent calls* plusieurs versions peuvent être envisagées par exemple avec recouvrement des communications par des routines de calcul. Mais bien d'autres demanderaient à être examinées, quelques-unes sont évoquées ci-dessous.

Au vu des performances du schéma de communication *EW-NS*, les *persistent calls* pourraient être utilisés pour implémenter le schéma *EW-NS* pour limiter le coût des instructions. Cela peut être réalisé en utilisant plusieurs appels à *MPI_Startall* et *MPI_Waitall* pour respecter les deux phases du schéma. Une implémentation avancée consisterait à modifier la librairie MPI pour pouvoir exécuter les principales étapes du schéma de communication *EW-NS* avec un unique appel à une instruction de *persistent calls*.

D'autre part, la librairie MPI fournit des moyens d'optimiser le schéma des *neighborhood collectives* via des arguments d'entrée lors de la phase d'initialisation. D'après la documentation en ligne openMPI (18.2.85. MPI_Dist_graph_create_adjacent Open MPI 5.0.x documentation,

s. d.) ajouter en argument de *MPI_Dist_graph_create_adjacent* des poids qui reflètent la quantité de données transférée vers chaque voisin permet d'améliorer les communications. Dans NEMO, on pourrait par exemple renseigner la taille de la zone de halo à échanger. Un argument de type *MPI_Info* peut aussi être utilisé, bien qu'il soit difficile de comprendre les tenants et aboutissants de son utilisation comme le souligne la documentation MPI. Enfin, des variables d'environnement peuvent être explorées pour changer le détail de l'exécution des communications par exemple la variable *I_MPI_COLL_EXTERNAL* qui permet l'utilisation d'autres librairies notamment la librairie hcoll.

Des fonctionnalités peuvent être ajoutées au schéma de communication avec recouvrement. Une routine de calcul ne modifiant pas le tableau objet de la communication peut être utilisée pour recouvrir du temps de communication sans avoir une gestion spécifique des différentes zones du tableau. La quantité de calcul serait ainsi augmentée par cette utilisation conjointe de différentes routines de calcul et permettrait ainsi un meilleur recouvrement.

Un plus grand éventail de schémas de communication peut être implémenté. Dans la continuité du travail présenté à la section 2.4, un schéma aurait pu être construit autour de l'instruction *MPI_Waitsome* qui semble prometteur et équilibré entre le coût des appels et les possibilités de recouvrements. De plus, l'impact de l'ordre de déclenchement des instructions non-bloquantes d'envoi et de réception devrait être étudié.

Signalons ici que notre analyse des schémas de communication est limitée par le choix de la configuration utilisée. Premièrement, en utilisant des conditions de périodicité bi-périodiques, nous avons laissé de côté les communications spécifiques utilisées pour la gestion du pôle nord dans les configurations globale. Deuxièmement, des communications spécifiques à la gestion des icebergs sont aussi écartées du fait que nous n'avons pas activé cette fonctionnalité dans le code. Ces échanges particuliers mériteraient des études spécifiques de leurs performances à la lumière des conclusions de ce chapitre et doivent faire l'objet d'une réécriture à partir de 2024.

Tout compte fait, notre exploration des schémas de communication permet en effet une réduction du temps de restitution des simulations. Cependant cela est principalement du fait d'améliorations basiques des routines comme la gestion des buffers et la fusion d'instructions lorsque cela est possible. La variété de schémas développés donne finalement des performances intéressantes seulement dans certains cas (4 voisins) demandant légèrement plus de développements, mais sont surtout presque indiscernables les uns des autres. Cette légère amélioration est possiblement le fait d'une meilleure gestion de l'asynchronisme sans que je puisse le confirmer de manière définitive, la synchronisation des horloges des différents nœuds (voir annexe B) n'est pas suffisamment précise pour permettre une telle évaluation.

Par ailleurs, la difficulté de la mesure des performances est ici limitante, non seulement pour une évaluation précise des gains en performances, mais aussi pour déterminer quel est le schéma de communication le plus efficace pour le mettre à disposition des utilisateurs de NEMO. Enfin, le développement de schémas est complexifié par les difficultés liées à la mesure des performances en elle-même. Sans mesures précises, le ou la développeur n'a pas d'informations claires lui permettant de conclure si les modifications apportées sont des optimisations ou non ou si elles n'ont simplement pas d'impact.

CHAPITRE

V

Optimisation de simulations avec zooms emboîtés

1 Le logiciel AGRIF

1.1 AGRIF et la montée en résolution

1.1.1 Le problème de la montée en résolution

L'adoption d'une haute résolution dans NEMO peut répondre à plusieurs nécessités. La plus évidente est l'amélioration de la précision de la simulation, une discrétisation spatiale plus fine permettant une meilleure représentation des gradients spatiaux. Les simulations à haute résolution permettent aussi de mieux résoudre des processus à petite échelle qui influencent le comportement à échelle plus large. Par exemple, les courants de bord ouest comme celui du Gulf Stream sont bien mieux représentés à haute résolution et l'impact d'une telle amélioration fait sentir ses effets à l'échelle globale, comme le montre Talandier et al. (2014). La montée en résolution permet aussi une meilleure représentation de la bathymétrie, particulièrement celle des détroits indonésiens et ainsi des courants qui les traversent (Van Sebille et al., 2014) ou de la circulation au travers de l'arc des Antilles (Jouanno et al., 2008).

Cette montée en résolution pose néanmoins un problème vis-à-vis du coût de calcul. Celui-ci est directement lié à la quantité d'opérations arithmétiques ainsi qu'à la mémoire utilisée et donc au nombre de points de grille de la simulation.

Les caractéristiques du problème font que la montée en résolution s'effectue de manière différente suivant les dimensions considérées (horizontales, verticale et temporelle). La discrétisation des équations de Navier-Stokes dans les modèles d'océan dont NEMO tire en effet partie de la géométrie du problème : la profondeur des océans est négligeable par rapport au rayon de la terre (approximation de couche mince ou thin-shell approximation (Madec et al., 2023)). Le problème que l'on étudie est donc traité différemment suivant l'horizontale ou la verticale. Ainsi, si l'on augmente la résolution horizontale, on n'augmente pas nécessairement la résolution ver-

ticale. En revanche, les deux dimensions horizontales ayant des comportements similaires, une montée en résolution dans l'une de ces dimensions s'accompagnera systématiquement d'une montée en résolution dans l'autre.

En ce qui concerne la dimension temporelle, la condition Courant–Friedrichs–Levy (CFL) énonce que le pas de temps ne doit pas dépasser une certaine valeur proportionnelle au raffinement spatial et définie telle que (en une dimension) :

$$v\frac{\Delta t}{\Delta x} \leq C_{max}$$

Où v est la magnitude de la vitesse de n'importe quel phénomène résolu et C_{max} une valeur maximale dépendant des schémas. Δt (resp. Δx) est l'intervalle de discrétisation temporel (resp. spatial). Une réduction de Δx doit donc s'accompagner d'une réduction de Δt pour que l'inégalité reste vérifiée. La condition CFL a pour conséquence qu'une augmentation de la résolution spatiale horizontale doit s'accompagner d'une augmentation équivalente de la résolution temporelle.

Des deux derniers points découle le fait qu'une augmentation de la résolution spatiale (horizontale) d'un facteur X se traduit par une augmentation du nombre de points de grille d'un facteur X^2 et de la quantité de calcul d'un facteur X^3. Une montée en résolution est donc extrêmement coûteuse. Pour garder un temps de calcul et une empreinte mémoire par nœud raisonnable, il faut donc augmenter, dans certains cas considérablement, le nombre de nœuds alloués au calcul de la simulation. Cette solution atteint rapidement ses limites, même en omettant le surcoût en argent et en énergie lié à l'utilisation de ressources informatiques supplémentaires. La limite de scalabilité du code fait qu'on ne peut pas utiliser efficacement toujours plus de ressources. Ce problème est d'autant plus critique que la parallélisation se faisant uniquement sur le domaine spatial, pour que chaque sous-domaine ait la même quantité de calcul à effectuer et maintenir en temps de restitution comparable, il faudrait, en théorie, utiliser X^3 plus de sous-domaines. En pratique, la montée en résolution a pour effet soit de limiter les simulations en nombre d'années simulées soit de causer des simulations très longues et très coûteuses en temps de calcul.

1.1.2 AGRIF : une solution pour la montée en résolution

Une manière d'aborder le problème de la montée en résolution est d'observer que dans un certain nombre de situations, la montée en résolution peut être limitée à une zone spatialement restreinte du domaine de la simulation. Une solution est alors de modifier la configuration de la simulation pour avoir une zone restreinte à haute résolution qui recouvre la région d'intérêt et qui soit englobée dans une zone à plus basse résolution qui s'étende sur la totalité du domaine de simulation. Ce type de configuration permet de limiter le besoin de ressources de calculs en limitant la taille de la région de haute résolution.

AGRIF (*Adaptive Grid Refinement In Fortran*) est une librairie qui permet un tel raffinement de maillage dans un modèle multidimensionnel à différences finies écrit en langage Fortran. Cette librairie est utilisée dans les modèles océaniques pour offrir la possibilité d'exécuter des zooms (aussi appelé grille enfant) de différentes résolutions emboîtés au sein d'une grille à plus basse résolution (appelée grille parent). Ces zooms interagissent doublement avec la grille parent dans laquelle ils sont emboîtés : la grille parent prescrit les frontières latérales de la grille enfant qui en retour renvoie ses données, calculées à plus haute résolution, vers la grille parent. Ainsi, le zoom n'est pas indépendant de la circulation océanique à plus grande échelle et inversement cette circulation est impactée par les phénomènes de plus fine échelle résolus sur le zoom.

Un exemple est fourni par la figure V.1 issue de la documentation en ligne NEMO (AGRIF_DEMO.jpg on Users/ModelInterfacing/AGRIF – Attachment – NEMO, s. d.). où le zoom délimité en vert a un facteur de raffinement de 4, c'est-à-dire que sa résolution est 4 fois plus importante que celle de sa grille parent. Il est aussi possible de définir plusieurs niveaux de zoom, emboîtés les uns dans les autres : le zoom délimité en jaune a un facteur de raffinement de 3 par rapport au zoom vert et donc un raffinement de 12 par rapport à la grille à plus basse résolution. De la même façon, il est possible de définir plusieurs zooms au même niveau, comme le zoom délimité en rouge et celui délimité en vert (qui ne partagent pas nécessairement le même facteur de raffinement).

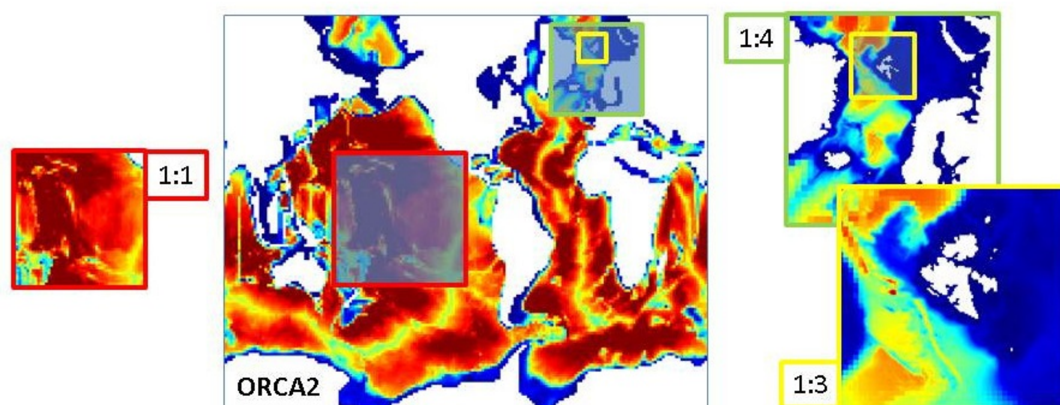


FIGURE V.1 – Exemple d'une simulation contenant des zooms de différentes résolutions.

Il est aussi possible d'utiliser les fonctionnalités d'AGRIF pour obtenir une résolution fine sur une région étendue, par exemple sur une longue ligne de côte en juxtaposant de nombreux zooms de même niveau (Petton et al., 2023).

Outre l'aspect amélioration des performances par rapport à l'utilisation d'une simulation haute résolution sur l'ensemble du domaine, il peut être intéressant d'utiliser AGRIF pour quantifier les impacts d'un phénomène haute résolution sur une zone donnée sur une échelle plus large. (Biastoch et al., 2008) a ainsi montré l'importance du courant des Aiguilles dans la variabilité décennale de la circulation atlantique en utilisant NEMO et AGRIF pour un zoom sur le courant des Aiguilles. AGRIF utilisé de la sorte permet de ne pas prendre en compte des phénomènes de haute résolution sur des zones qui ne sont pas pertinentes pour une étude, de cette façon AGRIF cible une région dans laquelle un mécanisme particulier est visé.

Les nombreuses qualités et applications d'AGRIF que nous venons de résumer, ont décidé le consortium NEMO à choisir cet outil pour atteindre localement une plus haute résolution en maintenant des coûts de calcul réduit. L'utilisation d'AGRIF permet d'améliorer la représentation des processus océaniques cruciaux pour le climat ou des utilisations opérationnelles, en particulier pour l'écoulement de fond des eaux denses, les mers épicontinentales et les zones côtières à géométrie complexe.

AGRIF est également utilisé par plusieurs autres modèles océaniques de pointe, notamment CROCO (www.croco-ocean.org) et le modèle MARS3D de l'IFREMER. AGRIF rend aussi possible des raffinements sur la dimension verticale des zooms mais cette fonctionnalité ne sera pas étudiée ici.

AGRIF est par sa nature même un moyen de traiter des problématiques de coût de calcul. Cependant, on le verra dans la suite, son fonctionnement a des aspects qui freinent les performances. Ce chapitre vise à exposer le travail effectué sur la quantification et l'amélioration des performances de simulations utilisant AGRIF.

J'ai effectué un profilage complet d'AGRIF qui met en évidence deux verrous aux performances, l'un étant lié aux communications et l'autre à un algorithme d'interpolation. Une optimisation sera alors détaillée pour chacun de ces verrous.

1.2 Fonctionnement général de AGRIF

D'après la documentation AGRIF en ligne (General introduction : AGRIF software AGRIF 1.10.0 documentation, s. d.) la librairie se compose de deux parties :

— Un convertisseur de code "source to source" ("CONV") qui transforme un code fonctionnant sur une grille unique en un code capable de fonctionner sur n'importe quelle grille (la grille principale ainsi que les éventuels zooms) via l'utilisation massive de pointeurs qui permettent aux variables physiques (température, courants...) de pointer au choix sur des données appartenant à une grille ou à une autre.

— Une librairie qui fournit les outils (*functions/subroutines*) nécessaires à la définition des interactions que l'on souhaite définir entre les différents zooms. Ces interactions sont détaillées dans les sections suivantes 1.2.1 et 1.2.2.

En complément des *functions* et *subroutines* "outils" disponibles dans la librairie AGRIF, des *functions* et *subroutines* liées à l'utilisation d'AGRIF sont présentes dans NEMO (src/NST) pour spécifier les interactions entre grilles que l'on souhaite réaliser dans une simulation NEMO utilisant des zooms. Comme on le verra par la suite, les interactions entre les grilles sont des points

clé pour les performances d'une simulation utilisant AGRIF. Toutes ces modifications et ajouts au code NEMO sont potentiellement autant de points bloquants pour les performances et devront faire l'objet d'un profilage adapté.

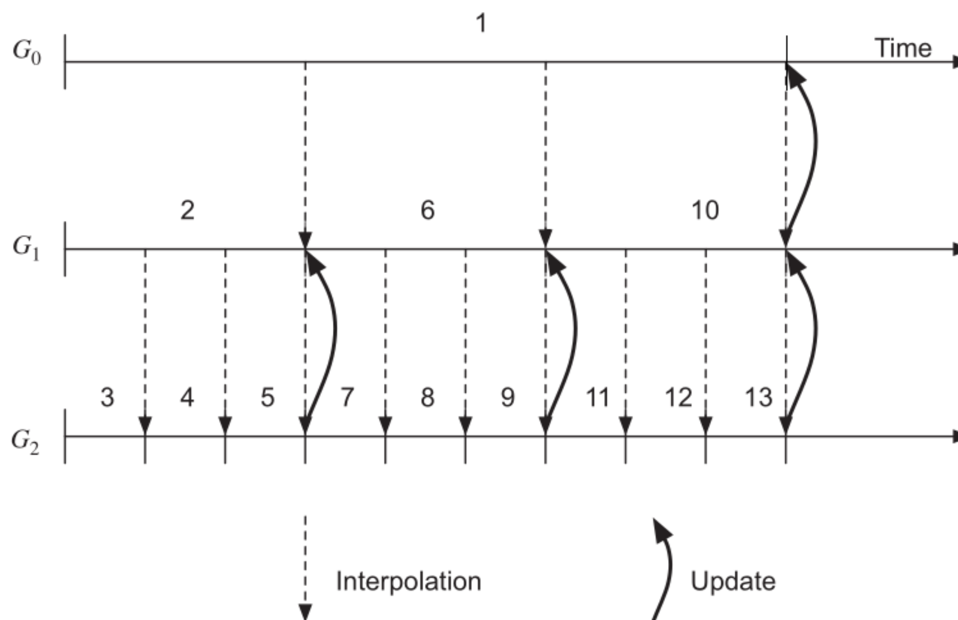


FIGURE V.2 – Progression d'une simulation sur trois grilles (G_0, G_1, G_2) imbriquées et présentant un facteur de raffinement temporel $\rho_t = 3$. Les nombres sur les axes temporels indiquent l'ordre de résolution des pas de temps.

Le code ainsi réécrit amène l'exécutable à résoudre les équations sur les différentes grilles dans un ordre prédéfini ainsi que de réaliser les interactions entre ces grilles. La figure V.2, issue de (Debreu et al., 2008), montre un exemple de l'ordre de résolution pour une simulation avec deux zooms imbriqués. La grille parent, G0, calcule son pas de temps (numéroté 1, sur la figure V.2). Puis le modèle "repart en arrière" pour calculer le pas de temps numéroté 2 de la première grille enfant, G1. À la fin du pas de temps 2, le modèle "repart de nouveau en arrière" pour calculer les pas de temps 3, 4, 5 de la grille G2. Les conditions limites utilisées aux frontières latérales de G2 pour calculer ces pas de temps proviennent de l'interpolation linéaire dans le temps des valeurs calculées sur G1 en début et fin de pas de temps 2, c'est la phase qu'AGRIF appelle "interpolation". De même les pas de temps (7, 8, 9 de G12 ; 11, 12, 13 de G2 ; 2, 6 et 10 de G1) utilisent les valeurs calculées sur (G1 ; G1 ; G0) en début et fin de pas de temps (6 ; 10 ; 1). À la fin du pas de temps 5 de G2 qui coïncide avec le pas de temps 2 de G1, les valeurs

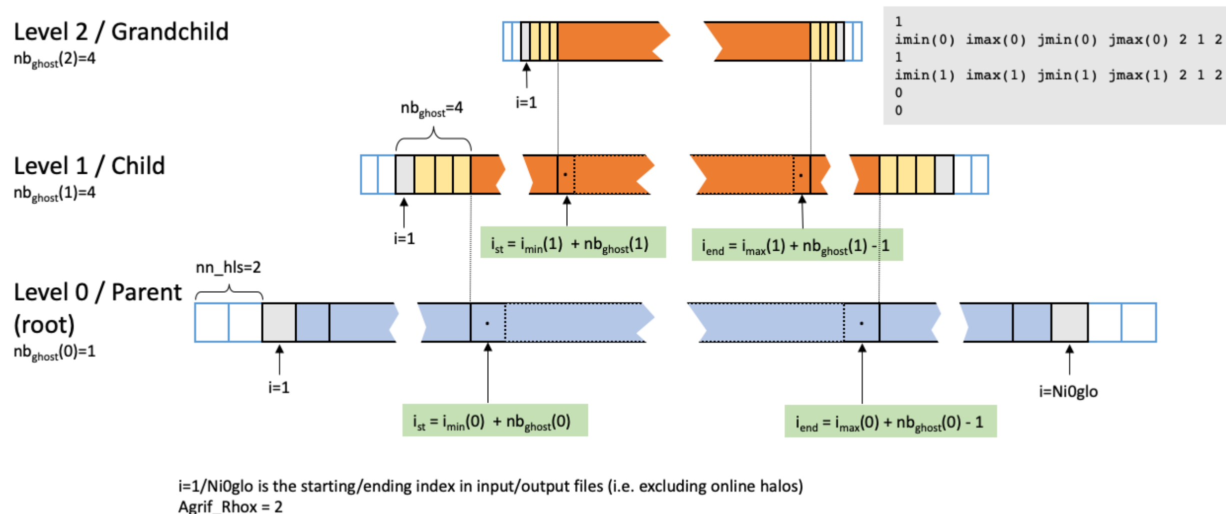
de G1 sont mises à jour avec les valeurs de G2 calculées à plus haute résolution, c'est la phase qu'AGRIF appelle "*update*". La grille G1 peut alors effectuer son pas de temps suivant, numéroté 6. L'*update* s'effectue à chaque fois que la fin du pas de temps de la grille enfant coïncide avec celui du parent : les pas de temps 5, 9, 12 de G2 coïncident avec les pas de temps 2, 6, 10 de G1 et le pas de 10 de G1 avec le pas de temps 1 de G0. Soit ρ_t le facteur de raffinement temporel, il y a donc ρ_t fois plus de phases d'interpolation que d'*update*.

1.2.1 Interpolation

Le premier aspect de l'interaction entre grilles est l'interpolation, ou, plus précisément, l'interpolation utilisant des valeurs de la grille parent pour prescrire le résultat sur la grille enfant.

AGRIF met à disposition des outils pour interpoler sur différentes parties de la grille enfant, cela peut être sur la totalité du domaine via *Agrif_Init_Variable()*, même si, comme son nom l'indique, cette routine sera presque exclusivement utilisée pour l'initialisation des valeurs physiques sur la grille enfant. Plus souvent, l'interpolation ne vise que les bords du domaine de la grille enfant. Le but de cette interpolation étant alors de redéfinir les conditions limites latérales du domaine enfant. La zone d'interpolation est fixée par les développeurs du code fortran sur lequel AGRIF sera appliqué. Dans le cas de NEMO et hors initialisation des variables, cette zone est limitée à une zone au bord du domaine de la grille enfant large de quelques points de grille. Sur la figure V.3, issue de la documentation NEMO en ligne (Embedded zooms - NEMO release-4.2.1 documentation, 2023), cela correspond aux points jaunes et gris des grilles enfants c'est-à-dire les ghost cells sur lesquels on applique les conditions limites latérales. Le reste du domaine enfant, en orange sur la figure V.3, est complètement libre de toute correction liée à la grille parent. Nous l'appellerons "cœur" du domaine enfant.

Dans NEMO, l'interpolation sur la totalité du domaine ne se produit qu'à l'initialisation de la grille enfant pour imposer des conditions initiales, donc une seule fois dans la simulation. Elle n'a que très peu d'impact sur les performances et elle ne sera pas analysée par la suite.



CASE 1a: Level 0 (root) grid has closed boundaries

FIGURE V.3 – Positionnement de trois grilles emboîtées comportant une seule dimension, chaque case représente un point de grille. Les traits pointillés entre les grilles indiquent les définitions du début et fin des zooms comme indiqué dans le fichier *AGRIF_FixedGrids.in*. Les cases jaunes et grises des grilles enfants sont les ghost cells et les cases orange (resp. bleu) sont les cases constituant le cœur des domaines enfants (resp. parent). L'ensemble des cases colorées forment le domaine interne des grilles et les cases transparentes sont les halos.

L'interpolation sur le bord survient quant à elle à chaque pas de temps de la grille enfant et a donc potentiellement un impact significatif en termes de performances sur la totalité de la simulation. Nos analyses se concentreront donc sur les interpolations effectuées à chaque pas de temps.

L'interpolation sur le bord ayant lieu à chaque pas de temps de la grille enfant, la première étape de l'interpolation est donc une interpolation temporelle. En effet, les valeurs ne sont pas définies sur la grille parent entre les pas de temps de la grille parent. Soit Δt_0 l'intervalle temporel sur la grille parent et Δt_1 celui sur la grille enfant, les valeurs ne sont pas définies à $t = n_0 \Delta t_0 + n_1 \Delta t_1$ avec $n_1 \in [\![1; \rho_t - 1]\!]$. Les valeurs issues de la grille parent utilisées dans la suite sont issues de l'interpolation $x_{interp}(n_0 \Delta t_0 + n_1 \Delta t_1) = (1 - \frac{n_1}{\rho_t}) x(n_0 \Delta t_0) + \frac{n_1}{\rho_t} x((n_0 + 1) \Delta t_0)$ où $x(t)$ est une valeur définie sur la grille parent aux pas de temps de la grille parent et $x_{interp}(t)$ le résultat de l'interpolation temporelle linéaire. Ce calcul est possible car les pas de temps n_0 et $n_0 + 1$ de la grille parent ont déjà été calculés. L'interpolation temporelle linéaire est effectuée de manière automatique par la librairie AGRIF.

L'interpolation spatiale est la deuxième étape de l'interpolation de la grille parent vers la grille enfant. C'est aux développeurs NEMO de spécifier, pour chaque variable, sur quels points de la grille enfant il faut effectuer l'interpolation (via la fonction *Agrif_Set_Bc()*) et avec quel schéma cette interpolation doit être réalisée (via la fonction *Agrif_Set_BcInterp()*). Dans les faits, cette zone est située sur le bord du domaine de la grille enfant, pour les tableaux 3D, la zone d'interpolation est identique sur tous les niveaux verticaux.

1.2.2 *Update*

La deuxième interaction entre grilles a lieu dans l'autre sens, de la grille enfant vers la grille parent. L'*update* se déroule après ρ_t pas de temps de la grille enfant, c'est-à-dire une fois que la grille enfant a rattrapé la grille parent sur l'axe temporel. De même que pour l'interpolation, la zone d'*update* (via *Agrif_Update_Variable()*) et le schéma d'*update* (via *Agrif_Set_UpdateType()*) sont à spécifier par les développeurs NEMO. En pratique, la zone d'*update* recouvre au moins une partie du cœur du domaine enfant. Dans NEMO, l'*update* se fait sur la totalité du cœur du domaine enfant.

Maintenant que les parties du code susceptible de constituer des freins aux performances sont listées, réalisons un profilage complet du code pour estimer et quantifier à quel point ces parties sont effectivement bloquantes.

2 Profilage de AGRIF

2.1 Construction de configurations AGRIF pour le profilage

Les performances d'une simulation NEMO avec zoom AGRIF dépendent fortement de la configuration exécutée. Par exemple, elles dépendent du nombre de zoom et du nombre de points de grille de chacun de ces zooms.

On choisit d'analyser les performances sur une configuration facile d'utilisation et construite de manière à avoir une charge de calcul la plus similaire possible sur les différents processus MPI. Ce dernier point est important car un trop mauvais équilibrage de charge oblige les processus MPI avec moins de charge à attendre les processus avec plus de charge lors des communications MPI. Ceci entraîne une baisse des performances due aux particularités de la configuration, on choisit de laisser de côté ce type de verrou aux performances trop dépendantes de la configuration.

Les entrées/sorties de fichiers de données sont supprimées pour simplifier le plus possible l'analyse et limiter les sources de perturbations des performances. Les entrées/sorties sont ainsi limitées au minimum, elles ont pour la plupart lieu durant l'initialisation (avec la lecture de fichier de namelist) et concernent des quantités de données très modestes durant le reste de la simulation. Les performances du système de fichier n'interviennent donc pas ici. Cette configuration est créée à partir de BENCH_OCE, présentée dans le chapitre II section 2, et comporte des caractéristiques similaires. Seul le cœur dynamique de NEMO est résolu (pas de forçages avec des données extérieures à NEMO, pas de glace de mer, pas de bio-géochimie), les sous-domaines de la grille parent sont choisis de taille identique. La configuration est compilée avec les appels à la librairie AGRIF et on y inclut un unique zoom pour faciliter l'analyse.

Les ressources de calculs dédiées à la grille parent sont les mêmes que celles dédiées à la grille enfant. Chaque processus MPI calcule une partie de la grille parent, puis une partie de la grille enfant. Le nombre de processus MPI dédié à chaque grille est donc identique. La taille du zoom est choisie de manière à ce que les sous-domaines de la grille enfant aient des tailles identiques entre eux mais aussi avec ceux de la grille parent de manière à accroître encore l'équilibrage de charge de calcul et ainsi simplifier l'analyse des performances. Avec un facteur de raffinement spatial de $\rho_x = 3$ le zoom doit recouvrir environ $\frac{1}{9}$ de la surface du domaine de la grille parent pour satisfaire ces conditions. La limite du cœur du domaine de la grille enfant est définie par l'utilisateur dans le fichier *AGRIF_FixedGrids.in* (correspondant aux pointillés dans la figure V.3 ou la ligne épaisse rouge dans la figure V.4). Il faut noter que le domaine de la grille

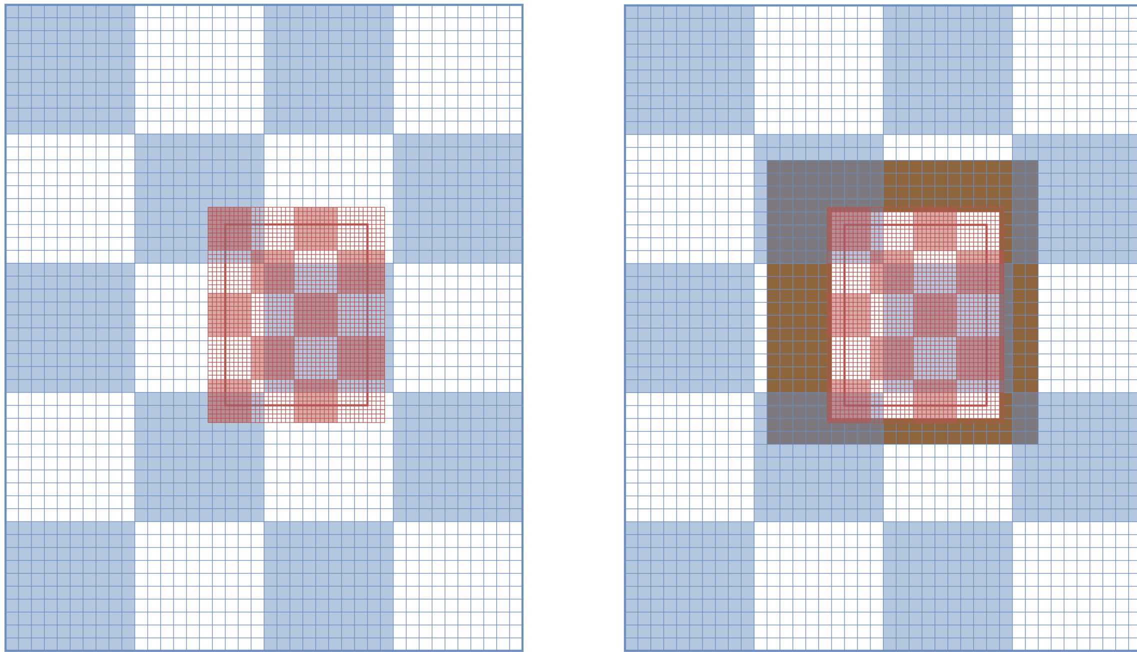


FIGURE V.4 – Schéma d'un niveau vertical pour deux configurations BENCH_AGRIF contenant un zoom. Les lignes et rectangles sont bleus pour la grille parent et rouges pour la grille enfant. Les lignes épaisses délimitent les domaines comme définis par l'utilisateur, pour la grille enfant cela correspond au cœur du domaine étendu par les ghost cells. Les traits fins forment les grilles parent et enfant. Les rectangles rouges ou bleus dessinent un damier correspondant au découpage MPI en sous-domaines. La configuration présentée sur la gauche est sans point terre et celle de droite comporte des continents indiqués en brun.

enfant est étendu d'un certain nombre de points, les ghost cells pour l'interpolation (en jaune et gris sur la figure) dont un point de terre toujours présent dans les simulations NEMO (en gris) et les halos MPI (en blanc). Il est nécessaire de prendre ces particularités en compte dans la définition de la taille du domaine pour arriver à des tailles de sous-domaines MPI spécifiques. Le facteur de raffinement temporel est fixé à $\rho_t = \rho_x = 3$ par cohérence avec le raffinement spatial. C'est un choix habituel pour conserver la validité de la condition CFL.

Sur la figure V.4 à gauche est présenté en schéma un exemple de configuration BENCH_AGRIF. La grille parent (resp. enfant) est matérialisée par les traits fins bleus (resp. rouges). Le découpage en sous-domaines MPI de la grille parent (resp. enfant) est représenté par les damiers de rectangles bleus (resp. rouges). Dans les deux cas, c'est un découpage en $4 \times 5 = 20$ sous-domaines et chaque sous-domaine est composé de 10 par 10 points de grille. Un seul niveau vertical est représenté, tous les niveaux verticaux étant identiques. Le cœur du domaine de la grille enfant est délimité par un trait rouge épais. La grille enfant dépasse du cœur du

domaine du fait de la présence de ghost cells et de points terre sur le bord. De même que pour BENCH_OCE, la grille parent est utilisée avec des conditions bi-périodiques pour assurer une répartition de charge identique pour chaque processus MPI lors du calcul et des communications sur la grille parent. En effet, cela permet d'assurer que chaque processus a un nombre identique de voisins lors des communications MPI et qu'aucune condition aux limites particulières ne doit être calculée sur certain sous-domaines seulement, du moins, pour le domaine parent.

Comme nous le verrons par la suite, la présence de continents dans le domaine de la simulation et donc de points terre sur la grille peut fortement impacter la phase d'interpolation et affecter les performances. Afin de quantifier cet aspect spécifique de l'interpolation, nous avons développé une seconde configuration comportant des continents. La figure V.4 à droite est un schéma représentant un niveau vertical de cette configuration, chaque niveau vertical étant là aussi identique. Les points terre sont disposés pour avoir un impact sur le déroulement de l'algorithme, ils n'ont aucun réalisme et ne sont pas pertinents du point de vue des résultats physiques. La zone d'interpolation de la grille enfant est entourée sur tous les côtés d'une bande de terre plus ou moins large afin de tester différents cas et d'amplifier les problèmes soulevés par la présence de continents comme on peut les rencontrer dans des configurations plus réalistes.

Un tel positionnement des points terre peut sembler exacerber artificiellement certaines limitations de l'algorithme d'interpolation, mais, dans les configurations réalistes la bathymétrie est telle que le nombre de points terrestres augmente avec la profondeur. À partir d'un certain niveau vertical les points terre peuvent être encore plus présents et avoir un plus grand impact sur les performances que dans notre configuration.

La configuration ne comportant pas de points terre est très facile d'utilisation, elle peut être adaptée pour différents nombre de sous-domaines simplement en modifiant les fichiers de namelist ainsi que *AGRIF_FixedGrids.in* pour modifier l'étendue du zoom en cohérence avec la taille de la simulation. La configuration comportant des points terre est légèrement plus difficile d'utilisation, un fichier d'entrée est nécessaire pour définir la position des points terre. Celle-ci doit être changée pour rester pertinente si la taille de la grille fille évolue, le fichier d'entrée corres-

	BENCH_OCE	BENCH_AGRIF sans zoom
Memory stalls	19.80 %	19.82 %
Vectorisation	48.57 %	48.51 %
Temps de calcul du pas de temps médian	$1.28\ 10^{-2}$ s	$1.29\ 10^{-2}$ s

TABLE V.1 – Pourcentage de memory stalls par rapport à la totalité des accès mémoires et pourcentage d'opération à virgule flottante vectorisée. Test réalisé avec le profiler aps sur une configuration de taille de sous-domaine 10 par 10 sur un nœud de 40 cœurs Cascadelake.

pondant doit donc être construit pour chaque configuration testée. Comme les performances ont été testées sur différents types de nœuds comportant des nombres de cœurs différents, le fichier d'entrée définissant les points terre doit être construit pour chacun de ces cas.

Nous avons donc mis en place deux configurations avec zoom AGRIF pour évaluer les performances dans deux situations différentes. Ces configurations sont définies de manière à simplifier l'analyse, notamment sur l'équilibrage de charge entre processus MPI.

2.2 Estimation des performances d'AGRIF

2.2.1 Impact du convertisseur de AGRIF

La partie 1.2 a mis en évidence que la première modification que AGRIF apporte à NEMO est une réécriture du code via un convertisseur de code "CONV". Il convient donc de tester, dans un premier temps, si la réécriture de NEMO par AGRIF modifie ses performances. Les points importants à regarder sont les performances mémoires et la vectorisation. On veut en effet vérifier que la réécriture à l'aide de pointeurs n'impacte pas la gestion de la mémoire et que la réécriture ne gêne pas le compilateur qui gère la vectorisation.

Pour cela, le code NEMO est comparé entre une configuration BENCH_OCE et une configuration BENCH_AGRIF, où la réécriture du code par AGRIF a donc eu lieu, mais où aucun zoom n'est défini dans le fichier *AGRIF_FixedGrids.in*. La seconde configuration fera donc tourner une simulation identique à la première et j'ai vérifié qu'elles produisent notamment des résultats physiques identiques au bit près, la seule différence étant que la seconde configuration fait tourner un code ayant subi une réécriture via l'outil CONV de AGRIF.

Ce test est réalisé sur un environnement *intel* et un compilateur *intel-mpi*. Les options de compilation sont celles fournies par la distribution NEMO adaptée au supercalculateur utilisé avec notamment l'option *-O3* qui fixe le niveau global des optimisations effectuées par le compilateur au niveau le plus haut et l'option *-xCORE-AVX512* qui active l'utilisation de la vectorisation utilisant les registres vectoriels disponibles sur les nœuds Intel CascadeLake 6248 utilisés. L'outil de profilage utilisé est aps, un outil fourni par *intel* dont le bon fonctionnement n'est garanti que sur les environnements *intel*. Ce test n'a donc été réalisé que sur un environnement *intel* du fait de la disponibilité de l'outil aps, les résultats seront peut-être différents en utilisant des compilateurs différents.

Les résultats de cette expérience sont mis en évidence dans le tableau V.1, les valeurs très proches des memory stalls indiquent que la réécriture de NEMO avec l'utilisation intensive de pointeurs n'impacte pas les performances liées à la mémoire. Le pourcentage de memory stalls indique le pourcentage de cycle d'horloge subissant un temps d'arrêt du fait d'une incapacité du système à fournir une donnée mémoire pour la complétion d'une instruction. Cette mesure indique comment les problèmes de mémoire affectent les performances, les valeurs presque identiques entre les deux expériences montrent que les problèmes liés à la gestion de la mémoire sont identiques.

De même, le pourcentage d'instructions vectorisées est le même, le compilateur n'est donc pas gêné par la réécriture du code. Le temps de calcul du pas de temps médian montre que les deux configurations sont tout autant performantes l'une que l'autre. Un test identique avec des sous-domaines MPI de 50 par 50 donne aussi des résultats inchangés entre les deux configurations.

Cette expérience nous permet de conclure que la réécriture de NEMO par AGRIF n'impacte pas la gestion de la mémoire, ni la vectorisation ou les performances générales du code, ce qui constitue un très bon résultat, condition nécessaire pour poursuivre l'exploration des performances d'AGRIF.

2.2.2 Performances d'AGRIF

Estimons maintenant les performances d'AGRIF sur la configuration BENCH_AGRIF sans points terre développée et présentée dans la section 2.1.

Comme détaillé précédemment, la configuration développée compte autant de points de grille sur la grille parent que sur la grille enfant et présente un facteur de raffinement temporel $\rho_t = 3$. Un pas de temps de calcul de la grille enfant représente une charge de calcul identique à un pas de temps de la grille parent (si les interactions entre grilles sont omises). La simulation de la configuration avec un zoom a au moins quatre fois plus de travail à effectuer que la simulation de la configuration sans zoom (uniquement sur la grille parent) : une charge de travail pour les calculs sur la grille parent, plus 3 charges de travail pour les calculs sur trois pas de temps sur la grille enfant. En pratique, il faut aussi ajouter à ces quatre charges de travail la charge de travail de l'*update* et de l'interpolation qui assurent les échanges entre les grilles.

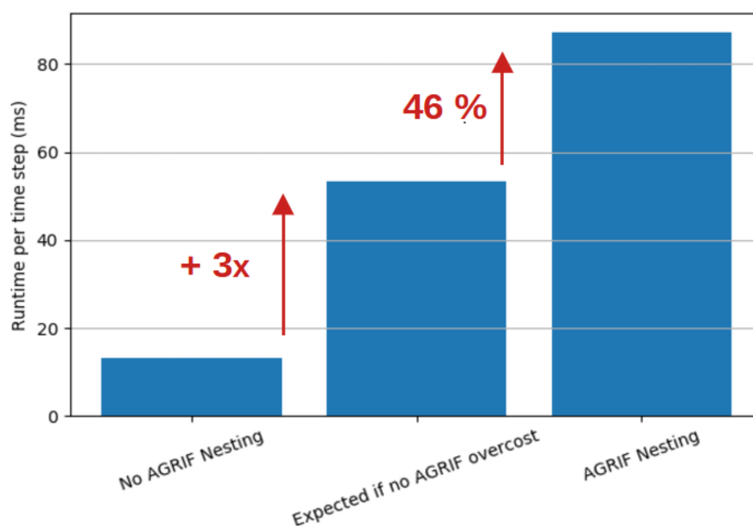


FIGURE V.5 – Temps de calcul moyen d'un pas de temps de simulation de gauche à droite pour une configuration sans zoom AGRIF, performances théoriques idéales pour une configuration avec zoom AGRIF, performances pour une configurations avec zoom AGRIF.

La figure V.5 montre que les performances de la simulation avec un zoom AGRIF dépassent de 46% les performances attendues dans un cas idéalisé, soit un coût de calcul multiplié par presque 6 au lieu d'un facteur théorique de 4 sans interactions entre grilles. Cette différence est due, entre autres, aux phases d'*update* et d'interpolation ainsi qu'à la charge en mémoire

supplémentaire venant du fait de traiter deux grilles à la place d'une par un même processus MPI. En plus de l'impact de la pure quantité de données, le fait de changer régulièrement entre la grille parent et la grille enfant implique que, au moment du changement, les caches mémoires contiennent des données de l'autre grille. Les données de la grille actuelle doivent donc être chargées en mémoire, ce qui est coûteux en temps. C'est cet écart entre performances idéales et performances effectives que j'essaye de réduire dans la suite de ce travail.

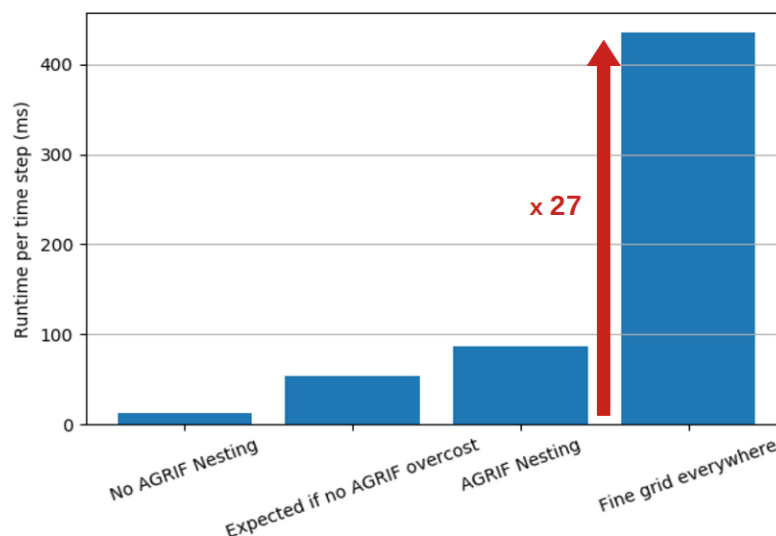


FIGURE V.6 – Temps de calcul moyen d'un pas de temps de simulation de gauche à droite pour une configuration sans zoom AGRIF, performances théoriques idéales pour une configuration avec zoom AGRIF, performances pour une configurations avec zoom AGRIF et performances théoriques pour une simulation sans zoom AGRIF et avec une résolution fine sur la totalité du domaine.

46% de temps de calcul en plus que l'idéal peut paraître beaucoup (surtout pour une configuration plutôt avantageuse : sous-domaines MPI de faible taille, pas de points terre), mais cela doit être mis en perspective avec les performances attendues sur une configuration sans zoom mais avec une grille aussi fine que la grille enfant sur l'ensemble du domaine. Avec des intervalles temporels et spatiaux 3 fois plus petits, une telle configuration a donc 9 fois plus de points de grille que la configuration grille grossière. La charge de calcul est 27 fois plus importante lorsque l'on prend une grille fine sur l'ensemble du domaine plutôt que d'utiliser des zooms via AGRIF (voir figure V.6). L'intérêt de l'utilisation d'AGRIF est alors flagrant même dans l'état actuel des performances d'AGRIF.

2.3 Profilage d'AGRIF

Pour quantifier et déterminer les causes de ce surcoût et voir s'il est possible de le réduire, on procède à l'identification des goulots d'étranglement introduits par AGRIF. On commence par une analyse des routines via l'outil de profilage gprof (chapitre II section 3.2.1).

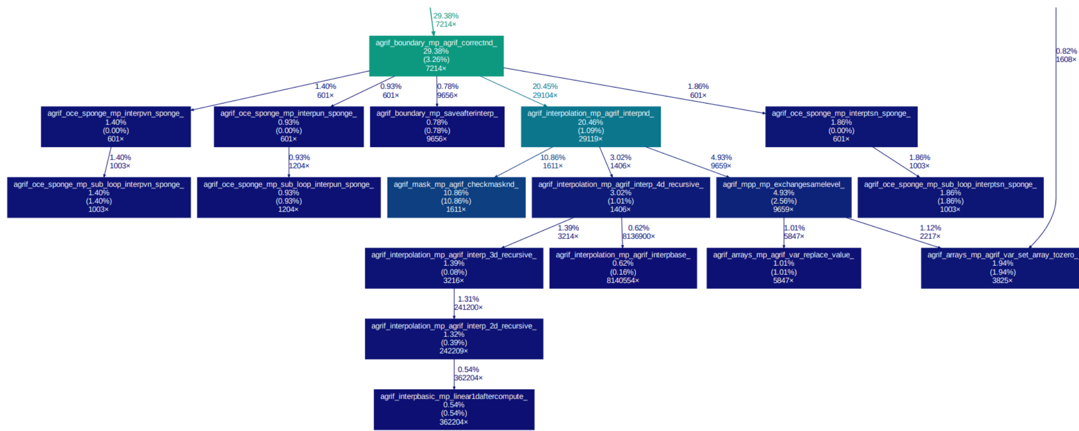


FIGURE V.7 – Profilage via gprof de la configuration BENCH_AGRIF sans continents.

La figure V.7 montre le profilage, effectué grâce au logiciel gprof, des principales routines d'AGRIF dans la configuration BENCH_AGRIF sans continents. Chaque case représente une routine, les cases sont coloriées suivant le pourcentage du temps de calcul de cette routine, aussi indiqué sur la case, en clair pour les temps élevés et en sombre pour les temps faibles.

Les routines AGRIF prennent une portion considérable du temps de la simulation ($\approx$ 30%) dont 20% dans la routine d'interpolation *Agrif_InterpnD* (indiqué *agrif_interpolation_mp_agrif_interpnd_* sur la figure V.7 étant donné que AGRIF renomme les routines), à son tour *Agrif_CheckMasknD* (ou *agrif_mask_mp_agrif_checkmasknd_*) concentre la moitié de ce temps et *ExchangeSameLevel* (*agrif_mpp_mp_agrif_exchangesamelevel_*) un quart. Cette analyse nous a donc permis d'identifier les routines les plus coûteuses, l'interpolation et, plus spécifiquement, *Agrif_CheckMasknD*, qui constituent des goulots d'étranglement.

TOP 5 MPI Functions	%
Init	14.65
Recv	3.84
Finalize	0.70
Isend	0.50
Waitall	0.50

TOP 5 MPI Functions	%
Recv	10.29
Allreduce	8.86
Init	6.00
Waitall	1.56
Isend	0.81

FIGURE V.8 – Profilage MPI via aps de la configuration BENCH_AGRIF sans zoom (à gauche) et avec un zoom (à droite) sur 1 nœud et sur des sous-domaines de 10 par 10. Les pourcentages sont donnés en pourcentage du temps de simulation total.

Les temps de calcul introduits par les routines AGRIF expliquent le surcoût d'une simulation AGRIF constaté figure V.5 (sauf environ 2%). Les problèmes de mémoire ne semblent pas jouer dans cette simulation, sûrement du fait de la faible taille des sous-domaines, la quantité de données gérée n'étant tout simplement pas assez grande pour que cela entre en compte. Des simulations avec des sous-domaines MPI plus grands pourraient faire apparaître ces limitations.

Regarder les performances d'un autre point de vue peut permettre de mettre en évidence d'autres goulots d'étranglement qui ne sont pas forcément visible au niveau des routines. Un profilage utilisant aps permet d'analyser la partie MPI de la simulation.

Le temps passé dans les fonctions MPI points à points (MPI Recv, Isend et Waitall) est plus important en présence d'un zoom AGRIF, passant d'un total de 5% à 12.5% du temps de simulation total. De plus, une fonction MPI (MPI_Allreduce) accapare presque 9% du temps de simulation total. Cette routine MPI réalise des communications globales, c'est-à-dire des communications dans lesquelles au moins un des processus MPI doit échanger, directement ou indirectement, des informations avec tous les autres processus, ce qui est un type de communication particulièrement coûteux. Cette communication n'apparaît qu'en présence d'un zoom AGRIF. Le zoom AGRIF augmente donc la charge des communications non globales et introduit un appel à une communication globale.

Cette analyse étant réalisée sur un seul nœud MPI, l'impact de ces goulots d'étranglement ira en s'accroissant sur des configurations utilisant plus de nœuds MPI. Cela est d'autant plus vrai pour les communications globales qui deviennent rapidement très coûteuses lorsque le nombre de nœud augmente.

On a ainsi mis en évidence 2 principaux goulots d'étranglement distincts, les routines d'interpolations et les communications MPI globales.

3 Optimisation des communications MPI dans AGRIF

3.1 Origine des communications globales dans AGRIF

Identifions dans un premier temps la raison de la présence d'une communication MPI globale dans les simulations AGRIF.

Les communications MPI globales sont produites par un appel à la routine fournie par la librairie MPI *MPI_Allreduce*. Cette routine prend en argument une opération à effectuer sur les données en entrée, cette opération combine les entrées fournies par les différents processus MPI pour obtenir des données de même dimension que les données d'entrée d'un processus, c'est l'étape "réduction" de la routine. La seconde étape consiste à communiquer à chaque processus MPI impliqué le résultat obtenu lors de la partie réduction. *MPI_Allreduce* nécessite donc des communications globales pour les deux parties de la routine.

Les communications MPI deviennent plus coûteuses lorsque le nombre de processus impliqué augmente. Une communication globale est composée d'un ensemble de communications d'un processus à un autre. Les communications globales comme *MPI_Allreduce* demandent un nombre croissant de communications d'un processus à un autre lorsque le nombre de processus augmente. Les performances de *MPI_Allreduce* avec un nombre de processus croissant sont donc limitées à la fois par le coût croissant d'une communication d'un processus à un autre et par le nombre croissant de communications nécessaires au bon fonctionnement de la routine. Des algorithmes existent pour limiter le plus possible le nombre de communications demandées par la routine et améliorer ses performances générales (Bienz et al. (2019)), mais la tendance persiste. Le temps d'exécution suit une progression logarithmique soit en fonction du nombre de nœuds et le nombre de processus par nœud soit en fonction du nombre de processus total. Seule la base du logarithme change selon l'algorithme utilisé et au moins l'un des termes étant en base

2. Pour une simulation sur $32 = 2^5$ nœuds on peut s'attendre à un temps d'exécution au minimum 5 fois plus grand de la routine *MPI_Allreduce*. Cette étape, qui limite déjà les performances dans notre cas, devient essentielle quand le nombre de nœuds utilisés augmente.

L'appel à *MPI_Allreduce* se trouve dans la routine *Agrif_get_var_global_bounds*. Cette routine a pour but d'obtenir l'étendue du tableau, c'est-à-dire les indices de début et de fin du tableau scindé en sous-domaines MPI et distribué sur des processus MPI. Ces indices sont ensuite utilisés dans les routines d'*update* (*Agrif_UpdatenD*) ou d'interpolation (*Agrif_InterpnD*) pour localiser avec précision la grille enfant et ses frontières par rapport à la grille parent. La routine *Agrif_get_var_global_bounds* requière un appel à une routine de communication pour obtenir ces indices car seuls les indices de début et de fin du tableau défini localement sur chacun des processus MPI est disponible.

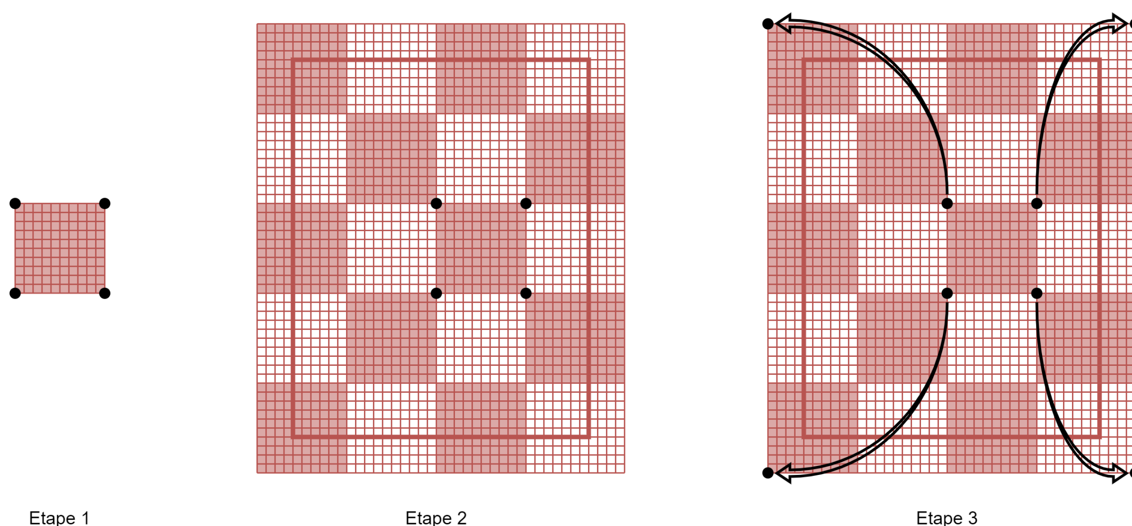


FIGURE V.9 – Schéma de la routine *Agrif_get_var_global_bounds*. Les traits fins rouges forment la grille enfant. Les rectangles rouges dessinent un damier correspondant au découpage MPI en sous-domaines. Les ronds noirs mettent en avant les indices manipulés par la routine.

Le fonctionnement de *Agrif_get_var_global_bounds* est illustré sur la figure V.9. La routine récupère les indices délimitant la variable de la grille enfant sur le processus MPI (étape 1). Rappelons ici que les tableaux Fortran ne commencent pas nécessairement à l'indice 1, mais peuvent commencer à n'importe quel indice choisi par la personne développant le code, d'où la nécessité de cette première étape. Ces indices sont donc directement disponibles. Une fois récupérés, ils sont convertis en indices communs à tous les processus MPI de la grille enfant

(étape 2), c'est-à-dire des indices localisant un point dans l'ensemble de la grille enfant comprenant tous les sous-domaines MPI. Une communication MPI *MPI_Allreduce* est alors utilisée sur les indices pour calculer le minimum (pour les frontières inférieures) ou le maximum (pour les frontières supérieures) (étape 3). Les limites de la grille enfant sont ainsi recueillies.

Étant donné que tous les indices manipulés dans la routine *Agrif_get_var_global_bounds* ne correspondent à priori qu'à la seule variable sur laquelle les indices sont calculés, l'appel à *Agrif_get_var_global_bounds* et donc à la communication globale *MPI_Allreduce* a ainsi lieu pour chaque interpolation ou *update*, c'est-à-dire pour une multitude de variables. La fréquence des appels à la routine *MPI_Allreduce* ainsi que le comportement intrinsèque des communications globales expliquent le temps de calcul important mesuré dans mon profilage de AGRIF.

3.2 Optimisation : suppression de la communication globale

AGRIF offre la possibilité d'avoir des zooms qui changent de position et de forme au cours des pas de temps. Certaines configurations comportent en effet des positions de zoom variant au cours du temps, par exemple pour suivre des phénomènes physiques, Blayo et Debreu (1999) utilise une configuration comportant un zoom AGRIF suivant un tourbillon. Les indices récupérés par la routine *Agrif_get_var_global_bounds* peuvent ainsi changer dans ce cas de figure. Une forte majorité des simulations AGRIF ont pour caractéristique d'avoir des positions de zoom immobiles et donc des indices localisant les frontières des tableaux constants dans le temps. Cette propriété rend possible l'optimisation présentée ici.

L'optimisation de cette partie se base sur l'idée de déplacer la routine de calcul des indices de délimitation du tableau *Agrif_get_var_global_bounds* à l'initialisation et stocker les valeurs de ces indices pour les utiliser ensuite directement durant la simulation. Ainsi, l'appel à *Agrif_get_var_global_bounds* et la communication globale associée à chaque pas de temps sont évités.

On choisit la structure de données *list_update* et pour stocker les indices pour une utilisation dans la routine d'*update Agrif_UpdatenD*, cette structure est déjà existante dans le code. Pareillement, on utilise *list_interp* pour une utilisation dans *Agrif_InterpnD*. Les variables contenues dans cette structure sont toutes liées au découpage MPI en sous-domaines et à la position du zoom, elles sont donc constantes au cours des pas de temps au même titre que les indices de délimitation des tableaux pour des zooms fixes.

L'ajout des indices aux structures *list_update* et *list_interp* permet ainsi de ne pas les recalculer à chaque pas de temps, sauf, lorsque la position du zoom change pour les simulations l'autorisant. Dans ce cas uniquement les indices ont besoin d'être recalculés. La communication globale *MPI_Allreduce* accompagnant la routine de calcul *Agrif_get_var_global_bounds* est ainsi supprimée pour l'immense majorité des configurations.

Le code produit par notre optimisation est aussi valable pour des simulations où la position du zoom varie, l'ensemble des variables contenues dans *list_interp* seront recalculées à chaque modification de la position du zoom et l'appel à *MPI_Allreduce* n'aura lieu que dans ce cas. Les résultats physiques sont alors préservés mais l'amélioration des performances apportée par l'optimisation ne sont pas disponibles.

Quantifier les apports aux performances apportées nécessite cependant une étude sur la scalabilité d'AGRIF, en effet c'est sur un grand nombre de cœurs que les différences seront les plus criantes. Vérifier que le code produit est aussi valable pour des configurations avec un zoom AGRIF se déplaçant au cours du temps demande de tester le code sur au moins une configuration de ce genre. Un certain travail est encore requis pour valider et évaluer précisément cette optimisation.

Les gains apportés sont indubitablement considérables au vu des performances des communications globales qui prennent déjà 9% du temps de simulation sur un seul nœud. Ce coût est totalement éliminé par cette optimisation.

4 Optimisation des routines d'interpolation

4.1 Fonctionnement de l'algorithme d'interpolation

L'objectif de l'interpolation est de définir les conditions limites latérales de la grille enfant en interpolant sur les bords de la grille enfant les valeurs issues de la grille parent. On a ainsi une interaction de la grille parent vers la grille enfant. La physique à plus grande échelle de l'écoulement peut ainsi influer sur la physique à plus petite échelle résolue dans le zoom AGRIF. Le problème auquel vient répondre l'algorithme d'interpolation est que les grilles parent et enfant n'ont pas la même résolution spatiale, un point de la grille enfant peut être localisé là où aucun point de la grille parent ne se trouve. L'interpolation identifie les points de la grille parent localisés aux alentours d'un point de la grille enfant sur lequel doit s'appliquer l'interpolation. L'ensemble de ces points de la grille parent, appelé stencil, est ensuite combiné pour imposer le résultat sur la grille enfant.

La figure V.10 montre un exemple d'une interpolation bilinéaire. Le stencil est composé des quatre points de la grille parent (notés "'O") les plus proches d'un point de la grille enfant donné (noté "x"). Les valeurs prises en ces points sont pondérées par des poids dépendant linéairement de leur écart de position avec le point de la grille enfant puis sommées, obtenant ainsi la valeur à imposer au tableau de la grille enfant en ce point. Le schéma d'interpolation, c'est-à-dire le stencil et la manière de combiner ces points, peut varier, plusieurs schémas étant mis à disposition par AGRIF. Certains de ces schémas utilisent des stencils plus étendus que le schéma bilinéaire.

L'interpolation rencontre un problème avec la présence de continents. En effet, si un point du stencil est défini comme un point terre, la valeur que prend le tableau en ce point ne peut pas être utilisée. Le schéma d'interpolation doit donc s'accommoder de la présence de valeurs manquantes.

Deux solutions peuvent répondre à ce problème :

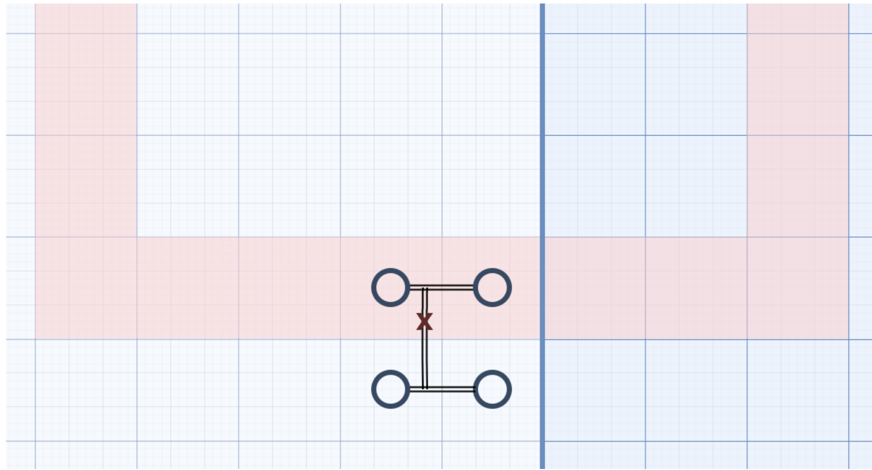


FIGURE V.10 – Schéma explicatif de l'interpolation réalisé par *Agrif_InterpnD*. La zone de la grille enfant où l'interpolation va être appliquée est coloriée en rouge. Ici est représentée une interpolation bilinéaire, 4 points océans de la grille parent (O bleu) sont interpolés vers un point de la grille enfant (x rouge).

— Calculer l'interpolation en utilisant la valeur 0 pour les points terre et renormaliser les poids utilisés pour les points océan restants. Cette méthode a pour inconvénient de modifier le schéma d'interpolation et les propriétés mathématiques qui lui sont associées.

— Garder le schéma d'interpolation inchangé mais "remplir" les points terre avec des valeurs "acceptables". Pour chaque point terre on utilise la moyenne des valeurs prises sur les points océan les plus proches de ce point terre. Même si le schéma d'interpolation est en principe inchangé on a introduit des points dont les valeurs sont définies artificiellement. C'est en substance un point remplacé par une moyenne d'autres points, en substituant cette moyenne dans la formule du stencil, il est clair que le schéma d'interpolation n'est plus le même et ne conserve à priori plus ses propriétés mathématiques.

Dans AGRIF, la deuxième solution a été choisie. Il faut donc identifier les éventuels points terre du stencil et remplacer leur valeur avant d'effectuer l'interpolation elle-même.

Les principales routines utilisées pour l'interpolation peuvent être trouvées dans les sources de la librairie AGRIF : *ext/AGRIF/AGRIF_FILES/modinterp.F90* pour *Agrif_InterpnD*, la routine principale d'interpolation, et *ext/AGRIF/AGRIF_FILES/modmask.F90* pour *Agrif_CheckMasknD*, la routine de recherche et de remplacement des points terre.

La structure de la routine d'interpolation *Agrif_InterpnD* est donnée par la figure V.11 et suit les étapes suivantes :

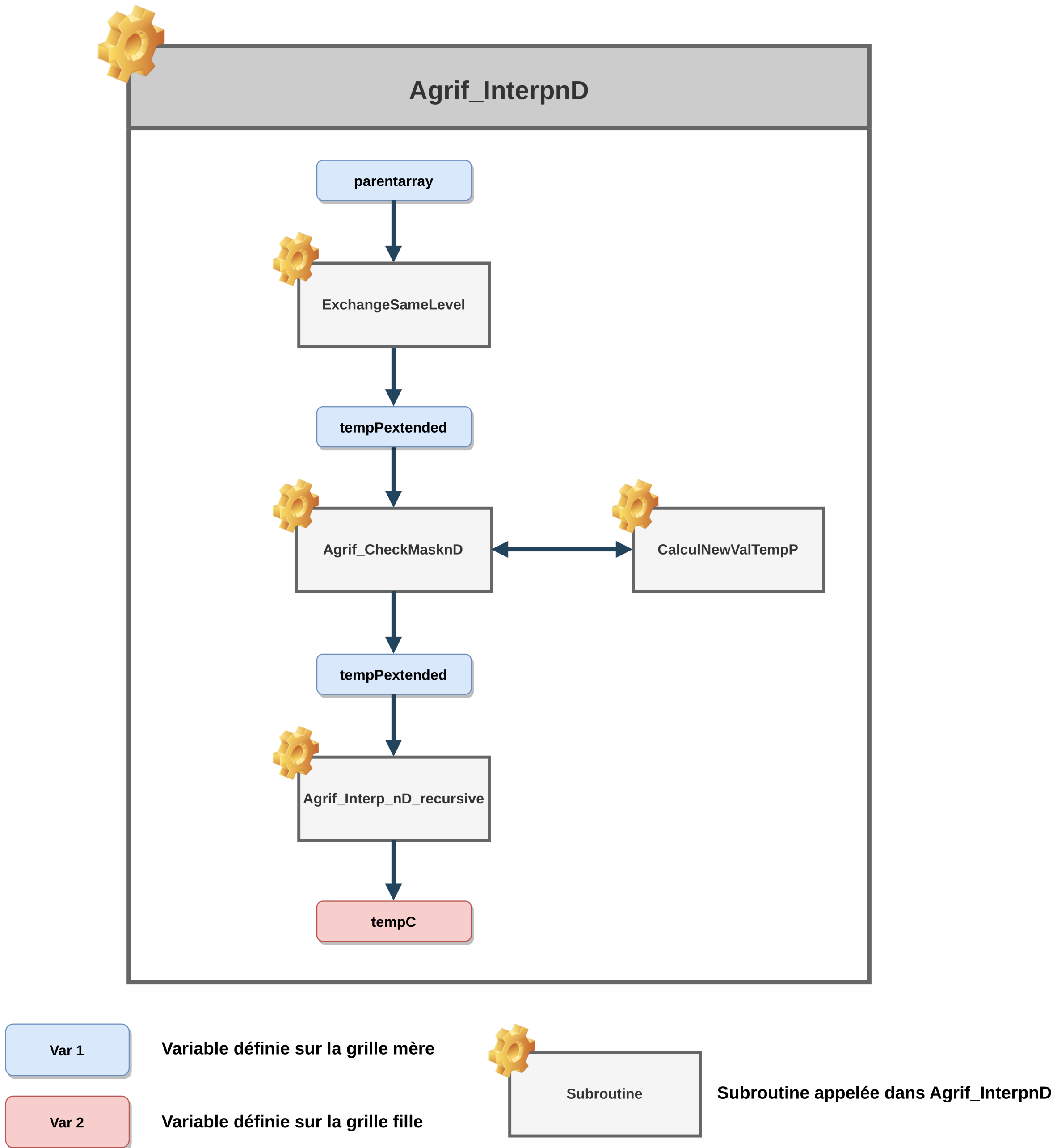


FIGURE V.11 – Structure de la routine d'interpolation *Agrif_InterpnD*.

— Plaçons-nous sur un sous-domaine MPI donné. Dans un premier temps, pour tous les points de la grille enfant appartenant à la zone d'interpolation, on sélectionne les valeurs de la grille parent situées dans les stencils d'interpolation.

— Les valeurs sélectionnées peuvent s'étendre sur plusieurs sous-domaines de la grille parent du fait de la taille du stencil mais aussi parce que la routine *Agrif_CheckMasknD* (et *CalculNewValTempP*) requière des points voisins du stencil, jusqu'à une distance de

$MaxSearch$ (où $MaxSearch$ est un paramètre fixé à 5 dans le code AGRIF et utilisé pour le remplissage des points terre discutés à la section précédente). Toutes les valeurs prises en ces points doivent être centralisées sur le processus MPI en charge de la portion de la zone d'interpolation de la grille enfant. Ce rassemblement est effectué par la routine *ExchangeSameLevel* qui réalise des communications MPI dans ce but. Il en résulte un tableau contenant des valeurs de la grille parent provenant de plusieurs sous-domaines : *tempPextended*.

— La routine *Agrif_CheckMasknD* est ensuite appelée pour détecter les éventuels points terre du tableau *tempPextended* qui feront partie d'un stencil. Pour chacun de ces points terre *CalculNewValTempP* est appelé pour calculer la valeur à utiliser pour ce point.

— Une fois toutes ces étapes préliminaires effectuées, l'interpolation est réalisée via les fonctions *Agrif_Interp_**n**D_recursive* où ***n*** est la dimension du tableau interpolé. Les valeurs des points du stencil sont combinées suivant le schéma d'interpolation puis le résultat est imposé aux points de la grille enfant sur le sous-domaine MPI.

4.1.1 Analyse de la routine de traitement des points terre *Agrif_CheckMasknD*

Analysons maintenant le goulot d'étranglement engendré par la routine *Agrif_CheckMasknD*.

Son algorithme est affiché sur Algorithme 2, il consiste en un appel à une routine de correction (*CalculNewValTempP*) pour tous les points terre détectés qui seront utilisés dans le stencil de l'interpolation.

CalculNewValTempP, quant à elle, remplace la valeur d'un point terre par la moyenne des points océan les plus proches. On notera que dans l'algorithme *CalculNewValTempP* (Algorithme 3) les valeurs prises à l'indice $point$ par le tableau sur la grille parent, désigné par $parent(point)$, sont à différencier des valeurs stockées dans le tableau $tempP$ qui seront uniquement utilisées dans le stencil d'interpolation. Le tableau $parent$ est une copie de $tempP$ effectuée avant l'entrée dans la routine *Agrif_CheckMasknD* et reste inchangé au cours de la

routine contrairement à $tempP$ qui est modifié au fur et à mesure. Utiliser les valeurs de $parent$ pour corriger des points de $tempP$ permet que le résultat ne dépende pas de l'ordre dans lequel on fait les corrections, ce qui serait le cas si uniquement $tempP$ était utilisé.

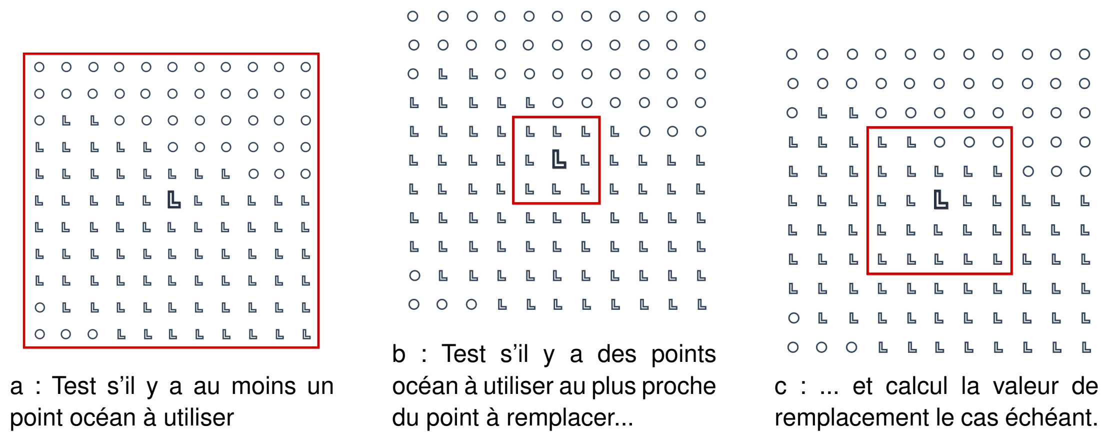
Algorithm 2 *Agrif_CheckMasknD*

```
while StencilPoint dans tous les points utilisés dans l'interpolation do
    if point Terre then
        call CalculNewValTempP( StencilPoint )
    end if
end while
```

Algorithm 3 *CalculNewValTempP*

```
whole_search_domain ← carré de largeur MaxSearch autour de StencilPoint
while point dans whole_search_domain do                 ▷ voir schéma V.12.a
    if parent(point) ≠ Landvalue then
        Existunmasked ← .TRUE.
        exit
    end if
end while
if ! Existunmasked then
    return
end if
i ← 1
while i ≤ MaxSearch do
    Res ← 0
    Nbvals ← 0
    search_domain ← carré de largeur i autour de StencilPoint
    while point dans search_domain do       ▷ voir schéma V.12 (b :i = 1, c :i = 2)
        if parent(point) ≠ Landvalue then
            Res ← Res + parent(point)
            Nbvals ← Nbvals + 1
        end if
    end while
    if Nbvals > 0 then
        tempP(StencilPoint) ← Res / Nbvals
        exit
    else
        i ← i + 1
    end if
end while
```

L'algorithme de *CalculNewValTempP* est constitué de 2 phases illustrées sur la figure V.12.



a : Test s'il y a au moins un point océan à utiliser

b : Test s'il y a des points océan à utiliser au plus proche du point à remplacer...

c : ... et calcul la valeur de remplacement le cas échéant.

FIGURE V.12 – Algorithme de *CalculNewValTempP*, les L bleus représentent des points terre, les O bleus des points océan et la ligne rouge délimite la zone de recherche des points océan.

La première phase vise à déterminer s'il existe au moins un point océan dans une zone de largeur $2 \times MaxSearch + 1$ centrée autour du point du stencil à remplacer. Cette zone est représentée sur le schéma V.12 à gauche et délimitée par une ligne rouge. Si un tel point n'existe pas, la routine se termine directement sans modification du point terre. La modification étant de toute façon impossible cela permet d'éviter la deuxième phase plus coûteuse en calcul. Ce cas de figure est courant car l'interpolation est effectuée aussi sur les dimensions de certains tableaux contenant uniquement des points terre. Par exemple, sur la dimension verticale des tableaux 3D, le niveau vertical le plus profond ne comporte que des points terre, mais l'algorithme *CalculNewValTempP* sera quand même appelé.

Si un point terre est trouvé au cours de la première phase, la routine passe dans la deuxième phase qui vise à trouver le plus petit carré, centré sur le point à remplacer, contenant au moins un point océan, puis à assigner la valeur du point à remplacer à la moyenne des points océan de ce carré (voir schéma V.12 centre et droite). Cette phase consiste en une boucle croissante sur la taille de la zone carrée centrée sur le point du stencil dans laquelle on cherche des points océan. Si de tels points sont trouvés dans une zone donnée, la valeur du point terre du stencil est remplacée et stockée dans le tableau $tempP$. Sinon on augmente la taille de la zone de recherche d'un point dans chaque direction. La première phase garantie que la boucle s'arrête avant que la zone de recherche ne dépasse $MaxSearch$ points dans chaque direction.

Il ressort de l'analyse de l'algorithme de traitement des points terre que la taille maximale de la zone de recherche de points océan définie via le paramètre $MaxSearch$ est centrale dans le temps de calcul de cette routine. En effet, la complexité de l'algorithme est en $O(MaxSearch^2)$. Nos efforts d'optimisation se concentrerons donc sur la réduction de la valeur de ce paramètre.

4.2 Test de la réduction de la taille de la zone de recherche

Avant de commencer l'implémentation d'une optimisation complexe de la routine de remplacement des points terre, on teste l'impact qu'aurait une telle optimisation de réduction de la taille de la zone de recherche. La configuration BENCH_AGRIF bénéficie des mêmes caractéristiques que le configuration BENCH, entre autres sa stabilité. Ce test tire parti de cette stabilité en changeant les conditions d'exécution de l'algorithme sans se soucier de garder un résultat physique identique au bit près. Ici, le test consiste simplement à fixer la variable $MaxSearch$ à 2 au lieu de 5 comme c'est le cas par défaut. La zone de recherche maximale est donc limitée à 2 points dans chaque direction. Un tel test n'est pas à considérer comme une optimisation mais simplement comme une étape permettant d'estimer si une optimisation sur la taille du domaine de recherche serait efficace et d'estimer ses performances.

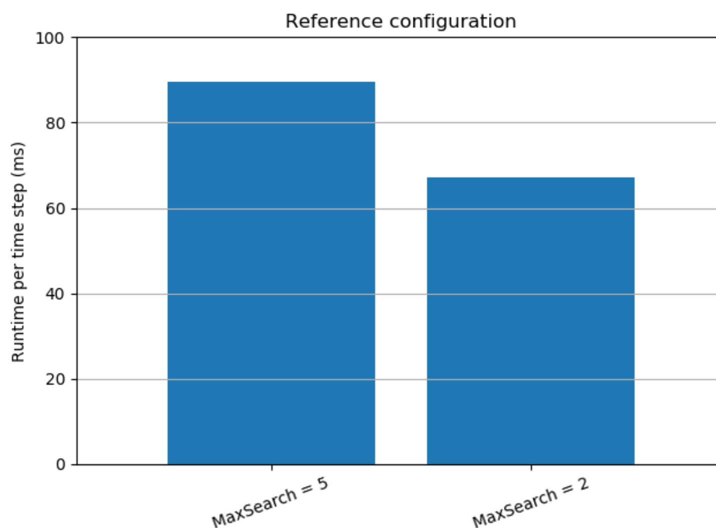


FIGURE V.13 – Performances du pas de temps moyen de BENCH_AGRIF avec $MaxSearch = 5$ ou $MaxSearch = 2$.

Temps de calcul par pas de temps (ms)	$MaxSearch = 5$	$MaxSearch = 2$
Exécution d'un pas de temps	87	69
$Agrif_CorrectnD$	19	9
$Agrif_InterpnD$	14	5.4
$Agrif_CheckMasknD$	11	3

TABLE V.2 – Profilage des routines de BENCH_AGRIF via gprof avec $MaxSearch = 5$ ou $MaxSearch = 2$.

Temps de calcul par pas de temps (ms)	$MaxSearch = 5$	$MaxSearch = 2$
$Recv$	18	10
$Allreduce$	17	8.7
$Isend$	0.9	0.9
MPI imbalance	25.25	10.6

TABLE V.3 – Profilage MPI de BENCH_AGRIF via aps avec $MaxSearch = 5$ ou $MaxSearch = 2$.

L'amélioration des performances est montrée figure V.13, le passage à $MaxSearch = 2$ permet une amélioration des performances de 20%. Plus précisément, cette amélioration est principalement due à une réduction du temps de calcul de la routine $Agrif_CheckMasknD$ de 73% comme cela est détaillée sur le tableau V.2. Par ailleurs, un profilage MPI (voir tableau V.3) montre une réduction du temps passé dans les communications, moins 44% dans les communications points à points (*Recv* et *Isend*) et moins 43% aussi dans les communications globales (*Allreduce*). Cette réduction n'est pas due directement à une optimisation des communications mais à une amélioration de la répartition de charge qui se traduit par une réduction du temps passé dans les routines MPI. Le tableau montre en effet une réduction de *MPI_Imbalance* de 58%, cette métrique cumule le temps passé dans des routines MPI à attendre des messages en étant inactif, une valeur élevée reflète donc une mauvaise répartition de charge. La répartition de charge de la simulation a été améliorée en éliminant une partie des calculs dont avait la charge les processus situés sur la zone d'interpolation.

L'amélioration des performances est considérable même évaluée sur une configuration sans continents. Les seuls points terre rencontrés sont donc ceux des tableaux qui comportent des points terre même en l'absence de continents comme le dernier niveau vertical pour certains tableaux 3D.

Ce test a donc permis de montrer qu'une optimisation sur la taille de la zone de recherche dans *CalculNewValTempP* donne effectivement lieu à des améliorations de performance. On s'intéresse donc maintenant à l'implémentation d'une telle optimisation qui garantirait un résultat physique identique au bit près à la simulation avant l'optimisation.

4.3 Optimisation : réduction de la taille de la zone de recherche

Pour réaliser une optimisation sans impact sur les résultats, on s'appuie sur une caractéristique de AGRIF lorsqu'il s'applique à NEMO. Dans NEMO, le masque définissant les points terre est différent pour la grille parent et enfant pour, par exemple, permettre une meilleure représentation de la ligne de côte dans la grille enfant. Ce raffinement de la bathymétrie dans la grille enfant doit cependant respecter la contrainte suivante : le volume d'océan d'une cellule parent doit être identique à la somme des volumes océans des $\rho_x \times \rho_x$ cellules enfants correspondantes. Le respect de cette contrainte impose donc que si une de ces $\rho_x \times \rho_x$ cellules enfants est de type océan, alors la cellule parent correspondante est aussi de type océan avec une épaisseur calculée de manière à conserver le volume d'océan entre les grilles parents et enfants. Sur la figure V.14, cela revient à dire que si le rond rouge est un point océan alors le point de la grille parent au milieu doit être un point océan. Ces adaptations de la grille parent en fonction de la bathymétrie de grille enfant sont effectuées dans une phase de pré-traitement de la simulation.

Cette caractéristique peut être mise à profit pour réduire la zone de recherche de *CalculNewValTempP*.

On appelle S^i la taille du stencil dans la dimension i, que l'on définit par le nombre de points sur la grille parent dans une direction de dimension i utilisé dans le stencil. $S = \max_i S^i$ est appelé la taille du stencil. Par exemple, pour une interpolation linéaire sur une dimension i deux points sont utilisés, un pour chaque direction donc $S^i = 1$. Spécifier la dimension est important puisque AGRIF offre la possibilité de spécifier, pour un même tableau, des schémas d'interpolation différents pour chaque dimension. Cette fonctionnalité et typiquement utilisée pour pouvoir

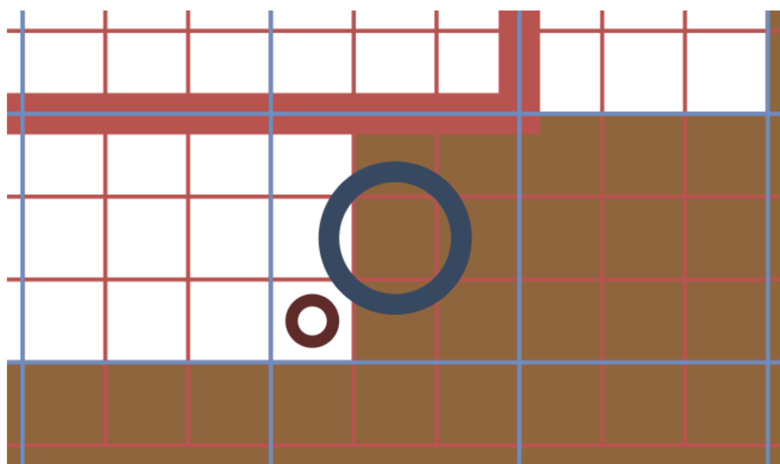


FIGURE V.14 – Exemple d'un point de la grille parent (rond bleu foncé) que NEMO impose d'être un point océan du fait d'un point océan sur la grille enfant (rond rouge foncé) localisé dans la cellule du point de la grille parent. La grille en bleu (resp. rouge) délimite les cellules de la grille parent (resp. enfant). Les points terre de la grille enfant sont représentés en brun. La ligne rouge épaisse est la limite du cœur du domaine enfant.

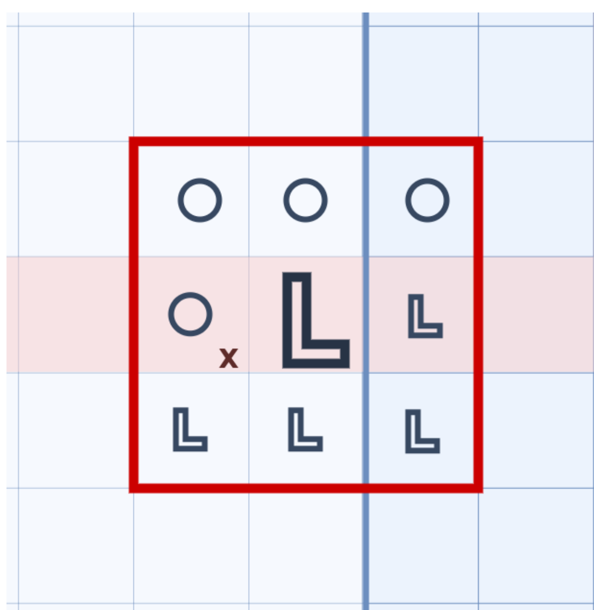


FIGURE V.15 – Exemple de remplacement d'un point terre dans le cas d'une interpolation avec un stencil de taille $S = 1$. La croix rouge représente le point de la grille enfant où l'interpolation va être appliquée. Les O (resp. L) sont des points océan (resp. terre) de la grille parent. Cette interpolation avec un stencil de taille $S = 1$ implique les trois points terre de la grille mère qui sont les plus proches de la croix rouge et qu'il faut donc remplacer avant interpolation. On se focalise sur Le point L bleu central en gras que l'algorithme va remplacer par la moyenne des points océan les plus proches. La taille de la fenêtre de recherche de ces points océan est de $S = 1$ dans chaque direction (rectangle rouge). 4 points océan seront donc utilisés pour remplacer le point terre central.

différencier les interpolations sur les dimensions zonales et méridionales pour des variables ayant des comportements particuliers suivant ces dimensions, par exemple les vitesses zonales et méridionales.

type d'interpolation	taille du stencil S^i
constant	0
linear, linearconserv	1
ppm	2

TABLE V.4 – Taille du stencil pour chaque type d'interpolation de AGRIF.

Un point du stencil est à une distance maximale de S^i points de la grille parent dans la dimension i du point de la grille enfant sur lequel l'interpolation va s'appliquer. Sur la figure V.15, le point terre central faisant parti du stencil de taille $S = 1$ est à moins d'un point sur la grille parent de la croix rouge. Si le point de la grille enfant où l'interpolation doit s'appliquer est un point terre, alors le résultat de l'interpolation n'a pas d'importance car il ne sera pas appliqué. Si le point de la grille enfant est un point océan alors nous avons vu que le point de la grille parent correspondant est aussi un point océan. Un point du stencil dans une direction de la dimension i est donc à une distance maximale de S^i d'un point océan.

Supposons que le point de la grille fille interpolé est un point océan, un point du stencil est alors au plus à une distance d'un point océan du maximum de la taille du stencil sur les dimensions c'est-à-dire $S = \max_i S^i$. On peut donc réduire la taille de la zone de recherche utilisée dans *CalculNewValTempP* à S. Sur la figure V.15, cela revient à dire que tous les points utilisés pour corriger la valeur sur le point terre centrale se trouvent obligatoirement dans le carré délimité par la ligne rouge.

Le tableau V.4 répertorie la taille des schémas d'interpolations mis à disposition par AGRIF, la taille maximum est de $S^i = 2$ bien inférieure à $MaxSearch = 5$. Les opérations dans *Calcul-NewValTempP* peuvent donc être fortement réduites.

Cette possibilité d'optimisation étant conditionnée à une propriété du code NEMO, elle ne peut pas être implémentée directement dans AGRIF car cela s'appliquerait alors aussi à des codes n'ayant pas cette propriété, par exemple le code CROCO. On implémente donc au sein de la librairie AGRIF la possibilité de spécifier un paramètre optionnel, appelé *msearch*, lors de la

définition de l'interpolation dans NEMO (dans les routines *Agrif_Set_bcinterp* et *Agrif_Set_interp* dans le fichier *src/NST/agrif_user.F90*). Ce paramètre est fixé à la main à $\max_i S^i$ dans NEMO lors du choix de chaque type d'interpolation.

On vérifie que l'implémentation laisse les résultats physiques inchangés à l'aide d'une exécution sur la configuration BENCH_AGRIF avec points terre. Les points terre sont placés sur cette configuration de manière à ce que le stencil de l'interpolation sur la grille parent s'étende en partie sur des points terre. Cette particularité permet de s'assurer que le code entre dans la routine de remplacement des points terre *CalculNewValTempP*.

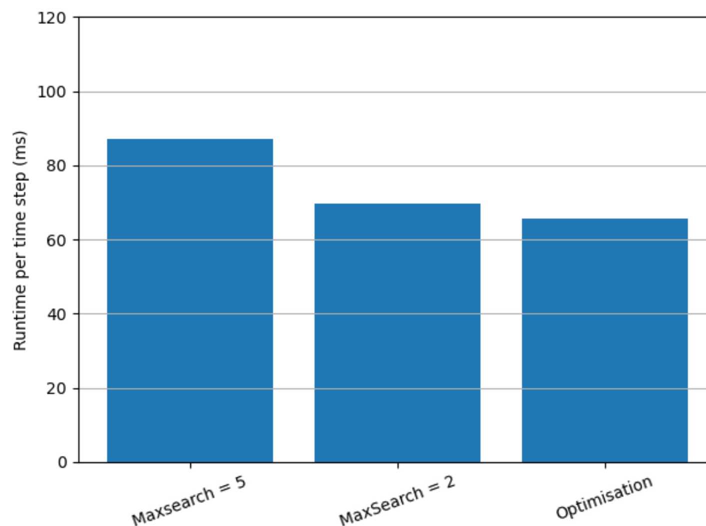


FIGURE V.16 – Performances sur la configuration BENCH_AGRIF sans continents.

La figure V.16 montre les performances apportées par l'implémentation de cette optimisation qui sont un peu meilleures que les performances apportées par le passage à $MaxSearch = 2$. La version *MaxSearch = 2* est 20% moins longue à l'exécution et notre optimisation est 25% plus rapide que la version de référence (indiqué *MaxSearch = 5*). Il est cohérent que notre optimisation soit meilleure que la version *MaxSearch = 2* car les stencils utilisés dans NEMO impliquent des zones de recherche dans le pire des cas égales à celles de *MaxSearch = 2*.

Le gain en performance est beaucoup plus impressionnant sur la configuration avec continents. La figure V.17 présente ces performances, la version optimisée du code étant 40% plus rapide à l'exécution que la version de référence. Le tableau V.5 montre que *Agrif_CheckMasknD*

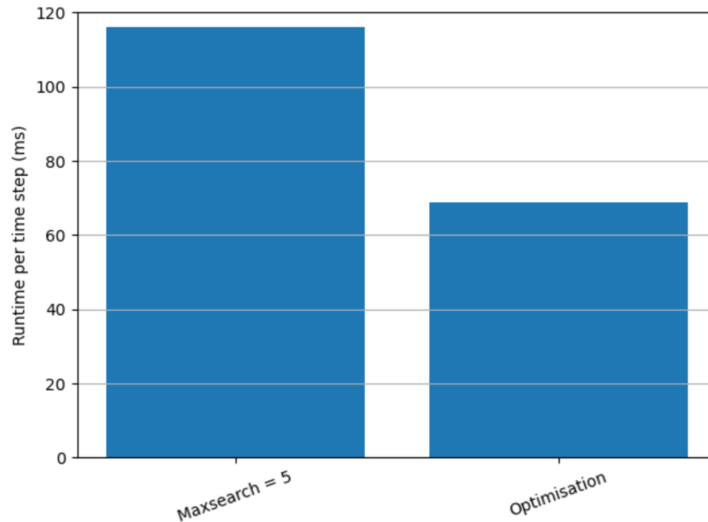


FIGURE V.17 – Performances sur la configuration BENCH_AGRIF avec continents.

Temps de calcul par pas de temps (ms)	$MaxSearch = 5$	$Optimisation$
Agrif_Correctnd	64	15
Agrif_InterpnD	58	10
Agrif_CheckMasknD	52	6

TABLE V.5 – Profilage des routines de BENCH_AGRIF via gprof pour $MaxSearch = 5$ et l'optimisation pour la configuration avec continents.

passe de $52ms$ de temps de calcul par pas de temps à seulement $6ms$, c'est-à-dire une réduction de 88%. Le gain en performances de l'optimisation est entièrement expliqué par cette amélioration, *Agrif_CheckMasknD* représentait 45% du temps de calcul et seulement 9% maintenant.

L'optimisation de la taille de la zone de recherche que nous avons implémentée apporte une amélioration des performances considérable.

4.4 Optimisation : Déplacement de l'algorithme de recherche à l'initialisation

L'objet de cette partie est de tester une optimisation visant à réduire le temps passé dans la routine *Agrif_CheckMasknD* qui reste encore à 9% du temps total pour notre configuration avec points terre.

La logique de cette optimisation est la suivante : étant donné que le schéma d'interpolation, la position de la zone d'interpolation du zoom et la position des points terre sont identiques tout au long de la simulation, les informations utilisées pour le remplacement des points terre du stencil de l'interpolation sont disponibles dès l'initialisation. La recherche à chaque pas de temps peut donc être évitée en effectuant la recherche pour le remplacement des points terre durant l'initialisation et stockant les indices des points à utiliser pour le remplacement. À chaque pas de temps de la grille enfant, le remplacement des points terre du stencil se résume alors à récupérer les valeurs prises aux points océans déjà identifiés et d'utiliser ces valeurs dans le schéma d'interpolation.

Cette optimisation est compatible avec l'optimisation sur la taille de la zone de recherche, mais la rend caduque. En effet, cette recherche n'étant alors effectuée qu'à l'initialisation, une optimisation de la recherche n'a que peu d'intérêt du point de vue de l'amélioration des performances. L'optimisation présentée ici lui est donc largement concurrente et vise à la dépasser en termes de performances.

La recherche pour le remplacement des points terre du stencil est la même à l'initialisation que celle effectuée à n'importe quel pas de temps. La routine *Agrif_CheckMasknD* peut donc être utilisée pour l'initialisation sans modification majeure. Il reste à définir une structure adaptée pour le stockage des informations récoltées et son utilisation pour l'interpolation.

Une structure est mise en place pour stocker efficacement les indices des points servant au remplacement des points terre, cette structure correspondant à l'organisation de la routine d'interpolation. AGRIF considère la zone d'interpolation de la grille enfant entourant son domaine comme étant composée de plusieurs sections schématisées sur la figure V.18. Par exemple, pour les configurations BENCH_AGRIF, il y a deux sections pour la dimension zonale (est et ouest) et deux pour la dimension méridionale (nord et sud). AGRIF classe les sections suivant la dimension selon laquelle est orientée la section d'interpolation (zonale, méridionale, verticale ainsi qu'une quatrième possibilité) et suivant la direction de l'orientation de la section (avec deux possibilités,

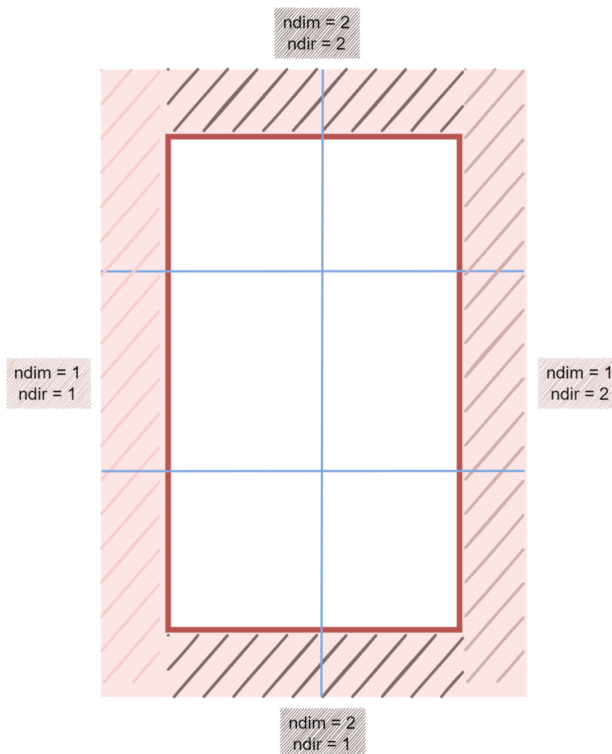


FIGURE V.18 – Schéma de la structure de la zone d'interpolation. Chaque section est hachurée dans une teinte différente. Un exemple de découpage MPI est fourni par les lignes bleues.

par exemple "est" et "ouest" pour la dimension zonale). Pour les processus MPI ayant plus qu'une seule section dans leur sous-domaine, comme pour les sous-domaines dans les coins de la grille enfant, l'interpolation est effectuée par AGRIF une section après l'autre.

La structure de stockage des indices prend une forme similaire schématisée figure V.19. Un tableau à double entrée, dimension et direction, est utilisé. Les indices de ce tableau ciblent une section de la zone d'interpolation. Une case est remplie par un pointeur si la section de la zone d'interpolation désignée par l'indice de la case est dans le sous-domaine MPI, le pointeur pointe alors vers un nouveau tableau dont chaque élément correspond à un point terre à corriger pour l'interpolation de cette section. Si la section n'est pas dans le sous-domaine la case est remplie par le pointeur *Null*. Le tableau contenant les points terre à remplacer est lui aussi un tableau de pointeurs dont chaque pointeur pointe vers une structure de données. La première donnée de la structure est l'indice du point terre à corriger, la deuxième est le nombre de plus proches voisins océan et la troisième est un tableau des indices de ses plus proches voisins.

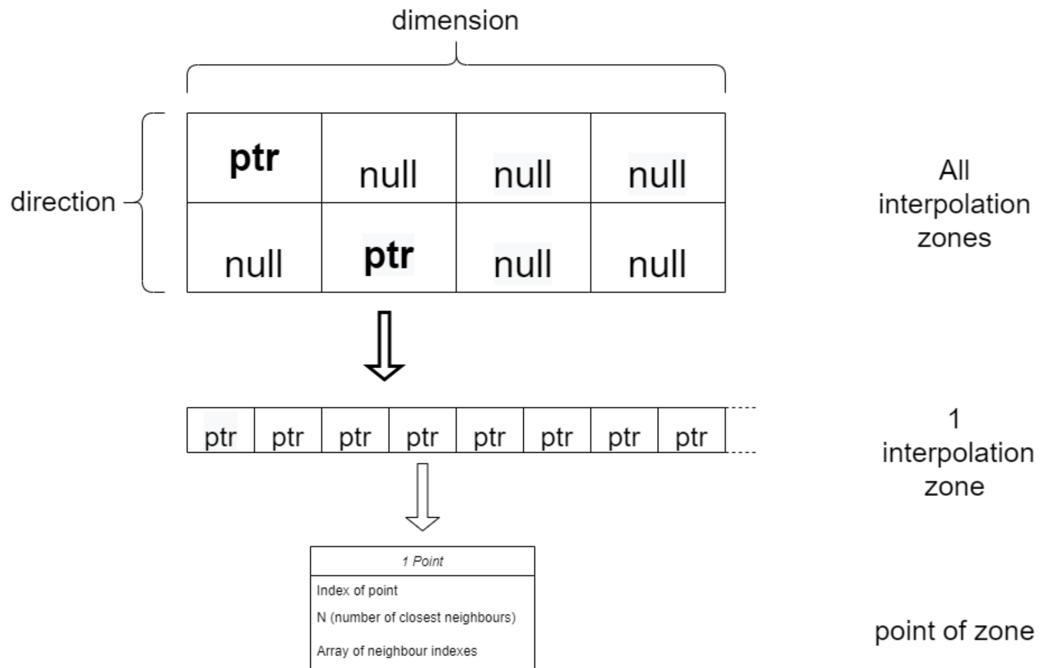


FIGURE V.19 – Schéma de la structure de stockage de données pour l'interpolation

Cette structure de stockage des indices doit être dupliquée pour toutes les variables. En effet, chaque variable est définie sur des points potentiellement différents (T,U,V) et a sa propre définition de sa zone d'interpolation et de son schéma d'interpolation dans ses dimensions zonales et méridionales dont découle un ensemble de points à remplacer.

La structure de stockage des indices est remplie durant l'initialisation via une routine similaire à *Agrif_CheckMasknD* modifiée pour seulement stocker les différents indices.

Une fois la structure de stockage des indices implémentée et remplie elle est utilisée pour l'interpolation. La correction des points terre du stencil consiste alors en une boucle sur le tableau des points terre à remplacer. Pour chacun de ces points terre, la structure de données donne l'indice du point à corriger et les indices des points océans à utiliser pour cette correction. Il suffit de calculer la moyenne des valeurs prises aux points océan pour l'utiliser dans la correction. L'étape de correction des points terre a ainsi été simplifiée en éliminant l'étape de recherche de points océan.

Les performances de cette modification sont présentées figure V.20, cette optimisation améliore elle aussi considérablement les performances par rapport à la version de référence, mais reste seulement équivalente à l'optimisation sur la taille de la zone de recherche. En regardant

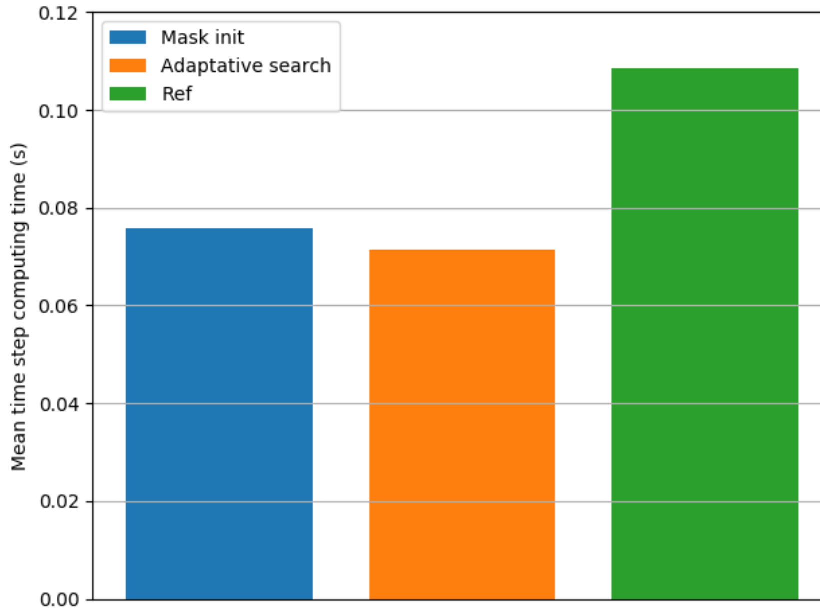


FIGURE V.20 – Performances sur la configuration BENCH_AGRIF avec continents. En bleu les performances pour l'optimisation présentée ici, en orange pour l'optimisation sur la zone de recherche et en vert pour la version de référence.

Temps de calcul par pas de temps (ms)	*Reference*	*Optim recherche*	*Initialisation*
Agrif_Correctnd	40	16	14
Agrif_CheckMasknD	27	6.5	2.3

TABLE V.6 – Profilage des routines de la configuration BENCH_AGRIF via gprof pour l'optimisation sur la taille de la zone de recherche et le déplacement à l'initialisation pour la configuration avec continents.

	Reference	*Optim recherche*	*Initialisation*
Memory stalls	16%	18%	19%
DRAM stalls	4.7%	6.7%	7.2%
MPI	37%	27%	31%
MPI unbalance	24%	10%	13%

TABLE V.7 – Profilage de la configuration BENCH_AGRIF via aps pour l'optimisation sur la taille de la zone de recherche et le déplacement à l'initialisation pour la configuration avec continents.

plus en détail les performances des routines d'interpolation (présenté tableau V.6) il apparaît que la routine de remplacement des points terre a été significativement optimisée et, par conséquent, la totalité de l'interpolation. La raison de cette apparente contradiction est mise en évidence tableau V.7. Le temps de simulation total n'est pas diminué du fait de l'empreinte mémoire supplémentaire demandée par le stockage des indices. Les problèmes de mémoires ralentissent

l'ensemble de la simulation et limitent les performances apportées par cette optimisation. On voit que le temps passé dans MPI et le déséquilibre de charge est plus grand que pour l'optimisation sur la taille de la zone de recherche ce qui est dû au fait que le ralentissement lié à la mémoire n'a lieu que sur les processus en charge de l'interpolation.

Il faut noter que les performances exposées figure V.20 et sur les tableaux V.6 et V.7 ne sont pas comparables aux performances exposées plus hauts. Cela est dû au fait que des nœuds différents ont été utilisés pour ces simulations ce qui entraîne des performances très différentes.

L'optimisation implémentée dans cette partie montre des performances intéressantes, équivalentes à celles de l'optimisation sur la taille de la zone de recherche sans pour autant les dépasser. Un travail plus poussé pourrait permettre d'encore améliorer les performances.

4.5 Pistes pour la réduction de l'empreinte mémoire du stockage des indices

Le problème de l'optimisation présentée à la section précédente étant l'empreinte mémoire demandée par le stockage des indices utilisés dans la correction des points terre du stencil, il est possible d'améliorer l'optimisation en réduisant son besoin en mémoire. Plusieurs pistes n'ayant pas encore été implémentées sont présentées ici.

4.5.1 Optimisation limitant la correction des points terre

L'algorithme actuel effectue donc une correction sur de nombreux points terre et calcule un schéma d'interpolation alors que le résultat ne pourra parfois pas s'appliquer sur la grille enfant comportant elle aussi des points terre. Le stockage de certains indices ainsi que le calcul du schéma d'interpolation peut être évité. Une telle optimisation limiterait donc l'empreinte mémoire en épurant la structure de stockage mais réduirait aussi le temps passé dans la correction des valeurs sur les points terre car un plus petit nombre serait concerné. L'effet de cette optimisation dépend fortement de la configuration et plus particulièrement de la bathymétrie autour du ou des zooms.

Une possibilité d'implémentation serait la suivante :

On supprime de la structure de données stockant les points terres des éventuelles stencils (figure V.19) tous les points terre qui sont utilisés uniquement pour des interpolations s'appliquant sur des points terre de la grille fille. On supprime ces points durant la phase d'initialisation de la simulation, l'algorithme pourrait consister à ajouter à chaque point terre de la structure de données un logical $lisused$. Dans un premier temps, $lisused$ est fixé à $True$ si un des points de la grille fille sous-jacente est océan et à $False$ dans le cas contraire. Une routine inspirée de *CalculNewValTemp* et prenant en compte la taille du stencil peut alors servir à fixer $lisused$ à $True$ pour tous les points qui doivent être gardés pour l'interpolation.

Une fois ces étapes effectuées, $lisused$ est à $True$ uniquement pour les points terre qui seront utilisés dans l'interpolation d'un point océan de la grille enfant. Une copie de la structure de données omettant les points terre dont $lisused$ est à $False$ permet d'avoir une nouvelle structure épurée des points inutiles à remplacer. Cette structure peut alors être utilisée pour mettre en place l'algorithme décrit dans la section précédente.

4.5.2 Mise en commun des structures de données

L'empreinte mémoire du stockage des indices est importante aussi parce qu'elle est répétée pour chaque variable interpolée. Or, un certain nombre de variable partage leur zone d'interpolation. Il semble donc possible de mettre en commun un certain nombre de structures de données pour le remplacement des points terre en définissant des structures différentes uniquement pour des points différents (T,U,V), pour des tableaux 2D et 3D et pour des zones d'interpolation différentes et plus systématiquement pour des variables différentes.

Les structures de données peuvent être créées en prenant le plus grand stencil d'interpolation accepté par AGRIF, certains points terre seront alors stockés et corrigés sans que cela soit nécessaire mais cette stratégie permet de réduire le nombre de structures de données utilisées.

Encore une fois la routine *CalculNewValTemp* peut être utilisée pour construire ces structures. La structure $parent$, reliée à une variable et contenant le tableau de ses valeurs et des informations la concernant, peut être utilisée en y ajoutant un pointeur vers la structure de données des points terre adaptée pour l'interpolation de cette variable.

4.5.3 Enlever les redondances pour les tableaux de multiples dimensions

Certaines variables NEMO regroupent plusieurs variables physiques en une seule, par exemple, la température et la salinité sont regroupées dans une seule variable NEMO. La dernière dimension d'un tel tableau permet de passer d'une variable physique (température, par exemple) à une autre (salinité). NEMO utilise parfois de tels tableaux, en particulier pour le traitement de la glace ou de la bio-géochimie. Or quand de tels tableaux sont regroupés, les variables physiques sont toujours définies aux même points (T,U,V) et partagent la même zone d'interpolation ainsi que le schéma d'interpolation. L'interpolation se passe donc de manière identique et la liste des points terre à corriger est la même. Il est donc possible d'enlever cette redondance en ne gardant en mémoire le stockage des indices qu'une seule fois.

L'implémentation pourrait consister en l'ajout d'une variable indiquant si une dimension d'une variable est une dimension permettant de regrouper plusieurs variables physiques en une seule. Le travail à effectuer pour le stockage des indices n'a donc besoin d'être fait qu'une fois en fixant un indice dans cette dimension.

L'implémentation de ces améliorations de l'optimisation consistant à effectuer la recherche des points terre à l'initialisation limiterait le besoin en mémoire de l'optimisation et permettrait de la rendre plus performante. Ces différentes améliorations sont de plus compatibles les unes avec les autres.

5 Conclusion et perspectives

J'ai montré dans ce chapitre que le code AGRIF pouvait être significativement optimisé, en particulier sur deux aspects : les communications MPI et l'algorithme d'interpolation.

Dans un premier temps j'ai développé deux configurations pour permettre un profilage représentatif de différentes configurations réalistes tout en assurant un profilage facile notamment grâce à un équilibrage de charge semblable entre les différents processus MPI.

Un profilage de ces configurations a permis d'identifier deux goulots d'étranglement à l'origine d'une part conséquente du temps de simulation.

Les parties du codes responsables de ces goulots d'étranglement ont été étudiées puis optimisées, tout d'abord la présence dans le code d'une communication MPI globale, puis la correction des points terre dans le stencil d'interpolation de la grille parent vers la grille enfant.

La communication MPI globale effectuée à chaque pas de temps a ainsi été attribuée au besoin de partage de données concernant la géométrie du zoom. Le zoom étant fixe dans un très grand nombre d'utilisations d'AGRIF, la communication globale peut être déplacée à l'initialisation sauf dans le cas d'un zoom AGRIF se déplaçant au cours de la simulation. J'ai ainsi trouvé un moyen de supprimer la communication globale de la quasi-totalité des simulations AGRIF.

L'algorithme responsable de la correction des points terre a été détaillé et une piste d'optimisation identifiée. L'optimisation implémentée a permis une réduction significative du temps passé dans l'algorithme. Une seconde optimisation visant à déplacer une partie de cette correction dans la phase d'initialisation, a été implémentée. L'empreinte mémoire demandée par cette optimisation limite ses performances, mais des pistes visant à pallier à ce problème ont été identifiées.

Il faut noter que notre analyse de AGRIF et les optimisations qui en ont découlé ont été faites dans un contexte donné que j'ai choisi puis construit. Ce contexte, qui se concrétise par la configuration BENCH_AGRIF, fait apparaître certaines limitations, que j'ai analysées et optimisées, et en masque d'autres. Certaines de ces limitations sont listées ci-dessous.

Les performances ont été analysées seulement sur des petits nombres de nœuds, les sous-domaines sont de petite taille ce qui permet de faire apparaître clairement les problèmes de performance liés aux communications MPI. Cependant, le comportement sur des simulations utilisant un grand nombre de nœuds reste inconnu ainsi que la scalabilité d'une simulation NEMO avec zoom AGRIF.

Les configurations développées ayant des tailles de sous-domaines MPI faibles, les aspects mémoire n'ont pas été testés excessivement. Une modification de nos configurations pour des sous-domaines plus grands fournirait un cadre pour l'analyse des problématiques liées à la gestion d'une quantité de données plus importantes. De plus, un unique processus MPI étant en charge d'au moins deux grilles (grille parent et grille enfant) et devant calculer tour à tour sur différentes grilles, cela accentuerait les éventuelles chutes de performances que pourrait causer la perturbation des accès mémoires par le changement de grille.

De manière générale, l'étendue des configurations utilisant AGRIF est gigantesque. La taille du zoom par rapport au domaine parent peut être modifié ainsi que le nombre de zooms ou le nombre de niveaux de zooms (un zoom dans un zoom dans une grille parent par exemple). Pour ce genre de configurations plus complexes le problème de l'équilibrage de charge que j'ai soigneusement mis de côté lors de la création de la configuration pourrait apparaître de manière criante. Pour un grand nombre de zooms, il n'est pas pertinent de calculer les résultats physiques sur chacun des zooms successivement. En effet, dans un tel cas un zoom comportera très probablement beaucoup moins de points de grille que la grille parent, la taille des sous-domaines sur les zooms sera probablement trop petite pour que l'exécution soit efficace. Pour aborder ce problème, des zooms peuvent être calculés de manière parallèle tant que les dépendances, c'est-à-dire les interactions entre les grilles, sont respectées. Cette solution pose néanmoins de nouvelles questions qui méritent d'être étudiées, par exemple, comment répartir les ressources de calculs pour le calcul des différents zooms.

Par ailleurs, les performances de AGRIF ont été évaluées uniquement sur des simulations NEMO. Or AGRIF est utilisé sur d'autres codes, notamment le code CROCO. Il convient d'analyser les performances de simulation AGRIF sur le code CROCO également, certaines particularités pouvant impacter les performances. Par exemple, CROCO utilise une discrétisation verticale en niveaux σ suivant la bathymétrie alors que NEMO qui utilise dans la majorité des cas une discrétisation en niveaux z. Le nombre de points terre d'une configuration n'est pas le même suivant qu'une ou l'autre discrétisation est utilisée et, mécaniquement, les performances de la routine de

correction des points terre de l'interpolation différent (section 4). De surcroît, les optimisations à apporter peuvent varier selon que NEMO, CROCO ou d'autres modèles sont utilisés. Dans le section 4.3, l'optimisation n'est possible que grâce à certaines particularités de NEMO.

CHAPITRE

VI

Discussion

Il convient tout d'abord de rappeler brièvement les travaux présentés dans ce document. Les configurations BENCH_OCE et TSUNAMI sont choisies pour mener à bien l'optimisation des performances de NEMO, elles ont pour caractéristique de mettre l'emphase sur le coût des communications. Des outils de profilage sont présentés et développés et une méthodologie d'évaluation des performances est désignée.

Sur ces bases, une analyse de NEMO met en évidence les principaux freins aux performances : le nombre de communications à chaque pas de temps du code, les ralentissements à plusieurs niveaux de l'exécution (au niveau du pas de temps avec différentes ampleurs et au niveau d'un cœur de la simulation). Le nombre de communications est réduit en éliminant les communications inutiles liées au traitement des frontières ouvertes et au calcul du gradient de pression. Une méthode est trouvée pour soulager considérablement le phénomène à l'origine des pas de temps fortement ralentis. Pour limiter l'impact des ralentissements au niveau d'un cœur, une stratégie est élaborée qui consiste à atténuer le ralentissement au fur et à mesure qu'il est propagé, de sous-domaine MPI en sous-domaine MPI, par le schéma de communication.

L'optimisation du schéma de communication fait l'objet d'une analyse approfondie. Dans un premier temps, le schéma de communication utilisé par défaut dans NEMO est optimisé, puis une variété de schémas sont développés, chacun utilisant une fonctionnalité MPI avancée ou une stratégie différente. Le schéma par défaut de NEMO optimisé présente les meilleures performances, sauf pour certaines portions du code où les coins des sous-domaines n'ont pas besoin d'être mis à jour par une communication.

Je me suis finalement intéressé à AGRIF, un outil permettant l'exécution de simulations avec zooms emboîtés et évitant ainsi le coût en temps d'exécution d'une montée en résolution sur l'ensemble du domaine. Des configurations adaptées à l'analyse de AGRIF ont été développées et une série de profilage a été réalisée. Il en ressort que AGRIF est un outil pertinent pour réduire le coût des simulations, mais peut encore être significativement optimisé, en particulier sur deux aspects : l'utilisation d'une communication globale et la gestion des points terre dans l'interpolation. Des optimisations ont été présentées pour chacun de ces aspects.

Un grand nombre de pistes d'optimisation et de recherche supplémentaires à explorer ont déjà été évoquées au cours des chapitres. Citons ici d'autres axes de développements qui n'ont pas pu être cités dans ces chapitres.

Dans l'optique non pas de réduire le temps d'exécution d'une simulation mais la consommation d'énergie, la fréquence d'horloge peut être réduite en fonction de l'intensité de l'utilisation du nœud via l'utilisation de l'outil BEO (*Bull Energy Optimizer*). Une paramétrisation adaptée au code ralentit l'exécution de 1 à 5% mais peut permettre une économie d'énergie de $\approx 10\%$. Le coût monétaire de la simulation ainsi que son impact carbone peut ainsi être réduit.

La sensibilité de nos résultats au déséquilibre de charge peut être exploré, en particulier pour les performances des schémas de communications et leurs performances relatives. Est-ce que les ralentissements de l'exécution resteraient des facteurs limitant des performances ou est-ce que le temps de restitution de la simulation dépend trop directement du cœur avec la plus grande charge de calcul ?

Certaines facultés de NEMO telles que les modules de bio-géochimie et de glace ont été laissées de côté dans nos analyses, mais méritent aussi une analyse complète. Des pistes ont déjà été évoquées pour l'optimisation des performances du module de simulation de la glace, mais rien ne garantit que ces solutions soient efficaces ou qu'un profilage ne fasse pas apparaître d'autres points limitant les performances.

Au cours de ces travaux notre perspective sur l'exécution du code a évolué. Initialement, nous présumions les processus étaient tous exactement aussi efficaces les uns que les autres, autrement dit que chacun a exactement la même rapidité d'exécution tout au long de la simulation. Chaque processus exécuterait au même instant les mêmes instructions, si tant est que la charge de calcul est identiquement répartie. Dans ce cadre-là, les phases d'attente bloquantes des communications n'ont que peu d'importance. Si un processus a une plus grande charge de calcul que les autres, ce serait continuellement lui qui serait limitant pour les performances de l'ensemble de la simulation. Or, on a vu que les processus sont loin d'être synchronisés. De plus, ce sont

des processus différents qui sont, tour à tour et, semble-t-il, de manière aléatoire, les plus lents. C'est ce changement de paradigme que l'on a tenté d'exploiter dans l'optimisation du schéma de communication.

Ce manque de synchronisation s'explique par les ralentissements qui touchent tous les supercalculateurs testés que nous pensions stables au début de nos travaux. Les caractéristiques de la rapidité d'exécution et son évolution dépendent du matériel et de l'environnement logiciel utilisé. Un soin particulier doit donc être apporté à la mesure des performances qui changent au cours du temps.

Enfin, nous avons désormais une vue plus complète de l'impact de l'ensemble de la pile logiciel sur les performances. Elle doit faire l'objet d'une attention particulière, notamment les options de compilation, les librairies utilisées, mais aussi les variables d'environnement fixées par défaut ou par l'utilisateur.

Par ailleurs, en ce qui concerne la manière d'aborder l'optimisation d'un code, on peut grossièrement dégager deux manières de procéder. La première correspondant à une logique ayant plutôt trait à l'ingénierie, c'est par exemple ce qui a été mis en œuvre pour les optimisations des simulations avec zoom emboîtés au chapitre V, la réduction du nombre de communications à l'annexe A ou encore pour les premières améliorations du schéma de communication au chapitre IV. C'est une démarche qui apporte rapidement des résultats significatifs.

Elle montre cependant certaines limites, tous les verrous aux performances ne pouvant pas être abordés de cette manière. Des études précises sont nécessaires devant des problèmes trop complexes ou lorsqu'une compréhension fine des phénomènes en jeu fait défaut. Une approche plus orientée recherche prend alors tout son sens et nous a permis de mettre à niveau nos connaissances quant à l'exécution précise du code. Cette approche doit être réalisée tout en gardant la maîtrise du contexte d'exécution pour mettre en évidence des problèmes qui pourront ensuite être résolus, potentiellement via des solutions relevant de l'ingénierie. C'est le cas dans notre étude des schémas de communication qui explore des possibilités fournies par la librairie

MPI ou de la congestion du réseau d'InterConnect. Une démarche orientée recherche est nettement plus compliquée à mettre en œuvre et peut aboutir à un faible gain en performances, mais reste la seule possibilité pour dépasser certaines limites.

ANNEXE

A

Réduction du nombre de communications

1 Frontières ouvertes

1.1 Traitement des frontières ouvertes

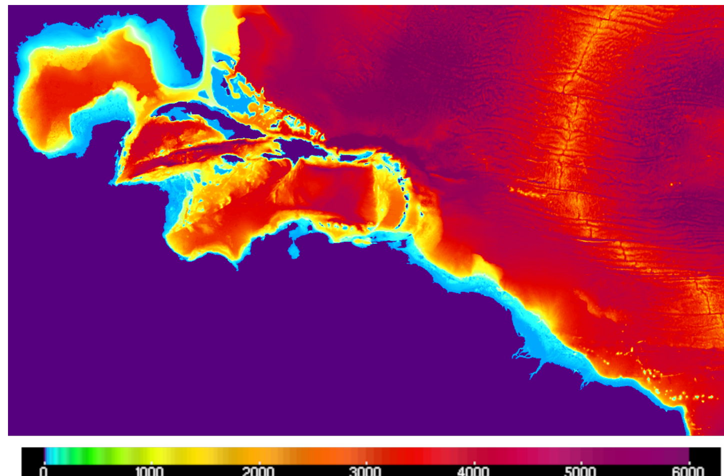


FIGURE I.1 – Bathymétrie (m) de la région couverte par la simulation : l'océan au large de l'Amérique centrale. Les continents sont représentés en violet.

NEMO offre la possibilité de ne simuler qu'une partie de l'océan. Sur la figure I.1 l'océan atteint le bord du domaine au nord, à l'est et au sud, alors que, dans la réalité, il se prolonge dans ces trois directions. Dans ce cas, la simulation doit permettre l'échange de quantités au travers des frontières au bord du domaine de simulation comportant de l'océan. Il faut donc, d'une part, des données sur l'état de l'océan de l'autre côté de la frontière, provenant par exemple d'une autre simulation à plus grande échelle, et, d'autre part, un schéma capable de faire sortir des quantités arrivant vers la frontière sans les réfléchir tout en assurant la compatibilité de la simulation avec les données extérieures.

Un tel schéma peut par exemple être construit à partir de la condition de Neumann $\frac{\partial \phi}{\partial n} = 0$, où n est la normale sortante au domaine et ϕ la variable sur laquelle on applique la condition. C'est une condition simple d'utilisation parce qu'elle ne nécessite pas de données extérieures.

Dans de nombreux autres cas la condition aux frontières est basée sur la condition de Sommerfeld

$$\frac{\partial \phi}{\partial t} + c\frac{\partial \phi}{\partial n} = 0 \tag{I.1}$$

Où c est la vitesse de l'onde incidente.

La condition est appliquée sur l'intérieur du sous-domaine, puis une phase de communication actualise les valeurs des halos.

NEMO permet deux types de frontières ouvertes : les frontières ouvertes rectilignes et les frontières ouvertes non structurées. Comme les frontières ouvertes rectilignes le long des bords du domaine sont beaucoup plus courantes et plus faciles à traiter, nous les examinerons en premier.

1.1.1 Les frontières ouvertes rectilignes le long des bords du domaine

La figure I.2.a montre une représentation schématique de la configuration BENCH avec des frontières ouvertes rectilignes de chaque côté qui sera utilisée pour expliquer les optimisations effectuées dans cette partie du code. La structure du code de NEMO exige que les domaines soient bordés par des points terrestres (en marron) dans toutes les directions, sauf lorsque des conditions cycliques sont appliquées. Les quatre frontières ouvertes (bandes rouges de chaque côté du domaine) sont donc situées à côté des points terrestres, sur les avant-dernières lignes et colonnes du domaine global.

Considérons un sous-domaine MPI situé à l'est du domaine global, représenté par le carré rouge sur la figure I.2.a et détaillé sur I.2.b. Avant l'optimisation, le traitement des frontières ouvertes n'était effectué que dans le domaine intérieur (bandes rouges sur les cellules bleues) et une phase de communication était utilisée pour mettre à jour la valeur sur les cellules fantômes (bandes rouges sur les cellules blanches). Dans le cas d'une frontière longitudinale rectiligne avec l'extérieur du domaine de calcul à l'est, $\frac{\partial \phi}{\partial n}|_{(x,y)}$ sera calculé en un point (x_i, y_i) par $\frac{\phi(x_{i-1},y_i)-\phi(x_i,y_i)}{\Delta_{x_{i,i-1}}}$ où $\Delta_{x_{i,i-1}}$ est la distance entre x_i et x_{i-1}. Lorsque ces conditions sont appliquées, seules les valeurs définies aux points orthogonaux à la frontière ouverte sont nécessaires, alors

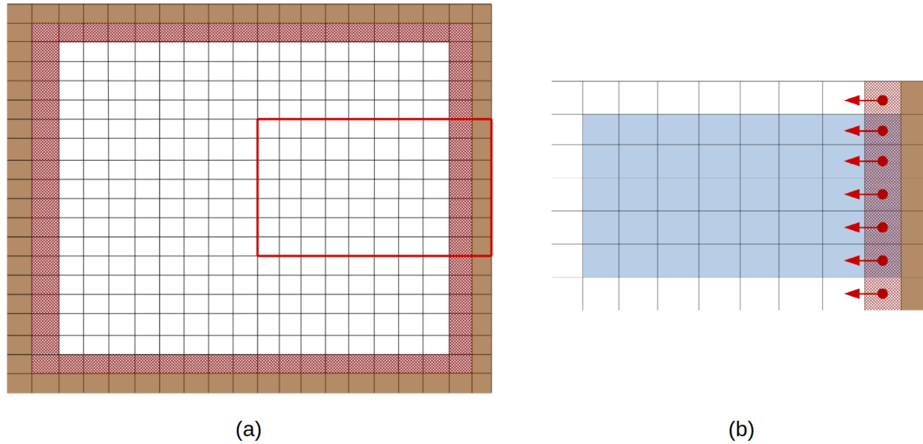
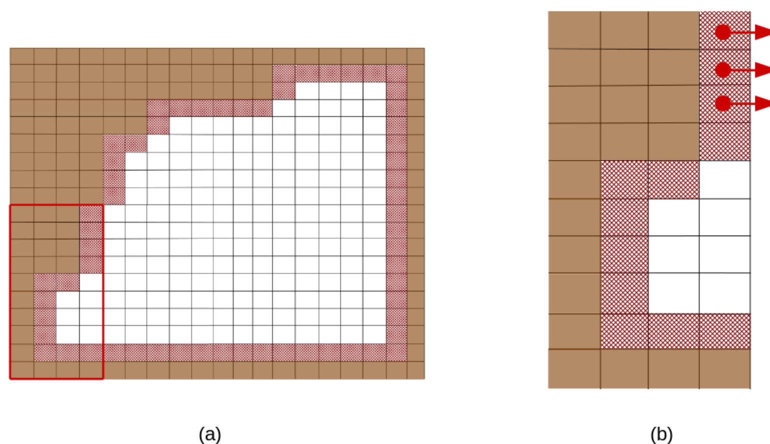


FIGURE I.2 – À gauche, une configuration avec des frontières ouvertes rectilignes, la ligne rouge délimite un sous-domaine MPI possible détaillé à droite. Les cellules marron sont des cellules terrestres et les cellules rouges hachurées sont des cellules à frontière ouverte. À droite, les cellules du domaine intérieur sont en bleu, les points rouges marquent les points T constituant la frontière ouverte ((x_i, y_i)) et les flèches rouges les points orthogonaux à la frontière ((x_{i-1}, y_i)) utilisés dans le calcul de $\frac{\partial \phi}{\partial n}|_{(x,y)=(x_i,y_i)}$.

que ces points se trouvent à l'intérieur du sous-domaine MPI, même lorsque (x_i, y_i) se trouve sur une cellule fantôme (bandes rouges sur les cellules blanches sur la figure I.2.b). De plus, dans le code, les champs sont toujours mis à jour sur les cellules fantômes avant que les conditions aux limites ouvertes ne soient appliquées, de sorte que le champ entier sur le sous-domaine MPI est correctement défini et peut être utilisé. Le calcul des conditions aux limites est donc possible sur l'ensemble du domaine, y compris sur les cellules fantômes. Aucune mise à jour de la communication n'est nécessaire. Lorsque des frontières ouvertes rectilignes le long des bords du domaine sont utilisées, cette optimisation permet de se débarrasser de toutes les communications liées au calcul des frontières ouvertes.

1.2 Les frontières ouvertes non structurées

Les frontières ouvertes non structurées permettent à l'utilisateur de définir n'importe quelle forme de frontière. La figure I.3.a montre un exemple d'une telle frontière ouverte définie à côté de points terrestres. Une utilisation possible d'une frontière ouverte non structurée est la suivante : un utilisateur pourrait souhaiter que la simulation n'inclue pas les points océaniques lorsque l'océan est trop peu profond et qu'elle définisse à la place une frontière ouverte non structurée délimitant une zone de faible profondeur (par exemple à proximité d'une île). Les points océa-



(a) (b)

FIGURE I.3 – À gauche, une configuration avec des frontières ouvertes rectilignes sur les côtés "est" et "sud" et des frontières ouvertes non structurées sur les côtés "ouest" et "nord". La ligne rouge délimite un sous-domaine MPI possible détaillé à droite. Les cellules brunes sont des cellules terrestres et les cellules rouges hachurées sont des cellules de frontière ouverte. Les points rouges marquent certains points T de la frontière ouverte $((x_i, y_i))$ et les flèches rouges les points orthogonaux à la frontière utilisés dans le calcul de $\frac{\partial \phi}{\partial n}|_{(x,y)=(x_i,y_i)}$.

niques peu profonds seront définis comme des points terrestres dans la définition du domaine, mais l'échange de masses d'eau et de propriétés sera possible à travers la frontière ouverte non structurée.

Selon la décomposition MPI, les cellules des frontières ouvertes peuvent se retrouver sur des cellules fantômes et faire face à l'extérieur du sous-domaine MPI, ce qui rend impossible le calcul direct de la condition limite. Par exemple, dans le sous-domaine MPI de la figure I.3.b, le calcul de la condition limite $\frac{\partial \phi}{\partial n}|_{(x,y)=(x_i,y_i)}$ sur certains points (x_i, y_i) mis en évidence par une flèche rouge nécessiterait la valeur de ϕ sur un point $((x_{i+1}, y_i))$ à l'extérieur du sous-domaine MPI. Notez que les frontières ouvertes rectilignes peuvent potentiellement être définies n'importe où dans le domaine (et pas seulement le long des bords du domaine). Dans ce cas, il est également possible qu'une frontière ouverte rectiligne soit juste tangente à la décomposition du domaine MPI. Dans ce cas rare, l'application des conditions de frontière ouverte nécessiterait également une communication MPI. Ces cas rares sont en fait traités de la même manière que pour les frontières ouvertes non structurées et sont donc automatiquement inclus dans la procédure que nous avons mise en œuvre pour les frontières ouvertes non structurées.

Nous avons choisi de détecter les points où le calcul direct (c'est-à-dire sans phase de communication) sera impossible pendant l'initialisation du modèle et de ne déclencher les communications que pour les sous-domaines MPI où au moins un tel point est présent. Cela permet à l'optimisation précédente d'être compatible avec les frontières ouvertes non structurées. Les points situés dans un coin d'une frontière ouverte non structurée requièrent également une attention particulière lors du suivi des points susceptibles de nécessiter une phase de communication. En effet, sur un coin extérieur, plusieurs points peuvent être considérés comme orthogonaux à la frontière ouverte. Le choix des points voisins impliqués dans ce calcul nous indiquera si le traitement du coin nécessite une communication MPI ou non. Lors de l'examen du traitement des points d'angle dans l'ancienne version de NEMO, nous avons réalisé que la méthode choisie dans certains cas ne garantissait pas les propriétés de symétrie (une symétrie de réflexion pourrait modifier les résultats). Nous avons donc décidé de corriger d'abord ce problème dans l'application physique de la condition de Neumann des points d'angle avant de trouver et de lister ceux qui nécessitent une communication MPI. Cette première étape est détaillée dans le paragraphe suivant même si, formellement, il ne s'agit pas d'une optimisation de performance.

L'application de la condition de Neumann, $\frac{\partial \phi}{\partial n} = 0$, à un point de frontière ouverte équivaut à fixer ce point à la valeur d'un (ou à la valeur moyenne de plusieurs) de ses voisins qui sont orthogonaux à la frontière ouverte. La méthode choisie doit présenter des propriétés de symétrie de réflexion et de rotation et permettre de fixer le point de la frontière ouverte à la valeur la plus réaliste possible. La méthode utilisée est illustrée sur la figure I.4 où les lignes de contour d'un champ ϕ sont en bleu avec la ligne de contour $\phi = 0$ passant par le point T d'une cellule de frontière ouverte (point rouge) sur un coin extérieur de la frontière ouverte. Les lignes de contour sont droites et le champ augmente linéairement dans les directions diagonales ou orthogonales. Les flèches rouges indiquent le meilleur choix pour la condition de Neumann. Dans la figure I.4, le meilleur choix consiste à prendre la moyenne des valeurs des points disponibles les plus proches. Ici, l'application de la condition de Neumann se fait en fixant $\phi(x_i, y_i)$ à $\frac{\phi(x_{i+1}, y_i) + \phi(x_i, y_{i-1})}{2}$. En effet, si un seul de ces deux points était utilisé, il n'y aurait pas de bonnes propriétés de symétrie (cas

1, 3 et 4) et si le point supérieur droit était également pris en compte dans la moyenne, le résultat serait un peu meilleur dans le cas 1 pour un champ non linéaire mais moins bon dans les cas 2, 3 et 4.

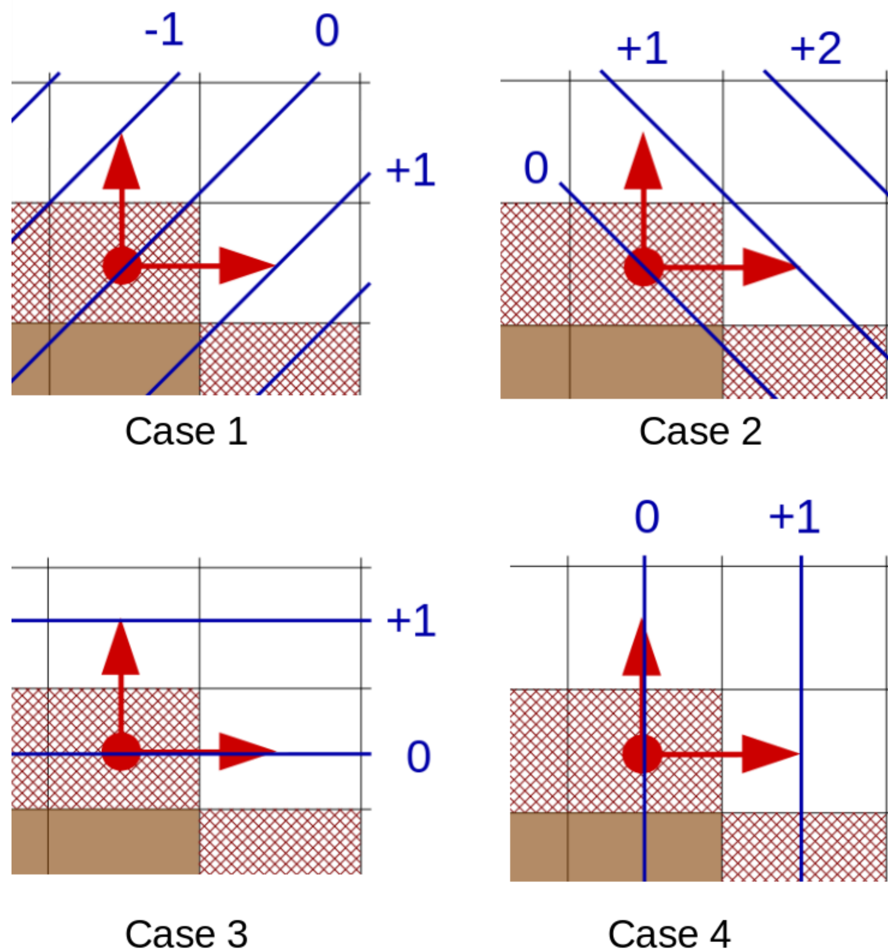


FIGURE I.4 – Les cellules brunes sont des cellules terrestres et les cellules rouges hachurées sont des cellules à frontière ouverte. Les points rouges marquent les points de la frontière ouverte et les flèches rouges les meilleurs points à utiliser pour calculer la condition de Neumann, les lignes bleues sont les lignes de contour de ϕ avec la valeur de la ligne de contour surlignée en bleu.

En utilisant la même méthode, nous avons finalement résumé toutes les conditions de Neumann et les contributions de leurs voisins en 5 cas présentés dans la figure I.5. Toutes les autres dispositions possibles sont des rotations de l'un de ces 5 cas. Grâce à cette classification, nous avons ensuite pu déterminer, dès la phase d'initialisation du modèle, le schéma de communication requis par le traitement des conditions de Neumann. Sur la base de cette information, nous pouvons limiter le nombre de phases de communication à leur strict minimum même dans le cas de frontières ouvertes non structurées.

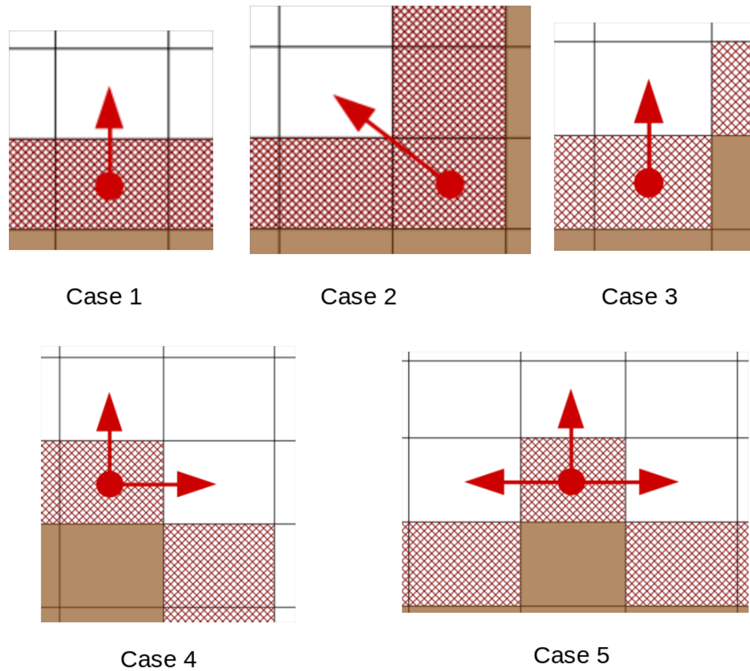


FIGURE I.5 – Les cellules marron sont des cellules terrestres et les cellules rouges hachurées sont des cellules de frontière ouverte. Les points rouges marquent les points de la frontière ouverte et les flèches rouges les points utilisés dans le calcul de $\frac{\partial \phi}{\partial n}|_{(x,y)=(x_i,y_i)}$ pour la condition de Neumann.

1.3 Contrôle de l'optimisation

Les tests de reproductibilités ne contrôlant que peu les frontières ouvertes aux géométries particulières, il a été décidé, pour valider cette optimisation d'un nouveau cas test (figure I.6). Cette configuration présente deux frontières ouvertes aux géométries irrégulières et permet de tester les différentes implémentations avec un faible coût de calcul.

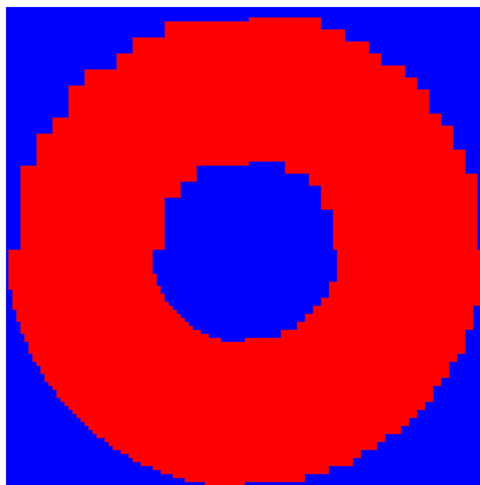


FIGURE I.6 – Exemple d'un domaine de simulation, en rouge l'océan et en bleu l'extérieur du domaine. La frontière ouverte n'est ici ni rectiligne ni sur le bord du domaine

Les communications liées au traitement des frontières ouvertes sont supprimées dans presque tous les cas. Pour certaines simulations cela correspond à une diminution presque de moitié du nombre de communications ou a environ 350 communications en moins par pas de temps.

2 Calcul du gradient de pression de surface

L'intégration sur la verticale des équations du mouvement a pour solution le mode barotrope $\overline{U_h}$, c'est-à-dire la vitesse intégrée sur la verticale. La hauteur du niveau de la mer η et le mode barotrope $\overline{U_h}$, vérifient les équations couplées suivantes :

$$\begin{cases} \frac{\partial \overline{U_h}}{\partial t} = -f\mathbf{k} \times \overline{U_h} - g\nabla_h \eta - \frac{c_b^U}{H+\eta}\overline{U_h} + \overline{G} \\ \frac{\partial \eta}{\partial t} = -\nabla[(H+\eta)\overline{U_h}] + P - E \end{cases} \tag{I.2}$$

Où f est la fréquence de Coriolis, g l'accélération de la pesanteur, η la hauteur de la surface de l'océan par rapport à son niveau de référence, H est la profondeur de l'océan par rapport au niveau de référence de la hauteur de la mer, $\overline{G}$ est un forçage comprenant par exemple la contribution de l'atmosphère, $\mathbf{k}$ est le vecteur unitaire vertical, c_b^U est un coefficient prenant en compte les frottements au fond de l'océan, P les précipitations et E l'évaporation.

Les ondes de gravité sont solutions de ces équations, or ces ondes ont une vitesse de $\sqrt{g(H+\eta)}$ de l'ordre de plusieurs centaines de $m.s^{-1}$. La résolution de ces équations nécessite donc l'utilisation d'un petit sous-pas de temps.

Dans la plupart des configurations basées sur NEMO, y compris dans le cas test BENCH et BENCH_OCE, le terme de gradient de pression de surface dans l'équation pronostique de la dynamique océanique est calculé en utilisant une formulation de fractionnement temporel "*Forward-Backward*" (Shchepetkin & McWilliams, 2005). À chaque pas de temps n du modèle, une dyna-

mique $2D$ simplifiée est résolue à un sous-pas de temps beaucoup plus petit Δt_* résultant en un sous-pas de temps m, avec m allant de 1 à $M(\approx 50)$. Cette dynamique $2D$ sera ensuite moyennée pour obtenir le terme de gradient de pression de surface.

Dans la version précédente de NEMO, avant notre optimisation, chaque sous-étape temporelle effectue les calculs suivants :

$$\begin{cases} \eta^{m+\frac{1}{2}} = (\frac{3}{2}+\beta)\eta^m - (\frac{1}{2}+2\beta)\eta^{m-1} + \beta\eta^{m-2} \\ \overline{U_h}^{m+\frac{1}{2}} = (\frac{3}{2}+\beta)\overline{U_h}^m - (\frac{1}{2}+2\beta)\overline{U_h}^{m-1} + \beta\overline{U_h}^{m-2} \\ \tilde{U_h}^{m+\frac{1}{2}} = D^{m+\frac{1}{2}}\overline{U_h}^{m+\frac{1}{2}}\Delta e \\ \color{blue}{Communication\ 1} \quad \text{sur } \tilde{U_h}^{m+\frac{1}{2}} \\ \eta^{m+1} = \eta^m - \Delta t_*[div(\tilde{U_h}^{m+\frac{1}{2}}) + P - E] \\ \color{blue}{Communication\ 2} \quad \text{sur } \eta^{m+1} \\ \eta' = \delta\eta^{m+1} + (1-\delta-\gamma-\epsilon)\eta^m + \gamma\eta^{m-1} + \epsilon\eta^{m-2} \\ \overline{U_h}^{m+1} = \frac{1}{D^{m+1}}[D^m\overline{U_h}^m + \Delta t_*\big((1-r)g\ grad_x(\eta') - D^{m+\frac{1}{2}} f k \times \overline{U_h}^{m+\frac{1}{2}} + \overline{G}\big)] \\ \color{blue}{Communication\ 3} \quad \text{sur } \overline{U_h}^{m+1} \end{cases}$$

Avec $\tilde{U_h}$ le flux, $D^m = H + \eta^m$ la hauteur de la colonne d'eau, Δe la longueur de la cellule, k le vecteur unitaire vertical et β, r, δ, γ et ϵ sont des constantes.

NEMO utilise une grille Arakawa C décalée, c'est-à-dire que certaines variables sont évaluées à différents endroits. Les vitesses zonales sont évaluées au milieu des bords est de la grille (points U sur la figure I.7), les vitesses méridiennes au milieu des bords "nord" de la grille (points V) et la hauteur de la surface de la mer au centre de la grille (points T).

En raison de cette caractéristique, des interpolations spatiales sont parfois nécessaires pour obtenir des variables à un autre endroit que celui où elles ont été initialement définies. Par exemple, $\eta^{m+\frac{1}{2}}$ doit être interpolé de T à U et V points pour être utilisé dans l'équation $\tilde{U_h}^{m+\frac{1}{2}} = D^{m+\frac{1}{2}}\overline{U_h}^{m+\frac{1}{2}}\Delta e = (H+\eta^{m+\frac{1}{2}})\overline{U_h}^{m+\frac{1}{2}}\Delta e$. Étant donné que l'interpolation nécessite des

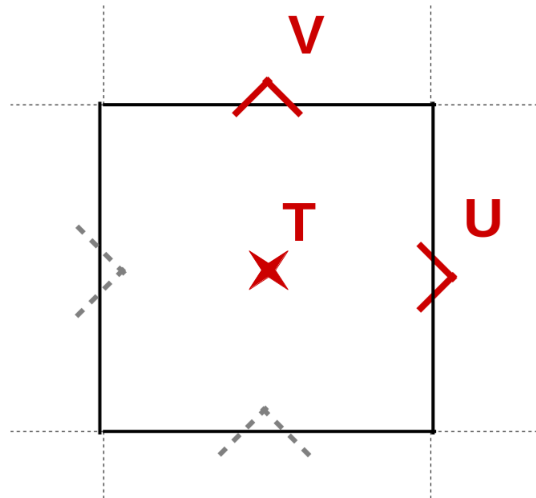


FIGURE I.7 – Discrétisation de champs 2D sur une grille Arakawa C décalée.

points adjacents de part et d'autre des cellules de la grille, $\eta^{m+\frac{1}{2}}$ ne peut pas être interpolée directement sur les cellules fantômes des points U orientaux (flèches grises sur la figure I.8). De même, $\eta^{m+\frac{1}{2}}$ ne peut être interpolé sur les cellules fantômes des points V du nord. Il en résulte que $\tilde{U_h}^{m+\frac{1}{2}}$ n'est pas directement calculé sur les cellules fantômes des points U à l'est et V au nord. Communication 1 est donc utilisée pour mettre à jour $\tilde{U_h}^{m+\frac{1}{2}}$ sur ces cellules fantômes.

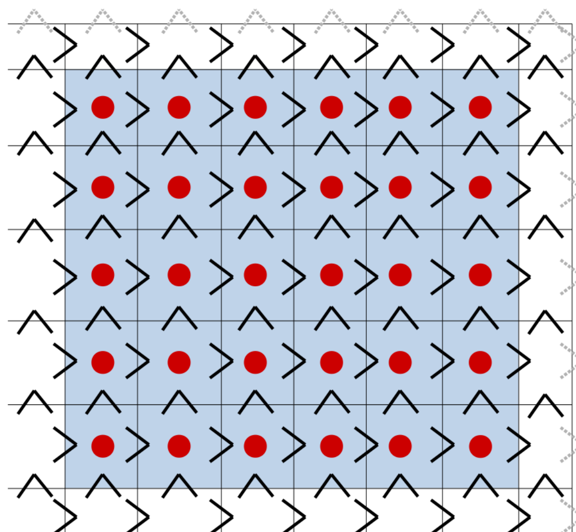


FIGURE I.8 – Le sous-domaine MPI est délimité par la grille, l'intérieur du sous-domaine MPI est démarqué en bleu tandis que les cellules fantômes sont en blanc. Les flèches noires indiquent les points U et V où $\eta^{m+\frac{1}{2}}$ (et donc $\tilde{U_h}^{m+\frac{1}{2}}$) peuvent être calculés directement et les flèches grises les points où ils ne peuvent pas être calculés sans communication. Les points rouges indiquent les points T où $div(\tilde{U_h}^{m+\frac{1}{2}})$ peut être calculé directement.

Le calcul de $div(\tilde{U_h}^{m+\frac{1}{2}})$ défini aux points T nécessite les valeurs de $\tilde{U_h}^{m+\frac{1}{2}}$ aux points U et V adjacents. Elle ne peut donc pas être effectuée sur les cellules fantômes "ouest" et "sud", Communication 2 met à jour le champ sur ces cellules. De même, $grad_x(\eta')$ ne peut être calculé sur l'ensemble du sous-domaine MPI, d'où la Communication 3.

Un examen attentif de cet algorithme montre cependant que cette séquence de communication peut être améliorée. Comme le montre la figure I.8, le calcul de $div(\tilde{U_h}^{m+\frac{1}{2}})$ (points rouges) défini aux points T n'exige que des valeurs correctes aux quatre points U et V adjacents (flèches noires). Les valeurs de $\eta^{m+\frac{1}{2}}$ aux points U et V des cellules fantômes "nord" et "est" ne sont pas nécessaires au calcul de $div(\tilde{U_h}^{m+\frac{1}{2}})$ à l'intérieur du sous-domaine MPI. Communication 1 sur $\tilde{U_h}^{m+\frac{1}{2}}$ peut donc être retardée et regroupée avec Communication 2 sur η^{m+1}. On notera que la communication sur $\tilde{U_h}^{m+\frac{1}{2}}$ ne peut pas être totalement supprimée car la variable est également utilisée à d'autres fins qui ne sont pas détaillées ici.

Suite à cette amélioration, le nombre de communications par pas de temps partiel dans la formulation de division temporelle a été réduit de 3 à 2, ce qui se traduit par une réduction de 135 (44%) à 90 (29%) communications par pas de temps dans la routine du gradient de pression de surface pour la configuration examinée, soit 45 communications par pas de temps.

ANNEXE

B

Algorithme de synchronisation des horloges

Les horloges des différents nœuds ne sont pas parfaitement synchronisées, c'est-à-dire que deux processus sur deux nœuds différents prenant une mesure de l'horloge au même instant n'obtiendront pas la même mesure. Pour avoir une vue cohérente de l'exécution sur l'ensemble des cœurs, les mesures issues du profilage doivent être modifiées afin de prendre en compte le décalage d'horloge. Le travail présenté ici vise à mesurer précisément ce décalage.

Plusieurs algorithmes ont été essayés et sont présentés tour à tour.

1 Synchronisation par barrière MPI globale

Le premier algorithme utilisé pour chercher à mesurer le décalage d'horloge est simple. Il consiste en un appel sur chaque processus à la routine *MPI_Barrier* directement suivi d'un appel à *system_clock*. La routine *MPI_Barrier* sert à synchroniser les processus et, supposément, faire que les processus exécutent l'instruction de mesure de l'horloge *system_clock* simultanément. Le décalage entre horloges est alors mesuré par le décalage entre les différentes mesures de *system_clock*. Pour limiter la variabilité, cette séquence *MPI_Barrier* puis *system_clock* est répétée 5 fois et moyennée.

Comme la mesure du décalage est prise par tous les processus de chaque nœud, on peut observer la variation de prise de la mesure pour chacun de ces processus. Si tous les processus sortent de la routine *MPI_Barrier* au même instant, alors la mesure de l'horloge devrait fournir des valeurs identiques pour des processus d'un même nœud. Or ce n'est pas le cas, ces mesures présentent une variabilité avec un écart type entre $10^{-4}s$ et $10^{-5}s$ (voir figure II.1).

Cette variabilité est due au fait que l'algorithme interne à *MPI_Barrier* libère les cœurs à des moments différents. En effet, cette routine a simplement pour objectif de libérer les processus une fois que tous sont entrés dans la routine et non de s'assurer que les processus quittent la routine au même instant. Cet algorithme doit être amélioré.

2 Synchronisation par barrière MPI 2 à 2

L'algorithme présenté ici vise à corriger le problème de l'utilisation de *MPI_Barrier* de manière globale. Les processus sortent de la routine à des instants différents car la routine doit libérer de nombreux processus ce qui a lieu nécessairement en plusieurs temps. Si seulement deux processus exécutent *MPI_Barrier*, il est plus probable que l'instruction synchronise effectivement les processus impliqués.

Algorithm 4 Algorithme de synchronisation par barrière MPI 2 à 2

```
Sélection d'un cœur de référence commun : cœur_ref
for Tous les autres cœur_loop do
    if cœur==cœur_loop .OR. cœur==cœur_ref then
        CALL MPI_Barrier(cœur_ref, cœur_loop)
        CALL system_clock()
    end if
end for
```

La méthode qui en résulte est présentée à l'algorithme 4 exécuté par tous les processus de la simulation. Un cœur de référence est choisi pour tous les processus de la simulation $cœur_ref$ puis une boucle est effectuée sur tous les cœurs de la simulation $cœur_loop$. Si le cœur exécutant cet algorithme $cœur$ est $cœur_loop$ ou $cœur_ref$, alors la mesure du décalage est faite. On obtient ainsi, pour chaque cœur de la simulation, une estimation du décalage d'horloge. L'écart type de la mesure au sein d'un nœud peut à nouveau être évalué, il est présenté figure II.1. Le résultat est cette fois-ci beaucoup plus régulier, l'écart type est entre $10^{-7}s$ et $10^{-6}s$.

Le problème de l'utilisation d'une barrière globale n'est pas seulement qu'elle introduit un bruit considérable, mais surtout que la mesure est faussée. On estime la disparité entre les méthodes en utilisant les deux algorithmes lors d'une seule exécution. La figure II.2 démontre que la différence entre les méthodes est considérable : autour des ordres de grandeur de $10^{-4}s$ et $10^{-3}s$, bien plus élevés que certains phénomènes à fine échelle que nous étudions autour de $10^{-5}s$ et $10^{-4}s$.

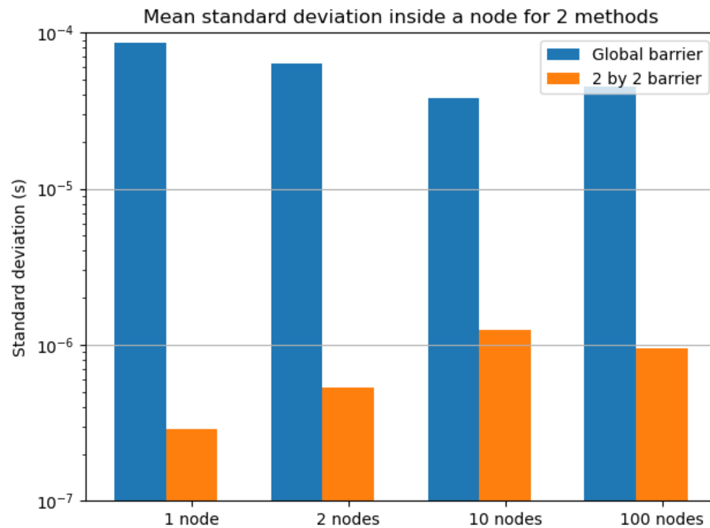


FIGURE II.1 – Écart type de la mesure de l'horloge au sein d'un nœud moyenné sur l'ensemble des nœuds de la simulation. En bleu les résultats pour la méthode de synchronisation par barrière MPI globale et en orange pour la méthode par barrière MPI 2 à 2.

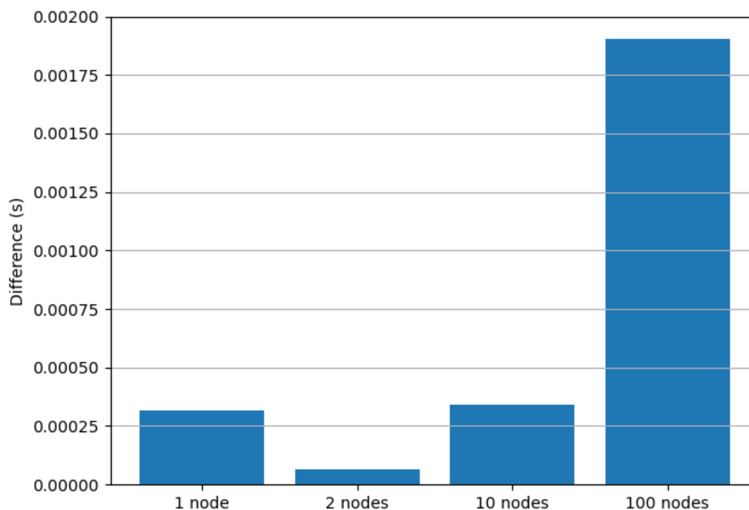


FIGURE II.2 – Différence moyenne de la mesure du décalage d'horloge entre les deux méthodes de synchronisation par barrière MPI globale.

La différence importante de résultats entre les deux méthodes montre qu'au moins l'une d'entre elles n'est pas suffisamment fiable. Par construction, la méthode utilisant la barrière MPI globale est nécessairement moins précise, son utilisation est donc à proscrire. Néanmoins, cela ne démontre pas que la méthode de synchronisation par barrière 2 à 2 est fiable.

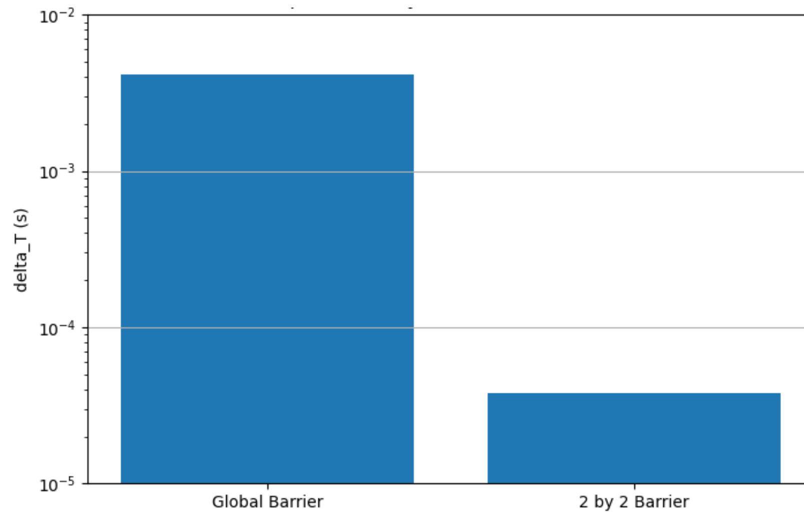


FIGURE II.3 – Variation des résultats sur 100 nœuds entre la méthode globale et 2 à 2 à gauche et entre deux mesures de la méthode 2 à 2 utilisant des cœurs de références différents à droite.

La figure II.3 cherche à quantifier la fiabilité de la méthode de barrière 2 à 2. Pour cela on compare les résultats obtenus en mesurant dans la même exécution le décalage en choisissant dans la méthode de synchronisation deux cœurs différents. On fait de même pour la méthode avec barrière globale à titre de comparaison.

Pour la méthode de barrière globale on note $T^{glo}_{0\to k}$ le décalage mesuré de l'horloge entre les processus 0 et k. Pour la méthode utilisant une barrière 2 à 2, la comparaison est plus difficile. En effet, le décalage mesuré est celui par rapport à un cœur de référence k_1 ou k_2 qui n'est pas le cœur 0 comme dans la méthode par barrière globale. Il faut donc convertir le décalage mesuré par rapport au processus 0 : $T^{2b2,k_1}_{0\to k} = T^{2b2}_{0\to k_1} + T^{2b2}_{k_1\to k}$

Soit k_{tot} le nombre total de cœurs de la simulation, sur la figure II.3, la différence moyenne entre les résultats de la méthode par barrière globale et 2 à 2 est affichée à gauche :

$$\frac{1}{k_{tot}}\sum_{k}|T^{glo}_{0\to k} - T^{2b2,k_1}_{0\to k}|$$

Et à droite la différence entre les résultats de la méthode 2 à 2 pour deux cœurs de références différents k_1 et k_2 :

$$\frac{1}{k_{tot}}\sum_{k}|T^{2b2,k_2}_{0\to k} - T^{2b2,k_1}_{0\to k}|$$

La différence entre deux mesures de la méthode de synchronisation est de l'ordre de $10^{-5}s$ sur 100 nœuds, c'est une première estimation de la précision de la méthode. Or c'est une incertitude trop élevée pour l'étude de phénomènes qui peuvent être de même durée que la variation de la mesure. Par conséquent, une autre méthode doit être utilisée.

3 Synchronisation par chronologie des communications

La méthode choisie pour augmenter la précision de la synchronisation utilise une caractéristique du schéma de communication. Elle est utilisable dans le cadre du profilage au chapitre II à la section 3.3 où les instants de début et la fin de la routine de communication sont mesurés. Les résultats qu'elle fournit ne sont pas utilisables, mais son exploration a été instructive.

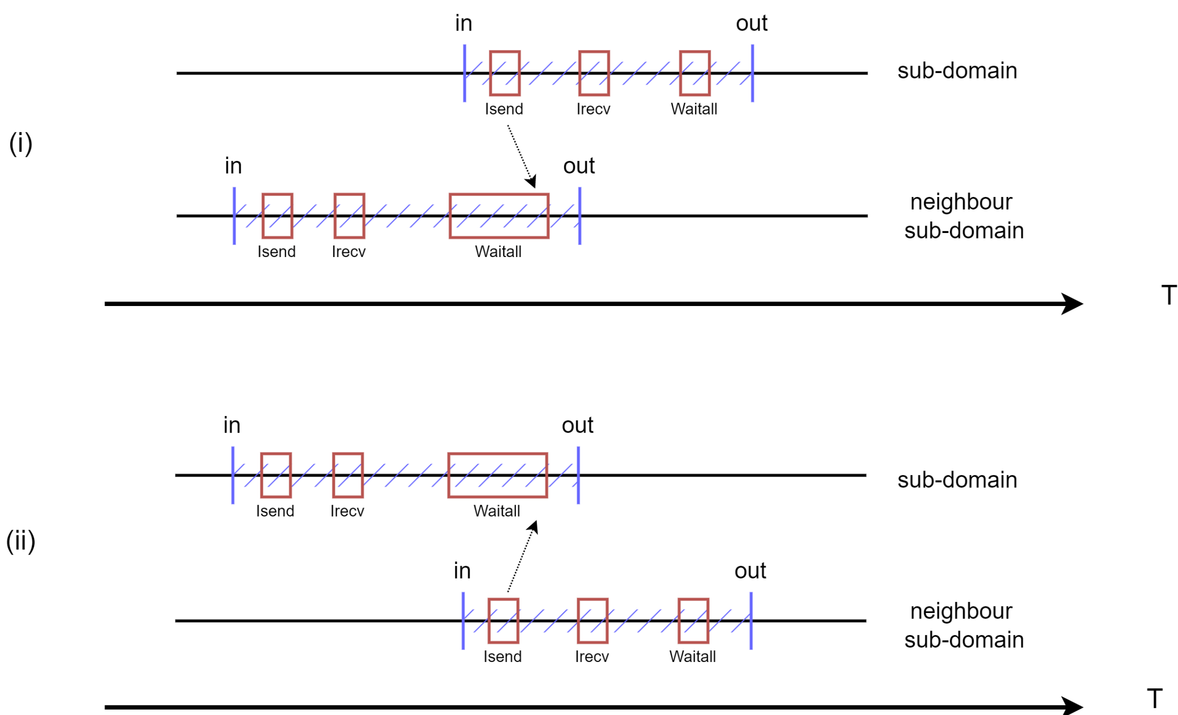


FIGURE II.4 – Principe de la stratégie de synchronisation de l'horloge par chronologie des communications. Les axes horizontaux sont des axes temporels dans deux situations (i) et (ii) pour un sous-domaine et un voisin. Les sections délimitées en bleu et indiquées in et out pour l'entrée et la sortie de la routine de communication.

Pour que le processus d'un sous-domaine sorte de la routine de communication, il faut que son voisin avec lequel il échange des informations ait envoyé des données et soit donc entré dans la routine (cas (ii) sur la figure II.4). De même le processus du sous-domaine voisin ne sort

de la routine de communication que si le processus est déjà entré dans cette routine (cas (i)). Cela peut se traduire dans les équations suivantes :

(i) $T^{in_k} \leq T_{nei}^{out_k}$

(ii) $T^{out_k} \geq T_{nei}^{in_k}$

Avec T le temps réel d'un processus et T_{nei} celui de son voisin, T^{in_k} et T^{out_k} sont les instants d'entrée et de sortie d'une communication k.

En pratique le temps mesuré est perturbé par le décalage d'horloge T^{ref}, donc $\tilde{T}^{in_k/out_k} = T^{in_k/out_k} + T^{ref}$. En utilisant les équations (i) et (ii) on a :

$$\tilde{T}^{in_k} - \tilde{T}_{nei}^{out_k} + T_{nei}^{ref,min} \leq T^{ref} \leq \tilde{T}^{out_k} - \tilde{T}_{nei}^{in_k} + T_{nei}^{ref,max}$$

En faisant varier les communications k, on peut déduire l'intervalle le plus petit :

$$T^{ref,min} \leq T^{ref} \leq T^{ref,max}$$

Cet intervalle est déduit de l'intervalle calculé sur le sous-domaine voisin $T_{nei}^{ref,min}$ et $T_{nei}^{ref,max}$. Il faut donc choisir un processus quelconque pour servir de référence et les intervalles de tous les autres sont déduits par rapport à lui de proche en proche comme indiqué sur la figure II.5.

La variabilité obtenue avec cette méthode est de l'ordre de $10^{-5}s$ et $10^{-4}s$ même sur un seul nœud et augmente avec le nombre de nœuds utilisés. Elle n'atteint donc pas la précision escomptée.

Cependant l'utilisation de cette méthode m'a permis de constater l'existence d'un phénomène de décalage de l'horloge non seulement entre les nœuds mais aussi au cours du temps. En effet, suivant que l'on calcule l'intervalle en utilisant les communications du début de l'exécution ou celles de la fin, un résultat très différent est obtenu. Ceci indique que le décalage d'horloge varie au cours du temps. J'ai confirmé ce résultat avec deux mesures de décalages avec la méthode de synchronisation par barrière MPI 2 à 2 au début et à la fin d'une exécution qui montrent

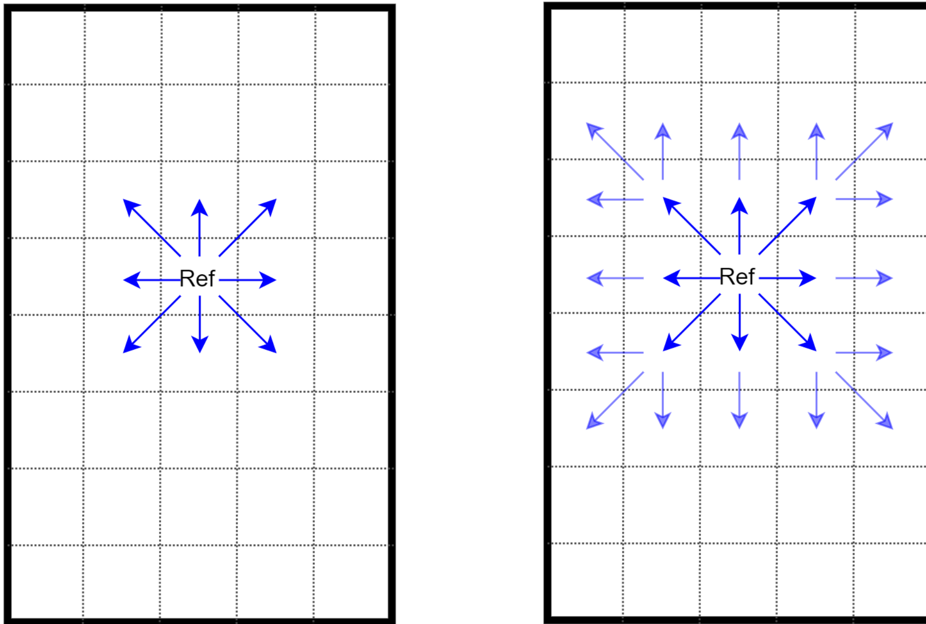


FIGURE II.5 – Schéma de propagation de la mesure du décalage d'horloge. Le sous-domaine indiqué Ref est pris comme référence et le décalage de tous les autres sont calculés par rapport au processus de ce sous-domaine. Les flèches bleues indiquent le calcul de l'intervalle en partant d'un sous-domaine voisin, avec à gauche la première étape de propagation et à droite la seconde.

un décalage de l'horloge d'un nœud par rapport à un nœud de référence de $\approx 10^{-5}s$ à chaque seconde d'exécution. Ce phénomène et l'ordre de grandeur du décalage est confirmé dans Jones et Koenig (2010) et Hunold et Carpen-Amarie (2015).

4 Synchronisation par aller-retour

Les travaux décrits dans Jones et Koenig (2010) fournissent un algorithme de synchronisation de l'horloge de différents nœuds qui prend en compte le décalage d'horloge au cours du temps. Pour cela l'algorithme procède en deux temps. Premièrement, la mesure du décalage d'horloge est mesuré par rapport à un processus de référence pour tous les processus et cela une multitude de fois. On obtient alors pour chaque processus un nuage de points avec en ordonnée le décalage par rapport à la référence et en abscisse l'instant de mesure de ce décalage. La deuxième étape consiste à réaliser une régression linéaire sur ce nuage de points pour déduire à la fois le

décalage et l'évolution de ce décalage au cours du temps. Dans notre cas, la régression linéaire n'est pas fiable et l'on n'observe presque pas de corrélation entre l'abscisse et l'ordonnée. Cette méthode doit donc être adaptée pour être utilisable ici.

Certaines techniques utilisées dans Jones et Koenig (2010) peuvent tout de même être exploitées dans notre cas. Tout d'abord, la méthode pour réaliser au même instant une instruction de mesure de l'horloge est révisée, elle commence par une phase d'initialisation présentée figure II.6. Elle consiste en une temps médian d'aller-retour de tous les processus avec le processus de référence. Tour à tour, chaque processus effectue une mesure de l'horloge, puis un envoi vers le processus de référence qui réceptionne puis envoie vers le processus de départ. Enfin une dernière mesure de l'horloge permet de déterminer la durée de la séquence. La séquence d'aller-retour est ensuite répétée, puis on en calcule la durée médiane.

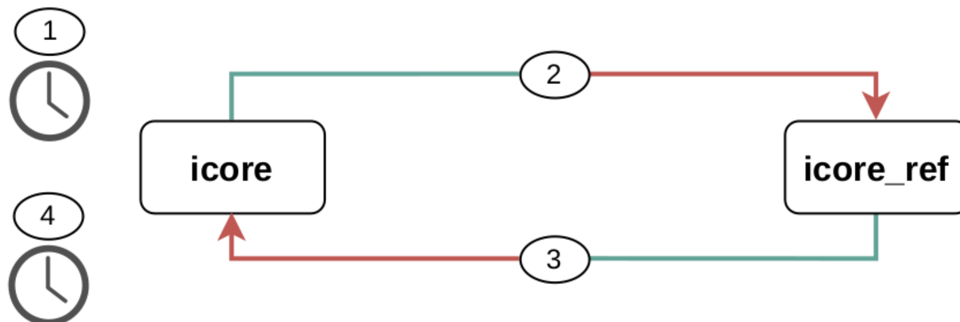


FIGURE II.6 – Schéma des quatre étapes de l'initialisation de la routine de mesure simultanée de l'horloge par aller-retour. Le temps de communication "aller-retour" est mesuré. Les deux processus impliqués sont associés à deux cœurs, un cœur quelconque indiqué **icore** et un cœur de références indiqué **icore_ref**. Les étapes 1 et 4 sont des mesures de l'horloge et 2 et 3 sont des communications.

De plus, un algorithme de mesure simultanée de l'horloge est proposé dans Jones et Koenig (2010), il est présenté figure II.7. Il n'est cette fois-ci pas fondé sur des instructions de barrières MPI, mais sur des envois et réceptions bloquants. Un processus fait un envoi vers le processus de référence qui, une fois le message réceptionné, effectue une mesure d'horloge indiqué 2 sur la figure. Le processus de référence envoie un message vers le processus de départ qui mesure ensuite son temps d'horloge indiqué 4. La mesure 4 est donc faite plus tard que la 2, pour simuler

une mesure simultanée, on soustrait de la mesure 4 la moitié du temps d'aller-retour médian entre les deux processus mesurés en amont. Les mesures ainsi produites sont sauvegardées et seront ultérieurement écrites dans un fichier.

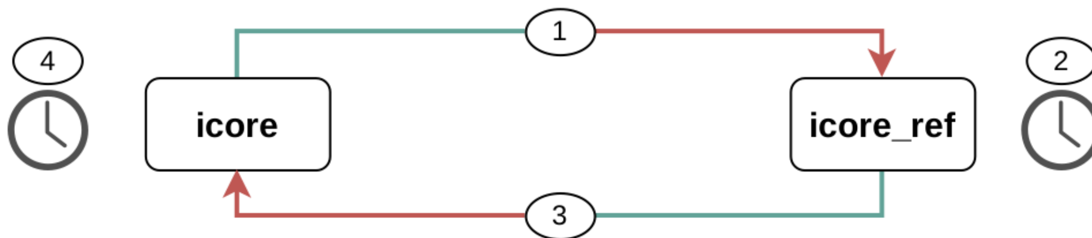


FIGURE II.7 – Schéma des quatre étapes de la routine de mesure simultanée de l'horloge par aller-retour. Les deux processus impliqués sont associés à deux cœurs, un cœur quelconque indiqué **icore** et un cœur de références indiqué **icore_ref**.

L'algorithme de mesure simultanée (indiqué *mesure_simultanee*) de l'horloge est utilisé $N_exchange$ fois sur chaque processus avec le processus de référence, voir figure 5. Cette procédure est répétée N_rep fois, dans Jones et Koenig (2010) cela est utilisé pour obtenir des mesures étalées temporellement et faciliter la régression linéaire. Ici, cela permet simplement d'avoir plus de mesures et de s'assurer qu'aucun processus prendra toutes ces mesures à un moment où la machine subit des perturbations. On prend $N_exchange = 50$ et $N_rep = 10$ Une fois que les mesures simultanées de l'horloge sont écrites, des soustractions donnent une série d'estimations du décalage et la médiane est calculée. C'est cette valeur qui est utilisée pour synchroniser les processus.

Algorithm 5 Algorithme de synchronisation par aller-retour

```
for i_rep ← 0 to N_rep − 1 do
    for cœur_loop dans tous les cœurs do
        if cœur==cœur_loop .OR. cœur==cœur_ref then
            for j_rep ← 0 to N_exchange − 1 do
                CALL mesure_simultanee( cœur_loop, cœur_ref )
            end for
        end if
    end for
end for
```

La figure II.8 quantifie la différence moyenne de cette méthode avec la méthode de barrière 2 à 2 dont on prend la médiane sur autant de mesures. La différence est importante à partir de 10 nœuds, entre 10^{-5} et $10^{-4}s$, c'est-à-dire du même ordre de grandeur que certaines communications. Encore une fois, cela ne dit rien de la qualité des méthodes mais prouve simplement que l'une des deux n'est pas suffisamment précise.

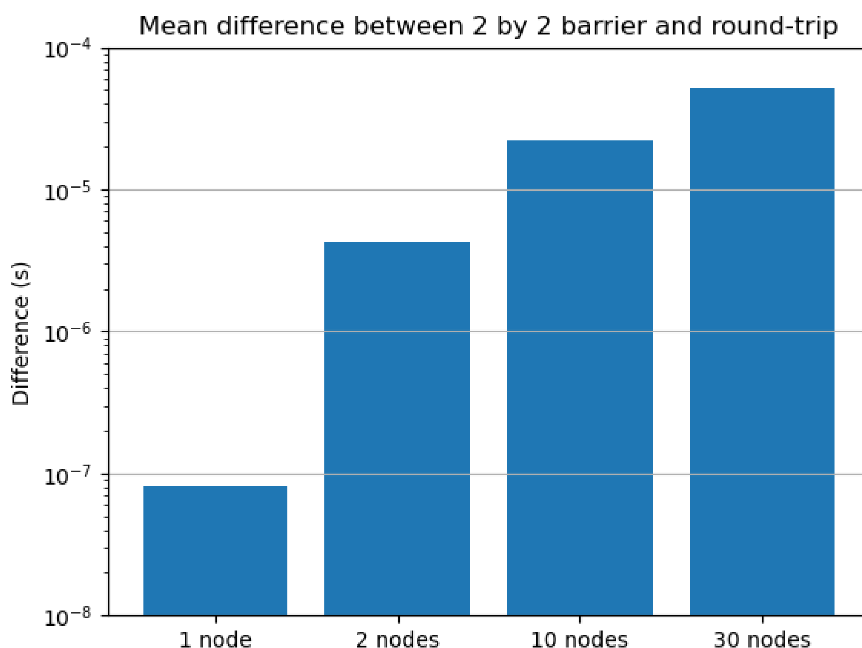


FIGURE II.8 – Différence moyenne de la mesure du décalage d'horloge entre la méthode de synchronisation par barrière MPI 2 à 2 et par aller-retour.

La consistance des résultats obtenus est examinée figure II.9, la mesure du décalage est quasiment identique pour différents cœurs du même nœud avec un écart type de $\approx 2.10^{-7}$ sur 30 nœuds pour la méthode par aller-retour, alors que l'écart type pour la méthode 2 à 2 est de $\approx 10^{-6}$. Ce résultat est plus que suffisant pour les phénomènes que l'on souhaite mesurer.

La fiabilité des méthodes est évaluée figure II.10. Pour cela on regarde comment évoluent les résultats au sein d'une même exécution en changeant les processus de références. Cela permet d'évaluer l'incertitude des méthodes. Ici encore, les résultats sont convertis en décalage par

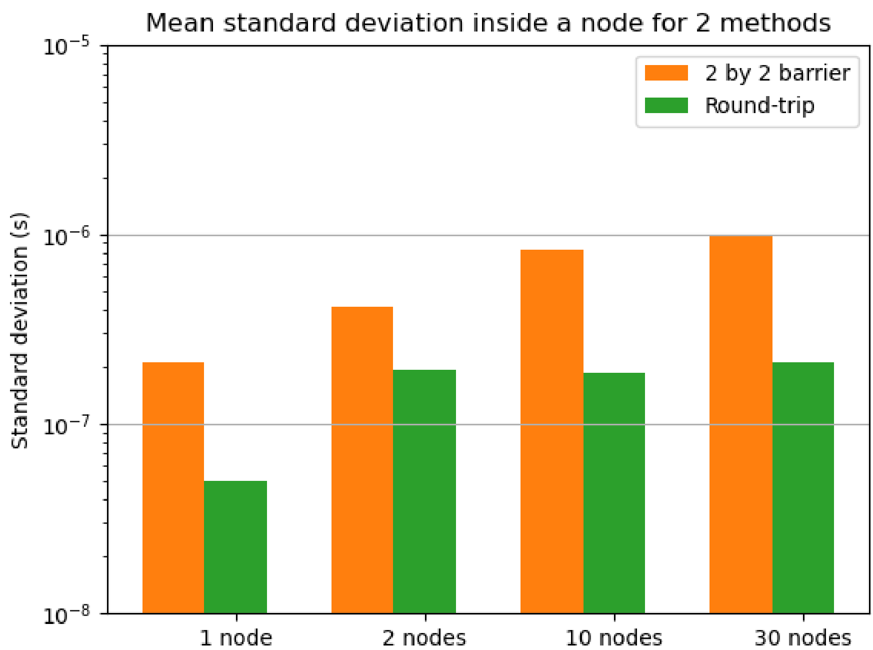


FIGURE II.9 – Écart type de la mesure de l'horloge au sein d'un nœud moyenné sur l'ensemble des nœuds de la simulation. En orange pour la méthode par barrière MPI 2 à 2 et en vert pour la méthode par aller-retour.

rapport au cœur 0 pour être comparés. Alors que la méthode par barrière affiche une incertitude de l'ordre de $10^{-5}s$ dès 10 nœuds la rendant ainsi inutilisable. La méthode par aller-retour reste de l'ordre de $10^{-6}s$ même sur 30 nœuds.

Cette précision est suffisante pour analyser qualitativement les communications dans NEMO, par exemple pour la propagation d'une instabilité. Cependant, une estimation précise de l'avance ou du retard de l'ensemble des processus de la simulation selon le schéma de communication utilisé est pour l'instant à proscrire. La capacité des schémas de communications à gérer l'asynchronisme entre processus ne peut pas être quantifiée et servir à évaluer les performances des schémas. En effet, quantifier l'asynchronisme sur une simulation revient a sommer la valeur absolue de la différence d'horloge entre processus à un moment donné de l'exécution. Dans ces conditions, l'erreur de désynchronisation des horloges est sommée, une mesure précise de l'asynchronisme est donc pour l'instant impossible.

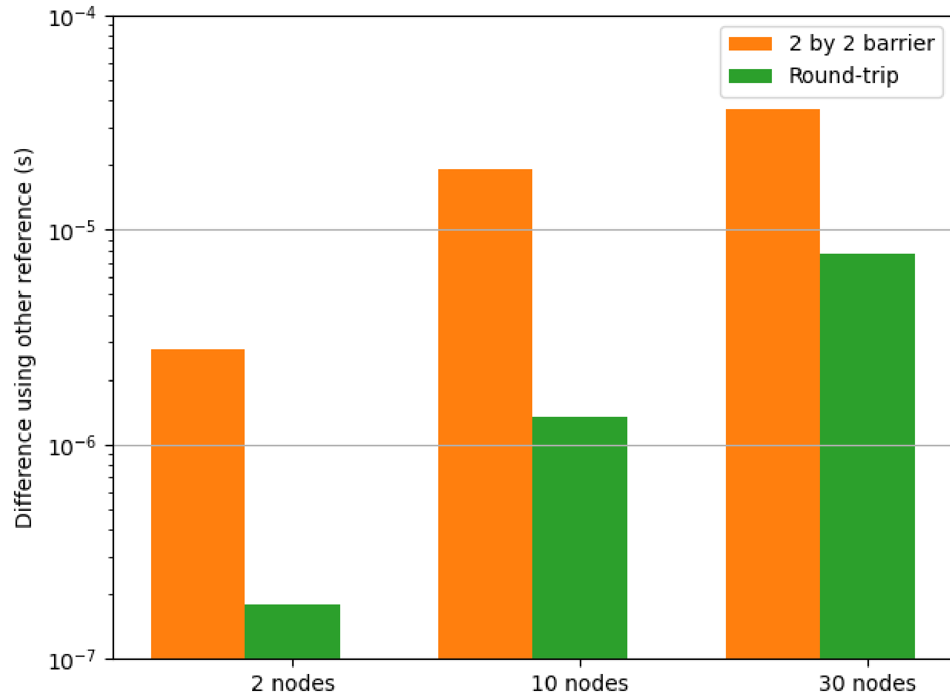


FIGURE II.10 – Variation des résultats entre la méthode 2 à 2 en orange et par aller-retour en vert entre deux mesures utilisant des cœurs de références différents.

Rappelons aussi que du fait de l'évolution au cours du temps du décalage d'horloge, l'asynchronisme des processus ne peut être estimé que à proximité de l'exécution de la méthode par aller-retour.

Références

18.2.85. MPI_dist_graph_create_adjacent Open MPI 5.0.x documentation. (s. d.). Consulté le 2023-12-19, sur https://docs.open-mpi.org/en/v5.0.x/man-openmpi/man3/MPI_Dist_graph_create_adjacent.3.html

AGRIF_demo.jpg on Users/ModelInterfacing/AGRIF – Attachment – NEMO. (s. d.). Consulté le 2023-12-16, sur http://forge.ipsl.jussieu.fr/nemo/attachment/wiki/Users/ModelInterfacing/AGRIF/AGRIF_DEMO.jpg

Annex II : Models. (2023). In Intergovernmental Panel on Climate Change (IPCC) (Ed.), Climate Change 2021 – The Physical Science Basis : Working Group I Contribution to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change (pp. 2087–2138). Cambridge : Cambridge University Press. Consulté le 2023-12-21, sur https://www.cambridge.org/core/books/climate-change-2021-the-physical-science-basis/annex-ii-models/9D2628824C65B0F31A0351D05CB12167 doi: 10.1017/9781009157896.016

Biastoch, A., Böning, C. W., & Lutjeharms, J. R. E. (2008, novembre). Agulhas leakage dynamics affects decadal variability in Atlantic overturning circulation. Nature, 456(7221), 489–492. Consulté le 2023-07-28, sur https://www.nature.com/articles/nature07426 doi: 10.1038/nature07426

Bienz, A., Olson, L. N., & Gropp, W. D. (2019, octobre). Node-Aware Improvements to Allreduce. arXiv. Consulté le 2023-09-15, sur http://arxiv.org/abs/1910.09650

Blayo, E., & Debreu, L. (1999, juin). Adaptive Mesh Refinement for Finite-Difference Ocean Models : First Experiments. Journal of Physical Oceanography, 29(6), 1239–1250. Consulté le 2023-09-28, sur http://journals.ametsoc.org/doi/10.1175/1520-0485(1999)029<1239:AMRFFD>2.0.CO;2 doi: 10.1175/1520-0485(1999)029<1239:AMRFFD>2.0.CO;2

Crupnicoff, D., Das, S., & Zahavi, E. (s. d.). Deploying Quality of Service and Congestion Control in InfiniBand-based Data Center Networks.

Debreu, L., Vouland, C., & Blayo, E. (2008, janvier). AGRIF : Adaptive grid refinement in Fortran. Computers & Geosciences, 34(1), 8–13. Consulté le 2023-06-28, sur https://www.sciencedirect.com/science/article/pii/S009830040700115X doi: 10.1016/j.cageo.2007.01.009

Dongarra, J., Hempel, R., Hey, A., & Walker, D. (1994, septembre). A draft standard for message passing in a distributed memory environment.. Consulté le 2023-12-09, sur https://www.semanticscholar.org/paper/A-draft-standard-for-message-passing-in-a-memory-Dongarra-Hempel/7055146d6ef55bf543e37b7e2527155e59a541ab#citing-papers

Embedded zooms - NEMO release-4.2.1 documentation. (2023). Consulté le 2023-12-12, sur https://sites.nemo-ocean.io/user-guide/zooms.html#overview

Etiemble, D. (2018, mars). 45-year CPU evolution : one law and two equations. arXiv. Consulté le 2023-08-30, sur http://arxiv.org/abs/1803.00254 doi: 10.48550/arXiv.1803.00254

Fieux, M. (2020). L'océan planétaire. Les presses de l'ENSTA.

General introduction : AGRIF software AGRIF 1.10.0 documentation. (s. d.). Consulté le 2023-12-13, sur https://agrif.imag.fr/Gen_intro.html

Gran, E. G., Reinemo, S.-A., Lysne, O., Skeie, T., Zahavi, E., & Shainer, G. (2012, mai). Exploring the Scope of the InfiniBand Congestion Control Mechanism. In 2012 IEEE 26th International Parallel and Distributed Processing Symposium (pp. 1131–1143). Shanghai, China : IEEE. Consulté le 2023-12-13, sur http://ieeexplore.ieee.org/document/6267917/ doi: 10.1109/IPDPS.2012.104

Gusat, M., Craddock, D., Denzel, W., Engbersen, T., Ni, N., Pfister, G., ... Duato, J. (2005, août). Congestion control in InfiniBand networks. In 13th Symposium on High Performance Interconnects (HOTI'05) (pp. 158–159). Consulté le 2023-12-13, sur https://ieeexplore.ieee.org/document/1544592 (ISSN : 2332-5569) doi: 10.1109/CONECT.2005.14

Hardware Event-based Sampling Collection. (s. d.). Consulté le 2023-12-09, sur https://www.intel.com/content/www/us/en/docs/vtune-profiler/user-guide/2023-0/hw-event-based-sampling-collection.html

Hunold, S., & Carpen-Amarie, A. (2015, septembre). On the Impact of Synchronizing Clocks and Processes on Benchmarking MPI Collectives. In Proceedings of the 22nd European MPI Users' Group Meeting (pp. 1–10). New York, NY, USA : Association for Computing Machinery. Consulté le 2023-12-11, sur https://dl.acm.org/doi/10.1145/2802658.2802662 doi: 10.1145/2802658.2802662

Intergovernmental Panel on Climate Change (IPCC). (2023). Climate Change 2021 – The Physical Science Basis : Working Group I Contribution to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change. Cambridge : Cambridge University Press. Consulté le 2023-12-21, sur https://www.cambridge.org/core/books/climate-change-2021-the-physical-science-basis/415F29233B8BD19FB55F65E3DC67272B doi: 10.1017/9781009157896

Irrmann, G., Masson, S., Maisonnave, , Guibert, D., & Raffin, E. (2022, février). Improving ocean modeling software NEMO 4.0 benchmarking and communication efficiency. Geoscientific Model Development, 15(4), 1567–1582. Consulté le 2023-08-31, sur https://gmd.copernicus.org/articles/15/1567/2022/ doi: 10.5194/gmd-15-1567-2022

Jones, T., & Koenig, G. A. (2010, octobre). A Clock Synchronization Strategy for Minimizing Clock Variance at Runtime in High-End Computing Environments. In 2010 22nd International Symposium on Computer Architecture and High Performance Computing (pp. 207–214). Consulté le 2023-12-11, sur https://ieeexplore.ieee.org/document/5644949 (ISSN : 1550-6533) doi: 10.1109/SBAC-PAD.2010.33

Jouanno, J., Sheinbaum, J., Barnier, B., Molines, J.-M., Debreu, L., & Lemarié, F. (2008, janvier). The mesoscale variability in the Caribbean Sea. Part I : Simulations and characteristics with an embedded model. Ocean Modelling, 23(3), 82–101. Consulté le 2023-07-31, sur https://www.sciencedirect.com/science/article/pii/S1463500308000486 doi: 10.1016/j.ocemod.2008.04.002

June 2023 | TOP500. (2023). Consulté le 2023-12-09, sur https://www.top500.org/lists/top500/2023/06/

Madec, G., Bell, M., Blaker, A., Bricaud, C., Bruciaferri, D., Castrillo, M., ... Wilson, C. (2023, juillet). NEMO Ocean Engine Reference Manual. Consulté le 2023-07-27, sur https://zenodo.org/record/8167700 doi: 10.5281/zenodo.8167700

Maisonnave, E., & Masson, S. (2019). NEMO 4.0 performance : how to identify and reduce unnecessary communications (Rapport technique). TR/CMGC/19/19, CECI, UMR CERFACS/CNRS No5318, France.

Masson-Delmotte, V., P. Zhai, H. O. P., Roberts, D., Skea, J., Shukla, P. R., & al, e. (2018).

Meurdesoif, Y. (2018). Xios fortran reference guide. December.

Mozdzynski, G. (2012). RAPS Introduction. Consulté sur https://www.ecmwf.int/node/14020

Müller, A., Deconinck, W., Kühnlein, C., Mengaldo, G., Lange, M., Wedi, N., ... New, N. (2019). The ESCAPE project : Energy-efficient Scalable Algorithms for Weather Prediction at Exascale. Geoscientific Model Development, 12(10), 4425–4441. Consulté sur https://gmd.copernicus.org/articles/12/4425/2019/ doi: 10.5194/gmd-12-4425-2019

Petton, S., Garnier, V., Caillaud, M., Debreu, L., & Dumas, F. (2023, février). Using the two-way nesting technique AGRIF with MARS3D V11.2 to improve hydrodynamics and estimate environmental indicators. Geoscientific Model Development, 16(4), 1191–1211. Consulté le 2023-07-28, sur https://gmd.copernicus.org/articles/16/1191/2023/ doi: 10.5194/gmd-16-1191-2023

Rupp, K. (2020, juillet). 48 Years of Microprocessor Trend Data. Consulté le 2023-08-30, sur https://zenodo.org/record/3947824 doi: 10.5281/zenodo.3947824

Shamis, P., Venkata, M. G., Lopez, M. G., Baker, M. B., Hernandez, O., Itigin, Y., . . . Bouteiller, A. (2015, août). UCX : An Open Source Framework for HPC Network APIs and Beyond. In 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects (pp. 40–43). Consulté le 2023-12-13, sur https://ieeexplore.ieee.org/document/7312665 (ISSN : 2332-5569) doi: 10.1109/HOTI.2015.13

Shchepetkin, A. F., & McWilliams, J. C. (2005, janvier). The regional oceanic modeling system (ROMS) : a split-explicit, free-surface, topography-following-coordinate oceanic model. Ocean Modelling, 9(4), 347–404. Consulté le 2023-08-30, sur https://www.sciencedirect.com/science/article/pii/S1463500304000484 doi: 10.1016/j.ocemod.2004.08.002

Talandier, C., Deshayes, J., Treguier, A. M., Capet, X., Benshila, R., Debreu, L., . . . Madec, G. (2014, avril). Improvements of simulated Western North Atlantic current system and impacts on the AMOC. Ocean Modelling, 76, 1–19. Consulté le 2023-07-27, sur https://www.sciencedirect.com/science/article/pii/S1463500313002217 doi: 10.1016/j.ocemod.2013.12.007

Tintó, O., Castrillo, M., Acosta, M. C., Mula-Valls, O., Sanchez, A., Serradell, K., . . . Doblas-Reyes, F. J. (2019). Finding, analysing and solving MPI communication bottlenecks in Earth System models. Journal of Computational Science, 36, 100864. Consulté sur http://www.sciencedirect.com/science/article/pii/S1877750318304150 doi: https://doi.org/10.1016/j.jocs.2018.04.015

Unified Communication - X Framework Library. (s. d.). Consulté le 2023-12-13, sur https://docs.nvidia.com/networking/display/HPCXv215/Unified+Communication+-+X+Framework+Library

Van Sebille, E., Sprintall, J., Schwarzkopf, F. U., Sen Gupta, A., Santoso, A., England, M. H., ... Böning, C. W. (2014, février). Pacific-to-Indian Ocean connectivity : Tasman leakage, Indonesian Throughflow, and the role of ENSO. Journal of Geophysical Research : Oceans, 119(2), 1365–1382. Consulté le 2023-07-31, sur http://doi.wiley.com/10.1002/2013JC009525 doi: 10.1002/2013JC009525