Université de Lille 1, Sciences et Technologies

PRESENTATION DES TRAVAUX

En vue de l'obtention de

L'HABILITATION A DIRIGER LES RECHERCHES

Christophe CALVIN

PARALLELISATION EFFICACE DE CODES SCIENTIFIQUES POUR LA SIMULATION NUMERIQUE APPLIQUEE AUX SYSTEMES NUCLEAIRES

DATE DE SOUTENANCE : LE 14 JUIN 2013 A LA MAISON DE LA SIMULATION

RAPPORTEURS :

PROF. M. DAYDE – ENSEEIHT

PROF. M. SATO – UNIV. DE TSUKUBA

PROF. W. JALBY – UNIV. DE VERSAILLES – ST QUENTIN

EXAMINATEURS :

D. BOUCHE – DIRECTEUR DE RECHERCHES – CEA

PROF. N. EMAD – UNIV. DE VERSAILLES – ST QUENTIN

T. NKAOUA – DIRECTION. DE LA RECHERCHE ET DE L'INNOVATION – AREVA

PROF. S. PETITON (GARANT) – UNIV. DE LILLE1, SCIENCES ET TECHNOLOGIES

Remerciements

Je tiens tout d'abord à remercier Messieurs les professeurs Dayde, Jalby et Sato d'avoir bien voulu rapporter mon mémoire. Mes remerciements vont également au membre du jury, Mme Emad, MM. Bouche et Nkaoua qui ont accepté de juger mon travail.

Un remerciement tout particulier à Serge Petiton, non seulement pour m'avoir convaincu de soutenir cette habilitation, mais également pour toutes les discussions de travail et autres que nous avons eu toutes ces années.

Tous ces travaux résumés dans ce mémoire n'auraient pu voir le jour sans les contributions des étudiants que j'ai encadré, notamment mes thésards passés, Eli LAUCOIN et Jérôme DUBOIS et actuels France BOILLOD-CERNEUX et Fan YE.

Ces travaux de recherche menés dans le cadre de mon activité professionnelle au CEA ont également été supportés par différents projets comme PRACE et le projet franco-japonais FP3C. J'ai donc également une pensée pour mes collègues français et étrangers avec qui j'ai eu l'opportunité de travailler dans le cadre de ces projets.

Le CEA et la Direction de l'Energie Nucléaire entretient des collaborations particulières avec ses homologues étrangers et notamment le Département de l'Energie Américain (DOE). J'ai pu ainsi avoir l'opportunité de travailler et de rencontrer nombre de chercheurs de ces grands laboratoires et les travaux et discussions ont toujours été enrichissants. Ces collaborations ont, je l'espère, vocation à être entretenues et étendues.

Résumé

Les travaux exposés dans ce document retracent mes différentes activités de recherche pour la parallélisation efficace d'applications scientifiques dans le domaine de la simulation numérique appliquée aux systèmes nucléaires.

Différents thèmes seront abordés comme l'étude, la conception et l'implémentation d'architectures logicielles pour la parallélisation efficace d'applications de simulations numériques dans le domaine de la thermohydraulique, mais également des recherches sur la problématique de la prise en compte de maillages dynamiques et d'équilibrage de charge pour des applications en mécanique des fluides. Au-delà de la parallélisation d'applications elles-mêmes, des recherches de fond sur des algorithmes numériques parallèles ont été conduites tant sur des méthodes itératives de résolutions de systèmes linéaires que sur les méthodes de Krylov pour la recherche de valeurs propres. Ces méthodes numériques sont extrêmement utilisées dans les applications scientifiques du domaine nucléaire (résolution des équations de Navier-Stokes ou de l'équation du transport des neutrons) et permettent non seulement de progresser dans l'algorithmique numérique parallèle mais également de tester et comparer de nouvelles approches pour le calcul hautes performances tant du point de vue matériel, avec l'implémentation de ces noyaux de calculs sur accélérateurs (Cell, GPU, MIC) que du point de vue des modèles et langages de programmation (CUDA, OpenCL, langages PGAS, …).

Ce document comporte quatre parties.

Nous présentons dans une première partie les travaux de recherche qui ont été menés sur la conception et l'implémentation d'une architecture parallèle de code pour la simulation d'écoulements in stationnaires 3D dans des géométries complexes, le code TRIO-U. L'axe majeur qui a guidé ces travaux a été de concevoir une architecture permettant de paralléliser à moindre coût tous les nouveaux développements intégrés dans le code. Ainsi nous avons fourni des services et des mécanismes de gestion du parallélisme (communications, entrées/sorties, gestion des maillages, structures de données pour la gestion des maillages découpés, solveurs linéaires …) permettant au développeur de se concentrer sur l'implémentation de son modèle physique ou de sa méthode numérique sans se préoccuper de la parallélisation du nouveau module.

La prise en compte de maillages non constants au cours du temps (raffinement de maillages, zoom, multi-grilles géométriques …) ont nécessité non seulement des modifications dans la gestion du parallélisme mais également la prise en compte des problèmes d'équilibrage de la charge. Ces recherches, dans le cadre de la thèse de doctorat d'Eli LAUCOIN, ont permis de proposer des nouvelles méthodes de raffinement de maillage pour les équations de Navier-Stokes en parallèle.

A la suite des travaux réalisés dans le domaine de la mécanique des fluides, je me suis intéressé à un autre grand domaine de la simulation numérique pour les systèmes nucléaires qu'est la physique des réacteurs en général et la neutronique en particulier. C'est l'objet de la deuxième partie de ce mémoire. Je me suis notamment penché sur les enjeux de l'utilisation du calcul intensif dans le domaine de la simulation numérique pour les systèmes nucléaires afin d'identifier les apports du HPC dans le domaine, mais également des défis futurs à relever.

Ces études ont été déclinées de manière plus pratique pour la résolution de Boltzmann pour le transport des neutrons dans le cadre du code de simulation en neutronique déterministe APOLLO3®.
Dans le cadre de la thèse de doctorat de J. DUBOIS, nous nous sommes intéressés à la programmation des nouvelles architectures de calcul comme les accélérateurs (GPU) en portant un solveur de l'équation de transport sur GPU. Nous avons également travaillé sur la ré-intégration de ces développements dans un code à vocation industriel comme APOLLO3® et mesuré les impacts. Cela nous a permis également de proposer en 2010 un des 1ers solveurs neutroniques mixant parallélisation par décomposition de domaines et accélération GPU sur chacun des domaines.

L'évolution très rapide ces dernières années des architectures de calcul et les « révolutions » à venir pour atteindre l'exascale vont avoir des impacts majeurs sur les codes de simulations numériques et les algorithmes.
Nous présenterons dans une troisième partie l'étude plus approfondie de la programmation efficace de ces nouvelles architectures. Ces recherches se sont appliquées aux méthodes de Krylov pour la résolution de systèmes linéaires et de problèmes à valeurs propres qui sont très représentatifs des applications qui nous intéressent.
Nous avons ainsi abordé plusieurs thématiques majeures inhérentes à ces nouvelles architectures comme la précision arithmétique des accélérateurs de calcul, l'étude de méthodes parallèles adaptées aux machines massivement multi-cœurs homogènes et hétérogènes, les co-méthodes offrant des propriétés intéressantes pour l'ultra-scale, les techniques d'auto-tuning et de smart-tuning appliquées aux méthodes d'Arnoldi.

Enfin nous présenterons les travaux actuels et les perspectives de recherches envisagées.

Sommaire

Table des figures

Table des tableaux

1 INTRODUCTION

La simulation numérique appliquée aux systèmes nucléaires, à l'image d'autres domaines techniques et scientifiques, prend une part de plus en plus importante. Le calcul intensif de manière générale et le parallélisme en particulier apportent des opportunités supplémentaires aux ingénieurs, concepteurs et physiciens pour concevoir des systèmes nucléaires de plus en plus performants et sûrs.

Il y a encore une petite décennie, on pouvait considérer le parallélisme comme un outil nécessaire à l'obtention de performances supérieures pour certaines classes d'applications ou indispensable à l'atteinte de résultats scientifiques majeurs. Ce constat a désormais évolué avec l'apparition des processeurs multi-cœurs. Concrètement cela signifie qu'auparavant, l'évolution des processeurs (augmentation de la fréquence et de la complexité intrinsèque) était auto-suffisante pour accroître la puissance de calcul. Cela n'est plus possible pour des raisons physiques de dégagement de chaleur, de complexité de gravure mais également de coûts de développement et de production.
Cela signifie que l'époque où il suffisait d'attendre un à deux ans l'évolution des processeurs pour voir l'amélioration des performances de son application est révolue. Pire, nous entrons dans une ère où les performances de l'application vont diminuer de part la baisse des performances unitaires des composants du processeur (puissance du cœur de calcul, fréquence d'horloge, fréquence et débits mémoires …).
Cela est extrêmement visible sur l'architecture des supercalculateurs et il suffit pour cela de regarder le nombre moyens de cœurs de calcul embarqué dans ces machines. La Figure 1 illustre le nombre moyen de cœurs de calcul par processeurs de 1993 à nos jours pour les supercalculateurs. Cette tendance s'accélère avec les machines de classe pétaflopiques[1] et les futures machines exascales[2].
Nous reviendrons un peu plus tard dans ce document sur les tendances concernant les architectures de machines de calcul, mais les quelques ordres de grandeur qui suivent suffisent déjà à illustrer ce point. Dans les années 2010, le nombre moyen de cœurs dans un nœud de calcul était de l'ordre de quelques dizaines. Pour les machines de l'ordre de 100 PFlops vers 2015, ce nombre va être de l'ordre d'une centaine, pour prendre encore un ordre de grandeur pour l'Exaflops aux alentours de 2020.

[1] Pétfalops : 10^{15} opérations flottantes par seconde.
[2] Exaflops : 10^{18} opérations flottantes par seconde.

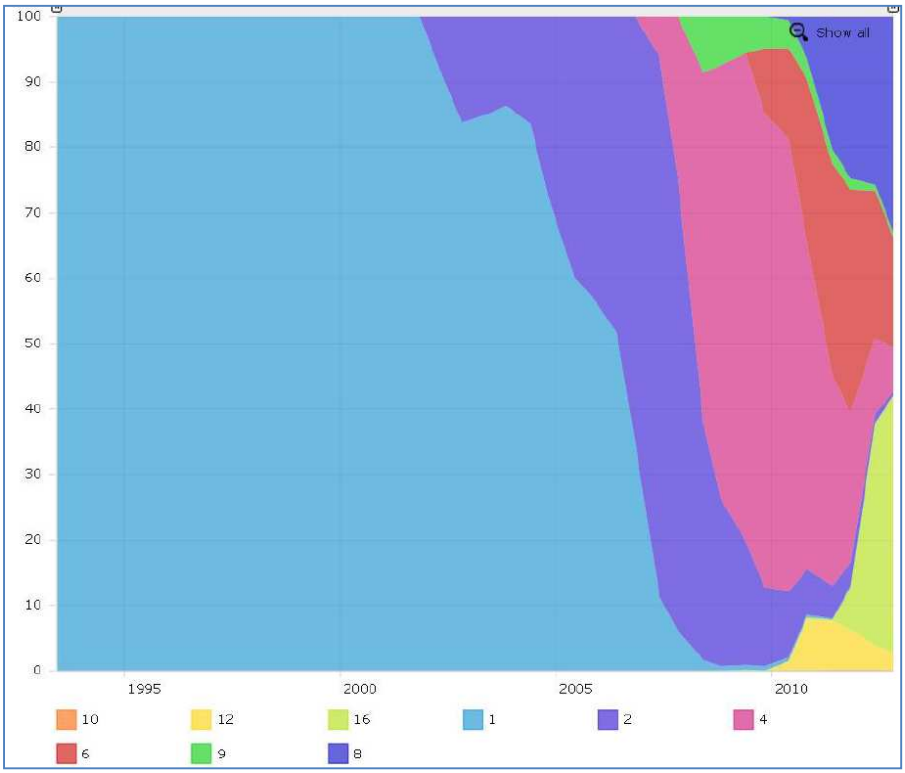


Figure 1: Evolution du nombre moyen de cœurs par processeur – Répartition en pourcentage dans les systèmes de calcul.

Depuis que le parallélisme est utilisé pour la simulation numérique, la recherche de l'efficacité de la parallélisation des applications est une constante et nécessite la combinaison de nombreux talents et compétences : physique, numérique, algorithmique et informatique.

Les travaux exposés dans ce document retracent les différentes activités de recherche dans le domaine de la parallélisation efficace d'applications scientifiques pour la simulation numérique dans le domaine du nucléaire.

Différents thèmes seront abordés comme l'étude, la conception et l'implémentation d'architectures logicielles pour la parallélisation efficace d'applications de simulations numériques dans le domaine de la thermohydraulique, mais également des recherches sur la problématique de la prise en compte de maillages dynamiques et d'équilibrage de charge pour des applications en mécanique des fluides. Au-delà de la parallélisation d'applications elles-mêmes, des recherches de fond sur des algorithmes numériques parallèles ont été conduites tant sur des méthodes itératives de résolutions de systèmes linéaires que sur les méthodes de Krylov pour la recherche de valeurs propres. Ces méthodes numériques sont extrêmement utilisées dans les applications scientifiques du domaine nucléaire (résolution des équations de Navier-Stokes [1] ou de l'équation du transport des neutrons [3]) et permettent non seulement de progresser dans l'algorithmique numérique

parallèle mais également de tester et comparer de nouvelles approches pour le calcul hautes performances tant du point de vue matériel, avec l'implémentation de ces noyaux de calculs sur accélérateurs (Cell, GPU, MIC) que du point de vue des modèles et langages de programmation (CUDA, OpenCL, langages PGAS, …).

Ce document comporte quatre parties. La première sera consacrée aux travaux réalisés sur la conception d'une architecture logicielle parallèle et l'utilisation de maillage dynamique pour une application de simulation numérique en mécanique des fluides.

Dans un second temps, nous retracerons les activités sur l'utilisation et le développement du calcul intensif pour la physique des réacteurs. Elles ont porté principalement sur la parallélisation de solveurs pour la résolution de l'équation du transport des neutrons [3].

Nous aborderons dans la troisième partie les travaux les plus récents effectués sur la programmation efficace des nouvelles architectures de calcul pour le petascale et au-delà. Cela couvre les aspects informatiques (nouveaux langages et environnements de programmation), algorithmiques et numériques. Ces recherches ont porté essentiellement sur les méthodes de Krylov pour la résolution de problèmes à valeurs propres [82, 87].

Enfin nous présenterons les travaux actuels et les perspectives de recherches envisagées.

Ces thématiques sont dans la continuité des recherches menées au cours de ma thèse de doctorat, où je me suis intéressé à la minimisation du sur-coût des communications dans la parallélisation des algorithmes numériques. La première explorée a consisté à optimiser des algorithmes de communications globales. Nous avons proposé notamment de nouveaux algorithmes pour réaliser une transposition de matrices carrées allouées par blocs, sur différentes topologies de réseaux d'interconnexion. Nous avons également étudié le problème de l'échange total. Ce schéma de communication se retrouve fréquemment dans les versions parallèles d'algorithmes numériques (comme dans l'algorithme du gradient conjugué). Nous avons proposé des algorithmes efficaces d'échange total pour des topologies toriques. La deuxième voie étudiée a consisté à recouvrir les communications par du calcul. Nous avons proposé quelques principes algorithmiques de base permettant de recouvrir au mieux les communications. Ceux-ci sont basés notamment sur des techniques d'enchaînement de phases de calcul et de communication ainsi que sur le ré-ordonnancement local de tâches afin d'optimiser le recouvrement. Ces techniques ont été illustrées sur des algorithmes parallèles de calcul de transformée de Fourier. Les différentes implémentations de ces algorithmes sur de nombreuses machines parallèles à mémoire distribuée (T3D de Cray, SP2 d'IBM, iPSC/860 et Paragon d'Intel) ont démontré le gain en temps d'exécution apporté par ces méthodes.

2 ARCHITECTURE LOGICIELLE PARALLELE POUR LA THERMO-HYDRAULIQUE ET MAILLAGES DYNAMIQUES

Résumé :

Nous présentons dans cette partie les travaux de recherche qui ont été menés sur la conception et l'implémentation d'une architecture parallèle de code pour la simulation d'écoulements in stationnaires 3D dans des géométries complexes, le code TRIO-U [16, 17, 18, 19]. L'axe majeur qui a guidé ces travaux a été de concevoir une architecture permettant de paralléliser à moindre coût tous les nouveaux développements intégrés dans le code. Ainsi nous avons fourni des services et des mécanismes de gestion du parallélisme (communications, entrées/sorties, gestion des maillages, structures de données pour la gestion des maillages découpés, solveurs linéaires …) permettant au développeur de se concentrer sur l'implémentation de son modèle physique ou de sa méthode numérique sans se préoccuper de la parallélisation du nouveau module.

La prise en compte de maillages non constants au cours du temps (raffinement de maillages, zoom, multi-grilles géométriques …) ont nécessité non seulement des modifications dans la gestion du parallélisme mais également la prise en compte des problèmes d'équilibrage de la charge. Ces recherches, dans le cadre de la thèse de doctorat d'Eli LAUCOIN, ont permis de proposer des nouvelles méthodes de raffinement de maillage pour les équations de Navier-Stokes en parallèle [30, 31].

Summary:

In this part we introduce the research conducted on the design and implementation of a parallel architecture code for non-stationary 3D flows simulation in complex geometries, the code TRIO-U [16, 17, 18, 19]. The main focus of this work was to design an architecture which minimizes the cost to parallelize all new developments of the code. Thus we have provided services and mechanisms to handle concurrency (communications, I / O, meshes, data structures for the management of split grids, linear solvers ...) allowing the developer to focus on the implementation of its physical model and its numerical method without worrying about the parallelization of the new module.

Taking into account non-constant mesh over time (mesh refinement, zoom, geometric multi-grid ...) required not only changes in the parallelism management but also taking into account the load balancing problems. The work that took place within the framework of Eli LAUCOIN PhD allowed to propose new methods for mesh refinement for the Navier-Stokes equations in parallel [30, 31].

2.1 PRINCIPES GENERAUX POUR LA CONCEPTION DE L'ARCHITECTURE PARALLELE

2.1.1 Objectifs généraux

A l'issue de ma thèse de doctorat, j'ai intégré l'équipe Trio Unitaire (TRIO-U), au CEA à Grenoble, pour le développement d'un code 3D parallèle orienté objet pour la thermo-hydraulique [18].

Les trois principaux objectifs du projet étaient les suivants :

1. La résolution des problèmes CFD (Computational Fluid Dynmaics) de très grandes tailles en 3D sur maillages structurées et non structurées. Un exemple typique est la simulation d'écoulements turbulents dans une géométrie 3D complexe comme celle d'un cœur de réacteur nucléaire (cf Figure 2).

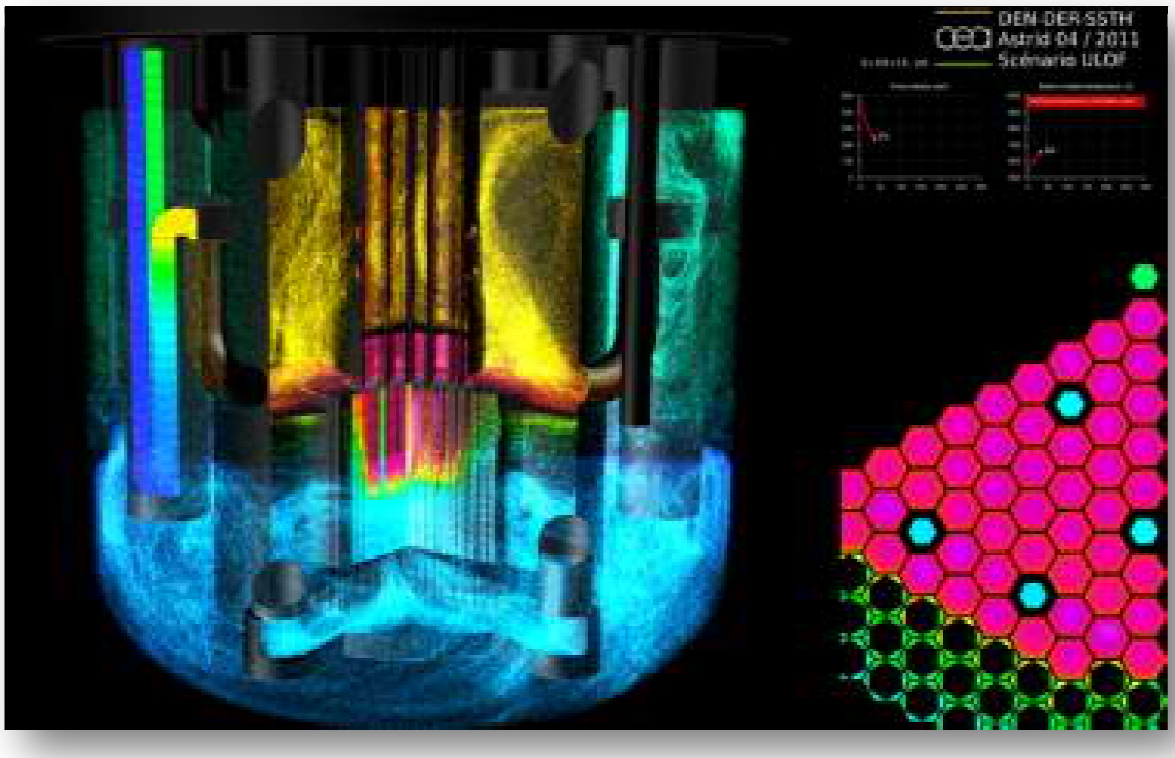


Figure 2: Exemple de simulations d'écoulement dans un réacteur nucléaire de 4ème génération à l'aide du code TRIO-U (©CEA/DEN)

2. Ce code devait être une structure ouverte pour le développement, le couplage et l'intégration d'autres applications: intégration des différents modèles physiques (compressible, incompressible, la turbulence ....), le développement et l'intégration de nouvelles méthodes numériques et de discrétisation (éléments finis, volumes éléments finis, méthodes de résolutions itératives, méthodes multigrilles, etc) et le couplage entre différentes disciplines (thermique, hydraulique, mécanique, ...).

3. La portabilité: le code doit pouvoir être exécuté sur différentes architectures de calcul, du poste de travail jusqu'aux machines parallèles à mémoire partagée et distribuée.

La simulation fine de phénomènes physiques 3D instationnaires entraine de manière automatique des maillages à plusieurs millions de mailles. Cela nécessite forcément l'utilisation de machines parallèles à mémoire distribuée pour résoudre les problèmes dans des temps raisonnables et acceptables pour le physicien et l'ingénieur. Le code a donc été conçu de manière intrinsèquement parallèle[3]. Cependant, dans le « cahier des charges », le code devait pouvoir fonctionner tout aussi bien sur des machines à mémoire partagée et sur des PCs de bureau en séquentiel. L'architecture du code doit tenir compte de toutes ces contraintes.

Afin d'atteindre les objectifs de structure ouverte et de portabilité, une conception orientée objet, a été choisie et implémentée en C++.
Ainsi, tous les mécanismes de gestion du parallélisme et des communications ont été encapsulés. Nous avions défini également des classes parallèles pour la gestion des entrées/sorties et des communications par surcharge des opérateurs standards C++ de lecture et d'écriture dans les flux (opérateurs << et >>) ce qui permet de communiquer et/ou de lire/écrire tous les objets du code de manière naturelle. C'est le type de base du flux utilisé qui détermine le comportement physique : communication MPI, écriture sur un fichier séquentiel, écritures/lectures parallèles …

En outre, l'encapsulation des routines de communication ont garanti la portabilité de l'application et ont permis une optimisation des méthodes de communication de base afin d'obtenir la meilleure performance possible sur l'architecture cible.

A cette époque, fin des années 1990, beaucoup de travaux de recherche et développement sont menés afin de développer des codes de CFD (Computational Fluid Dynamics) parallèles. La plupart de ces projets sont des études préliminaires, ne concernant que certains noyaux de base d'un code CFD. Le but de ce genre d'études est soit d'obtenir des points de repère [4, 5] soit de développer un prototype d'un code industriel parallèle. Certains d'entre eux concernent la parallélisation de codes séquentiels industriels existants comme N3S [6, 7]. Dans ce cas, chaque nouveau développement du code séquentiel (ajout d'un nouveau modèle physique, méthode numérique…) implique des travaux supplémentaires pour les paralléliser [8]. La dernière catégorie de projets porte sur la conception d'outils et de bibliothèques pour le développement d'applications CFD parallèles [9]. Notre approche est légèrement différente : nous voulions développer un code parallèle à vocation industrielle de telle sorte que l'introduction que de nouveaux modèles physiques ou méthodes numériques ne nécessitent pas de nouveaux développements, ou le moins possible, pour leur parallélisation.
L'ensemble de ces travaux est présenté dans les publications [16,17,18,19].

[3] Nous entendons par là que la gestion du parallélisme a été introduite au plus profond de l'architecture afin que l'ensemble de l'application puisse s'exécuter en parallèle.

2.1.2 Architecture parallèle du code

Un des axes majeurs qui conduit la conception du code est de n'avoir qu'une seule version du code qui puisse s'exécuter en parallèle comme en séquentiel.

2.1.2.1 Classes de gestion du parallélisme

Le principe est relativement simple et profite complètement de la conception orientée objet du code. Tous les objets du code TRIO-U héritent d'un même objet C++, `Objet_U`. Ce dernier hérite de la classe `Process`, qui représente un processus parallèle. Ainsi tout objet de TRIO-U est virtuellement un processus parallèle.

Le diagramme objet est présenté sur la Figure 3.

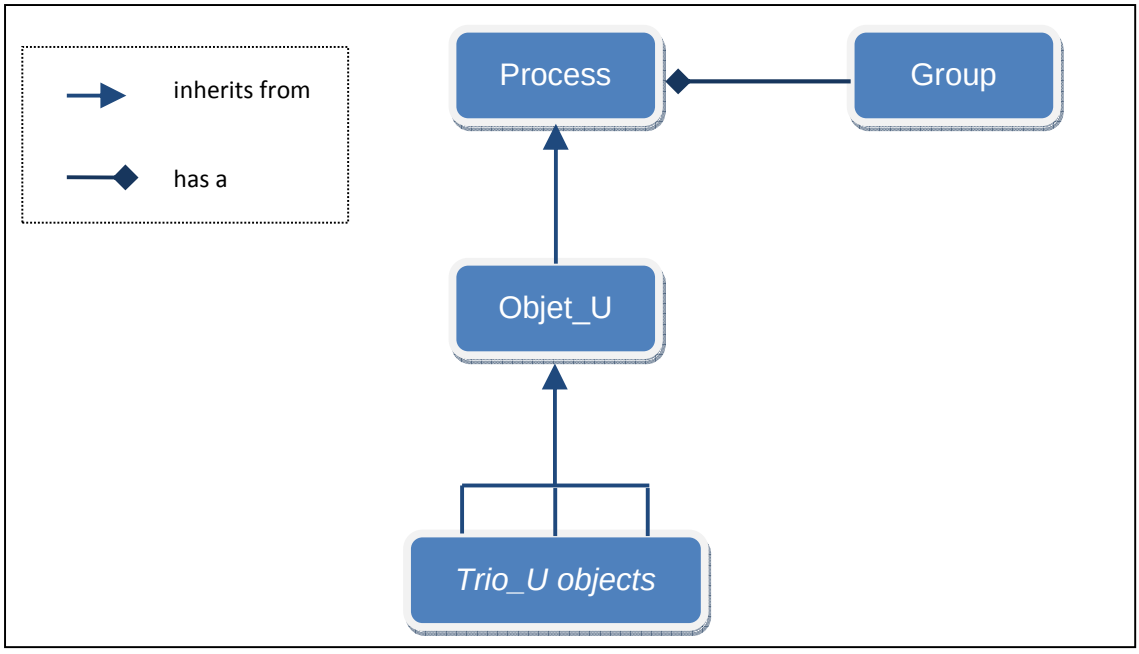


Figure 3: Diagramme objet de principe de la hiérarchie des objets dans le code TRIO-U.

2.1.2.2 Communications et entrées/sorties

Nous avons proposé des méthodes les plus simples et les plus naturelles possibles pour le développeur. Ainsi un message est envoyé d'un processus à un autre de la même façon qu'on lit ou qu'on écrit dans un fichier. Ce principe est illustré dans la Figure 4 qui est un exemple de l'échange de données entre deux processus.

```
{ //Program executed by all the proc.
    double array[10];
    char string[80];
    ArrOfInt X; // Class of the hierarchy
    SCHom SendBuffer;
    ECHom ReceiveBuffer;

    // Proc. 0 sends the message array to the proc. 1,
    SendBuffer << x << string << array << flush(0,1,99);
    // Proc. 1 receives the message from the proc. 0.
    ReceiveBuffer(0,1, 99) >> x >> string >> array;
}
```

Figure 4: Exemple d'utilisation des classes de communication.

Le même principe est utilisé pour les classes qui redéfinissent les entrées/sorties. Selon le type de fichier, le comportement est différent, bien que la syntaxe soit identique. Les méthodes de communication sont basées sur les classes précédentes. Elles mettent en œuvre les principaux schémas de communication [10] :

- Point-à-point: un échange de données entre les deux processeurs
- Diffusion: la même information est envoyée à tous les processeurs.
- Distribution: différentes données sont envoyées à chaque processeur.
- Synchronisation.

Grâce au polymorphisme du C + +, il existe seulement 4 méthodes. Ainsi, l'interface pour le développeur est grandement simplifiée. Les mêmes méthodes ou fonctions peuvent être utilisées pour envoyer n'importe quel type de données (entier, double, chaînes de caractères ou des objets).

2.1.2.3 Principe de parallélisation

TRIO-U est basé sur une technique de décomposition de domaine qui est différente des méthodes classiquement utilisées [11,12,13]. Le domaine de calcul est réparti de manière équilibrée entre les différents processeurs, en termes de nombres de mailles de calcul. Ensuite, chaque processeur résout le problème sur son propre sous-domaine de manière plus ou moins indépendante des autres. La distribution initiale est obtenue en utilisant des outils de partitionnement comme METIS [14] ou Scotch [15]. Les méthodes classiques utilisées dans les méthodes décomposition de domaine consistent à résoudre en parallèle les problèmes de chaque sous-domaine, puis de traiter les problèmes des frontières afin d'assurer la continuité de la solution. Puisque l'application est intrinsèquement parallèle, dans TRIO-U c'est un seul problème, le même qu'en séquentiel, qui est résolu par plusieurs processeurs. Les éléments qui sont situés sur une frontière sont traités comme

des éléments internes, et nous n'avons pas à résoudre un problème particulier à la frontière. De plus, si nous voulons développer de nouvelles méthodes numériques ou modèles physiques, nous n'aurons pas à traiter le parallélisme, puisque les structures de données sont parallèles. Pour ce faire, les objets qui représentent la géométrie et la discrétisation portent des informations sur l'interface entre les sous-domaines. Ainsi, la connectivité entre les sous-domaines est contenue dans la structure de données, et chaque objet de la géométrie sait donc comment obtenir des informations d'un autre sous-domaine.

Afin d'optimiser les communications, il y a un recouvrement des sous-domaines avec une copie des zones voisines, de sorte que les valeurs aux bords des sous-domaines ne sont échangées qu'à chaque pas de temps de la simulation (notion de mailles fantômes). Un exemple est donné dans la Figure 5.

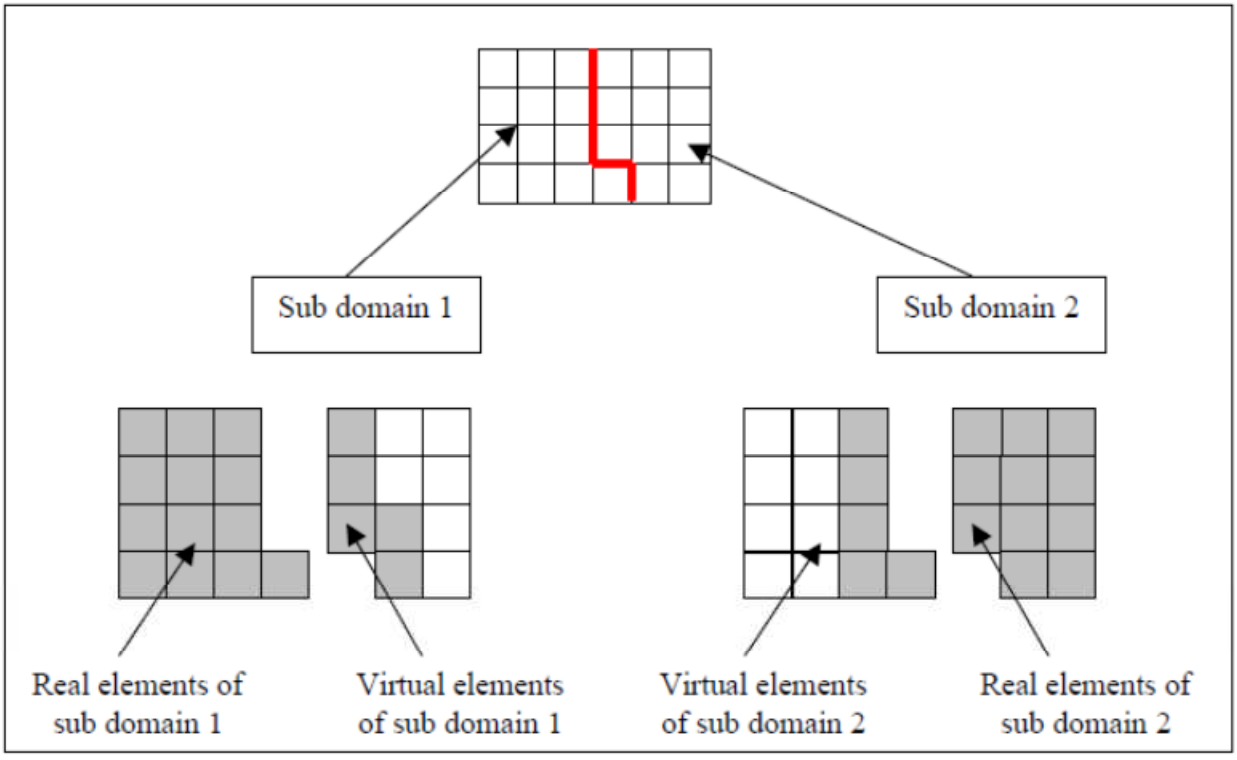


Figure 5: Exemple de décomposition de domaine dans TRIO-U avec zones de recouvrement.

Nous avons défini la notion de tableaux distribués. Chaque tableau distribué détient une partie propre (Partie Réelle) qui contient les valeurs de son sous-domaine et une partie virtuelle qui contient une copie des valeurs des autres sous-domaines voisins. La taille de la zone de recouvrement dépend des inconnues et du schéma numérique. Le principe est illustré sur la Figure 6.

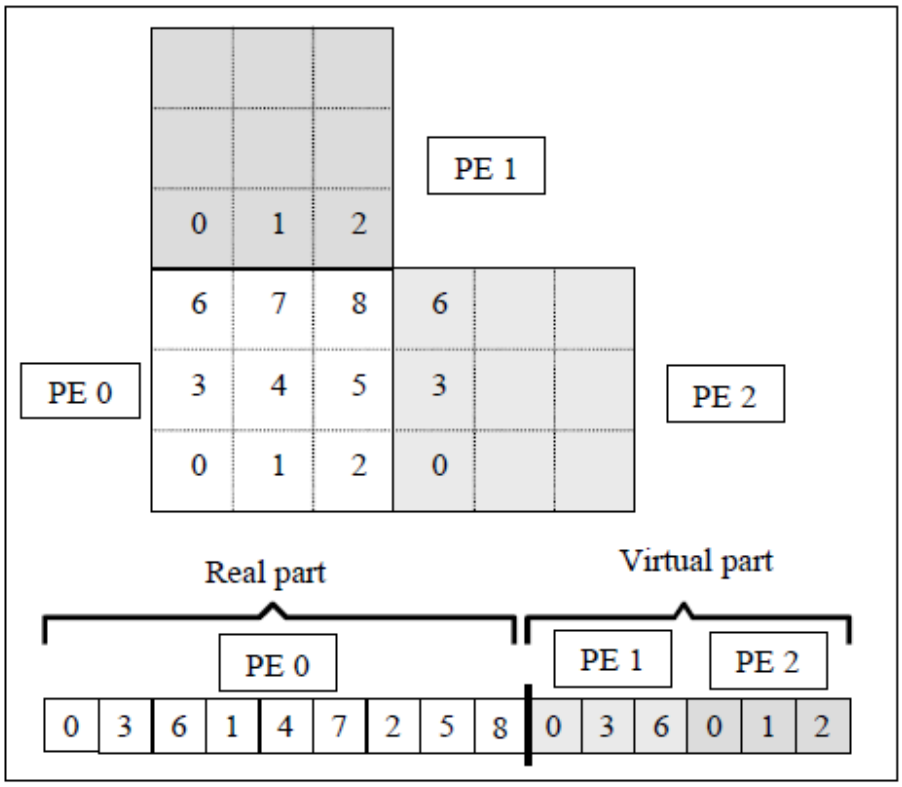


Figure 6: Schéma de principe des tableaux distribués dans TRIO-U.

2.1.2.4 La résolution des systèmes linéaires

La méthode classiquement utilisée pour les écoulements incompressibles pour une discrétisation de type volumes finis, est une méthode de type semi-implicite en pression. Cela implique qu'à chaque pas de temps, il est nécessaire de résoudre le système en pression.

La matrice en pression est symétrique et constante tout au long du calcul. Les valeurs des éléments sont proportionnelles aux volumes de contrôles de la discrétisation. Ainsi la structure creuse de la matrice est directement liée au maillage discret de la géométrie de calcul.

La première méthode utilisée pour résoudre ce système linéaire est l'algorithme du gradient conjugué avec un pré-conditionnement de type SSOR. Ce dernier, correspondant à une remontée de système triangulaire, est très peu parallèle et le choix avait été fait de négliger les blocs non diagonaux de la matrice afin d'éliminer les communications lors de l'application du pré-conditionnement. Cela permettait d'augmenter l'efficacité de la parallélisation de la phase de résolution du système linéaire bien que le nombre d'itérations pour atteindre la convergence avait tendance à augmenter avec le nombre de sous-domaines, du fait de la baisse d'efficacité du pré-conditionneur.

2.2 L'OPTIMISATION DE LA RESOLUTION DU SYSTEME EN PRESSION

Comme nous l'avons expliqué au paragraphe précédent, les caractéristiques de la matrice (structure, conditionnement, …) sont directement liées au maillage discret de la géométrie de

calcul. Les premiers éléments de discrétisations utilisés dans TRIO-U présentaient un point unique de pression au centre de la maille. Dans le cas non structuré, cet élément, P0 [20], n'offrait pas toutes les « bonnes propriétés » et un nouvel élément en pression, P0/P1B, a été introduit (voir Figure 7). Si ce dernier a rempli tous ces objectifs d'un point de vue représentation numérique, il a entrainé une raideur supplémentaire dans la résolution du système du fait du couplage faible entre la pression au centre de l'élément et la pression aux sommets.

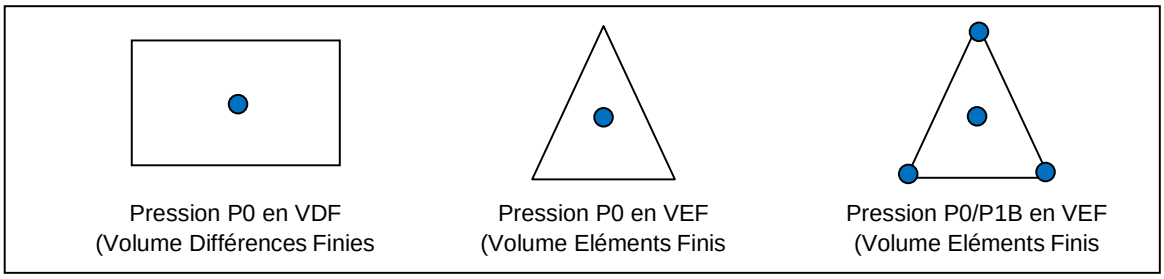


Figure 7: Principaux éléments de discrétisation de la pression dans TRIO-U.

Nous avons donc entamé différents travaux avec des étudiants en stage de fin d'études pour l'introduction et le test de nouvelles méthodes de résolution de systèmes linéaires et de pré-conditionneurs. Notamment les méthodes multi-grilles algébriques ou des pré-conditionneurs parallèles plus performants que le SSOR [86] par blocs initialement implémenté.

Tous ces travaux nous ont conduit assez rapidement à coupler le code TRIO-U avec différentes bibliothèques numériques parallèles comme PETSc [21] ou HYPRE [22] afin de pouvoir tester facilement et rapidement ces différents solveurs. Du fait des différences d'interfaces entre les différentes bibliothèques, le choix a été fait d'offrir une interface commune à toutes ces bibliothèques via NumericalPlaton ce qui a permis de n'implémenter qu'une seule interface dans le code TRIO-U vers NumericalPlaton et ensuite d'avoir accès à un grand nombre de bibliothèques numériques parallèles [21].

A titre d'illustration, nous avions fait des tests de comparaison de différents solveurs de systèmes linéaires en séquentiel et en parallèle sur cas réel de calcul 3D d'écoulement instationnaire (cf. Figure 8). Nous présentons sur la Figure 9 les résultats obtenus en fonction de différents solveurs utilisés, dans ce cas en séquentiel. Comme on peut le constater, le Gradient Conjugué avec pré-conditionnement SSOR originel est loin d'être le plus mauvais. Par ailleurs, le couple pré-conditionneur-solveur a une importance prépondérante sur le coût de calcul par pas de temps (jusqu'à un facteur 3). Ce n'est pas forcément le même constat en parallèle. Les résultats d'exécution parallèle sont présentés sur la Figure 10. Dans ce cas, on constate tout l'intérêt d'avoir des pré-conditionneurs véritablement parallèles qui permettent de conserver des accélérations plus importantes qu'avec un simple pré-conditionneur par blocs.

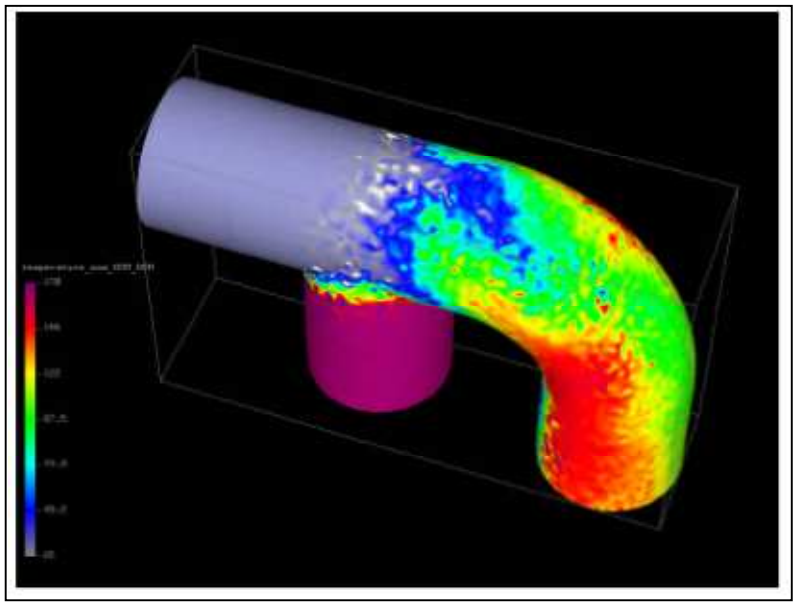


Figure 8: Exemple de calcul 3D d'écoulement instationnaire en LES[4] [24] servant de base de tests pour les solveurs en pression dans TRIO-U.

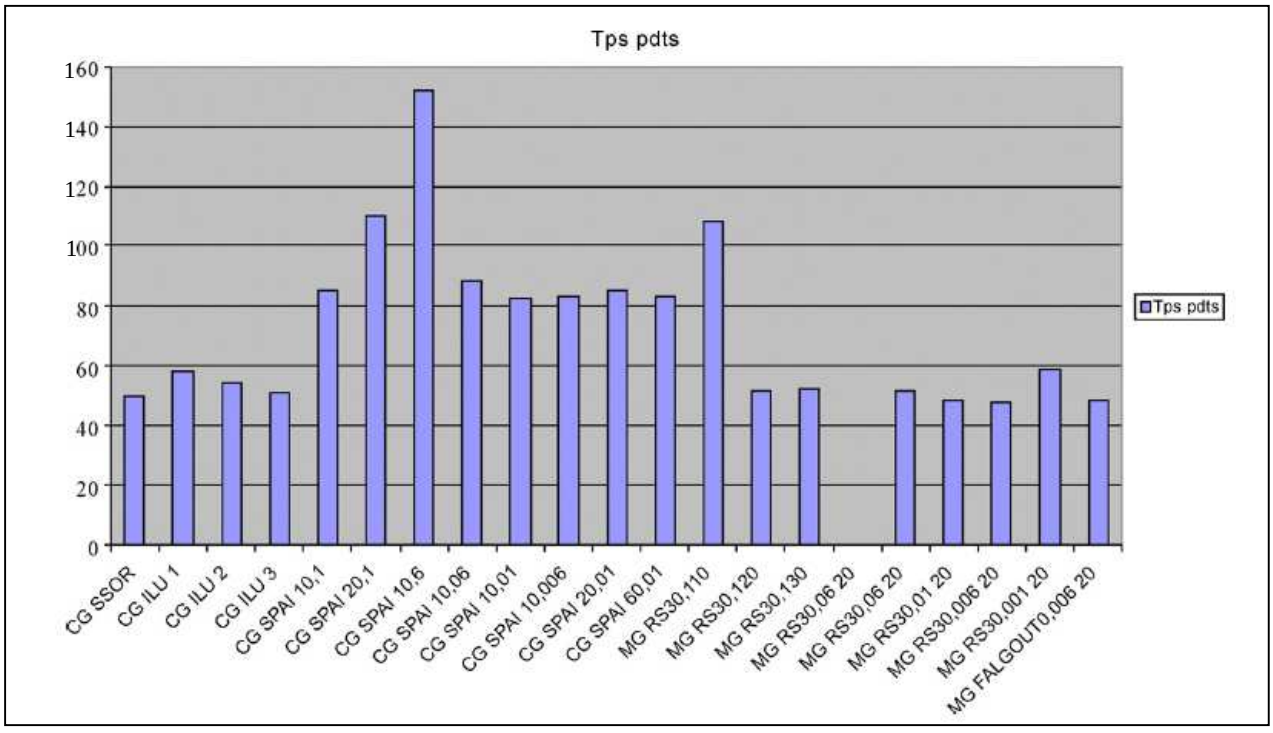


Figure 9: Coût de calcul en secondes par pas de temps en fonction des différents solveurs linéaires.

[4] LES : Large Eddy Simulations ou Simulation des Grandes Echelles (SGE)

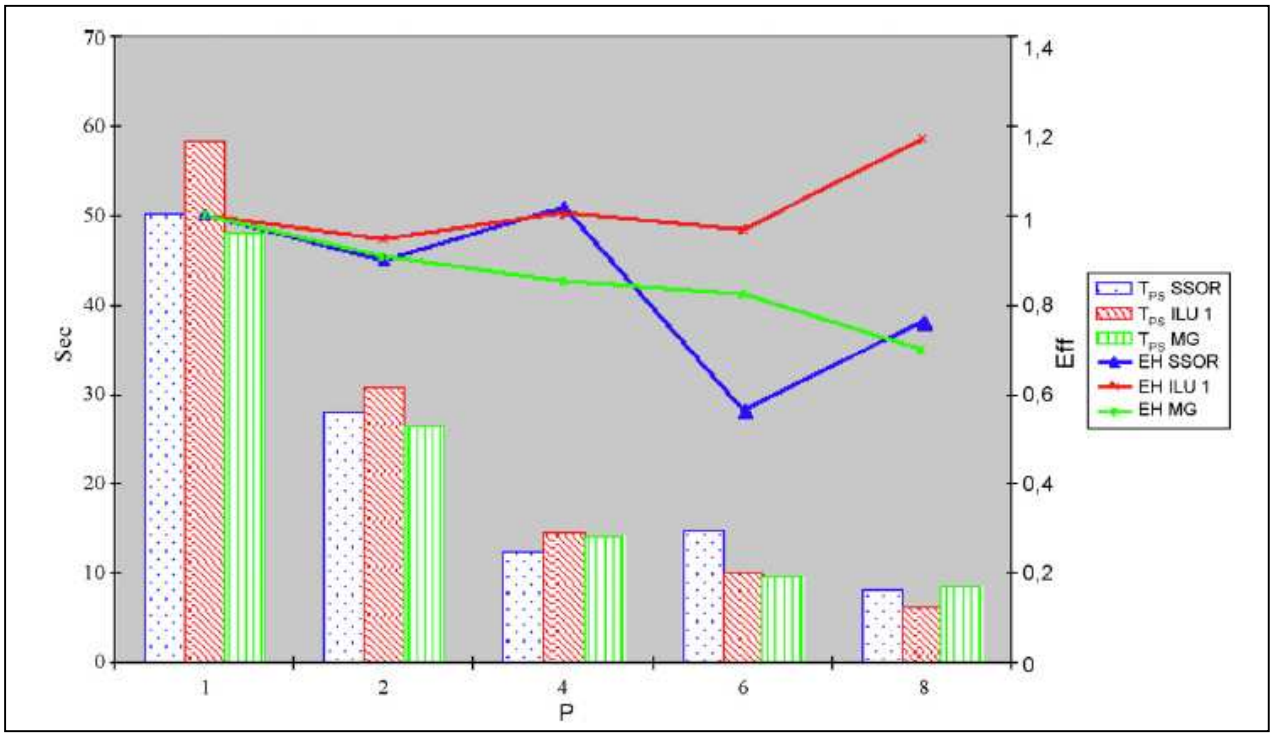


Figure 10: Temps d'exécutions en secondes (axe gauche des ordonnées) et accélération (axe droit des ordonnées) en utilisant différents pré-conditionneurs et fonction du nombre de processeurs.

2.3 QUELQUES EXEMPLES DE RESULTATS

Les principales applications qui ont été réalisées à l'époque avec le code TRIO-U ont été la simulation des écoulements turbulents en Simulation des grandes Echelles (SGE, LES pour Large Eddy Simulation en anglais), l'application PRICELES, ainsi que la simulation directe d'écoulements diphasiques par interfaces diffuses. Cette dernière application a été codée en 2 ou 3 semaines dans la plateforme TRIO-U et a été parallélisée immédiatement grâce aux méthodes décrites précédemment.

Un exemple de calcul de réacteur réalisé à l'aide de l'application TRIO-U/PRICELES est présenté sur la Figure 11[25, 26, 27].

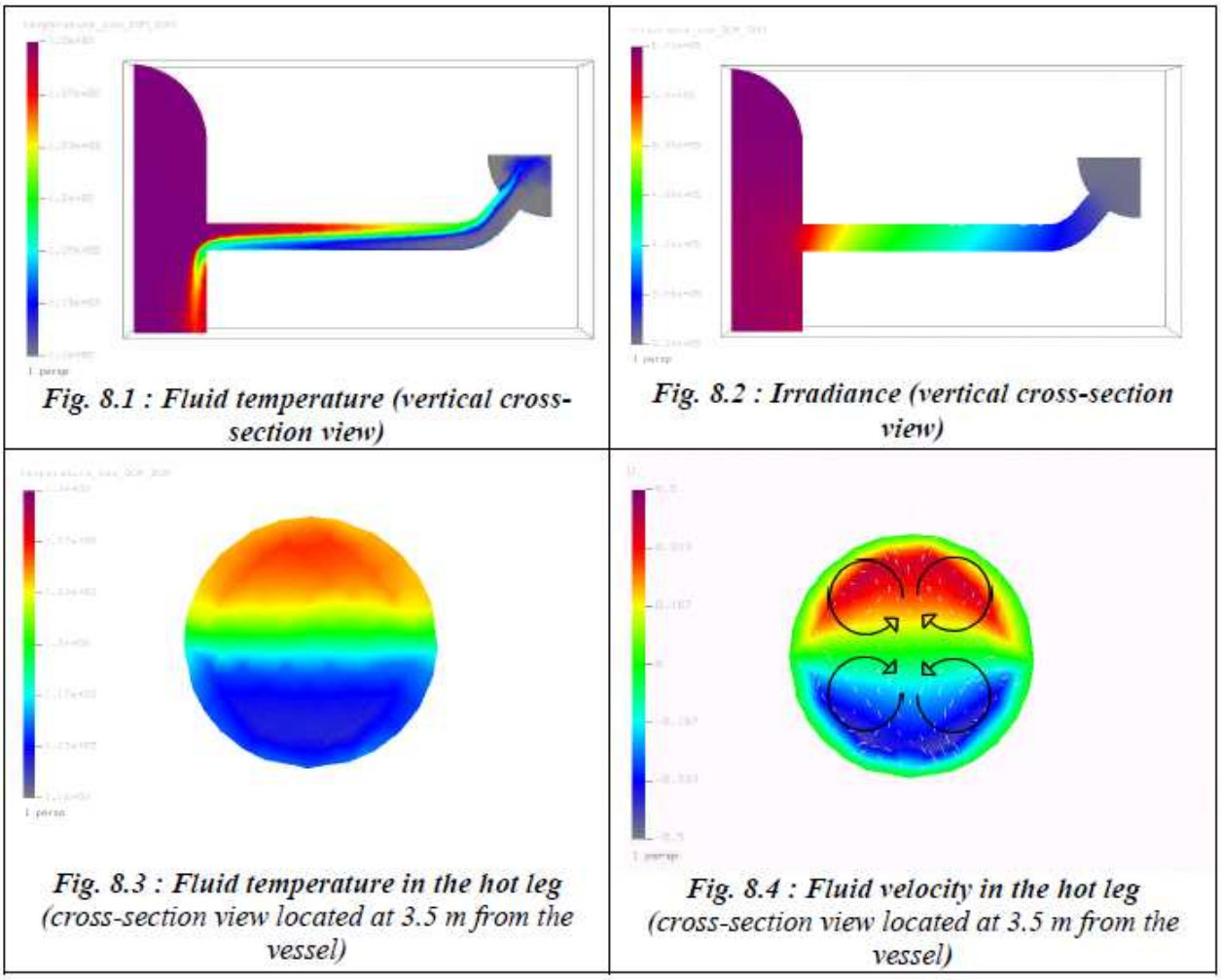


Fig. 8.1 : Fluid temperature (vertical cross-section view)

Fig. 8.2 : Irradiance (vertical cross-section view)

Fig. 8.3 : Fluid temperature in the hot leg (cross-section view located at 3.5 m from the vessel)

Fig. 8.4 : Fluid velocity in the hot leg (cross-section view located at 3.5 m from the vessel)

Figure 11: Exemple d'application réacteur du code TRIO_U/PRICELES.

Sur le cas de calcul présenté sur la Figure 8, nous résumons, Tableau 1, les performances obtenues sur un cluster de calcul HP/COMPAQ ES45 sur 5 processeurs avec un maillage de 3 millions de mailles et 10 processeurs avec 10 millions de mailles [28].

Calcul	Temps / pas de temps (s)	SpeedUp
Seq. 3M degrés de liberté	5.45	
Paral. 3M degrés de liberté	1.2	4.5
Seq. 10M degrés de liberté	23	
Paral. 10M degrés de liberté	2.8	8.2

Tableau 1 : Résultats de performances avec l'application TRIO_U/PRICELES.

Un exemple d'application en simulation directe diphasique est illustré sur la Figure 12, avec la simulation d'une création de bulles en paroi.

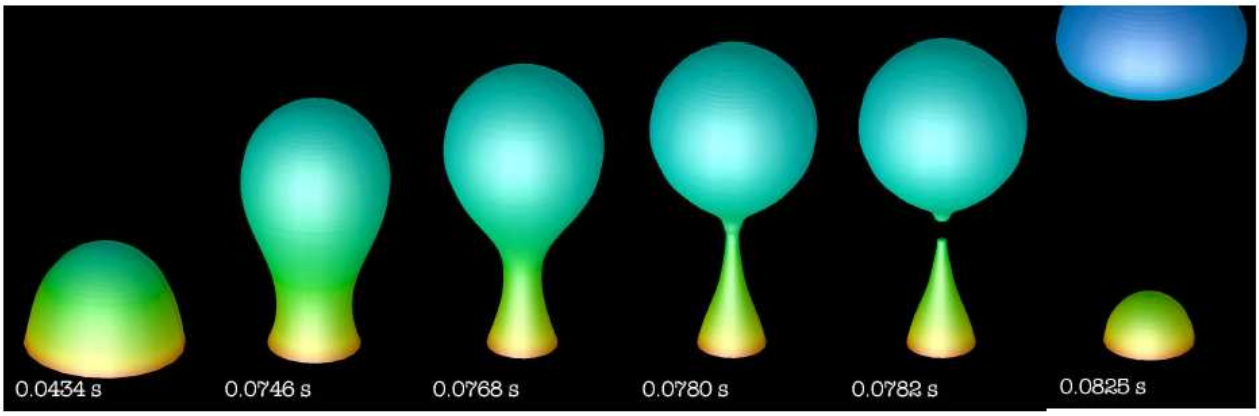


Figure 12: Simulation de création et détachement de bulles en simulation directe avec le code TRIO-U.

Sur cette dernière application, du fait que tous les schémas numériques sont explicites et qu'ainsi d'un point de vue parallèle les communications induites sont locales, nous avions obtenu d'excellentes efficacités (supérieure à 90% sur une vingtaine de processeurs) [27].

2.4 LA PROBLEMATIQUE DE LA V&V D'UN CODE PARALLELE

Le débogage et la validation des applications parallèles ne sont pas des problèmes triviaux. L'un des meilleurs moyens de valider les résultats d'une exécution parallèle est de les comparer avec les résultats d'un calcul séquentiel. Mais le problème est de savoir comment détecter la source de la différence entre les deux exécutions. Afin de simplifier le travail de mise au point et de validation, certains outils ont été développés dans le code afin de comparer automatiquement les résultats des deux exécutions [28]. Le principe est le suivant : tout d'abord, il faut instrumenter le code afin de préciser quelles sont les valeurs à vérifier: `Debog::check(msg,valeurs)`. Ensuite, le code est exécuté en séquentiel, et produit des traces contenant les résultats des valeurs à vérifier dans un fichier (fichier DEBOG). Par la suite, la même simulation se fait en parallèle, et chaque valeur à vérifier est comparée à la valeur séquentielle et produit des fichiers de trace contenant la différence entre les valeurs séquentielles et parallèles (fichiers `Err_DEBOG`). Le principe est décrit dans la Figure 13.

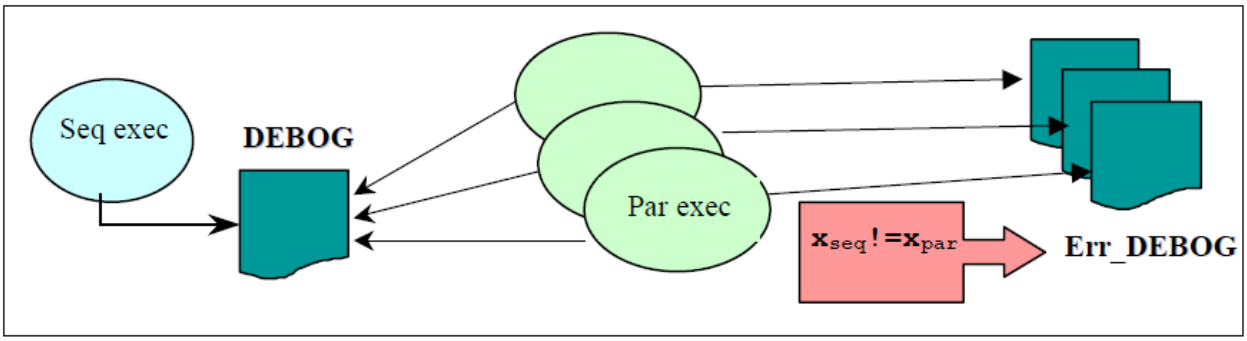


Figure 13: Principe d'utilisation de l'outil DEBOG.

Pour chaque message, sont donnés : le numéro séquentiel et parallèle de support de discrétisation, les valeurs séquentielles et parallèles et les différences relatives. Par exemple, supposons que nous

voulions vérifier les valeurs de pression après un opérateur spécifique `Op1`. Ensuite, nous devons instrumenter le code de la manière suivante :

```
// Call to compute operation of Op1
Op1.compute(pressure_values);
// Check the values
Debog::check("Pression values after Op1", pressure_values);
```

En supposant une discrétisation de la pression au centre de la maille, nous obtenons ce type de trace dans le fichier :

```
Pressure values after Op1 → // trace message msg
// sequential cell number – parallel cell number – seq. value –
par.value - relative difference
```

Un autre mode avait été également implémenté, le mode "driven", qui permettait d'imposer à l'exécution parallèle une valeur obtenue par l'exécution séquentielle. Le principe est illustré sur la Figure 14.

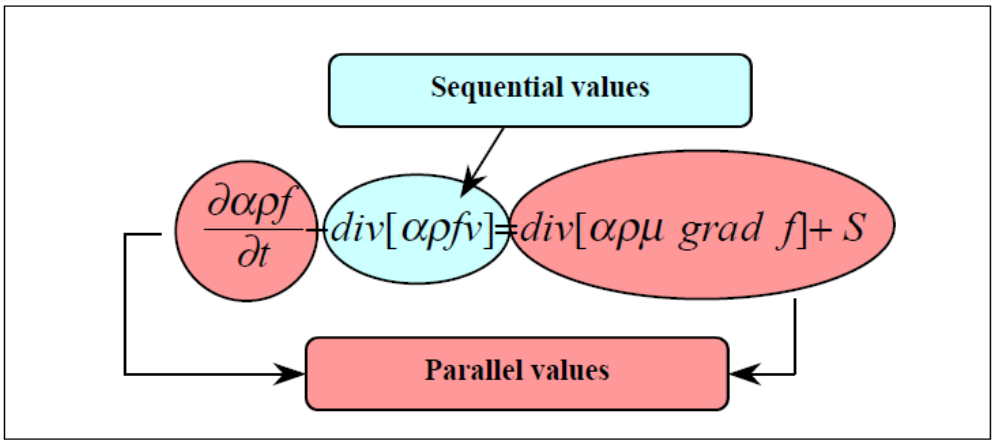


Figure 14: Principe du mode « driven » l'outil DEBOG.

2.5 MAILLAGE DYNAMIQUE ET EQUILIBRAGE DE CHARGE

2.5.1 Problématique

Au fur et à mesure des travaux effectués dans TRIO-U et des différentes applications métiers issues de la plateforme, un besoin nouveau est apparu : la notion de maillage dynamique. Sous ce vocable nous regroupons plusieurs notions :

1. La notion de maillage « mouvant » sur maillage fixe, comme dans le cas de méthode de type de suivi de fronts pour la simulation d'écoulements diphasiques [29] (cf. Figure 15);
2. La notion de raffinement dynamique de maillage ;
3. La notion de multi-grilles géométrique
4. La notion de zoom, qui est une généralisation des deux notions précédentes, dans le sens où un maillage raffiné local peut venir se superposer à un maillage grossier pour capter des

phénomènes locaux et/ou avec accélération de la convergence par l'utilisation des résultats obtenus sur la grille grossière (cf. Figure 16).

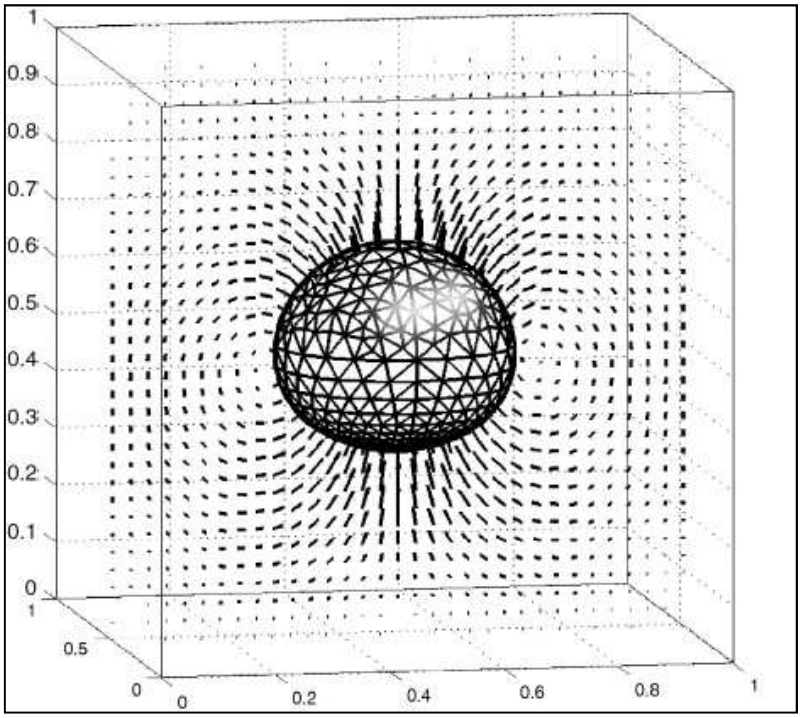


Figure 15: Illustration d'une méthode de type « suivi de fronts » (« Front Tracking ») pour la simulation d'écoulements diphasiques.

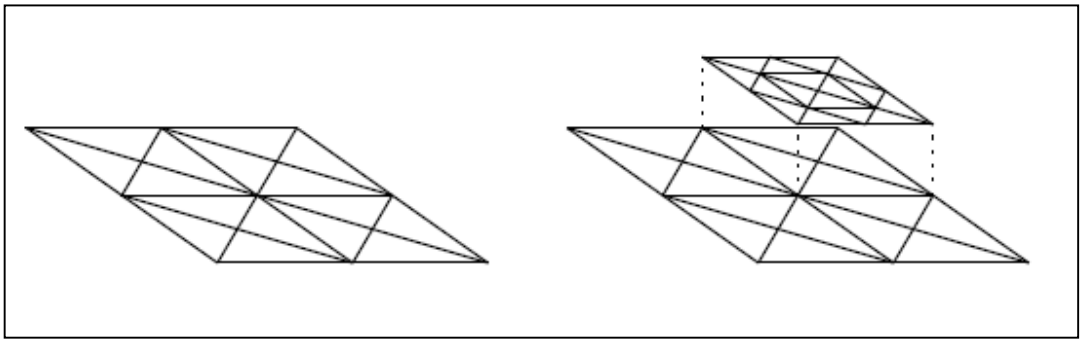


Figure 16: Illustration de maillage localement raffiné multi-niveaux.

Bien sûr cette nouvelle problématique devait être abordée dans un contexte d'exécution parallèle. Qui dit maillage dynamique et parallélisme entraîne forcément la notion d'équilibrage dynamique de la charge. A ce stade, l'équilibrage de la charge dans le code TRIO-U était réalisé de manière statique par découpage initial du domaine de calcul de manière équilibré entre les différents processus, l'hypothèse étant faite (et vérifiée pour le domaine d'application du code) que le coût unitaire de calcul par maille est uniforme.

Lorsque que la maillage se déplace, ou se raffine localement, un déséquilibre de charge apparait forcément entre les différents processus.

C'est l'ensemble de ces problématiques qui fut le cœur la thèse de doctorat de M. Eli LAUCOIN, soutenue en 2007 : «*Développement du parallélisme des méthodes numériques adaptatives pour un code industriel de simulation en mécanique des fluides* » [30].

2.5.2 Suivi de fronts en parallèle pour des écoulements diphasiques.

Les premiers travaux ont porté sur le suivi d'interfaces en simulation directe pour des écoulements diphasiques en parallèle [31]. L'algorithme de base pour ce type de problème est synthétisé sur la Figure 17.

1. build the projectors and extrapolators,
2. compute the phase indicator ϕ^n,
3. compute the interfacial source term f_σ^n on the interface,
4. extrapolate the interfacial source term onto the Eulerian discretization,
5. compute the velocity u^{n+1} and the pressure P^{n+1} using TRIO–U's linear solvers,
6. update the description of the interface, computing Γ^{n+1}.

Figure 17: Algorithme de base de la méthode de suivi de front.

Afin de paralléliser cet algorithme, il a été nécessaire d'étendre la notion de tableaux distribués pour prendre en compte le maillage de l'interface qui est convecté avec l'écoulement (cf. Figure 18).

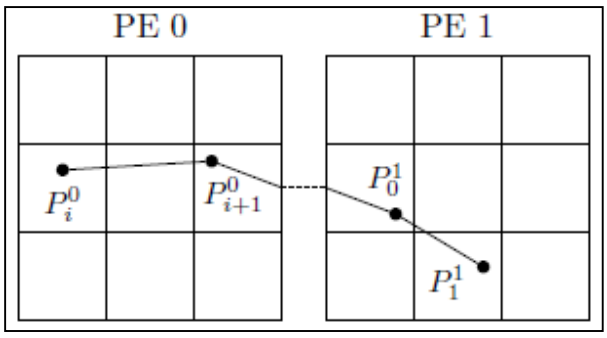


Figure 18: Maillage de l'interface distribuée sur différents processeurs.

Pour la parallélisation effective de l'algorithme général présenté en Figure 17, nous n'avons pas eu besoin de changer sa structure globale, mais chacune de ses étapes a du être parallélisée. Heureusement, grâce aux vecteurs distribués dynamiques, une grande partie de ce processus de parallélisation a été réalisé simplement. En effet, les étapes 1, 3 et 4, correspondant respectivement au calcul des coefficients des extrapolateurs, le calcul du terme source interfacial, et son extrapolation sur un champ eulérien, peuvent être effectués en parallèle de manière

transparente à l'aide des vecteurs dynamiques distribués. Les étapes 2 et 5 sont des calculs purement eulériens, et en tant que tels ils sont directement parallélisés en utilisant les vecteurs distribués statiques. Nous avons du, cependant, prendre un soin particulier lors de la parallélisation de la dernière étape, à savoir l'advection des interfaces. En effet, le calcul du déplacement de la grille d'interface doit respecter une exigence forte correspondant à la conservation de la masse de chacune des phases.

Le moyen le plus naturel pour déterminer le déplacement des nœuds d'interface est d'utiliser la valeur du champ de vitesse eulérien interpolée à la position de ces nœuds. Nous pouvons démontrer que le champ de vitesse est continu à travers l'interface, car il est à divergence nulle et parce que nous supposons qu'il n'y a pas de transfert de masse à travers l'interface. Malheureusement, cet argument est vrai uniquement pour le problème continu. En effet, l'interpolation du champ de vitesse discret introduit des erreurs numériques et l'exigence de conservation de la masse n'est donc pas satisfaite. Cette difficulté est accentuée si la discrétisation du champ de vitesse repose sur un élément de CROUZEIX-RAVIART P1 non-conforme, comme dans la méthode des Volumes Eléments Finis de TRIO-U. L'approche proposée a consisté à prédire une variation de masse correspondant au déplacement de chaque nœud d'interface, et ensuite à optimiser ces déplacements de telle sorte que la variation de masse souhaitée par nœud soit obtenue.

Ces travaux ont permis de réaliser des calculs qui étaient difficilement réalisables auparavant, comme la simulation d'une colonne à bulles (cf. Figure 19) qui a fait l'objet d'un « Grand Challenge » sur les machines du CCRT en 2009.

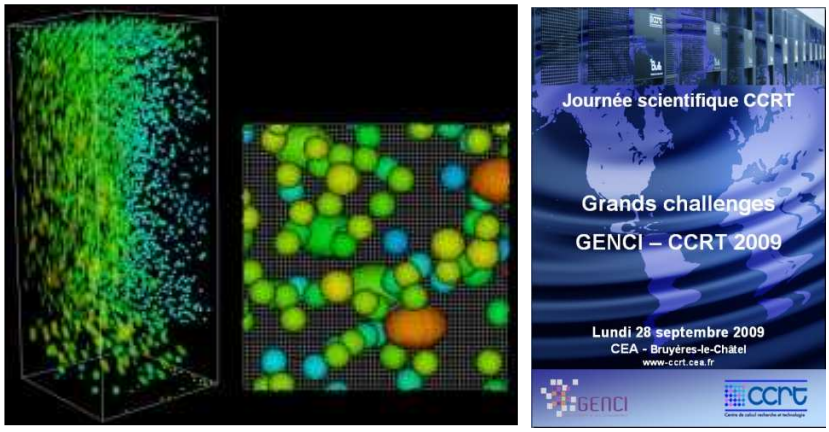


Figure 19: Simulation d'une colonne à bulles à l'aide du code TRIO-U – Grand Challenge CCRT 2009.

2.5.3 Adaptation dynamique de maillage.

Le reste de la thèse de doctorat a été principalement consacré à l'étude et l'implémentation d'une méthode dynamique de raffinement de maillages en parallèle dans le code TRIO-U.

Les travaux ont eu pour principal objectif de concevoir et implémenter une extension de la plate–forme TRIO-U permettant la simulation de problèmes tridimensionnels complexes issus du

domaine de la mécanique des fluides, en utilisant une démarche adaptative déséquilibrante dans un contexte d'exécution parallèle. Ce travail s'est articulé autour de diverses contributions :

- Dans le domaine de la géométrie, où nous avons réalisé une analyse critique approfondie de différentes méthodes d'adaptation de maillage, afin de déterminer la démarche la plus adaptée à la plate–forme TRIO-U;
- En analyse numérique, où nous avons établi un formalisme reposant sur la méthode des éléments joints, adapté à l'élément fini de CROUZEIX–RAVIART, qui constitue la base de la méthode de Volumes–Éléments finis de TRIO-U;
- En algorithmique, où nous avons développé des algorithmes parallèles permettant la mise en œuvre de notre méthode adaptative;
- En génie logiciel, où nous avons conçu une architecture à la fois souple et extensible, pour implémenter notre méthode au sein de la plate–forme TRIO-U.

Au cours de ce travail de thèse de doctorat, nous avons conçu, implémenté et validé une méthode numérique adaptative, basée sur la méthode des Mortars [32, 33] utilisable dans un environnement d'exécution parallèle pour la réalisation de simulations industrielles en thermo-hydraulique. Les idées fondatrices de cette méthode sont l'utilisation d'une technique d'adaptation de maillage à la fois simple, robuste et efficace, le traitement des non–conformités du maillage au niveau de la méthode numérique en utilisant les propriétés des méthodes d'éléments joints, et une implémentation parallèle reposant sur les concepts des méthodes de décomposition de domaines. Pour cela, nous avons spécifié un algorithme d'adaptation de maillage et une technique de raffinement adéquate. Nous avons également développé un formalisme d'éléments joints adapté à l'élément fini de CROUZEIX–RAVIART et à la forme des non–conformités générées par l'adaptation du maillage. Enfin, nous avons spécifié une démarche théorique générale à nos besoins, afin de concevoir l'implémentation parallèle de notre méthode adaptative (cf. Figure 20).

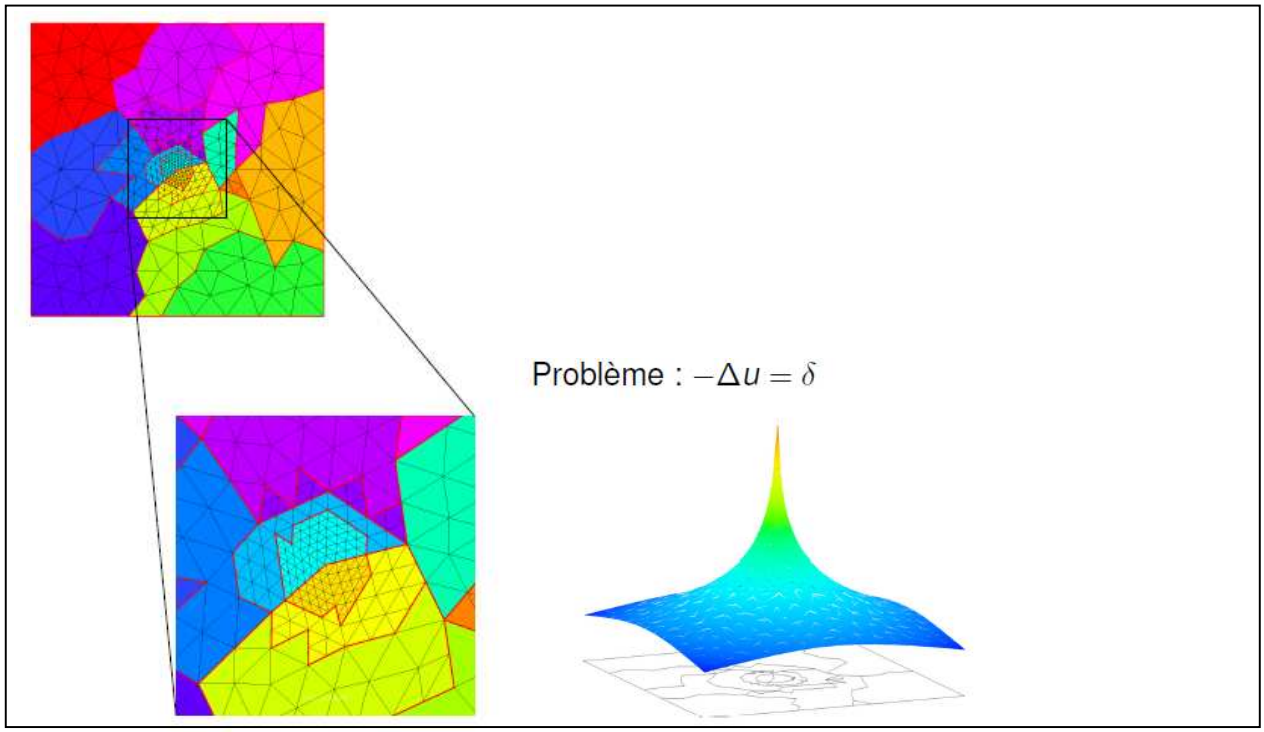


Figure 20: Validation de l'algorithme d'adaptation de maillage dans le code TRIO-U sur le cas du Laplacien 2D.

Nous avons également développé un module dans la plate–forme TRIO-U pour mettre en œuvre ces idées. Les résultats obtenus mettent en évidence le bon comportement de ce module lors de la résolution de divers problèmes.

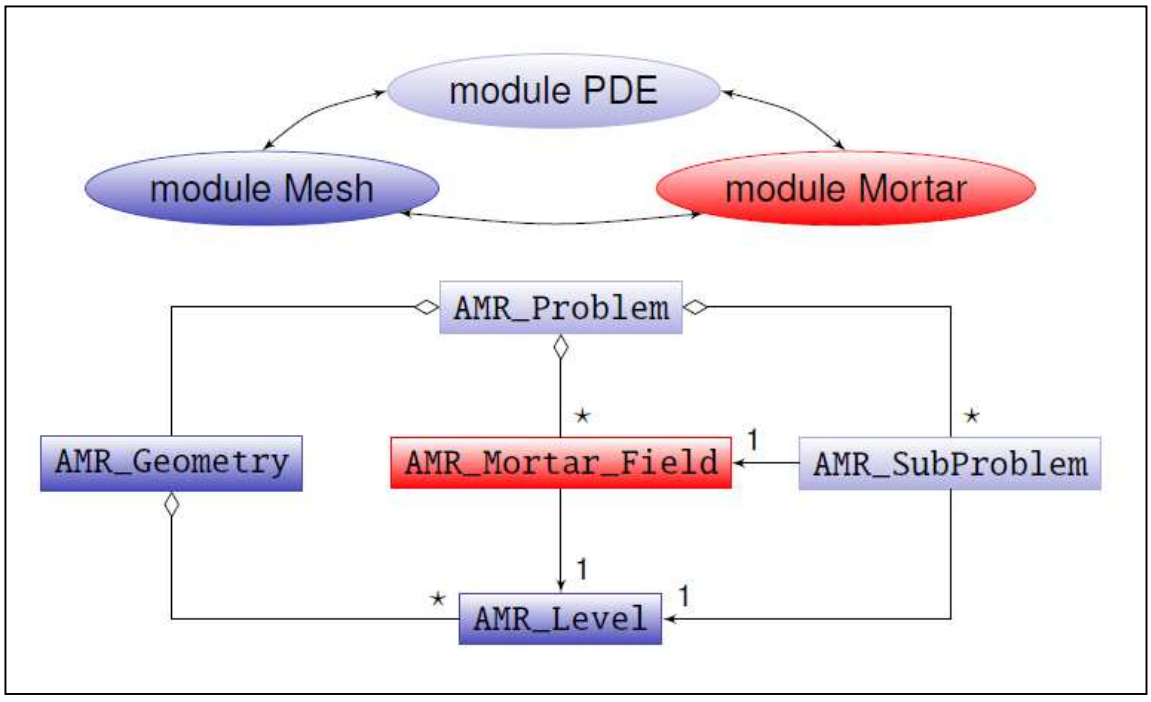


Figure 21: Diagramme objet général du module AMR (Adaptive Mesh Refinement).

La comparaison avec une autre approche des simulations adaptatives déjà présentes dans TRIO-U nous a conforté dans nos choix de conception et d'implémentation. Nous présentons les résultats obtenus pour le problème de Stokes en 2D sur la Figure 22 et le gain obtenu par cette méthode par rapport à un maillage totalement raffiné sur le Tableau 2.

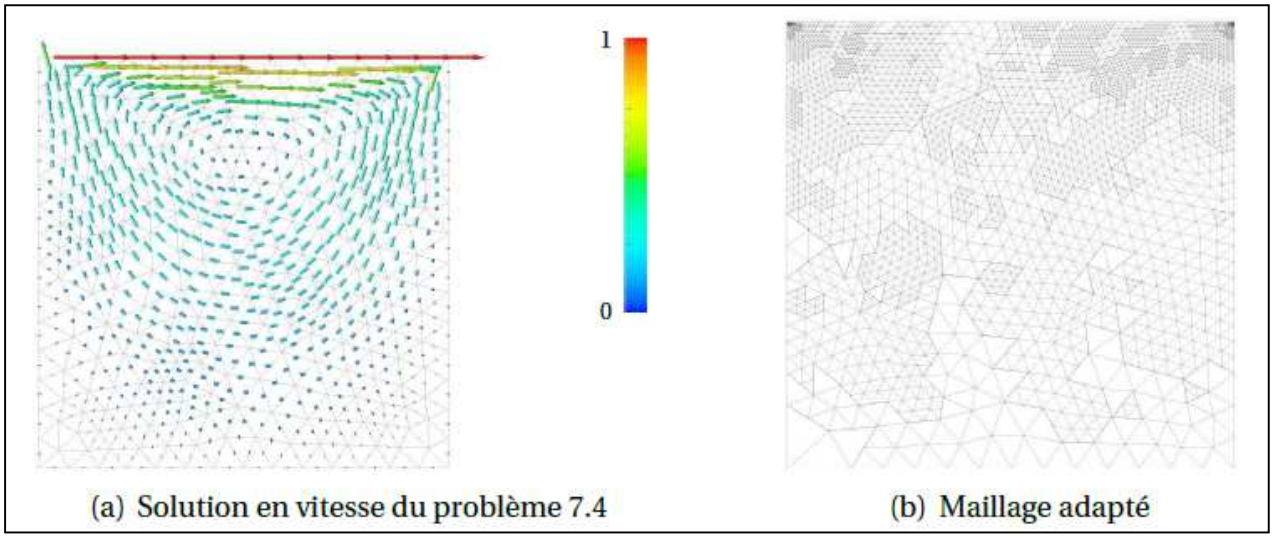


Figure 22: Adaptation dynamique de maillage appliqué au problème de Stokes.

Nombre initial de mailles	118
Nombre final de mailles	4 738
Nombre de niveaux de raffinement	9
Nombre de mailles du maillage équivalent totalement raffiné	30 932 992

Tableau 2 : Gain en nombre de mailles par adaptation dynamique de maillage appliqué au problème de Stokes.

2.5.4 Adaptation dynamique de maillage et parallélisme.

Nous détaillons ici la démarche et les solutions mises en œuvre pour la parallélisation de cette méthode de raffinement dynamique de maillage.

Nous présentons tout d'abord plus en détail la structure de notre méthode adaptative. Nous avons cherché à mettre en évidence les différentes formes de parallélisme présentes dans les algorithmes utilisés. La phase d'adaptation de la géométrie et la phase de résolution reposent sur des principes radicalement différents qui ont conduit à les paralléliser selon des schémas différents.

Étant donné que les données nécessaires à ces deux familles d'algorithmes sont radicalement différentes, il semble naturel que leur parallélisation repose sur des concepts différents.

- Parallélisation de la phase d'adaptation du maillage : Pour des raisons de simplicité d'implémentation, nous avons décidé de relaxer la contrainte d'équilibrage pour les algorithmes d'adaptation de la géométrie. Nous avons choisi de distribuer de façon équilibrée les éléments du maillage initial à l'aide d'un algorithme d'équilibrage statique, et de ne pas introduire de phase de redistribution des tâches qui leur sont associées. Ainsi, chaque processeur se voit confier la gestion du raffinement des éléments qui lui ont été

attribués à l'initialisation du calcul, et les schémas de communication nécessaires à la maintenance de la relation d'incidence restent déterministes.

- Parallélisation de la phase de résolution : La méthode numérique que nous proposons pour résoudre un problème posé sur une géométrie aplatie repose sur une approche similaire à celle présentée par C. BERNARDI, F. HECHT et Y. MADAY dans les articles [32,33]. Il s'agit essentiellement d'une méthode de décomposition de domaine, qui, présente un caractère intrinsèquement parallèle. Le cadre naturel pour l'implémentation parallèle de ce genre de méthode est un parallélisme de tâches, chaque tâche correspondant à la résolution de l'un des problèmes locaux. Nous avons par ailleurs renforcé cette idée en introduisant une extension au formalisme des éléments joints de façon à assurer le caractère bien posé de chacun des problèmes locaux. Tout ceci nous conduit à choisir un modèle de parallélisme fonctionnel pour le partitionnement de la phase de résolution. Les communications sont essentiellement matérialisées par les interfaces séparant les sous-domaines. Nous avons fait le choix, pour équilibrer dynamiquement la phase de résolution, de recourir à un algorithme de partitionnement de graphe de manière dynamique [34].

L'efficacité des algorithmes d'équilibrage de la charge peut être qualifiée de deux façons.
En premier lieu, on est en droit de se demander s'il est pertinent, voire utile, de chercher à équilibrer la charge de calcul. Nous allons voir dans le paragraphe qui suit que cela est en réalité essentiel à l'efficacité du calcul. Ensuite, il s'agit de vérifier que l'algorithme d'adaptation de la charge ne présente pas un coût trop important en regard de celui de la résolution du problème.
La pertinence de l'équilibrage de charge est plus qu'évidente au regard de l'expérience que nous avons réalisée. Nous avons cherché à résoudre un problème de LAPLACE 3D (cf. Figure 23).

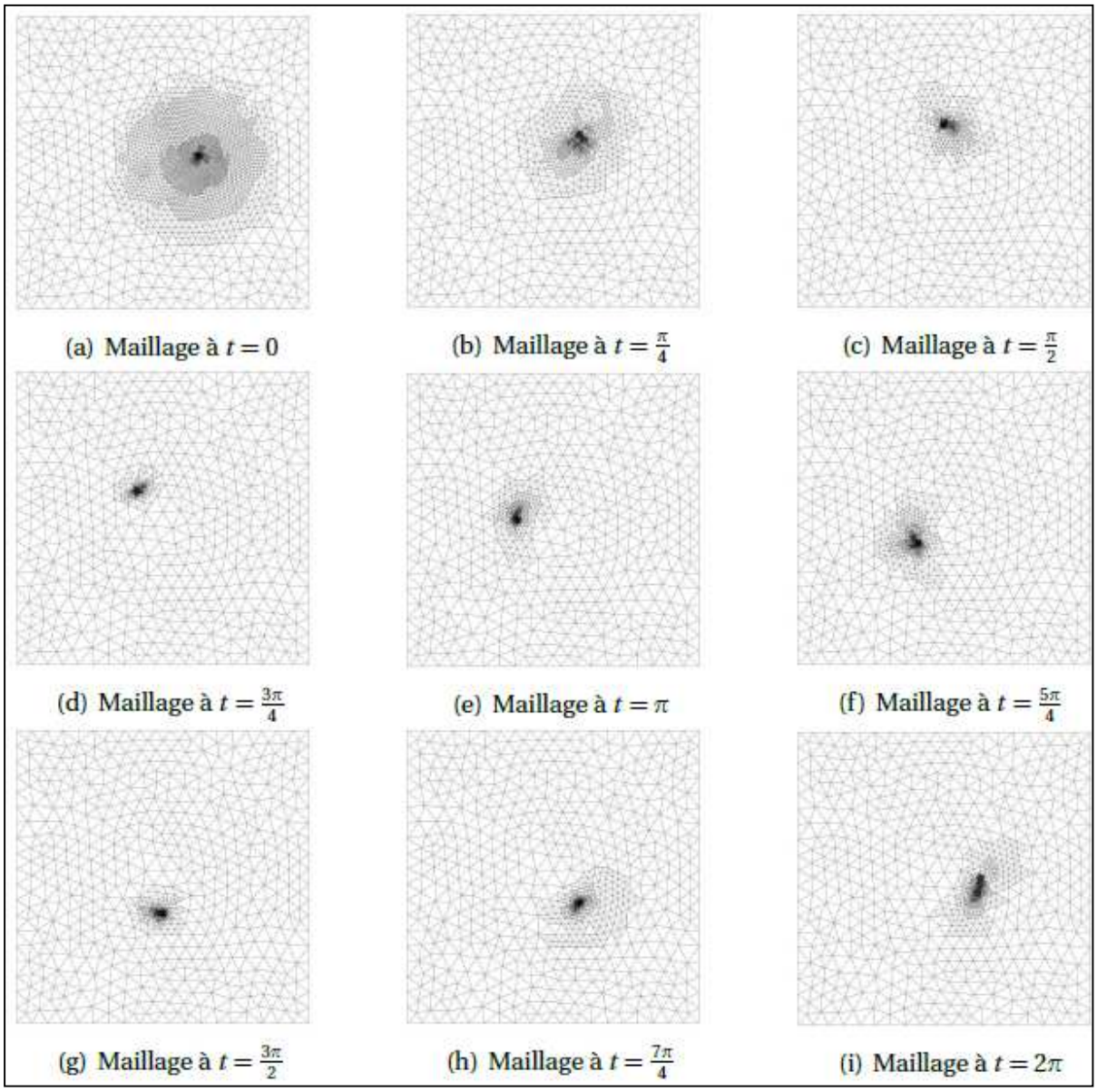


Figure 23: Résolution du problème de Laplace avec raffinement de maillage.

Nous avons tenté de résoudre ce problème dans trois configurations différentes :

- sans équilibrage de charge ;
- en utilisant l'équilibrage de charge grossier;
- en utilisant l'équilibrage de charge fin.

Nous avons réalisé ces calculs en utilisant 8 processeurs d'une machine massivement parallèle à mémoire distribuée reliés entre eux par un réseau rapide dédié (InfiniBand). Lors du lancement du calcul, nous avions choisi de limiter à 30 minutes la durée de nos calculs, afin de pouvoir disposer d'une priorité importante dans la file d'attente. Le résultat de cette expérience a été sans appel : alors que le calcul non équilibré et le calcul grossièrement équilibré n'ont pas pu se terminer avant la fin des 30 minutes allouées, le calcul finement équilibré a convergé au bout de 10 minutes et 34

secondes. Nous voyons donc clairement l'utilité des algorithmes d'équilibrage de charge à la lumière de cette expérience.

Le deuxième aspect concernant l'efficacité des algorithmes d'adaptation de la charge consiste à comparer leurs temps d'exécution avec celui de la résolution du même problème.

Là encore, les chiffres parlent d'eux mêmes, comme on peut le voir sur le Tableau 3 qui rassemble les statistiques d'exécution obtenues lors de la résolution du même problème.

Temps consacré à l'adaptation du maillage	3.85s	4.1%
Temps consacré à la résolution numérique	64.94s	69.2%
Temps consacré au post-traitement des champs	20.23s	21.6%
Temps consacré à l'équilibrage de la charge	1.42s	1.5%
Temps de résolution total	93.98s	100%

Tableau 3 : Statistiques d'exécution lors de la résolution du problème de Laplace avec raffinement de maillage et équilibrage dynamique de la charge en parallèle.

Nous voyons clairement que le temps pris par l'équilibrage de la charge reste parfaitement négligeable devant le temps nécessaire à la résolution du problème, ce qui confirme l'intérêt porté aux politiques d'équilibrage de la charge pour les calculs adaptatifs à vocation industrielle.

3 CALCUL INTENSIF ET PHYSIQUE DES REACTEURS

Résumé :

A la suite des travaux réalisés dans le domaine de la mécanique des fluides, je me suis intéressé à un autre grand domaine de la simulation numérique pour les systèmes nucléaires qu'est la physique des réacteurs en général et la neutronique en particulier.

Je me suis notamment penché sur les enjeux de l'utilisation du calcul intensif dans le domaine de la simulation numérique pour les systèmes nucléaires [35] afin d'identifier les apports du HPC dans le domaine, mais également des défis futurs à relever.

Ces études ont été déclinées de manière plus pratique pour la résolution de Boltzmann pour le transport des neutrons dans le cadre du code de simulation en neutronique déterministe APOLLO3® [47, 49].

Dans le cadre de la thèse de doctorat de J. DUBOIS [71], nous nous sommes intéressés à la programmation des nouvelles architectures de calcul comme les accélérateurs (GPU) en portant un solveur de l'équation de transport sur GPU. Nous avons également travaillé sur la ré-intégration de ces développements dans un code à vocation industriel comme APOLLO3® et mesuré les impacts. Cela nous a permis également de proposer en 2010 un des 1ers solveurs neutroniques mixant parallélisation par décomposition de domaines et accélération GPU sur chacun des domaines [53, 70].

Summary :

Following the work done in the field of fluid mechanics, I am interested in another major area of numerical simulation for nuclear systems which is reactor physics and more particular neutronics.

I particularly focused on the use of supercomputing in the field of numerical simulation for nuclear systems [35] to identify the contributions of HPC in this field as well as its future challenges.

These studies have been developed in a more practical way for solving the Boltzmann neutron transport equation in the frame of deterministic neutronics code APOLLO3® [47, 49].

As part of the thesis of J. DUBOIS [71], we worked on the programming of new computing architectures such as accelerators (GPU) and also transposing a transport equation solver on GPU. We also worked on the re-integration of these developments in a code industrial vocation as APOLLO3® and measured the impacts. This has also allowed us to propose in 2010 one the 1st neutronics solvers mixing parallelization by domain decomposition and GPU acceleration on each domain [53, 70].

3.1 ENJEUX DE L'UTILISATION DU CALCUL INTENSIF POUR LA SIMULATION DES SYSTEMES NUCLEAIRES

3.1.1 Introduction

Dans le domaine de la physique des réacteurs, de la neutronique et de la radioprotection, le calcul Haute Performance[5] permet des évolutions et des changements dans la façon d'utiliser la simulation numérique. Les évolutions des codes de calcul (algorithmes, méthodes numériques, …) et l'utilisation efficace de puissances de calculs croissantes font continuellement progresser la précision et la finesse des modélisations [35]. Les informations ainsi obtenues par la simulation numérique sont également plus complètes grâce à la généralisation des calculs tridimensionnels, la prise en compte de manière simultanée des phénomènes multi-physiques et d'un plus grand nombre de composants des centrales, comme par exemple la modélisation au sein d'une même simulation en trois dimensions du cœur du réacteur et de la chaudière en prenant en compte de manière couplée, les phénomènes neutroniques et thermo hydrauliques. Cela permet ainsi d'accéder à des précisions beaucoup plus importantes dans des temps de retour du même ordre de grandeur que des modèles simplifiés utilisés de manière courante.

Ces progrès constants dans le domaine de la modélisation et de la simulation numérique sont indispensables pour :

- Augmenter les marges en réduisant les incertitudes;
- Optimiser les conceptions des installations;
- Améliorer la sûreté;
- Optimiser le fonctionnement des installations;
- Améliorer la connaissance de la physique.

En fonction des objectifs de simulation que l'on cherche à atteindre à l'aide du HPC, différentes techniques peuvent être utilisées. On peut classer en quatre grandes catégories les objectifs que le HPC contribue à atteindre :

- Les calculs multi-paramétrés: c'est la technique de base pour l'optimisation. Le HPC est un excellent outil pour prendre en compte plus de paramètres et réduire le « time to market » de la solution optimisée. Cela permet également l'utilisation de techniques d'apprentissage, comme les réseaux de neurones, afin de trouver automatiquement un ensemble optimisé de paramètres.
- Une physique à très haute résolution: s'approcher le plus possible de la physique et de tenter de simuler au plus près l'ensemble des échelles des phénomènes physiques nécessite immanquablement une plus grande capacité mémoire et plus de puissance de calcul.
- Une physique plus réaliste en utilisant systématiquement le modèle physique le plus réaliste au lieu du modèle simplifié ou des valeurs pré-établies. Cela implique, non seulement une plus grande puissance de calcul, mais également des systèmes couplés robustes et facile d'utilisation.

[5] Le Calcul Haute Performance (High Performance Computing – HPC en anglais) désigne toutes les méthodes de traitement d'applications complexes par des machines puissantes capables de gérer d'importants volumes d'informations numériques.

- Simulation en temps réel : ce qui existe déjà, mais on peut imaginer améliorer la modélisation afin d'obtenir des simulateurs plus réalistes et diminuer le nombre d'hypothèses.

Nous allons, dans les paragraphes qui suivent, donner les grandes lignes actuelles et futures de l'utilisation et du développement du HPC via les codes de transport neutronique en Monte Carlo et déterministe.
Ce chapitre ne se veut certainement pas exhaustif sur les façons dont le HPC peut ou pourrait être utilisé dans les domaines de la physique des réacteurs, de la neutronique et de la radioprotection.

3.1.2 Le HPC pour les simulations en Monte Carlo

La méthode de Monte Carlo pour résoudre l'équation du transport des neutrons est un excellent candidat pour le parallélisme. Du fait que le processus de simulation est basé sur un traitement statique des trajectoires indépendantes des particules, chaque particule peut être calculée par un processeur différent. Les méthodes de Monte Carlo sont dites naturellement parallèle (« *embarrassingly parallel* »).

3.1.2.1 Principes pour une mise en œuvre parallèles

Afin de contrôler le processus de simulation, il faut un moniteur qui est en charge de contrôler l'ensemble de la simulation et un autre processus qui est en charge de recueillir les résultats des simulations. Ces deux processus peuvent être exécutés sur un même processeur ou sur des processeurs différents. Le modèle utilisé est une approche MPMD[6], sur la base d'un modèle maître / esclave avec l'identification d'un maître en charge du contrôle de la simulation et la séparation des processus de simulation et des processus de récupération [39,40,41,42].
Ce principe est utilisé sur la Figure 24 avec le code TRIPOLI-4 ™ [35,37,38,39].

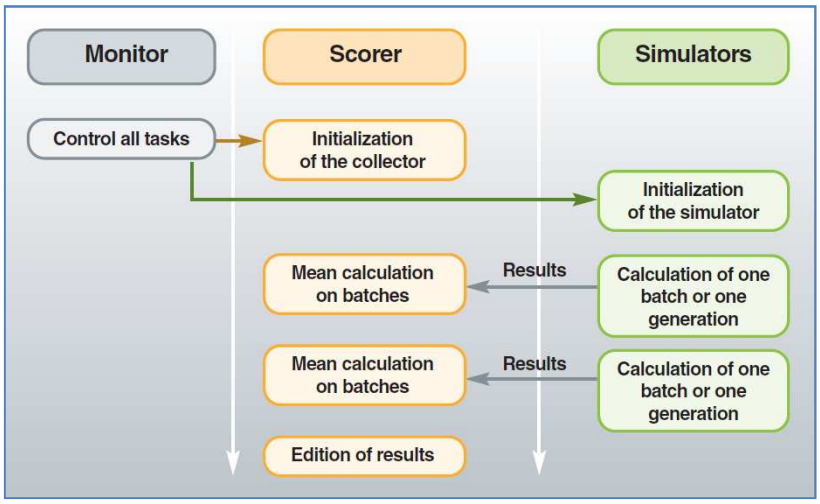


Figure 24: Principe de mise en œuvre du parallélisme dans une simulation MC – Application au code TRIPOLI-4™

[6] Multiple Program Multiple Data : des programmes différents s'exécutent en parallèle utilisant des données différentes.

Les principaux avantages de ce modèle de parallélisme dans le cade de simulations Monte Carlo, est qu'il est très facile à implémenter et à porter (échanges d'informations basiques entre les processus), il est bien adapté à la simulation des particules indépendantes. Le dernier avantage est que cette mise en œuvre est tolérante aux pannes. En effet, si l'un des simulateurs tombe, que cela soit dû à un défaut matériel ou à un problème numérique (non convergence), le moniteur permet de relancer les lots correspondants sur un autre simulateur. Grâce à ces caractéristiques, même si plus de la moitié des simulateurs est perdue, la simulation peut encore continuer.
D'autres caractéristiques ont également été mises en œuvre dans le TRIPOLI-4™, comme des procédures de sauvegarde/reprise : l'état courant de la simulation TRIPOLI-4™ est stocké dans des fichiers différents qui permettent la reprise de la simulation à tout moment.
D'un point de vue mathématiques et algorithmique, le principal problème concerne la version parallèle du générateur de nombres aléatoires [44]. Celui-ci doit répondre aux propriétés suivantes:

- Indépendance par rapport au nombre de processeurs;
- Pas de corrélation entre les suites pour différents processeurs;
- Sur chaque processeur, le générateur peut être initialisé de façon indépendante (pas de communication entre processus).

Un exemple de ce générateur aléatoire parallèle est basé sur l'algorithme GFSR (Generalized Feedback Shift Register) [43]. Les principaux avantages de ce générateur de nombres aléatoires sont qu'il est très rapide et qu'il a la propriété d'offrir une très longue période (2607)[7].

3.1.2.2 Performances

La façon d'évaluer les performances parallèle de simulations Monte Carlo est tout à fait différente de la mesure classique d'efficacité : elle est basée sur le « Figure Of Merit » (FOM). Cette quantité est définie comme suit :

$$FOM = \frac{1}{\sigma^2 \times \tau}, \text{où } \sigma \text{ est la déviation standard et } \tau \text{ le temps écoulé}$$

On s'attend à ce que l'accélération ou le FOM augmente de manière linéaire avec le nombre de processeurs. Ce qui est généralement vérifié (voir Figure 25).

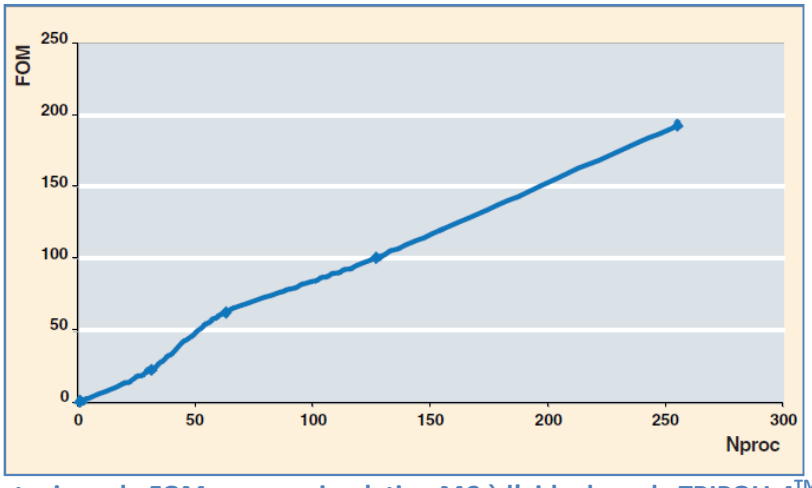


Figure 25: Evolution typique du FOM sur une simulation MC à l'aide du code TRIPOLI-4™.

[7] La période d'un générateur de nombres « pseudo-aléatoires » correspond grossièrement à la fréquence à laquelle le générateur retrouve son état initial et donc génère un nombre déjà produit auparavant.

Ce genre de comportement est régulièrement vérifié pour des simulations allant jusqu'à 1 000 processeurs ou l'efficacité reste de l'ordre de 90%.
Un des avantages de la mise en œuvre parallèle du code TRIPOLI-4T^{M}, est que le moniteur permet de contrôler dynamiquement l'équilibre de la charge de chaque simulateur, ainsi le nombre et la taille des lots peut être réglé afin que les simulateurs aient une attente minimale et ainsi maximiser l'efficacité globale de la simulation.
Cependant, il existe tout de même des goulots d'étranglement pour obtenir un rendement idéal. Même si la performance peut être ajustée, le collecteur peut être un frein à une accélération optimale du calcul pour un très grand nombre de simulateurs (>> 1 000). Certaines solutions sont à étudier, comme la mise en œuvre d'un arbre de collecteurs, afin de partager la charge de la collecte des résultats. Une autre solution pourrait être d'éviter la collecte en ligne et accumuler les résultats après la simulation. Cependant il faut être prudent avec la quantité de données à stocker.

3.1.2.3 Les défis à venir

Même si les méthodes de type Monte Carlo sont naturellement parallèles, l'accès à l'échelle du Pétaflop et au-delà (au-dessus de 10 000 cœurs) reste un défi, comme pour beaucoup d'autres applications.
Comme nous l'avons mentionné plus tôt, nous aurons à faire face à l'énorme quantité de données, soit sur le réseau ou le système de fichiers, afin de stocker les résultats des simulations.
Mais si nous continuons à augmenter la complexité de la géométrie ainsi que le nombre des variables de calcul (par exemple, les taux de réaction sur un cœur complet de réacteur à l'échelle du crayon ou de la pastille), le stockage de toutes ces informations dans la mémoire pose problème. En effet, dans le modèle de parallélisme utilisé, toutes les données d'entrée (géométrie, données nucléaires ...) et les données de sortie sont dupliquées sur chaque processeur (voir Figure 26). Ainsi, la limite n'est pas seulement le temps de calcul comme aujourd'hui, mais la limite de mémoire. Une solution serait de combiner le parallélisme de tâches avec un parallélisme de données, et de diviser le domaine de calcul afin de le répartir entre les différents processeurs (voir Figure 27). Mais ce parallélisme est contre nature pour des simulations MC et conduit à des problèmes délicats tels que le déséquilibre de charge ainsi que des pertes d'efficacité en raison du trafic de communications entre les processeurs.

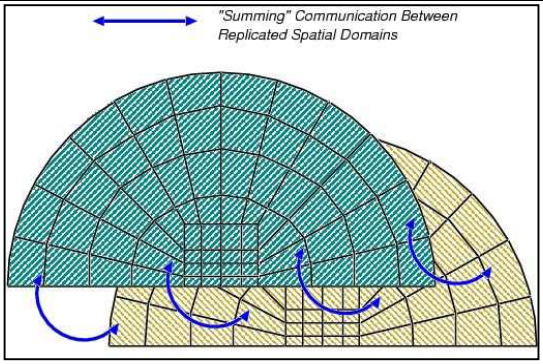


Figure 26 : Réplication de domaines (parallélisme suivant les particules).

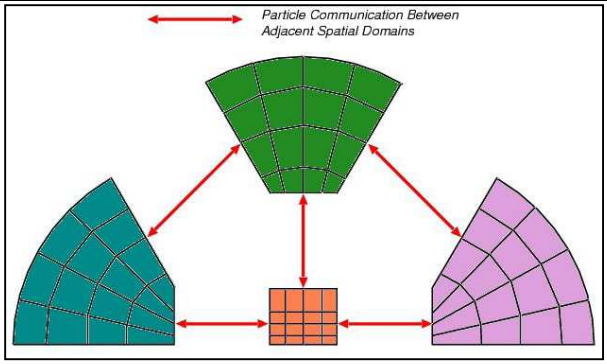


Figure 27 : Décomposition de domaines (parallélisme spatial).

Dans le code MERCURY, les deux modèles sont mis en œuvre [45], afin de profiter des avantages des deux modèles de parallélisme : spatial et suivant le parcours de particules (voir la Figure 28).

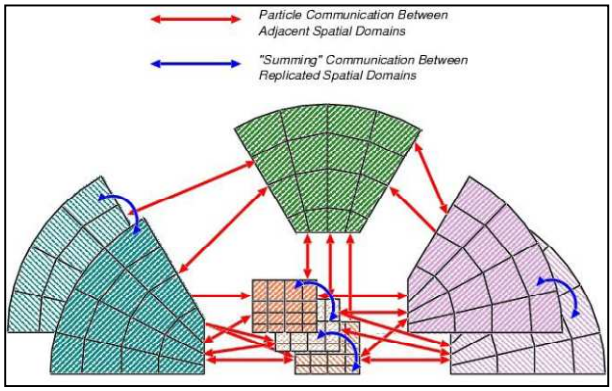


Figure 28 : Décomposition de domaines et réplication de domaines (parallélisme spatial et suivant les particules).

Ce type d'approche est également à l'étude autour du code MNCP où des paradigmes innovants de transferts de données sont à l'étude pour combiner parallélisme de tâches et parallélisme de données dans une simulation Monte Carlo [46].

Ces approches ne prennent pas totalement en compte l'évolution l'architecture des machines. Une autre solution, permettant de résoudre le problème de la duplication des données en mémoire, est de tenir compte du fait que le nombre d'unités de calcul partageant une mémoire commune est en constante augmentation. Ainsi l'idée est de n'avoir qu'une seule copie des données au sein d'un nœud de calcul possédant plusieurs unités de calcul qui se partagent ces données, et de les dupliquer entre les différents nœuds de calcul. Ce principe est illustré sur la Figure 29.

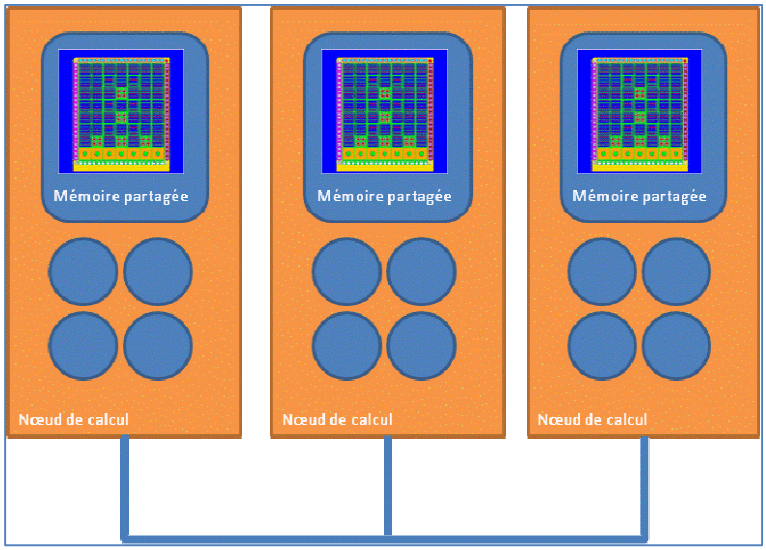


Figure 29 : Principe de partage des données au sein d'un même nœud de calcul et de duplication entre nœuds de calculs.

Il existe bien d'autres champs d'exploration du calcul haute performance et du transport Monte Carlo parmi lesquels on peut citer l'utilisation des accélérateurs graphiques, GPU, ou Intel Xeon Phi.

3.1.3 Le HPC en déterministe

3.1.3.1 Les différents niveaux de parallélisme

Dans les approches déterministes, on peut considérer trois niveaux principaux de parallélisme:

- Le premier niveau concerne les calculs multi-paramétrés. Il s'agit de réaliser un ensemble de calculs indépendants les uns des autres sur un ensemble de ressources de calcul. Il s'agit dans ce cas de figure de gérer les flots de calculs et de données (données d'entrées et résultats) sur l'ensemble des ressources de calcul. Un exemple classique est le calcul d'une bibliothèque multi paramétrée pour le calcul de cœur. Les calculs des différents assemblages et conditions de chargement sont tous indépendants les uns des autres et peuvent donc être calculés en même temps.
- Le second niveau correspond aux calculs multi-domaines. Il s'agit ici de gérer en parallèle un ensemble de calcul opérant sur des données qui ne sont pas forcément indépendantes. Par exemple, cela concerne tous les traitements portant sur les sections efficaces, calcul d'évolution, les contre-réactions thermo-hydrauliques ... Dans ce niveau, en général on peut considérer que la dépendance spatiale des données est très légère et une approche massivement parallèle est bien adaptée. Pour les cas où la dépendance spatiale des données est forte, des techniques de décomposition de domaine sont généralement utilisées. C'est typiquement le cas lors de la résolution d'équations de transport.
- Le troisième niveau concerne principalement la parallélisation à grain fin de solveurs en exploitant le parallélisme intrinsèque des méthodes numériques considérées.

Dans la suite nous allons illustrer ces différents niveaux, en décrivant quelques exemples typiques [47].

3.1.3.2 Premier niveau : calculs multi paramétrés.

Le principal intérêt de ce niveau est d'utiliser la force brute du HPC pour résoudre des problèmes comportant d'énormes quantités de calculs indépendants dans un temps à échelle «humaine». Nous présentons dans la suite différentes utilisations du HPC permettant par exemple de réduire les incertitudes des simulations, ou de gagner des marges à l'aide de techniques d'optimisations.

Une façon de réduire les incertitudes est bien évidemment d'utiliser des approches déterministes. Une autre façon consiste à utiliser une méthode moins intrusive basée sur une approche stochastique (échantillonnage). Bien sûr, cette approche est contestée en ce qui concerne les ressources de calcul nécessaires. Mais cette méthode d'échantillonnage se révèle très intéressante lorsque l'approche déterministe est trop complexe à mettre en œuvre. On peut citer les problèmes couplés (thermo - neutronique), ou les calculs d'évolutions.

En ce qui concerne les problèmes d'optimisation, un très bon exemple est l'optimisation de plans de chargement de cœurs à l'aide d'algorithmes génétiques [48]. Un outil basé sur URANIE / VIZIR et le code APOLLO3® [49] a été conçu et appliqué avec succès à l'optimisation du chargement du combustible modèle dans le cas de cœurs hétérogènes de Réacteurs à Eau Pressurisée (REP). Cet outil permet l'évaluation de plus de dix millions de configurations différentes en moins de 10

heures en utilisant plus de 4 000 processeurs. Une illustration de différents types de solutions est présentée sur la Figure 30.

Le principal avantage de ce type de méthode est de permettre aux ingénieurs de tester différentes configurations et de relâcher certaines contraintes de conception qui n'auraient pas été possible sans l'utilisation conjointe des algorithmes génétiques et du HPC. Des exemples de différentes solutions sont donnés sur la Figure 31.

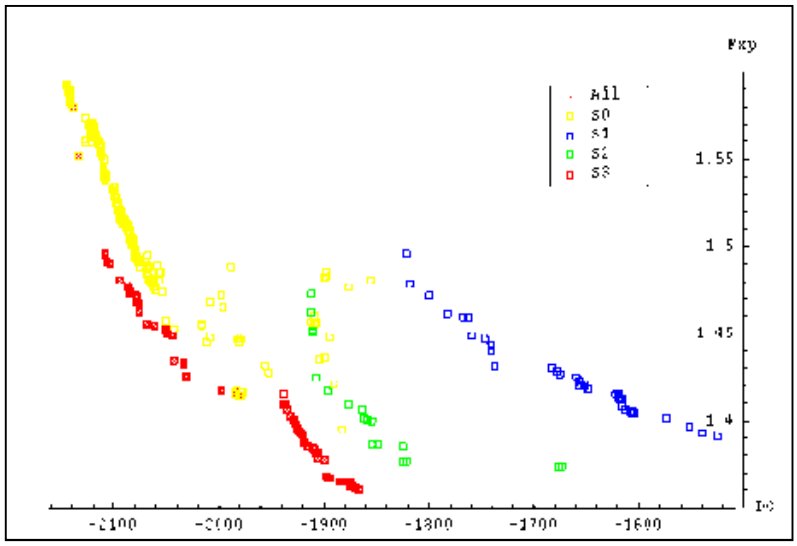


Figure 30 : Exemple d'ensembles de solutions trouvés par algorithmes génétiques suivant différentes stratégies d'optimisation (front de Pareto).

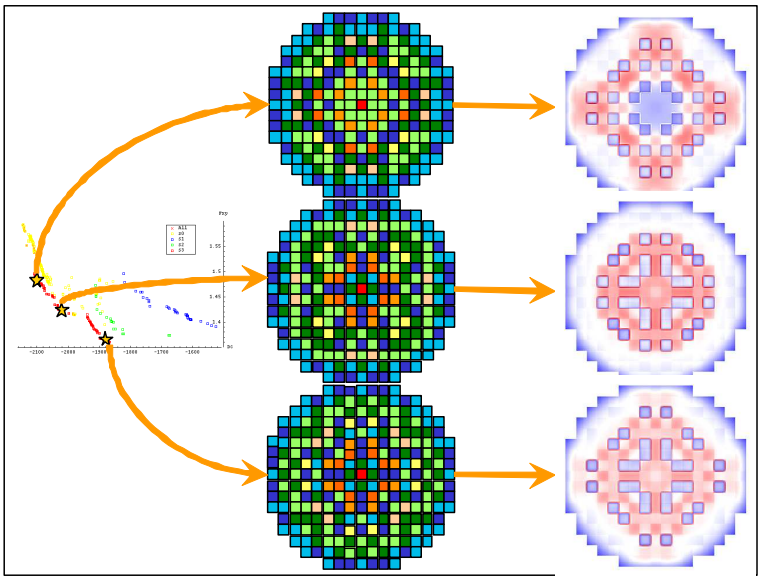


Figure 31 : Illustration de différentes solutions de plan de chargement : Fronts de Pareto sur la gauche – Plans de chargement au milieu – Nappes de puissances correspondantes à droite.

Ce type de démarche se généralise au CEA et s'applique désormais en support aux études ASTRID avec l'outil TRIAD. Il est intéressant de noter qu'à partir d'un exercice de prototypage, suite à

l'accès d'une grande ressource de calcul, cela permet de montrer aux physiciens et concepteurs les potentiels d'un outil et d'une approche grâce au HPC. Grâce à l'augmentation de la puissance de calcul, ce qui n'était qu'un exercice, peut devenir un outil de conception pour l'aide au design d'un réacteur.

3.1.3.3 Deuxième niveau : calculs multi domaines.

Ce niveau est le plus classique. Dans la plupart des applications scientifiques parallèles ce niveau est utilisé via la technique de décomposition de domaines. Appliquée à la neutronique, tous les calculs qui sont spatialement indépendants sont concernés. Par exemple, dans un calcul typique en 2 étapes, au niveau du calcul de cœur, toutes les étapes concernant la gestion des sections efficaces, les contre-réactions, l'évolution isotopique, ... sont locaux à la cellule du domaine géométrique, et peuvent donc se faire en parallèle. Pour résumer, dans un calcul standard de cœur en 3D, toutes les étapes sont spatialement indépendantes et peuvent donc se faire naturellement en parallèle. Toutes les étapes, sauf une, le calcul du flux. Même si toutes les étapes précédentes peuvent se faire en parallèle, le principal problème est toujours la gestion du flux de données et la distribution des données entre les processus. Il est donc nécessaire de bien penser à la conception de l'architecture du code afin d'avoir des structures de données optimales permettant de faciliter et d'optimiser la gestion des flux de données en parallèle.

En ce qui concerne le calcul du flux lui-même, des techniques classiques de décomposition de domaine peuvent être utilisées [52]. Concernant le solveur de l'équation du transport [55], des degrés supplémentaires de parallélisme peuvent être trouvés, car les autres dimensions de l'équation peuvent être utilisées, par exemple les dimensions angulaires ou énergétiques. Beaucoup de solveurs ont été mis en œuvre en parallèle, en exploitant le parallélisme soit angulaire et énergétique [53,54,61] soit la dimension spatiale [59,60,56] ou les deux [58].

Nous illustrons sur la Figure 32 les performances obtenues avec le solveur transport MINARET du code APOLLO3® par parallélisation suivant les directions angulaires. L'exercice, qui s'est déroulé sur les supercalculateurs TITANE du CCRT et TERA-100 du CEA/DAM, a permis de valider l'architecture et le comportement numérique du solveur pour une utilisation du parallélisme massif, jusqu'à 33.000 cœurs de calcul, sur un calcul réel : calcul de cœur 3D de type ESFR (même si pour des besoins de modélisation physique, il n'est pas nécessaire dans ce ca précis d'utiliser des ordres aussi grands).

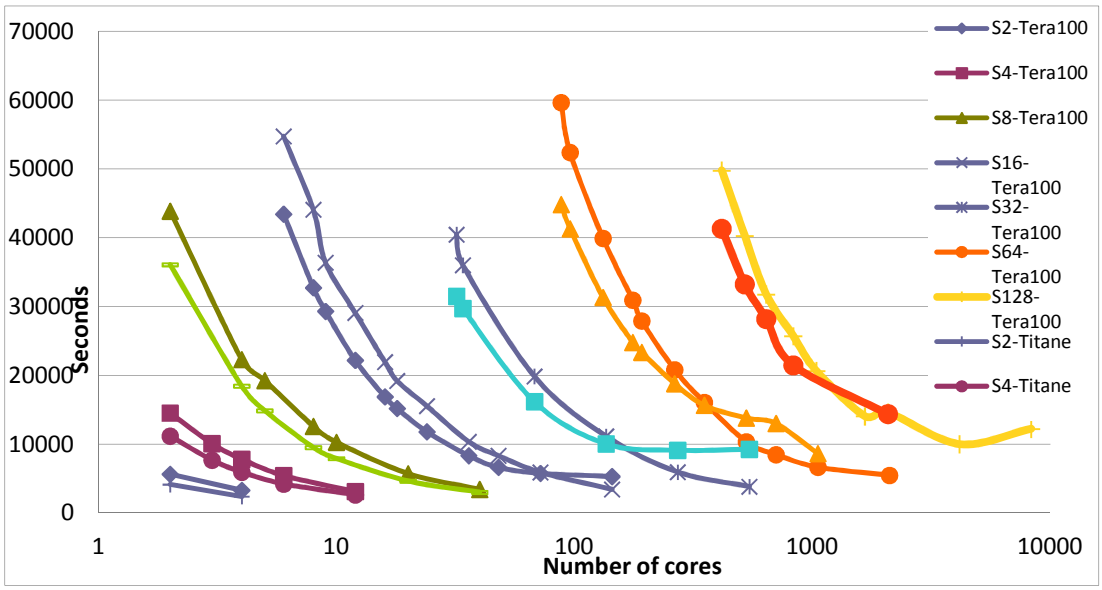


Figure 32 : Temps de calcul d'une simulation 3D avec le solveur transport MINARET du code APOLLO3® sur les supercalculateurs TITANE (CCRT) et TERA-100 (CEA/DAM).

Un autre degré de parallélisme peut être utilisé lors de l'exploitation des techniques multi niveaux. Par exemple le couplage d'une solution fine au niveau d'un assemblage de l'équation du transport avec une solution grossière 3D cœur. Des exemples typiques de ces techniques peuvent être trouvés dans le code COBAYA [62] (voir Figure 33) ou appliqués sur le solveur diffusion MINOS au sein d'une approche originale basée sur une synthèse mode composante [57].

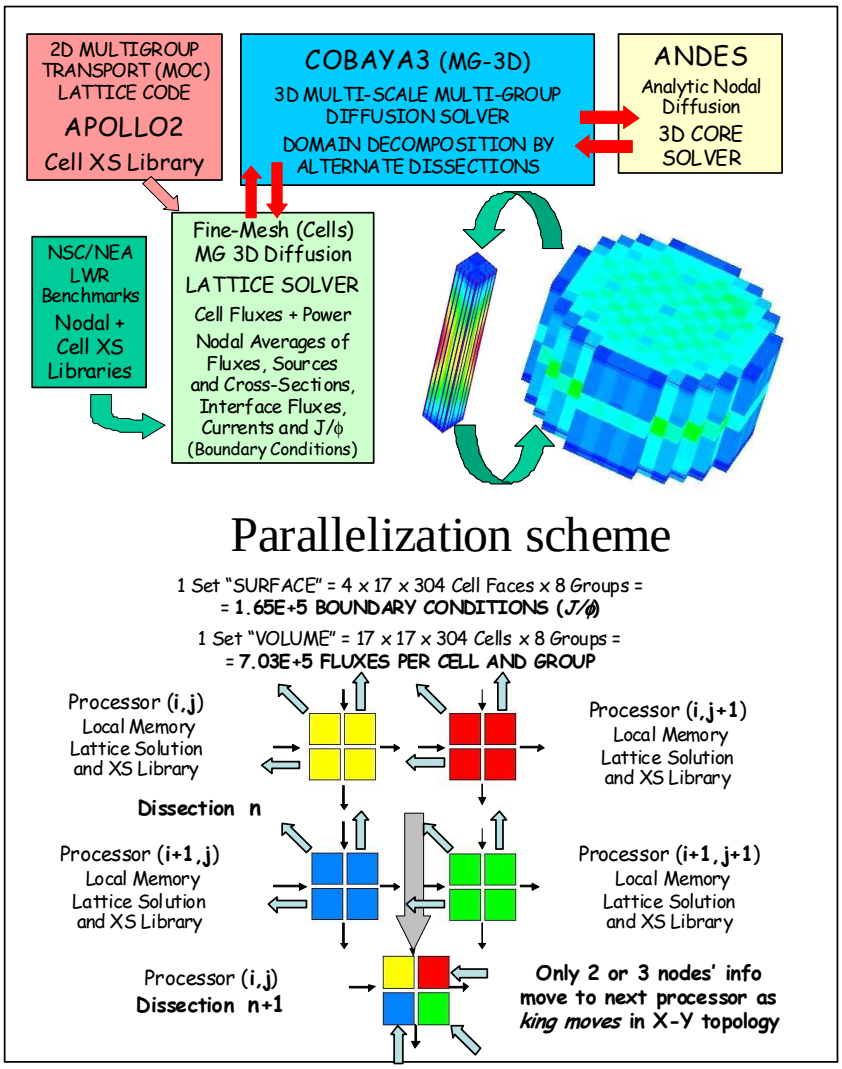


Figure 33 : Principe de parallélisme utilisé dans le code COBAYA (Crédit : Projet NURESIM)

Cette approche multi-niveaux du parallélisme est également bien adapté pour les calculs 3D, en particulier lorsque la méthode est intrinsèquement peu parallèle ou très difficile à paralléliser. Un exemple typique est la méthode des caractéristiques utilisée pour trouver la solution de l'équation du transport des neutrons. En effet, il est possible de paralléliser cette méthode, mais il faut employer des techniques très avancées en vue d'obtenir une efficacité intéressante [63,64,65]. Une autre façon est de coupler une approche 2D avec un couplage dans la troisième direction, et de calculer des plans 2D en parallèle (voir Figure 34). Cette approche est utilisée par exemple dans le code Dekart [66] dans le code UNIC [67].

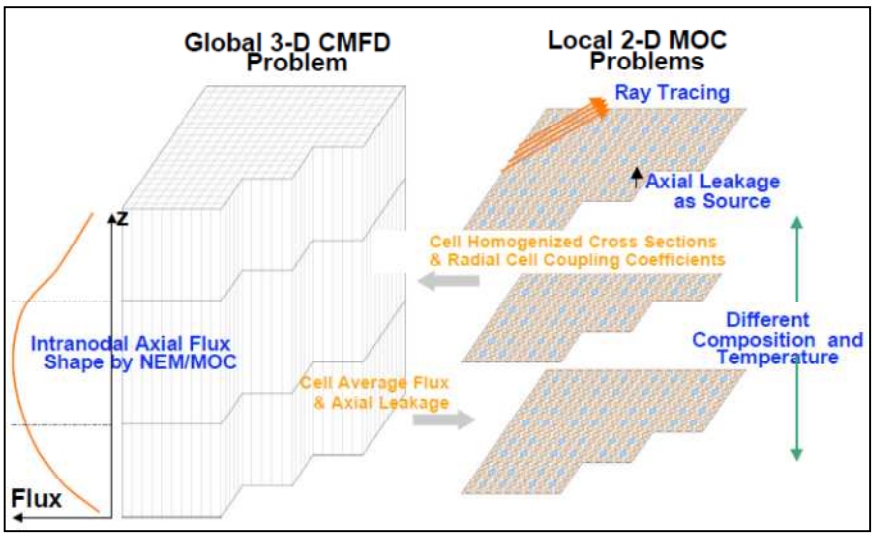


Figure 34 : Méthode MOC 3D à partir de synthèse de calculs 2D MOC plans.

3.1.3.4 Troisième niveau : parallélisme a grains fins

Ce niveau est généralement utilisé sur des architectures à mémoire partagée et exploite le parallélisme intrinsèque des algorithmes. Ces techniques ont eu beaucoup de succès au début des années 2000 avec le langage HPF puis OpenMP [59,60,68]. Il devient de plus en plus intéressant sur les architectures multi-cœurs actuelles de combiner ce niveau grain fin avec le niveau précédent afin d'améliorer les performances globales des algorithmes.

3.1.3.5 Parallélisme Hybride

La course à la puissance a conduit à utiliser désormais dans les supercalculateurs des accélérateurs de calcul, comme par exemple des cartes graphiques (GPU). Ces architectures de nouvelle génération présentent l'avantage d'offrir un rapport performance de calcul sur consommation électrique très intéressant. Cependant elles entrainent une hétérogénéité supplémentaire dans la programmation des codes pour les utiliser.

Des premières expérimentations ont été réalisées dans le cadre du projet APOLLO3® autour du solveur transport simplifié MINOS. Ainsi il est possible de mixer une approche décomposition de domaines et une accélération sur carte graphique [53].

Nous présentons sur la Figure 35 les résultats obtenus sur un calcul 3D diffusion 2 groupes cellule par cellule d'un cœur de REP. Comme on peut le constater, les accélérations jusqu'à 30 unités de calcul sont très intéressantes par rapport à des CPUs classiques.

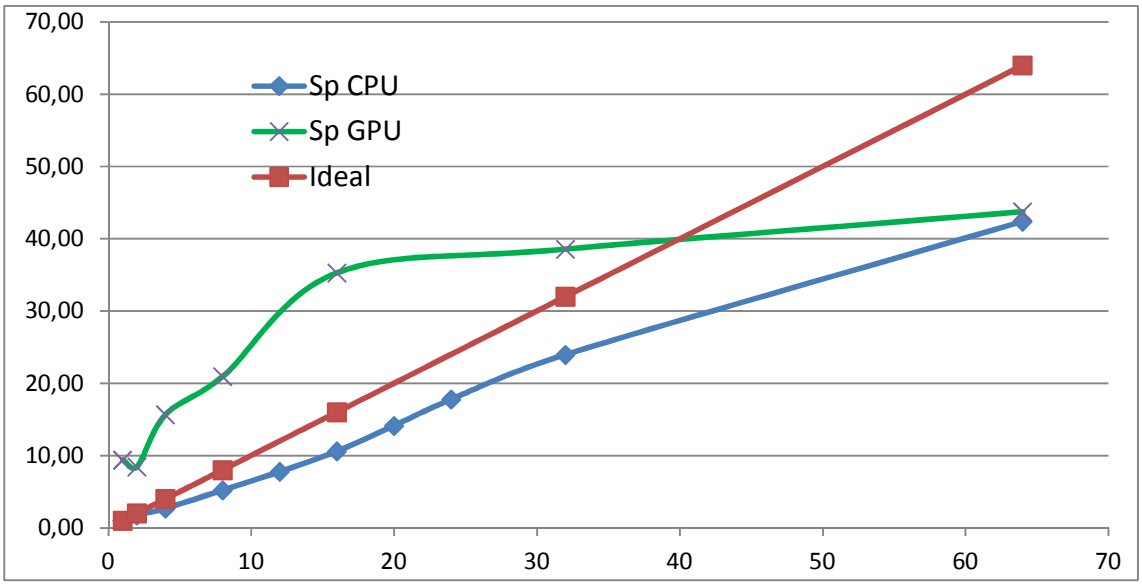


Figure 35 : Accélérations (Sp pour Speedup) comparées entre CPU (courbe bleue), GPU (courbe verte) et accélération idéale (courbe rouge). En abscisse le nombre d'unités de calcul.

Nous détaillons dans la section suivante les travaux sur l'utilisation des accélérateurs pour la résolution de l'équation de Boltzmann [3] dans le cadre du code APOLLO3®.

3.2 UTILISATION D'ACCELERATEURS DE CALCUL POUR LA RESOLUTION DE L'EQUATION DU TRANSPORT DES NEUTRONS

3.2.1 Introduction.

La résolution de l'équation du transport des neutrons permet de modéliser, comme son nom l'indique, le transport des neutrons dans un système nucléaire. Pour simuler le comportement d'un réacteur nucléaire, la résolution de cette équation est le cœur du calcul. Elle correspond à la résolution d'un problème à valeurs propres généralisé.

La simulation 3D d'un cœur de REP peut se contenter d'une simplification de la résolution de l'équation du transport par une approximation du transport (SPn).

C'est cette approche qui est utilisée dans le solveur MINOS du code APOLLO3® [49, 50, 51]. Des travaux précédents ont été réalisés pour paralléliser ce solveur en utilisant des méthodes de décomposition de domaines [56, 57]. Dans le cadre de la thèse de doctorat de Jérôme DUBOIS [71], nous nous sommes intéressés à l'accélération de ce solveur par l'utilisation de GPUs et mixer l'approche décomposition de domaines et multi-GPUs [69, 70].

3.2.2 Implémentation sur GPU.

L'objectif était de minimiser l'impact sur l'algorithme originel quitte à ne pas être optimal en performances. La structure de données de base dans MINOS est le vecteur et ainsi toutes les opérations d'algèbre linéaire sont de type vectoriel et sont codées dans les classes MINOS Utilities. Le principe d'implémentation sur GPU est alors simple : les méthodes de type BLAS1, BLAS2 sont implémentées via la bibliothèque CUBLAS et nous devons gérer les échanges de données entre CPU et GPU. Afin de minimiser les changements au sein de MINOS et de gérer au mieux les échanges,

nous avons donc défini un nouveau type de vecteur qui accepte une instance sur CPU ou sur GPU et qui gère de manière transparente les transferts de données (cf. Figure 36)

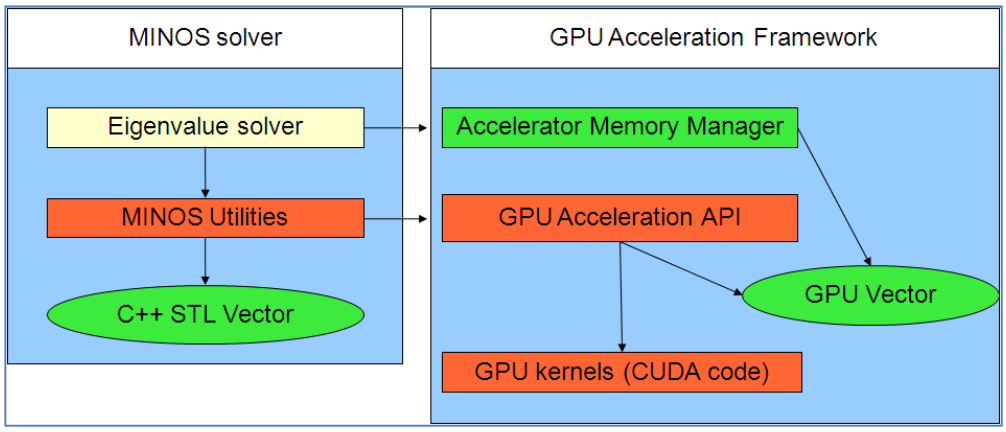


Figure 36 : Schéma de principe du « framework » d'accélération sur GPU du solveur MINOS.

3.2.3 La parallélisation du solveur MINOS par décomposition de domaines.

Pour paralléliser le solveur MINOS, une méthode de décomposition de domaines de type Schwarz additive a été utilisée. Par rapport à la formulation standard des équations de MINOS, une condition limite de type albedo a été rajoutée. L'algorithme parallèle entre deux processus MPI est présenté sur la Figure 37.

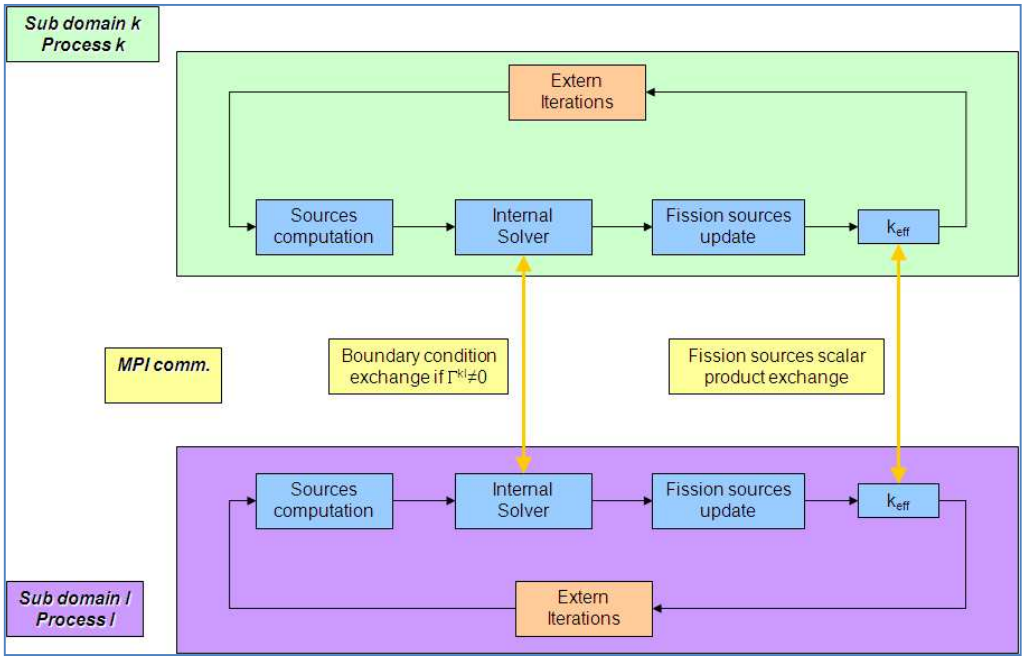


Figure 37 : Algorithme parallèle du solveur MINOS.

Pour combiner accélération GPU et décomposition de domaines via MPI, le principe est simple : sur chacun des sous-domaines, les opérations vectorielles et matricielles sont réalisées sur GPU et la méthode de décomposition de domaines est appliquée de la même façon que sur pur CPU. Cependant, étant donné qu'il ne peut y avoir d'échanges directs entre GPU, il est nécessaire de

repasser par le CPU pour réaliser les communications. Pour cela nous nous sommes servis du gestionnaire mémoire indiqué dans la section précédente. Le principe est résumé sur la Figure 38.

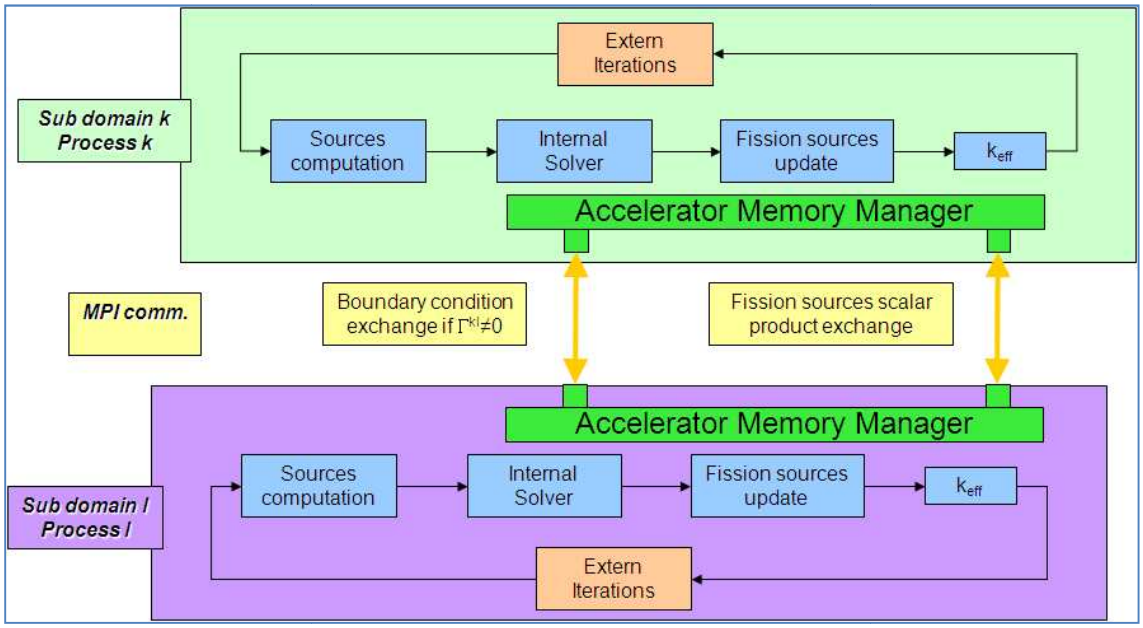


Figure 38 : Algorithme parallèle du solveur MINOS en multi GPUs.

3.2.4 Résultats de performances

Les cas tests utilisés sont des calculs de réacteurs de type REP (Figure 39), cas Hete3D, et de recherche (Réacteur Jules Horowitz - RJH) (Figure 40) (cas RJH2).

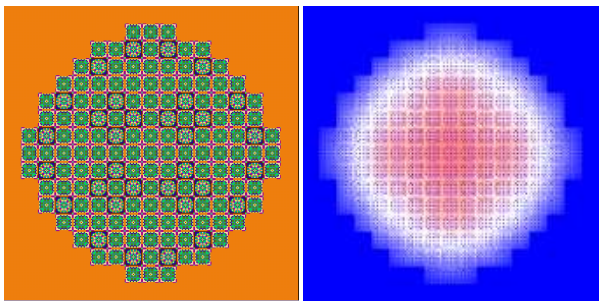


Figure 39 : Géométrie et nappe de puissance d'un REP en approche hétérogène - Diffusion à 2 groupes d'énergie.

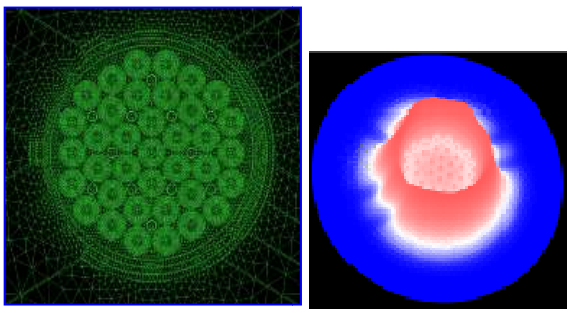


Figure 40 : Géométrie et nappe de puissance du RJH en approche hétérogène - SPn à 2 groupes d'énergie.

Tout d'abord nous présentons dans le Tableau 4, les résultats d'accélérations obtenus sur un seul GPU. Dans ces expérimentations nous avons comparé un GPU NVIDIA Tesla S1070 par rapport à un processeur Intel Nehalem à 2.93 GhZ.

	Hete3D Diffusion	RJH2 SP1	RJH2 SP3
CPU solver	284	1382,46	2390,32
GPU solver	52	572,71	932,52
Speedup	**5**	**2**	**3**

Tableau 4. Accélérations obtenues sur 1 GPU NVIDIA Tesla S1070 par rapport à un processeur Intel Nehalem à 2.93 GhZ.

Nous avons obtenu une meilleure accélération sur le cas Hete3D que sur les cas RJH2, du fait que le volume de calcul est beaucoup plus important sur le 1er cas test (calcul 3D) que sur les deux autres cas tests (calculs 2D). Les accélérations restent relativement modestes, mais il faut rappeler que l'objectif premier était de minimiser les impacts sur le code initial. Ainsi, en moins de 2 semaines, nous avons pu produire un code accéléré sur GPU quasi identique au code source initial.
Une autre expérimentation, menée sur une maquette, en relâchant quelques unes des contraintes inhérentes au code nous ont permis d'obtenir des accélérations beaucoup plus intéressantes (Figure 41). On voit donc que la portabilité et la généricité ont un coût non négligeables mais également qu'il n'y a pas de miracles : prendre comme contrainte initiale de conserver quasiment tel que le code source initial ne peut conduire à obtenir de bonnes performances.

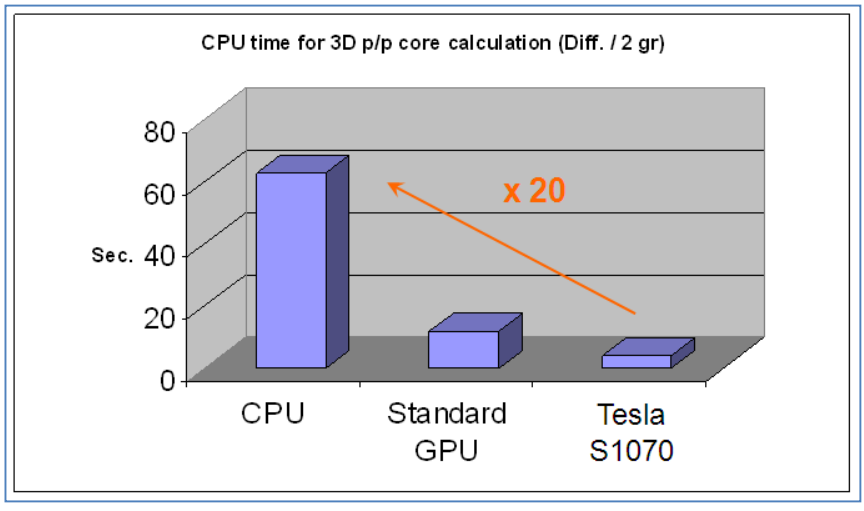


Figure 41 : Accélération sur une maquette de MINOS en GPU sur le cas test Hete3D.

Les Figure 42 et Figure 43 présentent les résultats de performances obtenues en décomposition de domaines pure.
Comme on peut le constater, les performances parallèles sont fortement dépendantes des cas. Sur le cas Hete3D, l'efficacité chute rapidement, puisque dès 4 processeurs elle est à 50%. Par contre pour le cas RJH2, l'efficacité reste excellente jusqu'à 128 processeurs. Il faut voir également que le 1er cas est un calcul de type industriel dont l'objet est d'aller le plus rapidement possible en

séquentiel, et donc les options de modélisation ont été choisies en conséquence. Cela remet également en perspective plus intéressante le facteur 5 obtenu avec l'utilisation d'un accélérateur.

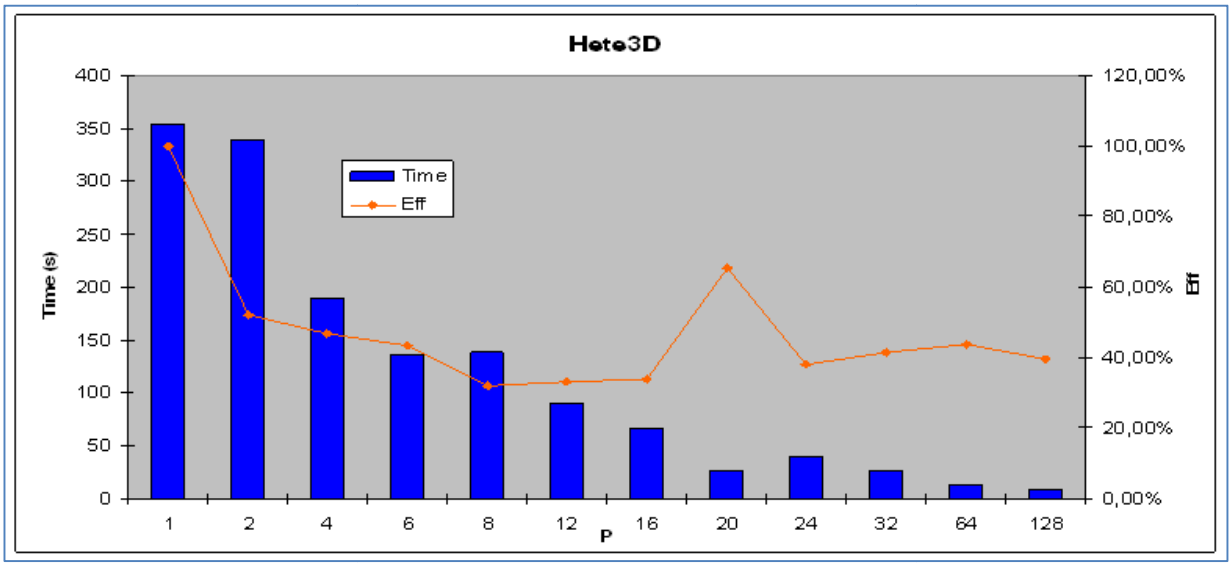


Figure 42 : Performance de la méthode de décomposition de domaines sur CPU pour le cas Hete3D.

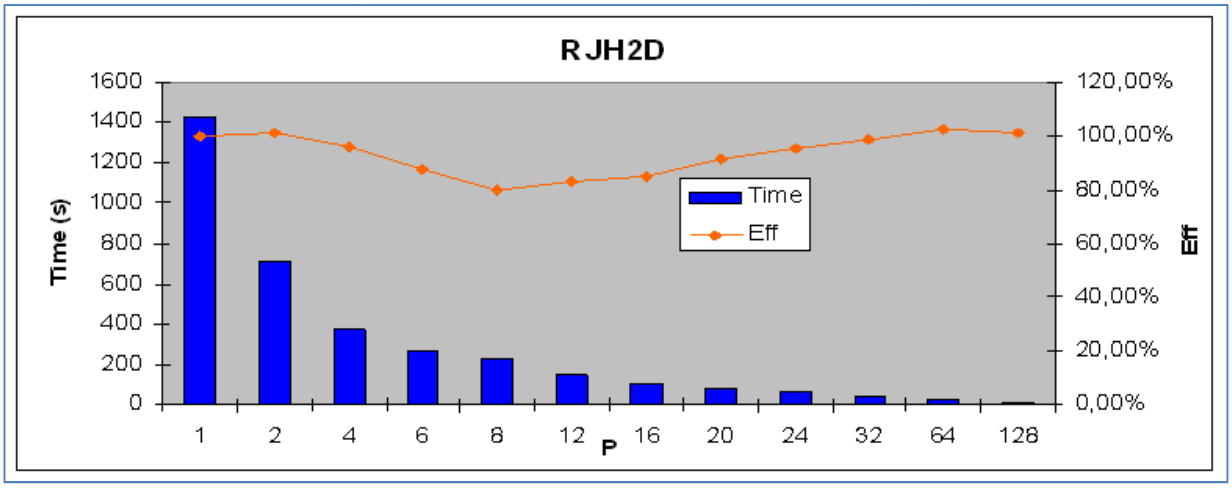


Figure 43 : Performance de la méthode de décomposition de domaines sur CPU pour le cas RJH2D.

Autre élément intéressant, ces résultats indiquent également que le cas intéressant pour l'accélération en GPU ne l'est pas en parallèle de type MPI et vice et versa. Cela illustre toute la complexité de mixer les différents niveaux de parallélisme.

Nous avons également étudié les performances mixant décomposition de domaines et accélération GPU (cf. Figure 44)

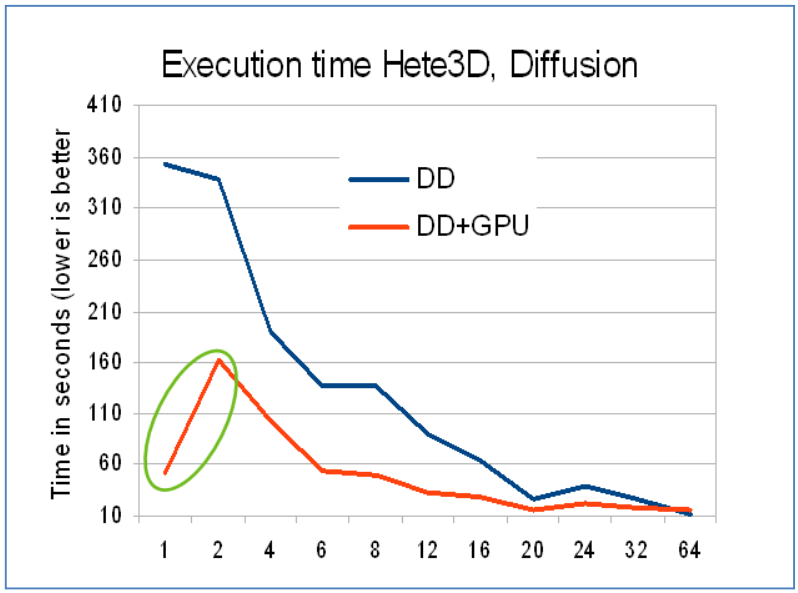


Figure 44 : Performance de la méthode de décomposition de domaines et accélération GPU sur le cas Hete3D.

Plusieurs remarques sont à faire sur cette courbe :

- Le gain de la méthode DD+GPU est réel par rapport à la méthode DD seule ;
- La scalabilité de la méthode DD+GPU est moindre qu'en méthode DD seule ;
- Le surcoût du passage de 1 GPU à 2 GPU est très important (> à un facteur 2) du fait des échanges de données entre GPU et CPU, plus la communication entre les CPU via le réseau MPI et de nouveau le transfert de la donnée entre le CPU et le GPU.

Nous avons donc différencié sur la courbe de la Figure 45 , la part des temps de transfert et des temps de calcul. Et le résultat est parlant : le surcoût dût au transfert des données est très pénalisant.

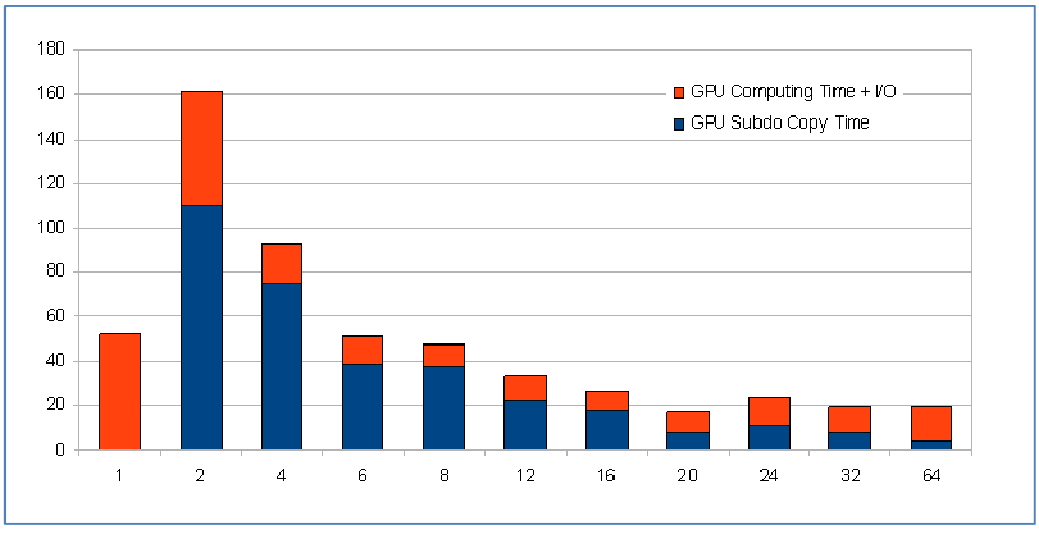


Figure 45 : Part respective du temps de calcul et du temps de transfert.

4 CONTRIBUTIONS A LA PROGRAMMATION EFFICACE DES NOUVELLES ARCHITECTURES DE CALCUL ULTRA-SCALE – APPLICATION AUX METHODES DE KRYLOV POUR LA RESOLUTION DE PROBLEMES A VALEURS PROPRES

Résumé :
L'évolution très rapide ces dernières années des architectures de calcul et les « révolutions » à venir pour atteindre l'exascale vont avoir des impacts majeurs sur les codes de simulations numériques et les algorithmes.
Nous avons donc poursuivi nos travaux dans l'étude plus approfondie de la programmation efficace de ces nouvelles architectures. Ces recherches se sont appliquées aux méthodes de Krylov pour la résolution de systèmes linéaires et de problèmes à valeurs propres qui sont très représentatifs des applications qui nous intéressent.
Nous avons ainsi abordé plusieurs thématiques majeures inhérentes à ces nouvelles architectures comme la précision arithmétique des accélérateurs de calcul [89, 90], l'étude de méthodes parallèles adaptées aux machines massivement multi-cœurs homogènes et hétérogènes, les co-méthodes offrant des propriétés intéressantes pour l'ultra-scale [97, 98, 103], les techniques d'auto-tuning et de smart-tuning appliquées aux méthodes d'Arnoldi [104, 110, 111, 113, 114,116].

Summary:
The rapid development of computing architectures in recent years and the future "revolutions" to overcome exascale will have a major impact on the numerical simulation codes and algorithms.
We continue our work in the further study of efficient programming of these new architectures. This research was applied to Krylov methods for solving linear systems and eigenproblems which are widely used for reactor physics applications.
We have discussed several inherent major themes within these new architectures such as accelerators arithmetic precision [89, 90], the study of suitable methods for massively parallel homogeneous and heterogeneous multi-core architectures, co-methods that offer interesting properties for ultra-scale [97, 98, 103], self-tuning and smart-tuning techniques applied to Arnoldi methods [104, 110, 111, 113, 114,116].

4.1 LES ENJEUX ET LES DEFIS

Comme nous l'avons abordé dans l'introduction la mutation rapide des nouvelles architectures implique une augmentation massive du parallélisme à tous les niveaux : multiprocessus au niveau d'un cœur de calcul, processeurs multi-cœurs, nœuds de calculs multiprocesseurs et cluster de ces mêmes nœuds. Pour fixer les idées, il est intéressant de regarder les degrés de concurrence actuels et futurs des supercalculateurs. La table présentée dans le Tableau 5 est issue d'un rapport de l'IESP (International Exascale Software Initiative) [72].

Systems	2009	2011	2015	2018
System Peak Flops/s	2 Peta	20 Peta	100-200 Peta	1 Exa
System Memory	0.3 PB	1 PB	5 PB	10 PB
Node Performance	125 GF	200 GF	400 GF	1-10 TF
Node Memory BW	25 GB/s	40 GB/s	100 GB/s	200-400 GB/s
Node Concurrency	12	32	O(100)	O(1000)
Interconnect BW	1.5 GB/s	10 GB/s	25 GB/s	50 GB/s
System Size (Nodes)	18,700	100,000	500,000	O(Million)
Total Concurrency	225,000	3 Million	50 Million	O(Billion)
Storage	15 PB	30 PB	150 PB	300 PB
I/O	0.2 TB/s	2 TB/s	10 TB/s	20 TB/s
MTTI	Days	Days	Days	O(1Day)
Power	6 MW	~10 MW	~10 MW	~20 MW

Tableau 5. Chiffres clefs des supercalculateurs depuis 2009.

Tout d'abord si on observe le degré de parallélisme sur un nœud de calcul (« Node Concurrency ») on passe d'une dizaine en 2009, à quelques dizaines de nos jours pour certainement tendre au millier vers 2018-2020. Le nombre de nœuds suit la même tendance, 20.000, 100.000 puis 1.000.000.000. Le degré global maximal de parallélisme de ces systèmes offre des chiffres en encore plus impressionnants : on passe de la centaine de milliers en 2009 à plusieurs dizaines de millions de nos jours et on prédit de l'ordre du milliard dans les 5 à 10 ans qui viennent.

Il s'agit donc d'une évolution majeure des systèmes de calcul et qui a un impact fort sur les algorithmes et les applications. Il y a plusieurs raisons à cela :

1. Les modèles de parallélisme classiquement utilisés ne permettent d'exprimer qu'un seul niveau de parallélisme. Cela a pour conséquence de perdre en efficacité sur un nœud de calcul mais également de créer potentiellement des goulots d'étranglement au niveau des communications. Il est donc nécessaire de repenser les algorithmes et les applications en mixant les modèles de parallélisation afin d'exploiter au mieux les différents niveaux de parallélisme;
2. Un des principaux freins à l'efficacité des applications parallèles est le surcoût engendré par les communications. Ceci ne fait que s'aggraver avec l'augmentation massive du nombre de

nœuds interconnectés en parallèle. Il est donc indispensable de minimiser les communications, en particulier les communications globales [73]. Cela passe par des méthodes de recouvrement des communications par le calcul mais également par la modification des algorithmes eux-mêmes afin réduire au maximum les communications globales et synchronisation (« communication avoiding ») [74].

3. L'accroissement du nombre de composants (cœurs, processeurs, mémoires, disques, …) augmente la probabilité de pannes. Le temps moyen entre 2 pannes (MTBF ou Minimum Time Between Failure) a tendance à diminuer, malgré l'augmentation de la fiabilité des composants. Si le MTBF est de l'ordre de la journée en moyenne pour les gros systèmes actuels, on prévoit un MTBF de l'ordre de l'heure pour les prochains systèmes au-delà de 100 PFLops [76,77]. Les systèmes classiques de sauvegarde/reprise de l'ensemble de la machine, qui sont une manière préventive de traiter les pannes, ne sont plus envisageables. En effet le temps de sauvegarde dépasse déjà largement l'heure sur les machines actuelles et le volume de stockage sur disque devient ingérable. Cela signifie qu'il est désormais indispensable de prendre en compte la tolérance aux pannes au niveau de l'application elle-même, mais aussi de l'algorithme lui-même.

A cela vient s'ajouter une autre évolution du point de vue des architectures de calcul : l'utilisation des accélérateurs de calcul. Le pas a été franchi en 2008 avec la machine IBM Roadrunner qui a été le premier calculateur à franchir la barrière du PFlops/s avec une architecture hétérogène à base de processeurs x86 standards AMD/Opteron et des accélérateurs de calcul IBM/Cell. Depuis on retrouve très régulièrement dans les machines les plus puissantes des machines à base d'accélérateurs de calcul, majoritairement des GPU (Graphic Processor Unit) et depuis l'année dernière les accélérateurs d'Intel Xeon Phi. Ceci est illustré sur les Tableau 6, Tableau 7 et Tableau 8 où nous faisons figurer les 10 machines les plus puissantes du monde d'après le classement du TOP500 [78] en novembre 2011, juin 2011 et novembre 2012.

Computer	Processor Technology	Proc. Speed (MHz)	Accelerator	Cores per Socket
K computer	Sparc	2000	None	8
NUDT YH MPP	Intel Nehalem	2930	NVIDIA 2050	6
Cray XT5-HE	AMD x86_64	2600	None	6
Dawning TC3600	Intel Nehalem	2660	NVIDIA 2050	6
HP ProLiant SL390s	Intel Nehalem	2930	NVIDIA 2050	6
Cray XE6	AMD x86_64	2400	None	8
SGI Altix ICE	Intel Core	3000	None	4
Cray XE6	AMD x86_64	2100	None	12
Bull bullx S6010/S6030	Intel Nehalem	2260	None	8
BladeCenter QS22/LS21	Power	3200	IBM PowerXCell 8i	9

Tableau 6. Classement du TOP 500 de novembre 2011.

Computer	Processor Technology	Proc. Speed (MHz)	Accelerator/ Co-Processor	Cores per Socket
BlueGene/Q	PowerPC	1600	None	16
K computer	Sparc	2000	None	8
BlueGene/Q	PowerPC	1600	None	16
iDataPlex	Intel SandyBridge	2700	None	8
NUDT YH MPP	Intel Nehalem	2930	NVIDIA 2050	6
Cray XK6	AMD x86_64	2200	NVIDIA 2090	16
BlueGene/Q	PowerPC	1600	None	16
BlueGene/Q	PowerPC	1600	None	16
Bullx B510	Intel SandyBridge	2700	None	8
Dawning TC3600	Intel Nehalem	2660	NVIDIA 2050	6

Tableau 7. Classement du TOP 500 de juin 2012.

Computer	Processor Technology	Proc. Speed (MHz)	Accelerator/ Co-Processor	Cores per Socket
Cray XK7	AMD x86_64	2200	NVIDIA K20x	16
BlueGene/Q	PowerPC	1600	None	16
K computer,	Sparc	2000	None	8
BlueGene/Q,	PowerPC	1600	None	16
BlueGene/Q	PowerPC	1600	None	16
iDataPlex	Intel SandyBridge	2700	None	8
PowerEdge C8220	Intel SandyBridge	2700	Intel Xeon Phi	8
NUDT YH MPP	Intel Nehalem	2930	NVIDIA 2050	6
BlueGene/Q	PowerPC	1600	None	16
Power 775	Power	3836	None	8

Tableau 8. Classement du TOP 500 de novembre 2012.

Il est bien évident que ces architectures hétérogènes entrainent une complexité supplémentaire à prendre en compte. Au-delà même de la simple différence d'architecture de l'unité de calcul (pour le processeur Cell ou les GPUs) qui implique des langages de programmation différents, les hiérarchies mémoires sont différentes et les modèles de parallélisme également.
Dans la suite nous aborderons les recherches qui ont été menées dans le cadre de deux doctorats, celle de Jérôme DUBOIS [71], soutenue en 2011 et celle de France BOILLOD-CERNEUX qui est en cours et qui contribuent à la recherche de solutions aux problématiques énoncées ci-dessus dans le cadre de méthodes itératives de recherche de valeurs propres.

4.2 METHODES ITERATIVES POUR LA RESOLUTION DE PROBLEMES A VALEURS PROPRES

4.2.1 Problème à valeurs propres

Pour une matrice carrée A d'ordre n, l'équation à résoudre pour obtenir les valeurs propres et vecteurs propres associés est la suivante :

$$Av_i = \lambda_i v_i \, , \lambda_i \in \mathbb{C} \; et \; v_i \in \; \mathbb{C}^n, \; i = 1 \ldots n \qquad (1)$$

Dans l'équation (1), (λ_i, v_i) représente un couple d'une valeur propre λ_i et un de ses vecteurs propres associés v_i. En effet pour une valeur propre donnée, plusieurs vecteurs propres différents peuvent exister.
De manière générale, pour résoudre (1), on calcule d'abord les valeurs propres λ_i, puis l'équation suivante est résolue, où *Id* représente la matrice identité de C^n :

$$Det(Av_i - \lambda_i Id) = 0, i = 1 \ldots n \qquad (2)$$

Ces étapes montrent la manière formelle de résoudre un problème à valeur propre, qui n'est pas forcément la plus adéquate numériquement. En effet, pour chaque vecteur propre, un calcul de

déterminant est nécessaire, opération très coûteuse en nombre d'opérations, et présentant des instabilités numériques. Différentes méthodes numériques existent pour résoudre ce problème, présentées dans la section suivante.

4.2.2 Méthodes itératives d'Arnoldi

Pour résoudre l'équation (1), deux grandes classes de méthodes existent :

1. Méthodes directes
2. Méthodes itératives

Plusieurs méthodes itératives de calcul de valeurs propres existent. Nous avons déjà évoqué la méthode de la puissance dans un paragraphe précédent. Nous nous intéressons ici à deux autres méthodes : Lanczos et Arnoldi. La méthode de la Puissance calcule la valeur propre dominante d'une matrice, là où Lanczos et Arnoldi calcule un sous-ensemble de valeurs propres de la matrice. D'autres méthodes existent, telles que Jacobi-Davidson ou la méthode de Ricatti.

Le but de la méthode de Lanczos [79,80,81] et d'Arnoldi [82] est de calculer un sous-ensemble des valeurs propres / vecteurs propres de la matrice problème. Lanczos ne s'applique qu'aux matrices symétriques ou hermitiennes, et Arnoldi reste plus général et résout les problèmes non symétriques ou non-hermitiens.

Ces deux méthodes ont pour base la méthode de la puissance, et l'améliorent en utilisant les informations générées à chaque étape. En effet, la méthode de la puissance calcule plusieurs vecteurs jusqu'à convergence à une étape p. On peut donc voir cette méthode comme la création d'une matrice $[Av_0, A^2v_0, \ldots, A^pv_0]$ dont A^pv_0 est le vecteur propre dominant. L'information utilisée pour une étape i est celle de l'étape précédente i-1, et pas celles des étapes 1 à i-2. Lanczos et Arnoldi propose d'effectuer une orthogonalisation des vecteurs des étapes 1 à p, pour extraire le plus d'information possible des étapes précédentes pour l'étape courante.

Ces deux méthodes calculent les matrices H et V telles que l'on a la relation :

$$H = V^* A V \quad (3)$$

H est une matrice de taille m+1 lignes par m colonnes, et *V* possède m+1 colonnes de taille n, le nombre de lignes de *A*. Cette relation correspond au calcul de H qui est la projection unitaire de la matrice A dans l'espace de Krylov de base.

Si l'on utilise Lanczos, alors *H* est tridiagonale, et de ce fait souvent appelée *T*. Dans le cas d'Arnoldi, la matrice *H* est de forme Hessenberg supérieure, c'est-à-dire triangulaire supérieure avec des éléments sous la diagonale. Dans les deux cas, *V(N,m)* doit être orthonormale, ce qui implique que ses vecteurs sont normés, et orthogonaux. Il faut prendre en considération l'arithmétique finie des ordinateurs, qui a une influence sur l'orthogonalité de la base *V*. En arithmétique exacte, *V* est forcément orthogonale suivant les algorithmes d'orthogonalisation que l'on utilise. En précision finie, l'algorithme utilisé tout comme le choix de l'orthogonalisation ont un impact important sur la précision des valeurs propres et vecteurs propres calculés [83].

4.2.2.1 Matrices symétriques

Pour calculer *T* et *V* par l'algorithme de Lanczos, on effectue les opérations suivantes :

```
entrée:
   -  matrice symétrique A de taille nxn
   -  vecteur  v1 normalisé
   -  nombre de valeurs propres désiré
sortie :
   -  αj et βj qui constituent la matrice T
v0 = 0
β1 = 0
pour j de 1 à m
faire
     1. wj = Avj
     2. αj = (wj , vj)
     3. wj = wj – βj.vj-1 – αj.vj
     4. βj+1 = ||wj||
     5. vj+1 = wj / βj+1
Fait
```

Figure 46. Algorithme de Lanczos.

Les coefficients α_j et β_j permettent de constituer la matrice T_m (où T_m est la matrice T sans la dernière ligne) suivant le motif de la figure 47.

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \mathbf{0} & \\ & \beta_3 & \ddots & \ddots & & \\ & & \ddots & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ \mathbf{0} & & & & \beta_m & \alpha_m \end{pmatrix}$$

Figure 47. Constitution de la matrice T après exécution de l'algorithme de Lanczos.

On peut ensuite résoudre le problème à valeurs propres d'ordre m de la matrice *T* pour calculer les couples de valeurs propres, vecteurs propres (λ_i, z_i). Les valeurs propres calculées correspondent à des approximations des valeurs propres de la matrice A, et les vecteurs propres z_i de taille m<n nécessitent une transformation avant de pouvoir être considérée comme des approximations des vecteurs propres de A. Pour ce faire, on calcule l'approximation des vecteurs propres $y_i = Vz_i$, où V = [v1, … , vm], et ensuite le résidu peut être calculé. Si la précision atteinte est suffisante, on arrête l'algorithme, sinon plusieurs solutions sont possibles. Premièrement, on peut ré-exécuter une itération de Lanczos, avec un sous espace plus grand, ou encore redémarrer un Lanczos de même taille avec un vecteur initial v1 constitué d'une combinaison linéaire des vecteurs propres de l'itération précédente [84]. On appelle ce redémarrage un redémarrage explicite de la méthode. Une dernière solution est le redémarrage implicite proposé dans [81].

4.2.2.2 Matrices non symétriques

Dans le cas des matrices non symétriques, on peut appliquer la méthode d'Arnoldi. Les mêmes principes que pour la méthode de Lanczos sont suivis, avec :

1. Une étape de réduction permettant de calculer les matrices H et V de l'équation 3. Cette étape est appelée réduction d'Arnoldi ou orthogonalisation.
2. Le problème à valeurs propres est résolu dans H.

3. Les vecteurs propres de H sont projetés grâce à la matrice V pour obtenir les vecteurs de Ritz de la matrice A, une approximation de quelques-uns de ses vecteurs propres.

4.2.2.2.1 Orthogonalisation

Pour l'orthogonalisation, plusieurs schémas peuvent être sélectionnés avec chacun ses avantages et ses inconvénients. Les schémas les plus connus sont l'orthogonalisation de Gram-Schmidt, les transformations de Householder ou les rotations de Givens. Dans le cadre de la thèse de doctorat de J. DUBOIS [71], nous nous sommes concentrés sur la version Gram-Schmidt, même si les autres versions restent également valables.
Parmi les orthogonalisations de Gram-Schmidt [85], on distingue 3 grandes variantes : l'orthogonalisation classique (CGS), l'orthogonalisation modifiée (MGS) et Gram-Schmidt avec réorthogonalisation (CGSr).

D'un point de vue performances, l'un des intérêts des orthogonalisations CGS* par rapport à MGS est leur parallélisme intrinsèque. Les opérations sont effectuées sur des matrices denses et des vecteurs, ce qui ajoute une bonne régularité des données. Dans le cas de la méthode MGS, les opérations portant sur V et H sont effectuées entre vecteurs, et dépendantes de l'étape précédente. On a donc une moins bonne parallélisation intrinsèque de cette orthogonalisation.
Concernant la précision numérique, ces trois orthogonalisations n'offrent pas la même précision sur l'orthogonalité de la base de Krylov V construite, point critique pour la précision des valeurs/vecteurs propres calculés. Généralement la méthode la plus précise est CGSr, suivie de MGS puis CGS. Nous arborerons ces différences ainsi que l'influence de la précision du matériel sur cette orthogonalité dans une section suivante.

4.2.2.2.2 Résolution du problème à valeurs propres et projection

Une fois les matrices H et V constituées, les valeurs propres et vecteurs propres de H sont calculés en utilisant une méthode numérique au choix. L'idéal est d'utiliser une méthode prenant en compte la forme Hessenberg de la matrice H. La factorisation de Schur proposée dans la méthode *hseqr* de LAPACK est une bonne option. Elle calcule la forme de Schur de la matrice H, telle que $H=STS^T$. La matrice S contient les vecteurs de Schur de H, et la matrice T tridiagonale, contient les valeurs propres de H sur sa diagonale. Un appel à la méthode de la puissance inverse permet par la suite de retrouver les vecteurs propres de H à partir de ses valeurs propres. Ensuite, pour obtenir les vecteurs propres associés aux valeurs propres de A, on utilise la matrice V pour projeter les vecteurs propres de la même manière que pour la méthode de Lanczos : $y_i = Vz_i$, avec z_i vecteur propre de H.
On peut alors évaluer le résidu ε_i lié à chaque couple de valeur propre λ_i et vecteur propre y_i suivant la formule suivante :

$$\varepsilon_i = \|(A - \lambda_i Id)y_i\|$$

Ce résidu peut également être calculé suivant la formule analytique proposée par Saad dans [86,87], qui ne fait intervenir que le produit de deux scalaires et borne le résidu. La formule est alors :

$$\varepsilon_i = |H(m+1,m)y_i(m)|$$

Si ce résidu est supérieur au seuil fixé, alors la méthode doit être redémarrée.

4.2.2.2.3 Arnoldi itéré

La méthode d'Arnoldi peut être itérée implicitement et explicitement. La manière implicite a été proposée par Sorensen [88]. Elle considère l'utilisation efficace de l'information liée aux valeurs propres intéressantes pour le problème, et la non-prise en compte de l'information des valeurs propres non désirées. La convergence est meilleure dans le cas implicite et demande en général moins d'itérations qu'un Arnoldi explicitement redémarré. Cependant le redémarrage explicite offre des possibilités intéressantes en termes d'hybridation comme nous le verrons au chapitre 5. La version explicite a été proposée par Saad [87], et offre l'avantage d'un schéma de redémarrage simplifié. Une combinaison linéaire des vecteurs de Ritz de l'itération précédente est utilisée pour constituer le vecteur initial de l'itération suivante. Ainsi, la convergence est forcée vers le sous-espace de Krylov contenant les valeurs propres désirées. L'algorithme de la méthode d'Arnoldi explicitement redémarrée (ERAM) peut s'écrire :

Méthode d'Arnoldi explicitement redémarrée (ERAM)

Entrée :
- vecteur v0 initial
- matrice A_{nxn}
- critères de convergence : seuil ε, nombre d'itérations maximum iter_max

Sortie :
- couples valeurs/vecteurs propres (λ_i, w_i)

```
iter = 0
tant que iter < iter_max et residu > ε
faire
   1. effectuer une réduction d'Arnoldi : A = V*HV
   2. résoudre le problème à valeurs propres de H : H.y_i = λ_i.y_i
   3. projeter les vecteurs propres y_i : wi = V.y_i
   4. mettre à jour le résidu
fait
```

Figure 48. L'algorithme de la méthode d'Arnoldi explicitement redémarrée (ERAM).

4.3 PRECISION ARITHMETIQUE SUR ACCELERATEURS DE CALCUL

Les premiers accélérateurs utilisables pour le calcul scientifique sont apparus en 2007, 2008. Cependant à cette époque, ces unités de calcul ne sont pas prévues pour faire des calculs en double précision et ne suivent pas la norme IEEE. Ceci peut avoir un impact non négligeable sur la précision des calculs numériques, notamment dans le cas de méthodes itératives. Dans [83,90], nous avons étudié justement l'impact de cette précision arithmétique sur la perte d'orthogonalité des bases de Krylov.

Nous avons étudié l'influence de la représentation arithmétique flottante de ces accélérateurs sur l'orthogonalisation de Gram-Schmidt en utilisant des calculs simple et double précision.

Plusieurs variantes du procédé d'orthogonalisation existent [98]: la méthode Householder, les rotations de Givens et le processus de Gram-Schmidt sont les plus courantes. Le processus de Gram-Schmidt est décliné en différentes versions : classique (CGS), modifié (MGS) et classique avec ré-orthogonalisation (CGSr). La version classique est la plus simple et facilement parallélisable comme on le voit dans [98]: il est plus rapide, car il peut utiliser des noyaux de calcul de type BLAS 2 (opérations matrice-vecteur) au lieu de BLAS 1 (opérations vecteur-vecteur) dans l'algorithme modifié.

Mais le principal inconvénient de CGS est la perte d'orthogonalité de la base calculée en raison d'une erreur d'arrondi [99]. L'algorithme de MGS fourni une version plus précise du processus de Gram-Schmidt. Mathématiquement les deux sont identiques, mais le degré de parallélisme de l'algorithme MGS est moindre. Le processus CGSr, quant à lui, fournit une version encore plus précise, par ré-orthogonalisation de chaque vecteur calculé. En pratique, il n'est pas nécessaire de le faire à chaque itération, et donc le coût de cet algorithme est proche de CGS, tout en étant plus précis. Ces différents algorithmes sont résumés sur la Figure 49.

CGS	MGS	CGSr
1. for i = 1 : m 2. $v_{i+1} = Av_i$ 3. $H_{1:i,i} = (v_{i+1}, v_{1:i})$ 4. $v_{i+1} = v_{i+1} - H_{1:i,i}.v_{1:i}$ 5. $H_{i,i+1} = \|v_{i+1}\|_2$ 6. $v_{i+1} = \frac{v_{i+1}}{H_{i,i+1}}$ 7. end	1. for i = 1 : m 2. $v_{i+1} = Av_i$ 3. for j = 1 : i 4. $H_{j,i} = (v_{i+1}, v_j)$ 5. $v_{i+1} = v_{i+1} - H_{j,i}.v_j$ 6. end 7. $H_{i,i+1} = \|v_{i+1}\|_2$ 8. $v_{i+1} = \frac{v_{i+1}}{H_{i,i+1}}$ 9. end	1. for i = 1 : m 2. $v_{i+1} = Av_i$ 3. $H_{1:i,i} = (v_{i+1}, v_{1:i})$ 4. $v_{i+1} = v_{i+1} - H_{1:i,i}.v_{1:i}$ 5. $C_{1:i,i} = (v_{i+1}, v_{1:i})$ 6. $v_{i+1} = v_{i+1} - C_{1:i,i}.v_{1:i}$ 7. $H_{1:i,i}$ += $C_{1:i,i}$ 8. $H_{i,i+1} = \|v_{i+1}\|_2$ 9. $v_{i+1} = \frac{v_{i+1}}{H_{i,i+1}}$ 10. end

Figure 49 : Les différentes versions du processus d'orthogonolisation de Gram-Schmidt.

Pour étudier l'influence de la précision arithmétique, nous avons vérifié la conservation de l'orthogonalité des bases calculées pour des matrices denses, puis pour des matrices creuses.

La précision du processus de Gram-Schmidt pour une taille de sous-espace est calculée en utilisant la valeur absolue maximale des produits scalaires des vecteurs de la base :

$$max\left|(v_i, v_j)\right| \; avec \; i \neq j \; et \; v_i, v_j \in V$$

Cela correspond donc au maximum de la perte d'orthogonalité entre deux vecteurs de la base. Nous ne détaillerons pas ici l'ensemble des résultats, le lecteur pourra se référer à la publication [90]. A titre d'illustration, nous avons repris sur la Figure 50 les résultats obtenus en simple précision avec le processus CGS dans le cas d'une matrice dense et sur la Figure 51 les résultats en double précision avec la même matrice.

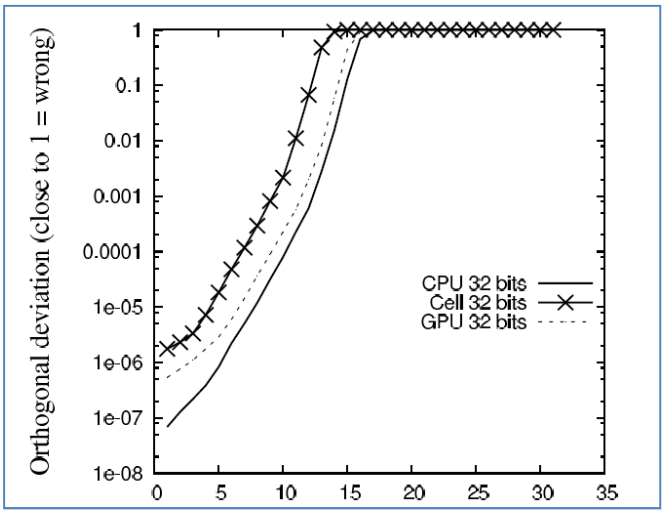


Figure 50 : Perte d'orthogonalité maximale suite au processus CGS en fonction de la taille de la base sur différentes unités de calcul (CPU, Cell et GPU). Calculs en simple précision sur une matrice dense de Hilbert de 10.240 éléments.

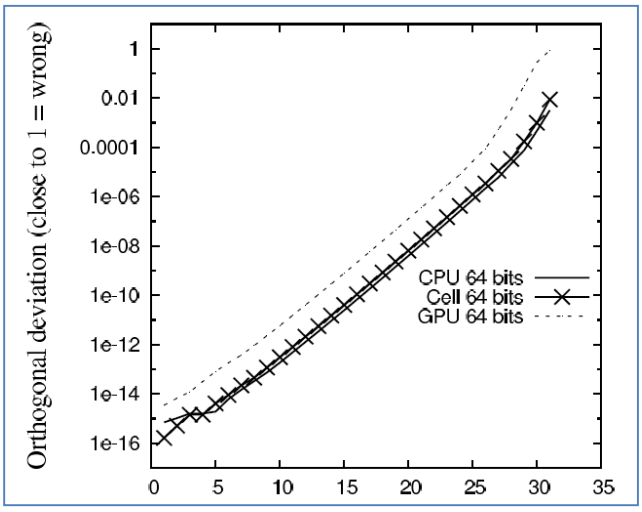


Figure 51 : Perte d'orthogonalité maximale suite au processus CGS en fonction de la taille de la base sur différentes unités de calcul (CPU, Cell et GPU). Calculs en double précision sur une matrice dense de Hilbert de 10.240 éléments.

En résumé, en ce qui concerne l'orthogonalité, nous avons observé pour les matrices denses une perte au pire de 1 chiffre pour les GPU et 2 chiffres pour le processeur Cell par rapport à un CPU totalement compatible IEEE. Pour les matrices creuses, le résultat entre le CPU et le GPU est très proche. En fin de compte, nous avons obtenu des accélérations entre un facteur 10 et 20 pour des matrices denses en utilisant les GPU, et un facteur 7 pour des matrices denses avec le processeur Cell.

Le processeur Cell s'est avéré être un bon accélérateur pour la double précision IEEE, mais insuffisante pour des applications nécessitant une précision simple conforme à la norme IEEE.

Les GPUs offrent de meilleures performances dues à une bande passante mémoire plus importante mais également une bonne conformité à la norme IEEE qui donne des résultats proches de ceux de CPU pour les deux précisions.

Désormais les GPUs respectent la norme IEEE en simple et double précision, cependant avec l'augmentation du nombre de cœurs de calcul pour résoudre des problèmes de taille de plus en plus importante, il faut toujours veiller à ce que les erreurs d'arrondi ne viennent pas dégrader les processus de convergence dans les méthodes itératives ou encore pire, converger sur un résultat faux. Des calculs à arithmétique étendue (128 ou 256 bits) sont à envisager pour le futur et des premiers résultats de recherche montrent que, même si une opération élémentaire en quadruple précision coûte plus chère qu'une opération en double précision, l'augmentation de la précision des résultats peut améliorer la convergence et au final s'avérer moins couteux en temps de calcul.

4.4 ERAM EN PARALLELE SUR MULTI-COEURS ET ACCELERATEURS

4.4.1 Principes de parallélisation

ERAM a été implémenté en utilisant le framework *Krylov bAsed Solvers for Hybrid architecture* (KASH), dont le but est d'abstraire le matériel de calcul utilisé à l'aide de concepts empruntés à la programmation objet. Grâce à ces abstractions, nous décrivons un algorithme utilisant des matrices et des vecteurs à l'aide de KASH, tel que la méthode d'Arnoldi redémarrée, et nous pouvons exécuter cette dernière sur processeurs standards x86 ou sur GPUs. La partie calcul pur est implémentée en appelant les BLAS de la machine cible, nommément la Math Kernel Library (MKL) [94] d'Intel pour les processeurs multi-cœurs et CUBLAS [95] de Nvidia pour les GPUs Nvidia. KASH supporte l'exécution multithreadée au travers des BLAS multithreadées auxquelles il fait appel. Ainsi, plusieurs threads peuvent être utilisés pour les opérations vectorielles ou matricielles de KASH. Lorsque nous exécutons ERAM sur GPUs, toutes les opérations sont effectuées sur le GPU, excepté la résolution du problème à valeurs propres dans le sous-espace. A cette étape, la matrice H est de taille réduite[8], et il est plus performant d'utiliser le processeur multi-cœur classique pour le calcul car cette dernière tient dans les caches du processeur.

Pour les communications inter-nœuds, l'interface MPI est utilisée. La matrice est découpée en sous-domaines suivant un schéma en échiquier. Pour une taille de matrice donnée, la forme de l'échiquier choisi est toujours la plus carrée possible. Ainsi pour 9 processus MPI, nous utiliserons un échiquier 3x3. Lorsque cela n'est pas possible, nous privilégions un nombre de lignes supérieur au nombre de colonnes car dans le cadre d'un produit matrice vecteur, opération dominante de la méthode, cela limite la quantité de communications [96].

4.4.2 Résultats de performances

Plusieurs expérimentations ont été menées sur différentes machines pour mesurer les performances de la méthode ERAM en parallèle sur machines multi-cœurs homogènes et hétérogènes [97, 98, 99]. Nous ne ferons que résumer les principaux résultats obtenus et leur analyse nous a conduits à explorer d'autres voies pour améliorer les performances : les co-méthodes et les techniques d'auto-tuning et de smart-tuning.

4.4.2.1 Performances sur machines homogènes :

Nous résumons sur le Tableau 9 les principales machines utilisées pour ces tests ainsi que leurs caractéristiques.

	Nœuds	Modèle	Cœurs	Procs	Ram	BP	Puiss.	Ram/cœur	Réseau
Titane	1068	Intel X5570	8	2	24	~50	~94	3 Go	Inf. DDR
Hopper	6384	AMD MC*	24	2	32	~80	~202	1,3 Go	Cray Gemini
Curie	5400	Intel X7560	32	4	128	~100	~375	4 Go	Inf. QDR

Tableau 9. Caractéristiques principales des machines Titane, Hopper et Curie. Le nombre de nœuds de Curie est une projection du total attendu. Le nombre de cœurs, de processeurs, la mémoire Ram, la bande passante et la puissance sont donnés par nœud de calcul. La Ram est en Go, la bande passante en Go/s. Le modèle de processeur AMD est Magny-Cours.

[8] L'ordre de la matrice H est égal à la taille du sous-espace de Krylov.

Sur la machine Titane, nous instancions un solveur ERAM utilisant l'orthogonalisation CGSr, avec un sous-espace de taille 32. La matrice DingDong est d'ordre 12.288, et la précision demandée est 10^{-13}. Les tests sont effectués en double précision, et la Figure 52 illustre la scalabilité d'ERAM.

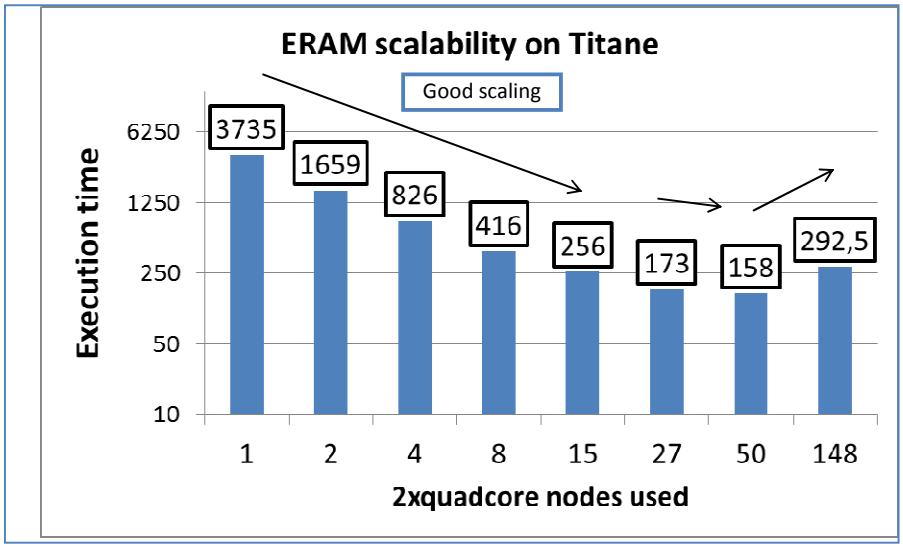


Figure 52. Scalabilité d'ERAM sur la machine Titane.

Nous constatons sur cette figure la très bonne efficacité de la méthode jusque 15 nœuds, avec ensuite une perte de performance régulière. Avec un grand nombre de nœuds, nous perdons en efficacité. Les communications prennent le pas sur la réduction du temps de calcul, et le gain en calcul est contrebalancé par l'augmentation des transferts réseaux. Pour illustrer ce fait, nous avons pu exécuter les mêmes calculs sur la machine Hopper du National Energy Research Scientific Computing Center (NERSC). Chaque nœud de cette machine est constitué de 2 processeurs AMD 12-cœurs Magny-Cours équipés de 32 Go de mémoire vive. Le réseau utilisé est Infiniband QDR, qui offre le double du débit du réseau de Titane. On s'attend donc à une meilleure scalabilité des calculs sur Hopper. La Figure 53 nous montre que c'est le cas, où nous exécutons le même cas que sur Titane, avec différentes orthogonalisations.

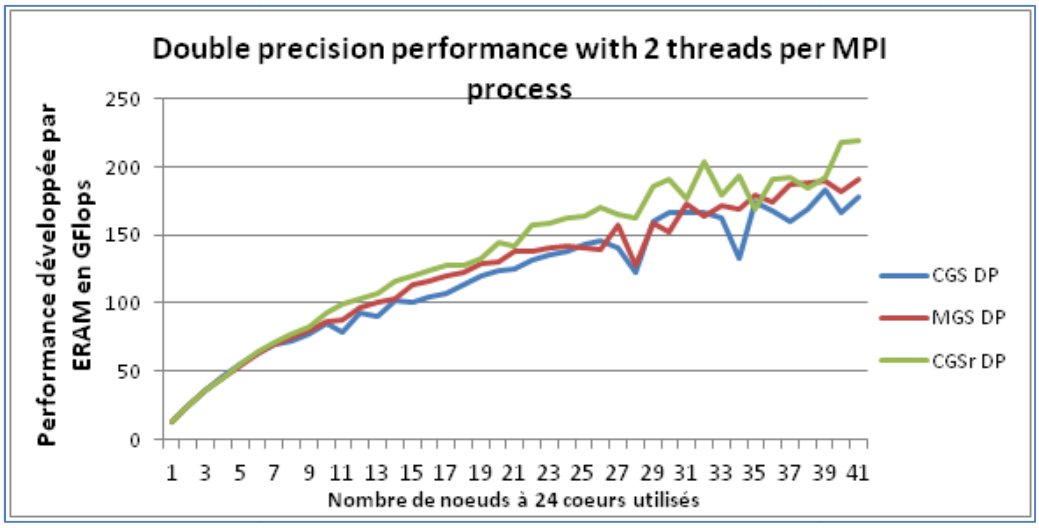


Figure 53. Performance de l'exécution d'ERAM sur la machine Hopper, de 24 à 984 cœurs.

Ces mêmes études de performances ont été menées sur la machine Curie afin de tester la scalabilité de la méthode à plus grande échelle. La taille du problème de base est une matrice pleine carrée d'ordre 22.680 soit $5{,}11.10^8$ éléments au total. Les calculs ont été effectués en double précision, pour un volume de données de 3,8 Go au total. Nous avons demandé le calcul de 135 valeurs propres avec un sous-espace de Krylov de taille 225, et une précision demandée de 10^{-8}. Les résultats obtenus sont présentés sur la Figure 54 .

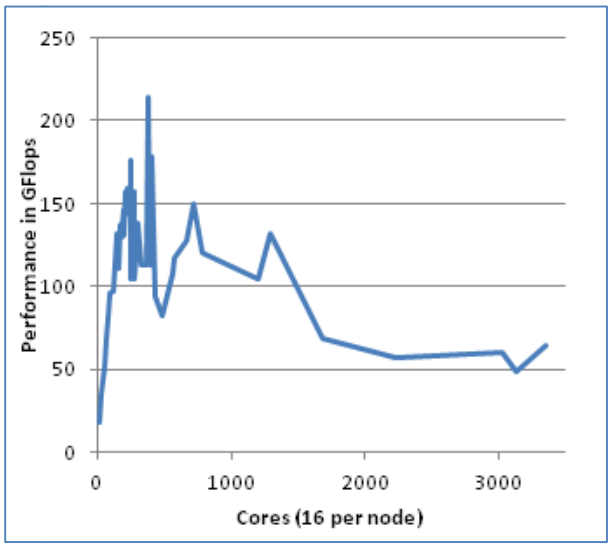


Figure 54. Evolution de la performance en strong scaling sur la partition fine de CURIE de 32 à 3360 cœurs.

Ces différentes expérimentations montrent une limite dans la scalabilité de la méthode ERAM. Nous analyserons dans une section suivante quelques unes des causes.

4.4.2.2 Performances sur machines hétérogènes :

Le même type d'études de performances a été conduit sur la partie hybride GPU de Curie, dans le cadre d'un Grand Challenge GENCI-TGCC. Nous avons réalisé des études de performance en *weak-scaling*[9] et en *strong-scaling*[10].

Pour l'étude en *weak-scaling*, la taille du problème de base est une matrice pleine carrée d'ordre 26 000, soit $6{,}76.10^8$ éléments. Les calculs ont été effectués en double précision, pour un volume de données de 5 Go par GPU. Nous nous intéressons au calcul des 8 plus grandes valeurs propres avec un sous-espace de Krylov de taille 16, et une précision exigée de 10^{-8}. Les résultats sont présentés sur la Figure 55.

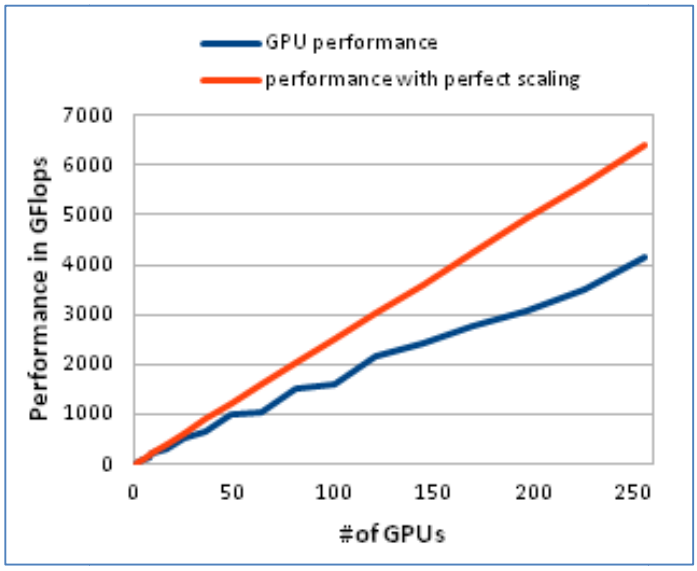


Figure 55. Performance en weak-scaling de 1 à 280 GPUs.

Pour expliquer ces performances, nous avons proposé un modèle de performance sur multi-GPUs pour l'algorithme ERAM qui est le suivant :

$$T_{exec}(p,N) = T + T_{comm\_pmv}(p,N)$$

$$T_{comm\_pmv}(p,N) = 2 \times [\beta_{PCIe} + N/\tau_{PCIe}] + 2 \times log\sqrt{p} \times [\beta_{MPI} + N/\tau_{MPI}]$$

Où :

- p est le nombre d'unités de calculs (GPU dans notre cas)
- N quantité de donnée à transférer par GPU (Ordre de la matrice / $p^{1/2}$ dans notre cas)
- β_{PCIe} = latence du PCI Express de 100 à 400 nanosecondes (point à point), prenons 100 ns [100]
- β_{MPI} = latence du réseau MPI >= 1 microseconde (point à point, QDR), prenons 1 microseconde [101]
- τ_{PCIe} = taux de transfert (ou bande passante) du PCI-express = 1 Go/s) [100]
- τ_{MPI} = taux de tranfsert (ou bande passante) de l'interconnexion QDR de 6 Go/s) [101]

[9] Dans le cas du *weak-scaling*, la taille du problème à résoudre augmente de façon proportionnelle au nombre d'unités de calcul utilisées de telle sorte que la taille du problème à résoudre par unité de calcul reste constante.

[10] Dans le cas du *strong-scaling*, la taille du problème reste constante, quelque soit le nombre d'unités de calcul utilisées.

Ce modèle est comparé aux mesures sur la Figure 56. Nous avons donc un modèle des performances qui permet bien de prédire le temps d'exécution d'ERAM sur plusieurs GPUs.

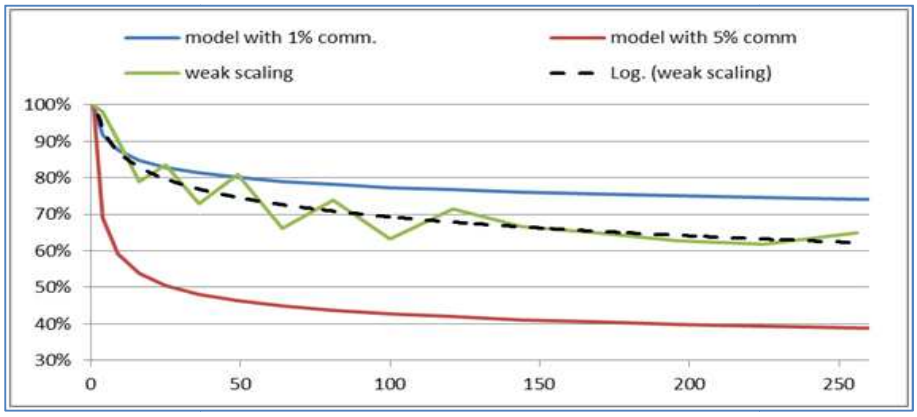


Figure 56. Confrontation de la modélisation de l'accélération GPU en weak-scaling (courbes bleue et rouge) avec les expérimentations (courbe verte) et une courbe de tendance de type logarithme basée sur les expérimentations en weak scaling.

Nous pouvons à l'aide de ce modèle analyser plus finement les performances obtenues. Si on s'intéresse à la partie uniquement des communications, nous pouvons séparer la part prise par les transferts entre le CPU et CPU-host via le lien PCI-e. Ceci est illustré dans la Figure 57.

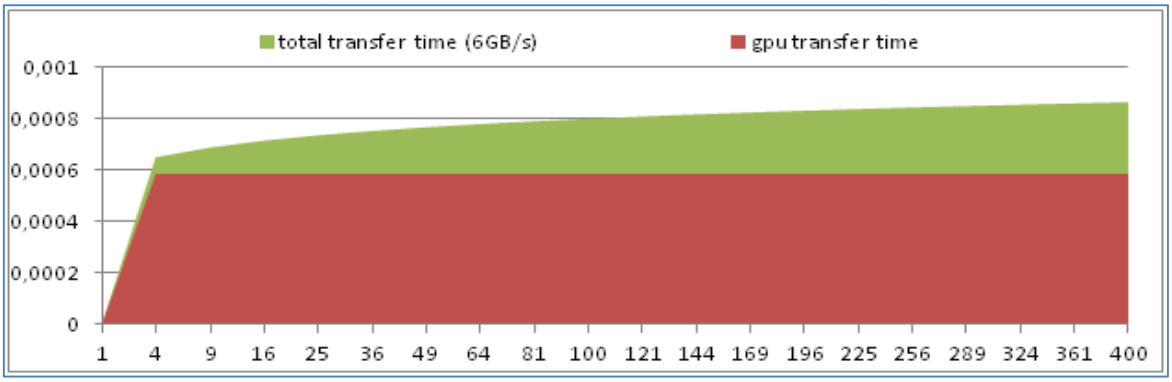


Figure 57. Modélisation de la part du temps de transfert entre GPU et CPU en fonction du nombre de GPUs.

Nous avons donc une explication aux performances obtenues : le temps de transfert entre GPU et CPU est prédominant sur le temps de communication.

Nous voyons ainsi que selon la proportion des communications dans le temps d'exécution, le speed-up d'un facteur 5x théorique entre 4 GPUs et 4 CPUs multi-cœurs peut être compris entre 4,5x avec des communications très faibles (1%), ou quasi-inexistant (1x) avec des communications très importantes (50%). Avec l'augmentation du nombre de GPUs, l'accélération décroit au point de devenir une décelération dans les cas où les communications représentent 25 et 50% (cf. Figure 58).

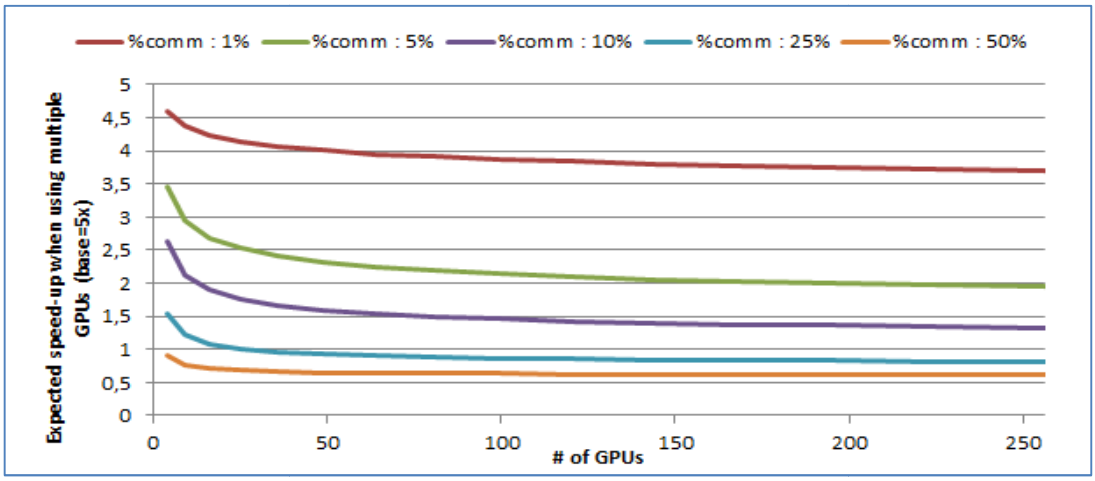


Figure 58. Modélisation de l'évolution du speed-up attendu en fonction du nombre de GPUs, en suivant le modèle de weak-scaling d'ERAM de 4 à 256 GPUs.

Pour le cas d'étude en *strong-scaling*, la taille du problème de base est une matrice pleine carrée d'ordre 22.680 soit $5{,}11.10^8$ éléments au total. Les calculs ont été effectués en double précision. Nous avons demandé le calcul de 135 valeurs propres avec un sous-espace de Krylov de taille 225, et une précision demandée de 10^{-8}. Les résultats sont présentés sur la Figure 59.

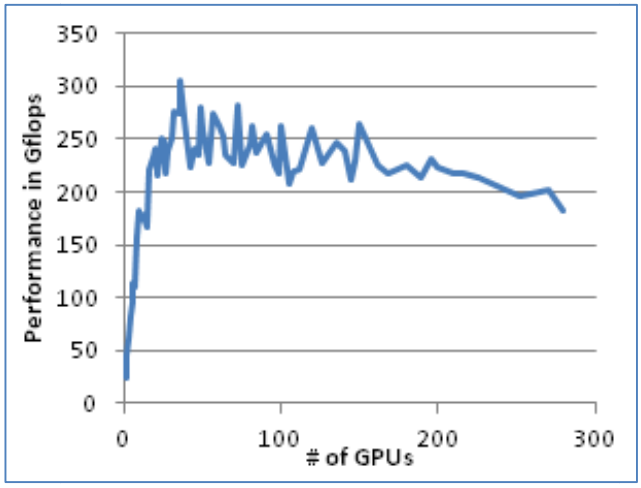


Figure 59: Performance en strong-scaling de 1 à 280 GPUs.

Dans le cadre du *weak-scaling*, nous avons pu proposer un modèle pour expliquer le comportement de la performance. Proposer un modèle similaire dans le cadre du *strong-scaling* est plus difficile pour différentes raisons. En *weak-scaling*, seul le temps de communication varie, car le temps de calcul reste fixe sur chaque unité de calcul. Dans le cadre du *strong-scaling*, les deux temps, calcul et communications, varient pour chaque nombre d'unités de calcul. De plus, le comportement de la performance du GPU dépend non seulement de la quantité de données mais également de leur distribution entre les différents GPUs comme le montre le tableau suivant.

Matrix order	22680	16037	13094	11340	8019	5670
Exec time (s)	193,07	114,21	74,87	54,17	28,28	15,54
Perf. Index	2,66	2,25	2,29	2,37	2,27	2,07
Efficacité	100%	85%	86%	89%	85%	78%

Matrix order	4009	2835	2315	2070	1750	1512
Exec time (s)	9,49	6,78	5,74	5,28	4,76	4,39
Perf. Index	1,69	1,19	0,93	0,81	0,64	0,52
Efficacité	64%	45%	35%	30%	24%	20%

Tableau 10. Evolution de la performance du solveur ERAM en fonction de la taille de matrice. L'indice de performance est calculé en divisant le nombre d'éléments correspondant à la taille de la matrice par le temps d'exécution. L'efficacité est obtenue en divisant l'indice de performance courant par l'indice de référence : la performance avec un ordre de 22680.

L'introduction de ces différentes variables rend la modélisation bien plus complexe. Cependant, quelques phénomènes propres ou non aux GPUs, viennent aider à la compréhension du comportement d'ERAM en *strong scaling* sur GPUs.

Premièrement, la quantité de données tend à décroitre sur chaque GPU, avec une perte de performance sur chaque unité de calcul. Ainsi, utiliser 225 GPUs implique une quantité de données sur chaque GPU équivalente à celle d'une matrice carrée d'ordre 1512. Pour cette taille, nous utilisons le GPU à 20% de ses capacités. Autrement dit, même en négligeant les communications, notre efficacité sera inférieure ou égale à 20% avec ce nombre de GPUs. En outre, comme nous avons pu le constater la forme carrée de la matrice locale en mémoire du GPU permet de conserver des bonnes performances. Une forme « très rectangulaire » de la matrice locale ne permet pas de tirer partie des performances du GPU et peut encore diminuer l'efficacité de calcul en dessous de 20%. Ces phénomènes influencent tous deux la performance de calcul développée par chaque GPU lors du *strong-scaling*, indépendamment des communications.

Deuxièmement, alors que le temps de calcul diminue, l'impact des communications va en augmentant. C'est un phénomène que l'on retrouve sur toute architecture parallèle lorsque l'on effectue des tests de *strong-scaling*. Le temps d'exécution se décompose en temps de calcul pur et temps de communication pur. Le temps de calcul pur tend à diminuer proportionnellement avec le nombre d'unités de calcul lorsque l'on utilise des processeurs standards. Parallèlement, la quantité d'information à échanger reste à peu près identique, mais le nombre de transferts augmente. Ces

transferts s'effectuent en deux phases : initialisation de la communication (latence) puis transfert de données (utilisation de la bande passante). Le temps de transfert diminue proportionnellement avec l'augmentation du nombre d'unité de calcul car il y a de moins en moins de données sur chaque processeur. Par contre, les initialisations ou latences réseau restent identiques, et il y en a de plus en plus lorsque l'on ajoute des unités de calcul.
Dans le cas du GPU, il y a des transferts supplémentaires liés au PCI-Express. Ces derniers induisent une bande passante comparable au réseau Infiniband QDR, mais une latence bien supérieure à celle de ce type de réseau. Il y a un facteur de l'ordre de 100 à 1000 entre ces deux technologies, au désavantage du PCI-Express. Ce rapport reste constant lors du *strong-scaling*, et induit donc une pénalité permanente lors de l'utilisation de plusieurs GPUs, pénalité non présente lorsque l'on utilise uniquement des processeurs standards.
Par rapport aux performances étudiées dans le cas du *weak-scaling* un phénomène supplémentaire entre en jeu dans la dégradation des performances en *strong-scaling* : la taille du problème local diminuant, les performances de calcul pur développées par le GPU diminuent, phénomène similaire à celui des machines de calculs vectoriels.

4.4.2.3 Limitations de la parallélisation d'ERAM

La parallélisation d'ERAM implémentée peut utiliser les deux types de mémoires usuelles en calcul parallèle : mémoire partagée au sein d'un nœud et mémoire distribuée sur plusieurs nœuds. ERAM possède une performance intrinsèquement limitée par la bande passante mémoire à cause de ses noyaux de calculs BLAS 1 et 2 et des synchronisations globales nécessaires à son fonctionnement.
Il existe plusieurs pistes pour gagner en scalabilité. On peut tenter d'augmenter la densité de calcul en réorganisant ces derniers, regroupant les opérations BLAS 1 pour faire du BLAS 2, et de même avec les BLAS 2 pour faire du BLAS 3. Cela permettrait de ne plus être limité par la bande passante mémoire, et de tirer plus partie de la puissance de calcul à disposition. De plus, il serait possible d'essayer de recouvrir les communications par des opérations de calcul. Par exemple, la densification des calculs se fait pour le calcul de la factorisation QR d'une matrice, permettant un gain de performance significatif autant sur architecture multicoeur que GPUs [107]. Dans notre cas, cela semble assez difficile à cause des fortes dépendances de données. Dans [108], l'auteur propose également d'éviter les communications pour la méthode GMRES en modifiant légèrement l'algorithme de la projection de Krylov, avec une accélération substantielle de 1.3x à 4x en mémoire partagée. Ce genre de technique serait envisageable pour ERAM. Plus proche d'Arnoldi, les auteurs utilisent dans [109] une technique de regroupement de la synchronisation de réorthogonalisation et du calcul de norme de l'orthogonalisation CGSr, permettant un gain de performance lorsqu'un grand nombre de nœuds est utilisé. Une approximation est utilisée pour décaler une communication globale et augmenter la scalabilité de l'orthogonalisation de Gram-Schmidt avec réorthogonalisation. Le gain atteint est intéressant en termes de performances, mais l'erreur introduite par les manipulations numériques n'est pas bornée.
Nous allons voir dans les sections suivantes d'autres voies d'améliorations de performances de cette méthode sur architectures homogènes et hétérogènes.

4.5 AUTO-TUNING DES METHODES D'ARNOLDI

4.5.1 Introduction

Le terme anglais de *tuning* peut être traduit par « adaptation ». Dans le domaine de l'informatique on peut l'étendre à la notion d'adaptation d'un logiciel, code, … à son environnement informatique d'exécution. Par adaptation, on peut entendre optimisation. Ainsi, lorsqu'on parle de tuning en informatique cela signifie : **adaptation d'un programme à son environnement informatique en vue d'en tirer une exécution optimisée en termes de performances**.

L'auto-tuning revient donc à fournir des procédés automatique d'optimisation en fonction du contexte informatique d'exécution : caractéristiques de la machine (vitesse du processeur, tailles mémoires et tailles de caches …) ou du middleware (compilateurs, bibliothèques numériques …). Cet auto-tuning peut se faire de manière statique, à savoir avant l'exécution, ou de manière dynamique, c'est-à-dire en cours d'exécution.

Dans le problème qui nous intéresse, l'implémentation parallèle de méthodes de Krylov pour la recherche de valeurs propres sur architectures multi-cœurs homogènes et hétérogènes, ces techniques d'auto-tuning sont certainement un moyen d'optimiser les performances, car comme on a pu le voir dans les paragraphes précédents, les paramètres influants sur les performances sont très nombreux.

Dans les sections suivantes nous présenterons quelques voies qui ont été explorées dans le cadre de la thèse de doctorat de Jérôme DUBOIS [71].

4.5.2 Autotuning du produit matrice-vecteur sur GPU

Au-delà de l'aspect arithmétique IEEE, d'autres caractéristiques matérielles évoluent. Essentiellement, le contrôleur mémoire a gagné en souplesse d'accès. Avec les premières cartes, il fallait accéder la mémoire de manière contigüe ou très régulière. Ainsi, si lors de la lecture un seul thread GPU lisait une donnée différemment des autres threads, la performance pouvait être divisée par deux. Pour chaque différence, une division peut avoir lieu, impactant très fortement la performance dans le cas de lectures très irrégulières. Sur la génération 1.3 des cartes (S1070 et C1060 par exemple), les contrôleurs mémoires ont été améliorés, permettant plus de souplesse et moins de pénalité en cas d'accès irréguliers. La génération 2.x (Tesla C2050 et plus) a encore amélioré la performance du contrôleur mémoire, et surtout ajouté une autre amélioration matérielle : deux niveaux de caches, L1 et L2.

Pour ces détails techniques, un produit matrice vecteur creux, à implémentation et bande passante équivalente, n'obtiendra pas la même performance selon le type de matériel GPU utilisé. Par exemple, nous présentons les résultats du produit matrice vecteur creux pour la matrice nlpkkt80 décrite dans le tableau 12, provenant de l'Université de Floride, sur les GPUs de la machine Titane.

Format	CSR scalar	CSR vector	COO	ELL	HYB
Perf . (GFlops)	2	6.5	3.5	17	17
Speed-up	Réf.	3.25x	1.75x	8.5x	8.5x

Tableau 11. Performances obtenues en fonction du format de stockage.

La performance varie de 2 GFlops à 17 GFlops selon le format, soit un speed-up de 8.5x. Sur un GPU de station de travail, une quadro FX 4600, qui ne supporte pas la double précision, le speed-up est de 5x au maximum entre CSR vector et CSR scalar. Dans [103,104], nous avons proposé une recherche automatique de ce format optimal, à l'exécution.

L'idée appliquée est très basique, et peut être grandement améliorée comme nous le verrons par la suite. Pour une exécution d'Arnoldi, la matrice n'est utilisée qu'en lecture, autant de fois qu'il y a d'itérations de la méthode. Il y a donc un intérêt à optimiser sa structure pour une exécution globale plus rapide. Nous proposons de le faire juste avant l'exécution du solveur, à la construction de la matrice. Cette solution cherche la meilleure solution pour chaque matrice, avec un léger surcoût. Dans [105], l'utilisation d'une métaheuristique permet, par analyse de la structure de la matrice, de trouver le format optimal avec un taux de réussite moyen de 66%.

Dans notre cas, un par un, nous mesurons la performance des différents formats de matrices à disposition dans notre bibliothèque. Ces formats incluent tous les noyaux optimaux proposés dans [106]. Nous choisissons alors le format de matrice le plus performant, et instancions cette matrice pour notre solveur. L'avantage de cette approche est d'utiliser le format de matrice optimum pour notre GPU. L'inconvénient est un léger surcoût lié à l'évaluation avant exécution du solveur. Cependant, ce type d'approche peut être amélioré en tirant partie des transferts mémoires asynchrones des GPUs.

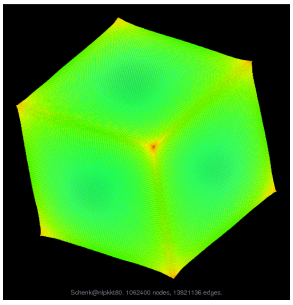
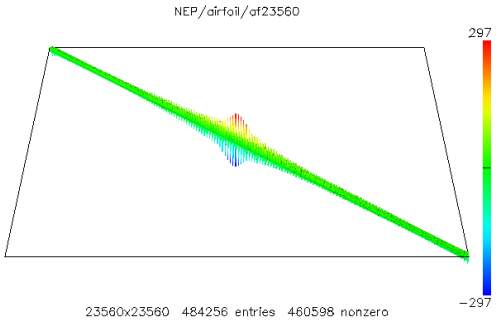
Matrice	Nlpkkt80	Matrice	AF23560
Ordre	1.062.400	Ordre	23560
Non-zéros	28.192.672	Non-zeros	460.598
Problème	Problème d'optimisation	Problème	Eigenvalue problem
Graphe		Graphe	

Tableau 12. Description des matrices nlpkkt80 et AF23560 de l'Université de Floride.

Ainsi, un autre schéma pourrait être le suivant. Nous lançons l'exécution du solveur avec le format de matrice par défaut, et mesurons sa performance. Pendant les calculs effectués à l'itération 1 de la projection (et l'itération 1 du solveur), nous instancions un autre format de matrice de manière asynchrone. A l'itération 2 de la projection (et toujours la première du solveur), nous utilisons alors ce format pour effectuer le produit matrice vecteur, tout en mesurant la performance. Pendant l'itération 2 de la projection, nous pouvons alors charger un troisième format de matrice, et continuer ce processus jusqu'au parcours de toute la bibliothèque de produit matrice vecteur. Une fois tous les noyaux benchmarkés, nous choisissons le meilleur pour les itérations de la projection courante et des projections restantes nécessaires. Avec cette solution, le temps passé à benchmarker les noyaux est utilisé à bon escient, avec certes une performance variable.
Les résultats obtenus avec la version basique implémentée sont plutôt encourageants. Nous utiliserons les matrices nlpkkt80 et AF23560.

On obtient les performances suivantes sur l'exécution complète d'un ERAM de sous-espace égal à 16 avec orthogonalisation CGSr, en GFlops et avec les GPUs de la machine Titane :

	AF23560	Nlpkkt80
Processeur quadcore intel Nehalem	2.33	1.55
Tesla C1060 CSR	0.5 (spmv) – 1.2 (eram)	1.8 (spmv) – 2 (eram)
Tesla C1060 Auto	7.5 (spmv) – 1.7 (eram)	17.5 (spmv) – 13 (eram)
Speed-ups Auto / CSR	15 (spmv) – 1.42 (eram)	9.72 (spmv) – 6.5 (eram)

Tableau 13. Performances obtenues sur les différents matériels suivant les deux matrices exemples.

Pour les GPUs, la première performance indique les GFlops délivrés par le produit matrice vecteur creux (spmv) selon le format de matrice. Le deuxième nombre représente la performance globale du solveur ERAM. On note une amélioration d'un facteur de 15 pour la matrice AF23560 lorsque l'on active l'autotuning, et 9,72 pour la matrice nlpkkt80. Cependant, cela se transcrit par une augmentation de la performance du solveur d'un facteur 1,42 pour AF23560 et 6,5 pour nlpkkt80. Cela est dû en partie à la structure même de ces matrices et la répartition des calculs entre opérations de type matrice-matrice et matrice-vecteur ou vecteur-vecteur. En effet, les opérations BLAS1 peuvent être prédominantes dans les calculs. Pour nlpkkt80, les opérations matricielles sont plus dominantes que pour AF23560. L'ordre de cette dernière est 23.560, d'où des opérations effectuées sur de « petits » vecteurs, et une performance fortement impactée par la latence de la mémoire et le bus PCI-Express reliant la carte graphique au processeur central. L'ordre de nlpkkt80 est beaucoup plus important, 1.062.400, ce qui est assez large pour mieux recouvrir la latence du bus PCI-Express et donc d'obtenir une meilleure performance globale. Malgré tout, l'auto-tuning apporte un gain de performance appréciable, qui potentiellement s'appliquera dans un environnement où plusieurs GPUs sont utilisés conjointement.

4.5.3 Approche statistique de l'auto-tuning

Les techniques d'auto-tuning présentées précédemment se sont concentrées sur un paramètre particulier. Cependant, une très grande quantité d'autres paramètres peuvent influer sur les performances : caractéristiques matérielles, logicielles, problème à résoudre … Il est difficile de

connaitre à l'avance l'influence de ceux-ci sur la performance de l'algorithme ainsi que leurs éventuels corrélations. Ci-dessous et à titre d'illustration, une liste de paramètres pouvant influer sur les performances du produit matrice-vecteur :

- Dimensions de la matrice
- Distribution de la matrice (par blocs, par lignes, par colonnes …)
- Architecture de la machine de calcul (CPU, GPU, MIC, CPU+GPU, CPU+MIC…)
- Modèles de programmation (MPI, OpenMP, OpenACC, TBB, CILK+, CUDA, MPI+OpenMP, MPI+CUDA …)
- Variations dans les degrés de parallélisme (Nombre de processus MPI (P), Nombre de threads (T), Nombre de threads par processus MPI …)
- Compilateurs et paramètres d'optimisation (0x, Loop unrolling, Inlining, …)
- Bibliothèques utilisées
- …

Dans le cadre d'un stage de fin d'études, nous avons entamé une première étude sur l'utilisation de méthodes statistiques pour l'auto-tuning appliquées à un noyau de calcul simple : le produit matrice-vecteur [111].
Etant donné sa mise en œuvre algorithmique, nous utilisons des méthodes statistiques pour trouver des relations entre les paramètres de performance. Ces paramètres et leurs relations ne peuvent pas être mesurés avec des unités quantitatives. Ces relations ne sont pas prévisibles sans expérimentation et une analyse approfondies et elles peuvent changer d'un cas à l'autre.
Sur le cas du produit matrice-vecteur nous avons cherché à identifier les paramètres qui influencent le temps d'exécution, ainsi que les corrélations entre les paramètres de performance. Dans cette étude, les paramètres pris en compte ont été les suivants : optimisation du compilateur, formats de stockages de la matrice, nombre d'éléments non nuls, tailles des blocs et précision (simple ou double). Nous avons généré un «échantillon» de plus de 8.000 individus (un individu étant une exécution du produit matrice-vecteur avec des valeurs fixes pour chacun des paramètres). L'objectif étant de trouver des relations entre les paramètres et de les utiliser pour développer des techniques de smart-tunings afin d'affiner ces paramètres.
La méthode statistique utilisée est l'Analyse en Composantes Principales (ACP) [110, 112]. Le but est de construire plusieurs axes permettant de projeter l'ensemble des observations et paramètres sur ces axes afin d'obtenir une bonne représentation planaire et ainsi de visualiser les relations entre les paramètres et les observations.

Dans cette tout première étude, deux axes principaux concentrent l'essentiel de l'information.

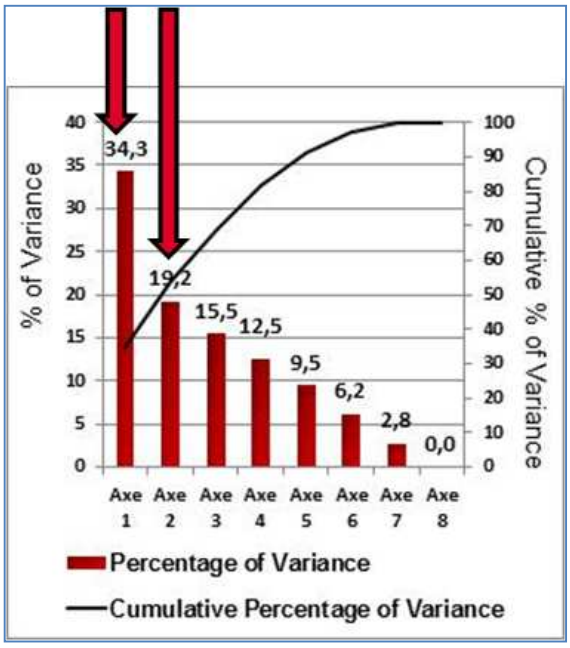


Figure 60. Identification des deux axes principaux résultants de l'ACP.

Cette toute première étude nous a permis de valider l'approche et de montrer que l'utilisation de méthodes statistiques auraient une utilité pour tenter de systématiser la démarche d'auto-tuning en prenant en compte un grand nombre de paramètres. En outre, si jusqu'à présent le principal facteur d'optimisation est la performance, la mise au point d'outils pour une approche systématique de l'auto-tuning permettraient d'optimiser facilement d'autres critères, comme la consommation électrique, le rapport performance/consommation électrique …

4.6 SMART-TUNING DES METHODES D'ARNOLDI

4.6.1 Introduction

Nous avons vu dans le paragraphe précédent quelques techniques d'auto-tuning permettant d'améliorer les performances des méthodes d'Arnoldi. Ces techniques sont relativement indépendantes de l'algorithme lui-même.

Intéressons nous maintenant à des techniques dont le but est toujours d'optimiser les performances de manière dynamique et automatique mais prenant en compte les spécificités de la méthode numériques elle-même.

Dans ERAM, il y a de nombreux paramètres qui peuvent influer sur le taux de convergence et donc les performances, dont notamment la taille du sous-espace de Krylov et les stratégies de redémarrage.

4.6.2 Smart-tuning sur la taille de sous-espace

La taille du sous-espace de Krylov a une influence directe sur la convergence de la méthode ERAM. Une taille trop petite peut conduite à divergence du processus. Une taille trop grande, bien

qu'offrant de meilleures garanties de convergence, entraine une occupation mémoire beaucoup plus importante et un coût d'itération supérieur. Cette taille « optimale » de sous-espace dépend bien évidemment des propriétés de la matrice et des valeurs propres, mais également du nombre de paires (valeur propre, vecteur propre) que l'on cherche à calculer.

L'idée est donc de proposer une méthodologie de détermination auto-adaptative et dynamique de cette taille de sous-espace [115, 71, 111, 113].

Une première heuristique mise en place pour adapter dynamiquement la taille *m* du sous-espace de Krylov est basée sur le calcul d'un taux de convergence. Supposons que nous désirions calculer k couples (λ_i, v_i). Pour chaque vecteur de Ritz, nous calculons le résidu associé r_i. Nous définissons alors le taux de convergence de la méthode ERAM à l'itération *j*, *CRR*, comme :

$$CRR_j = max_{i=1\ldots k}(r_i)$$

En cours des itérations, nous calculons un taux de convergence global moyen, à l'itération *l* définit comme :

$$GCR_l = \frac{1}{l}\sum_{j=1}^{l} CRR_j$$

L'heuristique est alors la suivante :

- Si $GCR_l \leq CR_{min}$, alors la convergence est considérée comme lente (ou divergence) et la taille *m* du sous-espace est augmentée ;
- Si $CR_{min} < GCR_l < CR_{lmax}$, alors convergence est considérée comme « optimale » ;
- Si $GCR_l \geq CR_{max}$, alors la convergence est considérée comme rapide et la taille du sous-espace *m* est diminuée.

Nous avons expérimenté cette heuristique sur le problème suivant :

- Matrice CRY10000, d'ordre 10.000 comportant 49.699 éléments non nuls ;
- Recherche de 20 valeurs propres ;
- Précision recherchée : $10^{e\text{-}09}$

Afin de quantifier le nombre d'opérations effectuées, nous avons pris comme mesure le nombre de produits matrice-vecteur effectués. Les résultats sans smart-tuning sont présentés dans le Tableau 14.

Subspace size	Amount of MVs	Execution time
19 < Subspace < 75	20000< MVs < 75000	No conv. in 1000 it.
75	1875	15,7s
80	800	7,8s
85	595	5,8s
90	450	4,4s

Tableau 14. Performances obtenues sans smart-tuning sur m.

Cet exemple illustre bien la difficulté de trouver la bonne taille de sous-espace pour optimiser le taux de convergence et le temps d'exécution.
Maintenant, nous appliquons notre heuristique de smart-tuning avec différentes valeurs initiales de *m*. Les résultats sont présentés dans le Tableau 15.

Subspace size	With smart-tuning (MVs)	Without smart-tuning (MVs)
***ERAM(20)* (min)**	5.090	No conv
ERAM(75)	2.550	19.125
ERAM(85)	675	680

Tableau 15. Comparaison des performances obtenues avec et sans smart-tuning sur m en termes de nombre de produit matrice-vecteur.

Comme on peut le constater, l'heuristique mise en place permet de converger quelque soit la taille de sous-espace initiale et garanti un nombre d'opérations minimal.

4.6.3 Smart-tuning sur les stratégies de redémarrage

Les stratégies de redémarrages sont un autre paramètre essentiel à la convergence d'ERAM. A l'image de la taille de sous-espace, leur efficacité est extrêmement dépendante de la matrice et des valeurs propres/vecteurs propres recherchés. Cela implique que la détermination a priori d'une stratégie de redémarrage garantissant la convergence « optimale » d'ERAM est quasi impossible. Le travail de smart-tuning autour des stratégies de redémarrage est en cours dans le cadre de la thèse de doctorat de France BOILLOD-CERNEUX, cependant nous avons pu d'ores et déjà proposer une première approche de smart-tuning dans le cadre des co-méthodes qui offrent des propriétés très intéressantes pour le calcul exascale.
Le principe des co-méthodes et leur déclinaison au problème de recherche de valeurs propres est abordé dans le paragraphe suivant.

4.7 LES CO-METHODES

4.7.1 Introduction et principes généraux des co-méthodes

Nous avons décrit dans un paragraphe précédent quelques unes des modifications majeures qui attendent les algorithmes numériques pour passer à l'échelle et pour pouvoir s'exécuter sur les prochaines machines exascales :

- La nécessaire utilisation de tous les degrés de parallélisme offerts par les nouvelles architectures ;
- La minimisation des communications et surtout des communications globales ;
- La tolérance aux pannes.

Dernier facteur à prendre en compte : la minimisation de l'énergie consommée. Ce dernier point sera partiellement abordé ici et fera partie des perspectives à ce travail.

Une des voies que nous avons explorées dans le cadre de nos recherches est la mise en œuvre de co-méthodes pour la recherche de valeurs propres [71, 114, 116, 117].

L'idée des co-méthodes est relativement simple et répond aux exigences citées ci-dessus. Le principe est d'utiliser plusieurs instances d'une même méthode avec des jeux de paramètres différents pour résoudre un problème, chacune des instances de ces méthodes résolvant le même problème. Le principe est une combinaison des méthodes de type « diviser pour régner » et d'une approche multi-paramétrées. En effet, dans une approche de type « diviser pour régner », un seul et même problème est résolu de manière concurrente par un ensemble de processus avec échanges d'informations entre ces processus pour la résolution. Dans une approche multi-paramétrées le même problème est résolu par des processus différents, chacun d'entre eux résolvant le problème en entier, sans qu'il n'y ait d'échanges d'information entre ces processus, le but étant, par exemple, d'obtenir une sensibilité à un jeu de paramètres d'entrées.

Dans les co-méthodes, nous sommes à mi-chemin, car chaque processus va résoudre le même problème intégralement avec un jeu de paramètres différent en entrée, mais des échanges d'informations vont être effectués régulièrement dans le but d'accélérer le processus global de résolution du problème.

Cette approche de co-méthodes possède des propriétés intéressantes vis-à-vis des contraintes liées à l'exascale.

1. Minimisation des communications globales : supposons que nous ayons *P* ressources de calcul pour résoudre notre problème. Alors, si on utilise *N* instances de méthodes pour résoudre notre problème, les communications globales ne se feront que sur un périmètre de P/N à comparer à P si l'on utilise toutes les ressources pour résoudre ce même problème ;
2. Tolérances aux pannes : du fait que l'on a N instances pour résoudre un seul et même problème, on ajoute une propriété de tolérance aux pannes que la méthode initiale de possède pas, puisque l'on peut se permettre de perdre jusqu'à N-1 instances ;
3. La meilleure exploitation des différents degrés de parallélisme : le principe des co-méthodes permet d'exploiter le niveau le plus haut du parallélisme qui est le parallélisme distribué de type MPMD[11] [121], qui n'est généralement pas exploité lors d'une approche classique de type SPMD[12] [121].

Néanmoins à la base on peut considérer que cette approche par co-méthodes est moins efficace en termes de performances qu'une approche classique du fait que l'on utilise N fois moins de processus pour résoudre le problème initial. Cependant, deux facteurs permettent d'espérer un gain notable :

- L'accélération mutuelle des méthodes concurrentes ;
- La minimisation des communications globales et donc le gain en terme de scalabilité.

Nous allons donc illustrer cette approche sur la recherche de valeurs propres et la mise en œuvre de MERAM (Multiple Explicit Restarted Arnoldi Method).

[11] MPMD : Multiple Program Multiple Data
[12] SPMD : Single Program Multiple Data

4.7.2 MERAM

4.7.2.1 Principe de MERAM

L'application du principe des co-méthodes à la résolution d'un problème à valeurs propres consiste à utiliser plusieurs méthodes ERAM concurrentes, qui collaborent ensemble dans le but de résoudre le même problème : le calcul de certaines valeurs propres d'une matrice. Cette méthode, développée dans [114, 116, 117] se nomme Multiple ERAM (MERAM).

La figure 61 montre un exemple d'exécution de MERAM avec quatre solveurs ERAM utilisant des sous-espaces de tailles variées. En effet, les ERAMs peuvent utiliser des paramètres d'entrées différents tels que la taille du sous-espace ou le vecteur initial. Les autres éléments pouvant différer entre les co-méthodes peuvent être par exemple l'orthogonalisation utilisée, la précision flottante, ou encore le nombre de valeurs propres calculées.

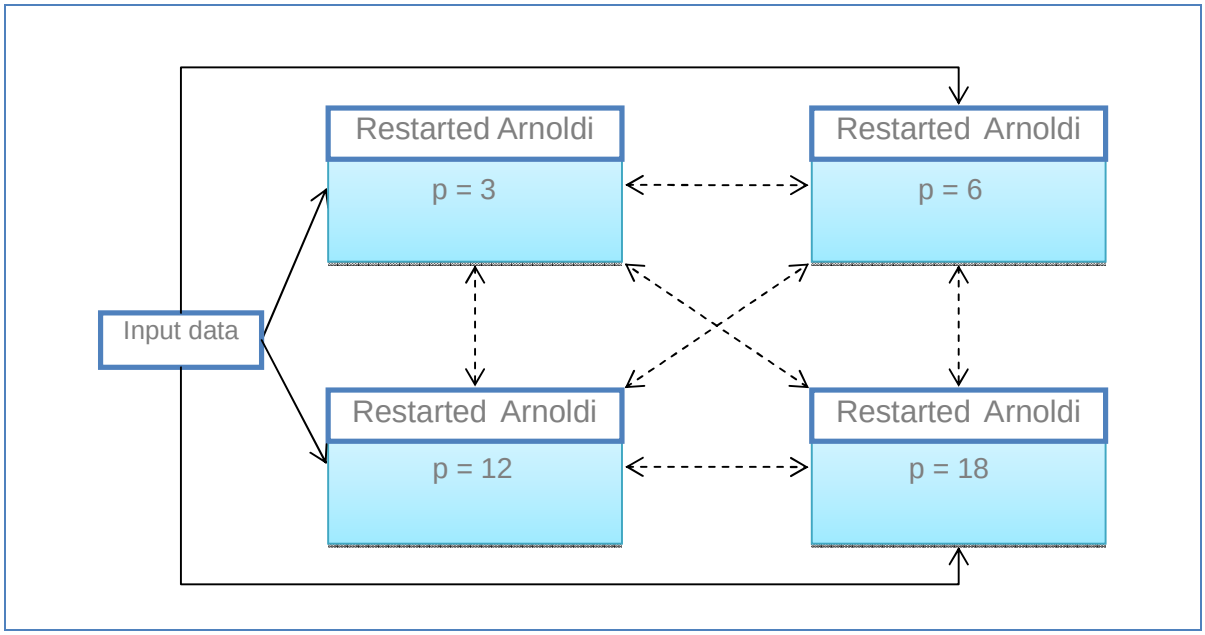


Figure 61. Exemple d'exécution de Multiple ERAM, une méthode hybride pour le calcul de valeurs propres d'une matrice.

Le fonctionnement de base de MERAM est assez simple : les ERAMs sont exécutés en parallèle, de manière asynchrone. Pour rappel, à chaque calcul de valeurs propres effectué par un ERAM, ce dernier va utiliser une combinaison linéaire de ses vecteurs propres pour calculer le vecteur de redémarrage pour l'itération suivante. MERAM propose que les ERAMs échangent leurs informations à la fin de chaque itération de manière asynchrone, et réutilisent potentiellement l'information des autres ERAMs pour calculer le vecteur de redémarrage.

Intuitivement, on peut comprendre l'intérêt de garder le meilleur de l'information entre tous les solveurs pour espérer avoir une convergence optimale. Si l'on prend l'exemple de deux ERAMs collaborant ensemble, avec l'un utilisant un sous-espace de taille 3 (ERAM(3)) et l'autre un sous-espace de taille 6 (ERAM(6)), alors la question de l'utilité pour ERAM(6) d'utiliser l'information d'ERAM(3) peut se poser. En effet, ERAM(6) est plus précis grâce à son plus grand sous-espace, et ne devrait donc pas bénéficier de l'exécution concurrente d'ERAM(3). La réalité est différente, car le comportement de convergence d'un ERAM n'est pas déterministe. Ainsi, il est possible qu'ERAM(3) converge plus rapidement dans un premier temps, et puisse faire bénéficier ERAM(6) d'une meilleure information sur le sous-espace contenant les valeurs propres. La figure 62 illustre ce comportement.

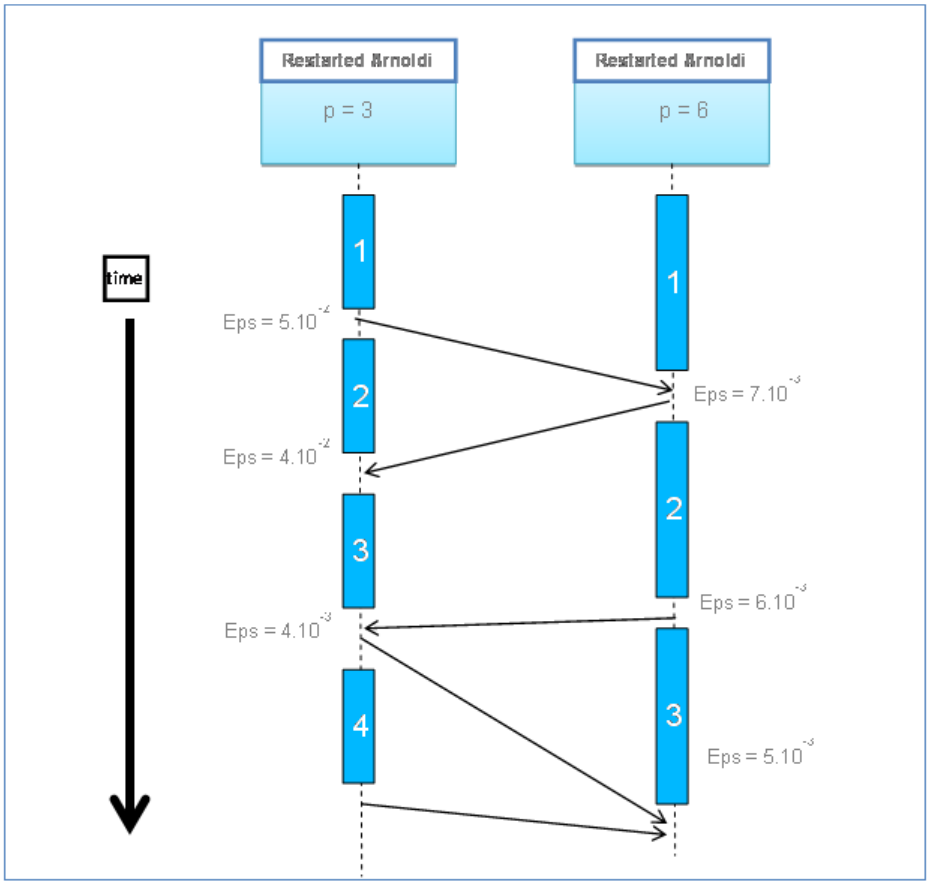


Figure 62. Illustration de l'évolution des échanges entre deux ERAM lors de l'exécution de MERAM. Les rectangles pleins représentent le temps d'une itération, et les flèches l'envoi d'informations à un autre solveur. A la fin de l'itération 2, le solveur ERAM(3) utilise l'information de l'autre solveur, bénéficiant d'une accélération numérique. ERAM(6) bénéficie à son tour d'un meilleur résultat pour l'itération 4, non présentée sur le schéma.

Dans ce schéma, les deux solveurs ont bénéficié du résultat de l'autre pendant l'exécution, et ont globalement convergé plus rapidement que s'ils avaient été exécutés séparément avec le même nombre de processeurs. Nous illustrons ainsi à la fois l'accélération numérique que peut apporter

MERAM, mais aussi l'augmentation de la scalabilité. En effet, chaque solveur conserve sa scalabilité propre, et le fait d'en exécuter plusieurs de manière concurrente permet d'utiliser plus de processeurs, sans perte de performance. De plus, si un ou plusieurs d'entre eux venaient à s'arrêter à cause d'une faute matérielle, alors il suffirait qu'au moins un solveur s'exécute pour que les calculs puissent continuer à converger. MERAM est ainsi intrinsèquement tolérant aux pannes.
Dans [116], nous avons présenté une implémentation de cette méthode avec optimisation des communications.

4.7.2.2 Performances à l'échelle d'un supercalculateur

Dans un paragraphe précédent, nous avions rencontré une limite dans la scalabilité d'ERAM(9) sur le supercalculateur Titane avec une matrice de Dingdong[13], ainsi que pour le supercalculateur Curie avec une matrice EXP46080. Nous avons expérimenté le cas Dingdong avec MERAM, en utilisant 3 solveurs avec des tailles de sous-espaces respectivement de 4, 7 et 9. La différence ici est que ces trois solveurs se partagent le même nombre de processeurs total qui était uniquement alloué au solveur ERAM(9) du chapitre précédent. Les résultats sont présentés sur la Figure 63.
Nous voyons que le temps de résolution du problème est fortement réduit par rapport à ERAM, pour toutes les exécutions. Pour rappel, il était de 3.800s sur un seul nœud avec un ERAM(9). Il nous permet d'atteindre un temps d'exécution réduit à 4,4 secondes, contre 158s au mieux avec un seul ERAM(9) précédemment. Le speed-up entre 1 nœud (8 cœurs) et 148 nœuds (1184 cœurs) est ainsi de 64,6x, soit une efficacité de 44%. Comparativement, l'utilisation d'un seul ERAM nous permettait une efficacité de 9% dans le chapitre précédent.

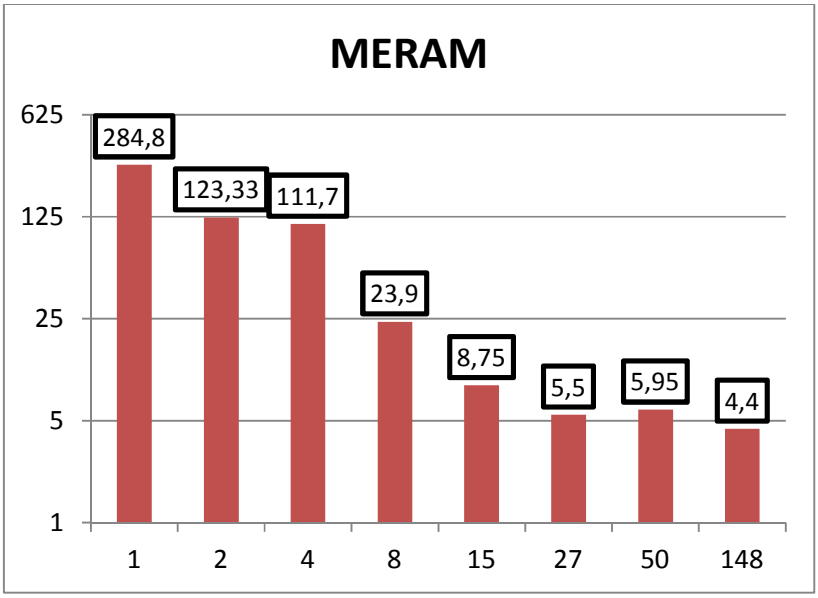


Figure 63. Performances de MERAM utilisant trois solveurs ERAM avec des sous-espaces de taille 4, 7 et 9. Le cas traité est le même que dans le chapitre précédent, lors de l'étude de la scalabilité d'ERAM. L'axe des abscisses indique le nombre de nœuds de 2*quadcores de la machine Titane. Sur les ordonnées, on peut lire le temps en secondes.

[13] Matrix Market - http://math.nist.gov/MatrixMarket/

4.7.2.3 Smart-tuning et co-méthodes : adaptation de la stratégie de redémarrage

Comme nous l'avons abordé précédemment, la stratégie de redémarrage est un paramètre très important pour la convergence de MERAM. Lorsqu'un solveur semble avoir le meilleur résultat mais offre une convergence plus lente, les autres solveurs se mettent alors au pas de ce dernier.

Nous avons proposé dans [116, 113, 111] d'améliorer cette stratégie en affinant les critères de redémarrage en utilisant une heuristique. A la fin de chaque itération, nous essayons d'avoir le meilleur résultat provenant des autres solveurs, que nous appellerons *best_result*. Le résultat de l'itération précédente est *previous_result*, et celui de l'itération courante est *current_result*. Nous calculons le taux de convergence du solveur avec la formule :

$$Taux\_convergence = current\_result / previous\_result,$$

et le comparons au taux de convergence que nous aurions eu en utilisant le meilleur résultat à la place de celui obtenu à l'itération actuelle :

$$Taux\_hypothetique = best\_result / previous\_result.$$

Par la suite, nous étudions le ratio $speed\text{-}up = Taux\_hypothetique / Taux\_convergence$, et si ce dernier est suffisamment grand, alors le solveur a un intérêt à utiliser le meilleur résultat en provenance d'un autre solveur. Avec un speed-up suffisamment grand, le solveur est assuré de bénéficier d'une amélioration importante de la qualité de sa solution.

Cette technique a permis d'obtenir une efficacité proche de 100% dans les cas où nous observions des pertes d'efficacité de la méthode MERAM. Des expérimentations sur la matrice EXP46080 ont ensuite été menées avec cette stratégie de redémarrage activée, pour obtenir les résultats présentés dans la figure 64. Nous avons pris comme base l'exécution à 1056 cœurs sur la machine CURIE. Nous avons choisi de faire appel à MERAM avec trois ou quatre solveurs ERAM utilisant chacun 1024 cœurs. Par rapport à une exécution d'un seul ERAM à 1056 cœurs, le temps est diminué suivant les accélérations du tableau 16.

On voit qu'un MERAM utilisant 3073 cœurs offre une meilleur performance qu'un ERAM utilisant le même nombre de cœurs, avec une amélioration d'un facteur 1,5x. Ce même MERAM offre également un niveau de performances presque équivalent à un ERAM utilisant 5121 cœurs. Lorsque l'on ajoute un solveur supplémentaire à MERAM, utilisant donc 1024 cœurs de plus pour un total de 4096, alors la performance s'améliore légèrement, offrant un gain de 18% par rapport à l'ERAM utilisant 5121 cœurs.

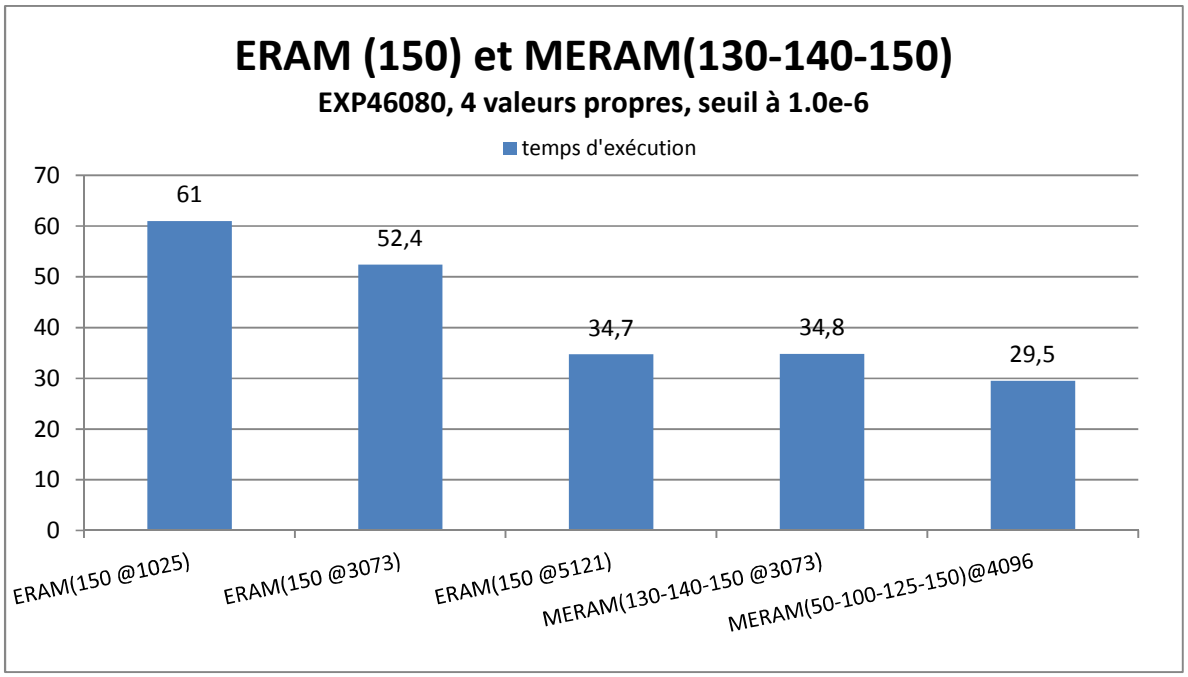


Figure 64. Exécutions d'ERAM et MERAM sur la machine Curie pour calculer les 4 valeurs propres de plus grande valeur absolue de la matrice EXP46080 avec un seuil de 1.0e-6.

	ERAM (150@1025)	ERAM (150@3073)	ERAM (150@5121)	MERAM (130-140-150 @3073)	MERAM (50-100-25-50 @4096)
itérations	44	75	54	14	19
Temps d'exécution (s)	61	52.4	34.7	34.8	29.5
Speed-up	1	1,16	1,76	1,75	2,07

Tableau 16. Accélération apportée par le parallélisme "brut" d'ERAM ou l'utilisation de co-méthodes via MERAM.

Nous avons vu qu'utiliser plusieurs ERAMs permet un gain de performance, mais ce dernier n'est pas équivalent à celui qui aurait été obtenu si un seul ERAM avait pu maintenir sa scalabilité avec un nombre de processeurs équivalent. Cependant, on constate une amélioration grâce à MERAM, qui n'aurait pas été possible par utilisation de parallélisme « brut ». MERAM introduit également d'autres propriétés, telles que la tolérance aux pannes, que nous verrons dans la partie suivante.

4.7.2.4 Tolérance aux pannes

La tolérance aux pannes est une capacité intrinsèque de la méthode MERAM, et nous proposons de la tester dans l'hypothèse de l'utilisation d'une machine très large qui subirait des pannes matérielles. Plusieurs séries de tests ont été effectuées : l'arrêt à un nombre d'itérations donné d'un des solveurs, qui représente une panne aléatoire sur un ou plusieurs des nœuds d'un seul solveur, et une étude statistique de la résistance de MERAM à une dégradation massive de la stabilité de la machine.

4.7.2.4.1 Stabilité lors de la perte d'un solveur

Nous avons choisi une matrice, AF23560 en provenance du site MatrixMarket[14], et nous avons fixé différentes configurations d'exécutions de MERAM, par rapport à un ERAM standard. Cet ERAM standard utilise un sous-espace de taille 10 et converge en 1001 itérations à une précision de 10^{-15}. Les deux premières configurations testées utilisent respectivement des ERAMs de sous-espaces 4 et 10, et 3, 4, 7 et 10. Leurs convergences s'effectuent respectivement en 140 et 100 itérations. Le tableau 17 résume ces configurations.

Solveur	Sous-espaces	Itérations
ERAM(10)	10	1001
MERAM(4-10)	4 - 10	140
MERAM(3-4-7-10)	3-4-7-10	100

Tableau 17. Différentes configurations pour tester la résilience de MERAM, en comparant le résultat à un ERAM standard.

La figure 65 et la figure 66 montrent le résultat de quelques expérimentations. Pour MERAM(4-10), le solveur 0 est ERAM(4) et le solveur 1 ERAM(10). Il en va de même avec MERAM(3-4-7-10), où les solveurs 0, 1, 2, 3 sont respectivement les ERAM 3, 4, 7 et 10.

Sur ces deux graphes, nous voyons premièrement que MERAM est toujours capable de fournir un résultat malgré la perte d'un solveur. Cependant, dans le graphe concernant MERAM(4-10), les tests où le solveur ERAM(10) est arrêté, il ne reste plus alors que le solveur ERAM(4), ne convergent pas ou plus vers la précision demandée en un nombre d'itérations inférieur à celui d'un ERAM(10) seul. Ce comportement est normal car en réalité le solveur ERAM(4) est incapable d'atteindre une précision de 10^{-15} à cause d'une taille de sous-espace trop petite. Par contre, lorsqu'il ne reste plus que le solveur ERAM(10), nous voyons que ce dernier converge vers un résultat satisfaisant, en ayant bénéficié de l'accélération d'ERAM(4).

Dans le deuxième graphe, l'arrêt d'un des solveurs n'a pas une incidence majeure sur la convergence finale de MERAM(3-4-7-10). Le pire scénario est lorsque le solveur de plus petit sous-espace ERAM(4) est arrêté rapidement, ne faisant plus bénéficier les autres solveurs de ses calculs. Alors le nombre d'itération nécessaires augmente de 100 à 202, diminuant la performance finale de MERAM, qui reste cependant meilleure que celle d'un seul ERAM(10) utilisant tous les processeurs pour son calcul : 1001 itérations. Dans les autres cas, MERAM(3-4-7-10) converge en 96 à 120 itérations contre 100 itérations pour MERAM(3-4-7-10). On voit que lorsque le solveur 2 est coupé artificiellement prématurément, MERAM converge légèrement plus vite. Il doit donc encore y avoir des possibilités d'améliorer la stratégie de redémarrage.

[14] http://math.nist.gov/MatrixMarket/

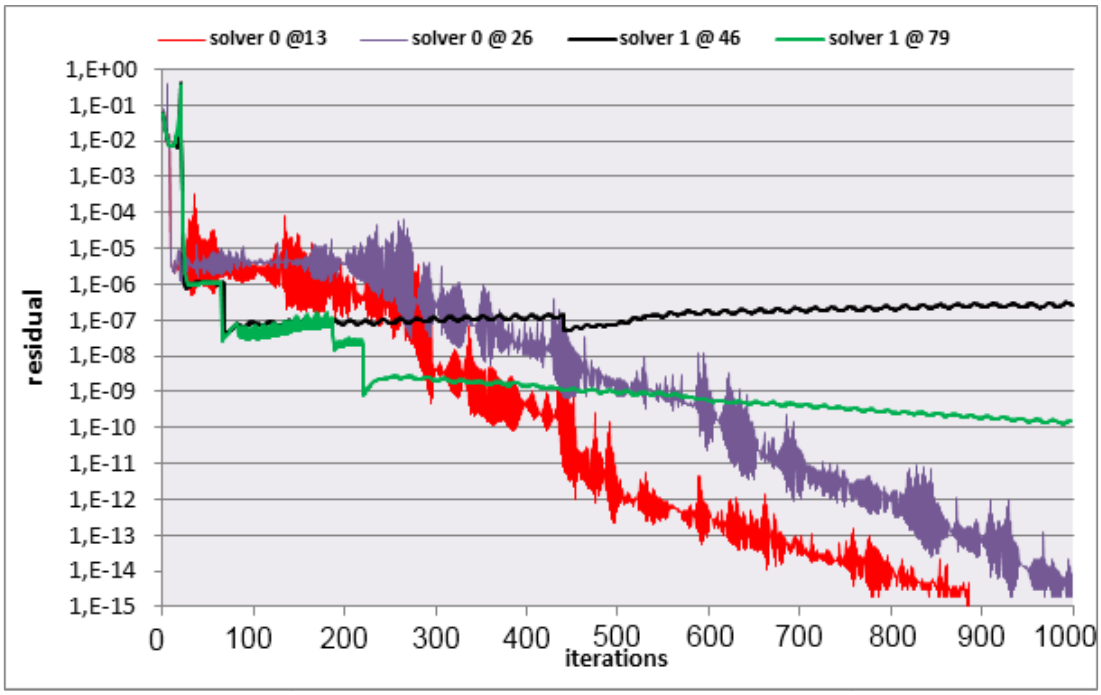


Figure 65. Évolution de la convergence lorsque MERAM(4-10) perd un de ses solveurs. La légende indique quel solveur a été arrêté artificiellement à quelle itération.

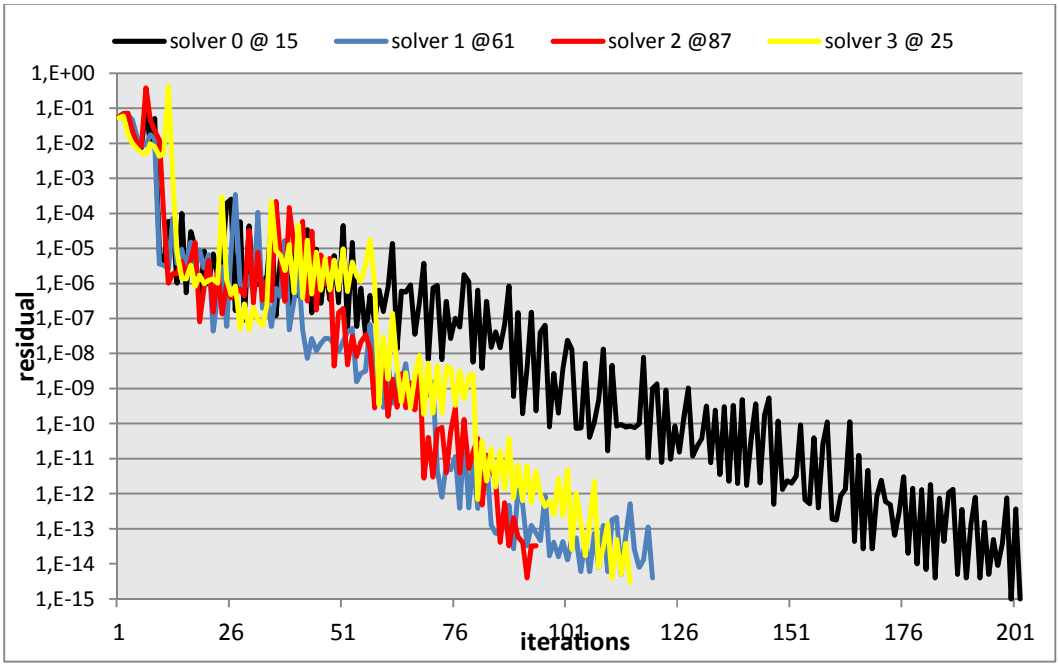


Figure 66. Évolution de la convergence lorsque MERAM(3-4-7-10) perd un de ses solveurs. La légende indique quel solveur a été arrêté artificiellement, à quelle itération.

4.7.2.4.2 Statistiques lors de la perte progressive de plusieurs solveurs

Nous avons repris les tests effectués sur la machine Curie avec la matrice EXP46080 comme base d'une étude de différents scénarii. Étant limité à environ 5000 cœurs, nous avons fixé un ERAM

utilisant 4097 cœurs comme base de résultats. Nous avons ensuite choisi d'exécuter un MERAM avec 4 solveurs ERAM, dont les tailles de sous-espaces sont 50, 100, 125, 150. Les tests effectués sont plutôt pessimistes : 2 cas où 2 solveurs sont arrêtés rapidement, et deux cas où trois solveurs sont arrêtés. Les résultats sont fournis dans le tableau suivant, où un astérisque symbolise les solveurs subissant une panne.

Dans chaque case, le nombre d'itérations est donné pour chaque solveur, avec le temps d'exécution global dans la colonne de droite.

solveur	50	100	125	150	Temps (s)
Cas 0	57	29	23	19	29,5
Cas 1	53	10*	23	8*	29,96
Cas 2	13*	62	3*	44	52,7
Cas 3	13*	10*	3*	44	59,6
Cas 4	900	10*	3*	8*	365,86

Dans les cas 1 à 3, MERAM converge dans un temps égal à celui sans défaillance ou en mettant deux fois plus de temps. Le seul cas qui n'apporte pas une convergence au seuil de 10^{-6} est le cas 4. Cependant, le résultat final obtenu est d'une précision de 1.1×10^{-6}. Si l'on compare les cas 1 à 3 à l'exécution d'un seul ERAM avec un sous-espace de taille 150 dont le temps d'exécution est 61 secondes, alors on voit que l'on est au pire comparable au cas où un seul solveur est utilisé. Le cas 4, plus pathologique, n'est pas satisfaisant.

Pour avoir une statistique de plus grande échelle, nous avons pris comme base un solveur utilisant 256 cœurs, et nous avons instancié un MERAM utilisant 10 solveurs ERAM de 256 cœurs chacun avec des sous-espaces de 75, 100, 110, 120, 125, 130, 135, 140, 145, 150. Les scenarii envisagés sont les suivants :

- Cas 1 : les 7 solveurs de sous-espaces inférieurs à 140 s'arrêtent
- Cas 2 : les 3 solveurs de sous-espace supérieurs à 135 s'arrêtent
- Cas 3 : les solveurs de rang pairs s'arrêtent
- Cas 4 : les solveurs de rang impairs s'arrêtent
- Cas 5 : le solveur de rang le plus grand s'arrête

Les résultats sont présentés dans le tableau suivant, avec les solveurs marqués d'un astérisque subissant une panne.

solveur	75	100	110	120	125	130	135	140	145	150	Tps
Cas 0	102	73	66	58	58	55	40	47	32	35	208
Cas 1	4*	10*	6*	8*	2*	15*	20*	77	49	51	274
Cas 2	77	56	50	47	41	41	28	5*	8*	7*	154
Cas 3	5*	81	8*	59	15*	60	20*	49	11*	29	220
Cas 4	119	10*	80	8*	69	15*	44	5*	43	7*	240
Cas 5	-	-	-	-	-	-	-	-	-	7*	215

Contrairement à l'expérimentation précédente, tous les cas atteignent le critère de convergence. Les temps sont globalement proches ou supérieurs, augmentant de 3 à 32% par rapport à un cas non perturbé par des pannes. Par contre le cas 2 est intéressant car il offre un temps meilleur de

26% et un nombre d'itérations plus faible que dans le cas où il n'y a pas de panne. Cela indique que certains solveurs n'apportent peut-être pas toujours une contribution utile aux calculs, et peuvent perturber la convergence globale de MERAM. Il existe encore des voies d'amélioration à MERAM au niveau de la gestion des co-méthodes, qui mériteraient d'être investiguées dans des travaux ultérieurs.

5 CONCLUSIONS ET PERSPECTIVES

Nous avons présenté dans ce document les recherches que j'ai menées, notamment dans le cadre de l'encadrement de stages de fin d'études et de thèses de doctorat, et qui ont contribué au développement de l'utilisation du calcul parallèle dans les applications scientifiques dans le domaine de la simulation numérique pour les systèmes nucléaires. Ces contributions ont été de plusieurs ordres : informatiques, algorithmiques et numériques.

Au cours de ces recherches, j'ai pu travailler sur la conception d'un code parallèle tridimensionnel en thermo-hydraulique, proposant des solutions novatrices et toujours d'actualité pour la simulation d'écoulements 3D dans des géométries complexes tels que l'on peut les trouver dans les systèmes nucléaires. La poursuite de ces travaux s'est orientée la gestion de maillages non constant en temps (adaptation de maillages, multi grilles géométrique …) dans le cadre d'une exécution parallèle. Ces travaux ont montré toute l'importance de penser et concevoir en amont le parallèlisme de l'application ainsi que les objets/composants le mettant en œuvre au plus profond du code et ce afin d'assurer une efficacité et une pérennité de l'application parallèle.

Après ces premiers travaux sur une application à vocation industrielle en thermo-hydraulique, mes recherches m'ont amenées à travailler sur un autre grand domaine de la simulation numérique pour les systèmes nucléaires qu'est la neutronique et la physique des réacteurs. J'ai pu ainsi, dans ce cadre m'intéresser à l'adaptation de codes et de solveurs à l'émergence des nouvelles architectures de calculateurs comme les accélérateurs de calcul. Ces recherches se sont appuyées sur une étude de plusieurs années sur les enjeux et les défis à l'utilisation du HPC dans le domaine de la physique des réacteurs et des codes de simulation numérique pour les systèmes nucléaires.

L'évolution rapide des architectures de calcul pour atteindre le petascale et qui vont encore survenir pour aller jusqu'à l'exascale entrainent des modifications profondes des applications et des algorithmes. Afin de mieux cerner ces évolutions et leurs impacts sur les codes de simulation (langages de programmation, nouvelle algorithmique, impacts sur l'architecture des codes…) nous avons fait le choix de nous pencher sur la résolution de problème à valeurs propres par des méthodes Krylov, car ces problèmes sont non seulement représentatifs d'une grande classe d'applications scientifiques et les méthodes de Krylov sont très largement répandues dans le monde académique et industriel.

Grâce à ces recherches, nous avons pu ainsi monter en expertise sur la programmation des accélérateurs, proposer des méthodologies pour la programmation efficace des architectures massivement multi-cœurs homogènes et hétérogènes comme les techniques d'auto-tuning et de smart-tuning. Nous avons également travaillé sur de nouvelles classes de méthodes, les co-méthodes, avec MERAM, qui, comme nous l'avons montré, offrent de nombreuses propriétés intéressantes pour l'ultra-scale.

Cependant le travail ne s'arrête pas là, puisque deux thèses de doctorat sont en cours. La première, celle de France BOILLOD-CERNEUX, se concentre les techniques de smart-tuning autour d'ERAM et de MERAM. Ce point a été abordé brièvement dans le document. Nous avons cependant bien insisté sur le rôle critique de la stratégie de redémarrage dans le cadre d'ERAM. Tout le travail consiste donc à bien maitriser ce processus de converger et le contrôler par le biais de l'ajustement dynamique de la stratégie de redémarrage pour assurer une convergence optimale en fonction de l'environnement d'exécution et du problème à résoudre. Ces stratégies vont bien sûr être appliquées à MERAM. Au-delà des méthodes d'Arnoldi pour la résolution de problèmes à valeurs propres, ces techniques peuvent être généralisées à d'autres méthodes de Krylov, notamment pour la résolution de systèmes linéaires.

Un deuxième axe de recherche, dans le cadre de la thèse de doctorat de Fan YE, plus lié à l'informatique, concerne la programmation efficace des accélérateurs de calcul et la mise en œuvre d'algorithmes numériques génériques sur architectures massivement multi-cœurs homogènes et hétérogènes. Lors des travaux que nous avons réalisés sur la programmation des GPUs, nous avons vu la limitation intrinsèque à la scalabilité due à l'interconnexion des accélérateurs à un CPU hôte. Nous avons également insisté sur la nécessaire optimisation des communications. C'est donc un des axes de recherche de cette thèse de doctorat. Un autre thème concerne l'exploitation de tous les niveaux de parallélisme, notamment par le parallélisme de type SIMD qui arrive avec les technologies AVX d'INTEL et que l'on retrouve sur les derniers processeurs Xeon (SandyBridge, IvyBridge, futur Haswell et les accélérateurs MIC Xeon Phi). Ainsi, toujours en prenant comme base de travail les méthodes de Krylov, nous nous attaquons à la programmation efficace des INTEL Xeon Phi en étudiant les différents niveaux de parallélisme et les langages de programmation offerts pour cela.

Enfin, le dernier axe de recherche en cours, concerne la conception, l'intégration et la pérennisation de ces développements. Au cours de la thèse de doctorat Jérôme DUBOIS un environnement de travail avait été développé : KASH. Bien que répondant totalement au cahier des charges initial, il s'est avéré très rapidement lourd à maintenir et à développer dans le cadre d'un travail de R&D amont. En outre, notre but étant, non seulement de progresser dans la connaissance et la maitrise de l'évolution de la simulation numérique parallèle, mais également de pérenniser et capitaliser ces acquis pour qu'ils puissent être réutilisables dans les grandes applications numériques dans le domaine de la simulation numérique pour le nucléaire. En outre des travaux très intéressants menés par Nahid EMAD, Tony DRUMMOND et Makarem DANDOUNA ont permis de progresser vers la conception de bibliothèques numériques réutilisables [123, 124, 125, 122]. C'est pourquoi nous avons décidé de travailler sur l'intégration de tous ces développements dans une bibliothèque numérique internationalement reconnue : TRILINOS [126]. Nous nous proposons donc d'étendre cette bibliothèque afin de pouvoir proposer nos algorithmes itératifs de

résolution de systèmes linéaires et de problèmes à valeurs propres avec auto-tuning, smart-tuning sur multi-cœurs et accélérateurs (GPU et Intel MIC).

Comme nous l'avons mentionné à plusieurs reprises dans ce document la parallélisation efficace d'applications scientifiques couvre de nombreux domaines qui vont du numérique à l'informatique et un lien fort avec les phénomènes physiques simulés. La simulation numérique est désormais un passage incontournable dans la conception de systèmes nucléaires performants et sûrs. Les exigences et les contraintes liées au développement de codes à vocation industrielle et plus particulièrement dans le domaine de l'énergie nucléaire, ajoutent encore un degré de complexité supplémentaire à l'utilisation du calcul intensif. Nous en avons illustré quelques points comme la portabilité des codes développés, la problématique de la vérification et la validation en exécution parallèle, l'intégration d'environnements et de modèles de programmation spécifiques adaptés à une architecture matérielle dans une application qui doit s'adapter à de nombreux contextes d'utilisations …

Ainsi les défis qui se présentent à nous pour l'ère de l'ultra-scale (machines de plus de 100 PFlops avec des degrés de parallélisme au-delà du million) sont à relever dès aujourd'hui et doivent couvrir l'ensemble des domaines : physique, numérique, algorithmique et informatique. Les échelles de temps des mutations informatiques ne sont pas du tout les mêmes que celles du développement de grandes applications de simulations pour le nucléaire. Il est donc indispensable d'anticiper au mieux les évolutions à venir et de concevoir des architectures logicielles, des algorithmes, des méthodes numériques … suffisamment adaptables pour accepter les évolutions non prévues.

Il y a donc de nombreux champs d'études à couvrir, comme la tolérance aux pannes au niveau de l'application et de l'algorithme, réinventer des méthodes numériques et des algorithmes permettant de passer à l'échelle (des algorithmes polymorphes qui n'ont pas nécessairement la même implémentation en fonction du degré de parallélisme visé), l'émergence de nouveaux modèles de programmation autorisant l'expression la plus simple possible de tous les niveaux de parallélisme qui s'offrent …

Toutes ces voies de R&D sont à explorer via des collaborations nationales et internationales. C'est déjà le cas dans les travaux qui ont été présentés (collaboration avec le Lab. d'Informatique Fondamentale de Lille[15], les liens avec la Maison de la Simulation[16], les actions de recherches menées au sein du projet PRACE[17], le projet Franco-Japonais FP3C[18], la collaboration avec le Lawrence Berkeley National Lab.) mais elles doivent continuer et se développer afin de progresser

[15] www.lifl.fr/
[16] www.maisondelasimulation.fr/
[17] Partnership for Avanced Computing in Europe - http://www.prace-ri.eu/
[18] Framework and Programmng for Post Petascale Computing - http://jfli.nii.ac.jp/medias/wordpress/?page_id=327

dans la connaissance et la maitrise de ces challenges. Il est également indispensable, en parallèle des progrès faits au sein des collaborations, de conserver et augmenter les compétences au sein des équipes de développement des grands applicatifs dans les domaines du calcul hautes performances.

6 REFERENCES

1. **C. CALVIN**, *« Minimisation du surcoût des communications dans la parallélisation des algorithmes numériques. »,* Thèse de doctorat de l'INPG (1995).
2. G. K. BATCHELOR, *"An Introduction to Fluid Dynamics",* Cambridge University Press, ISBN 0-521-66396-2 (1967).
3. G. I. MARCHUK AND V. I. LEBEDEV, *"Numerical Methods in the Theory of Neutron Transport."* Taylor & Francis. p. 123. ISBN 978-3-7186-0182-0 (1986).
4. C. CRAWFORD, D. NEWMAN, AND G. KARNIADIS, *"Parallel Benchmarks of Turbulence in Complex Geometries"*, in Proceedings of Parallel CFD'95 (1995).
5. O. JAYASIMHA, M. HAYDER, AND S. PILLAY, "*Parallelizing Navier-Stokes Computations on a Variety of Architectural Platfrorms"*, in Proceedings of Supercomputing'95 (1995).
6. G. DEGREZ, L. GIRAUD, M. LORIOT, A. MICELTA, B. NITROSSO, AND A. STOESSEL, *"Parallel Industrial CFD Calculations with N3S"*, in Proceedings of HPCN'95, Lecture Notes in Computer Science, pp. 820-825 (1995).
7. C. FARHAT AND S. LANTERI, *"Simulation of Compressible Viscous Flows on a Variety of MPPs: Computational Algorithms for Unstructured Dynamic Meshes and Performances Results"*, Comp. Meth. in Appl. Mech. and Eng., pp. 35-60 (1994).
8. S. LANTERI, *"Parallel Solutions of Three-Dimensional Compressible Flows"*, tech. rep., INRIA, Sophia Antipolis (1995).
9. E OLSSON, J. RANTAKKA, AND M. THUNE, "*Software Tools for Parallel CFD on Composite Grids"*, in Proceedings of Parallel CFD'95 (1995).
10. F T. LEIGHTON, *"Introduction to Parallel Algorithms and Architectures: Arrays - Trees – Hypercubes"*, Morgan Kaufman Publishers, Inc. (1992).
11. K. HOFFMANN AND J. ZOU, *"Parallel Efficiency of Domain Decomposition Methods"*, Parallel Computing, pp. 1375-1391 (1993).
12. G. MEURANT, *"Domain Decomposition Methods for P.D.E.s on Parallel Computers"*, International Journal Supercomputer Applications, pp. 5-12 (1988).
13. P. LE TALLEC, *"Domain decomposition methods in computational mechanics"*, Computational mechanics advances, pp. 121-220 (1994).
14. G. KARYPIS AND V. KUMAR, METIS, *"Unstructred Graph Partionning and Sparse Matrix Ordering"*, tech. rep., University of Minnesota, Dept. of Computer Science, Minneapolis, MN 55455 (1995).
15. 15. F. PELLEGRINI, *"SCTOCH3.1: User's Guide"*, tech. rep., LaBRI, Université de Bordeaux I, (1996).
16. **C. CALVIN**, PH. EMONOT, *"The TRIO-Unitaire Project: A Parallel CFD 3-Dimensional Code."* ISCOPE'97, pp. 169-176 (1997)
17. **C. CALVIN**, PH. EMONOT, *"The parallelism in the Trio-Unitaire Project."* 8th International Topical Meeting on Nuclear Reactor Thermal Hydraulics (NURETH-97), Kyoto, Japan (1997).
18. **C. CALVIN**, OLGA CUETO, P. EMONOT, *"An object-oriented approach to the design of fluid mechanics software."* M2AN, Vol. 36, N°5, pp. 907-921 (2002)
19. **C. CALVIN**, *"A development framework for parallel CFD applications: Trio-U project "*, 5th International Conference on Supercomputing for Nuclear Applications (SNA - 2003), Paris, France (2003).

20. M. CROUZEIX AND P-A. RAVIART, *"Conforming and non conforming finite element methods for solving the stationary Stokes equations."* RAIRO, R3, pp 33-75 (1973).
21. S. BALAY, W.D. GROPP, L.C. MCINNES AND B. F. SMITH, "*Efficient Management of Parallelism in Object Oriented Numerical Software Libraries*", in Modern Software Tools in Scientific Computing, E. Arge and A. M. Bruaset and H. P. Langtangen eds, pp 163-202 (1997).
22. E. CHOW, A.J. CLEARY AND R.D. FALGOUT, "*Design of the hypre Preconditioner Library*", in Proc. of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing, Mike Henderson, Chris Anderson, and Steve Lyons, eds, Philadelphia (1998).
23. B. SÉCHER, M. BELLIARD, AND **C. CALVIN**, *"Numerical Platon: A unified linear equation solver interface by CEA for solving open for scientific applications."*, Nuclear Engineering and Design, Volume 239, Issue 1, Pages 87-95 (2009)
24. P. SAGAUT, *"Large Eddy Simulation for Incompressible Flows",* (Third ed.). Springer. (2006).
25. U. BIEDER, **C. CALVIN** ET AL, *" Detailed Thermal Hydraulic Analysis of Induced Break Severe Accidents using the massively parallel CFD code Trio_U/PRICELES"*, 5th International Conference on Supercomputing for Nuclear Applications (SNA - 2003) ; 22/09/2003 - 24/09/2003 ; Paris ; France
26. U. BIEDER, **C. CALVIN**, P. EMONOT, *"PRICELES: A Parallel CFD 3-Dimensional Code for Industrial Large Eddy Simulations"*, Parallel Cfd (2000)
27. U. BIEDER, **C. CALVIN**, P. EMONOT, *"Priceles: An Object Oriented Code for Industrial Large Eddy Simulations"*, 8th Annual Conference of the CFD Society of Canada, Montreal, Canada (2000)
28. **C. CALVIN**, "*A development framework for parallel CFD applications: Trio-U project* ", 5th International Conference on Supercomputing for Nuclear Applications (SNA - 2003) ; Paris ; France (2003)
29. G. TRYGGVASON, A. ESMAEELI, J. LU, S. BISWAS, "*Direct numerical simulations of gas/liquid multiphase flows"*, Fluid Dynamics Research Journal, 38, 660–681 (2006)
30. E.LAUCOIN, *"Développement du parallélisme des méthodes numériques adaptatives pour un code industriel de simulation en mécanique des fluides."*, Ph Thesis Université J. Fourier, Grenoble, France (2007).
31. É. LAUCOIN AND **C. CALVIN**, "*A Parallel Front-Tracking Method for Two-Phase Flows Simulations."* Parallel Cfd (2004)
32. C. BERNARDI et F. HECHT, *"Mesh adaptivity in the mortar finite element method."*, 12th International Conference on Domain Decomposition Methods (2001).
33. C. BERNARDI et Y. MADAY, *"Mesh adaptivity in finite elements using the mortar method."*, Revue Européenne des Éléments Finis, 9:451–465 (2000).
34. K. SCHOEGEL, G. KARYPIS et V. KUMAR, "Parallel multilevel diffusion schemes for repartitioning of adaptive meshes." Rapport technique 97-014, Department of Computer Science, University of Minnesota, Minneapolis (1997).
35. **C. CALVIN** and D. NOWAK, "*High Performance Computing in Nuclear Engineering*", chapter in Handbook of Nuclear Engineering, Springer (2010).
36. J.P. BOTH, Y.K. LEE, A. MAZZOLO, Y. PENELIAU, O. PETIT and B. ROESSLINGER, *"TRIPOLI-4 : code de transport Monte Carlo - Fonctionnalités et applications.",* SFRP (2003).

37. J.C. TRAMA, *"Overview of TRIPOLI-4.5"*, in proceedings of ICRS Conference (2008).
38. F.X. HUGOT, Y.K. LEE and F. MALVAGI, *"Recent R&D around the Monte Carlo code TRIPOLI-4 for criticality calculations"*, in proceedings of PHYSOR Conference (2008).
39. JC TRAMA and FX HUGOT, *"TRIPOLI-4: parallelism capability"*, ANS Winter meeting (2007).
40. F. BROWN, J. GOORLEY and J. SWEEZY, *"MCNP5 parallel processing workshop"*, Workshop M&C 2003, http://mcnp-green.lanl.gov/publication/mcnp_publications.html (2003)
41. L. DENG and Z-S XIE, *"Parallelization of MCNP Monte Carlo Neutron and Photon Transport Code in Parallel Virtual Machine and Message Passing Interface"*, Journal of Nuclear Science and Technology, Vol. 36, No. 7 p.626-629 (1999)
42. T. GOORLEY, F. BROWN, and L. J. COX, *"MCNP5 improvements for Windows PCs"*, Nuclear Mathematical and Computational Sciences, Gatlinburg, Tennessee, April 6-11, 2003, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2003).
43. T. G. LEWIS and W. H. PAYNE, *"Generalized Feedback Shift Register Pseudorandom Number Algorithm"*, Journal of the ACM, Volume 20, Issue 3, pages 456-468 (1973).
44. P. HELLEKALEK, *"Don't trust parallel Monte Carlo!"*, ACM SIGSIM Simulation Digest archive, Volume 28 , Issue 1 (1998).
45. R. PROCASSINI, M. O'BRIEN and J. TAYLOR, *"Load Balancing Of Parallel Monte Carlo Transport Calculations"*, M&C Conference (2005).
46. P. ROMANO, B. FORGET, F. BROWN, *"Towards Scalable Parallelism in Monte Carlo Particle Transport Codes Using Remote Memory Access,"* Proc. SNA + MC2010, Tokyo, Japan, October 17-20 (2010).
47. **C. CALVIN**, *"HPC Challenges for Deterministic Neutronics Simulations Using APOLLO3® Code."* Progress in Nuclear Science and Technology Journal – 2 – 7007-705 (2011).
48. J.M. DO et al, *"Fuel loading pattern for heterogeneous EPR core configuration using a distributed evolutionary algorithm."*, M&C Conference (2009).
49. A.M. BAUDRON, J.J. LAUTARD and D. SCHNEIDER, *"Mixed Dual Methods for Neutronic Reactor Core Calculations in the CRONOS System"*, Proc. ANS Topical Mtg., Mathematical Methods and supercomputing in Nuclear Applications, Madrid, Spain (1999).
50. A.M. BAUDRON and J.J. LAUTARD, *"Minos: a SP_N Solver for Core Calculation in the Descartes System"*, Proc. ANS Topical Mtg., Mathematical Methods and supercomputing in Nuclear Applications, Avignon, France (2005).
51. H. GOLFIER et al, *"APOLLO3®: a common project of CEA, AREVA and EDF for the development of a new deterministic multi-purpose code for core physics analysis."*, M&C Conference (2009).
52. B. F. SMITH, P. BJORSTAD and W. GROPP, *"Domain Decomposition – Parallel Multilevel Methods for Elliptic Partial Differential Equations"*, Cambridge University Press (1996).
53. E. JAMELOT, J. DUBOIS, J-J. LAUTARD, **C. CALVIN**, A-M. BAUDRON, *"High Performance 3d Neutron Transport On Petascale And Hybrid Architectures Within Apollo3 Code"*, Conference M&C 2011, Rio de Janeiro, Brésil (2011)
54. M. ZEYAO and F. LIANXIANG, *"Parallel Flux Sweep Algorithm for Neutron Transport on Unstructured Grid"*, The Journal of Supercomputing, vol. 30 issue 1 (2004).
55. R. ROY and Z. STANKOVSKI, *"Parallelization of neutron transport solvers"*, in Recent Advances in Parallel Virtual Machine and Message Passing Interface, LCNS, pp 494-501, vol. 1332 (1997).

56. P. GUÉRIN, A-M BAUDRON and J-J. LAUTARD, *"Domain decomposition methods for core calculations using the MINOS solver."*, M&C Conference (2007)
57. P. GUÉRIN, A-M BAUDRON and J-J. LAUTARD, "*Component mode synthesis methods applied to 3D heterogeneous core calculations, using the mixed dual finite element solver MINOS.*", PHYSOR (2006)
58. G. E. SJODEN, "*PENTRAN: A parallel 3-D S(N) transport code with complete phase space decomposition, adaptive differencing, and iterative solution methods*", Thesis (PhD). THE PENNSYLVANIA STATE UNIVERSITY, Source DAI-B 58/05, p. 2652, 301 pages (1997)
59. J. RAGUSA, *"Application of Multithread Computing and Domain Decomposition to the 3-D Neutronics FEM Code CRONOS"*, International Conference on Supercomputing in Nuclear Applications, Paris, France (2003)
60. J. RAGUSA, *"Implementation of Multithreaded Computing in the Neutronics FEM Solver Minos"*, Proceedings of the ANS Mathematics and Computations International Conference, Gatlinburg, Tennesse (2003)
61. Z. STANKOVSKI, A. PUILL, L. DULLIER, *"Advanced plutonium assembly parallel calculations using the APOLLO2 code."*, M&C Conference (1997).
62. J.J. HERRERO, C. AHNERT, J.M ARAGONÉS, *"Spatial Domain Decomposition for LWR Cores at the Pin Scale."*, ANS Winter meeting (2007)
63. M. DAHMANI and R. ROY, *"Scalability Modeling For Deterministic Particle Transport Solvers"*, International Journal of High Performance Computing Applications, Vol. 20, No. 4, 541-556 (2006).
64. G. WU and R. ROY, *"Parallelization of Characteristics Solvers for 3D Neutron Transport"*, in Recent Advances in Parallel Virtual Machine and Message Passing Interface, LCNS, pp 344-351 (2001)
65. M. DAHMANI and R. ROY, *"Parallel solver based on the three-dimensional characteristics method: Design and performance analysis"*, Nuclear science and engineering (2005)
66. J. HAN GYU ET AL, *"Methods and performance of a three-dimensional whole-core transport code DeCART"*, PHYSOR (2004)
67. G. PALMIOTTI et al, *"UNÌC: Ultimate Neutronic Investigation Code."* M&C Conference (2007)
68. F. COULOMB, *"Parallelization of the DSN Multigroup Neutron Transport Equation on the CRAY-T3D using CRAFT",* Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, PPSC (1997)
69. **C. CALVIN**, S. PETITON, J. DUBOIS, AND E. JAMELOT, *"Iterative Eigenvalue Computation on multicore and hybrid architecture"*, 11th Copper Mountain conference on iterative methods, Copper Mountain, Colorado, USA (2010)
70. **C. CALVIN**, E. JAMELOT AND J. DUBOIS, "*GPGPU Programming to Solve the Boltzman Neutron Transport Equation.*" 14th SIAM Conference on Parallel Processing for Scientific Computing, USA (2010).
71. J. DUBOIS**,** *"Contribution à l'algorithmique et à la programmation efficace des nouvelles architectures parallèles comportant des accélérateurs de calcul dans le domaine de la neutronique et de la radioprotection",* thèse de l'Université de Lille 1 (2011)
72. J. DONGARRA ET AL., *"The International Exascale Software Project roadmap"*, International Journal of High Performance Computing Applications, vol. 25, no. 1, pp. 3-60 (2011).

73. S. L. GRAHAM, M. SNIR, AND C. A. PATTERSON eds., *"Getting Up to Speed: The Future of Supercomputing"*, National Academies Press, Washington, D.C. (2005).
74. L. GRIGORI, J. W. DEMMEL AND H. XIANG, "*Communication avoiding Gaussian elimination*", SC '08 Proceedings of the 2008 ACM/IEEE conference on Supercomputing (2008).
75. J. DEMMEL, L. GRIGORI, M. HOEMMEN AND J. LANGOU**,** "Communication-optimal Parallel and Sequential QR and LU Factorizations." SIAM J. Sci. Comput., 34(1), A206–A239 (2012).
76. F. CAPPELLO, *"Fault tolerance in petascale/exascale systems: current knowledge, challenges and research opportunities"*, International Journal of High Performance Computing Applications, vol. 23, no. 3, pp. 212–226 (2009).
77. F. CAPPELLO ET AL., "*Toward exascale resilience*", International Journal of High Performance Computing Applications, vol. 23, no. 4, pp. 374–388 (2009).
78. http://www.top500.org/
79. G. MEURANT. *"The Lanczos and Conjugate Gradient Algorithms: From Theory to Finite Precision Computations",* Software, Environments, and Tools. s.l. : SIAM (2006).
80. J. CULLUM AND R. A. WILLOUGHBY, *"Computing eigenvalues of very large symmetric matrices--An implementation of a Lanczos algorithm with no reorthogonalization.",* Journal of Computational Physics, Vol. 44, pp. 329-358 (1981)
81. G. H. GOLUB, R. R. UNDERWOOD, AND J. H. WILKINSON, *"The Lanczos Algorithm for the Symmetric Ax =Lambda Bx Problem."*, Stanford University. Stanford (1972).
82. W. E. ARNOLDI**, "***The principle of minimized iterations in the solution of the matrix eigenvalue problem."*, Quarterly of Applied Mathematics, Vol. 9, pp. 17-29. (1951)
83. L. GIRAUD, J. LANGOU, AND M. ROZLOZNIK, *"The loss of orthogonality in the Gram-Schmidt orthogonalization process."*, Comput. Math. Appl., Vol. 50, pp. 1069-1075. (2005)
84. H. SIMON AND W. KESHENG, *"Thick-Restart Lanczos Method for Large Symmetric Eigenvalue Problems.***"**, SIAM J. Matrix Anal. Appl., Vol. 22, pp. 602-616. (2000)
85. W. GREUB, *"Linear Algebra"*, Springer (1975). 30. Saad., Y. Iterative methods for sparse linear systems. New York : Publishing Company, 199-.
86. Y. SAAD, "*Iterative methods for sparse linear systems – 2nd edition"*, SIAM (2003).
87. Y. SAAD, *"Numerical solution of large nonsymetric eigenvalue problems."*, Comput. Phys. Commun (1989).
88. R. B. SORENSEN AND D. C LEHOUCQ, *"Deflation Techniques for an Implicitly Restarted Arnoldi Iteration"*, SIAM J. Matrix Anal. Appl., Vol. 17, pp. 789-821 (1996)
89. J. DUBOIS, S. PETITON, **C. CALVIN**, *"Krylov Subspace Computation on Hybrid Multicore Platforms"*, SIAM Annual meeting (2009).
90. J. DUBOIS, S. PETITON, **C. CALVIN**, *"Performance and Numerical Accuracy Evaluation of Heterogeneous Multicore Systems for Krylov Orthogonal Basis Computation "*, Conference VECPAR (2010).
91. G. H. GOLUB AND C. F. VAN LOAN. *"Matrix Computations",* Johns Hopkins Studies in Mathematical Science. The Johns Hopkins University Press (1996).
92. T. YOKOZAWA, D.TAKAHASHI, T. BOKU, AND M. SATO, "*Parallel implementation of classical gram-schmidt orthogonalization using matrix multiplication."* IPSJ SIG Technical Reports (2006).

93. L. GIRAUD, J. LANGOU, M. ROZLOZNIK, AND J. VAN DEN ESHOF, "*Rounding error analysis of the classical Gram-Schmidt orthogonalization process.*", Numerische Mathematik, 101(1):87–100 (2005).
94. B. GREER, *"The most important technical library in the world",* SIGPLAN Fortran Forum 17, New-York (1998).
95. Nvidia, *"CUDA Toolkit 4.2: CUBLAS Library"*. http://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/CUBLAS_Library.pdf (2012)
96. J. RUFINUS and Y. KORTSARTS, *"Domain Performance Study of Domain Decomposed Parallel Matrix-Vector Multiplication Programs",* International Conference on Computational Science and Education, Rochester (2006).
97. **C. CALVIN**, J. DUBOIS, *"Performances of Krylov Solvers for Reactor Physics Simulation on Petascale Architectures",* SIAM Conference on Computational Science and Engineering Boston, USA (2013).
98. **C. CALVIN**, S. PETITON, J. DUBOIS AND F. BOILLOD-CERNEUX, *"An Eigenvalue Solver using a Linear Algebra Framework for Multi-core and Accelerated Petascale Supercomputers",* 15th SIAM Conference on Parallel Processing for Scientific Computing, Savannah, USA (2012).
99. **C. CALVIN**, "*Some Lessons on Hybrid High Performance Computing in Reactor Physics*", SIAM ICIAM 2011, 7th International Congress on Industrial and Applied Mathematics, Vancouver, Canada (2011).
100. AGILENT TECHNOLOGIES, *"PCI Express Performance Measurements. ",* Application Note 1565. ", http://cp.literature.agilent.com/litweb/pdf/5989-4076EN.pdf (2005).
101. DELL, http://www.dell.com/downloads/global/power/ps1q10-20100215-Mellanox.pdf (2010).
102. S. PETITON, **C. CALVIN**, J. DUBOIS, L. DECOBERT, R. ABOUCHANE, P-Y. AQULIANTI, F. BOILLOD-CERNEUX, "*Krylov Subspace and Incomplete Orthogonalization Auto-tuning Algorithms for GMRES on GPU Accelerated Platforms",* 15th SIAM Conference on Parallel Processing for Scientific Computing, Savannah, USA (2012).
103. J. DUBOIS, **C. CALVIN** AND S. PETITON, *"Accelerating the explicitly restarted Arnoldi method with GPUs using an auto-tuned matrix vector product."* SIAM Journal on Scientific Computing, Volume 33, Issue 5, Pages 3010-3019 (2011)
104. J. DUBOIS, S. PETITON AND **C. CALVIN** "*Auto-Tuned Linear Algebra Computations for Krylov Methods on Multicore and GPUs*". SIAM Conference on Computational Science and Engineering, Reno, Nevada (2011)
105. Y. KUBOTA AND D. TAKAHASHI**,** *"Autotuning of Sparse Matrix-Vector Multiplication."***,** SIAM CSE (2011).
106. M. GARLAND AND N. BELL, *"Implementing sparse matrix-vector multiplication on throughput-oriented processors."*, Conference on High Performance Computing Networking, Storage and Analysis (SC '09) (2009).
107. E. AGULLO, ET AL., *"QR Factorization on a Multicore Node Enhanced with Multiple GPU Accelerators."*, University of Tennessee (2010).
108. M. HOEMMEN, "*Communication-Avoiding Krylov Subspace Methods.* University of California at Berkeley Thèse de doctorat. Advisor(s) James W. Demmel (2010).

109. V. HERNANDEZ, J.E. ROMAN and A. TOMAS, "Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement.", Parallel Computing, Vol. 33, pp. 521-540 (2007).

110. L. DRUMMOND, S. PETITON, **C. CALVIN**, *"Parametric Steering for Autotuning Numerical Kernels and Scientific Codes in Multicore Environments"*, SIAM Annual Meeting, Minneapolis, USA (2012).

111. **C. CALVIN**, L. DRUMMOND, F. BOILLOD-CERNEUX, J. DUBOIS, G. NDONGO EBOUM, "*Auto-tuning and Smart-tuning Approaches for Efficient Krylov Solvers on Petascale Architectures*", SIAM Conference on Computational Science and Engineering, Boston, USA (2013).

112. J-P BENZECRI, *« Histoire et Préhistoire de l'Analyse des données : Partie 5 »*, Les Cahiers de l'analyse des données, vol. 2, no 1, p. 9-40 (1977)

113. S. PETITON, **C. CALVIN**, J. DUBOIS AND F. BOILLOD-CERNEUX, *"Auto-tuned Hybrid Asynchronous Krylov iterative eigensolver on petascale computer."*, PMAA 2012, Londres, Royaume-Uni (2012).

114. **C. CALVIN**, N. EMAD, S. PETITON, J. DUBOIS AND M. DANDOUNA, *"MERAM for neutron physics applications using YML environment on post petascale heterogeneous architecture"*, SIAM Conference on Applied Linear Algebra (SIAM ALA 2012), Valence, Espagne (2012).

115. T. KATAGIRI, P-Y. AQUILANTI, S. PETITON, *"A Smart Tuning Strategy for Restart Frequency of GMRES(m) with Hierarchical Cache Sizes.",* Seventh International Workshop on Automatic Performance Tuning (iWAPT), RIKEN, Japan (2012).

116. J. DUBOIS, S. PETITON AND **C. CALVIN**, *"Improving Scalability with Asynchronous Hybrid Methods for non-Hermitian Eigenproblems"*, ICCS 2011, Tsukuba, Japan (2011).

117. **C. CALVIN**, "*Some Lessons on Hybrid High Performance Computing in Reactor Physics",* SIAM ICIAM 2011, 7th International Congress on Industrial and Applied Mathematics, Vancouver, Canada (2011).

118. N. EMAD, S. PETITON AND G. EDJLALI, *"Multiple Explicitly Restarted Arnoldi Method for Solving Large Eigenproblems.***"** SIAM J. Sci. Comput., Vol. 27, pp. 253-277 (2005).

119. N. EMAD, S-A. SHAHZADEH-FAZELI AND J. DONGARRA, "*An asynchronous algorithm on the NetSolve global computing system.",* Future Gener. Comput. Syst., Vol. 22, pp. 279-290 (2006).

120. A. SEDRAKIAN, « *Vers une aide à la décision pour les méthodes itératives hybrides parallèles réutilisables ».* Université Pierre et Marie Curie, Thèse de doctorat. (2005)

121. M. FLYNN, *"Some Computer Organizations and Their Effectiveness.",* IEEE Trans. Comput. C–21: 948 (1972).

122. M. DANDOUNA, "*Librairies numériques réutilisables pour le calcul distribué à grande échelle. »* Thèse de doctorat, Université de Versailles-St Quentin (2012).

123. N. EMAD, M. DANDOUNA, L. A. DRUMMOND, "*A Proposed Programming Model for Writing Sustainable Numerical Libraries for Extreme Scale Computing"* , Submitted on Concurrency and Computation Practice and Experience Journal (CCPE) (2013).

124. L. DRUMMOND, N. EMAD, M. DANDOUNA, *"In Search of A More Sustainable Approach to Implement Scalable Numerical Kernels"*, SIAM Conference on Parallel Processing for Scientic Computing, Savannah, Georgia, USA, February 15-17 (2012).

125. M. DANDOUNA, N. EMAD, L. DRUMMOND, *"Design and Development of Sustainable Libraries for Numerical Computation on Many-core Architectures."*, SIAM Annual Meeting, Valencia, Spain, June 18-22 (2012).

126. THE TRILINOS PROJECT. http://trilinos.sandia.gov/index.html