Calcul parallèle et méthodes numériques pour la simulation de plasmas de bords

Matthieu Kuhn

UNIVERSITÉ DE STRASBOURG

msii
ÉCOLE DOCTORALE

N° d'ordre :

École Doctorale Mathématiques, Sciences de l'Information et de l'Ingénieur

THÈSE

présentée pour obtenir le grade de

Docteur de l'Université de Strasbourg

Discipline : Informatique

par

Matthieu Kuhn

Calcul parallèle et méthodes numériques pour la simulation de plasmas de bords

Soutenue publiquement le 29 Septembre 2014

Membres du jury :

Directeur de thèse :	Nicolas Crouseilles, Chargé de Recherches, INRIA, Rennes
Co-directeur :	Stéphane Genaud, Professeur , ENSIIE, Strasbourg
Rapporteur :	Jean-François Méhaut, Professeur, UdG, Grenoble
Rapporteur :	Eric Sonnendrücker, Professor, IPP, Garching
Examinateur :	Philippe Clauss, Professeur, UdS, Strasbourg
Examinateur :	Guillaume Fuhr, Maître de Conférences, AMU, Marseille

Laboratoire ICube, UMR 7357 **INRIA Rennes**

Résumé

La fusion nucléaire contrôlée présente des atouts considérables, mais sa mise en oeuvre nécessite l'aboutissement d'un programme de recherches en amont de la phase industrielle. Notamment, une meilleure connaissance des phénomènes de transport et de turbulence qui ont lieu dans le plasma de bord d'un tokamak est importante pour la qualité du confinement et donc pour les performances de la machine. Dans ce cadre, la simulation des plasmas de bords de tokamak représente un enjeu majeur. La compréhension plus fine des phénomènes physiques apparaissant sur des temps longs (ELMs, barrières de transport par exemple) et à des échelles spatiales petites (îlots magnétiques) nécessite des simulations dont les temps de calculs sont prohibitifs, typiquement de l'ordre de plusieurs mois. L'objectif de cette thèse est de contribuer à l'amélioration des schémas numériques d'un code de bord électromagnétique d'une part et d'accroître ses performances sur des machines de calcul informatiques d'autre part. Le code sur lequel s'appuie nos travaux est le code Emedge3D.

Du point de vue des schémas numériques, Emedge3D s'apparente aux problèmes généraux d'advection-diffusion. Ceux-ci posent le problème de forte condition de stabilité sur le pas de temps dans le cas de méthodes de type explicite. De plus, dans le modèle de tokamak considéré, la diffusion est anisotrope : elle est plus forte dans la direction parallèle aux lignes de champs que dans la direction perpendiculaire. Un des résultat de cette thèse est une maîtrise plus fine de l'erreur produite sur cet opérateur anisotrope. Concernant l'aspect coût de calcul, la limitation de la bande passante mémoire représente un des points bloquants pour cette application. La double discrétisation (variables réelle et Fourier) ainsi que la prise en compte des 3 dimensions spatiales sont les principales raisons de ce goulot d'étranglement sur la mémoire. Enfin, le traitement des termes non linéaires est effectué par la méthode de référence de Arakawa qui représente une charge de calcul prépondérante au sein des simulations.

Dans cette thèse, l'amélioration du code Emedge3D est abordée sous plusieurs axes. Concernant le premier axe, nous apportons tout d'abord des innovations sur les méthodes numériques. Au niveau schéma temporel, nous montrons l'avantage qu'offrent les méthodes de type semi-implicites, dont la stabilité inconditionnelle permet l'augmentation de la valeur du pas de temps,

et donc la diminution du nombre d'itérations temporelles requises pour effectuer une simulation. En particulier, une nouvelle classe de méthodes semi-implicite est proposée et confrontée aux méthodes récentes sur plusieurs cas tests. Ensuite, nous mettons en évidence l'importance de la montée en ordre en espace et en temps pour ces schémas semi-implicites. Cela permet en outre d'assurer le contrôle de l'erreur supplémentaire provoquée par l'augmentation du pas de temps.

Comme deuxième axe, nous proposons des réponses quant à la mise en place d'algorithmes parallèles dans le code Emedge3D. Classiquement, les parties coûteuses du code ont tout d'abord été optimisées séquentiellement. Ensuite, une parallélisation a été conçue en visant premièrement une architecture à mémoire partagée. Des solutions sont proposées pour surmonter les problèmes de limitation de bande passante mémoire. Néanmoins, pour la partie du code la plus sensible aux contraintes de bande passante mémoire, une étude de parallélisation sur machine à mémoire distribuée est présentée. En effet, l'ajout de nœuds de calcul permet d'augmenter la bande passante mémoire totale. Cette dernière étude est effectuée dans le cadre d'un problème plus général d'advection-diffusion non linéaire, très proche de la structure mathématique des équations résolues dans le code Emedge3D.

Cette thèse s'inscrit dans le projet interdisciplinaire ANR E2T2 qui porte, entre autres, sur le code Emedge3D. Ce code est développé depuis plusieurs années au sein du laboratoire de Physique du PIIM (Aix-Marseille Université).

Mots clés : fusion nucléaire, physique des plasmas de bord, simulation numérique, advection-diffusion, diffusion anisotrope, schémas temporels semi-implicite, discrétisation semi-spectrale, méthode d'Arakawa, calcul haute performance, parallélisation, mémoire partagée, mémoire distribuée, bande passante mémoire limitée.

Abstract

Controlled thermonuclear fusion has several advantages, nevertheless simulations are required in order to reach an industrial scale. In particular, a better knowledge of turbulent transport taking place at the edges of tokamaks is important to ensure the confinement quality, and hence the reactor performances. On this issue, plasma boundary simulations represent a major stake. To adress characteristic physical phenomena appearing at large time scales (as those of the ELMs or the transport barriers) and at a relatively small spatial case (magnetic islands), high computational cost are induced. Typical simulations can require several months on a single processing unit. Tools with such settings are not really easily usable. The code that has been the main target is Emedge3D. This PhD thesis proposes improvements of some numerical schemes to shorten time to solution for the physicists that use this code, and also describes parallel algorithms to use multiple processing units.

On the numerical scheme side, Emedge3D is close to the classical advection-diffusion problem. Using standard solutions and explicit integration time methods, a hard stability condition is imposed on the time step. Furthermore, in the kind of models that we consider, the diffusion operator is anisotropic : it is stronger in the parallel direction to the magnetic field lines (compared to to the perpendicular direction). One result of this work is a solution to better control the error produced by the anisotropic diffusion solver. Regarding the computational cost side, we look at the memory bandwidth limitation, partly caused by the switching between two 3D spatial discretizations : real and Fourier variables. Finally, non linear operators (Poisson brackets) are computed with a well known method (Arakawa), whose stencil implies a high computational cost.

The improvements of Emedge3D are along two axes. On the first hand, we bring innovations concerning the numerical methods used. Concerning the time integration scheme, we show what are the benefits of semi-implicit methods. Their unconditional stable property allows one to consider larger time step values, diminishing the time needed to perform simulations. In particular, a new class of semi-implicit method is described and compared to several other semi-implicit methods on different test cases. Then, we show the importance

of using high order methods in time and in space. This allows one to control the additional error produced by the larger time step.

On the second hand, we propose solutions for the parallelization of the Emedge3D code. A first parallelization has been done targeting shared memory architectures. A memory bandwidth limitation is observed, techniques are shown to overcome it. However, for one part of the code, the performance is not satisfactory enough and also, the number of processing units available in one shared memory node remains usually low. Hence, a parallelization study on distributed memory architecture is presented. Adding computational nodes allows one to increase the total memory bandwidth and computing units. This last study is performed on the more general non linear advection-diffusion problem, very close to the mathematical structure of the Emedge3D equations.

This PhD thesis is part of an interdisciplinary ANR project, named E2T2, that operates on the Emedge3D code and other related topics. This code is developed for several years in the PIIM physics laboratory (Aix-Marseille University).

Keywords : nuclear fusion, boundary plasma physics, numerical simulation, advection-diffusion, anisotropic diffusion, semi-implicit time scheme, semi-spectral, Arakawa scheme, high performance computing, parallelization, shared memory architecture, distributed memory architecture, memory bound programs.

Remerciements

Dans un premier temps, je tiens à remercier les personnes qui ont été au plus proche de ce travail de thèse, qui m'ont beaucoup épaulé, énormément appris, mais également orienté, rassuré, cadré, encadré, recadré, et surtout accompagné le long de ces trois années et demi de travaux. Je veux parler ici de mes trois géniaux encadrants, Nicolas Crouseilles, Stéphane Genaud et Guillaume Latu, merci.

Je remercie ensuite Jean François Méhaut et Eric Sonnendrücker, pour leurs lectures attentives, leurs remarques et leur présence au sein de mon jury, pour le temps qu'ils y ont consacré, pour l'honneur qu'ils m'ont fait en rapportant ce travail de thèse.

Je dois émettre des remerciements spéciaux à Philippe Clauss, en temps que membre de mon jury, collaborateur, pour avoir été un super "chef" pendant mes trois ans à Strasbourg. Merci aussi pour ton soutien psychologique et logistique. Je remercie Alain Ketterlin, pour toutes les discussions intéressantes qu'on a pu avoir sur nos projets respectifs et pour m'avoir appris pleins de choses.

Je remercie également Guillaume Fuhr en tant que membre examinateur de mon jury, mais aussi pour toute l'aide qu'il a pu m'apporter au cours de cette thèse. Merci à Peter Beyer, coordinateur du projet ANR E2T2 dont ma thèse fait partie. Merci à Arnaud Monnier pour les échanges que nous avons pu avoir au sein de ce projet. Merci à Patrick Tamain pour nous avoir apporté son expertise ainsi que son regard extérieur pour certains travaux de ce manuscrit.

De manière plus générale, merci à toute l'équipe ICPS qui m'a accueilli, pour avoir partagé tant de moments agréables, autant d'un point de vu professionnel que personnel : merci à Vincent, Eric, Julien, Jens, Cédric et Romaric, à Alexandre, Alexandra, Aravind, Benoît, Etienne, Jean-François, Juan et Willy. Merci à Jean Christophe Beyler, que j'ai croisé un peu plus tôt dans mon parcours, alors thésard dans cette équipe, pour avoir su me transmettre son goût pour l'informatique.

Merci à toi aussi Céline Caldini-Queiros, pour tous les moments où l'on se croise, et pour avoir partagé ton expérience de thèse, ayant eu un peu d'avance

sur la mienne.

Merci à l'Agence Nationale pour la Recherche, pour avoir financé le projet ANR E2T2 dont ma thèse fait partie. Tout cet argent m'a permis de bien manger et bien boire en Alsace pendant 3 ans. Merci à Erwan Faou et à l'ERC pour m'avoir permis de prolonger ma thèse au sein de l'équipe INRIA IPSO pendant les six derniers mois, et donc de bien manger et de bien boire en Bretagne aussi.

Je tiens à remercier tous les gens que j'ai côtoyés au long de ce travail. A mes amis d'enfance, pour toutes ces soirées. A mes amis de Swing and Feelings, pour toutes ces notes. A mes amis de Soufflenheim, pour tous ces accords. Avec vous, j'ai pu partager, grandir, évoluer, et surtout bien me marrer.

Je remercie ma famille pour tout le soutien qu'elle m'a apporté et pour avoir cru en moi. Merci particulièrement à Damien, et, bien sûr, à mes merveilleux parents.

Enfin, je souhaite remercier Morgane pour notre belle rencontre. Pour m'avoir redonné confiance, pour avoir égayé tant de moments dans la fin de ce parcours, pour m'avoir guidé dans cette nouvelle vie bretonne, pour tous nos projets. Merci.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

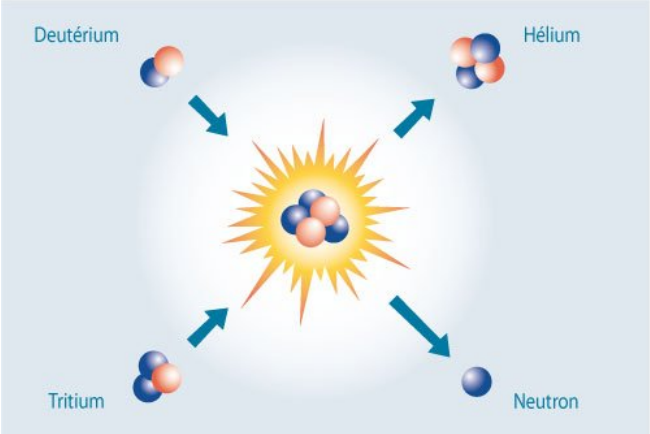
L'énergie représente actuellement l'un des enjeux majeurs de notre société. Les énergies fossiles étant non renouvelables, l'étude des énergies nucléaires présente un intérêt non négligeable. Il existe deux types de réactions nucléaires produisant de l'énergie. La première est appelée réaction de fission. Elle consiste en la génération de deux atomes légers à partir d'un atome lourd, généralement de l'uranium ou du plutonium. Cette réaction libère également de l'énergie, dont il est possible de se servir pour générer de l'énergie électrique. C'est la réaction nucléaire qui s'opère dans les centrales nucléaires actuelles.

La seconde est appelée réaction de fusion nucléaire. C'est le principe inverse de la fission : partant de deux atomes légers, cette réaction produit un atome de noyau plus lourd tout en libérant de l'énergie. Ce type de réaction a lieu au sein du soleil, et plus généralement au sein des étoiles.

La fusion présente trois grands avantages par rapport à la fission. Premièrement, elle permet de créer 3 à 4 fois plus d'énergie, à quantité égale de carburant. Ensuite, cette réaction a aussi la propriété d'être propre, au sens que son produit n'est pas radioactif. Enfin, le carburant nécessaire à la réaction de fusion se trouve plus facilement et en plus grande quantité que l'uranium. C'est pourquoi cette technique de production d'énergie fait actuellement l'objet de nombreux travaux exploratoires.

La réaction de fusion la plus simple à mettre en œuvre consiste à prendre des atomes de Deutérium et de Tritium, isotopes de l'hydrogène, pour produire un atome d'Hélium et un neutron chargé en énergie (voir Figure 1). Cette réaction s'effectue au sein de plasmas, qui ne sont autre qu'un quatrième état de la matière, au coté des solides, liquides et gaz. Ils sont obtenus en chauffant un gaz à très haute température. Le mélange ainsi obtenu est de charge neutre, comportant les noyaux des atomes desquels se sont détachés les ions et électrons. Une méthode consiste à confiner ce plasma suffisamment longtemps pour pouvoir obtenir des réactions nucléaires.

FIGURE 1.1 – Réaction nucléaire par fusion de Deutérium et de Tritium



Maintenir le plasma confiné à très haute température pose un problème technologique. La réponse apportée dans la fusion par confinement magnétique est d'imposer un champ magnétique intense pour confiner le plasma dans une enceinte. Le courant électrique ainsi obtenu permet alors de maintenir le plasma à haute température, même si des techniques de chauffage sont nécessaires (chauffage par onde ou par ions lourds). Notons qu'une autre approche existe : la fusion par confinement inertiel permet aussi d'accéder aux mécanismes de fusion à l'aide de faisceaux laser intense déposant leur énergie sur une bille de Deuterium-Tritium très dense.

Actuellement, la fusion nucléaire par confinement magnétique en est à un stade expérimental. Les travaux de recherche sont menés sur plusieurs fronts. Un premier front concerne les expériences physiques, permettant de faire des essais dans des tokamaks et d'observer le comportement du plasma en temps réel. Un second axe est la simulation numérique, dont le but est de simuler par ordinateur une expérience pour la fusion nucléaire. La simulation numérique présente le double avantage de limiter les risques et le coût par rapport à une expérience physique. Elle permet par exemple d'observer des phénomènes physiques peu compris, comme les phénomènes de bords dans les tokamaks. C'est justement l'objet du code de simulation pour les plasmas de bord Emedge3D sur lequel porte ce travail de thèse. Cette thèse s'inscrit dans une collaboration inter-disciplinaire regroupant physiciens, mathématiciens et informaticiens, dans le cadre du projet ANR (Agence Nationale de la Recherche) au titre du programme Investissements d'Avenir portant la référence ANR-10-BLAN-0940.

Dans la suite de ce chapitre, nous allons tout d'abord présenter des modèles

pour la fusion nucléaire. Ensuite, nous aborderons les méthodes numériques que nous appliquerons au modèle de bord d'Emedge3D. Puis, nous introduirons le calcul haute performance dans lequel il sera question d'algorithmique et de programmation parallèle. Enfin, nous détaillerons le cadre ainsi que les objectifs de ce travail de thèse.

Sommaire

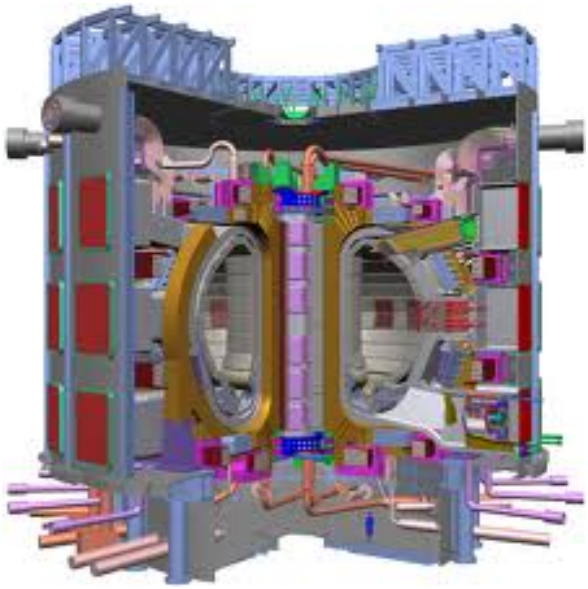
1.1 La fusion nucléaire et ses modèles

Dans cette section, nous introduisons la fusion nucléaire et des modèles physiques qui lui sont associés de manière générale. Pour une description plus détaillée, nous renvoyons le lecteur aux références [10, 14, 51, 31, 32].

1.1.1 La fusion nucléaire par confinement magnétique

Le confinement magnétique consiste à contrôler le plasma à l'aide d'un champ magnétique pendant un temps assez long pour augmenter les probabilités de fusion, augmentant par la même occasion l'énergie produite. Ce confinement prend place dans un dispositif torique, cœur du réacteur que l'on appelle Tokamak (voir Figure 1.1.1). Ce mode de confinement est actuellement au centre de l'étude menée pour le projet ITER (International Thermonuclear Experimental Reactor), actuellement en cours de construction à coté du centre

FIGURE 1.2 – Représentation du tokamak ITER, Cadarache, France



du CEA (Commissariat à l'Energie Atomique) sur le site de Cadarache (voir [50] et www.iter.org).

Lors de la mise en route du réacteur, le gaz est chauffé pour obtenir un plasma. Les particules présentes dans ce plasma se mettent ensuite à bouger dans la chambre toroïdale. Ces particules sont alors soumises à une force centrifuge, qui les attire vers les bords du réacteur. Pour éviter qu'elles ne l'atteignent, sous peine d'infliger des dégâts aux parois, on impose un champ magnétique intense pour les piéger autour de ce que l'on appelle des lignes de champs. Les particules parcourent alors des cercles qui "s'enroulent" autour de ces lignes d'un rayon plus ou moins petit, en fonction de l'intensité du champ magnétique imposé et de l'énergie portée par les particules. Plus ce champ sera fort, et mieux les particules seront piégées, assurant le confinement du plasma dans la chambre du tokamak.

Malgré cette stratégie de confinement, les plasmas sont encore peu compris car ils mettent en jeu des phénomènes non linéaires à des échelles de temps et d'espace multiples. Certains phénomènes le rendent sujet à des instabilités de plusieurs types et empêchent d'obtenir un confinement assez long pour obtenir une réaction de fusion rentable, dans le sens où l'énergie produite reste inférieure à l'énergie introduite pour provoquer la réaction. D'autres peuvent avoir des conséquences plus graves, comme la disruption. Elle provoque une perte du confinement du plasma, pouvant aller jusqu'à provoquer des dégâts

matériels sur le tokamak. Pour mieux comprendre ces phénomènes, il existe différents modèles physiques qui permettent de reproduire les processus physiques mis en jeu, et cela au sein de simulations numériques réalisées sur ordinateur.

1.1.2 Les modèles pour la fusion

Nous allons introduire ici différents modèles physiques pour l'étude de la fusion par confinement magnétique. Dans la suite, la notation ∂_d désignera la dérivée partielle par rapport à la direction d, soit $\partial_d = \frac{\partial}{\partial d}$. Nous aborderons les modèles cinétique et fluide avant d'introduire le modèle de bord utilisé dans Emedge3D.

Le modèle cinétique (K)

Les équations pour le modèle cinétique sont utilisées pour décrire l'évolution des particules chargées à l'intérieur d'un plasma. L'inconnue est une fonction f, appelée fonction de distribution, dépendant du temps t, de l'espace $x \in \mathbb{R}^3$ et de la vitesse $v \in \mathbb{R}^3$. On appelle (x, v) l'espace des phases et $f_s(t, x, v)$ représente la densité de particules s dans un volume élémentaire de l'espace des phases centré autour du point (x, v) au temps t. On parle ici de modèle microscopique. Pour plus de lisibilité, les équation seront présentées sous forme normalisée, d'où l'absence des constantes usuelles (masse et charge des particules, etc). Pour les mêmes raisons, nous faisons abstraction du type de particule considéré.

Le modèle qui gouverne l'évolution de cette fonction de distribution f s'appelle l'équation de Vlasov et s'écrit :

$$\partial_t f + v \cdot \partial_x f + (E + v \times B) \cdot \partial_v f = Q(f, f), \tag{1.1.1}$$

avec (E, B) le champ électromagnétique et Q un opérateur de collision (par exemple l'opérateur de Fokker-Planck-Landau). Elle a la structure d'une équation de transport dans l'espace des phases qui comprend les trois dimensions de l'espace physique et les trois dimensions de l'espace des vitesses.

Il est possible de considérer les champs électromagnétiques de manière auto-consistante, *i.e.* de calculer les champs créés par les particules chargées elles-mêmes. Pour cela, le champ électromagnétique auto-consistant peut être calculé grâce a un couplage avec l'équation de Maxwell dont les sources sont déterminées à partir de f. Les équations de Maxwell s'écrivent :

$$\begin{aligned} -\partial_t E + \mathrm{rot}(B) &= J, \\ \partial_t B + \mathrm{rot}(E) &= 0, \\ \mathrm{div}(E) &= \rho, \\ \mathrm{div}(B) &= 0, \end{aligned}$$

avec la densité de particules $\rho(x,t) = \int f(x,v,t)dv$, la densité de courant $J = \int f(x,v,t)vdv$, $\mathrm{rot}(E) = (\partial_{x_2}E_2 - \partial_{x_3}E_3, \partial_{x_3}E_3 - \partial_{x_1}E_1, \partial_{x_1}E_1 - \partial_{x_2}E_2)$ et $\mathrm{div}(E) = \partial_{x_1}E_1 + \partial_{x_2}E_2 + \partial_{x_3}E_3$ avec $x = (x_1, x_2, x_3)$ et $E = (E_1, E_2, E_3)$.

Si l'on néglige les effets magnétiques auto-consistants, il est aussi possible remplacer l'équation de Maxwell par l'équation de Poisson :

$$\begin{aligned} \mathrm{div}(E) &= \rho, \\ E &= -\partial_x \phi, \end{aligned}$$

où ϕ désigne le potentiel électrostatique.

Ce modèle, bien qu'assez réaliste au niveau des informations qu'il permet d'obtenir, s'avère vite très coûteux en terme de simulation. En effet, pour effectuer des simulations en tenant compte de la taille des structures physiques pertinentes, il faudrait stocker un grand volume de données, étant donné que f dépend de 6 dimensions (espace et vitesse) et du temps.

Ainsi, on envisage des modèles réduits qui peuvent être vus comme des simplifications de modèle cinétique de type Vlasov. On appelle ces modèles réduits des modèles fluides ou hydrodynamiques.

Le modèle fluide

Une formulation fluide pour la dynamique des plasmas s'obtient à partir de la formulation cinétique, en partant de l'Equation (1.1.1) couplée aux équations de Maxwell ou Poisson. Les inconnues sont des quantités qualifiées de macroscopiques qui peuvent être interprétées en terme de moyenne en vitesse de la fonction de distribution f. Ainsi, en considérant les premiers moments en vitesse de l'équation de Vlasov, on obtient un système d'équations satisfait par la densité $n(t,x)$, vitesse moyenne $u(t,x)$ et température $T(t,x)$. On se propose de détailler ces calculs dans le cas simplifié unidimensionnel électrostatique, *i.e.* $x \in \mathbb{R}$, $v \in \mathbb{R}$ et $B = 0$ (pour plus de détails, voir [54]). Posons tout d'abord :

$$m(v) = \begin{pmatrix} 1 \\ v \\ \frac{v^2}{2} \end{pmatrix}, \text{donnant } \partial_v m = \begin{pmatrix} 0 \\ 1 \\ v \end{pmatrix}$$

On définit ensuite $U(t,x)$ comme le vecteur des trois premiers moments en vitesse de $f(t,x,v)$:

$$\int_{\mathbb{R}} m(v) f(t,x,v) dv = U(t,x) = \begin{pmatrix} U_1(t,x) \\ U_2(t,x) \\ U_3(t,x) \end{pmatrix} = \begin{pmatrix} n \\ nu \\ \frac{1}{2}n(T+u^2) \end{pmatrix} \qquad (1.1.2)$$

Afin d'obtenir les équations pour le modèle fluide, on intègre en vitesse les équations de Vlasov (1.1.1) contre $m(v)$. Ces intégrales donnent, en considérant

un opérateur de collision Q qui préserve la masse, l'impulsion et l'énergie :

$$\begin{aligned} & \partial_t \int_{\mathbb{R}} m(v) f_s dv + \partial_x \int_{\mathbb{R}} m(v) v f_s dv + \int_{\mathbb{R}} m(v) E \partial_v f_s dv = 0, \\ \Leftrightarrow \quad & \partial_t \begin{pmatrix} U_1 \\ U_2 \\ U_3 \end{pmatrix} + \partial_x \begin{pmatrix} U_2 \\ 2U_3 \\ \int \frac{v^3}{2} f_s(v) dv \end{pmatrix} - \begin{pmatrix} 0 \\ EU_1 \\ EU_2 \end{pmatrix} = 0. \end{aligned} \tag{1.1.3}$$

On remarque que la troisième composante de l'Equation (1.1.3) fait intervenir un moment d'ordre supérieur (ordre 4 ici). On pourrait alors l'ajouter à nos calculs, mais l'ajout de cette nouvelle expression ne ferait qu'apparaître un moment d'ordre supérieur. Il faut donc ajouter une information supplémentaire afin d'avoir autant d'équations que d'inconnues pour pouvoir résoudre ce système. On appelle cet élément supplémentaire la *relation de fermeture*.

Comme relation de fermeture, il est commun d'utiliser la propriété physique qu'à l'équilibre thermodynamique, la fonction de distribution des particules est proche d'une Maxwellienne, que nous noterons $f_M(t, x, v)$ et qui peut s'exprimer en fonction des inconnues du problème fluide n, u et T la température du fluide :

$$f_M(t, x, v) = \frac{n}{(2\pi T)^{\frac{1}{2}}} \exp\left(-\frac{(v-u)^2}{2T}\right), \tag{1.1.4}$$

dont les trois premiers moments sont justement ceux de f :

$$\begin{aligned} \int_{\mathbb{R}} f_M(t, x, v) dv &= n, \\ \int_{\mathbb{R}} f_M(t, x, v) v dv &= nu, \\ \int_{\mathbb{R}} f_M(t, x, v) \frac{v^2}{2} dv &= \frac{nT}{2} + \frac{nu^2}{2}. \end{aligned}$$

Bien que non explicités plus haut, les trois premiers moments sont consistants avec la définition des grandeurs n, u et T à partir d'une fonction de distribution quelconque en inversant la relation (1.1.2) :

$$n = U_1, u = \frac{U_2}{U_1}, T = \frac{2U_1U_3 - U_2^2}{U_1^2}$$

Avec l'approximation $f \approx f_M$, nous sommes à présent en mesure de fermer le système et de continuer les calculs (voir [54]).

Les équations pour le modèle fluide s'écrivent :

$$\partial_t n + \partial_x(nu) = 0, \tag{1.1.5}$$

$$\partial_t(nu) + \partial_x(n(T + u^2)) = nE, \tag{1.1.6}$$

$$\partial_t(n(T + u^2)) + \partial_x\left(nu(3T + u^2)\right) = 2Enu, \tag{1.1.7}$$

Ces équations doivent encore une fois être couplées avec l'équation de Poisson pour le calcul du champ électromagnétique auto-consistant. Tout ces calculs se généralisent dans le cas multidimensionnel et / ou d'un couplage avec les équations de Maxwell.

Notons enfin qu'au prix d'une hypothèse Maxwellienne sur la distribution en vitesse des particules, il s'agit ici de résoudre un système pour des champs 3D en espace, contre la résolution d'un système 3D en espace et 3D en vitesse (soit 6 dimensions spatiales en tout) pour le modèle cinétique. Cette approximation est en particulier valide dans le cas de plasmas fortement collisionnels. Par contre, lorsque les collisions sont peu présentes, la fonction de distribution reste loin de l'équilibre f_M et cette approximation n'est pas justifiée. Néanmoins, les modèles fluides restent très utilisés dans la description des phénomènes physiques dans les tokamaks car leur résolution numérique est moins coûteuse, impliquant moins de données à stocker et à traiter.

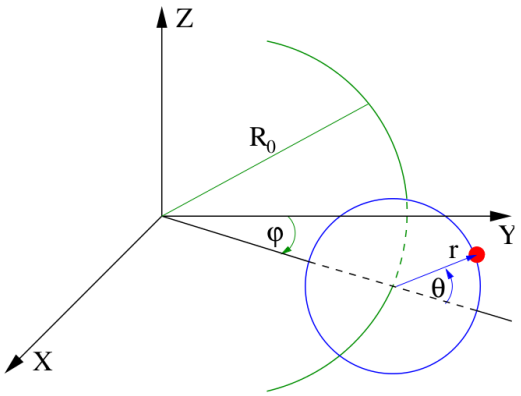
1.1.3 Les plasmas de bords

Dans cette sous section, nous allons nous intéresser à un modèle spécifique modélisant ce qui se passe au bord des tokamaks. En effet, certaines simulations sont en désaccord avec les expériences physiques. Les temps de confinement observés par simulation avec des modèles présentés en Sous Section 1.1.2 sont de l'ordre de 10 secondes, alors que les temps observés lors d'expériences réelles sont de l'ordre de la seconde (voir [24]), ce qui donne un ordre de grandeur de différence entre la simulation et l'expérimentation réelle. Ceci signifie que décrire les phénomènes physiques par les modèles présentés précédemment est insuffisant. On parle alors de transport anormal / turbulent, dans le sens où les pertes d'énergies ne sont pas uniquement dues aux collisions entre particules.

Il est possible de diminuer la turbulence du plasma et d'améliorer son confinement en augmentant les gradients de densité et de pression pour constituer ce qui s'appelle une barrière de transport. Mais ceci fait aussi apparaître un autre type d'instabilité localisée au bord (par exemple les Edge Localized Modes ou la turbulence de bord). Ces modes provoquent une dégradation de la barrière, laissant brutalement passer les particules et l'énergie qu'elles transportent vers la paroi du tokamak. L'énergie transportée peut alors être transmise à la paroi et potentiellement causer des dégâts. On parle de relaxation de la barrière de transport.

Les modèles fluides présentés en Sous Section 1.1.2 sont trop coûteux pour accéder aux résolutions temporelle et spatiale permettant de reproduire ces instabilités. Il faut donc passer par des modèles plus dédiés à ces phénomènes. C'est l'objectif du modèle utilisé dans le code Emedge3D, qui a pour but la simulation du bord des tokamaks.

FIGURE 1.3 – Système de coordonnées toroïdales (issu de [24])



Avant d'écrire le modèle de bord pour Emedge3D, nous allons présenter la géométrie du tokamak.

La géométrie du tokamak

Afin de décrire la géométrie torique du tokamak, il est nécessaire de choisir un système de coordonnées approprié, appelé système de coordonnées toroïdales :

$$X = R\sin\varphi, Y = R\cos\varphi, Z = R\sin\theta \text{ avec } R = R_0 + r\cos\theta$$

où R_0 est le rayon moyen du tokamak (jusqu'au centre de la chambre toroïdale), φ est l'angle dans la direction toroïdale, θ l'angle dans la direction poloïdale et r le petit rayon du tore. Le schéma en Figure 1.3 illustre ce système de coordonnées.

Pour représenter les données et permettre une discrétisation spatiale plus aisée, on simplifie la géométrie : on assimile le tore à un parallélépipède par homéomorphisme. On prend alors comme nouvelles coordonnées (x, y, z), qui correspondent respectivement à (r, θ, φ). Pour "refermer" le parallélépipède sur lui même tel le tore de départ, on applique des conditions de bord périodiques dans les directions y et z. Des conditions de bord de Neumann sont appliquées dans la direction radiale. Cette géométrie est couramment appelée géométrie SLAB.

Modèle fluide pour les plasmas de bord

Le modèle fluide pour les plasmas de bord simulés par le code Emedge3D découle du modèle fluide présenté en Sous Section 1.1.2. Son obtention étant assez technique, nous renvoyons le lecteur à [5] et [24]. Les inconnues sont le potentiel électrostatique ϕ, le potentiel électromagnétique ψ et la pression p. Les crochets de Poisson sont définis comme suit

$$\{f, g\} = \partial_x f \partial_y g - \partial_x g \partial_y f,$$

ainsi que le gradient parallèle $\nabla_\parallel$

$$\nabla_\parallel f = \partial_z f - \{\psi + \psi_0, f\},$$

et le terme de courbure

$$Gf = \sin(y)\partial_x f + \cos(y)\partial_y f.$$

Le modèle s'écrit alors

$$\begin{aligned}
\partial_t(\nabla_\perp^2\phi) + \{\phi, \nabla_\perp^2\phi\} &= -C_1\nabla_\parallel(\nabla_\perp^2\psi + \nabla_\perp^2\psi_0) + \nabla_\perp \cdot (\nu(x)\nabla_\perp(\nabla_\perp^2\phi)) \\
&\quad -Gp + \mu(\nabla_\perp^2\phi - \nabla_\perp^2\phi_0) \qquad (1.1.8) \\
\partial_t p + \{\phi, p\} &= C_2\nabla_\parallel(\nabla_\perp^2\psi + \nabla_\perp^2\psi_0) + G(\delta_c\phi - \Gamma p) \\
&\quad +\nabla_\perp \cdot (\xi_\perp(x,y)\nabla_\perp p) + \xi_\parallel(z)\nabla_\parallel^2 p + S(x) \qquad (1.1.9) \\
\partial_t\psi &= -\nabla_\parallel(\phi - \Gamma p) + \eta\nabla_\perp^2\psi + \lambda(x)(\tilde{\psi} - \psi). \qquad (1.1.10)
\end{aligned}$$

Dans ce modèle, les quantités avec des indices $(\psi_0, \tilde{\psi}, \phi_{imp}, \phi_0)$ ainsi que $\nu(x), \lambda(x), \xi_\perp(x,y), \xi_\parallel(z), S(x)$ sont des fonctions imposées. Les autres quantités sont des constantes $(C_1, C_2, \mu, \delta_c, \Gamma, \eta)$.

Ce modèle a été développé par les physiciens de Aix-Marseille Université (laboratoire PIIM), et notamment par Peter Beyer qui le décrit dans [5]. Il est aussi l'objet de la thèse de Guillaume Fuhr (voir [24]), et il est au centre du projet ANR E2T2 dans lequel s'inscrit cette thèse.

1.2 Introduction aux méthodes numériques

Les expériences physiques, parfois coûteuses dans bien des domaines applicatifs, sont précédées par des phases de simulations assistées par ordinateur bien moins chères. Cette approche permet d'explorer un grand nombre de pistes à moindre coût. Ces simulations suivent ce que l'on appelle un modèle physique. Concrètement, dans de nombreux champs applicatifs, cela consiste en la résolution d'équations aux dérivées partielles.

Les méthodes numériques sont un outil pour la résolution d'équations aux dérivées partielles. Ces équations ne pouvant être résolues de manière analytique dans la plupart des cas, les méthodes numériques permettent de traduire ces problèmes mathématiques continus afin de pouvoir les résoudre numériquement sur ordinateur.

Il y a au moins un, souvent deux, axe(s) de discrétisation :

- la discrétisation en espace, qui consiste à découper le domaine du problème physique continu en points ou volumes élémentaires, sur lesquels seront observé les valeurs que prennent les inconnues en espace,
- et la discrétisation en temps lorsque le problème en dépend, ou la droite continue du temps est là aussi divisée en valeurs ponctuelles, donnant une "photographie" des inconnues à un instant t donné.

Le fait de faire ces approximations a un coût. En effet, à chaque calcul, une erreur est produite. Il faut alors contrôler ces erreurs, afin qu'elles ne polluent pas la solution approchée du problème.

Dans la suite, nous présenterons tout d'abord les types de discrétisation en espace, puis différentes discrétisations temporelles que nous abordons. Le problème général suivant sera traité afin d'illustrer notre propos :

$$\partial_t u(t,x) = f(u(t,x)), \tag{1.2.11}$$

où t est la variable temporelle, x la variable d'espace (éventuellement multidimensionnelle), ∂_t et la dérivée partielle en temps. f est une fonction continue s'appliquant sur le temps t et sur les dérivées partielles spatiales de $u(t,x)$. Enfin nous illustrerons cette section par un exemple appliqué au modèle fluide pour la simulation des plasmas de bord : celle d'Emedge3D. Pour une description plus poussée des méthodes numériques, nous renvoyons le lecteur aux références [28, 29, 30].

1.2.1 Discrétisation en espace

La discrétisation en espace permet de calculer le second membre de l'Equation (1.2.11) de manière discrète, *i.e.* de calculer numériquement les dérivées partielles en espace. Nous présentons dans la suite les trois types de discrétisations utilisés dans le code Emedge3D ainsi que dans la partie numérique de cette thèse.

Les différences finies

Dans le cas des différences finies, on considère une grille $(x_i)_{i=0,1,\dots,N_x}$ du domaine en espace, souvent uniforme, dont la maille est de taille $x_{i+1} - x_i = \Delta x$. On cherche alors une relation satisfaite par les valeurs de la fonction en ces

points du maillage $u_i = u(x_i)$. Pour cela, on a besoin d'une approximation des opérateurs différentiels (dérivées par exemple). La méthode des différences finies utilise des formules basées sur des développements de Taylor. Une formule centrée pour la dérivée partielle en espace dans la direction x est, par exemple :

$$\partial_x u(x_i) \simeq \frac{u_{i+1} - u_i}{\Delta x}.$$

Cette dernière formulation est dite "en avant", elle fait intervenir les points i et $i+1$. De la même manière, on peut écrire les formulations en arrière et centrée.

Les volumes finis

Contrairement aux différences finies où l'on s'intéresse à des valeurs ponctuelles, il s'agit dans le cadre des volumes finis de regarder des valeurs moyennes de la fonction inconnue $\bar{u}_i$ sur des volumes élémentaires $\mathcal{V}_i$ de volume Δx, aussi appelés volumes de contrôle, qui pavent le domaine d'étude. En 1D, on peut définir le pavage constant $\mathcal{V}_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ avec $\Delta x = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}$ par exemple. Pour calculer la valeur d'une fonction en un volume donné, on l'intègre en espace sur ce volume :

$$\bar{u}_i = \frac{1}{\Delta x} \int_{\mathcal{V}_i} u(x) dx.$$

Afin de résoudre l'Equation (1.2.11), on l'intègre sur un volume élémentaire $\mathcal{V}_i$ pour obtenir la formule :

$$\partial_t \bar{u}_i = \frac{1}{\Delta x} \int_{\mathcal{V}_i} f(u(t,x)) dx$$

Il reste à exprimer le second membre en fonction des $\bar{u}_i$. Ce type de méthode est bien adapté aux équations hyperboliques ou paraboliques puisque la formule de Green permet de transformer une intégrale volumique en intégrale surfacique. Pour plus de détails, nous renvoyons le lecteur à [17, 36].

Les éléments finis

Il s'agit ici d'une discrétisation provenant de la formulation variationnelle ou faible du problème posé par l'Equation (1.2.11). Le domaine initial en espace est subdivisé en valeurs ponctuelles, similairement à la méthode des différences finies. Un pavage (généralement composé de triangles ou carrés, régulier ou non) est ensuite défini en traçant des lignes entre les points voisins. Les volumes élémentaires $\mathcal{V}_i$ obtenus sont ce que l'on appelle les éléments finis. Plus précisément, un élément fini regroupe à la fois la valeur de la cellule ainsi que

les fonctions de base qui forment la cellule. La solution est ensuite projetée sur la base de fonctions $(\varphi_i)_{i=1,\dots,N}$ des éléments finis qui forment le pavage :

$$u(x) = \sum_{i=1}^{N} u_i \varphi_i(x)$$

Une fois cette projection effectuée, on multiplie l'Equation (1.2.11) par une fonction de base $\varphi_j(x)$ et on intègre sur le domaine pour obtenir

$$\partial_t u_i \int \varphi_i(x)\varphi_j(x)dx = \int f\left(\sum_{i=1}^{N} u_i \varphi_i(x)\right)\varphi_j(x)dx.$$

En fonction de l'expression de f, on peut se ramener à un système différentiel de la forme

$$B\partial_t u = Au,$$

où $u = (u_i)_{i=1,\dots,N}$ est le vecteur représentant la solution et A, B sont des matrices de taille N (appelée matrice de rigidité et de raideur). Un bon choix de fonctions de base φ_i permet de rendre la matrice B diagonale ou facile à inverser. Plus d'informations sur cette méthode peuvent s'obtenir dans les références [15, 13].

La représentation spectrale

La discrétisation spectrale consiste à représenter les données en espace dans l'espace des fréquences. Cette discrétisation est très utilisée lorsque les conditions aux bord du domaine sont périodiques. Les données étant décomposées en sommes de sinus et cosinus, cette discrétisation présente l'avantage de pouvoir simplifier certaines opérations, par exemple le calcul des dérivées partielles.

En effet, en considérant la projection de u dans la base de Fourier :

$$u(x_i) = \sum_k \hat{u}_k(x_i)\exp(\mathrm{i}kx_i)$$

on obtient un nouveau vecteur solution $\hat{u}$ dans cette nouvelle base. La dérivée partielle dans cette représentation devient :

$$\widehat{\partial_x u}_k = \mathrm{i}k\hat{u}$$

Pour passer d'une base à l'autre, on effectue une transformée de Fourier discrète. En pratique, elles sont obtenues via la bibliothèque *fftw3*. La complexité asymptotique en nombre d'opérations pour un vecteur de taille N est en $O(N\log(N))$.

1.2.2 Discrétisation en temps

Dans ce manuscrit, nous nous concentrons sur trois types de discrétisations en temps, chacune présentant ses avantages et inconvénients. Ces types sont présentés dans les paragraphes suivant. A partir d'une des discrétisations spatiales présentées précédemment, on considère le système différentiel $\partial_t u = Au$, où $u = (u_1, u_2, \dots u_N)$ et A est une matrice de taille N issue de la discrétisation spatiale de l'Equation (1.2.11).

Les schémas explicites en temps

Les schémas explicites permettent d'exprimer la solution $u^{n+1} = u(t^{n+1})$ d'un problème physique donné en fonction de ses valeurs aux temps précédents connus u^k, avec $0 \leq k \leq n$. Le plus simple de ces schémas est le schéma d'Euler explicite, qui s'écrit pour l'Equation (1.2.11) :

$$u^{n+1} = u^n + \Delta t^n A u^n, \tag{1.2.12}$$

où $\Delta t^n = t^{n+1} - t^n$. Ce schéma est plutôt simple à mettre en place, étant donné qu'à chaque nouveau temps t^n, le membre de droite dans l'Equation (1.2.12) est entièrement déterminé par l'étape précédente en temps. Par contre, lorsque l'on considère des équations aux dérivées partielles, une contrainte reliant le pas d'espace et de temps doit généralement être imposée. Si cette contrainte n'est pas respectée, le schéma est alors instable. Par exemple, la contrainte sur le pas de temps Δt pour l'équation de la chaleur en une dimension est $\Delta t < \frac{1}{2}\Delta x^2$, où Δx désigne le pas en espace. C'est ce qu'on appelle la condition de Courant-Friedrichs-Lewy. En pratique, pour garantir la stabilité de la méthode numérique, on pose $\Delta t = \text{ncfl}\frac{1}{2}\Delta x^2$ avec ncfl < 1, où ncfl s'appelle le nombre de Courant (nombre CFL). Cette expression indique que plus la grille en espace est fine, et plus il faudra effectuer d'itérations temporelles afin d'atteindre un temps final donné. Ce schéma temporel produit une erreur d'ordre 1 en temps. Il est possible de monter en ordre en mettant en place des méthodes de Runge Kutta (RK, voir [28]).

Les schémas implicites en temps

Les schémas implicites permettent d'exprimer la solution $u^{n+1} = u(t^{n+1})$ d'un problème physique donné en fonction de ses valeurs aux temps précédents connu $u^{k \leq n}$ ainsi que du temps courant. Le plus simple de ces schémas est le schéma d'Euler implicite, qui s'écrit pour l'Equation (1.2.11) :

$$u^{n+1} = u^n + \Delta t^n A u^{n+1}. \tag{1.2.13}$$

Ce type de schéma permet d'éviter les problèmes de stabilité des schémas explicites. Les pas de temps et d'espace peuvent alors être choisis indépendamment du point de vue de la stabilité de la méthode. Par contre, il présente l'inconvénient d'être plus technique à mettre en place. En effet, le membre de droite de l'Equation (1.2.13) dépendant lui aussi de la fonction u au temps t^{n+1}, il faut alors inverser le système. En effet :

$$\begin{aligned} u^{n+1} &= u^n + \Delta t^n A u^{n+1} \\ \Leftrightarrow (I - \Delta t^n A) u^{n+1} &= u^n \\ \Leftrightarrow u^{n+1} &= (I - \Delta t^n A)^{-1} u^n \end{aligned}$$

Inverser ce système peut s'avérer coûteux en terme de calcul. Si f ne dépend pas du temps, comme dans (1.2.11), la matrice $I - \Delta t^n A$ peut être inversée une fois pour toute. Mais, dans le cas où f dépend du temps (et donc $A^n = A(t^n)$ aussi), cela supposerait de ré-assembler et d'inverser la matrice $I - \Delta t^n A^n$ à chaque itération temporelle, ce qui est encore plus coûteux. Enfin, si le problème est non linéaire (*i.e.* A dépend de u), des algorithmes de type point fixe ou Newton doivent être mis en place.

Les schémas de type implicites-explicites (IMEX) en temps

Les schémas semi-implicites sont un mélange des deux méthodes précédentes. Elles ont pour but de rendre le schéma temporel inconditionnellement stable, tout en essayant de rendre la partie implicite du problème la moins coûteuse possible en temps de calcul. Le plus simple de ces schémas est le schéma d'Euler semi-implicite. En supposant que $A = A_1 + A_2$, il s'écrit pour (1.2.11) :

$$u^{n+1} = u^n + \Delta t^n A_1 u^n + \Delta t^n A_2 u^{n+1}, \tag{1.2.14}$$

$$\Leftrightarrow u^{n+1} = (I - \Delta t^n A_2)^{-1}(u^n + \Delta t^n A_1 u^n) \tag{1.2.15}$$

Ici, l'inversion de $(I - \Delta t^n A_2)$ a pour rôle de stabiliser la méthode. Si A_2 est choisie constante, $(I - \Delta t^n A_2)^{-1}$ peut être calculée une fois pour toute à l'initialisation, et faire économiser un temps de calcul conséquent dans le cas d'une matrice A non constante en temps. L'inconvénient ici est de mesurer l'impact de l'augmentation du pas de temps sur l'erreur d'approximation commise. De telles méthodes ont déjà été mises en place dans le cadre d'études sur des équations de type diffusion ou advection-diffusion (voir [53], [57] et [19]). Le schéma présenté ici est d'ordre 1 en ce qui concerne l'évolution de l'erreur par rapport au pas de temps. Il est possible de monter en ordre en mettant en place des méthodes de type Additive Runge Kutta (ARK), présentées dans la référence [57] et dans le Chapitre 2 de cette thèse.

1.2.3 Méthodes numériques appliquées au modèle de bord fluide d'Emedge 3D

Cette section donne tout d'abord une introduction aux méthodes numériques mises en œuvre pour le modèle de bord présenté en Sous Section 1.1.3. En deuxième partie, nous allons aborder les contraintes imposées par cette discrétisation.

Méthodes numériques pour le modèle de bord

Nous allons ici énumérer les méthodes numériques utilisées pour résoudre les équations écrites en Sous Section 1.1.3 au point de départ de cette étude.

Intégration en temps Pour l'intégration en temps, une méthode de Runge Kutta d'ordre 2 ou 4 est utilisée. Cette méthode fait partie de la famille des méthodes explicites (présentée en Sous Section 1.2.2) et suppose alors le respect d'une condition de stabilité pour le pas de temps en fonction de la discrétisation en espace.

Discrétisation en espace Une discrétisation de type différence finies est utilisée pour la direction radiale. Les directions poloïdale et toroïdale sont périodiques, une représentation spectrale est mise en place. Pour le champ de pression, cela donne :

$$p(x,y,z,t) = \sum_{n,m} p_{m,n}(x,t)\exp(\mathrm{i}(n\kappa_z z + m\kappa_y y))$$

avec $\kappa_y = 2\pi/L_y$ et $\kappa_z = 2\pi/L_z$, où L_d désigne la longueur du domaine dans la direction d. Le domaine est discrétisé en une grille 3D de taille (M_x, M_y, M_z), et on fait correspondre $p_{i,j,k}$ à $p_{m_j,n_k}(x_i,t)$. On réécrit alors les opérateurs de dérivées partielles spatiales comme suit :

$$\begin{aligned}
\partial_x u_i &\rightarrow \frac{u_{i+1} - u_{i-1}}{2\delta x} \\
\partial_x^2 u_i &\rightarrow \frac{u_{i+1} - 2u_i + u_{i-1}}{2\delta x^2} \\
\partial_y u_m &\rightarrow im\kappa_y u_m \\
\partial_z u_n &\rightarrow in\kappa_z u_n
\end{aligned}$$

Avec ces dernières expressions, il nous est possible de résoudre numériquement tous les opérateurs spatiaux pour le modèle présenté en Sous Section 1.1.3 à la page 28. Toutefois, pour des raisons de complexité de calcul, les crochets de Poisson présents dans l'opérateur de diffusion parallèle sont résolus par une

méthode d'Arakawa d'ordre 2 (voir [1]) dans l'espace des variables en espace physiques réelles, ce qui suppose un changement de base par transformées de Fourier discrètes (voir la Sous Section 3.1.2 ainsi que [24] pour plus d'explications).

Contraintes imposées par les méthodes numériques pour Emedge3D

Les discrétisations en temps et en espace qui viennent d'être présentées imposent des contraintes sur le plan numérique. En effet, les termes de diffusion présents dans les Equations (1.1.8), (1.1.9) et (1.1.10) en page 28 nous indiquent la présence d'une condition de stabilité très restrictive lors de l'utilisation d'un schéma temporel de type explicite. La diffusion la plus forte numériquement étant la diffusion parallèle, uniquement présente dans l'équation de pression (1.1.9), la condition sur le pas de temps est de la forme :

$$\delta t < \mathrm{ncfl} \frac{\max(\delta x^2, \delta y^2, \delta z^2)}{4\varrho(\xi_{\parallel}(z)\nabla_{\parallel}^2)}$$

où ncfl est le nombre de Courant (posé inférieur à 1 afin de respecter la condition de stabilité), et ϱ la fonction donnant le rayon spectral d'un opérateur. Le pas de temps alors imposé est très petit. Les relaxations de barrière de transport n'apparaissant qu'à des temps très longs, cela suppose un très grand nombre d'itérations temporelles (typiquement 10^7 itérations).

Une stratégie serait de s'affranchir de la condition de stabilité en passant par des méthodes de type implicite en temps. Néanmoins, la taille de maillage imposée par un autre phénomène que l'on désire observer, à savoir les îlots magnétiques, est très élevée (de l'ordre de 4×10^7 points). Et le problème de bord étant non linéaire en temps, cela supposerait l'inversion d'un système non linéaire (par une méthode de point fixe) à chaque pas de temps, ce qui serait extrêmement coûteux.

Une autre contrainte numérique est imposée par les opérateurs de diffusion. Dans sa globalité, la diffusion dans l'Equation (1.1.9) est à caractère anisotrope : elle est plus forte dans la direction parallèle aux lignes de champ. Le risque dans ce cas de figure est de produire une erreur trop élevée en résolvant la diffusion parallèle, erreur qui viendrait polluer numériquement la diffusion dans la direction perpendiculaire.

Enfin, la dernière contrainte forte est imposée par la présence de deux discrétisations : semi-spectrale et variables d'espace réelles. Elles impliquent en effet la présence de transformées de Fourier discrètes 2D en un nombre élevé (jusqu'à 5 allers-retours par pas de temps), qui impliquent un coût supplémentaire en temps de calcul.

En pratique, les physiciens utilisent Emedge3D en mettant en œuvre une partie des équations seulement. Cela consiste en la simulation de cas tests intermédiaires, moins coûteux, pouvant faire l'objet de simulation avec des temps d'exécution raisonnables.

1.3 Introduction au calcul haute performance (HPC)

La demande de performances en simulation numérique est croissante. En effet, les domaines applicatifs souhaitent mettre en œuvre de plus en plus de calculs tout en considérant des tailles de données grandissantes, supposant des temps d'exécution longs. En guise d'illustration, nous pouvons citer les programmes pour la simulation de prévisions météorologiques et de l'étude du climat, dont les besoins ne cessent de croître.

Le calcul haute performance (High Performance Computing en anglais) est une discipline qui a pour but de répondre à cette problématique. Il consiste en l'adaptation et l'exécution des programmes lourds en calcul sur ce que l'on appelle des superordinateurs ou supercalculateurs. Ces machines comprennent habituellement un grand nombre de processeurs ou unités de calculs, leur permettant d'exécuter de nombreuses tâches en parallèle. Elles mettent aussi à disposition un grand espace mémoire pour traiter les problèmes larges en taille de données. Le système d'entrée/sortie est lui aussi étudié de manière à fournir une large bande passante.

Exécuter un programme sur un superordinateur demande des développements informatiques spécifiques. Le programme original séquentiel doit tout d'abord être optimisé. Ensuite, la mise en œuvre de la parallélisation suppose d'identifier et de construire des tâches de calcul qui peuvent s'exécuter simultanément. On parle alors d'indépendance : les tâches dont les résultats ne dépendent pas les uns des autres peuvent être exécutées simultanément.

Dans la suite, un outil de quantification de l'accélération possible par parallélisation sera présenté. Puis, la taxonomie de Flynn sera introduite avant de présenter deux grands types de parallélisme : le parallélisme sur machine à mémoire partagée et sur machine à mémoire distribuée.

1.3.1 La loi d'Amdahl

La loi d'Amdahl a été formulée par Gene Amdahl en 1967. Elle permet de quantifier le gain en temps de calcul atteignable en parallélisant une application. Ce gain est fonction de la portion de code parallélisable au sein du code total de l'application ciblée et du nombre d'unités de calcul utilisées. Si on

appelle T_{seq} le temps d'exécution du programme original (en séquentiel), T_{par} le temps d'exécution en parallèle, p la proportion de code parallélisable (avec $0 \leq p \leq 1$) et Acc le facteur d'accélération obtenu pour la partie parallèle seule, on peut écrire dans un premier temps :

$$T_{par} = (1-p)T_{seq} + \frac{pT_{seq}}{Acc}$$

Cette dernière relation permet de calculer le facteur d'accélération global de l'application, qui est ici appelé SU_1 (SU pour speedup) :

$$SU_1 = \frac{T_{seq}}{T_{acc}} = \frac{1}{(1-p) + \frac{p}{Acc}}$$

Le speedup maximum s'obtient à partir de la dernière équation, en faisant tendre Acc vers l'infini :

$$SU_{\max} = \lim_{Acc \to \infty} \frac{1}{(1-p) + \frac{p}{Acc}} = \frac{1}{1-p}$$

La loi d'Amdahl s'exprime en ajoutant une dépendance en nombre d'unités de calcul N :

$$SU_2(p, N) = \frac{1}{1-p+\frac{p}{N}}$$

La formule pour $SU_{\max}$ seule donne une indication sur l'utilité d'une parallélisation pour une application donnée. En effet, un programme s'exécutant en une durée d'une heure et dont la portion de code parallélisable est 0.1 ne peut être ramené au mieux qu'à 54 minutes d'exécution, soit un gain de seulement 6 minutes.

Les expressions de $SU_{\max}$ et SU_2 prises conjointement donnent une bonne information sur le nombre d'unités de calcul "utiles" à considérer. En effet, en prenant l'exemple d'un programme ayant une portion $p = 0.9$ parallélisable, on obtient $SU_{\max} = 10$. Pour $SU_2(0.9, N)$, on observe les valeurs $SU_2(0.9, 10) = 5.3$, $SU_2(0.9, 100) = 9.2$ et $SU_2(0.9, 1000) = 9.9$. On note alors que passer de 10 à 100 unités de calcul présente un intérêt non négligeable, permettant la division du temps de calcul par presque 2. Par contre, passer de 100 à 1000 unités de calcul ne changera que très peu le gain relatif obtenu (facteur 1.1) tout en monopolisant beaucoup plus de ressources, et aller au dessus de 1000 se révélera presque inutile en terme d'augmentation du speedup et de réduction du temps d'exécution total.

1.3.2 La taxonomie de Flynn

Par opposition à l'archétype d'une machine séquentielle, où les instructions s'exécutent une à une au rythme des cycles horloges sur une seule donnée, les

machines parallèles peuvent suivre différents modèles d'exécution. Ces modèles ont été proposés par Michael J. Flynn [20]. Ils sont caractérisés par l'unicité ou la multiplicité des flots (streams) d'instructions et de données considérés, donnant lieu à un total de quatre modèles :

- le modèle SISD (Single Instruction stream, Single Data stream) comporte un seul flot d'instruction qui s'applique sur un seul flot de données. Il correspond au cas des ordinateurs séquentiels avec une seule unité de calcul.
- le modèle MISD (Multiple Instruction stream, Single Data stream), cas de plusieurs flots d'instructions qui s'exécutent sur un même flot de données. D'un point de vue pratique, ce modèle peut se voir comme un pipeline matériel sur les données : les données passent d'une unité de calcul à une autre où elles se voient appliquer un flot d'instruction différent.
- le modèle SIMD (Single Instruction stream, Multiple Data stream), où un même flot d'instruction est exécuté sur plusieurs flots de données. Les instructions s'exécutent de manière synchrones sur plusieurs données à la fois. Les processeurs superscalaires, capables d'exécuter une instruction en simultanée sur plusieurs données en sont un exemple (on parle aussi d'instruction vectorielle).
- le modèle MIMD (Multiple Instruction stream, Multiple Data stream), dernier modèle permettant l'expression la plus large du parallélisme. Les flux d'instructions s'appliquent à un flot de donnée qui leur sont propre, permettant une segmentation de la charge totale de travail au niveau des instructions et des données.

1.3.3 Architectures des supercalculateurs

Techniquement, la mise en place de la parallélisation se fait en fonction de la technologie visée. Il existe deux catégories particulièrement répandues sur les supercalculateurs actuels :

- les architectures à mémoire partagée,
- les architectures à mémoire distribuée.

Il est possible de mélanger ces technologies afin de former une machine parallèle hybride, machines de plus en plus communes sur le marché des superordinateurs. En guise d'illustration, nous pouvons citer la machine japonnaise Helios (voir Figure 1.3.3), située à Rokkasho, mise en place par le CEA (Commissariat à l'Energie Atomique) en soutien du projet ITER. Elle affiche une puissance de calcul de 1,5 pétaflops. Elle est composée de 4410 nœuds de calcul bullx B510 reliés par réseau InfiniBand, pour un cumul de 8820 processeurs Intel Xeon E5-2600, soit 70560 coeurs en tout. Elle comprend un total de 280 teraoctets

FIGURE 1.4 – Helios, supercalculateur pour le projet ITER, Rokkasho, Japon

de mémoire vive et 5,7 petaoctets d'espace de stockage. C'est une machine permettant un parallélisme hybride : à mémoire distribuée entre les nœuds, et à mémoire partagée entre les cœurs appartenant au même nœud.

Il existe un troisième type d'architecture répandu : l'accélérateur. Ce moyen de calcul a été conçu dans le but d'atteindre de hautes puissances de calcul tout en minimisant les coûts énergétiques et l'encombrement par rapport aux processeurs multi-coeurs classiques. Un accélérateur consiste principalement en l'accumulation d'un grand nombre d'unités de calcul et un grand espace mémoire partagé entre ces unités. Les unités de calcul sont moins puissantes et spécialisées pour le calcul, donc moins polyvalentes que celles d'un processeur généraliste. C'est pourquoi la programmation parallèle sur cette technologie demande un effort de programmation plus conséquent, les codes doivent être adaptés en fonction de l'architecture propre à l'accélérateur. En guise d'exemple, nous pouvons citer les cartes graphiques (GPGPU pour general-purpose processing on graphics processing units) concaténant un grand nombre d'unités de calcul réparties en groupes partageant des ressources (cache mémoire, pile d'instruction, ...). La machine Helios présentée plus haut (Figure 1.3.3) est notamment équipée d'accélérateurs MIC (Many Integrated Core) d'Intel pour un total de 360 noeuds Xeon Phi accessibles, ajoutant une puissance de calcul théorique de 400 téraflops.

Dans la suite, nous allons présenter les deux types de parallélismes qui nous intéressent pour ce projet, à savoir le parallélisme sur architecture à mémoire partagée et le parallélisme sur architecture à mémoire distribuée.

1.3.4 Le parallélisme sur machine à mémoire partagée

Les machines à mémoire partagée partagent une même zone mémoire, accessibles par toutes les unités de calcul. Un exemple de ce type de machine est le processeur Intel Xeon X5675 basé sur l'architecture Westmere qui sera utilisé comme architecture test dans le Chapitre 3. La machine que nous avons utilisé comprend deux de ces processeurs pour un total de 12 cœurs, qui auront tous accès à un espace mémoire commun : les mémoires RAM de chacun des processeur sont unifiées grâce à la technologie NUMA (Non Uniform Memory Access), comme indiqué par le schéma de la Figure 1.5 page 41. Nous avons aussi eu accès à une autre machine à mémoire partagée, SMP (pour symmetric multiprocessing), composée de 64 cœurs. Ces deux machines font partie du mésocentre d'Aix-Marseille université.

Comme moyen de parallélisation, nous avons choisi le paradigme OpenMP (Open Multi-Processing, voir [7]), qui se compose d'une bibliothèque et d'un ensemble de directives de compilation. Ce moyen de parallélisation permet de définir des tâches à exécuter en parallèle, par exemple en distribuant les itérations d'une boucle for via une simple directive. Les tâches ainsi définies sont ensuite distribuées sur différents threads, eux même répartis sur les unités de calcul de la machine à mémoire partagée considérée. Pour plus de détails, nous renvoyons le lecteur à la référence [7].

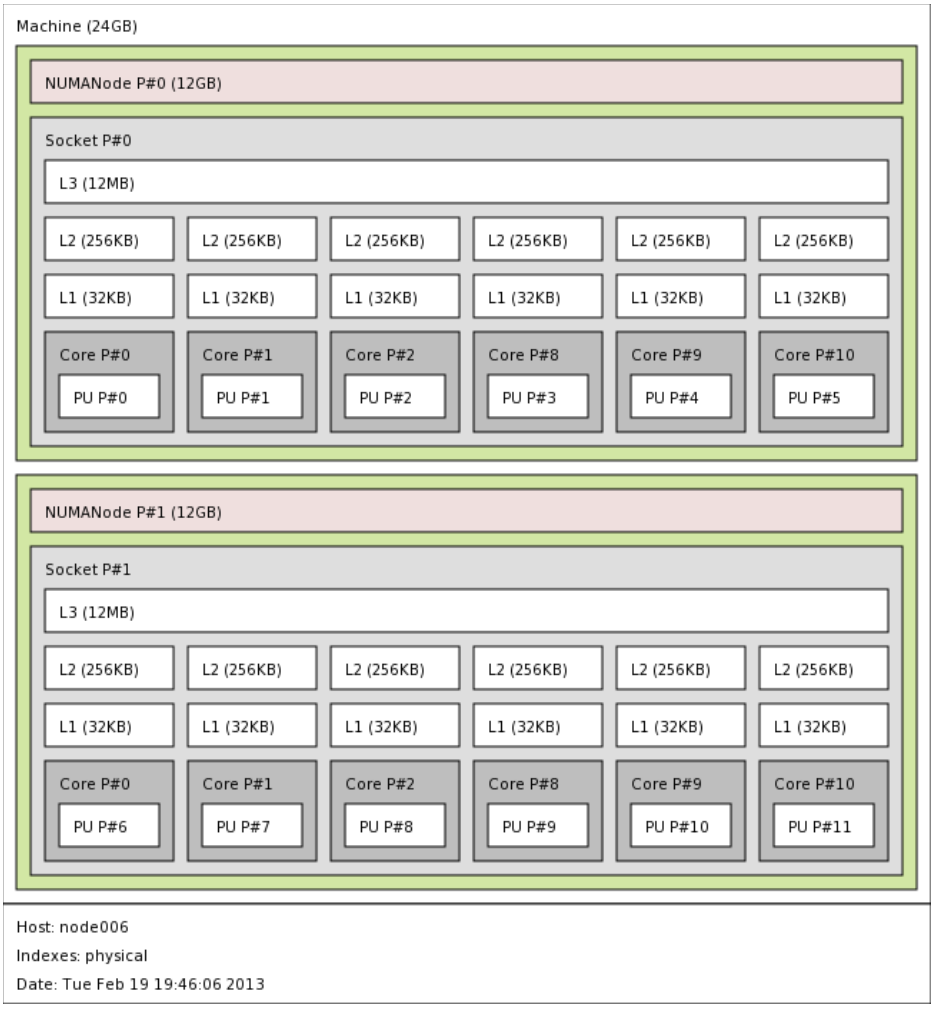
L'utilisation du langage de directive présente l'avantage d'être peu intrusive au niveau du code, permettant d'obtenir un code parallèle rapidement tout en gardant un code proche de la version séquentielle originale. Néanmoins, pour tirer pleinement parti des performances des architectures à mémoire partagées, il est souvent nécessaire d'apporter de nombreuses optimisations (voir le Chapitre 3 de ce manuscrit).

1.3.5 Le parallélisme sur machine à mémoire distribuée

Typiquement, les machines à mémoires distribuées sont composées d'un grand nombre de processeurs ou multiprocesseurs ayant chacun leur zone de mémoire privée, on parle aussi de nœud de calcul. Ces nœuds sont reliés entre eux par un réseau de communication rapide (réseau infiniBand avec un très haut débit par exemple) afin qu'ils puissent communiquer entre eux. Le cluster du mésocentre d'Aix Marseille Université a été utilisé durant cette thèse. Il regroupe 1152 cœurs Intel X5675 organisés en nœuds de 12 cœurs (voir la Figure 1.5 page 41).

Dans ce travail de thèse, la norme Message Passing Interface (MPI, voir [21]) a été utilisée. Comme son nom l'indique, MPI est un outil de programmation par échange de messages. C'est un standard de la programmation par

FIGURE 1.5 – Schéma de l'organisation du processeur multicœur Intel X5675. Obtenu avec l'outil hwloc.



passage de messages. Il existe plusieurs implémentations/bibliothèques MPI, dont la bibliothèque Open MPI (open source) qui est utilisée dans le Chapitre 4 de cette thèse.

La programmation MPI consiste à distribuer la charge de travail totale sur plusieurs processus, qui peuvent être exécutés sur plusieurs nœuds de calcul, chacun possédant un espace mémoire qui lui est propre. C'est au programmeur de diviser explicitement la charge de travail, par exemple en distribuant le volume total de données à traiter sur les différents processus dans le cadre de la résolution d'une équation aux dérivées partielles.

Lors de l'exécution d'un processus MPI, toutes les variables propres à ce processus sont à priori privées et visibles uniquement dans ce processus. C'est pourquoi la bibliothèque MPI propose différentes fonctions de communication, bloquantes ou non, point à point ou collectives, permettant d'échanger des données entre plusieurs processus. C'est aussi au programmeur qu'incombe la tâche d'organiser les communications entre les processus en fonction des besoins.

Du point de vue du développement, la mise en place d'un tel parallélisme est particulièrement intrusive, impliquant beaucoup de modifications dans le cas des algorithmes pour la résolution des équations aux dérivées partielles. Ces modifications sont principalement la distribution des données, des calculs et l'organisation des communications. C'est pourquoi l'étude du Chapitre 4 est réalisée sur un code distinct mais proche d'Emedge3D au niveau des structures mathématiques des équations à résoudre, comme une étude préliminaire à la mise en place d'une parallélisation de type MPI dans le code Emedge3D.

1.4 Cadre, objectifs et plan de la thèse

Ce travail s'inscrit dans un projet financé par l'Agence Nationale pour la Recherche appelé E2T2 pour *Effets électromagnétiques sur le transport turbulent dans les plasmas chauds magnétisés*. Ce projet associe plusieurs équipes de corps de recherche différents : des physiciens auteurs du code Emedge3D (Université de Marseille, PIIM), des numériciens (CEA/INRIA) et des informaticiens (Université de Strasbourg/Icube).

L'ANR E2T2 a pour but l'étude des phénomènes multi-échelles dans un plasma de fusion magnétique, et plus particulièrement au bord des tokamaks. Ces phénomènes sont encore peu compris pour l'instant. Il s'agit tout d'abord des barrières de transport (voir [6]) et de leur relaxation (voir la Sous Section 1.1.3 page 26), déjà en partie caractérisées par le code Emedge3D. Ensuite, un autre phénomène important que l'on observe dans les machines type tokamak, est la formation d'îlots magnétiques. Ce sont des structures magnétiques sec-

ondaires qui apparaissent comme conséquence de certaines instabilités, dites de déchirement. Elles consistent en la formation de surfaces magnétiques ovales fermées dans la direction poloïdale du tore, en y emprisonnant des particules, surfaces qui peuvent s'étendre progressivement jusqu'au bord du tokamak (voir [16] et [46]). Les îlots magnétiques sont donc nuisibles au fonctionnement de la machine, car ils augmentent le transport d'énergie et de particules, pouvant provoquer ce que l'on appelle une disruption.

La principale difficulté dans l'étude de l'évolution couplée îlot-turbulence réside en son caractère multi-échelle, et tout d'abord en terme d'échelle spatiale. Les îlots mesurent en effet quelques centimètres, les turbulences sont millimétriques, alors que la taille des tokamaks est de l'ordre du mètre. Cela suppose une taille de maillage conséquente, et donc un gros volume de données à traiter. Ensuite, en terme de temps, les phénomènes de relaxation de la barrière de transport s'observent à des temps élevés, ce qui suppose un grand nombre d'itérations temporelles.

L'objectif principal de cette thèse est alors d'accélérer le code Emedge3D, principalement par deux moyens.

Le premier axe de travail est de modifier les schémas numériques employés afin de réduire le nombre d'itérations temporelles requises par une simulation. Ce nombre d'itérations temporelles est fixé tout d'abord par la taille de la grille en espace en ce qui concerne les schémas temporels de type explicite, et aussi par la précision souhaitée au niveau des résultats. Le challenge consiste à trouver un compromis nombre d'itérations/précision afin d'obtenir des résultats plus rapidement tout en respectant les contraintes imposées en terme de précision par les phénomènes que l'on désire observer. Cette étude fait l'objet du Chapitre 2.

Un second axe de travail est l'optimisation et la parallélisation du code Emedge3D. Il s'agit tout d'abord d'optimiser le code séquentiellement en tentant de simplifier les algorithmes et d'optimiser leurs calculs et accès mémoire. Ensuite, différents types de parallélisations seront envisagés. Nous explorerons tout d'abord la parallélisation sur machine à mémoire partagée via l'outil OpenMP dans le Chapitre 3.

Puis, nous présentons une parallélisation hybride MPI (Message Passing Interface) et OpenMP dans le Chapitre 4. L'enjeu dans ce chapitre est de palier au caractère *memory bound* du code (*i.e.* limité par la bande passante vers la mémoire vive) lors du passage en parallèle : dans le Chapitre 3. Les facteurs d'accélération obtenus en fonction du nombre de cœurs sont en effet limités par la bande passante mémoire si l'on se borne à une étude sur machine à mémoire partagée. Le but est d'obtenir un code hybride MPI/OpenMP performant et efficace pour réduire les temps d'exécutions trop longs. Ce travail est réalisé non plus sur le code Emedge3D, mais sur une équation de type advection

diffusion très proche des équations d'Emedge3D, étant donné le caractère très intrusif que suppose la stratégie de parallélisation choisie au niveau du code.

Enfin, nous rappellerons les différentes contributions propres à chaque chapitre dans la conclusion de ce manuscrit au Chapitre 5. Nous y présenterons aussi les ouvertures possibles au travail effectué que nous jugeons particulièrement pertinentes.

Chapitre 2

Amélioration de la diffusion anisotrope

Le modèle résolu par le code Emedge3D comporte un terme de diffusion anisotrope particulièrement délicat à traiter du point de vue des méthodes numériques. Il s'agit du terme de gradient parallèle carré dans l'équation de pression $\nabla^2_{\parallel} p$ (voir l'Equation (1.1.9) page 28). Cet opérateur est également abordé dans sa totalité dans le Chapitre 3 (voir l'Equation (3.1.3) page 92) et de manière partielle dans le Chapitre 4 (voir l'Equation (4.0.1) page 114). Il pose principalement deux problèmes. Tout d'abord, il impose une forte condition de stabilité sur la valeur du pas de temps utilisable dans le cas de méthodes d'intégrations temporelles de type explicite. Ceci a pour effet la nécessité d'effectuer un grand nombre de pas de temps pour les simulations, et implique un long temps de calcul. D'autre part, l'anisotropie de cette diffusion amplifie l'erreur de magnitude dans la direction parallèle aux lignes de champs, produisant une diffusion perpendiculaire numérique venant polluer la valeur de la diffusion effective dans la direction perpendiculaire aux lignes de champs. Cette pollution est reconnue par les physiciens comme étant un problème majeur impactant la qualité de leur simulation (voir [52, 25, 55]).

C'est pourquoi ce chapitre est dédié à la comparaison de méthodes numériques pour la diffusion anisotrope, particulièrement étudiée dans la physique des plasmas. On s'intéresse aux méthodes d'intégration en temps pour cette équation, ainsi qu'à la discrétisation spatiale de l'opérateur de diffusion. Des méthodes temporelles de type implicites et semi-implicites sont couplées à des méthodes de volumes finis. Elles sont comparées sur un jeu de cas tests pertinents pour la fusion par confinement magnétique. L'approche semi-implicite dite de pénalisation avec un schéma temporel de type Additive Runge Kutta associée à une résolution de l'opérateur spatial d'ordre élevée apparaît comme la plus efficace des méthodes présentées, autant du point de

vue du gain potentiel sur la valeur du pas de temps maximale utilisable que du point de vue de la qualité des résultat qu'elle produit, surtout lorsqu'il s'agit de traiter des problèmes de diffusion non linéaires comme l'opérateur de diffusion parallèle d'Emedge3D.

2.1 Introduction

Les problèmes de diffusion apparaissent dans de nombreux domaines d'application tels que le traitement d'image [49], l'imagerie par résonance [2] ou le transport dans les couches géologiques [4]. Dans ce chapitre, nous nous concentrons sur la diffusion intervenant dans les équations pour la simulation de plasmas pour la fusion nucléaire par confinement magnétique [8]. Dans ce contexte, la diffusion est anisotrope : elle est plus forte dans la direstion parallèle aux lignes de champs que dans la direction perpendiculaire. Ceci ce traduit dans les équations par la présence de deux coefficients : $\chi_\perp$ et $\chi_\parallel$ qui viennent pondérer les opérateurs de diffusion dans les directions perpendiculaires et parallèles respectivement. On s'intéresse souvent aux rapport de ces deux coefficients : $\chi_\perp/\chi_\parallel$, qui peut être petit (typiquement $\chi_\perp/\chi_\parallel \sim 10^{-3}$ dans le cas des plasmas de bords, voir [5, 24]). Par conséquent, la diffusion dans notre cadre pose deux types de problèmes. Dans un premier temps, la diffusion dans la direction parallèle étant forte, elle impose une forte condition de stabilité sur le pas de temps (*i.e.* $\Delta t << 1$), ce qui implique un fort coût en calcul. Ensuite, l'erreur produite dans la direction parallèle vient polluer la diffusion perpendiculaire effective ainsi que le transport (voir [25, 26, 53, 52]), ce qui fausse les résultats.

Plusieurs travaux ont été menés pour améliorer les méthodes numériques existantes pour résoudre les équations de diffusion, portant essentiellement sur deux aspects. Le premier porte sur l'intégration en temps des équations. En effet, au vu de la contrainte imposée sur le pas de temps, il est intéressant de considérer des méthodes de type implicites ou semi-implicites permettant d'assurer une stabilité inconditionnelle. Dans la référence [53], une méthode semi-implicite couplée avec un splitting directionnel est proposée. Dans le papier [19], une technique de pénalisation basée sur un splitting de l'opérateur de diffusion permet de considérer l'inversion d'un problème plus simple (un laplacien) au lieu d'inverser le système non linéaire en totalité, dans le cas d'une méthode de type implicite. Dans la référence [43], les auteurs proposent un schéma qui préserve le modèle asymptotique (ou encore schéma AP), permettant de traiter des problèmes de diffusion non linéaires fortement anisotropes. Dans [25], une méthode implicite en temps est associée à une méthode d'éléments finis d'ordre élevé.

Le deuxième aspect concerne les méthodes numériques pour la résolution en espace de l'opérateur. La plupart du temps, des méthodes de type volume finis d'ordre 2 sont mises en œuvre. Pour pouvoir traiter de forts gradients et atténuer l'erreur dans la direction perpendiculaire aux lignes de champs magnétique, il est naturel de considérer des méthodes d'ordres plus élevés et d'observer leurs influences sur la solution approchée, comme rapporté par [25] où une méthodes des éléments finis d'ordre élevé est employée. Néanmoins, plusieurs modèles comportant une diffusion anisotrope impliquent aussi le traitement d'opérateurs de transport, pour lesquels l'emploi d'une méthode de type volume finie est plus adaptée. Un autre point important dans le cas de la diffusion est le principe de conservation du maximum et de la monotonie le long de la direction de diffusion. Nous renvoyons le lecteur à [53, 52, 48] pour plus de détails.

L'objectif principal de ce travail est d'associer, de valider et d'évaluer différentes méthodes numériques récentes pour la diffusion anisotrope. Dans un premier temps, une discrétisation de type volumes finis du second et du quatrième ordre (voir [57]). Ensuite, on s'intéresse à la discrétisation en temps en mettant en place des méthodes de type implicite et semi-implicites, afin d'éviter la présence d'une condition de stabilité. Notamment, nous évaluerons les méthodes semi-implicites présentées dans les références récentes [53, 19].

De manière plus détaillée, les différentes approches citées dans le paragraphe ci-dessus sont restreintes à l'ordre 1 en temps et à l'ordre 2 en espace. L'utilisation d'un ordre élevé en temps n'a pour l'instant pas fait l'objet de beaucoup de travaux de recherches. Pour atteindre cet ordre élevé, les méthodes de type Additive Runge-Kutta (ARK) sont particulièrement intéressantes car elles permettent d'atteindre un ordre arbitraire (similairement aux méthodes de Runge-Kutta explicites). Dans cette étude, nous combinerons les idées de [57] et [19] pour en dériver une méthode d'ordre 4 en temps et en espace sans aucune condition de stabilité sur le pas de temps. L'idée principale, que l'on retrouve dans les références [19, 18], est de pénaliser la diffusion linéaire ou non linéaire par un laplacien. Ce laplacien est traité de manière implicite et la partie restante (correspondant à la diffusion totale moins le laplacien) est traitée de manière explicite. Il en résulte un schéma numérique à coût mesuré, consistant en l'inversion d'un laplacien pour chaque itération temporelle pour lequel le coût en terme de calcul est réduit, par rapport à l'inversion du système non linéaire en sa totalité. Un gain potentiel d'un facteur 10 au moins (voir [19]) peut être obtenu. Une version d'ordre 1 en temps est obtensible par la méthode IMEX par exemple, qui se généralise aux ordres plus élevés par les méthodes ARK. Notons que des méthodes préservant le modèle asymptotique (schémas AP) ont été développées pour la diffusion anisotrope (voir [38, 43, 47, 9, 56]). Elles produisent une précision uniforme, quelque soit le

degré d'anisotropie. Mais même si ces techniques assurent une précision uniforme, elles supposent l'inversion d'un grand système non linéaire qui s'avère coûteuse en terme de calcul.

Concernant la discrétisation spatiale, une bonne approche est présentée dans [57], basée sur les volumes finis. En particulier, lorsque l'on désire coupler la diffusion au transport, il semble naturel d'utiliser des méthodes de volumes finis pour les deux opérateurs. De plus, lorsque des conditions aux bords non périodiques sont considérées, les méthodes spectrales ne sont plus intéressantes alors que différentes techniques permettent aux méthodes de type volumes finis d'atteindre des ordres élevés en espace, même en ce qui concerne les bords du maillage (voir [57, 19]). Des méthodes de volumes finis d'ordre 2 s'obtiennent à partir de la formule du point milieu par exemple pour évaluer les intégrales surfaciques. Récemment, des formules quadratiques d'ordres plus élevés ont été utilisées pour atteindre un ordre 4 en espace. Une extension au transport non linéaire à été proposée dans [42] dans le cadre d'un problème hyperbolique que nous adaptons ici à la diffusion anisotrope non linéaire.

Dans ce travail, nous évaluons les méthodes présentées ci-dessus et proposons des extensions à des ordres plus élevés pour certaines d'entre elles. En outre, afin d'avoir un aperçu du comportement de ces méthodes pour la modélisation dans le cas de problèmes 2D ou 3D, nous les comparons pour certains problèmes classiques pertinents pour la simulation des plasmas de bords (voir [6, 23, 55]). Plus précisément, les simulations numériques pour les codes de turbulence fluide électromagnétique demandent l'utilisation de hautes résolutions spatiale et temporelle afin d'observer les phénomènes de relaxation de la barrière de transport (voir [6, 23]) et d'apparition et évolution des îlots magnétiques ([46, 16]). De plus, les méthodes explicites en temps sont un frein à l'observation de ces phénomènes apparaissant à des temps longs de par la contrainte qu'elles imposent sur le pas de temps, d'où la nécessité d'utiliser des méthodes implicites ou semi-implicites.

L'association d'une méthode semi-implicite d'ordre élevé (ARK2 ou ARK4) et d'un schéma spatial de type volumes finis d'ordre 4 produit un schéma numérique performant, autant au niveau de l'augmentation du pas de temps (et donc de la réduction du temps de calcul) qu'au niveau du contrôle de l'erreur de diffusion dans la direction perpendiculaire. Une discussion sur le "domaine de validité" de la valeur du pas de temps dans le cas de méthodes implicites et semi-implicites est également menée. En effet, même si ces schémas temporel sont inconditionnellement stables, un pas de temps trop grand entraîne la perte de l'ordre de précision en espace. Néanmoins, l'utilisation d'ordres plus élevés en temps permet de surmonter cette difficulté. Enfin, en vue d'un couplage de la diffusion anisotrope avec des termes de transport pour décrire des phénomènes plus complexes (voir [6, 23, 44, 46]), les méthodes doivent être efficace du

point de vue des temps de calcul qu'elles impliquent : en effet, les méthodes (complètement) implicites apparaissent comme trop coûteuses alors que les méthodes que nous proposons permettent de se restreindre à l'inversion de laplaciens 1D ou 2D, présentant un bon compromis entre précision et efficacité.

Nous nous concentrerons sur la simulation numérique pour l'équation de diffusion anisotropique satisfaite par la fonction $T(x, y)$, avec $(x, y) \in [0, L_x] \times [0, L_y]$, $(L_x, L_y > 0)$

$$\partial_t T = -\nabla \cdot \vec{\mathcal{Q}}, \qquad \vec{\mathcal{Q}} = -B\,\nabla T \tag{2.1.1}$$

où $\nabla = (\partial_x, \partial_y)$ désigne le gradient en espace, $\vec{\mathcal{Q}}$ le flux et B une matrice symétrique définie positive.

La matrice peut prendre diverses formes, telles que $B = (\vec{b} \otimes \vec{b})$; dans ce cas, $\vec{\mathcal{Q}}$ représente le flux le long des lignes de champs et $\vec{b} = \vec{b}(x, y) = (b_x, b_y)$ désigne le vecteur champ magnétique unitaire. La matrice de diffusion B peut également dépendre de l'inconnue T donnant lieu à un problème de diffusion non linéaire. Ce cas de figure ajoute un degré de difficulté supplémentaire étant donné qu'un schéma de type implicite en temps implique la mise en œuvre d'une méthode (coûteuse) du point fixe.

La suite du chapitre est organisée comme suit. Premièrement, nous présentons différentes méthodes d'intégration temporelle. Ensuite, nous nous concentrons sur les schémas spatiaux en introduisant la méthode des volumes finis adaptée aux problème ciblé. Puis, une analyse de stabilité est exposée dans la Section 2.4. Enfin, la Section 2.5 est dédiée à un ensemble de cas tests pour la diffusion anisotrope.

Sommaire

2.2 Discrétisation en temps

Dans cette section, nous présentons et rappelons quelques schémas temporels semi-implicites permettant de construire des méthodes inconditionnellement stable pour les Equations (2.1.1).

2.2.1 Méthode semi-implicite SH pour Sharma-Hammett (voir [53])

Détaillons l'équation de diffusion satisfaite par T

$$\begin{aligned}\frac{\partial T}{\partial t} &= \nabla \cdot (B \nabla T) \\ &= \left[\partial_x(b_{xx}\partial_x T) + \partial_x(b_{xy}\partial_y T) + \partial_y(b_{xy}\partial_x T) + \partial_y(b_{yy}\partial_y T)\right],\end{aligned}$$

où B est donné par

$$B = \begin{pmatrix} b_{xx} & b_{xy} \\ b_{xy} & b_{yy} \end{pmatrix},$$

avec $b_{xx}b_{yy} \geq b_{xy}^2$.

Cette discrétisation en temps est basée sur un splitting directionnel et se compose de deux étapes. Tout d'abord, on considère la direction x :

$$\frac{T^\star - T^n}{\Delta t} = \partial_x(b_{xx}\partial_x T^\star) + \partial_x(b_{xy}\partial_y T^n),$$

qui se réécrit en posant la notation $\mathcal{D}_{ij} = -\partial_i(b_{ij}\partial_j)$:

$$T^\star = (1 + \Delta t \mathcal{D}_{xx})^{-1}(1 - \Delta t \mathcal{D}_{xy})T^n.$$

La deuxième étape traite la direction y :

$$\frac{T^{n+1} - T^\star}{\Delta t} = \partial_y(b_{xy}\partial_x T^\star) + \partial_y(b_{yy}\partial_y T^{n+1}),$$

qui se réécrit elle aussi

$$T^{n+1} = (1 + \Delta t \mathcal{D}_{yy})^{-1}(1 - \Delta t \mathcal{D}_{yx})T^\star.$$

On obtient alors T^{n+1} en fonction de T^n, que l'on écrit sous forme compactée

$$T^{n+1} = (1+\Delta t\mathcal{D}_{yy})^{-1}(1-\Delta t\mathcal{D}_{yx})(1+\Delta t\mathcal{D}_{xx})^{-1}(1-\Delta t\mathcal{D}_{xy})T^n.$$

Un des principaux avantages de cette méthode est qu'elle implique l'inversion d'un opérateur spatial de dimension 1, ce qui est intéressant au niveau du coût en calcul. Néanmoins, la généralisation au cas tridimensionnel et au cas non linéaire n'est pas immédiate. En effet, comme mentionné dans [53], le splitting présenté ci-dessus doit être modifié dans le cas 3D pour assurer la stabilité inconditionnelle de la méthode. De plus, dans le cas non linéaire $B = B(T)$,

$$B(T) = \begin{pmatrix} b_{xx}(T) & b_{xy}(T) \\ b_{xy}(T) & b_{yy}(T) \end{pmatrix},$$

une généralisation directe nécessiterait l'application d'une méthode de type point fixe afin d'inverser les termes diagonaux (∂_x^2 et ∂_y^2). Pour la première étape, cela donnerait :

$$\frac{T^\star - T^n}{\Delta t} = \partial_x(b_{xx}(T^\star)\partial_x T^\star) + \partial_x(b_{xy}(T^n)\partial_y T^n),$$

qui peut s'écrire $\mathcal{F}(T^\star) = T^n + \Delta t\partial_x(b_{xy}(T^n)\partial_y T^n)$ avec $\mathcal{F}(T^\star) = T^\star - \Delta t\partial_x(b_{xx}(T^\star)\partial_x T^\star)$. On peut résoudre cette équation avec la méthode du point fixe ou par l'algorithme de Newton, qui se trouvent être deux méthodes coûteuses en terme de calcul, et particulièrement lorsque l'on considère des méthodes d'ordre élevé ainsi qu'une haute résolution spatiale. Une solution serait de considérer une version semi-implicite : $\partial_x(b_{xx}(T^n)\partial_x)T^{n+1}$ et $\partial_y(b_{yy}(T^n)\partial_y)T^{n+1}$. Cette solution sera mise en œuvre dans les cas tests non linéaires présentés en Section 2.5. Enfin, la généralisation de cette méthode à des ordres plus élevés en temps et en espace n'est pas immédiate et nécessite un grand nombre d'étapes supplémentaires.

2.2.2 Méthode de type Additive Runge-Kutta (ARK)

Cette intégration en temps peut être vue comme l'adaptation de la méthode de pénalisation présentée dans [19, 18]. Rappelons tout d'abord la stratégie dans notre contexte. En partant des équations sous forme continue,

$$\frac{\partial T}{\partial t} = \nabla\cdot(B\nabla T),$$

on introduit le paramètre $\lambda \in \mathbb{R}^+$ tel que la plus grande valeur propre $|\mu|$ de la matrice B satisfasse $|\mu| < \lambda$. Ensuite, on introduit l'opérateur $\lambda\Delta$, et on applique la discrétisation en temps suivante (voir [19])

$$\frac{T^{n+1} - T^n}{\Delta t} = \nabla\cdot(B\nabla T^n) - \lambda\Delta T^n + \lambda\Delta T^{n+1}.$$

Le faible coût calculatoire de l'inversion du laplacien rend cette méthode attractive. De plus, ce laplacien est indépendant du temps et de l'espace, ce qui est intéressant du point de vue de la charge de calcul ainsi que des besoins en mémoire pour l'application. La généralisation à un ordre plus élevé en temps peut en outre se faire par la méthode Additive Runge-Kutta (ARK). Avec la notation $T^{(1)} = T^n$, on calcul les valeurs aux étapes $T^{(s)}, s = 2, 3, 4, 5, 6$ en résolvant

$$(I - \Delta t \gamma \lambda \Delta)\, T^{(s)} = T^n + \Delta t \tilde{L},$$

où

$$\tilde{L} = \sum_{j=1}^{s-1} a_{s,j} \left[\nabla \cdot (B\nabla) - \lambda\Delta\right] T^{(s)},$$

avec $\gamma = a_{s,s}$, qui peut être imposé constant pour toutes les étapes. En particulier, nous sélectionnons deux méthodes présentées dans [37] (voir aussi [33, 57]) : la méthode d'ordre 2 en temps (ARK2) correspond à ARK.2.A.1 ($s = 2$) et la méthode d'ordre 4 en temps (ARK4) correspond à ARK4.A.1 ($s = 6$). Pour ces deux méthodes, le terme de diffusion $\tilde{L}$ (et par conséquent $\nabla \cdot (B\nabla T)$) est traité de manière explicite alors que le laplacien Δ est résolu de manière implicite. Le principal avantage est d'avoir à inverser un système creux. D'autres choix sont possibles en ce qui concerne les méthodes ARK, pour obtenir des méthodes dites A-stables ou L-stables.

Remarque 2.2.1 *Comme présenté dans [18, 34, 19], l'opérateur de "pénalisation" offre un large choix de possibilités. Une manière de l'observer est de considérer une méthode (complètement) implicite :*

$$\frac{T^{n+1} - T^n}{\Delta t} = \nabla \cdot \left(B(T^{n+1})\nabla T^{n+1}\right).$$

Un algorithme de point fixe standard peut être donné par, avec $T^{n+1,0} = T^n$:

$$\left[I - \Delta t \nabla \cdot \left(B(T^{n+1,s})\nabla\right)\right] T^{n+1,s+1} = T^n, \quad s \geq 0,$$

jusqu'à ce que le critère $|T^{n+1,s+1} - T^{n+1,s}| < \varepsilon$ soit satisfait pour ε petit donné. Cet algorithme peut s'avérer lourd en charge de calcul.

Une autre stratégie consiste à linéariser le terme non linéaire et à l'approcher ensuite :

$$\begin{aligned}
\nabla \cdot \left(B(T^{n+1})\nabla T^{n+1}\right) &\simeq \nabla \cdot \left(B(T^n)\nabla T^{n+1}\right) \\
&= \nabla \cdot (B(T^n)\nabla T^n) + \nabla \cdot \left(B(T^n)\nabla(T^{n+1} - T^n)\right) \\
&\simeq \nabla \cdot (B(T^n)\nabla T^n) + \nabla \cdot \left(\lambda I\, \nabla(T^{n+1} - T^n)\right) \\
&= \nabla \cdot (B(T^n)\nabla T^n) + \lambda\Delta(T^{n+1} - T^n) \\
&= \nabla \cdot ([B(T^n) - \lambda I]\nabla T^n) + \lambda\Delta T^{n+1}
\end{aligned}$$

Cette stratégie peut être interprétée comme une unique itération d'une méthode de point fixe, dans laquelle le terme non linéaire $B(T^n)$ (matrice de diffusion) a été approchée de manière à obtenir un faible coût en calcul. Nous choisissons ici un laplacien, mais d'autres choix sont possibles. On pourrait par exemple considérer une matrice diagonale dont les coefficients sont non constants. Une étude plus poussée serait intéressante à mener quant au choix de cette approximation (voir [34]). Il est également possible de se référer à [12, 39] pour des études du même type.

Remarque 2.2.2 *On remarque que la combinaison des deux dernières approches (SH et ARK) ne donne pas lieu à une méthode inconditionnellement stable. En effet, la matrice de diffusion B n'étant pas diagonale, il n'est pas possible de stabiliser les termes extra diagonaux avec des termes de la forme $\lambda\partial_{xx}$.*

2.3 Discrétisation en espace

On pose $x_{i+1/2} = x_{\min}+(i+1/2)\Delta x$ avec $\Delta x = (x_{\max}-x_{\min})/N_x$ et $y_{j+1/2} = y_{\min}+(j+1/2)\Delta y$ avec $\Delta y = (y_{\max}-y_{\min})/N_y$, $N_x, N_y \in \mathbb{N}$. On présente deux types de discrétisation spatiale, toutes deux basées sur la méthode des volumes finis. On appelle $T_{i,j}$ la valeur moyenne de l'inconnue sur le volume de contrôle $V_{i,j}$

$$T^n_{i,j} = \frac{1}{\Delta x\Delta y}\int_{V_{i,j}} T(t^n, x, y)dxdy.$$

Ensuite, on intègre l'équation de départ sur un volume de contrôle $V_{i,j} = C^x_i \times C^y_j := [x_{i-1/2}, x_{i+1/2}] \times [y_{j-1/2}, y_{j+1/2}]$ pour obtenir $\partial_t T_{i,j} = \frac{1}{\Delta x\Delta y}\int_{V_{i,j}}[\nabla\cdot(B\nabla T)]dxdy$, ce qui donne

$$\begin{aligned}
\partial_t T_{i,j} &= \frac{1}{\Delta x\Delta y}\int_{C^y_j}[(b_{xx}\partial_x T)(x_{i+1/2}, y) - (b_{xx}\partial_x T)(x_{i-1/2}, y)]dy \\
&+\frac{1}{\Delta x\Delta y}\int_{C^y_j}[(b_{xy}\partial_y T)(x_{i+1/2}, y) - (b_{xy}\partial_y T)(x_{i-1/2}, y)]dy \\
&+\frac{1}{\Delta x\Delta y}\int_{C^x_i}[(b_{yy}\partial_y T)(x, y_{j+1/2}) - (b_{yy}\partial_y T)(x, y_{j-1/2})]dx \\
&+\frac{1}{\Delta x\Delta y}\int_{C^x_i}[(b_{xy}\partial_x T)(x, y_{j+1/2}) - (b_{xy}\partial_x T)(x, y_{j-1/2})]dx \qquad (2.3.2)
\end{aligned}$$

Jusque là, la relation ci-dessus est exacte. Il faut à présent en approcher les différents termes, *i.e.* donner des expressions reliant les moyennes sur les faces des cellules aux valeurs moyennes de l'inconnue sur la cellule $T_{i,j}$. Nous rappelons quelques formules utiles dans l'Annexe A.1 page 149 (voir aussi [57]). Notons

que les volumes finis permettent également d'inclure naturellement les termes d'advection ainsi que de considérer des conditions de bords non périodiques (voir [57]).

La première étape consiste à exprimer l'intégrale d'un produit en fonction d'un produit d'intégrales, en utilisant la proposition A.1.1 (Annexe A.1), à l'ordre désiré.

Ensuite, nous devons traiter différents types de termes. La première ligne de l'Equation (2.3.2) donne lieu à des intégrales de la forme $\int_{C_j^y} \partial_x T(x_{i+1/2}, y)dy$ et $\int_{C_j^y} b_{xx}(x_{i+1/2}, y)dy$. Elles sont respectivement approchées par la proposition A.1.3 et la proposition A.1.2. Le même traitement est appliqué pour la troisième ligne de l'équation, ce qui donne les deux intégrales suivantes : $\int_{C_i^x} \partial_y T(x, y_{j+1/2})dx$ (nécessitant la proposition A.1.3) et $\int_{C_i^x} b_{yy}(x, y_{j+1/2})dx$ (nécessitant la proposition A.1.2). Pour les termes de la seconde ligne, les intégrales de type $\int_{C_j^y} \partial_y T(x_{i+1/2}, y)dy$ sont traitées par la proposition A.1.4 et les termes de type $\int_{C_j^y} b_{xy}(x_{i+1/2}, y)dy$ par la proposition A.1.2. Enfin, la quatrième ligne de (2.3.2) fait apparaître des termes de la forme $\int_{C_i^x} \partial_x T(x, y_{j+1/2})dx$ (qui sont approchés par la proposition A.1.4 page 151) et $\int_{C_i^x} b_{xy}(x, y_{j+1/2})dy$ (approchés par la proposition A.1.2). Il résulte que l'Equation (2.3.2) devient un système d'équations satisfait par $T_{i,j}$, étant donné que le membre de droite de l'équation a été exprimé en fonction des valeurs de l'inconnue $T_{i,j}, i = 0, \ldots, N_x, j = 0, \ldots, N_y$.

Lorsque des problèmes non linéaires sont considérés, il faut évaluer des termes de la forme $\int_{C_j^y} b_{xx}(x_{i+1/2}, y)dy$ où b_{xx} dépend de la valeur moyenne de l'inconnue sur la cellule $T_{i,j}$. Quand b_{xx} est une fonction donnée (dans le cas de diffusion linéaire), il est aisé d'approcher cette intégrale à l'ordre désiré (par des méthodes de quadrature standard). Néanmoins, pour les problèmes non linéaires, cette opération est plus délicate, surtout lorsque l'on désire obtenir un ordre élevé (typiquement l'ordre 4 dans ce chapitre). En s'inspirant du travail réalisé dans la référence [42] dans le cadre de problèmes hyperboliques non linéaires, nous proposons une stratégie pour approcher le terme $\int_{C_j^y} b_{xx}(x_{i+1/2}, y)dy$ from $T_{i,j}$ à l'ordre 4.

Premièrement, à partir de l'égalité (*point to cell*)

$$T_{i,j} = T(x_i, y_j) + \frac{\Delta x^2}{24}(\partial_x^2 T)(x_i, y_j) + \frac{\Delta y^2}{24}(\partial_y^2 T)(x_i, y_j) + \mathcal{O}(\Delta x^4) + \mathcal{O}(\Delta y^4),$$

nous pouvons reconstruire la valeur ponctuelle $T(x_i, y_j)$ à l'ordre 4 en utilisant des différences finies à l'ordre 2, à partir des valeurs moyennes de l'inconnue

sur la cellule (*cell to point*)

$$\begin{aligned} T(x_i,y_j) = \quad & T_{i,j} - \frac{\Delta x^2}{24}\frac{T_{i+1,j}-2T_{i,j}+T_{i-1,j}}{\Delta x^2} - \frac{\Delta y^2}{24}\frac{T_{i,j+1}-2T_{i,j}+T_{i,j-1}}{\Delta y^2} \\ & +\mathcal{O}(\Delta x^4)+\mathcal{O}(\Delta y^4). \end{aligned}$$

Ensuite, le coefficient de diffusion est évalué au point (x_i, y_j) pour obtenir par exemple $b_{xx}(T)(x_i, y_j)$. Une fois les valeurs ponctuelles des coefficients de diffusion obtenus, pour chaque point i, j, nous pouvons reconstruire les valeurs moyennes sur les cellules de ces coefficients $b_{xx}(T)_{i,j}$ à un ordre élevé de précision avec la formule (*point to cell*)

$$\begin{aligned} b_{xx}(T)_{i,j} \quad := \quad & \frac{1}{\Delta x\Delta y}\int_{C_i^x}\int_{C_j^y} b_{xx}(T)(x,y)dxdy \\ = \quad & b_{xx}(T)(x_i,y_j) + \frac{\Delta x^2}{24}\frac{b_{xx}(T)_{i+1,j}-2b_{xx}(T)_{i,j}+b_{xx}(T)_{i+1,j}}{\Delta x^2} \\ & +\frac{\Delta y^2}{24}\frac{b_{xx}(T)_{i,j+1}-2b_{xx}(T)_{i,j}+b_{xx}(T)_{i,j-1}}{\Delta y^2} \\ & +\mathcal{O}(\Delta x^4)+\mathcal{O}(\Delta y^4). \end{aligned}$$

Ensuite, on utilise la proposition A.1.2 page 150 pour déterminer les valeurs moyennes sur les faces des cellules :

$$\int_{C_j^y} b_{xx}(T)(x_{i+1/2},y)dy,$$

comme fonction des valeurs moyennes $b_{xx}(T)_{i,j}$. Remarquons que, pour obtenir une expression exacte à l'ordre 2, nous n'avons pas besoin de ces calculs, étant donné que $b_{xx}(T)_{i,j} = b_{xx}(T)(x_i, y_j) + \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta y^2)$.

Il faut aussi considérer les moyennes sur les cellules des valeurs initiales et finales pour obtenir l'ordre de précision désiré. Par exemple, pour une valeur ponctuelle initiale $T(x_i, y_j)$, on calcule les valeurs moyennes sur les volumes élémentaires :

$$\begin{aligned} T_{i,j} = \quad & T(x_i,y_j) + \frac{\Delta x^2}{24}\frac{T(x_{i+1},y_j)-2T(x_i,y_j)+T(x_{i-1},y_j)}{\Delta x^2} \\ & +\frac{\Delta y^2}{24}\frac{T(x_i,y_{j+1})-2T(x_i,y_j)+T(x_i,y_{j-1})}{\Delta y^2}, \end{aligned}$$

expression correcte à l'ordre 4 de précision.

Remarque 2.3.1 *L'extension du schéma SH à l'ordre 4 en espace supprime la localité dans la direction y pour la première étape du splitting (et en x pour la seconde étape). Ceci complique le traitement implicite dans chaque direction, impliquant l'inversion d'un système 2D.*

Remarque 2.3.2 *Le choix du paramètre λ pour le cas non linéaire est sujet à discussion étant donné que le rayon spectral peut augmenter au fil de la simulation. Nous suivons la stratégie proposée dans [19]. Elle consiste à initialiser λ à deux fois le rayon spectral de la matrice de diffusion initiale, et à le mettre à jour si nécessaire de la même manière (*i.e. *si le rayon spectral de la matrice devient plus grand que λ).*

2.4 Analyse de stabilité linéaire

Dans cette section, nous présentons une analyse de Von Neumann des méthodes numériques introduites plus haut. Pour ce faire, nous considérons un unique mode de Fourier $T(t,x,y) = r(t)\exp(-i(k_x x + k_y y)$ où $r(t)$ est le facteur d'amplification temporel, et k_x, k_y dénotent le nombre d'onde dans les directions x et y respectivement.

Nous nous mettons dans le cas d'une matrice de diffusion B constante en temps et en espace, de la forme

$$B = \begin{pmatrix} b_{xx} & b_{xy} \\ b_{xy} & b_{yy} \end{pmatrix},$$

avec $b_{xx}b_{yy} \geq b_{xy}^2$. Ainsi, le schéma numérique s'écrit, à l'ordre 2 :

$$\begin{aligned} \frac{dT_{i,j}}{dt} &= \frac{1}{\Delta x^2} b_{xx}\left[T_{i+1,j} - 2T_{i,j} + T_{i-1,j}\right] \\ &\quad + \frac{2}{\Delta x \Delta y} b_{xy}\left[T_{i+1,j+1} - T_{i+1,j-1} - T_{i-1,j+1} + T_{i-1,j-1}\right] \\ &\quad + \frac{1}{\Delta y^2} b_{yy}\left[T_{i,j+1} - 2T_{i,j} + T_{i,j-1}\right], \end{aligned}$$

alors qu'à l'ordre 4, on obtient

$$\begin{aligned} \frac{dT_{i,j}}{dt} &= \frac{1}{24\Delta x^2} b_{xx}\left[-T_{i+2,j} + 16T_{i+1,j} - 30T_{i,j} + 16T_{i-1,j} - T_{i-2,j}\right] \\ &\quad + \frac{4}{\Delta x \Delta y} b_{xy}(D_x D_y T)_{i,j} \\ &\quad + \frac{1}{24\Delta y^2} b_{yy}\left[-T_{i,j+2} + 16T_{i,j+1} - 30T_{i,j} + 16T_{i,j-1} - T_{i,j-2}\right], \end{aligned}$$

où la notation $(D_x D_y T)_{i,j}$ est définie dans les Equations (A.1.1) et (A.1.2). Nous détaillons dans la suite les calculs à l'ordre 2. Pour l'ordre 4, nous renvoyons le lecteur à l'Annexe A.2 page 152.

2.4.1 Méthode semi-implicite SH

Nous nous concentrons à la stabilité pour la méthode SH. La première étape du splitting directionnel s'écrit :

$$\begin{aligned} T^{\star}_{i,j} &= T^{n}_{i,j} + \frac{\Delta t}{\Delta x^2} b_{xx} \left[T^{\star}_{i+1,j} - 2T^{\star}_{i,j} + T^{\star}_{i-1,j} \right] \\ &\quad + \frac{\Delta t}{\Delta x \Delta y} b_{xy} \left[T^{n}_{i+1,j+1} - T^{n}_{i+1,j-1} - T^{n}_{i-1,j+1} + T^{n}_{i-1,j-1} \right], \end{aligned}$$

alors que pour la deuxième étape, on a

$$\begin{aligned} T^{n+1}_{i,j} &= T^{\star}_{i,j} + \frac{\Delta t}{\Delta y^2} b_{yy} \left[T^{n+1}_{i,j+1} - 2T^{n+1}_{i,j} + T^{n+1}_{i,j-1} \right] \\ &\quad + \frac{\Delta t}{\Delta x \Delta y} b_{xy} \left[T^{\star}_{i+1,j+1} - T^{\star}_{i+1,j-1} - T^{\star}_{i-1,j+1} + T^{\star}_{i-1,j-1} \right]. \end{aligned}$$

Conformément à l'analyse de Von Neumann, on injecte une solution de type onde plane : $T(t,x,y) = r(t)\exp^{-\mathrm{i}(k_x x + k_y y)}$ pour obtenir le facteur d'amplification provenant de la première étape du splitting

$$r_1 = \frac{1 - \frac{\Delta t}{\Delta x^2} b_{xy} \sin(k_y \Delta y) \sin(k_x \Delta x)}{1 + \frac{4\Delta t}{\Delta x^2} b_{xx} \sin^2(k_x \Delta x/2)}.$$

De la même manière, on obtient le facteur d'amplification r_2 pour la deuxième étape :

$$r_2 = \frac{1 - \frac{\Delta t}{\Delta x^2} b_{xy} \sin(k_y \Delta y) \sin(k_x \Delta x)}{1 + \frac{4\Delta t}{\Delta y^2} b_{yy} \sin^2(k_y \Delta y/2)},$$

ce qui nous permet d'écrire le facteur d'amplification total pour un pas de temps $r = r_1 r_2$. On introduit le nombre de Courant ncfl

$$\Delta t = \mathrm{ncfl}\ \frac{\Delta x^2}{4},$$

de telle sorte que le facteur d'amplification se réécrit, avec $A_x = \sin(k_x \Delta x/2)$ et $A_y = \sin(k_y \Delta y/2)$

$$r := r(\mathrm{ncfl}, k_x \Delta x, k_y \Delta y) = \frac{(1 - (\mathrm{ncfl}/2) b_{xy} A_x A_y \cos(k_x \Delta x/2) \cos(k_y \Delta y/2))^2}{(1 + (\mathrm{ncfl}/2) b_{xx} A_x^2)(1 + (\mathrm{ncfl}/2) b_{yy} A_y^2)}.$$

Démontrons que $|r| \leq 1$

$$\begin{aligned}
|r| &\leq \frac{(1+(\text{ncfl}/2)|b_{xy}||A_x||A_y|)^2}{(1+(\text{ncfl}/2)b_{xx}A_x^2)(1+(\text{ncfl}/2)b_{yy}A_y^2)} \\
&\leq \frac{(1+(\text{ncfl}/2)\sqrt{b_{xx}}\sqrt{b_{yy}}|A_x||A_y|)^2}{(1+(\text{ncfl}/2)b_{xx}A_x^2)(1+(\text{ncfl}/2)b_{yy}A_y^2)} \\
&= \frac{(1+(\text{ncfl}^2/4)b_{xx}b_{yy}A_x^2A_y^2+\text{ncfl}\sqrt{b_{xx}}\sqrt{b_{yy}}|A_x||A_y|)}{(1+(\text{ncfl}^2/4)b_{xx}b_{yy}A_x^2A_y^2+(\text{ncfl}/2)(b_{xx}A_x^2+b_{yy}A_y^2))} \\
&\leq \frac{(1+(\text{ncfl}^2/4)b_{xx}b_{yy}A_x^2A_y^2+(\text{ncfl}/2)(b_{xx}A_x^2+b_{yy}A_y^2))}{(1+(\text{ncfl}^2/4)b_{xx}b_{yy}A_x^2A_y^2+(\text{ncfl}/2)(b_{xx}A_x^2+b_{yy}A_y^2))} = 1,
\end{aligned}$$

ce qui prouve l'inconditionnelle stabilité de la méthode. La Figure 2.1 trace le facteur d'amplification $|r| = |r_1 r_2|$ avec $b_{xx} = b_{yy} = b_{xy} = 1/2$, pour différentes valeurs de ncfl. Comme attendu, on observe que les valeurs obtenues sont inférieures à 1 sur tout le domaine, ce qui illustre la stabilité du schéma. De plus, lorsque ncfl est diminué (ce qui signifie que Δt diminue lui aussi), les graphes se rapprochent du facteur d'amplification exact, tracé dans la Figure 2.2 :

$$\exp[-(b_{xx}k_x^2\Delta x^2 + b_{yy}k_y^2\Delta y^2 + 2b_{xy}k_x\Delta x k_y\Delta y)\text{ncfl}/4].$$

Néanmoins, quand ncfl augmente, l'atténuation dans la direction perpendiculaire n'est pas assez forte (premier et troisième quadrants). Enfin, la Figure 2.3 montre les facteurs d'amplification pour une discrétisation d'ordre 4 en espace. On observe que la propriété de stabilité est conservée. Dans les premier et troisième quadrants, l'effet d'atténuation dans la direction perpendiculaire est plus faible que pour l'ordre 2 en espace. Néanmoins, sur la diagonale (deuxième et quatrième quadrants), le facteur d'amplification est plus proche de la solution analytique, ce qui est une bonne propriété pour un schéma d'ordre élevé.

2.4.2 Méthode Additive Runge Kutta

Cette sous section est dédiée à l'analyse de Von Neumann pour les schémas ARK présentées dans ce chapitre. Nous nous restreignons à l'ordre 1 en temps. Cette analyse s'étend de manière immédiate à n'importe quel intégrateur temporel de type ARK.

Rappelons la méthode numérique :

$$T^{n+1} = T^n + \Delta t\nabla\cdot(B\nabla T^n) - \Delta t\lambda\Delta T^n + \Delta t\lambda\Delta T^{n+1},$$

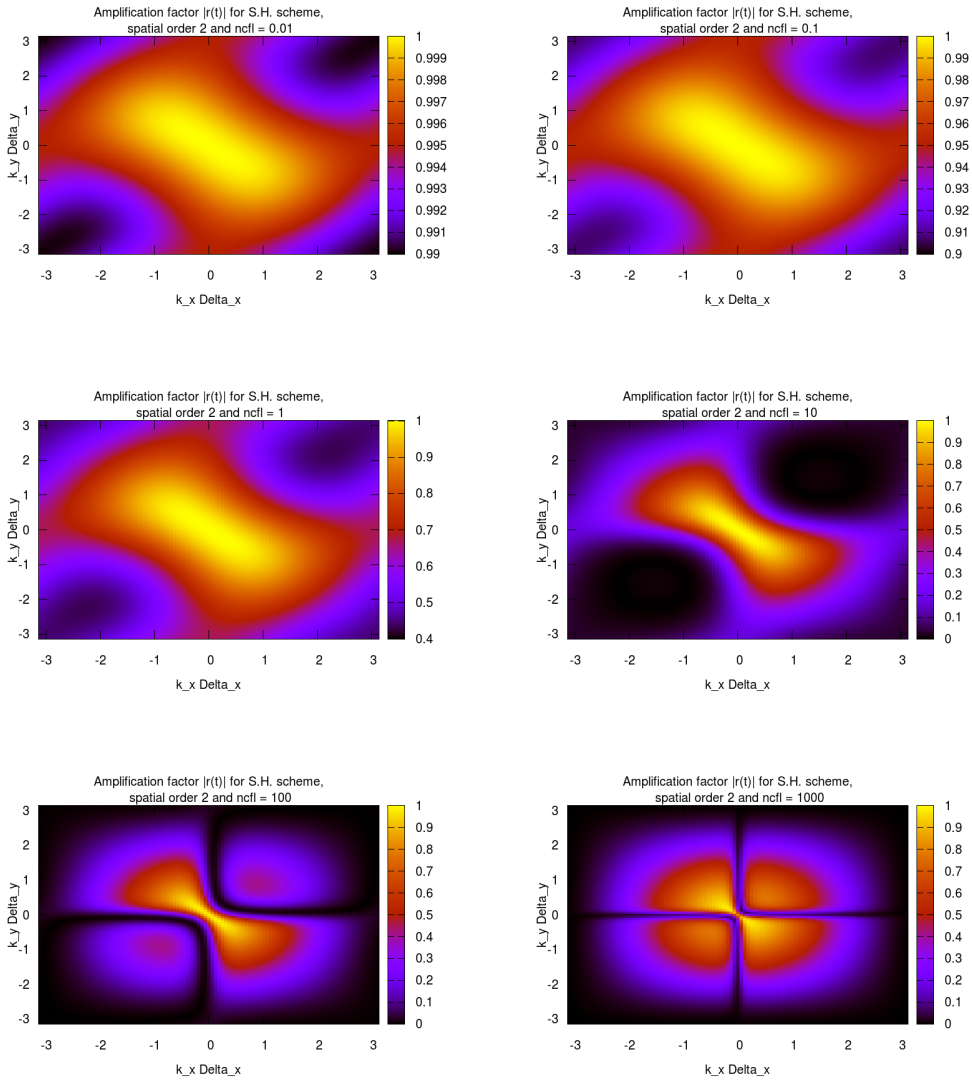


FIGURE 2.1 – Facteur d'amplification $|r(t)|$ pour ncfl$= 0.01, 0.1, 10, 100, 1000$ pour le schéma SH d'ordre 2 en espace.

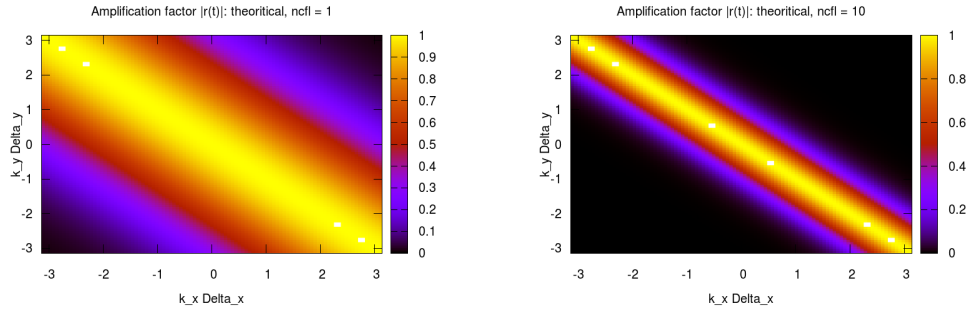


FIGURE 2.2 – Facteur d'amplification exact $|r(t)|$ pour ncfl$= 1, 10$ à l'ordre 2 en espace.

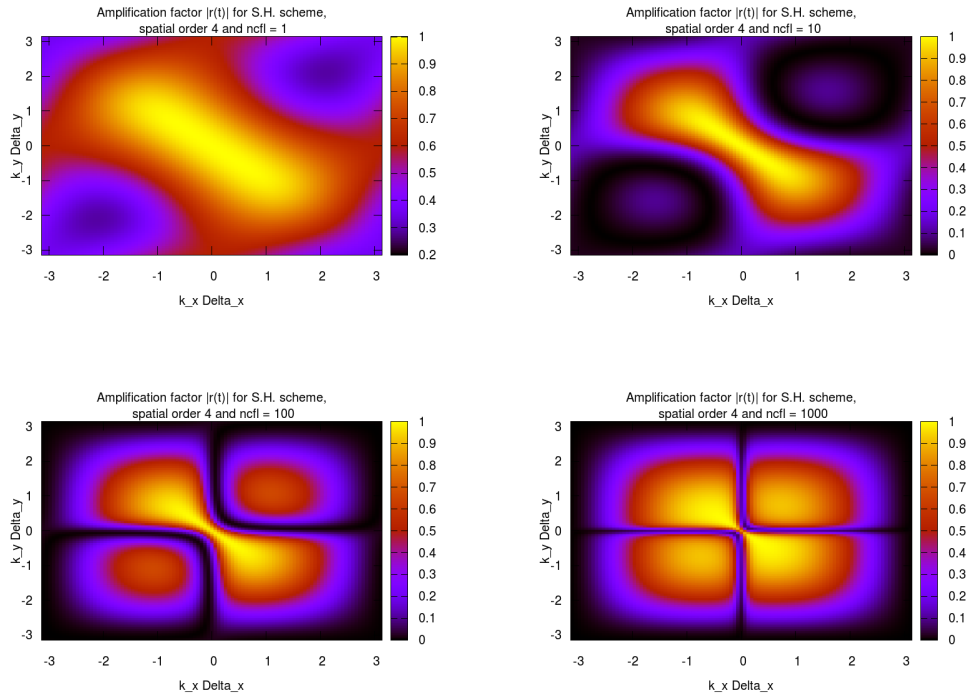


FIGURE 2.3 – Facteur d'amplification $|r(t)|$ pour ncfl$= 1, 10, 100, 1000$ pour le schéma SH d'ordre 4 en espace.

ou encore, de manière plus détaillée

$$\Big[1-\Delta t\lambda(\partial_x^2+\partial_y^2)\Big]T^{n+1}=\Big[1+\Delta t(b_{xx}\partial_x^2+b_{yy}\partial_y^2+2b_{xy}\partial_{xy}^2)-\Delta t\lambda(\partial_x^2+\partial_y^2)\Big]T^n.$$

Le coefficient de pénalisation λ est pris plus grand que le rayon spectral de la matrice de diffusion B, choisie à nouveau constante en temps et en espace. Cela signifie que $\lambda \geq |X_{1,2}|$ où $X_{1,2}$ sont les deux valeurs propres de B

$$X_{1,2}=\frac{1}{2}(\mathrm{Tr}(B)\pm\sqrt{(b_{xx}-b_{yy})^2+4b_{xy}^2}).$$

Nous faisons l'étude à l'ordre 2 en espace, associé à un ordre 1 en temps. La méthode numérique s'écrit

$$\begin{aligned}
& T_{i,j}^{n+1}-\frac{\Delta t\lambda}{\Delta x^2}[T_{i+1,j}^{n+1}+T_{i,j+1}^{n+1}-4T_{i,j}^{n+1}+T_{i-1,j}^{n+1}+T_{i,j-1}^{n+1}] \\
= \quad & T_{i,j}^{n}+\frac{\Delta t(b_{xx}-\lambda)}{\Delta x^2}[T_{i+1,j}^{n}-2T_{i,j}^{n}+T_{i-1,j}^{n}] \\
& +\frac{\Delta t(b_{yy}-\lambda)}{\Delta x^2}[T_{i,j+1}^{n}-2T_{i,j}^{n}+T_{i,j-1}^{n}] \\
& +2\frac{\Delta t b_{xy}}{4\Delta x^2}(T_{i+1,j+1}^{n}-T_{i+1,j-1}^{n}-T_{i-1,j+1}^{n}+T_{i-1,j-1}^{n}).
\end{aligned}$$

En insérant l'onde plane $T(t,x,y)=r(t)\exp^{-\mathrm{i}(k_x x+k_y y)}$, on obtient le facteur $r=r_1/r_2$ (avec ncfl$=(4\Delta t)/\Delta x^2$)

$$\begin{aligned}
r_1 &= 1-\frac{4\Delta t(b_{xx}-\lambda)}{\Delta x^2}\sin^2(k_x\Delta x/2)-\frac{4\Delta t(b_{yy}-\lambda)}{\Delta x^2}\sin^2(k_y\Delta x/2) \\
&\quad -2\frac{\Delta t b_{xy}}{\Delta x^2}\sin(k_x\Delta x)\sin(k_y\Delta x), \\
r_2 &= 1+\frac{4\Delta t\lambda}{\Delta x^2}(\sin^2(k_x\Delta x/2)+\sin^2(k_y\Delta x/2)).
\end{aligned}$$

Si l'on pose $A_x=\sin(k_x\Delta x/2)$ and $A_y=\sin(k_y\Delta y/2)$, on obtient

$$\left|\frac{r_1}{r_2}\right|=\left|1-\frac{\text{ncfl }(b_{xx}A_x^2+b_{yy}A_y^2-2b_{xy}A_xA_y\cos(k_x\Delta x/2)\cos(k_y\Delta x/2))}{1+\text{ncfl }\lambda(A_x^2+A_y^2)}\right|.$$

Il faut maintenant vérifier que $-1<r_1/r_2<1$. L'inégalité $r_1/r_2<1$ est trivialement vérifiée car le numérateur est toujours positif :

$$\begin{aligned}
& b_{xx}A_x^2+b_{yy}A_y^2-2b_{xy}A_xA_y\cos(k_x\Delta x/2)\cos(k_y\Delta x/2) \\
\geq \quad & b_{xx}A_x^2+b_{yy}A_y^2-2b_{xy}A_xA_y,
\end{aligned}$$

et

$$\begin{aligned} b_{xx}A_x^2 + b_{yy}A_y^2 - 2b_{xy}A_xA_y &= \frac{1}{b_{yy}}\left(b_{yy}b_{xx}A_x^2 + b_{yy}^2A_y^2 - 2b_{yy}b_{xy}A_xA_y\right) \\ &\geq \frac{1}{b_{yy}}\left(b_{xy}A_x - b_{yy}A_y\right)^2. \end{aligned}$$

On réécrit l'inégalité $-1 < r_1/r_2$ comme suit :

$$\frac{\text{ncfl } (b_{xx}A_x^2 + b_{yy}A_y^2 - 2b_{xy}A_xA_y\cos(k_x\Delta x/2)\cos(k_y\Delta x/2))}{1 + \text{ncfl } \lambda(A_x^2 + A_y^2)} \leq 2.$$

On approche ensuite le numérateur par

$$\begin{aligned} & \left|b_{xx}A_x^2 + b_{yy}A_y^2 - 2b_{xy}A_xA_y\cos(k_x\Delta x/2)\cos(k_y\Delta x/2))\right| \\ \leq & \left|b_{xx}A_x^2 + b_{yy}A_y^2 + 2\sqrt{|b_{xx}|}\sqrt{|b_{yy}|}A_xA_y\right| \\ \leq & \left|b_{xx}A_x^2 + b_{yy}A_y^2 + b_{xx}b_{yy}A_x^2A_y^2\right| \\ \leq & \ 2\left|b_{xx}A_x^2 + b_{yy}A_y^2\right|, \end{aligned}$$

ce qui nous permet de déduire

$$\frac{\text{ncfl } (b_{xx}A_x^2 + b_{yy}A_y^2)}{1 + \text{ncfl } \lambda(A_x^2 + A_y^2)} \leq 1,$$

relation toujours vérifiée car $\lambda = \max(|X_1|, |X_2|) \geq \max(b_{xx}, b_{yy})$.

Les Figures 2.4 et 2.5 tracent les facteurs d'amplification pour différents nombres ncfl dans le cas de l'ordre 2 et 4 en espace respectivement. Les calculs pour l'ordre 4 sont présentés dans l'Annexe A.2 page 152. De la même manière que pour la méthode SH, on choisit $b_{xx} = b_{yy} = b_{xy} = 1/2$.

Comme attendu, le facteur d'amplification est toujours plus petit que 1, ce qui traduit le caractère inconditionnellement stable des méthodes ARK aux ordres 2 et 4 en espace. Enfin, la Figure 2.6 trace les facteurs d'amplification pour la combinaison des méthodes ARK et SH. Il apparaît que, pour l'exemple que nous considérons, la méthode n'est pas inconditionnellement stable (sauf dans le cas ncfl≤ 1).

2.5 Cas tests

Dans cette section, nous proposons différents cas tests pour valider les méthodes numériques que nous avons décrit dans les sections précédentes. Ces

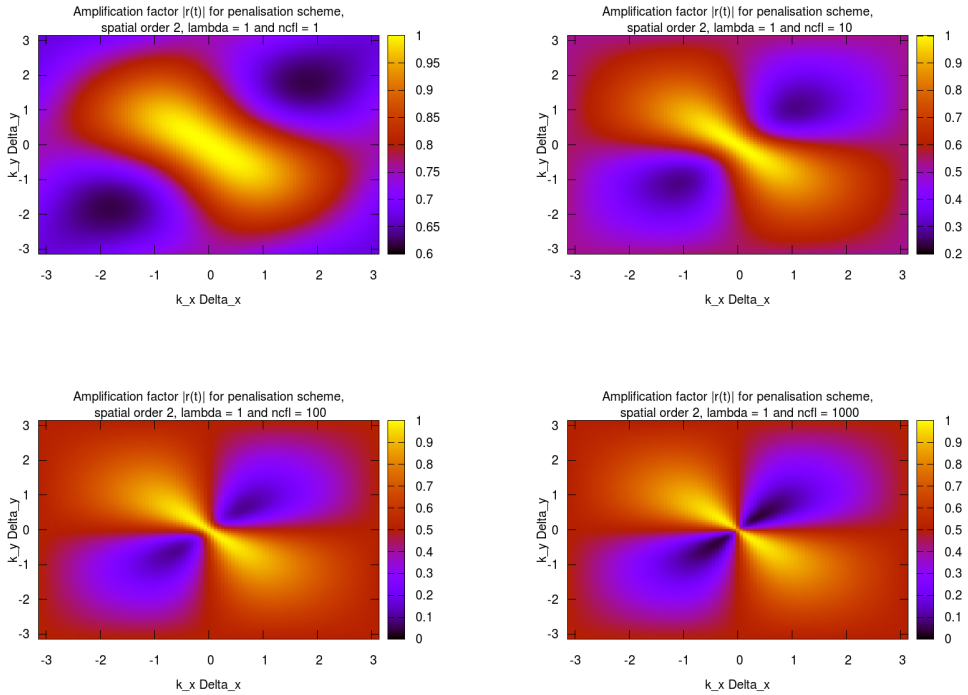


FIGURE 2.4 – Amplification factor $|r(t)|$ for ncfl$= 1, 10, 100, 1000$ for ARK scheme of order 2 in space.

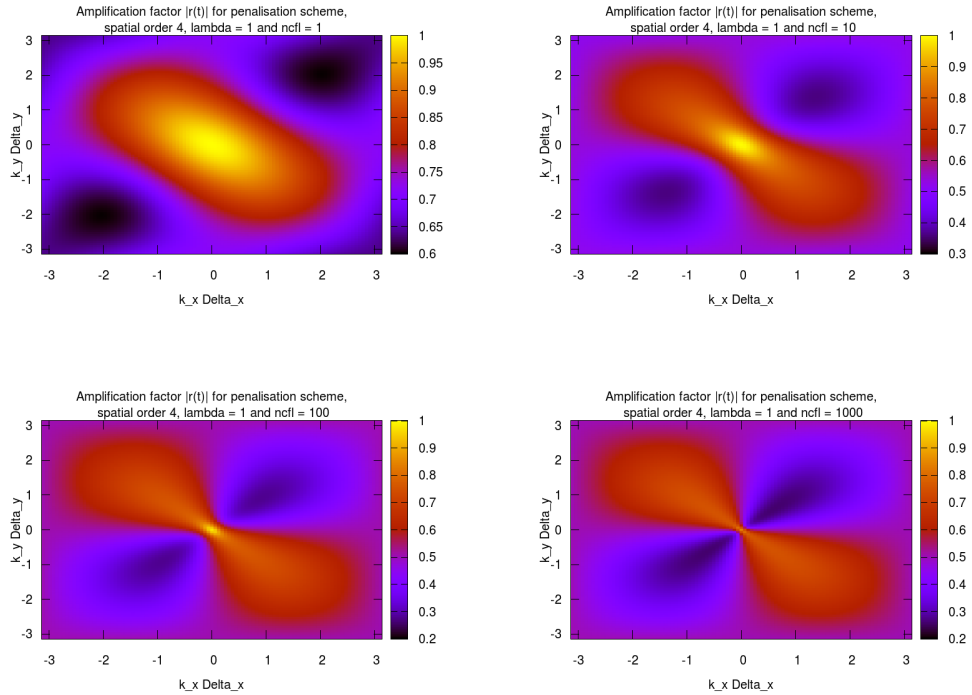


FIGURE 2.5 – Facteur d'amplification $|r(t)|$ pour ncfl$= 1, 10, 100, 1000$ pour le schéma ARK à l'ordre 4 en espace.

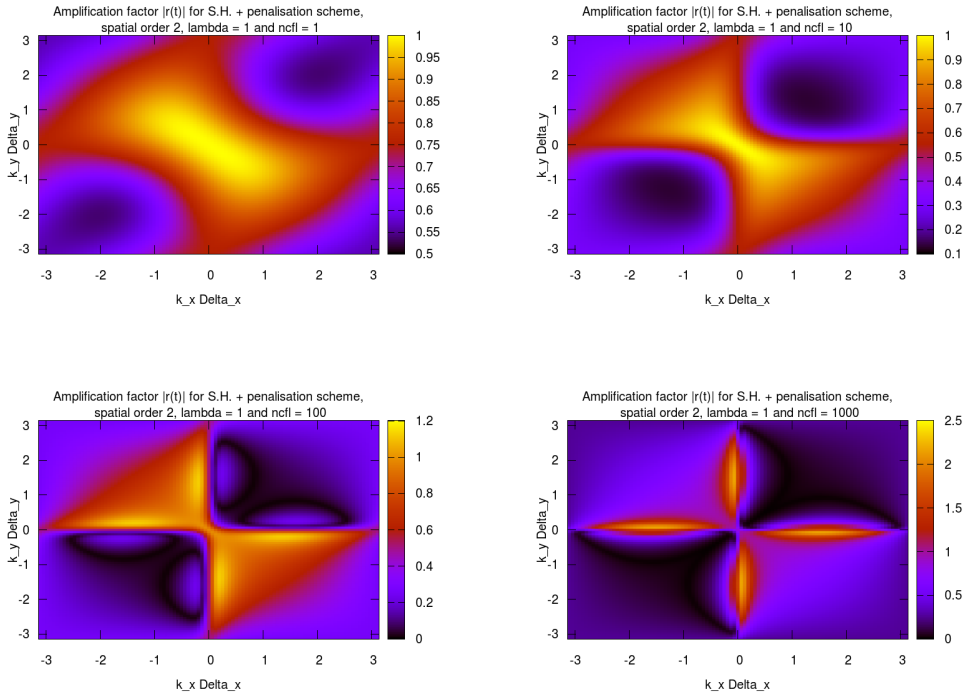


FIGURE 2.6 – Facteur d'amplification $|r(t)|$ pour ncfl$= 1, 10, 100, 1000$ pour le schéma "SH-ARK" à l'ordre 2 en espace.

tests peuvent être regroupés en deux catégories : les cas tests analytiques et les cas tests physiques. Les tests analytiques ont pour but l'observation des erreurs de manière quantitative (en magnitude et en ordre de précision), alors que les tests physiques visent ensuite à évaluer les méthodes d'un point de vue qualitatif. Dans la suite, les cas présentés sont de difficulté croissante, incluant des problèmes non constants en espace ainsi qu'en temps. Les tests présentés sont 2D en espace, mais la généralisation de ceux-ci au cas 3D est immédiate pour la plupart.

Notre intention dans ce qui suit est d'observer l'amélioration qu'entraîne la mise en place d'une discrétisation d'ordre 4 en espace couplée à un ordre élevé en temps sur l'erreur de diffusion commise dans la direction perpendiculaire aux lignes de champs. Nous allons également souligner les avantages qu'offrent l'utilisation de méthodes semi-implicites (*e.g.* SH, ARK2 ou ARK4), en observant le gain potentiel sur la valeur du pas de temps Δt. Nous verrons que ce gain doit être équilibré avec l'erreur en magnitude et en ordre de précision, car augmenter Δt peut modifier le comportement de l'erreur commise.

Dans cette section, Δt désigne le pas de temps, Δx le pas en espace sur un maillage N_x^2, et on définit Δt par :

$$\Delta t = \text{ncfl}\Delta x^2/(4\rho(B)), \tag{2.5.3}$$

où B est la matrice de diffusion, $\rho(B)$ le rayon spectral de B et ncfl le nombre de Courant, qui sera déterminé en fonction du cas test considérés.

2.5.1 Cas test analytique : diffusion constante en temps et en espace

On considère la solution analytique $T(t,x,y) = e^{-10t}\sin(\pi x)\cos(\pi y)$, $x, y \in [-1,1], t \geq 0$. Avec cette solution, on calcule $\nabla \cdot (B\nabla T)$ où

$$B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix},$$

ce qui donne :

$$\nabla \cdot (B\nabla T) = \partial_x^2 T + 2\partial_{x,y}^2 T + \partial_y^2 T = -2\pi^2 T - 2\pi^2 e^{-10t}\cos(\pi x)\sin(\pi y).$$

Donc, $T(t,x,y)$ est solution de

$$\partial_t T = \nabla \cdot (B\nabla T) + Q,$$

avec $Q(t,x,y) = -(10 + 2\pi^2)T(t,x,y) - 2\pi^2 e^{-10t}\cos(\pi x)\sin(\pi y)$.

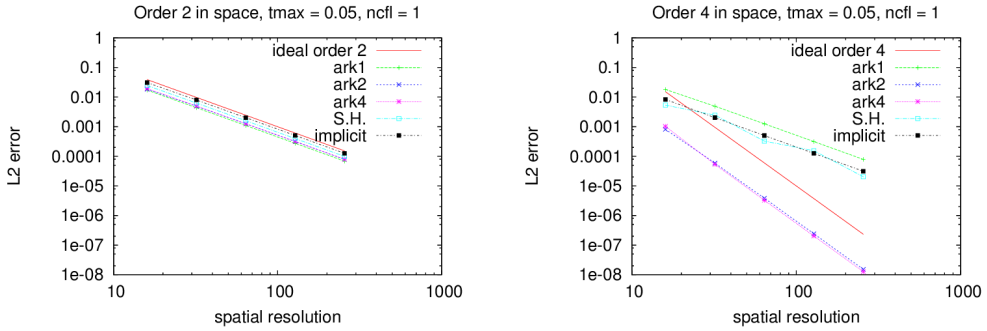


FIGURE 2.7 – Ordres d'erreur pour les schémas implicites et semi-implicites pour ncfl = 1 couplés aux ordres 2 (gauche) et 4 (droite) en espace.

Nous comparons les différentes méthodes proposées dans ce chapitre. Pour toutes ces méthodes, nous observons l'erreur en norme L_2 de la différence entre la solution analytique et la solution numérique. Ces erreurs évoluent en $\mathcal{O}(\Delta t^p + \Delta x^q)$ où p est l'ordre en temps et q l'ordre en espace. Un point important à observer est l'erreur totale en fonction de Δt. En particulier, lorsque Δt est de l'ordre de Δx^2, prendre $p = 2$ et $q = 4$ produit une erreur totale d'ordre 4. Par contre, si $\Delta t = \mathcal{O}(\Delta x)$, l'ordre 4 est perdu pour ces mêmes valeurs de p et q, produisant une erreur totale en $\mathcal{O}(\Delta x^2)$. Pour pallier à ce problème, il faudrait employer une méthode d'ordre 4 en temps.

Ce phénomène est illustré par la Figure 2.7, qui montre les ordres d'erreurs pour ncfl = 1 à $t_{\max} = 0.05$ pour les différent schémas, à l'ordre 2 et 4 en espace. On rappelle que les schémas temporels sont d'ordre 1 pour ARK1, SH et implicite, d'ordre 2 pour ARK2 et d'ordre 4 pour ARK4. Les erreurs affichées correspondent à la norme L_2 de la différence entre la solution analytique et la solution numérique. Le graphe de gauche, correspondant à un ordre 2 en espace, montre que les erreurs sont à l'ordre 2, quelque soit l'ordre de précision du schéma temporel considéré. Ces résultats sont cohérent pour notre problème de diffusion car $\Delta t^p = \mathcal{O}(\Delta x^{2p})$ (voir l'Equation (2.5.3) avec ncfl = 1). Ici, cela donne $\Delta t^p = \mathcal{O}(\Delta x^2)$, indépendamment de la valeur de $p \in \mathbb{N}^*$ choisie. Le graphe de droite, correspondant à un ordre de précision 4 en espace, affiche des résultats d'ordre 2 pour les schémas temporels d'ordre 1, alors que les schémas temporels d'ordre 2 et 4 produisent des résultats à l'ordre 4 pour l'erreur totale. A nouveau, la relation $\Delta t^p = \mathcal{O}(\Delta x^{2p})$ avec $p = 1$ confirme que, avec un schéma temporel d'ordre 1, nous ne sommes capable d'obtenir qu'un ordre 2 sur l'erreur totale produite par les méthodes numériques.

Ensuite, un autre phénomène intéressant à observer est l'impact du nombre de Courant ncfl sur l'erreur de magnitude ainsi que sur son ordre. La condition de stabilité pour notre problème de diffusion dans le cas d'un schéma temporel

de type explicite s'écrit :

$$\Delta t < \frac{1}{4\rho(B)} \times \Delta x^2.$$

Utiliser des valeurs de Δt plus grandes que cette condition de stabilité (*i.e.* ncfl > 1) est possible dans le cas de méthodes implicite ou semi-implicite. Cela entraîne toutefois l'augmentation de l'erreur de magnitude, et peut provoquer la dégradation de l'ordre global de la méthode. Dans la suite, on assume que $\Delta x = \Delta y = L/N_x$ avec L la longueur du domaine et N_x le nombre de points dans chaque direction. On considère à présent l'erreur $\mathrm{Err}(N) := ||T^N - T(t_{\max})||$ au temps $t_{\max} = N\Delta t$ (où N est le nombre d'itérations temporelles). Partant du principe que le schéma temporel est d'ordre p et que le schéma en espace est d'ordre q, on a, par l'Equation (2.5.3)

$$\mathrm{Err}(N) = \mathcal{O}(\Delta t^p + \Delta x^q) = \mathcal{O}\left(\left(\mathrm{ncfl} \times \frac{\Delta x^2}{4\rho(B)}\right)^p + \Delta x^q\right).$$

A présent, si nous voulons obtenir une erreur totale à l'ordre de précision q, il faut satisfaire la relation :

$$\left(\mathrm{ncfl} \times \frac{\Delta x^2}{4\rho(B)}\right)^p \leq C\Delta x^q \Leftrightarrow \frac{\mathrm{ncfl}}{4C^{1/p}\rho(B)} \leq \left(\Delta x^{(q-2p)}\right)^{1/p},$$

où C est une constante positive. Remplacer Δx par sa valeur et poser $\mathrm{KCFL} = \mathrm{ncfl}/(4C^{1/p}\rho(B))$ donne :

$$\mathrm{KCFL} \leq \left(\left(\frac{L}{N_x}\right)^{(q-2p)}\right)^{1/p}.$$

Les Tableaux 2.1, 2.2 et 2.3 montrent les valeurs maximales pour KCFL qui préservent respectivement l'ordre 1, 2 et 4 pour l'erreur totale. Ces tableaux nous montrent clairement que plus la résolution spatiale et l'ordre du schéma temporel sont élevés, plus le nombre de Courant ncfl peut être choisi grand. Considérons deux exemples :

- Pour préserver un ordre de précision 4 avec $N_x = 128$, on regarde la colonne correspondant à $N_x = 128$ dans le Tableau 2.3. Un ordre 4 en temps permet d'utiliser un nombre $\mathrm{KCFL} = 64$, alors qu'un ordre 2 ne permet que d'utiliser $\mathrm{KCFL} = 1$. De plus, un ordre 1 en temps restreint KCFL à approximativement 2×10^{-4}, donnant lieu à un pas de temps très petit. Cette observation motive l'utilisation d'ordres élevés en temps pour les méthodes numériques.

TABLE 2.1 – KCFL maximum to preserve order 1 in space. Here, $L = 2$, N_x is the number of mesh points in both directions and n denotes the time scheme order.

	$N_x = 16$	$N_x = 32$	$N_x = 64$	$N_x = 128$	$N_x = 256$	$N_x = 512$
n= 1	8.0	16.0	32.0	64.0	128.0	256.0
n= 2	22.6	64.0	181.0	512.0	1448.2	4096.0
n= 4	38.1	128.0	430.5	1448.2	4871.0	16384.0

TABLE 2.2 – KCFL maximum to preserve order 2 in space. Here, $L = 2$, N_x is the number of mesh points in both directions and n denotes the time scheme order.

	$N_x = 16$	$N_x = 32$	$N_x = 64$	$N_x = 128$	$N_x = 256$	$N_x = 512$
n= 1	1.0	1.0	1.0	1.0	1.0	1.0
n= 2	8.0	16.0	32.0	64.0	128.0	256.0
n= 4	22.6	64.0	181.0	512.0	1448.2	4096.0

– A présent, si nous regardons la deuxième ligne du Tableau 2.2 (schéma temporel d'ordre 2), dans lequel nous cherchons à préserver un ordre 2 pour l'erreur totale commise, on voit que multiplier le nombre de points dans chaque direction par 2 permet de doubler la valeur de KCFL. Plus intéressant encore, la troisième ligne, correspondant à un schéma temporel d'ordre 4, nous indique qu'il est possible de (presque) tripler la valeur de KCFL dans cette configuration lorsque l'on double le nombre de points dans chaque direction. Ceci motive à nouveau l'emploi de schémas temporels d'ordre élevés et souligne également les avantages qu'offrent les méthodes semi-implicites.

TABLE 2.3 – KCFL maximum to preserve order 4 in space. Here, $L = 2$, N_x is the number of mesh points in both directions and n denotes the time scheme order.

	$N_x = 16$	$N_x = 32$	$N_x = 64$	$N_x = 128$	$N_x = 256$	$N_x = 512$
n= 1	1.6e-02	3.9e-03	9.8e-04	2.4e-04	6.1e-05	1.5e-05
n= 2	1.0	1.0	1.0	1.0	1.0	1.0
n= 4	8.0	16.0	32.0	64.0	128.0	256.0

Cette dernière information sur le nombre KCFL maximal préservant les ordres d'erreurs désirés sont vérifiés par le cas test analytique que nous considérons dans cette sous section. La Figure 2.8 montre le même type de graphes que la Figure 2.7, avec une précision d'ordre 4 en espace, et pour différents nombres de Courant ncfl. A nouveau, on observe la norme L2 de la différence entre la solution analytique et la solution numérique, pour les différentes méthodes mises en place. Pour ncfl = 1, on voit que les méthodes ARK2 et ARK4 produisent un résultat précis à l'ordre 4, alors que les autres méthodes en temps, d'ordre 1, produisent une erreur précise à l'ordre 2. Lorsque l'on augmente ncfl, on observe que l'ordre de précision des méthodes d'ordre 1 en temps se dégrade plus vite que pour les méthodes ARK2 et ARK4. De plus, on observe également que, dans la plupart des cas, augmenter le nombre de points du maillage augmente l'ordre de précision de l'erreur commise. Globalement, on obtient de bons résultats jusque ncfl = 100 avec un schéma d'ordre 4 en espace et au moins 2 en temps.

En conclusion de ce test, nous avons clairement validé nos méthodes sur un cas test analytique pour une diffusion constante en temps et en espace. En outre, nous avons motivé l'utilisation de méthodes numériques d'ordre élevé en temps de manière à pouvoir utiliser des valeurs de pas de temps plus élevées, tout en gardant le contrôle sur l'erreur supplémentaire que cela implique.

2.5.2 Cas test analytique : évaluation de l'erreur de diffusion perpendiculaire numérique

Nous considérons ici un second cas test analytique, dont le but est de mesurer l'erreur de diffusion numérique produite pour les différentes méthodes introduites dans ce chapitre. Cette erreur est produite par la diffusion parallèle aux lignes de champs. Elle est amplifiée par l'anisotropie de la diffusion, et peut venir polluer des termes de transport ou de diffusion perpendiculaire additionnels (voir la Sous Section 1.2.3 page 34).

L'idée principale pour ce test est d'initialiser la température T et le champ magnétique (b_x, b_y) de telle sorte que l'opérateur spatial de diffusion s'annule, donnant lieu à une température analytiquement constante en temps. Néanmoins, les méthodes approchées que nous utilisons produisent une erreur de diffusion numérique dans la plupart des cas. Il en résulte que l'erreur de magnitude observée entre la solution analytique et la solution numérique prend la forme d'une diffusion, que l'on qualifie de numérique et que les physiciens souhaiteraient la plus basse possible.

On considère le domaine périodique $[-1, 1] \times [-1, 1]$ et l'équation à résoudre est

$$\partial_t T = \nabla \cdot (B \nabla T),$$

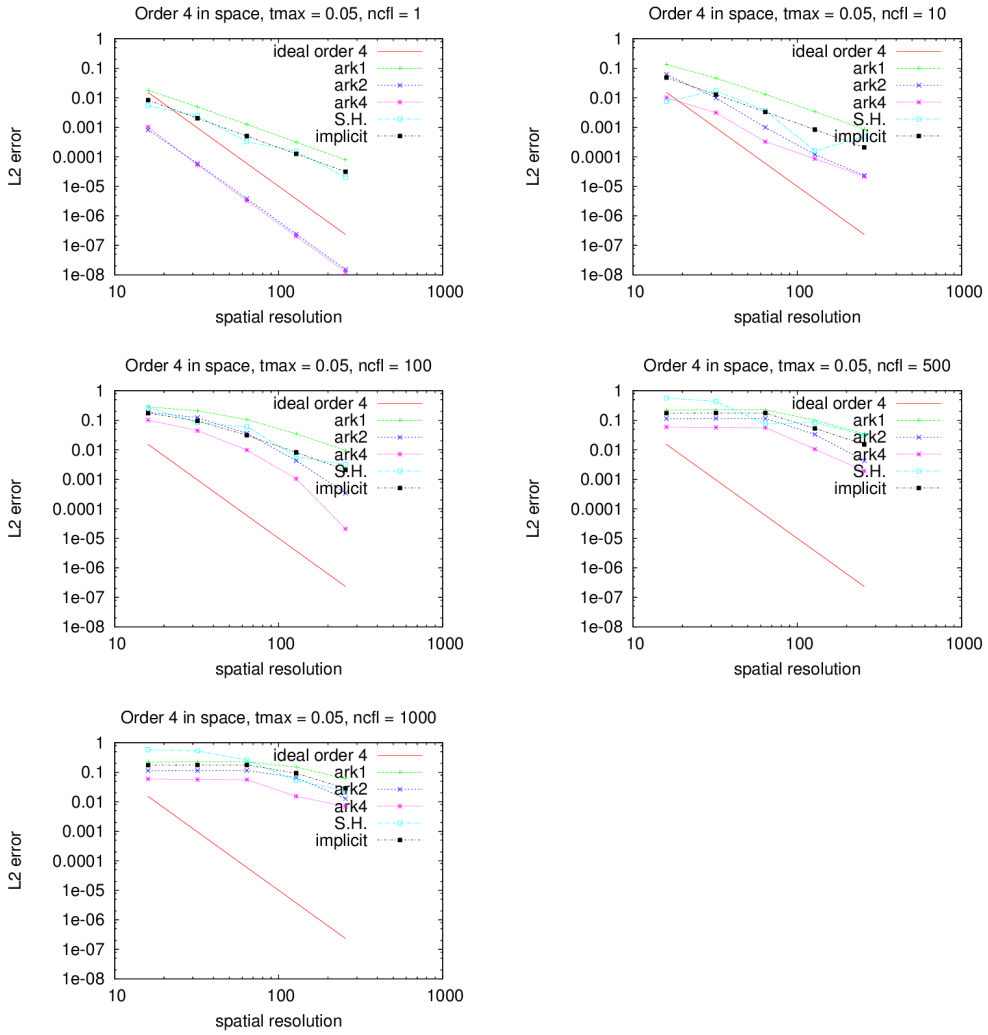


FIGURE 2.8 – Ordres d'erreur pour les méthodes semi-implicites et implicite en temps, pour ncfl $= 1, 10, 100, 500$ et 1000, avec une méthode d'ordre 4 en espace.

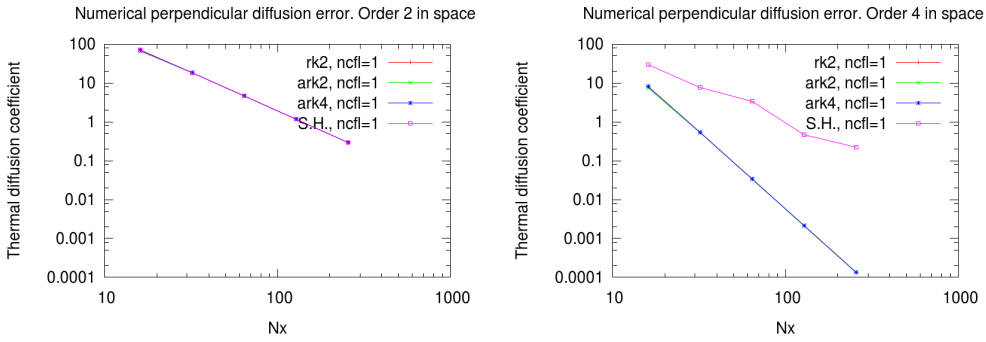


FIGURE 2.9 – Coefficient de diffusion thermique numérique : ordres 2 (gauche) et 4 (droite) en espace.

avec $B = \vec{b}\vec{b}$ constante en espace et en temps

$$\vec{b}\vec{b} = \begin{pmatrix} b_x^2 & b_x b_y \\ b_x b_y & b_y^2 \end{pmatrix}.$$

L'état initial pour la température est donné par :

$$T(t=0,x,y) = \cos(m\pi x - n\pi y) = \cos(\mu x - \nu y)$$

où $\mu = m\pi$ et $\nu = n\pi$.

Déterminons à présent B telle que $\nabla \cdot (B\nabla T) = 0$:

$$\begin{aligned}
\nabla \cdot (B\nabla T) &= b_x^2\partial_x^2 \cos(\mu x - \nu y) + b_y^2\partial_y^2 \cos(\mu x - \nu y) + 2b_x b_y \partial_x\partial_y \cos(\mu x - \nu y) \\
&= -b_x^2\mu^2 \cos(\mu x - \nu y) - b_y^2\nu^2 \cos(\mu x - \nu y) + 2b_x b_y \mu\nu \cos(\mu x - \nu y) \\
&= -\cos(\mu x - \nu y)\left((b_x\mu)^2 - 2b_x\mu b_y\nu + (b_y\nu)^2\right) \\
&= -T(b_x\mu - b_y\nu)^2
\end{aligned}$$

Donc, prendre $b_x = \nu$ et $b_y = \mu$ donne $\nabla \cdot (B\nabla T) = 0$. Dans les simulations pour la fusion nucléaire, il est commun de définir le facteur de sécurité $q(r)$ comme le rapport m/n, variant typiquement de 1 au centre du plasma à 3 ou 4 aux bords. Pour ce test, nous considérons $m = 2$ et $n = 1$, ce qui donne un facteur de sécurité $q = 2$. Comme nous ne voulons observer que la diffusion numérique, nous ne considérons qu'une unique valeur ncfl $= 1$.

La Figure 2.9 montre les courbes de coefficient de diffusion thermique pour les schémas temporels RK2, ARK2, ARK4 et SH, pour différentes tailles de maillage et pour des ordres de précision en espace 2 et 4. Rappelons que la température est censée rester constante au fil du temps, ce qui implique que

les coefficients de diffusion thermique traduisent l'erreur numérique commise. A l'exception de la méthode SH à l'ordre 4 en espace, on observe que les coefficients de diffusion thermique sont indépendants du schéma temporel choisi. On observe aussi clairement l'amélioration induite par l'ordre 4 en espace par rapport à l'ordre 2 en espace, qui donne lieu à des valeurs beaucoup plus faibles en terme de magnitudes ainsi qu'à un ordre de précision plus élevé. De plus, la méthode temporelle SH se révèle moins bonne que les autres méthodes lorsque couplée à l'ordre 4 en espace (voir la courbe violette du graphe de droite), la méthode SH étant d'ordre 1, ce qui est insuffisant pour tirer bénéfice de l'ordre 4 en espace.

Nous avons motivé par ce cas test l'emploi de méthodes numériques d'ordre 4 en espace, couplées à des méthodes d'ordre de précision au moins 2 en temps. Elles permettent ici de limiter fortement l'erreur de diffusion numérique produite par le diffusion parallèle aux lignes de champs.

2.5.3 Diffusion sur un anneau

Dans cette partie, on considère un cas test physique, où l'opérateur de diffusion (et donc la matrice de diffusion) est non constante en espace. On définit $r = \sqrt{x^2 + y^2}$. Le champ magnétique est définit par

$$(b_x, b_y) = \left(\frac{-y}{r}, \frac{x}{r}\right)$$

et l'état initial pour la température est donné par :

$$T(t = 0, x, y) = 0.1 + 10e^{-((x-0.6)^2+y^2)/0.02}.$$

Dans ce cas, la température est censée se stabiliser au cours du temps, vu que les lignes de champs magnétique décrivent des cercles concentriques fermés.

Ce cas test est aussi pris en compte dans la référence [53]. Nous changeons principalement la condition initiale. En effet, notre schéma à l'ordre 4 nécessite que la dérivée seconde de la température T soit définie en tout point du maillage pour éviter des problèmes d'oscillations (qui apparaît si l'on utilise une condition initiale de type Dirac, comme dans [53]). En second lieu, le coefficient de diffusion parallèle $\chi_\parallel$ est fixé à 1 dans notre cas, contre 10^{-2} dans le travail [53]. Pour ce cas test, nous avons effectué des simulations de référence en utilisant une méthode temporelle RK2 classique à deux temps finaux : $t_{\max} = 0.2$ et 5 (voir la Figure 2.10).

La Figure 2.11 montre les résultats pour ce cas test à $t_{\max} = 0.2$, pour les méthodes semi-implicites et implicite en temps et pour différents nombres ncfl. La taille de grille employée est 256×256. Notons que, pour $\chi_\parallel = 1$, atteindre $t_{\max} = 0.2$ est équivalent à effectuer le même test que dans le travail [53] avec

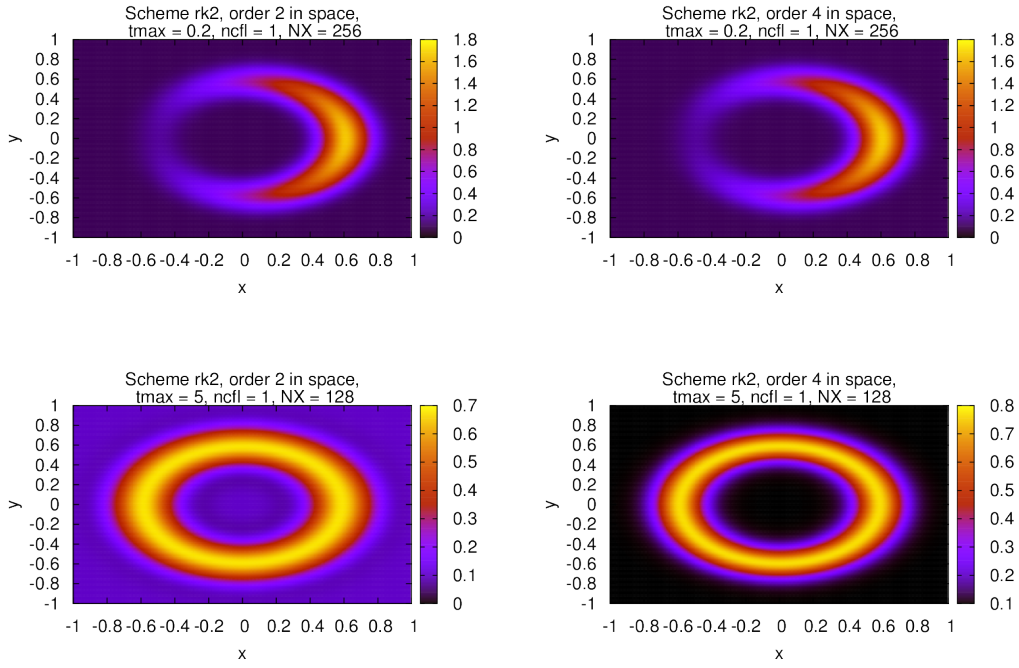


FIGURE 2.10 – Diffusion référence sur un anneau à $t_{max} = 0.2$ et 5, schéma temporel RK2.

$t_{\max} = 20$ et $\chi_{\parallel} = 0.01$ (étant donné qu'il n'y a pas de diffusion perpendiculaire explicite). Nos résultats pour les différentes méthodes temporelles sont très proches des simulations de références (voir la Figure 2.10). On observe la même forme de "croissant" pour l'allure de la température dans le plan (x, y) que celles observées dans [53], ce qui nous permet de valider les méthodes que nous avons mises en place dans le cas d'une matrice de diffusion non constante en espace. Néanmoins, les résultats pour les méthodes de pénalisation / ARK apparaissent comme plus sensibles à l'augmentation du nombre de Courant ncfl (voir la Figure 2.11 pour ncfl = 1000) : la vitesse de diffusion est plus lente. Une explication possible pour cette dernière observation est l'influence du paramètre additionnel λ intervenant dans la méthode de pénalisation sur l'erreur globale commise. Notons tout de même que le schéma ARK4 est plus proche de la solution de référence que la méthode ARK2, motivant l'emploi d'une méthode d'ordre élevé en temps pour la méthode de pénalisation. On remarque également sur ces figures que les ordres 2 et 4 en espace donnent des résultats très proches en terme de formes et de valeurs pour la température. Ceci est du à la faible valeur de t_{max}, qui ne permet pas à l'erreur de diffusion numérique de se développer.

La Figure 2.12 montre les résultats pour le même cas test à une valeur $t_{\max} = 5$ (équivalent à $t_{\max} = 500$ pour $\chi_{\parallel} = 0.01$ dans [53]), pour ncfl = 20. Il est maintenant possible d'observer le bénéfice de la méthode numérique en espace d'ordre 4 couplée à une méthode en temps d'ordre plus grand que 1. En effet, on observe à présent que l'erreur diffusion numérique produite par la diffusion parallèle aux lignes de champs est moins importante lorsque l'on emploie des méthodes spatiales d'ordre 4 pour les schémas ARK2 et ARK4. L'anneau qu'on observe est plus fin, indiquant un meilleur contrôle de l'erreur dans la direction perpendiculaire aux lignes de champs. On note aussi que la valeur maximale de la température est plus grande pour les méthodes d'ordre 4 en espace (proche de 0.8). Le même résultat est observé sur les simulations de références sur la Figure 2.10. Ces améliorations ne sont pas observables pour les schémas en temps SH et implicite qui sont d'ordre de précision 1 (ne permettant pas l'observation de l'ordre 4 en espace, comme montré dans la Sous Section 2.5.1).

Enfin, les Figures 2.13 et 2.14 montrent les résultats à ncfl = 1000 fixé, avec une taille de maillage qui augmente. Ces graphes confirment l'observation faites en Sous Section 2.5.1 en ce qui concerne l'influence de ncfl sur l'erreur de magnitude. En effet, on voit que de hautes valeurs ncfl ne sont atteignables que si la taille de grille est assez grande pour compenser l'erreur additionnelle en temps, produite par la (grande) valeur de Δt.

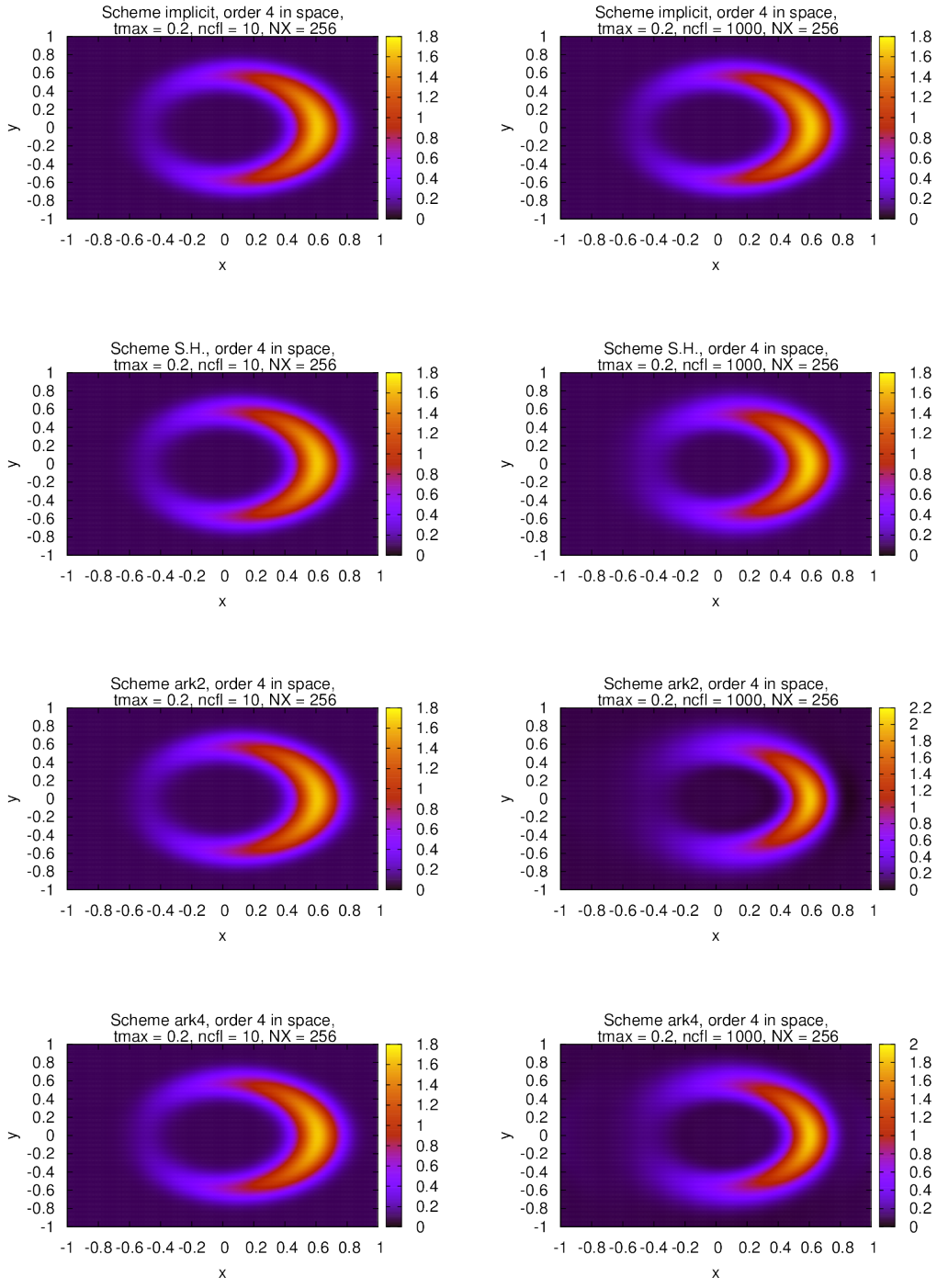


FIGURE 2.11 – Diffusion sur un anneau à $t_{\max} = 0.2$, schémas temporels implicite et semi-implicites, avec $N_x = 256$ et ncfl $= 10$ et 1000.

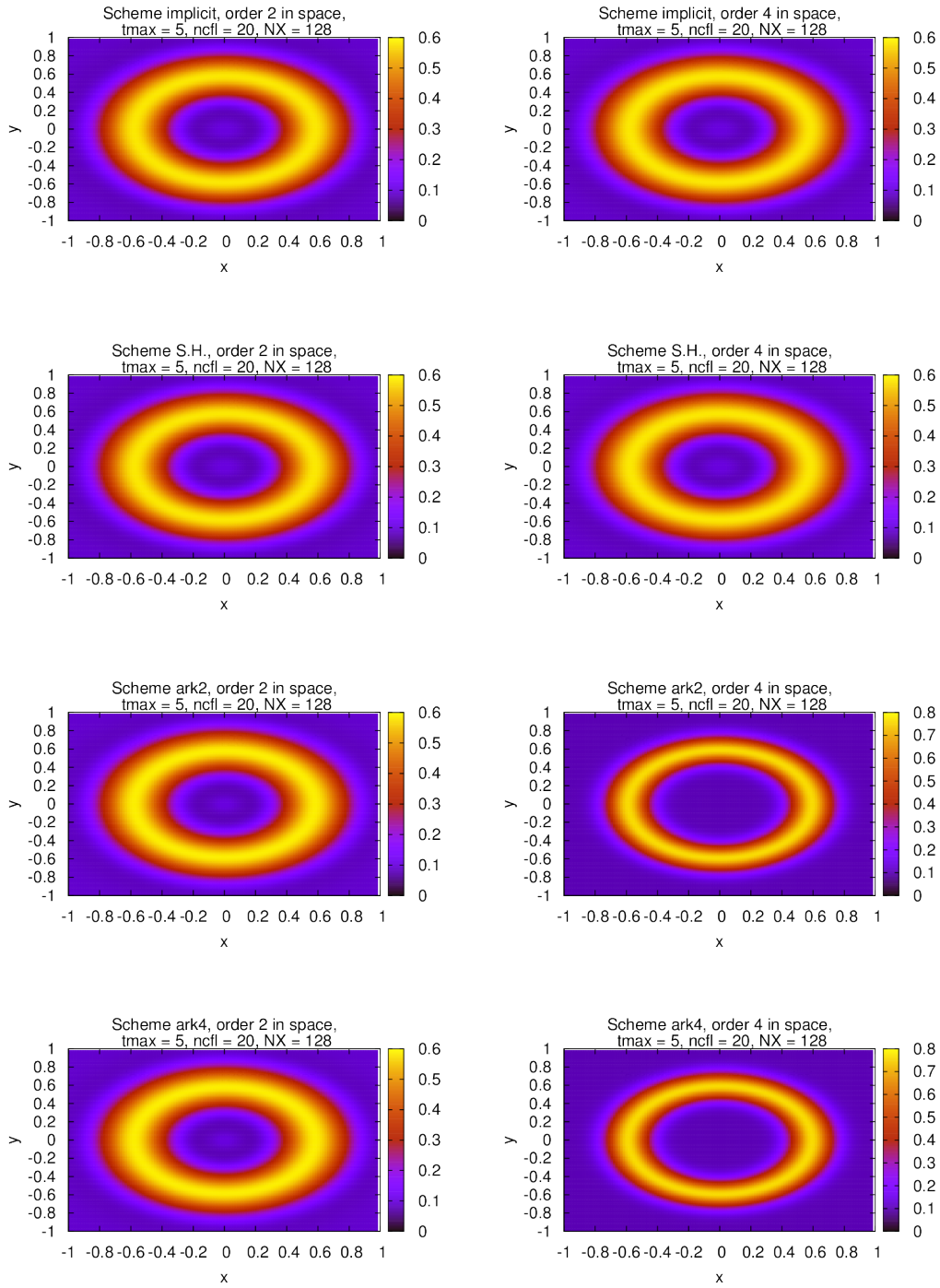


FIGURE 2.12 – Diffusion sur un anneau $t_{\max} = 5$, pour différents schémas temporels, avec $N_x = 128$ et ncfl $= 20$.

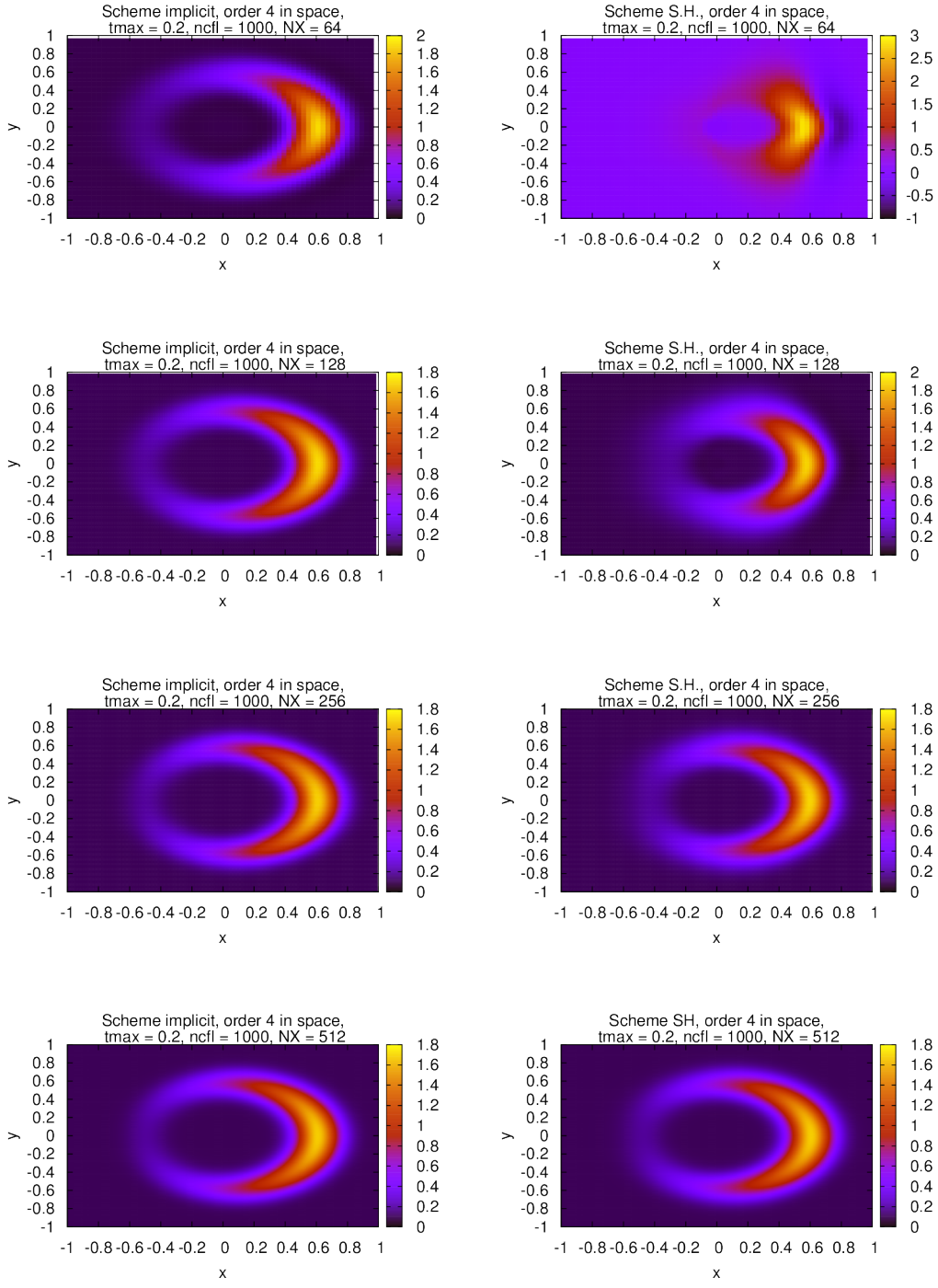


FIGURE 2.13 – Diffusion sur un anneau $t_{\max} = 0.2$, schémas temporels implicite et SH, avec $N_x = 64, 128, 256$ et ncfl= 1000 fixé.

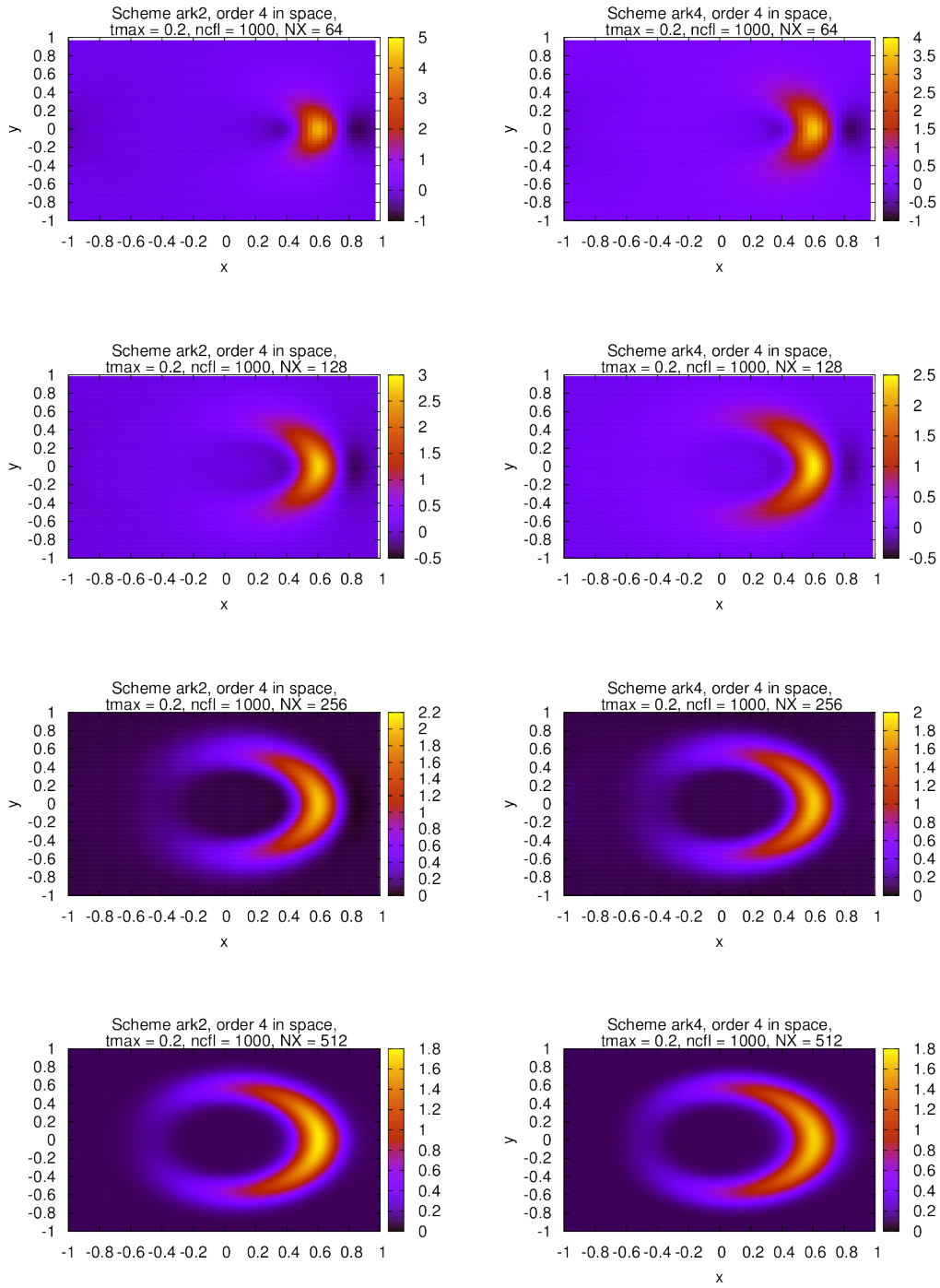


FIGURE 2.14 – Diffusion sur un anneau $t_{\max} = 0.2$, schémas temporels ARK2 et ARK4, avec $N_x = 64, 128, 256$ et ncfl= 1000 fixé

2.5.4 Diffusion constante sur une bande périodique

Ce cas test a pour objectif de valider et comparer les différentes méthodes présentées précédemment sur un cas test linéaire avec une matrice de diffusion constante en espace et en temps. Ceci peut être vu comme un préambule des tests non linéaires présentés par la suite puisque l'allure et la dynamique de la solution est semblable dans les deux configurations.

Le domaine spatial est $[-1,1]\times[-1,1]$ et l'équation à résoudre s'écrit

$$\partial_t T = \nabla\cdot(B\nabla T),$$

avec $B = \overrightarrow{bb}$ et

$$\overrightarrow{bb} = \begin{pmatrix} b_x^2 & b_x b_y \\ b_x b_y & b_y^2 \end{pmatrix}.$$

où $(b_x, b_y) = (1, \frac{1}{2})$. La condition initiale est donnée par

$$T(t=0,x,y) = \begin{cases} 1+3e^{-2r^2} & \text{if } r < \frac{2\pi}{5}, \\ 1 & \text{sinon,} \end{cases}$$

où $r = \sqrt{x^2+y^2}$.

Des conditions périodiques aux bords sont considérées dans les directions x et y. Ainsi, la température se propage au cours du temps sur une bande diagonale périodique.

Ce cas test permet d'observer l'erreur de diffusion perpendiculaire numérique. En effet, sans cette erreur, la solution est supposée atteindre une état stationnaire, devenant constante le long des lignes de champ magnétique. Cependant, la diffusion perpendiculaire intervient à cause des erreurs numériques du schéma en espace. Pour ce cas, la solution de référence est donnée par un schéma classique explicite RK2, pour deux temps finaux $t_{\max} = 0.2$ et 5 (voir Figure 2.15).

Pour ce test, les conclusions sont proches de celles faites pour le test de l'anneau (Section 2.5.3). Les graphes pour les simulations en temps courts (Figure 2.16) sont très proches de la solution de référence en terme d'allure et de magnitude (voir la Figure 2.15).

De plus, l'utilisation de l'ordre quatre est clairement valorisée sur la Figure 2.17, où l'on peut voir que les simulations en temps longs sont moins diffusifs dans la direction perpendiculaire. Dans ce cas (ordre quatre en espace), l'ordre en temps doit être au moins deux.

2.5.5 Cas test analytique : diffusion non linéaire

On considère le domaine spatial $[-0.5,0.5]\times[-0.5,0.5]$ et l'équation à résoudre est

$$\partial_t T = \nabla\cdot(B(T)\nabla T) + Q,$$

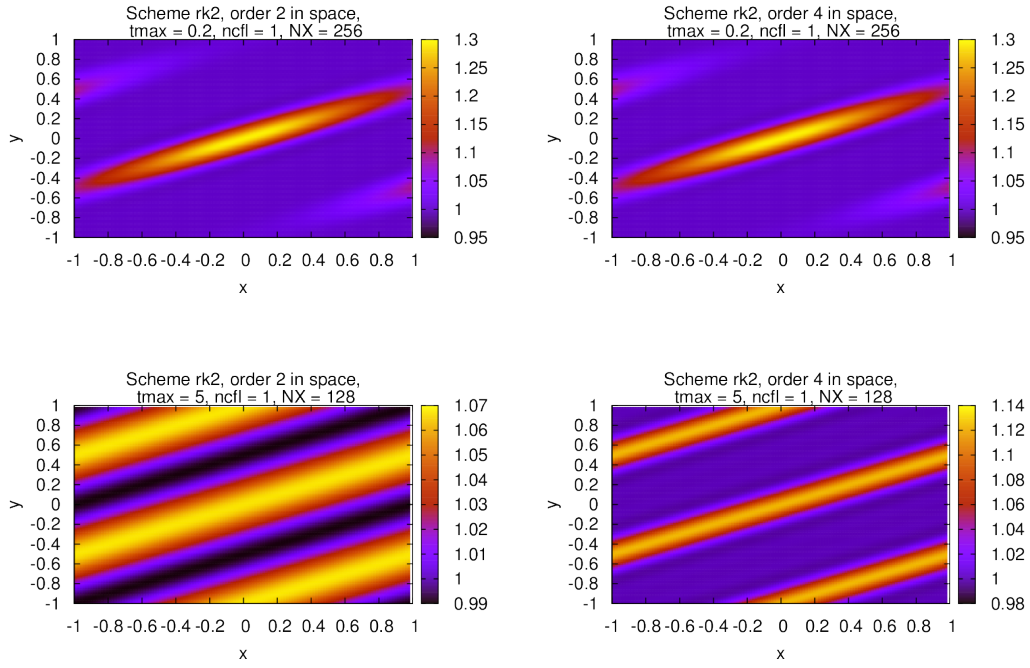


FIGURE 2.15 – Diffusion sur une bande périodique à $t_{\max} = 0.2$ et 5, avec le schéma RK2 en temps.

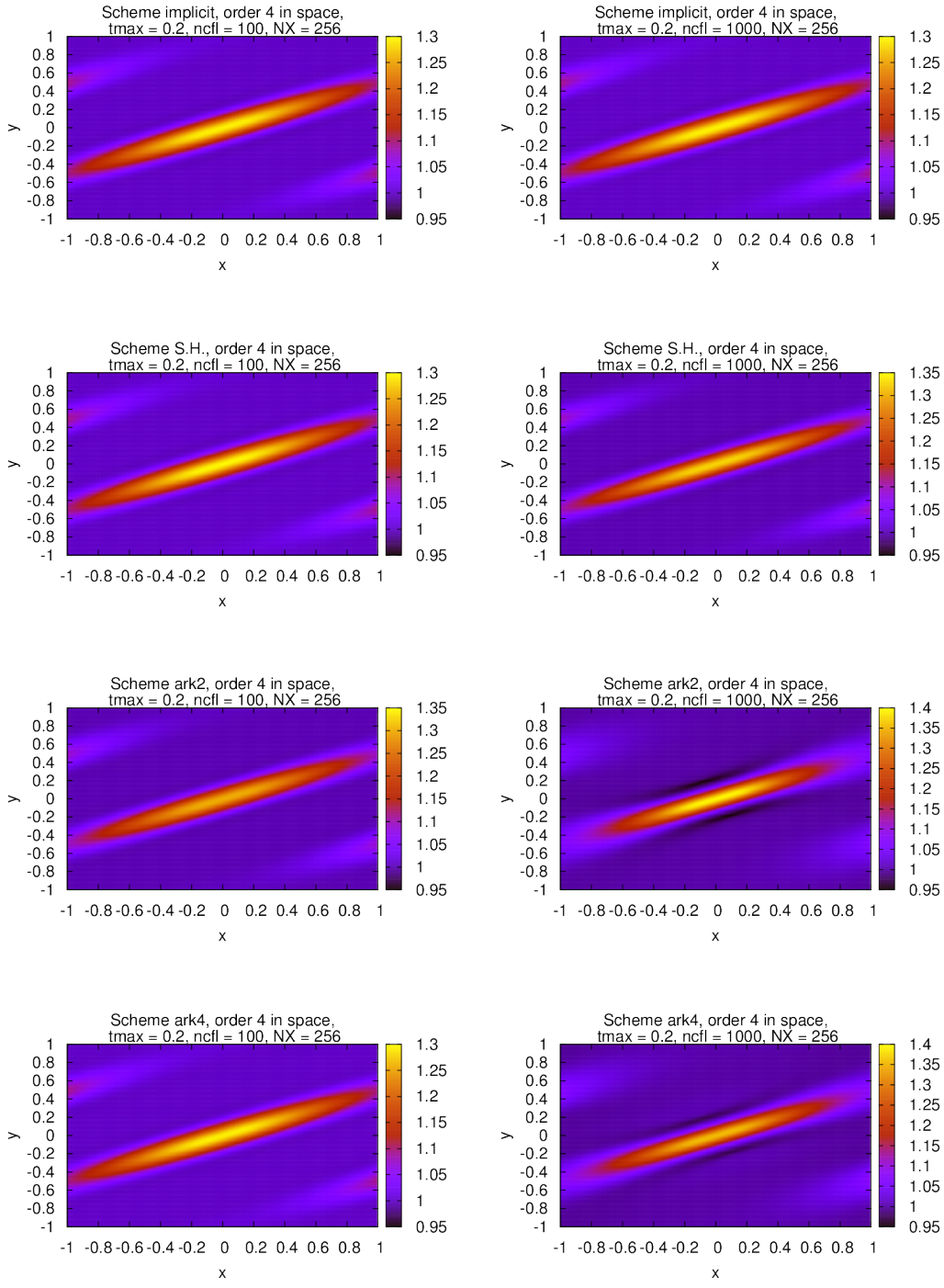


FIGURE 2.16 – Diffusion sur un anneau à $t_{\max} = 0.2$, pour les schémas semi-implicite et implicite, avec $N_x = 256$, ncfl= 100 et 1000.

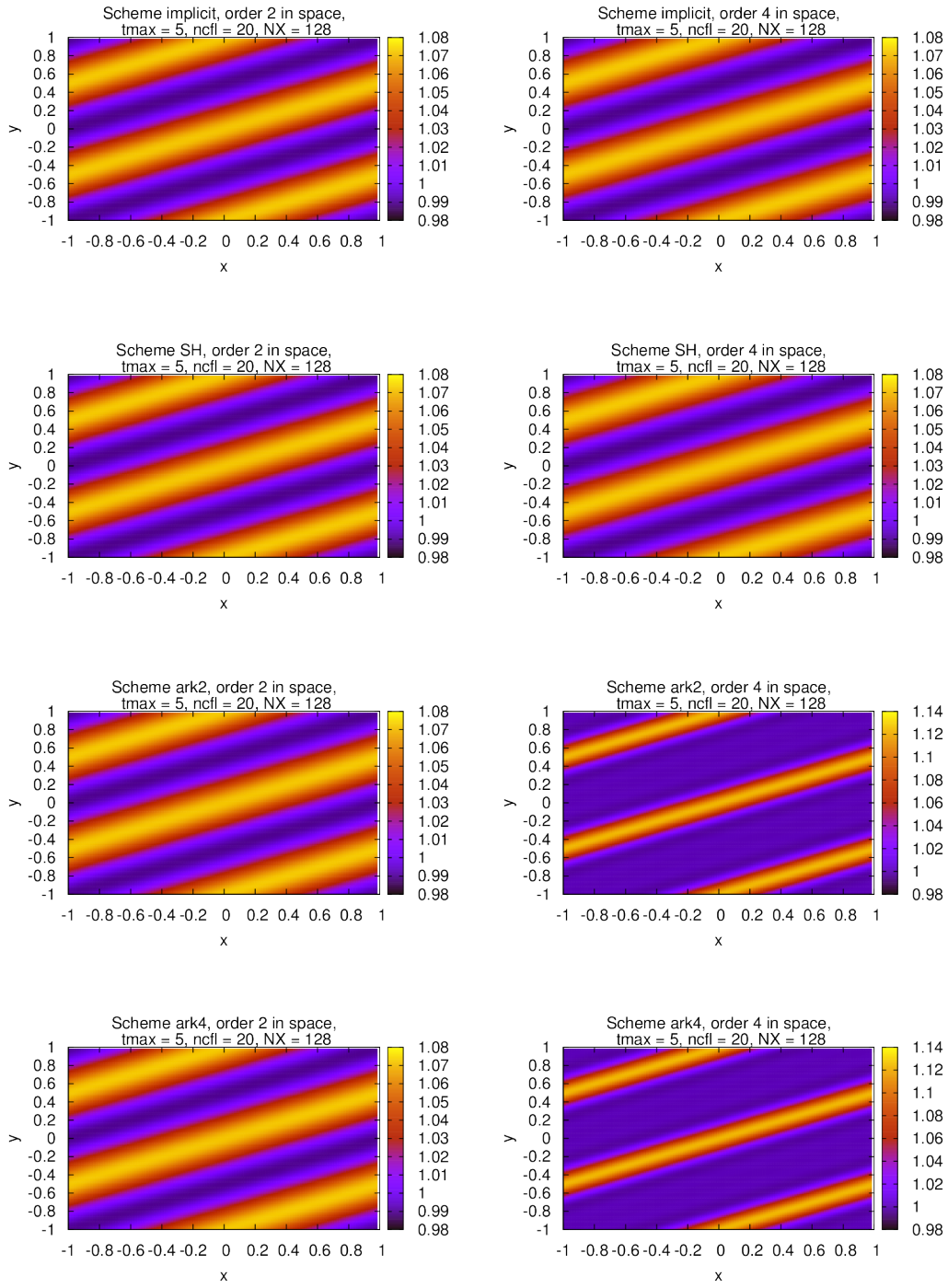


FIGURE 2.17 – Diffusion sur une bande périodique à $t_{\max} = 5$, pour différents schémas temporels, avec $N_x = 128$ et ncfl $= 20$.

avec $B(T) = \frac{T^{5/2}}{\epsilon}\overrightarrow{bb} + (I - \overrightarrow{bb})$ et

$$\overrightarrow{bb} = \begin{pmatrix} b_x^2 & b_x b_y \\ b_x b_y & b_y^2 \end{pmatrix},$$

où $\vec{b} = (b_x, b_y) = (1, \frac{1}{2})$, et Q est un terme source déterminé plus bas. La condition initiale est la suivante

$$T(t, x, y) = c_1 + c_2 \left(\sin(2\pi x) + \epsilon \cos(2\pi x) \sin(2\pi y)\right) e^{-c_3 t},$$

qu'on injecte dans l'équation. Le calcul de $\nabla \cdot (B(T)\nabla T)$ donne

$$\begin{aligned} \nabla \cdot (B(T)\nabla T) = & \quad \tfrac{5}{2\epsilon} T^{\frac{3}{2}} (\partial_x T)^2 + \tfrac{1}{\epsilon} T^{\frac{5}{2}} \partial_x^2 T \\ + & \quad \tfrac{5}{2\epsilon} T^{\frac{3}{2}} \partial_x T \partial_y T + \left(\tfrac{1}{\epsilon} T^{\frac{5}{2}} - 1\right) \partial_x \partial_y T \\ + & \quad \tfrac{1}{4} \left(\tfrac{5}{2\epsilon} T^{\frac{3}{2}} (\partial_y T)^2 + \left(\tfrac{1}{\epsilon} T^{\frac{5}{2}} - 3\right) \partial_y^2 T\right) \end{aligned}$$

avec

$$\begin{aligned} \partial_x T = & \quad -2\pi c_2 \epsilon \sin(2\pi x) \sin(2\pi y) e^{-c_3 t} \\ \partial_y T = & \quad 2\pi c_2 \cos(2\pi y)(1 + \epsilon \cos(2\pi x)) e^{-c_3 t} \\ \partial_x^2 T = & \quad -4\pi^2 c_2 \epsilon \cos(2\pi x) \sin(2\pi y) e^{-c_3 t} \\ \partial_y^2 T = & \quad -4\pi^2 c_2 \sin(2\pi y)(1 + \epsilon \cos(2\pi x)) e^{-c_3 t} \\ \partial_x \partial_y T = & \quad -4\pi^2 c_2 \epsilon \sin(2\pi x) \cos(2\pi y) e^{-c_3 t}, \end{aligned}$$

ainsi, on définit le terme source $Q(t, x, y) = -\nabla \cdot (B(T(t, x, y))\nabla T(t, x, y))$ permettant d'avoir une solution analytique.

Plus qu'un cas de validation des méthodes numériques dans un cadre non linéaire, ce cas test permet aussi de mesurer la diffusion numérique perpendiculaire produite par la diffusion parallèle. En effet, il est possible d'étudier la diffusion parallèle en introduisant le petit paramètre ϵ (typiquement $\epsilon = 10^{-3}$, 10^{-6} or 10^{-9}, voir [25, 38]). Remarquons que dans les travaux traitant le même problème de diffusion numérique perpendiculaire, c'est plutôt l'influence du rapport $\chi_\perp/\chi_\parallel$ qui est étudié. On considère ici une formulation normalisée avec $\chi_\perp = 1$ et $\epsilon = \chi_\parallel$, ce qui revient à considérer le même problème.

Pour les cas test, on ne considérera que les schémas ARK2 et ARK4. En effet, les méthodes de pénalisation se sont avérées plus performantes en termes de temps d'exécution lorsqu'elles sont couplées aux volumes finis d'ordre quatre. La partie implicite à inverser est la même à chaque itération, elle peut donc être inversée une fois pour toute.

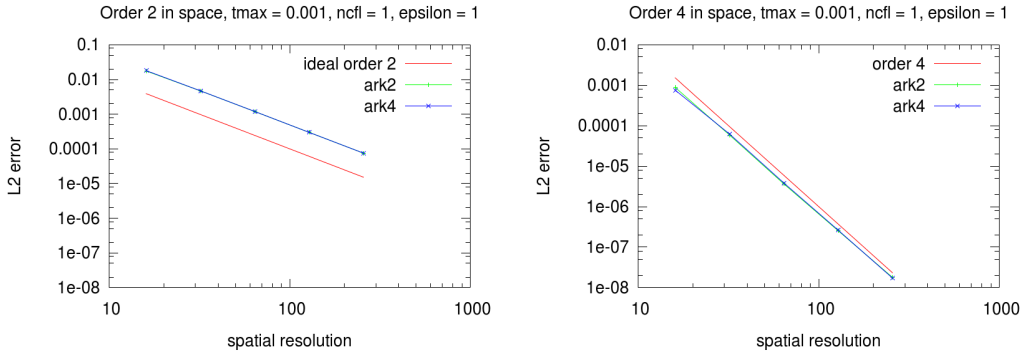


FIGURE 2.18 – Erreur pour ARK2 et ARK4, ncfl = 1, avec l'ordre 2 et 4 en space et $\epsilon = 1$.

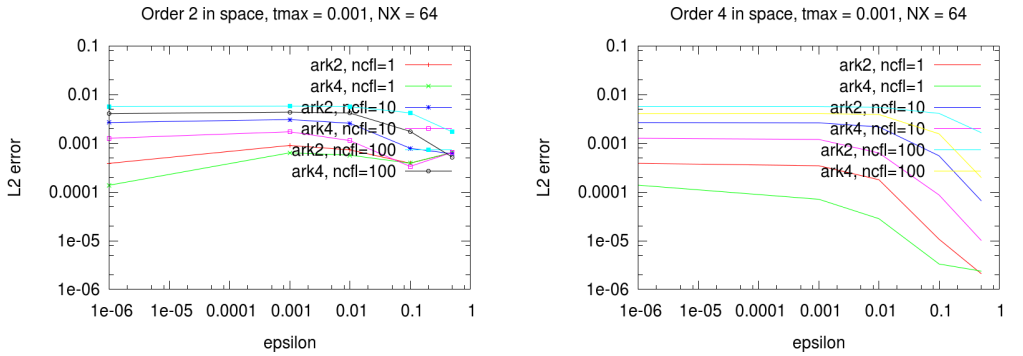


FIGURE 2.19 – Error our ARK2 et ARK4, ncfl = 1, 10 et 100 avec l'ordre 2 et 4 en espace et différentes valeurs de ϵ.

La Figure 2.18 montre les erreurs en norme L_2 pour l'ordre deux et l'ordre quatre. Les deux graphes montrent clairement que les ordres deux et quatre sont obtenus, validant l'implémentation des méthodes sur ce cas non linéaire.

La Figure 2.19 montre l'erreur en fonction du paramètre ϵ pour ARK2 et ARK4 en temps, avec l'ordre deux et quatre en espace, pour différentes valeurs du paramètre ncfl. Ici, le pas de temps Δt est choisi indépendamment de ϵ. Ainsi, les schémas considérés sont clairement inconditionnellement stables. La version à l'ordre quatre est meilleure en terme d'amplitude de l'erreur. On peut remarquer que nos schémas ne possèdent pas la propriété Asymptotic Preserving [38, 43], mais l'objectif ici est de construire de schémas efficaces pour une anisotropie fixée, typiquement $\chi_\perp/\chi_\parallel < 10^3$.

2.5.6 Diffusion non linéaire sur une bande périodique

Ce cas est très proche de celui présenté dans la Section 2.5.4, excepté qu'on considère ici la même matrice de diffusion non linéaire que dans la Section 2.5.5. Le domaine spatial est $[-1,1]\times[-1,1]$ et l'équation à résoudre est

$$\partial_t T = \nabla\cdot(B(T)\nabla T) + Q,$$

avec $B(T) = \frac{T^{5/2}}{\epsilon}\vec{b}\vec{b} + (I - \vec{b}\vec{b})$ et

$$\vec{b}\vec{b} = \begin{pmatrix} b_x^2 & b_x b_y \\ b_x b_y & b_y^2 \end{pmatrix},$$

où $(b_x, b_y) = (1, \frac{1}{2})$. La condition initiale est

$$T(t=0,x,y) = \begin{cases} 1 + 3e^{-2r^2} & \text{if } r < \frac{2\pi}{5}, \\ 1 & \text{sinon.} \end{cases}$$

où $r = \sqrt{x^2+y^2}$.

Des conditions périodiques sont considérés dans les directions x et y, ainsi la température se propage au cours du temps sur une bande diagonale périodique. La Figure 2.20 montre les résultats obtenus pour différentes valeurs de ϵ avec le schéma en temps ARK2, $N_x = 128, t_{\max} = 0.005$ et ncfl $= 1$. Sur ces graphes, on observe que la valeur de ϵ détermine la vitesse de la diffusion parallèle, comme attendu. En effet, pour $\epsilon = 0.01$, la bande de forte température est plus diffusée que celle obtenue pour $\epsilon = 0.5$ par exemple. On peut aussi voir que la bande de diffusion devient plus grande lorsque des valeurs de ϵ plus petites sont considérées. Ceci est dû, ici aussi, au fait que le pas de temps est choisi indépendamment de ϵ, ce qui génère une plus grande erreur lorsque ϵ est petit, augmentant la diffusion numérique perpendiculaire. Malgré cela, la méthode est capable d'atteindre l'état stationnaire avec $\epsilon = 2\cdot10^{-3}, 10^{-4}, 2\cdot10^{-4}, 10^{-4}$.

2.6 Conclusion

Dans ce chapitre, différents schémas numériques sont proposés, rappelés et comparés pour des problèmes de diffusion. Le schéma semi-implicite (SH) proposés dans [53], une extension (ARK) du schéma proposé dans [19] et le schéma complètement implicite sont comparés en utilisant un schéma volume finis d'ordre deux et quatre pour la discrétisation spatiale. Le schéma SH est très efficace lorsqu'il est couplé à un schéma volume finis d'ordre deux en espace, mais s'avère beaucoup moins performant lorsqu'un schéma d'ordre quatre est utilisé. Dans ce cas, le schéma de pénalisation ARK est alors meilleur. En

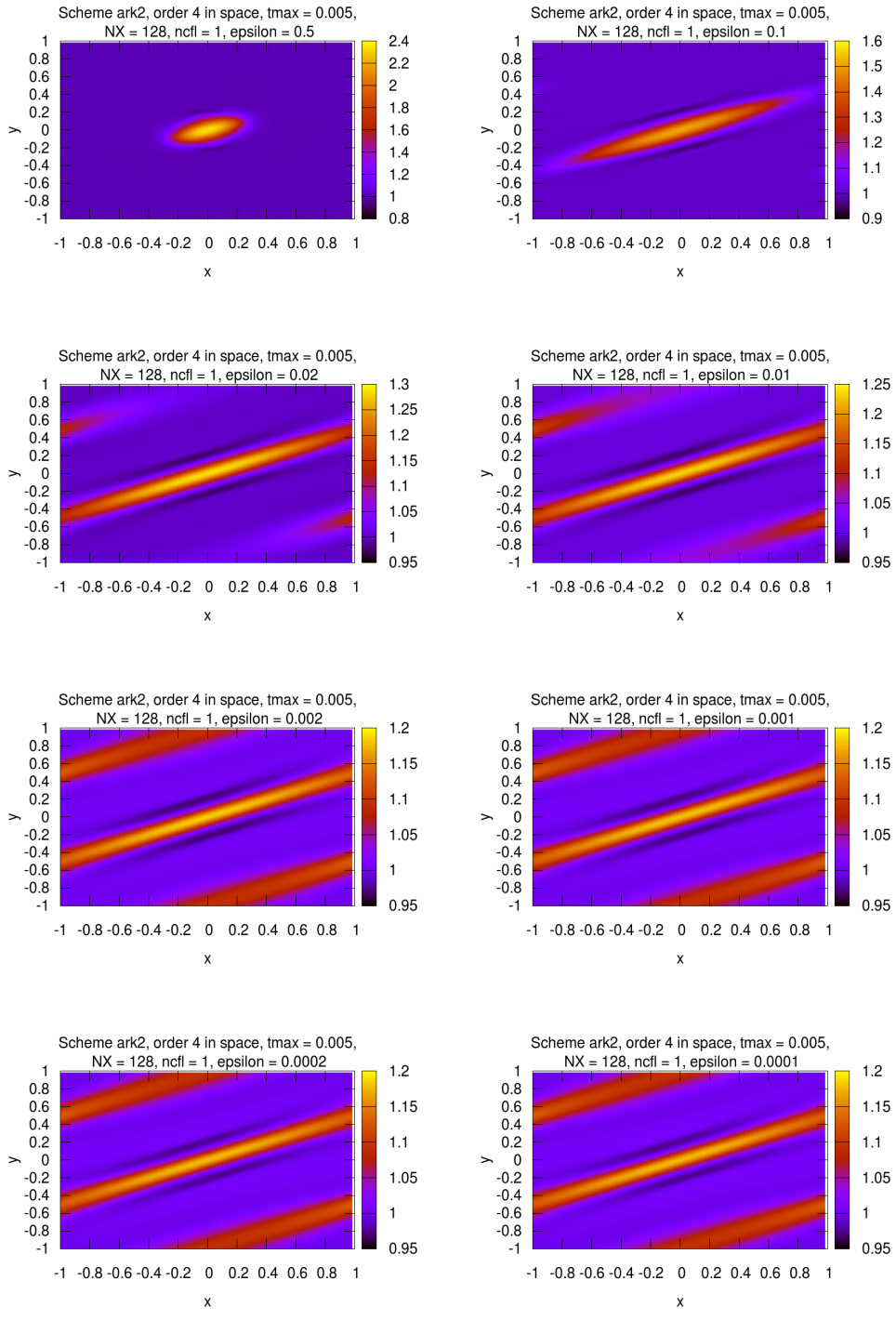


FIGURE 2.20 – Diffusion non linéaire sur une bande périodique à $t_{\max} = 0.005$, pour différentes valeurs de ϵ, avec $N_x = 128$ et ncfl $= 1$.

effet, dans les cas tests étudiés, la diffusion numérique perpendiculaire est plus faible lorsque des schémas d'ordres élevés sont utilisés. De plus, les schémas d'ordres élevés en temps permettent d'atteindre une meilleure précision ainsi que l'ordre théorique même pour des nombres de Courant (ncfl) grands. C'est une propriété très intéressante, en particulier lorsque l'on souhaite effectuer des simulations en temps longs, puisque le pas de temps Δt peut être choisi plus grand. En pratique, on est capable d'atteindre des valeurs pour ncfl entre 10 et 100, tout en gardant une erreur satisfaisante.

Plusieurs extensions à travail sont possibles. D'abord, au niveau du choix du paramètre de pénalisation λ : afin de suivre au mieux l'anisotropie de l'opérateur étudié, il semble intéressant de choisir λ comme une fonction dépendant de la variable spatiale. D'autres choix plus élaborés peuvent être faits pour approcher au mieux la matrice de diffusion ; cependant, un bon compromis précision/efficacité doit être trouvé puisque ce nouvel opérateur $\nabla \cdot (\lambda \nabla T^{n+1})$ devra être inversé. Un second point important serait d'étendre les schémas au cas non périodique afin d'aborder des cas test plus réalistes d'un point de vue de la physique. Afin de ne pas briser l'ordre en espace, des techniques doivent être utilisées comme celles proposées dans [57]. Enfin, l'aspect calcul haute performance serait intéressant à approfondir en étudiant plus finement les coûts de calcul de la méthode de pénalisation ARK, afin de mesurer le bénéfice potentiel sur le temps d'exécution.

Des extensions moins immédiates peuvent aussi être envisagées. Plusieurs schémas ont récemment été publiés [43, 47, 9, 56] possédant la propriété Asymptotic Preserving. Ces schémas garantissent une précision uniforme indépendamment du rapport d'anisotropie $\chi_\perp/\chi_\parallel$. Cependant, ces schémas ne sont pas d'ordre élevé en espace ou en temps et il parait intéressant de combiner les techniques développées dans ce chapitre aux reformulations du problème initial proposées dans les travaux [43, 47, 9, 56]. Enfin, l'analyse numérique des schémas de type ARK serait intéressante à mener ; en effet, les propriétés de L-stabilité ou A-stabilité sont des notions importantes lorsque le rapport d'anisotropie tend vers zéro. Une étude comparative approfondie permettrait de confronter ces propriétés sur des cas test pertinents.

Chapitre 3

Optimisation et parallélisation OpenMP pour Emedge3D

Ce chapitre présente des stratégies pour la réduction du temps d'exécution du code Emedge3D, présenté en Section 3.1. Ces stratégies interviennent autant au niveau optimisation séquentielle qu'au niveau de la parallélisation OpenMP. Cette étude a été principalement menée sur deux architectures à mémoire partagée. La première est une machine 64 cœurs (8×8), composée de processeurs Intel E7-8837 @ 2.67 GHz. Dans la suite, nous appellerons cette machine SMP. Aussi, nous utiliserons un des nœuds d'un cluster composé de 96 nœuds de calcul, comportant chacun deux processeurs Intel X5675 @ 3.06 GHz (6 cœurs), soit 1152 coeurs au total. Ces machines sont partie intégrante du mésocentre Aix Marseille Université.

Comme pour beaucoup d'applications du même type, les performances de la parallélisation du code numérique cible sont limitées, entre autre, par la bande passante mémoire. Les techniques que nous présentons dans ce document montrent certains moyens que nous avons mis en œuvre pour surmonter cette limitation. Afin d'obtenir des accélérations efficaces, différentes stratégies ont été envisagées au niveau des calculs, mais aussi au niveau de l'accès aux données. Les optimisations principales sont la minimisation du nombre d'accès à la mémoire, la simplification et le ré-ordonnancement des calculs et le tiling pour maximiser l'utilisation des caches mémoire. Sur un nœud de 12 cœurs, l'accumulation de ces optimisations et la parallélisation permettent d'obtenir un code 21.6 fois plus rapide par rapport à la version initiale sur un seul cœur.

Dans la suite, nous ne considérons que la partie la plus coûteuse du code en terme de temps de calcul. On appelle M_x, M_y, M_z le nombre de points dans les directions radiale, poloïdale et toroïdale (respectivement) en représentation semi-spectrale. De la même manière, on pose N_x, N_y, N_z (avec $N_x = M_x$) pour désigner le nombre de points utilisés dans la représentation des variables de

l'espace physique (*i.e.* dans $\mathbb{R}$).

En terme de stockage mémoire, cela implique la manipulation de tableaux de nombre complexe 3D dans la représentation semi-spectrale, du type tab$[Re|Im][M_z][M_y][M_x]$, où $Re|Im$ correspond au choix entre partie réelle ou imaginaire.

Sommaire

3.1 Présentation du code Emedge3D

Emedge3D [23] est une application numérique pour la simulation des plasmas de bord de tokamak. Ce code est issu d'un modèle fluide dont les équations peuvent s'écrire de manière abstraite :

$$\partial_t U = L(U) + NL(U),$$

où $U = U(t, X)$ est le vecteur des variables physiques (champs scalaires 3D), $X = (x, y, z)$ le vecteur des variables d'espace, L un opérateur différentiel spatial linéaire et NL un opérateur différentiel spatial non linéaire. Cette équation est résolue en utilisant des méthodes numériques classiques de type différences finies ou spectrales.

Néanmoins, les simulations cibles en terme d'échelle sont pour l'instant irréalisables. En effet, ces dernières supposent des temps d'exécution beaucoup trop longs. Ceci provient du caractère fortement multi-échelle des phénomènes que le code a pour but de simuler. En effet, les *relaxations de barrière de transport* arrivent à des échelles de temps élevées, alors que le pas de temps

imposé pour des raisons de stabilité numérique est très faible, requérant un volume approximatif de 10^7 itérations. Ajouté à cela, les phénomènes tels que les *îlots magnétiques* apparaissent à des échelles spatiales fines, imposant une taille de grille conséquente, et donc un grand volume de données et de calcul à traiter lors de la résolution numérique des opérateurs spatiaux (tout ceci pour chaque itération temporelle). Une stratégie de parallélisation est alors nécessaire pour réduire les temps de calcul, typiquement à quelques jours ou quelques heures.

Dans ce chapitre, nous présenterons tout d'abord le code Emedge3D ainsi que ses opérateurs. Ensuite, nous explorerons les stratégies mises en œuvre pour une parallélisation sur machine à mémoire partagée. Enfin, ce chapitre s'ouvrira sur le cas d'une parallélisation sur machine à mémoire distribuée, qui sera présentée au Chapitre 4.

3.1.1 Emedge3D : aspects physique et numérique

Emedge3D est un code de simulation numérique de type fluide écrit en langage C. Il a pour but l'observation de l'évolution en temps de trois quantités physiques (champs scalaires) sur un domaine spatial 3D donné.

Les trois inconnues sont la pression p, le potentiel électrostatique ϕ et le potentiel électromagnétique ψ d'un plasma de tokamak. Ces trois inconnues (p, ϕ, ψ) dépendent de la variable d'espace $X = (x, y, z) \in \mathbb{R}^3$ qui décrivent une géométrie torique simplifiée, ainsi que du temps t. Les différentes directions sont appelées radiale (x), poloïdale (y) et toroïdale (z). Des conditions de bord de type Neumann ou Dirichlet sont appliquées dans la direction radiale. Pour les autres directions, les conditions de bords sont périodiques.

Ce code vise à simuler des phénomènes physiques tels que les relaxations de barrière de transport ([6]), et comment contrôler ces relaxations par l'introduction de résonances magnétiques ([44]), ainsi que de caractériser le comportement des îlots magnétiques ([46]). Le modèle physique résolu par ce code, et donc le code lui même, peut être décomposé en deux parties : une première partie linéaire (L), dans laquelle les opérateurs spatiaux sont linéaires au sens mathématique du terme, et une seconde partie non linéaire (NL).

Pour réduire le coût en terme de calcul, deux discrétisations spatiales sont utilisées (voir aussi la Sous Section 3.1.2) : une représentation semi-spectrale des champs scalaires pour le calcul de la partie linéaire, dans laquelle les dimensions poloïdale et toroïdale sont exprimées dans la base de Fourier, et la dimension radiale dans la base de la variable réelle physique, et une seconde représentation dans l'espace réel pour les trois directions, pour résoudre la partie non linéaire.

Pour intégrer les équations en temps, un schéma Runge-Kutta (RK) de l'or-

dre désiré est appliqué. En espace, les opérateurs sont résolus par des méthodes classiques de type différence finie, donnant lieu à une complexité en temps de calcul linéaire en fonction de la taille du domaine dans la plus grande partie du code.

Enfin, le code permet de dégrader le modèle physique résolu afin d'étudier différents types de cas plus élémentaires.

3.1.2 Description des opérateurs

Partie linéaire

Pour chaque itération temporelle, plusieurs opérateurs spatiaux doivent être résolus numériquement :

– trois opérateurs de diffusion perpendiculaire (dans le plan (x, y)) :

$$\nabla_\perp \cdot (\nu(x)\nabla_\perp W),\ \nabla_\perp \cdot (\chi(x)\nabla_\perp p),\ \nabla^2_\perp \psi, \tag{3.1.1}$$

avec $\nabla_\perp = (\partial_x, \partial_y)$, $W = \nabla^2_\perp \phi$, et $J = \nabla^2_\perp \psi$.

– deux opérateurs de courbure :

$$Gp,\ G(\delta_c \phi - \Gamma p), \tag{3.1.2}$$

avec $G = \cos(y)\partial_y + \sin(y)\partial_x$.

Ces opérateurs sont appliqués éventuellement plusieurs fois par pas de temps, en fonction du schéma RK utilisé. La complexité en temps de calcul est linéaire pour ces deux opérateurs, en $\Theta(M_x \times M_y \times M_z)$, où M_d désigne le nombre de points dans la direction d.

Partie non linéaire

De même que pour la partie linéaire, pour chaque pas de temps et autant de fois que requis par le schéma RK, le code doit résoudre numériquement des opérateurs spatiaux appelés gradients parallèle ou encore crochets de Poisson, que l'on exprime comme suit :

$$\nabla_\parallel \cdot (\chi_\parallel(x, y)\nabla_\parallel p), \nabla_\parallel J, \nabla_\parallel \phi, \tag{3.1.3}$$

avec $\nabla_\parallel = \partial_z - \{\psi, .\}$ et $\{f, g\} = \partial_x f \partial_y g - \partial_y f \partial_x g$.

Pour résoudre un crochet de Poisson sur deux champs scalaires 3D en représentation semi-spectrale $\hat{f}(x, y, z)$ et $\hat{g}(x, y, z)$, pour une position radiale et des modes m et n donnés, on applique la formule suivante :

$$\{\hat{f}, \hat{g}\}_{m,n} = \sum_{i=0}^{M_z} \sum_{j=0}^{M_y} \left((m - j)\partial_x \hat{f}_{j,i} \partial_y \hat{g}_{m-j,n-i} - j\partial_y \hat{f}_{j,i} \partial_x \hat{g}_{m-j,n-i} \right). \tag{3.1.4}$$

Cette dernière formule présente l'avantage d'être numériquement exacte dans la représentation de Fourier. Néanmoins, la complexité en terme de calculs pour un tableau en représentation semi-spectrale est en $\Theta(M_y^2 \times M_z^2)$ (il s'agit en fait d'une convolution). La technique présentée dans [24] pour réduire ce coût en calcul est d'effectuer un changement de base en appliquant des transformées de Fourier discrètes, afin de résoudre l'opérateur dans la base des variables spatiales physiques (*i.e.* dans $\mathbb{R}^3$). Il en résulte une complexité en calcul en $\Theta(M_y \times M_z \times \log(M_y \times M_z))$ induite par les transformées de Fourier 2D. La méthode numérique utilisée pour résoudre les crochets de Poisson est une méthode numérique d'ordre 2 introduite par A. Arakawa dans [1], méthode reconnue pour ses propriétés de conservation d'énergie et du carré de la vorticité, très répandue et utilisée dans la communauté de physique des plasmas.

3.2 Optimisation et parallélisation de la partie linéaire

Cette partie présente l'optimisation et la parallélisation multicoeur de la résolution numérique des opérateurs de diffusion perpendiculaire et de courbure. Cette résolution s'effectue dans la représentation semi-spectrale des champs scalaires. Les optimisations consistent principalement en l'amélioration de l'organisation des données en mémoire ainsi que l'accès à ces données. Quant à la parallélisation, les calculs traitent des données locales : chaque thread traite une zone de mémoire qui lui est propre.

3.2.1 Optimisation séquentielle

Le code Emedge3D a été écrit avec une vision orientée objet des opérateurs et des champs scalaires. Au niveau du traffic en mémoire, cela implique des accès aux tableaux 4D (3 dimensions de l'espace, plus une dimension pour les parties réelle et imaginaire) qui stockent les valeurs des champs à chaque fois qu'un opérateur doit être résolu numériquement. L'Algorithme 1 illustre la manière de résoudre les opérateurs présentés dans les Equations (3.1.1) et (3.1.2). Dans cet algorithme, pour chaque appel de fonction, une traversée complète en lecture et écriture est nécessaire pour le tableau en sortie ($*\_out$), ainsi qu'une traversée en lecture pour l'entrée ($*\_in$). Ces accès et calculs s'effectuent à l'intérieur de trois boucles imbriquées itérant sur les points de la grille 3D, parties réelle et imaginaire étant traitées lors de la même itération. Pour plus d'information sur le type de code à traiter, nous renvoyons le lecteur à l'annexe B page 155.

Algorithme 1 Algorithme initial pour la partie linéaire.

```
∇²⊥(p_out, p_in, ...);
∇²⊥(W_out, W_in, ...);
∇²⊥(psi_out, psi_in, ...);
G(p_out, phi_in, ...);
G(p_out, p_in, ...);
G(W_out, p_in, ...);
```

La première optimisation concerne l'amélioration de la localité des données en espace. L'ordre des dimensions en mémoire des données en discrétisation semi-spectrale est transposé, passant de $(Re|Im, M_z, M_y, M_x)$ à $(M_z, M_y, M_x, Re|Im)$ (la dernière dimension étant contiguë en mémoire dans cette écriture). En effet, parties réelles et imaginaires sont traitées l'une après l'autre pour une position donnée sur la grille. Rapprocher les parties réelle et imaginaire via cette transposition permet d'améliorer la localité spatiale et temporelle lorsque l'on applique les opérateurs. Cette transposition permet d'obtenir un code en moyenne 1.4 fois plus rapide pour l'Algorithme 1 sur les machines présentées en début de section. Pour la seconde optimisation séquentielle, le chargement des données est "pipeliné" d'une itération spatiale à la suivante, permettant de garder au maximum les données dans des variables locales, au plus proche des unités de calcul. On parle aussi de "software pipelining". Il s'agit ici d'amélioration de la localité des données en temps : on garde les données le plus longtemps et le plus haut (au sens proche des unités de calcul là aussi) possible dans la hiérarchie mémoire en vue d'une réutilisation proche. Concrètement, les données sont chargées dans des variables locales et sont mises à jour d'une itération spatiale à l'autre, en gardant les accès communs entre les différentes itérations spatiales. Ce software pipelining permet de baisser le nombre d'accès mémoire de 35 à 15 par itération spatiale pour l'opérateur de diffusion seul, produisant un facteur d'accélération de 2.3.

3.2.2 Parallélisation de la partie linéaire

La parallélisation pour cette partie du code consiste à distribuer les indices de boucle de la dimension la plus externe du point de vue de la contiguïté mémoire M_z sur plusieurs threads via le paradigme OpenMP. Une répartition statique des itérations sur les différentes unités de calcul est utilisée dans le souci de garantir une charge de travail égale (à plus ou moins une itération près). La Figure 3.1 montre les speedups obtenus (en bleu) en fonction du nombre de threads utilisés sur les deux architectures cibles. On y observe une

limite dans la progression du speedup lorsque l'on utilise plus de 4 (respectivement 16) sur l'architecture 12 cœurs (respectivement 64 cœurs). Cette limite s'explique par les besoins en bande passante mémoire (vitesse d'accès à la mémoire vive) de l'application qui augmente lorsque l'on ajoute des threads. Afin d'illustrer cela, les courbes noires sur les graphes de la Figure 3.1 montrent le scaling de la bande passante mémoire en fonction du nombre de threads utilisés. Ces valeurs de bande passante mémoire ont été obtenues avec l'outil stream, permettant entre autre la mesure de bande passante pour des architecture multicoeurs ([40]). Les courbes bleues représentent le speedup de l'ensemble de la partie linéaire. Les courbes bleues et noires sont proches qualitativement et quantitativement pour les deux architectures. En outre, cette limitation de la bande passante mémoire a également été observée par des mesures de compteurs matériels (en utilisant la bibliothèque PAPI, voir [45]).

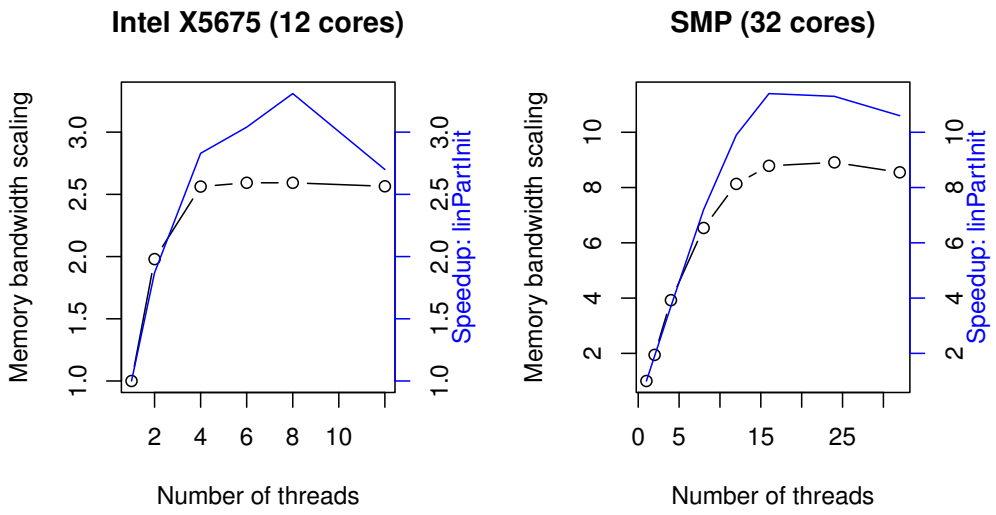


FIGURE 3.1 – Scaling de la bande passante mémoire et speedups pour la partie linéaire en fonction du nombre de threads.

3.2.3 Optimisation après parallélisation

Les optimisations pour la version parallèle ont pour but de surmonter la limitation imposée par la bande passante mémoire. La stratégie est ici de garder les données le plus haut possible dans la hiérarchie mémoire, notamment dans les différents niveaux de cache.

La première optimisation parallèle consiste à externaliser la boucle la plus externe (dimension M_z) afin de résoudre tous les opérateurs spatiaux pour

une même tranche (x, y) les uns à la suite des autres. Ce remaniement permet d'améliorer la localité temporelle des données de la tranche (x,y). En outre, cela est possible car il n'y a aucune dépendance sur l'ordre d'application des opérateurs, ni dans la direction z. Néanmoins, prenant en considération les tailles de grille envisagées pour Emedge3D, le volume de donnée accédé par tranche (x, y) peut éventuellement être trop grand pour rester en cache entre chaque opérateur.

La seconde optimisation appliquée est le réordonnancement des appels de fonctions. En effet, il est possible de juxtaposer de manière optimale les appels de fonctions faisant référence aux mêmes zones mémoires pour encore une fois augmenter la localité temporelle des accès mémoire. Par exemple, sur les Equation (3.1.1) et (3.1.2), on remarque que le champ de pression intervient dans une des diffusion perpendiculaire, mais aussi dans deux courbures. Il est alors naturel de faire en sorte que ces opérations soient effectués les unes à la suite des autres. On obtient alors l'Algorithme 2. Les performances sont alors meilleurs dans certains cas, mais cette optimisation reste insuffisant pour bénéficier d'effets cache vraiment efficients dans toutes les configurations.

Algorithme 2 Algorithme pour la partie linéaire : tranches 2D et réordonnancement des appels.

```
#pragma omp parallel for private(idz)
for ( idz=0;idz<M_z;idz++ ){
  G_Z(p_out, phi_in, ..., idz);
  ∇²⊥_Z(p_out, p_in, ..., idz);
  G_Z(p_out, p_in, ..., idz);
  G_Z(W_out, p_in, ..., idz);
  ∇²⊥_Z(W_out, W_in, ..., idz);
  ∇²⊥_Z(psi_out, psi_in, ..., idz);
}
```

La solution trouvée pour passer au delà de la limite de la bande passante mémoire est d'appliquer un tiling des boucles sur les dimensions radiale et poloïdale pour appliquer les opérateurs en série sur une même tuile de taille (Tx, Ty). Ceci permet de contrôler la taille des données accédées de manière plus fine, pour atteindre un volume par tuile suffisamment petit, et ainsi garder un maximum de données en cache L2 d'un appel de fonction à l'autre. L'Algorithme 3 montre le résultat obtenu après cette optimisation.

Algorithme 3 Linear parallel algorithm : tiled version.

```
#pragma omp parallel for private(idz,iblockX,iblockY)
  for ( idz=0;idz<M_z;idz++ ){
    for(iblockY=0; iblockY<nblockY; iblockY++)
    {
      for(iblockX=0; iblockX<nblockX; iblockX++)
      {
        G_Z_tiled(p_out, phi_in , ..., idz,
           iblockX, blocksizeX, iblockY, blocksizeY);
        ∇²⊥_Z_tiled(p_out, p_in , ..., idz,
           iblockX, blocksizeX, iblockY, blocksizeY);
        G_Z_tiled(p_out, p_in , ..., idz,
            iblockX, blocksizeX, iblockY, blocksizeY);
        G_Z_tiled(W_out, p_in , ..., idz,
            iblockX, blocksizeX, iblockY, blocksizeY);
        ∇²⊥_Z_tiled(W_out, W_in , ..., idz,
            iblockX, blocksizeX, iblockY, blocksizeY);
        ∇²⊥_Z_tiled(psi_out, psi_in, ..., idz,
            iblockX, blocksizeX, iblockY, blocksizeY);
      }
    }
  }
```

3.2.4 Evaluation des performances pour la partie linéaire

Pour trouver la meilleure taille de tuile, et donc le volume optimal de données accédées par tuile, un profiling du code a été effectué sur les deux architectures multicoeurs. Plusieurs tailles de grille ont été testées, mais les résultats étant très similaires, nous ne présentons ici les résultats que pour le maillage $(M_x, M_y, M_z) = (256, 256, 192)$. Les Tableaux 3.1 et 3.2 donnent les meilleurs temps d'exécution (et donc les meilleures tailles de tuile) pour les différents nombres de threads utilisés sur les deux architectures considérées. Ces résultats montrent clairement que la taille de tuile optimale dépend directement du nombre de threads utilisés ainsi que de l'architecture. La Figure 3.2 montre que les codes "tilés" avec la taille de tuile appropriée sont maintenant capables d'outrepasser la limitation de la bande passante mémoire, permettant d'atteindre des speedups bien plus intéressants et proches du nombre de cœurs utilisés.

TABLE 3.1 – Meilleurs temps d'exécution (parmi toutes les tailles de tuiles testées) par nombre de threads sur Intel X5675.

Tile size (Tx, Ty)	Threads	Time (μs)	Speedup
(256,256)	1.00	1401605.00	1.00
(256,128)	1.00	1400773.00	1.00
(256,64)	2.00	717327.00	1.95
(128,32)	4.00	373240.00	3.76
(32,4)	6.00	259935.00	5.39
(32,4)	8.00	199425.00	7.03
(16,2)	12.00	137913.00	10.16

Pour expliquer ces résultats, un profiling des performances caches a été effectué à l'aide de la bibliothèque PAPI [45]. La Figure 3.3 montre les performances du cache L3 pour les architectures Intel X5675 et SMP. Les courbes montrent clairement que la performance des accès en cache L3 en fonction du nombre de threads utilisé est améliorée en choisissant les bonnes tailles de tuile, réduisant le caractère memory bound du programme. Par exemple, sur Intel X5675, avec la taille de tuile $(16, 2)$, le nombre de cache miss diminue linéairement indiquant que les tuiles sont assez petites pour garder les données en cache d'un appel de fonction à l'autre. Par contre, avec une taille de $(128, 32)$, le nombre de cache miss augmente à partir de l'utilisation de 8 threads, signifiant que la taille de tuile est trop grande pour bénéficier des effets cache. De

TABLE 3.2 – Meilleurs temps d'exécution (parmi toutes les tailles de tuiles testées) par nombre de threads sur SMP.

Tile size (Tx, Ty)	Threads	Time (μs)	Speedup
(256,256)	1.00	1776686.00	1.00
(256,32)	1.00	1733988.00	1.02
(8,8)	2.00	952863.00	1.86
(4,16)	4.00	515770.00	3.44
(4,2)	6.00	357161.00	4.97
(2,8)	8.00	276104.00	6.43
(4,2)	12.00	182989.00	9.71
(4,2)	16.00	138960.00	12.79
(4,2)	24.00	95690.00	18.57
(4,2)	32.00	71879.00	24.72
(2,2)	64.00	44876.00	39.59

plus, les performances caches sur X5675 identifient presque la meilleure taille de tuile en fonction du nombre de threads utilisés par l'application (voir aussi le Tableau 3.1).

En bilan à cette optimisation et parallélisation de la partie linéaire, nous avons atteint des speedups de 10.16 sur un nœud 12 coeurs, ainsi qu'un speedup de 39.59 sur l'architecture 64 cœurs.

3.3 Optimisation et parallélisation de la partie non linéaire

Dans cette section, il est principalement question de la résolution numérique des crochets de Poisson, opérateurs au cœur de la partie non linéaire du code Emedge3D. Il s'agit plus précisément d'optimiser et de paralléliser via OpenMP la méthode d'Arakawa employée pour le calcul de l'opérateur, ainsi que les transformées de Fourier discrètes utilisées pour le changement de discrétisation variables physiques réelles vers semi-spectral (aller et retour). Les optimisations consistent ici en l'amélioration des performances en terme d'accès mémoires et de réduction des coûts en calcul.

Pour plus de compréhension, le format de stockage de données en mémoire est explicité ci-après :

– comp[$M_x * M_z * M_y * 2$] pour la représentation en semi-spectral, signifiant que Re(comp(idx, idy, idz)) est stocké en position $idx * (M_z * M_y * 2) +$

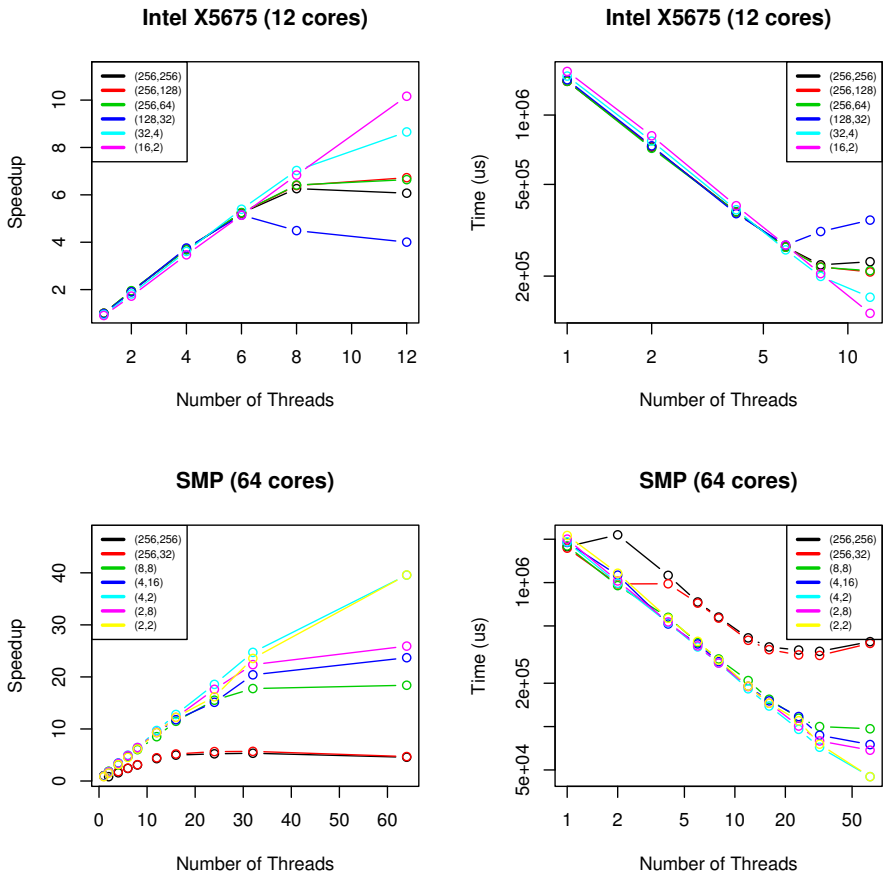


FIGURE 3.2 – Performance correspondante aux meilleures tailles de tuile sur les deux architectures (cf. Tableaux 3.1 et 3.2).

$idz * (M_y * 2) + idy * 2$ et $\mathrm{Im}(\mathrm{comp}(idx, idy, idz))$ à l'indice $idx * (M_z * M_y * 2) + idz * (M_y * 2) + idy * 2 + 1$,

– real[$N_x * N_z * N_y$] pour la représentation en variable réelle physique, avec $\mathrm{real}(idx, idy, idz)$ en position $idx * (N_z * N_y) + idz * (N_y) + idy$

3.3.1 Description de l'algorithme de la partie non linéaire

Comme expliqué en Sous Section 3.1.2, les crochets de Poisson sont résolus dans l'espace réel. Nous avons pour hypothèse ici que les données en entrée sont dans la représentation semi-spectrale, ainsi que les sorties. Cela signifie que cette partie doit inclure les TFD (transformées de Fourier discrètes) pour

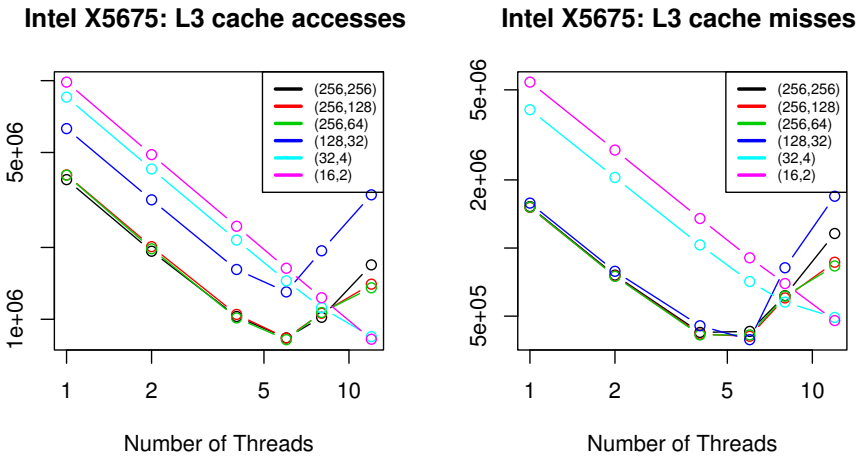


FIGURE 3.3 – Performance en cache L3 correspondante aux meilleures tailles de tuile des Tableaux 3.1 et 3.2 sur les deux architectures.

le passage d'une discrétisation à l'autre (voir l'Algorithme 4 page 102). Dans cet algorithme, $f1comp$ et $f2comp$ sont les données (tableaux, ou champs de complexe) en entrée pour la méthode d'Arakawa. Ces tableaux sont tout d'abord transportés dans l'espace des variables physiques réelles par TFD, via la bibliothèque fftw3. Les résultats de cette transformation sont stockés dans les tableaux $f1\_in$ et $f2\_in$. Ensuite, la méthode d'Arakawa est appliquée aux tableaux $f1\_in$ et $f2\_in$, et le résultat est écrit dans f_out. Enfin, f_out est transporté dans l'espace semi-spectral par TFD dans $f3comp$. Notons que pour cette partie la variable la plus externe au niveau des boucles est la variable x, car y et z sont couplées dans la fft dans le plan (y, z).

3.3.2 Optimisations séquentielles pour la partie non linéaire

La méthode d'Arakawa

Cette méthode est utilisée pour résoudre numériquement les crochets de Poisson. Dans de nombreux codes, elle est très appréciée pour ses propriétés de conservation de la norme L^2 de l'énergie du système. Dans la suite, des simplifications au niveaux des calculs et des accès mémoires sont présentées et évaluées.

Pour i et j donnés, la formule calculant la méthode d'Arakawa est donnée

Algorithme 4 Algorithme principal pour la partie non linéaire

```
/* FFT 2D: Complex ----> Real */
int lenComp = M_y*M_z;
int lenReal = N_y*N_z;
for(idx=0;idx<M_x;idx++)
{
  int sliceComp = idx*lenComp;
  int sliceReal = idx*lenReal;
  fftw_execute_dft_c2r(p2D_C2R,&(f1comp[sliceComp]),
                               &(f1_in[sliceReal]));
  fftw_execute_dft_c2r(p2D_C2R,&(f2comp[sliceComp]),
                               &(f2_in[sliceReal]));
}
ArakawaMultiplication(f1_in,f2_in,f_out,...);
/* FFT 2D: Real ----> Complex */
for(idx=0;idx<N_x;idx++)
{
  int sliceComp = idx*lenComp;
  int sliceReal = idx*lenReal;
  fftw_execute_dft_r2c(p2D_R2C,&(f_out[sliceReal]),
                               &(f3comp[sliceComp]));
}
```

par :

$$\begin{aligned} J_{i,j}(\xi,\psi) &= -\frac{1}{\Delta x \Delta y}\left[\left(\psi_{i,j-1}+\psi_{i+1,j-1}-\psi_{i,j+1}-\psi_{i+1,j+1}\right)\left(\xi_{i+1,j}+\xi_{i,j}\right)\right. \\ &\quad -\left(\psi_{i-1,j-1}+\psi_{i,j-1}-\psi_{i-1,j+1}-\psi_{i,j+1}\right)\left(\xi_{i,j}+\xi_{i-1,j}\right) \\ &\quad +\left(\psi_{i+1,j}+\psi_{i+1,j+1}-\psi_{i-1,j}-\psi_{i-1,j+1}\right)\left(\xi_{i,j+1}+\xi_{i,j}\right) \\ &\quad -\left(\psi_{i+1,j-1}+\psi_{i+1,j}-\psi_{i-1,j-1}-\psi_{i-1,j}\right)\left(\xi_{i,j}+\xi_{i,j-1}\right) \\ &\quad +\left(\psi_{i+1,j}-\psi_{i,j+1}\right)\left(\xi_{i+1,j+1}+\xi_{i,j}\right) \\ &\quad -\left(\psi_{i,j-1}-\psi_{i-1,j}\right)\left(\xi_{i,j}+\xi_{i-1,j-1}\right) \\ &\quad +\left(\psi_{i,j+1}-\psi_{i-1,j}\right)\left(\xi_{i-1,j+1}+\xi_{i,j}\right) \\ &\quad \left.-\left(\psi_{i+1,j}-\psi_{i,j-1}\right)\left(\xi_{i,j}+\xi_{i+1,j-1}\right)\right]. \end{aligned}$$

C'est un stencil 2D dans le plan (x, y) de largeur 2 dans chaque direction. Après développement, on remarque que les produits de type $\psi_{*,*}\xi_{i,j}$ se simplifient, ce qui donne lieu à une économie de 8 additions :

$$\begin{aligned} J_{i,j}(\xi,\psi) &= -\frac{1}{\Delta x \Delta y}\left[\left(\psi_{i,j-1}+\psi_{i+1,j-1}-\psi_{i,j+1}-\psi_{i+1,j+1}\right)\xi_{i+1,j}\right. \\ &\quad -\left(\psi_{i-1,j-1}+\psi_{i,j-1}-\psi_{i-1,j+1}-\psi_{i,j+1}\right)\xi_{i-1,j} \\ &\quad +\left(\psi_{i+1,j}+\psi_{i+1,j+1}-\psi_{i-1,j}-\psi_{i-1,j+1}\right)\xi_{i,j+1} \\ &\quad -\left(\psi_{i+1,j-1}+\psi_{i+1,j}-\psi_{i-1,j-1}-\psi_{i-1,j}\right)\xi_{i,j-1} \\ &\quad +\left(\psi_{i+1,j}-\psi_{i,j+1}\right)\xi_{i+1,j+1} \\ &\quad -\left(\psi_{i,j-1}-\psi_{i-1,j}\right)\xi_{i-1,j-1} \\ &\quad +\left(\psi_{i,j+1}-\psi_{i-1,j}\right)\xi_{i-1,j+1} \\ &\quad \left.-\left(\psi_{i+1,j}-\psi_{i,j-1}\right)\xi_{i+1,j-1}\right]. \end{aligned}$$

Nous appelons cette première optimisation AddOpti.

Ensuite, aux vues du nombre de transactions mémoire, les chargements de données pour ψ puis pour ψ et ξ sont "pipelinés" dans des variables locales de la même manière que dans la partie linéaire (voir la Section 3.2), pour maximiser la réutilisation des chargements dans les registres des processeurs. Ces optimisations sont appelées SemiLoadReuse (ψ seulement) et LoadReuse (ψ et ξ). Les Algorithmes 5 et 6 (original puis optimisé) montrent la mise en place de ces optimisations. Ces optimisations incluent AddOpti.

De plus, il est aussi possible d'économiser des opérations sur les parenthèses qui impliquent ψ. En effet, si l'on pose :

$$\begin{aligned} \psi^A_{i,j} &= \left(\psi_{i,j-1}+\psi_{i+1,j-1}-\psi_{i,j+1}-\psi_{i+1,j+1}\right), \\ \psi^C_{i,j} &= \left(\psi_{i+1,j}+\psi_{i+1,j+1}-\psi_{i-1,j}-\psi_{i-1,j+1}\right), \\ \psi^E_{i,j} &= \left(\psi_{i+1,j}-\psi_{i,j+1}\right), \\ \psi^G_{i,j} &= \left(\psi_{i,j+1}-\psi_{i-1,j}\right), \end{aligned}$$

Algorithme 5 Méthode d'Arakawa : version initiale

```
void ArakawaMultiplication(psi,xi,result,
                  N_x,N_y,N_z,dx,dy)
{
...
  for(i=0;i<N_x;i++){
    for(j=0;j<N_z;j++){
      for(k=0;k<N_y;k++){
        result[resIndex] = (psi[ijm1] + psi[ip1jm1] -
                            psi[ijp1] - psi[ip1jp1])
                            * ...;
      }
    }
  }
}
```

la dernière expression donnant la formule pour la méthode d'Arakawa devient :

$$\begin{aligned} J_{i,j}(\xi,\psi) &= -\frac{1}{\Delta x \Delta y}\left[\psi^A_{i,j}\xi_{i+1,j} - \psi^A_{i-1,j}\xi_{i-1,j}\right. \\ &\quad +\psi^C_{i,j}\xi_{i,j+1} - \psi^C_{i,j-1}\xi_{i,j-1} \\ &\quad +\psi^E_{i,j}\xi_{i+1,j+1} - \psi^E_{i-1,j-1}\xi_{i-1,j-1} \\ &\quad \left.+\psi^G_{i,j}\xi_{i-1,j+1} - \psi^G_{i+1,j-1}\xi_{i+1,j-1}\right]. \end{aligned}$$

Cette dernière formulation montre qu'il est possible de réutiliser des calculs sur ψ d'une itération à l'autre. Par exemple, à l'itération $i' = i, j' = j+1$, $\psi^C_{i',j'-1}$ a déjà été calculé lors de l'itération i,j (exploitation de la localité temporelle de ces opérations dans les caches). Néanmoins, forts de l'expérience de parallélisation multicoeur de la partie linéaire et de son caractère memory bound, nous ne considérons ici que la réutilisation de ψ^C (PsiReuse) ou de ψ^C et ψ^A (PsiACReuse) pour des raisons de stockage supplémentaires impliqués par ces réutilisations de calcul. Ces optimisations incluent LoadReuse par défaut, SemiLoadReuse lorsque indiqué dans la version utilisée.

Le Tableau 3.3 montre l'impact de ces différents niveaux d'optimisation sur les temps d'exécution pour la méthode d'Arakawa pour le processeur Intel X5675. PsiACReuse y est identifié comme meilleure optimisation possible pour la version séquentielle. Les mêmes résultats sont observés sur l'architecture SMP. Cette étude est réalisé en utilisant un seul coeur de calcul, nous verrons par la suite qu'il est nécessaire d'affiner celle-ci dans une configuration multi-coeurs et OpenMP.

Algorithme 6 Méthode d'Arakawa : AddOpti et LoadReuse

```
void ArakawaMultiplication(psi,xi,result,
                  N_x,N_y,N_z,dx,dy)
{
...
  for(i=1;i<NX-1;i++){
    for(j=0;j<N_z;j++){
      /* load values in registers */
      psi_im1jkm1 = psi[im1jkm1]; /* i=i-1, j=j, k=0 */
      psi_im1jk = psi[im1jk]; /* i=i-1, j=j, k=1 */
      psi_im1jkp1 = psi[im1jkp1]; /* i=i-1, j=j, k=2 */
      ... /* and so on for i and i+1 */ ...
      for(k=1;k<N_y-1;k++){
        result[resIndex] = (psi_ijkm1 + psi_ip1jkm1 -
                            psi_ijkp1 - psi_ip1jkp1) * ...;
        /* swap variables to avoid extra loads */
        psi_im1jkm1 = psi_im1jk;
        psi_im1jk = psi_im1jkp1;
        psi_im1jkp1 = psi[im1jkp2];
        ... /* and so on for i and i+1 */ ...
      }
    }
  }
}
```

TABLE 3.3 – Méthode d'Arakawa : Temps par optimisation séquentielle sur Intel X5675. Taille du maillage : (512,512,128).

Version	Time (μ s)
Original	1144520.00
AddOpti	987053.00
PsiCSemiLoadReuse	727701.00
PsiACSemiLoadReuse	697066.00
LoadReuse	687248.00
SemiLoadReuse	683341.00
PsiCReuse	676677.00
PsiACReuse	654606.00

Les transformées de Fourier discrètes (TFDs)

Chaque fois qu'un crochet de Poisson doit être évalué, le code doit exécuter trois principales tâches (voir l'Algorithme 4) en plus de la méthode d'Arakawa :

- deux transformées de 2D Fourier inverse (semi-spectral vers réel) pour les entrées de la méthode d'Arakawa, f1comp and f2comp,
- et une pour transformée de Fourier 2D pour la sortie, f3comp.

Ces étapes supplémentaires consistent à calculer M_x transformées de Fourier 2D pour les tableaux en entrée de la partie linéaire (f1comp et f2comp), et pour le tableau en sortie de cette partie (f3comp). Ces TFD sont effectuées par la version séquentielle de la bibliothèque *fftw* [22] version 3.3.3, déjà optimisée. On effectue alors des FFT (Fast Fourier Transform) pour réaliser les TFD.

Néanmoins, augmenter ou baisser le nombre de points dans les directions poloïdale ou toroïdale tout en gardant un volume proche peut impliquer des variations sur les temps d'exécution, jusqu'à un facteur 4.

Remarque 3.3.1 *Les calculs de FFT sont plus efficaces lorsque l'on considère des tailles en puissance de 2, efficacité provenant de l'algorithme employé (semblable à une réduction binaire). Dans Emedge3D, les plans pour les FFTs (information sur les tailles, choix de la méthode, ...) sont définies une fois pour toute à l'initialisation.*

La mise en place de transformées de Fourier discrète passe par une création de plan dans la terminologie de la bibliothèque fftw. La création de plan nécessite de spécifier les tailles des tableaux à transformer dans la base des variables physiques réelles. Or, les données pour Emedge3D sont initialisées dans l'espace semi-spectral (tableaux de complexes de taille (M_x, M_y, M_z)). Les tailles

pour l'espace réel doivent alors être déduites pour créer les plans des FFT par la formule $(N_x, N_y, N_z) = (M_x, (M_y - 1) \times 2, M_z)$.

Par exemple, pour une taille 2D de $(M_y, M_z) = (\mathbf{512}, 128)$*, Le temps d'exécution des FFTs est 4 fois plus long que pour une taille* $(M_y, M_z) = (\mathbf{513}, 128)$*. Ceci s'explique par le fait que* $M_y = 513$ *implique* $N_y = 1024 = 2^{10}$ *en taille pour la représentation réelle, donnant une FFT 2D de taille totale* $(\mathbf{1024}, 128)$*, alors que* $M_y = 512$ *donne une taille de* $(\mathbf{1022}, 128)$.

3.3.3 Parallélisation pour la partie non linéaire

De la même manière que dans la Sous Section 3.2.2, la boucle la plus externe en espace (ici la dimension radiale x) est parallélisée avec OpenMP. Cette direction de parallélisation est imposée par les FFT 2D dans le plan (y, z) qui doivent être faite d'un bloc. La méthode d'Arakawa est elle aussi parallélisée selon l'axe radial afin d'éviter une transposition des données en mémoire supplémentaire.

Méthode d'Arakawa

Le Tableau 3.4 donne les meilleurs temps par nombre de threads, ainsi que la meilleure optimisation parmi celles présentées dans les paragraphes précédents sur processeur Intel X5675. Les speedups atteints sont intéressants mais pas les plus hauts possibles en raison de la faible intensité de calcul (ratio nombre de calcul sur nombre d'accès mémoire) de la méthode, ils sont à nouveau limités par la bande passante mémoire. Ce tableau nous montre aussi que la meilleure version parallèle *i.e.* SemiLoadReuse n'est pas la même que pour la version séquentielle (PsiACReuse). En effet, PsiACReuse demande plus d'accès mémoire (ajout d'un tableau 1D pour stocker les calculs) et accentue la limitation par la bande passante mémoire. De même, SemiLoadReuse est meilleur que LoadReuse car la première version demande moins de mouvement dans les registres, fait observable dans le code assembleur de ces deux versions. Sur la SMP, la meilleure optimisation parallèle est là aussi SemiLoadReuse pour les mêmes raison, donnant un speedup de 19.99 par rapport à la version séquentielle optimisée sur les 64 coeurs. Ce résultat reste néanmoins satisfaisant, même si de meilleures performances seraient atteignable si l'on pouvait s'affranchir de la limite due à la bande passante mémoire. En outre, il vient confirmer celui obtenu sur le processeur Intel X5675.

Les transformées de Fourier discrètes

Pour le calcul des FFTs en parallèle, la boucle spatiale la plus externe est parallélisée avec OpenMP, *i.e.* celle sur la dimension radiale. Le Tableau

TABLE 3.4 – Méthode d'Arakawa : meilleur temps par nombre de threads sur Intel X5675. Taille de maillage : $(512, 512, 128)$.

Version	Threads	Time (μ s)	Speedup
PsiACReuse	1.00	654606.00	1.00
PsiACReuse	2.00	338784.00	1.93
SemiLoadReuse	4.00	193072.00	3.39
SemiLoadReuse	6.00	151341.00	4.33
SemiLoadReuse	8.00	137241.00	4.77
SemiLoadReuse	12.00	126286.00	5.18

3.5 compare les temps d'exécution en parallèle pour différentes tailles de domaine sur l'architecture Intel X5675, pour un aller retour FFT sur un tableau 3D. Pour expliquer ces résultats, nous devons considérer le volume total impliqué pour une opération FFT. Pour un maillage de taille (M_x, M_y, M_z), on a $M_x \times M_y \times M_z \times 2$ nombres flottants par tableau (le $\times 2$ est du au format complexe). Les valeurs étant stockées en double précision, cela donne un volume de données de $\frac{M_x \times M_y \times M_z \times 2 \times 8}{1024^2}$ MB à parcourir pour une opération FFT. Enfin, ces transformations ne sont pas faites "inplace" (le tableau d'entrée est différent du tableau de sortie) car cela nécessiterait des copies mémoires supplémentaires, et allongerait les temps d'exécutions. Il faut donc multiplier la taille de données par deux pour avoir le volume global accédé lors d'une opération, ce qui donne un volume total de $\frac{M_x \times M_y \times M_z \times 2^2 \times 8}{1024^2}$

On interprète à présent le Tableau 3.5 comme suit :

- les deux premières tailles de maillage $(128, 257, 8)$ et $(128, 257, 16)$ supposent le traitement de 4 et 8 Mo de données respectivement. Comme le processeur Intel X5675 comprend deux caches L3 de taille 12 Mo, cela nous indique que les données tiennent en cache L3 durant une opération FFT.
- Les deux tailles suivantes $(128, 257, 32)$ et $(128, 257, 64)$ donnent 32 et 64 Mo comme volume total à traiter, ce qui excède la taille des deux caches L3 et donc suppose des aller-retour en mémoire globale pour les données des tableaux à traiter. Ceci augmente d'autant les besoins en bande passante mémoire.

Le Tableau souligne clairement que plus la taille des TFD dépasse les tailles de cache L3, moins la parallélisation OpenMP est efficace. Bien que l'algorithme FFT soit connu pour avoir une haute intensité de calcul, le dernier résultat (taille 128x257x64, 12 threads) souligne que l'algorithme se heurte à une limitation de bande passante mémoire. Le dépassement de capacité du

cache L3 en est la cause, et c'est ce cas de figure qui se produit aussi dans Emedge3D.

Sur la SMP, le meilleur speedup atteint est 32.3 sur les 64 cœurs, pour une taille de domaine de $(1024, 65, 32)$. Néanmoins, les speedups en général ne sont pas aussi bons : ils varient de manière difficilement prédictible d'une taille de domaine à une autre. Pour une taille de domaine donnée, les speedups obtenus ne sont pas toujours croissants en fonction du nombre de threads. Ces résultats sont dus au fait que l'architecture de la machine SMP est plus complexe : deux coeurs en plus par noeud, effets NUMA et hiérarchie mémoire plus haute. Une analyse plus approfondie devrait être menée pour affiner l'analyse.

TABLE 3.5 – TFDs : Comparaison pour différentes tailles de maillage sur Intel X5675.

DFT Size	Threads	Time (mu s)	Speedup
128x257x**8**	1	14401.00	1.00
	2	7380.00	1.95
	4	3763.00	3.83
	8	1964.00	7.33
	12	1446.00	9.96
128x257x**16**	1	29094.00	1.00
	2	15431.00	1.89
	4	7806.00	3.73
	8	4020.00	7.24
	12	2970.00	9.80
128x257x**32**	1	61598.00	1.00
	2	32892.00	1.87
	4	17100.00	3.60
	8	9357.00	6.58
	12	7273.00	8.47
128x257x**64**	1	131233.00	1.00
	2	69716.00	1.88
	4	36929.00	3.55
	8	22447.00	5.85
	12	19193.00	6.84

Limitations

Dans les deux derniers paragraphes, nous avons montré que la parallélisation de la méthode d'Arakawa ainsi que des TFDs souffrent d'une limitation

de la bande passante mémoire.

Le tiling pourrait être utilisé comme en Sous Section 3.2.3 pour limiter les accès en mémoire globale. Malheureusement, les tailles de tuile applicables pour cette partie du code sont trop grandes pour améliorer la localité temporelle des données. En effet, la méthode d'Arakawa pour un indice radial donné idx nécessite la connaissance des données aux positions $idx - 1$ et $idx + 1$ en plus de l'indice idx (stencil dans la direction radiale). Ceci implique que les TFDs 2D (directions toroïdale et poloïdale) doivent être effectuées pour ces indices avant d'appeler la méthode d'Arakawa. Ajouté à cela, les TFDs 2D ne peuvent être décomposées, *i.e.* on ne peut pas les calculer sur des tuiles plus petites que la tranche de donnée 2D entière.

Remarque 3.3.2 *Dans le Chapitre 4, nous mettons en place une stratégie de parallélisation à grain plus fin, permettant d'améliorer la localité temporelle des données en mémoire. Ici, chaque thread travaille sur sa propre tranche de données 2D. Dans le Chapitre 4, les threads collaborent pour traiter la même tranche 2D.*

3.4 Performances globales et conclusion

Ce paragraphe présente un résumé des optimisations et parallélisations mises en place détaillées dans les sections précédentes. Le Tableau 3.6 montre un pas Runge Kutta sur Intel X5675, pour une taille de maillage $(512, 512, 128)$, qui représente un gros maillage pour les utilisateurs actuels. Le cas test ciblé ici est appelé RMP_C1 [44]. Ce cas test simule le phénomène de relaxation de barrière de transport au bord des tokamaks.

Le Tableau montre que les performances pour la partie linéaire du code sont satisfaisantes, exhibant un bon passage à l'échelle pour les speedups relatifs aux temps séquentiels optimisés.

Concernant la partie non linéaire, les speedups pour les transformées de Fourier restent acceptables. Mais ceux pour la méthode d'Arakawa restent très limités par sa faible intensité de calcul, expliquant l'augmentation de la part relative qu'occupe cet algorithme par rapport au code global. Aussi, il est visible que le speedup total de l'application est gouverné par le speedup des TFDs, ce qui est cohérent avec le poids que représente ces transformations.

Dans ce chapitre, nous avons abordé les aspects optimisation et parallélisation du code Emedge3D sur machine à mémoire partagée. Notre principal résultat ici est d'avoir atteint un speedup de 7.2 par rapport à la version séquentielle optimisée, soit 21.6 par rapport à la version initiale, sur les 12 cœurs du processeur Intel X5675. Cette amélioration a été obtenue en réduisant la limitation imposée par la bande passante mémoire lors de l'ajout de threads.

TABLE 3.6 – Résultats Emedge3D pour l'optimisation et la parallélisation sur Intel X5675. $\text{Speedup}_{\text{init}}$ est obtenu relativement aux temps initiaux, $\text{Speedup}_{\text{opt}}$ relativement aux temps optimisés. Taille de maillage : $(512, 512, 128)$

Computations		DFTs	Arakawa	Linear part	Total
Initial (seq)	Time (s)	13.47	2.29	5.30	21.06
	Percent	63.9	10.9	25.2	100
	Speedup	1	1	1	1
Optimized (seq)	Time (s)	4.10	1.34	1.70	7.14
	Percent	57.5	18.7	23.8	100
	$\text{Speedup}_{\text{init}}$	3.3	1.7	3.1	3
Parallel	Time (s)	0.58	0.25	0.16	0.99
	Percent	58.6	25.2	16.2	100
	$\text{Speedup}_{\text{opt}}$	7.0	5.3	10.5	7.2
	$\text{Speedup}_{\text{init}}$	23.1	9.0	32.6	21.6

La principale stratégie employée a été la maximisation de la réutilisation des données chargées dans les caches du processeur, et principalement au niveau du cache L3. Pour ce faire, nous avons optimisé les opérations en mémoire, en "pipelinant" les chargements de tableaux à l'intérieur des boucles, en réordonnant les appels de fonction. Nous avons aussi réduit le nombre de calculs, notamment pour la méthode numérique d'Arakawa. Le tiling s'est révélé particulièrement efficace et crucial pour la partie linéaire du code, permettant la réutilisation des données en cache d'un opérateur à un autre.

Pour la partie non linéaire, l'accélération parallèle reste limitée par la bande passante mémoire. Pour augmenter la bande passante mémoire et éviter les conflits de cache, une possibilité est de passer sur machine à mémoire distribuée, afin de bénéficier d'une bande passante cumulée sur l'ensemble des nœuds. Ceci fait l'objet d'une étude dans le Chapitre 4.

Chapitre 4

Parallélisation hybride MPI/OpenMP de l'équation d'advection diffusion

Suite aux résultats obtenus pour la parallélisation sur architecture à mémoire partagée pour les parties linéaire et non linéaire du code Emedge3D au Chapitre 3, nous souhaitons proposer une solution de parallélisation sur architecture à mémoire distribuée. Le but est ici d'augmenter les ressources en bande passante mémoire qui faisaient défaut en ce qui concernait la partie non linéaire du code Emedge3D. Il s'agira également de changer la stratégie de parallélisation et donc la représentation des données en mémoire (ordre de stockage des dimensions de l'espace) afin de distribuer les données sur les différents nœuds de calcul conformément aux dépendances des calculs. Le grain de parallélisme OpenMP mis en place au Chapitre 3 sera affiné, notamment en utilisant des transformées de Fourier 1D. Ce grain plus fin permet de bénéficier de plus d'effets cache.

Néanmoins, les changements que nous envisageons étant lourds en terme de mise en place, nous ne le ferons pas directement dans le code Emedge3D comme au Chapitre 3, mais plutôt sur un code à part, résolvant le même type d'équations. Il s'agit d'une équation de type advection diffusion plus générale que celles traitées dans le code de référence. C'est pourquoi, dans ce chapitre, on souhaite construire et proposer un algorithme de parallélisation pour une équation de type advection diffusion similaire à l'équation sur la pression utilisée dans le code Emedge.

Pour cela, on considère pour l'inconnue $T = T(t, x, y, z)$ l'opérateur $\nabla^2_{\|} T = \nabla \cdot (A\nabla)T$ (avec $\nabla = (\partial_x, \partial_y, \partial_z)$ et A une matrice 3×3 à préciser), couplé avec un opérateur de transport $\{\phi, T\} = \partial_x \phi \partial_y T - \partial_y \phi \partial_x T$. Ainsi, on considère

l'équation suivante

$$\partial_t T + \{\phi, T\} = \nabla \cdot (A \nabla T), \quad x, y, z \in [-1, 1], t \geq 0. \tag{4.0.1}$$

avec A une matrice de diffusion 3×3.

Dans le code Emedge3D, l'équation de pression comporte la diffusion parallèle anisotrope contraignant la valeur du pas de temps, impliquant un grand nombre d'itérations temporelles pour une simulation. De plus, elle s'inscrit dans la partie non linéaire du code Emedge3D (présentée en Sous Section 3.1.2), dont les performances sont limitées en grande partie par la bande passante mémoire dans le cadre d'une parallélisation sur architecture à mémoire partagée.

Nous visons ici une parallélisation hybride MPI-OpenMP pour l'équation d'advection diffusion appliquée à Emedge3D. L'augmentation du nombre de nœuds de calcul permet d'augmenter la valeur cumulée de la bande passante mémoire. Cette parallélisation hybride, couplée à un schéma temporel de type semi-implicite, permettra d'augmenter la valeur du pas de temps, diminuant ainsi le nombre d'itération temporelles requise pour une simulation donnée (voir le Chapitre 2).

Sommaire

4.1 Les équations de type advection diffusion

Les équation d'advection diffusion se rencontrent fréquemment dans différents modèles physique. En guise d'illustration, nous pouvons citer les équa-

tions de Navier-Stokes, qui s'écrivent dans leur forme incompressible :

$$\rho\left(\frac{\partial v}{\partial t} + v \cdot \nabla v\right) = -\nabla p + \Delta v + f$$

où v est la vitesse du fluide, ρ la densité, p la pression, et f la force volumique. Ici, une advection-diffusion s'exerce sur la vitesse, sous forme d'un laplacien 3D pour la diffusion avec le terme Δv, et par les opérateur d'advection 3D $v \cdot \nabla$ et $-\nabla p$. Dans le cas que nous traitons dans ce chapitre, la partie advection est 2D, dans le plan (x, y), et se traduit sous la forme d'un crochet de Poisson. Le même opérateur est présent dans les équations pour Emedge3D.

D'un point de vue numérique, les problèmes d'advection diffusion sont abordées dans [57] par exemple, où une discrétisation spatiale d'ordre 4 en volume finis et des méthodes d'intégrations temporelles de type semi-implicite sont présentées. La diffusion y est toutefois restreinte à un laplacien. Nous abordons ici une forme plus évoluée, où la matrice de diffusion dépend de la direction radiale. Cette matrice, que nous nommons A_x, est de la forme :

$$A_x = \begin{pmatrix} a(x) & 0 & 0 \\ 0 & b(x) & d(x) \\ 0 & d(x) & c(x) \end{pmatrix}, \tag{4.1.2}$$

forme proche de la diffusion anisotrope du modèle d'Emedge3D.

Nous mettrons en place une discrétisation spatiale de type semi-spectrale, à la manière d'Emedge3D. Ce type de discrétisation est aussi présent dans différent codes pour la fusion nucléaire. Nous pouvons ici citer les codes GKV et GENE (modèles gyrocinétiques, japonnais et allemand respectivement), ainsi que dans les codes de simulation XTOR et JOREK (modèles fluides magnéto-hydrodynamique, tous deux français). Ces codes mettent également en place des stratégies de parallélisation sur architectures à mémoire partagées et distribuées, dans le cadre de discrétisation semi-spectrale pour GKV et GENE (voir les références [41] et [27] respectivement) notamment.

Dans Emedge3D, le terme d'advection-diffusion le plus coûteux en terme de calcul concerne l'équation de pression (voir [23, 24, 6, 5, 44]). Elle peut s'écrire sous la forme donnée dans l'Equation (4.0.1). La matrice A définie positive peut être décomposée sous la forme $A = A_x + \tilde{A}$, avec A_x définie par l'Equation (4.1.2). Cette dernière décomposition permet d'isoler la partie forte de la diffusion A_x afin de lui appliquer un traitement spécifique en terme de schéma temporel par une méthode de type semi-implicite (voir le Chapitre 2 ainsi que la référence [19]).

En ce qui concerne la discrétisation, on remarque dans un premier temps que la matrice de diffusion A_x ne dépend que de la variable x. De plus, les

variables y, z sont périodiques, une discrétisation spectrale est très appropriée. C'est cela que l'on souhaite exploiter dans ce travail. La partie restante de l'opérateur de diffusion pour l'équation de pression d'Emedge3D $A - A_x$ pourra être discrétisée à l'aide d'une méthode classique de type volumes finis.

L'objectif de ce chapitre est de proposer un schéma de parallélisation hybride MPI/OpenMP efficace couplé à un schéma numérique adapté aux contraintes du problème (CFL de diffusion sur le pas de temps). Dans un premier temps, il s'agira de valider numériquement le cas sans advection. On regardera alors en détail le problème préliminaire suivant :

$$\partial_t T = \nabla \cdot (A \nabla T),$$

avec des matrices de diffusion A de plus en plus complexes. Cette étude pourra être utilisée pour construire un schéma semi-implicite inconditionnellement stable. Elle sera aussi utilisée pour valider la parallélisation des méthodes numériques de manière incrémentale. Ensuite, une étude de performance sur le problème d'advection-diffusion complet $\partial_t T + \{\phi, T\} = \nabla \cdot (A_x \nabla T)$ sera menée.

4.2 Méthodes numériques

Nous présentons ici les méthodes numériques utilisées pour résoudre une équation d'advection-diffusion donnée par :

$$\partial_t T + \{\phi, T\} = \nabla \cdot (A_x \nabla T) \tag{4.2.3}$$

4.2.1 Discrétisation spatiale

Rappelons ici que l'on considère des conditions périodiques en (y, z) et Dirichlet en x, de manière similaire au modèle Emedge3D.

Terme d'advection Pour les mêmes raisons qu'en Sous Section 3.1.2, le terme d'advection est calculé dans la représentation directe (et non en représentation semi-spectrale) avec une discrétisation de type différences finies. Le calcul s'effectue grâce au schéma d'Arakawa d'ordre 2, comme dans Emedge3D. Ce schéma est souvent utilisé dans la communauté de la physique des plasmas pour résoudre les opérateurs de crochet de Poisson, en raison de ses propriétés de conservation d'énergie (voir [1]). Cette discrétisation met en jeu les directions x et y, la variable z joue le rôle d'un paramètre. Le schéma d'Arakawa est un schéma aux différences finies, qui pour $\phi = \phi_{i,j,k} = \phi(x_i, y_j, z_k)$ et $T = T_{i,j,k} = T(x_i, y_j, z_k)$ donnés s'écrit dans sa forme 2D :

$$
\begin{aligned}
\{\phi, T\}_{i,j,k} \;=\; & \left[\left(T_{i,j-1,k} + T_{i+1,j-1,k} - T_{i,j+1,k} - T_{i+1,j+1,k}\right)\left(\phi_{i+1,j,k} + \phi_{i,j,k}\right)\right. \\
& - \left(T_{i-1,j-1,k} + T_{i,j-1,k} - T_{i-1,j+1,k} - T_{i,j+1,k}\right)\left(\phi_{i,j,k} + \phi_{i-1,j,k}\right) \\
& + \left(T_{i+1,j,k} + T_{i+1,j+1,k} - T_{i-1,j,k} - T_{i-1,j+1,k}\right)\left(\phi_{i,j+1,k} + \phi_{i,j,k}\right) \\
& - \left(T_{i+1,j-1,k} + T_{i+1,j,k} - T_{i-1,j-1,k} - T_{i-1,j,k}\right)\left(\phi_{i,j,k} + \phi_{i,j-1,k}\right) \\
& + \left(T_{i+1,j,k} - T_{i,j+1,k}\right)\left(\phi_{i+1,j+1,k} + \phi_{i,j,k}\right) \\
& - \left(T_{i,j-1,k} - T_{i-1,j,k}\right)\left(\phi_{i,j,k} + \phi_{i-1,j-1,k}\right) \\
& + \left(T_{i,j+1,k} - T_{i-1,j,k}\right)\left(\phi_{i-1,j+1,k} + \phi_{i,j,k}\right) \\
& \left. - \left(T_{i+1,j,k} - T_{i,j-1,k}\right)\left(\phi_{i,j,k} + \phi_{i+1,j-1,k}\right)\right] \times \left(-\frac{1}{12\Delta x \Delta y}\right).
\end{aligned}
$$

où x_i, y_j, z_k désigne la grille en espace. Cette méthode numérique a déjà fait l'objet d'une étude dans la Sous Section 3.3.2. Il existe aussi une version plus complexe à l'ordre 4, présentée dans la référence [1].

Terme de diffusion L'opérateur de diffusion A_x que nous souhaitons résoudre, caractérisé par les fonctions $a(x), b(x), c(x)$ et $d(x)$ dans l'Equation (4.1.2) s'écrit de façon particulièrement simple en variables de Fourier. Ainsi, l'inconnue T est développée en série de Fourier dans les directions (y, z) de sorte que l'on manipule la quantité $\hat{T}(x_i, m, n)$ où x_i désigne les points de la grille dans la direction x et (m, n) les modes de Fourier en (y, z). On résout alors l'opérateur de diffusion en discrétisation semi-spectrale par le schéma suivant :

$$
\begin{aligned}
\partial_t \hat{T}(x_i, m, n) \;=\; & a(x_{i+1/2}) \frac{\hat{T}(x_{i+1}, m, n) - \hat{T}(x_i, m, n)}{\Delta x^2} \\
& - a(x_{i-1/2}) \frac{\hat{T}(x_i, m, n) - \hat{T}(x_{i-1}, m, n)}{\Delta x^2} \\
& - \Big(b(x_i) m^2 + c(x_i) n^2 + 2 d(x_i) mn \Big) \hat{T}(x_i, m, n),
\end{aligned}
$$

où Δx désigne la taille du maillage en x. Notons que, dans ce cas, on est amené à utiliser une approximation pour le terme $\partial_x(a(x)\partial_x T)$. On utilise un schéma de type volumes finis d'ordre 2 même si l'extension à un schéma d'ordre supérieur ne présenterait aucune difficulté.

4.2.2 Discrétisation temporelle

Afin de résoudre l'Equation d'advection-diffusion (4.2.3), un objectif que nous ajoutons est d'utiliser des schémas de type implicite ou semi-implicite afin

d'éviter la contrainte sur le pas de temps $\Delta t \simeq C \min(\Delta x^2, 1/m_{\max}^2, 1/n_{\max}^2)$, où C désigne l'amplitude maximale du coefficient de diffusion et $m_{\max}$ et $n_{\max}$ désignent les fréquences maximales en y et z, qui dépendent du problème physique.

Nous prenons le parti d'effectuer un splitting entre l'advection et la diffusion, au regard des différentes discrétisations employées pour résoudre les deux opérateurs spatiaux (espace réel et espace semi-spectral). On note $T^k = T^k(x, y, z)$ la solution au temps $t^k = k\Delta t$ dans la représentation directe, et $\hat{T}^k = \hat{T}^k(x, m, n)$ la solution au temps $t^k = k\Delta t$ dans la représentation semi-spectrale, où m (respectivement n) désigne le numéro de mode dans la direction poloïdale (respectivement toroïdale). Ainsi, on considère dans un premier temps l'advection seule :

$$\partial_t T + \{\phi, T\} = 0,$$

que nous choisissons de résoudre à l'aide d'une méthode d'Euler explicite classique :

$$T^* = T^k + \Delta t \left\{\phi, T^k\right\},$$

qui peut aisément être étendue à un schéma temporel d'ordre plus élevé (méthode de Runge-Kutta par exemple). La deuxième étape consiste à résoudre la partie diffusion du problème :

$$\partial_t \hat{T} = \nabla \cdot (A \nabla \hat{T}).$$

Rappelons ici que la diffusion est résolue dans la représentation semi-spectrale. C'est également dans cet espace que le schéma temporel sera appliqué. Nous proposons différents schémas pour cette partie du splitting.

Avant d'énumérer les schémas temporels, on définit deux matrices carrées $\mathcal{A}$ et $\mathcal{D}$ de taille N_x :

$$\mathcal{A} = \frac{1}{\Delta x^2} \text{ tridiag } (a(x_{i-1/2}), -[a(x_{i+1/2}) + a(x_{i-1/2})], a(x_{i+1/2})),$$

correspondant à la partie radiale de la diffusion pour un mode (m, n) donné et

$$\mathcal{D} = -(b(x_i)m^2 + c(x_i)n^2 + 2d(x_i)mn)Id$$

pour la partie spectrale de la diffusion.

Schéma temporel d'Euler explicite

$$\hat{T}^{k+1}(\cdot, m, n) = \hat{T}^*(\cdot, m, n) + \Delta t(\mathcal{A} + \mathcal{D})\hat{T}^*(\cdot, m, n).$$

Les schémas de type Runge-Kutta peuvent se déduire aisément. Ce type de schémas nécessite l'utilisation de pas de temps vérifiant

$$\Delta t \leq C \min(\Delta x^2, 1/m_{\max}^2, 1/n_{\max}^2).$$

Schéma temporel semi-implicite d'ordre 1

$$\hat{T}^{k+1}(\cdot, m, n) = \hat{T}^{*}(\cdot, m, n) + \Delta t(\mathcal{A} + \mathcal{D})\hat{T}^{k+1}(\cdot, m, n).$$

On est amené à inverser une matrice de taille N_x pour chaque mode de Fourier (m, n)

$$\hat{T}^{k+1}(\cdot, m, n) = (Id - \Delta t(\mathcal{A} + \mathcal{D}))^{-1}\hat{T}^{*}(\cdot, m, n),$$

où Id désigne la matrice identité de taille N_x. Dans notre cas, la matrice $Id - \Delta t(\mathcal{A} + \mathcal{D})$ est tridiagonale à cause du schéma d'ordre 2 mis en place dans la direction x. Ceci permet de l'inverser de manière simple et rapide par une méthode LU 1D par exemple. On peut étendre aux ordres supérieurs en utilisant des schémas de types Additive Runge Kutta (voir la Sous Section 2.2.2). Rappelons ici que ce type de schéma permet d'utiliser des pas de temps plus grands, étant donné qu'ils sont inconditionnellement stable du point de vue de l'opérateur de diffusion. Nous pouvons donc utiliser des nombres de Courant-Friedrich-Levy plus grands que 1 (voir l'analyse de stabilité en Section 2.4).

4.3 Vérification incrémentale de l'implémentation

On valide l'implémentation de l'opérateur de diffusion $\nabla \cdot A_x \nabla$ de façon incrémentale, pour les schémas temporels présentés dans la section précédente. Pour le crochet de Poisson, on considère $\phi(x, y) = \cos(\pi x)\cos(\pi y)$. Pour rappel, l'équation à résoudre est :

$$\partial_t T + \{\phi, T\} = \nabla \cdot (A_x \nabla T) + f,$$

où $f = f(t, x, y, z)$ est une fonction donnée, qui jouera le rôle de terme source dans les cas test que nous présentons. Dans la suite, nous proposons différents sous-modèles de complexité croissante, du 1D au 3D, pour lesquels la solution analytique suivante sera utilisée

$$T(t, x, y) = 1 + \sin(\pi x)\sin(\pi y)\sin(\pi z)e^{-t}, \quad x, y, z \in [-1, 1], t \geq 0. \qquad (4.3.4)$$

Ainsi,

$$\begin{aligned}
\partial_t T &= -(T-1) \\
\partial_x T &= \pi \cos(\pi x)\sin(\pi y)\sin(\pi z)e^{-t} \\
\partial_y T &= \pi \sin(\pi x)\cos(\pi y)\sin(\pi z)e^{-t} \\
\partial_y^2 T &= -\pi^2 \sin(\pi x)\sin(\pi y)\sin(\pi z)e^{-t} = -\pi^2(T-1) \\
\partial_x^2 T &= -\pi^2 \sin(\pi x)\sin(\pi y)\sin(\pi z)e^{-t} = -\pi^2(T-1) \\
\partial_{x,y} T &= \pi^2 \cos(\pi x)\cos(\pi y)\sin(\pi z)e^{-t} \\
\partial_{y,z} T &= \pi^2 \sin(\pi x)\cos(\pi y)\cos(\pi z)e^{-t} \\
\partial_z^2 T &= = -\pi^2(T-1).
\end{aligned}$$

Et, pour les dérivées partielles de ϕ :

$$\begin{aligned}
\partial_x \phi &= -\pi \sin(\pi x)\cos(\pi y) \\
\partial_y \phi &= -\pi \cos(\pi x)\sin(\pi y)
\end{aligned}$$

Cas 1

Dans un premier temps, on considère la matrice A_1 de la forme

$$A_1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

de sorte que l'Equation (4.0.1) se réécrive

$$\partial_t T + \{\phi, T\} = \partial_y^2 T.$$

On a alors que l'Equation (4.3.4) est la solution analytique de

$$\partial_t T + \{\phi, T\} = \partial_y^2 T + f,$$

avec

$$\begin{aligned}
f(t,x,y,z) &= \partial_t T + \{\phi, T\} - \partial_y^2 T \\
&= -(T-1) + \partial_x \phi \partial_x T - \partial_y \phi \partial_y T + \pi^2 (T-1).
\end{aligned}$$

Cas 2 **:**

On considère la matrice A_2 de la forme

$$A_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

de sorte que l'Equation (4.0.1) se réécrive

$$\partial_t T + \{\phi, T\} = \partial_y^2 T + \partial_z^2 T.$$

On a alors que l'Equation (4.3.4) est la solution analytique de

$$\partial_t T + \{\phi, T\} = \partial_y^2 T + \partial_z^2 T + f,$$

avec $f(t,x,y,z) = -(T-1) + \partial_x\phi\partial_x T - \partial_y\phi\partial_y T + 2\pi^2(T-1)$.

Cas 3

On considère la matrice A_3 de la forme

$$A_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & b(x) & 0 \\ 0 & 0 & c(x) \end{pmatrix},$$

de sorte que l'Equation (4.0.1) se réécrive

$$\partial_t T + \{\phi, T\} = b(x)\partial_y^2 T + c(x)\partial_z^2 T.$$

On a alors que l'Equation (4.3.4) est la solution analytique de

$$\partial_t T + \{\phi, T\} = b(x)\partial_y^2 T + c(x)\partial_z^2 T + f,$$

avec $f(t,x,y,z) = -(T-1) + \partial_x\phi\partial_x T - \partial_y\phi\partial_y T + b(x)\pi^2(T-1) + c(x)\pi^2(T-1)$.
On choisit dans notre cas

$$b(x) = (2+\sin(\pi x))^2, \quad c(x) = (2+\cos(\pi x))^2.$$

Cas 4

On considère la matrice A_4 de la forme

$$A_4 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & b(x) & d(x) \\ 0 & d(x) & c(x) \end{pmatrix},$$

de sorte que l'Equation (4.0.1) se réécrive

$$\partial_t T + \{\phi, T\} = b(x)\partial_y^2 T + c(x)\partial_z^2 T + 2d(x)\partial_{y,z}^2 T.$$

On a alors que l'Equation (4.3.4) est la solution analytique de

$$\partial_t T + \{\phi, T\} = b(x)\partial_y^2 T + c(x)\partial_z^2 T + 2d(x)\partial_{y,z}^2 T + f,$$

avec

$$\begin{aligned} f(t,x,y,z) &= -(T-1) + \partial_x\phi\partial_x T - \partial_y\phi\partial_y T + b(x)4\pi^2(T-1) \\ &\quad + c(x)\pi^2(T-1) - 2d(x)\partial_{y,z}^2 T. \end{aligned}$$

On considère ici

$b(x) = (2+\sin(\pi x))^2$, $c(x) = (2+\cos(\pi x))^2$, $d(x) = (2+\sin(\pi x))(2+\cos(\pi x))$.

Cas 5

Dans ce cas, on considère un cas 3D qui permet d'utiliser un schéma temporel de type semi-implicite pour l'Equation (4.0.1). En effet, on considère la matrice A_x la plus complète dans notre cadre, qui est de la forme :

$$A_x = \begin{pmatrix} a(x) & 0 & 0 \\ 0 & b(x) & d(x) \\ 0 & d(x) & c(x) \end{pmatrix},$$

de sorte que l'Equation (4.0.1) se réécrive

$$\partial_t T + \{\phi, T\} = \partial_x(a(x)\partial_x T) + b(x)\partial_y^2 T + c(x)\partial_z^2 T + 2d(x)\partial_{y,z}^2 T.$$

On a alors que l'Equation (4.3.4) est la solution analytique de

$$\partial_t T + \{\phi, T\} = \partial_x(a(x)\partial_x T) + b(x)\partial_y^2 T + c(x)\partial_z^2 T + 2d(x)\partial_{y,z}^2 T + f,$$

avec

$$\begin{aligned} f(t,x,y,z) \;=\; & -(T-1) + \partial_x\phi\partial_x T - \partial_y\phi\partial_y T + b(x)\pi^2(T-1) + c(x)\pi^2(T-1) \\ & -a'(x)\partial_x T - a(x)\partial_x^2 T - 2d(x)\partial_{y,z}^2 T. \end{aligned}$$

On prend pour coefficients

$$\begin{aligned} & a(x) = (2+\sin(\pi x)), \;\; b(x) = (2+\sin(\pi x))^2, \\ & c(x) = (2+\cos(\pi x))^2, \;\; d(x) = (2+\sin(\pi x))(2+\cos(\pi x)). \end{aligned}$$

Validation des méthodes numériques

Afin de valider les méthodes mises en place ainsi que l'implémentation, nous vérifions les ordres d'erreur en espace des 5 cas tests décrits plus haut. Ils ont ont été lancés pour 1000 itérations temporelles à un pas de temps de 10^{-8}. La Figure 4.1 montre les ordres d'erreur en espace pour les différents cas tests, obtenus avec la méthode d'Euler pour l'intégration en temps. On observe bien ici l'ordre 2 en espace, en conformité avec les méthodes numériques utilisées pour résoudre les opérateurs spatiaux. La méthode d'intégration temporelle semi-implicite (pour ncfl= 1) décrite en Sous Section 4.2.2 donnant les mêmes convergences en ordre, elles ne sont pas présentées ici. Ces courbes valident les méthodes ainsi que l'implémentation pour les cinq cas tests considérés.

4.4 Parallélisation OpenMP / MPI

L'algorithme parallèle pour l'advection diffusion peut se décomposer en 4 parties : l'advection (crochet de Poisson), la diffusion, les transformées de

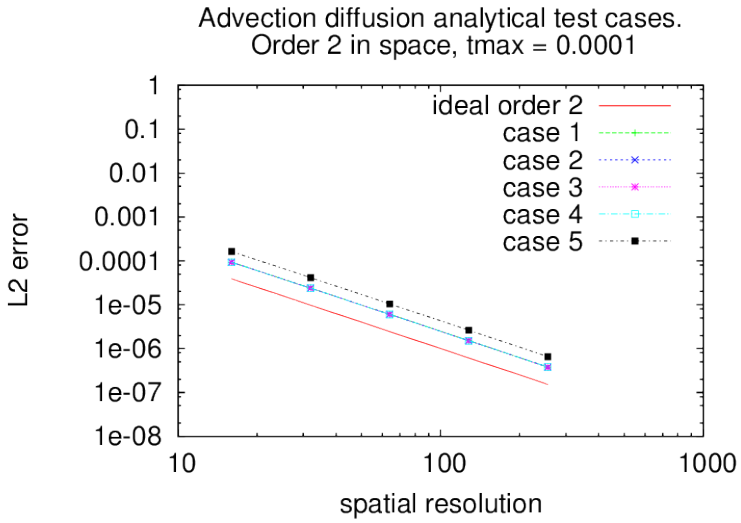


FIGURE 4.1 – Ordres de l'erreur en espace en norme L2 pour les cas tests 1 à 5 dans le cas de maillages carrés.

Fourier et les transferts ou restructurations de données. Dans la suite, nous allons présenter ces parties en explorant les possibilités de parallélisation dans le cas d'une machine à mémoire distribuée.

Le crochet de Poisson s'effectue dans la représentation directe (réelle) de l'inconnue. Il agit sur les variables x, y, impliquant le calcul d'un stencil 2D dans ces directions. La direction z n'est ici qu'un paramètre, et peut donc être la dimension selon laquelle on parallélise. La diffusion quant à elle mélange une discrétisation directe dans la direction x et spectrale dans la direction y, z. La résolution de la partie radiale de l'opérateur en un point x_i donné implique ses voisins x_{i-1} et x_{i+1}. Les directions y et z étant résolues dans l'espace spectral, elles ne mettent en jeu que la valeur de la cellule courante. Il est alors possible de distribuer les calculs et les données selon la direction y ou la direction z (ou même les deux directions). Entre le calcul des deux opérateurs, il est logique d'effectuer un changement de décomposition du domaine : on distribue les données et les calcul selon l'axe z pour l'advection, et selon l'axe y et / ou z pour la diffusion. De plus, un autre changement à lieu au même moment pour le passage de la représentation réelle des données à la représentation semi-spectrale, lors du calcul des transformées de Fourier. Ce changement de discrétisation s'effectue par des transformées de Fourier discrètes, via la bibliothèque FFTW3 ([22]). Elles agissent sur le plan (y, z), correspondant aux directions poloïdale et toroïdale. Nous prenons le parti d'effectuer ces FFTs dimension par dimension, en opposition avec les routines 2D fournies par la

bibliothèque. Cela présente un triple avantage :

- dans un premier temps, le fait de considérer des FFT 1D permet d'opérer sur un volume de donnée plus petit, plus propice pour bénéficier d'effets cache favorables,
- ensuite, cela offre plus de possibilités en ce qui concerne les optimisations et la parallélisation du code,
- enfin, ce changement n'occasionne pas de perte de performances dans le cas séquentiel. En particulier, nous serons capable de réutiliser des chargements en mémoire entre le calcul des FFT 1D et les autres parties de l'algorithme, ainsi que de mettre en place un parallélisme OpenMP à grain plus fin par rapport au Chapitre 3.

Enfin, les axes de parallélisme changeant entre les différentes parties (l'axe de parallélisation MPI est z pour les crochets de Poisson, et les FFTs 1D ne sont pas parallélisables selon cet axe), il reste à effectuer des transpositions et redistributions, afin d'accéder aux données selon les besoins. Ces transpositions dépendront de l'algorithme choisi pour résoudre le problème, et plus particulièrement de la manière de calculer les FFTs. Le Tableau 4.1 offre une synthèse des propositions exposées dans les derniers paragraphes.

TABLE 4.1 – Potentiel de parallélisation sur une machine à mémoire distribuée pour l'advection-diffusion dans le cadre d'Emedge3D. Les dépendances incluent les dépendances en lecture, considérées comme bloquantes pour une parallélisation sur machine à mémoire distribuée (même si des stratégies utilisant des cellules fantômes pourraient être néanmoins envisagées).

Etape	Axe	Dépendances en (i, j, k)	Parallélisation envisagée
Advection	x	$i-1, i, i+1$	non
	y	$j-1, j, j+1$	non
	z	k	oui
Diffusion	x	$i-1, i, i+1$	non
	y	j	oui
	z	k	oui
FFT 1D y	x	i	oui
	y	$j = *$	non
	z	k	oui
FFT 1D z	x	i	oui
	y	j	oui
	z	$k = *$	non

Dans la suite, nous présentons tout d'abord l'algorithme séquentiel pour

l'advection diffusion. Ensuite, nous détaillons deux algorithmes parallèles : une première version OpenMP sans parallélisation MPI, et une seconde hybride OpenMP / MPI. L'utilisation du parallélisme OpenMP à grain fin permet de tirer parti du caractère multi-coeur des nœuds de calcul. Ces algorithmes feront l'objet d'une étude de performances en terme de temps d'exécutions et de facteur d'accélération relatif aux ressources utilisées (speedup et efficacités) dans la Section 4.5. Pour rappel, on cherche à résoudre :

$$\partial_t T + \{\phi, T\} = \nabla \cdot (A_x \nabla T) + f,$$

avec A_x comme dans l'Equation (4.1.2), $\phi = \phi(x, y)$ donné, $T = T(t, x, y, z)$ l'inconnue et f un terme source. Pour plus de lisibilité et de simplicité dans l'écriture du code, nous avons pris le parti de séparer le terme source en deux parties, chacune correspondant à un des opérateur spatiaux : $f = f_{advection} + f_{diffusion}$. Les algorithmes qui suivent sont implémentés en langage C.

4.4.1 Algorithme séquentiel pour l'advection diffusion

Cette section détaille l'organisation de la boucle en temps opérée par le code pour l'advection diffusion. Elle reste inchangée quel que soit le type de parallélisation.

Afin d'effectuer une itération temporelle, deux tableaux $T_1[N_z, N_y, N_x]$ et $T_2[N_z, N_y, N_x]$ sont utilisés pour stocker la valeur du champ de température dans la représentation semi-spectrale des données. Ici, la notation $T_1[z, y, x]$ correspond à la valeur stockée en position $z * (N_y * N_x) + y * (N_x) + x$, avec N_d le nombre de points dans la direction d. Les étapes de ce moteur temporel sont détaillées dans l'Algorithme 7.

Algorithme 7 Advection diffusion : boucle en temps

Entrées : $\Delta t, T_1 = T(t^n), \phi = \cos(\pi x)\cos(\pi y)$

$T_2 \leftarrow T_1 + \Delta t \nabla \cdot (A_x \nabla T_1)$: Diffusion (dont FFT)

$T_2 \leftarrow T_2 + \Delta t f^n_{diffusion}$: Terme source de Diffusion

$T_1 \leftarrow T_2 - \Delta t \{\phi, T_2\}$: Advection

$T_1 \leftarrow T_1 + \Delta t f^n_{advection}$: Terme source d'advection

Sortie : $T_1 = T(t^{n+1})$

Les termes sources $f_{advection}$ et $f_{diffusion}$ étant connus analytiquement, ils sont calculés au besoin.

Les changements de base (réelle vers semi-spectrale et inversement) sont encapsulés dans l'étape de diffusion. Ainsi, l'étape de diffusion se divise en trois parties, qui vont être détaillées ci après. La première concerne les transformées

de Fourier "aller" (de $\mathbb{R}$ dans $\mathbb{C}$) dans la dimension spatiale y. Elle est détaillée dans l'Algorithme 8. Dans cet algorithme, $\text{buffer}_y[*]$ et $\text{buffer2}_y[*]$ sont des buffers de taille N_y, utilisés pour stocker les tranches $(z, y = *, x)$ de manière contigüe en mémoire, afin de pouvoir appliquer les FFT dans la dimension y sur ces tableaux. Le mot clef aller_y dans l'appel de la fonction fft() désigne à la fois le sens de la transformée de Fourier ainsi que la dimension sur laquelle elle s'applique. De même, les notations aller_z, retour_y et retour_z seront utilisées. Enfin, la présence des $*$ désigne une opération vectorielle sur la dimension que celle-ci remplace. Cette notation reviendra aussi dans les algorithmes suivants. La notation $\hat{T}$ signifie que les données sont en représentation semi-spectrale en y. Notons que l'ordre de stockage des dimensions change entre l'entrée et la sortie de l'Algorithme 8, passant de $[z, y, x]$ à $[y, z, x]$. Nous opérons cette transposition à la volée, car les étapes de diffusion et de transformées de Fourier dans la direction z (aller et retour) sont encapsulées dans une même boucle en y.

Algorithme 8 Diffusion : transformées de Fourier en y aller

Entrée : $T_1[z, y, x] = T(t^n)$

```
for all z do
  for all x do
    buffer_y[*] ← T_1[z, *, x]
    buffer2_y[*] ← fft(aller_y, buffer_y[*])
    T_2[*, z, x] ← buffer2_y[*]
  end for
end for
```

Sortie : $T_2[y, z, x] = \hat{T}(t^n)$

L'Algorithme 9 détaille les étapes de transformées de Fourier pour la dimension spatiale z (aller et retour) et la diffusion. Ici, Le buffer $\text{buffer}_z[*]$ est de taille N_z, utilisé pour les FFT dans la direction z. Notons que les FFT sont appliquées en place (tous les calculs se font dans le même tableaux). Ceci permet de bénéficier d'effets caches favorables, le tableau 1D $\text{buffer}_z[*]$ jouissant d'une bonne localité temporelle en mémoire. Les buffers $\text{in}_{xz}[x, *]$ et $\text{out}_{xz}[x, *]$ sont des zones de mémoire servant à stocker une tranche de donnée 2D (x, z), $\text{in}_{xz}[x, *]$ servant en entrée du calcul effectif de diffusion et $\text{out}_{xz}[x, *]$ en sortie.

Enfin, l'Algorithme 10 détaille les transformées de Fourier pour la dimension y dans la direction retour (de $\mathbb{C}$ dans $\mathbb{R}$). Il est similaire à l'Algorithme 8, les étapes étant juste inversées.

Algorithme 9 Diffusion : transformées de Fourier en z (aller et retour) et diffusion

Entrée : $T_2[y, z, x] = \hat{T}(t^n)$

for all y **do**
  for all x **do**
    $\text{buffer}_z[*] \leftarrow T_2[y, *, x]$
    $\text{buffer}_z[*] \leftarrow \text{fft}(\text{aller}_z, \text{buffer}_z[*])$
    $\text{in}_{xz}[x, *] \leftarrow \text{buffer}_z[*]$
  end for
  for all x **do**
    $\text{out}_{xz}[x, *] \leftarrow \text{in}_{xz}[x, *] + \Delta t \nabla \cdot (A_x \nabla \text{in}_{xz}[x, *])$
  end for
  for all x **do**
    $\text{buffer}_z[*] \leftarrow \text{out}_{xz}[x, *]$
    $\text{buffer}_z[*] \leftarrow \text{fft}(\text{retour}_z, \text{buffer}_z[*])$
    $T_2[y, *, x] \leftarrow \text{buffer}_z[*]$
  end for
end for

Sortie : $T_2[y, z, x] = \hat{T}(t^n) + \Delta t \nabla \cdot (A_x \nabla \hat{T}(t^n))$

Algorithme 10 Diffusion : transformées de Fourier en y retour

Entrées : $T_2[y, z, x] = \hat{T}(t^n) + \Delta t \nabla \cdot (A_x \nabla \hat{T}(t^n))$

for all z **do**
  for all x **do**
    $\text{buffer2}_y[*] \leftarrow T_2[*, y, x]$
    $\text{buffer}_y[*] \leftarrow \text{fft}(\text{retour}_y, \text{buffer2}_y[*])$
    $T_1[z, *, x] \leftarrow \text{buffer}_y[*]$
  end for
end for

Sortie : $T_1 = T(t^n) + \Delta t \nabla \cdot (A_x \nabla T(t^n))$

4.4.2 Version parallèle OpenMP

Dans un premier temps, nous nous intéressons à une version parallèle OpenMP. Celle-ci va permettre de mettre en exergue la limitation de la bande passante mémoire exhibée dans le Chapitre 3 en ce qui concernait la partie non linéaire du code Emedge3D.

En comparaison avec le Chapitre 3, la stratégie de parallélisation reste globalement la même. Les opérateurs en espace incluent un nid de boucle de profondeur 3 (un pour chaque dimension de l'espace), et la directive de par-

allélisation est placée au niveau de la boucle la plus externe. Toutefois, forts du bilan de la parallélisation des parties du code Emedge3D, plus particulièrement en ce qui concerne la partie non linéaire du code (voir le Tableau 3.6 page 111), nous avons mis en place ici un parallélisme à grain plus fin par l'ajout de la clause de boucle imbriquée OpenMP collapse(2), permettant la fusion du niveau de boucle intermédiaire avec la boucle externe, distribuant ainsi les itérations sur 2 dimensions de l'espace (au lieu d'une seule précédemment). Cette mise en place n'était pas possible dans Emedge3D, impliquant une lourde refonte des parties du code concernées.

La boucle en temps reste inchangée, restant comme présentée en Sous Section 4.4.1. Les quatre étapes dont elle est composée sont parallélisées à l'aide du paradigme OpenMP. En ce qui concerne la méthode d'Arakawa ainsi que le traitement des termes sources d'advection et de diffusion, les directives de parallélisation sont placées au niveau de la boucle la plus externe (*i.e.* la boucle itérant sur la direction z). La clause collapse(2) étant présente, ceci permet la distribution des itérations correspondant aux dimensions z et y sur les différents threads. En ce qui concerne la diffusion, pour les Algorithmes 8 et 10, la directive de parallélisation OpenMP est placée au niveau de la boucle externe sur la dimension z (similairement aux étapes d'advection et de traitement des termes sources). Pour la partie FFT en z et diffusion, on place la directive au niveau de la boucle intermédiaire sur la dimension x. En effet, cela permettra de placer les communications ainsi que les FFT dans la direction z et la diffusion à l'intérieur d'une même boucle sur la dimension y (pour améliorer la localité temporelle des tranches de données en y), ceci car les communications ne feront pas l'objet d'une parallélisation OpenMP. On appelle cette version `seq-optim`.

4.4.3 Version parallèle hybride OpenMP / MPI

Nous abordons ici la parallélisation MPI du code pour l'équation d'advection diffusion. L'algorithme reste proche de celui présenté dans la Sous Section précédente 4.4.2, mais avec les données distribuées sur les différents processus MPI. Il y a deux dimensions selon lesquelles les données sont distribuées, en fonction du traitement que l'on souhaite opérer. Le premier axe de distribution est la dimension z. Les données sont organisées dans l'ordre $[z, y, x]$, et les tranches 2D $[y, x]$ sont uniformément réparties par bloc sur les différents processus. Cette distribution concerne la méthode d'Arakawa, le traitement des termes sources d'advection et de diffusion, ainsi que les Algorithmes 8 et 10. L'étape correspondant à l'Algorithme 9, (FFT en z et diffusion) est distribuée selon la dimension y. En effet, afin d'effectuer les FFT 1D dans la direction z, il est nécessaire d'avoir connaissance de tous les points en z par tranche 2D

(x, y), d'où ce choix.

Il y a donc deux étapes de transposition des tableaux impliquant des communications MPI inter-nodales, à ajouter dans l'Algorithme 9. Ces communications sont effectuées en mode non bloquant, à l'aide des fonctions MPI ISend et IRecv. Elles sont effectuées sans parallélisation OpenMP (par l'utilisation de la directive master). Nous avons envisagé trois versions pour cet algorithme, mettant en œuvre plusieurs schémas de communications. Nous ne présenterons ici que la dernière version, qui est aussi la plus efficace et la plus économe en terme de transfert de données. Elle effectue les communications à l'intérieur de la boucle en y (au début et à la fin).

Les envois de données sont distribués dans la dimension y. Le passage de la distribution en z à la distribution en y pour un indice iy donné est détaillé par l'Algorithme 11. Nous ne présentons pas le passage à la distribution inverse, le procédé étant symétrique.

Algorithme 11 Communications : distribution en z vers distribution en y pour un indice iy donné

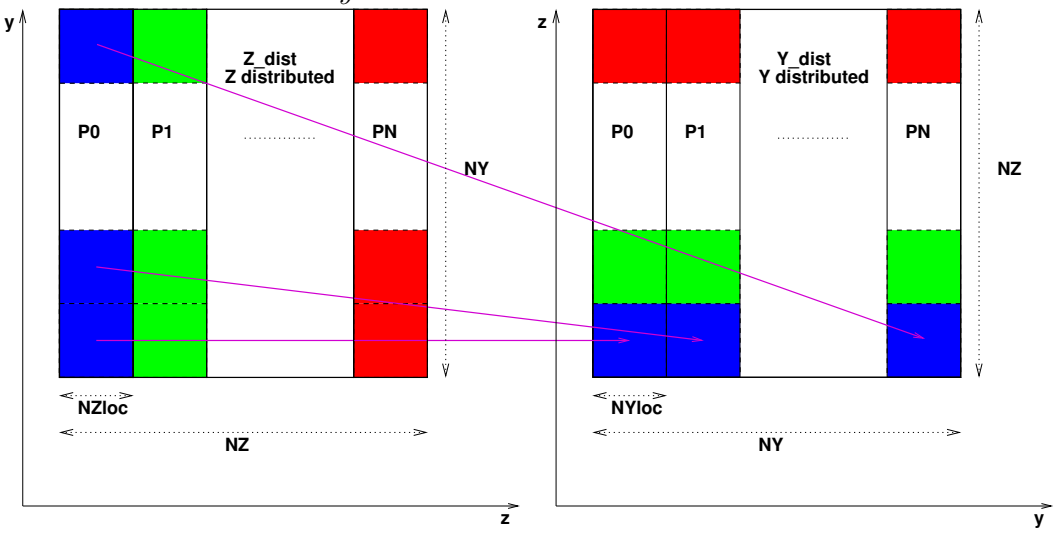
Entrées : Z_dist[NY,NZloc,NX],iy

pid ← current process rank
for all process p ≠ pid **do**
Irecv from process p in Y_dist[iy,NZloc × process,NX])
end for
for all process p ≠ pid **do**
Isend to process p data Z_dist[iy × process,NZloc,NX])
end for

Sortie : Y_dist[iy,NZ,NX]

La Figure 4.2 montre les communications à effectuer pour passer d'une distribution des données dans la direction z vers une distribution des données dans la direction y. Cette Figure représente schématiquement la représentation des données pour les deux distributions : un tableau Z_dist pour la distribution en z sur la gauche et un tableau Y_dist pour la distribution en y sur la droite, tout deux distribués sur les N processus MPI P0, P1, ... PN. La direction x joue le rôle d'un paramètre, elle n'est pas explicitée sur le schéma. Les données sont stockées dans l'ordre $[y, z, x]$ dans les deux cas. Pour le tableau Z_dist (respectivement Y_dist), chaque processus dispose des données de température [0..NY,0..NZloc,0..NX] (respectivement [0..NYloc,0..NZ,0..NX]), avec NX, NY et NZ le nombre de points dans chaque direction et NZloc (respectivement NYloc) le nombre de points pour la dimension z (respectivement y) par processus MPI dans la distribution en z (respectivement y). Pour passer de la distribution en z à la distribution en y, on divise la grille en blocs élémentaires

FIGURE 4.2 – Communications MPI : distribution des données en z vers distribution des données en y



de taille [NYloc, NZloc, NX]. Sur la Figure 4.2, ces blocs correspondent aux blocs de couleur. Chaque processus doit alors échanger des blocs avec tous les autres processus, pour passer d'une distribution à l'autre. Ainsi, le processus P0 distribue ses blocs de couleur bleue dans la distribution Z_dist à tous ses voisins, éclatant les blocs sur l'ensemble des processus MPI dans la distribution Y_dist. Il en va de même pour P1 en vert et jusqu'à PN en rouge. Les envois de données sont symbolisés par les flèches sortantes.

Enfin, l'Algorithme 12 détaille les étapes de communications, les FFT dans la direction z et la diffusion pour la version distribuée du code. Dans cet algorithme, le mot clef comm_{zy} renvoie à l'étape de changement de distribution des données en z vers la distribution en y (voir l'Algorithme 11), et inversement pour le mot clef comm_{yz}.

La version présentée dans cette Sous Section s'appelle `mpi-optim` dans la suite.

4.5 Résultats et performances

Dans cette section, nous présentons les résultats et performances des algorithmes précédemment décrits pour résoudre l'équation de type advection diffusion. Ces résultats ont été obtenus en effectuant 10 itérations temporelles du cas test 5 (présenté dans le Paragraphe 4.3), pour deux tailles de maillage différentes en ce qui concerne le cas multi-nœud ($(256, 256, 128)$ et

Algorithme 12 Communications, FFT en z et diffusion en distribué

Entrées : Z_dist$[y, z, x] = \hat{T}(t^n)$
  pid ← current process rank
  for all y local to process pid **do**
    comm$_{zy}$: Y_dist[y,*,*] ← Z_dist[y,*,*]
    for all x **do**
      buffer$_z[*]$ ← Y_dist$[y, *, x]$
      buffer$_z[*]$ ← fft(aller$_z$, buffer$_z[*]$)
      in$_{xz}[x, *]$ ← buffer$_z[*]$
    end for
    for all x **do**
      out$_{xz}[x, *]$ ← in$_{xz}[x, *] + \Delta t \nabla \cdot (A_x \nabla \text{in}_{xz}[x, *])$
    end for
    for all x **do**
      buffer$_z[*]$ ← out$_{xz}[x, *]$
      buffer$_z[*]$ ← fft(retour$_z$, buffer$_z[*]$)
      Y_dist$[y, *, x]$ ← buffer$_z[*]$
    end for
    comm$_{yz}$: Z_dist[y,*,*] ← Y_dist[y,*,*]
  end for
Sortie : Z_dist$[y, z, x] = \hat{T}(t^n) + \Delta t \nabla \cdot (A_x \nabla \hat{T}(t^n))$

$(1024, 1024, 512))$, et un seul maillage de taille $(256, 256, 128)$ pour le cas d'un seul nœud de calcul. Le schéma temporel choisi pour cette étude de performances est la méthode d'Euler uniquement.

Nous utilisons ici le compilateur fourni par Intel ainsi que la bibliothèque Open MPI version Intel 1.6.3 (machine de Marseille) et BullxMPI (machine Helios). Le programme est notamment compilé avec les options -O2 pour les optimisations et -axSSE4 pour la vectorisation du code. Nous utilisons deux machines parallèles. la première est le cluster de la machine Rhéticus du mésocentre d'Aix-Marseille Université, déjà utilisée dans le Chapitre 3. Pour rappel, elle regroupe 1152 cœurs Intel X5675 organisés en nœuds de 12 cœurs (voir Figure 1.5 page 41). La seconde est Helios, présentée page 39. Ses nœuds de calcul (au nombre de 92) sont composés de processeurs Sandy Bridge E5-2670 (2.60GHz, 8 cœurs par processeur, soit 16 cœurs par nœud), incluant 64 Go de mémoire par nœud.

Les tableaux que nous présenterons dans cette section comprendront, dans l'ordre :

- le nombre d'unités de calcul : NCU (avec NCU = NTH × NP),
- le nombre de threads OpenMP : NTH,

- le nombre de processus MPI : NP,
- le temps d'exécution en microsecondes : Time,
- l'accélération relative au nombre d'unités de calcul : Speedup,
- l'efficacité relative au nombre d'unités de calcul : Eff %,
- et le pourcentage de temps passé relatif au temps total de la boucle en temps de l'algorithme : Total %.

Notons aussi que l'analyse des performances ne prennent pas en compte les temps d'initialisation, ni ceux des entrées sorties. Par défaut, nous présentons les résultats obtenus sur la machine du mésocentre d'Aix-Marseille Université.

Dans un premier temps, nous montrerons les performances dans le cas d'une parallélisation OpenMP seule, puis les versions MPI et OpenMP sur un nœud, avant de passer aux performances sur plusieurs nœuds de calcul.

4.5.1 Versions parallèles OpenMP

Cette section a pour but l'évaluation de la mise en place de la parallélisation OpenMP à granularité plus fine (en contraste avec l'étude du Chapitre 3), obtenue par l'ajout de clauses collapse(2) ainsi que la réalisation des FFT en deux étapes (1D × 1D). Nous présentons ici des résultats dans le cas d'un maillage de taille $(N_x, N_y, N_z) = (256, 256, 128)$, la taille du maillage n'influant que peu sur les résultats.

Le Tableau 4.2 montre les résultats en terme de temps d'exécution, ainsi que le speedup et l'efficacité du code, pour la boucle en temps. Le nombre de processus MPI vaudra toujours 1 ici. Ce tableau montre que nous sommes capables d'atteindre un speedup de 7.7 lors de l'utilisation des 12 coeurs d'un nœud de calcul, pour une efficacité de 64 %. La décroissance de l'efficacité que l'on observe dans ce tableau provient à nouveau des besoins croissants en bande passante mémoire lors de l'ajout d'unités de calcul.

TABLE 4.2 – Execution times, speedups and efficiency. Version `seq-optim`. Counter time loop (without initialization and IOs). Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	8763729.00	1.00	100.0	100.0
4	4	1	2549626.00	3.44	85.9	100.0
8	8	1	1497951.00	5.85	73.1	100.0
12	12	1	1140755.00	7.68	64.0	100.0

Les Tableaux 4.3, 4.4 et 4.5 montrent les performances pour les trois étapes de la boucle en temps (la diffusion, la méthode d'Arakawa et le terme source

relatif à l'advection respectivement). Lorsqu'on augmente le nombre de threads pour la partie advection (Tableaux 4.4 et 4.5), les temps d'exécution et donc les speedup passent bien à l'échelle. Par contre, la partie diffusion souffre d'une perte d'efficacité, jusqu'à 55.3 % sur les 12 threads du nœud de calcul, augmentant sa proportion de temps de calcul au sein de la boucle en temps. Nous donnons plus loin une courte analyse de cette dégradation.

TABLE 4.3 – Execution times, speedups and efficiency. Version `seq-optim`. Counter Diffusion (+ source fft et transpo). Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	5772093.00	1.00	100.0	65.9
4	4	1	1795794.00	3.21	80.4	70.4
8	8	1	1117292.00	5.17	64.6	74.6
12	12	1	870228.00	6.63	55.3	76.3

TABLE 4.4 – Execution times, speedups and efficiency. Version `seq-optim`. Counter Arakawa scheme. Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	1504395.00	1.00	100.0	17.2
4	4	1	379593.00	3.96	99.1	14.9
8	8	1	192517.00	7.81	97.7	12.9
12	12	1	145216.00	10.36	86.3	12.7

TABLE 4.5 – Execution times, speedups and efficiency. Version `seq-optim`. Counter Arakawa source term. Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	1487229.00	1.00	100.0	17.0
4	4	1	374227.00	3.97	99.4	14.7
8	8	1	188130.00	7.91	98.8	12.6
12	12	1	125302.00	11.87	98.9	11.0

Pour expliquer cette perte d'efficacité, on regarde les performances pour le calcul du terme source de la diffusion et des FFT 1D (dans les deux directions).

Les temps du Tableau 4.3 représentent la somme des temps des Tableaux 4.6, 4.7 et 4.8. Notons que le dernier tableau comprend le calcul de la diffusion en plus des FFT dans la direction z. Là encore, la gestion du terme source ne pose pas de problème au niveau efficacité de la parallélisation (Tableau 4.6). En revanche, les FFT dans la direction z couplées au calcul de diffusion affiche une efficacité de 64.7 % sur 12 threads, efficacité qui chute à 41.4 % dans le cas des FFT dans la direction y. Ceci indique que la perte d'efficacité globale du code lors de l'augmentation du nombre de ressources provient directement du calcul de ces transformées de Fourier 1D, résultat cohérent avec la conclusion du Chapitre 3. Ces résultats sont aussi dus à l'augmentation des besoins en bande passante mémoire, étant donné les manipulations nécessaires pour effectuer les FFT, qui sont les recopies de données dans les buffers, voir l'algorithme en Sous Section 4.4.2.

TABLE 4.6 – Execution times, speedups and efficiency. Version `seq-optim`. Counter Diffusion source term. Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	1635758.00	1.00	100.0	18.7
4	4	1	410725.00	3.98	99.6	16.1
8	8	1	206597.00	7.92	99.0	13.8
12	12	1	137570.00	11.89	99.1	12.1

TABLE 4.7 – Execution times, speedups and efficiency. Version `seq-optim`. Counter fft 1D Y. Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	2757837.00	1.00	100.0	31.5
4	4	1	986361.00	2.80	69.9	38.7
8	8	1	671625.00	4.11	51.3	44.8
12	12	1	555082.00	4.97	41.4	48.7

Ces résultats viennent motiver le passage à une parallélisation sur machine à mémoire distribuée pour l'équation de type advection diffusion.

4.5.2 Versions parallèles MPI et OpenMP

Nous évaluons ici les performances des versions parallèles hybrides MPI et OpenMP que nous avons présentées en Sous Section 4.4.3. Par celles-ci, nous

TABLE 4.8 – Execution times, speedups and efficiency. Version `seq-optim`. Counter Diffusion + fft 1D Z. Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	1378461.00	1.00	100.0	15.7
4	4	1	398672.00	3.46	86.4	15.6
8	8	1	239038.00	5.77	72.1	16.0
12	12	1	177529.00	7.76	64.7	15.6

cherchons à augmenter le nombre d'unités de calcul tout en augmentant la bande passante mémoire, qui fait défaut sur un seul nœud. Nous étudierons tout d'abord plusieurs configurations (nombre de threads, nombre de processus) sur un seul nœud, afin de déterminer la meilleure configuration possible en terme de déploiement. Ensuite, nous présenterons les résultats sur plusieurs nœuds de calcul.

Nous présentons tout d'abord les résultats pour un maillage $(N_x, N_y, N_z) = (256, 256, 128)$ comme en Sous Section 4.5.1, avant de présenter un cas impliquant un volume plus conséquent de données à traiter, de taille $(N_x, N_y, N_z) = (1024, 1024, 512)$.

Résultats pour le maillage $(N_x, N_y, N_z) = (256, 256, 128)$

TABLE 4.9 – Execution times, speedups and efficiency. Version `mpi-optim`. Counter time loop (without initialization and IOs). Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	8868458.00	1.00	100.0	100.0
12	3	4	1138887.00	7.79	64.9	100.0
12	6	2	1281582.00	6.92	57.7	100.0
12	12	1	1428172.00	6.21	51.7	100.0

Nous testons tout d'abord plusieurs couples (nombre de threads, nombre de processus), de manière à occuper toutes les unités de calcul. Nous nous intéressons aux couples (3,4), (6,2) et (12,1). Notre implémentation impose une limitation sur le nombre de processus qui doit être une puissance de 2 (devant diviser les dimensions distribuées de la grille). Les résultats sont présentés dans le Tableau 4.9, auxquels s'ajoute un résultat de référence sur une seule unité de calcul. Il apparaît dans ce tableau que la répartition la plus performante est la

configuration 3 threads et 4 processus (surligné en vert dans le Tableau 4.9). Ce cas affiche des temps d'exécution quasi identiques à ceux de la parallélisation OpenMP seulement pour l'algorithme seq-optim (voir le Tableau 4.2), ce qui est a priori étonnant car cette version MPI / OpenMP comporte plusieurs copies mémoire supplémentaires dues aux communications entre processus.

TABLE 4.10 – Execution times, speedups and efficiency. Version `mpi-optim`. Counter time loop (without initialization and IOs). Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	8868458.00	1.00	100.0	100.0
12	3	4	1131686.00	7.84	65.3	100.0
48	3	16	313165.00	28.32	59.0	100.0
96	3	32	191147.00	46.40	48.3	100.0
192	3	64	108992.00	81.37	42.4	100.0

Le Tableau 4.10 montre à présent le passage à l'échelle du code hybride MPI / OpenMP sur plusieurs nœuds, en utilisant 3 threads par processus et 4 processus par nœud [1], pour 1, 4, 8 et 16 nœuds de calcul, soit 12, 48, 96 et 192 unités de calcul. Pour 192 unités de calcul, on arrive à une efficacité de 42.4 %, soit un speedup de 81. Pour mieux comprendre la baisse d'efficacité, nous montrons les résultats pour chacune des parties du code.

Les Tableaux 4.11, 4.12 et 4.13 montrent les résultats pour la méthode d'Arakawa et les termes sources d'advection et de diffusion respectivement. Ils bénéficient d'excellentes performances, et ne viennent donc pas pénaliser le passage à l'échelle général lors de l'augmentation du nombre de ressources.

Pour les FFT dans la direction y, qui ne comportent toujours pas de communications, le Tableau 4.14 affiche de bonnes performances, montrant un speedup de 157 pour 192 coeurs. On note ici une accélération surlinéaire lors du passage de 12 à 48 unités de calcul (soit de 1 à 4 nœuds), qui s'explique par des effets cache positifs, le volume de données à traiter par nœud devenant suffisamment petit pour tenir dans le cache L3. C'est donc la partie comprenant les FFT dans la direction z et la diffusion, incluant aussi les communications qui passent moins bien à l'échelle.

En effet, le Tableau correspondant 4.15 affiche une baisse d'efficacité due à la présence des communications MPI, nécessaires pour réorganiser les données entre les différents nœuds pour le calcul des FFT dans la direction z. On

1. Une étude de performance sur plusieurs nœuds avec les autres déploiements NTH / NP sera effectuée d'ici la soutenance de la thèse, la meilleure configuration possible pouvant varier en fonction du nombre de nœuds utilisés.

TABLE 4.11 – Execution times, speedups and efficiency. Version `mpi-optim`. Counter Arakawa scheme. Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	1503370.00	1.00	100.0	17.0
12	3	4	126570.00	11.88	99.0	11.2
48	3	16	31710.00	47.41	98.8	10.1
96	3	32	15884.00	94.65	98.6	8.3
192	3	64	7890.00	190.54	99.2	7.2

TABLE 4.12 – Execution times, speedups and efficiency. Version `mpi-optim`. Counter Advection terme source. Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	1529573.00	1.00	100.0	17.2
12	3	4	127843.00	11.96	99.7	11.3
48	3	16	31164.00	49.08	102.3	10.0
96	3	32	15566.00	98.26	102.4	8.1
192	3	64	7779.00	196.63	102.4	7.1

TABLE 4.13 – Execution times, speedups and efficiency. Version `mpi-optim`. Counter Diffusion : terme source. Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	1639683.00	1.00	100.0	18.5
12	3	4	137149.00	11.96	99.6	12.1
48	3	16	34304.00	47.80	99.6	11.0
96	3	32	17254.00	95.03	99.0	9.0
192	3	64	8681.00	188.88	98.4	8.0

remarque notamment une grosse perte d'efficacité lors du passage de 1 à 12 unités de calcul, qui est due à la présence supplémentaire de communications inter-processus, faisant passer le nombre de processus de 1 à 4. Ensuite, le passage de 12 à 48 unités de calcul suppose l'utilisation de plusieurs nœuds, ce qui met en jeu des communications inter-nodales qui prennent plus de temps (passage par le réseau infiniband) ; ceci peut expliquer la baisse supplémentaire d'efficacité, passant de 39.7 % à 21 %. Au delà de 4 nœuds, le rapport entre

TABLE 4.14 – Execution times, speedups and efficiency. Version `mpi-optim`. Counter fft 1D Y. Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	2738975.00	1.00	100.0	30.9
12	3	4	433708.00	6.32	52.6	38.3
48	3	16	70870.00	38.65	80.5	22.6
96	3	32	38844.00	70.51	73.5	20.3
192	3	64	17449.00	156.97	81.8	16.0

les temps de calcul pour cette partie du code reste intéressant, et permet de plus de lancer des cas plus volumineux en terme de taille de maillage. Ceci fait l'objet du Paragraphe suivant.

TABLE 4.15 – Execution times, speedups and efficiency. Version `mpi-optim`. Counter fft 1D Z et diffusion. Case size : 256x256x128.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	1456793.00	1.00	100.0	16.4
12	3	4	305604.00	4.77	39.7	27.0
48	3	16	144676.00	10.07	21.0	46.2
96	3	32	101624.00	14.34	14.9	53.2
192	3	64	66628.00	21.86	11.4	61.1

Le même étude de passage à l'échelle sur plusieurs nœuds a été menée sur la machine Helios. Cette machine comporte 2 processeurs de 8 cœurs par nœud (contre 2 fois 6 cœurs pour la machine de Marseille). Le meilleur déploiement sur un nœud consiste à utiliser un processus et 16 threads dans ce cas. Le Tableau 4.16 montre les résultats pour l'ensemble des itérations temporelles pour 1 à 256 unités de calcul avec le déploiement précédent. Dans leur ensemble, les résultats obtenus sont proches de ceux obtenus sur le cluster d'Aix-Marseille Université (voir le Tableau 4.10), affichant un speedup de 92 pour une efficacité de 36% sur 256 unités de calcul (soit 16 nœuds dans ce cas).

Résultats pour un maillage $(N_x, N_y, N_z) = (1024, 1024, 512)$

Nous regardons ici les performances pour un maillage plus conséquent, proche des configurations que l'on souhaite atteindre avec le code Emedge3D.

TABLE 4.16 – Execution times, speedups and efficiency. Version `mpi-optim`. Counter time loop (without initialization and IOs). Case size : 256x256x128. Helios supercomputer.

NCU	NTH	NP	Time (μs)	Speedup	Eff %	Total %
1	1	1	9560797.00	1.00	100.0	100.0
16	16	1	940978.00	10.16	63.5	100.0
32	16	2	530755.00	18.01	56.3	100.0
64	16	4	299082.00	31.97	49.9	100.0
128	16	8	164423.00	58.15	45.4	100.0
256	16	16	103608.00	92.28	36.0	100.0

Cette taille de maillage suppose la manipulation de champs scalaires 3D occupant plus de 4 Go en mémoire. Avec l'implémentation n'utilisant pas MPI, la capacité mémoire d'un nœud n'était pas suffisante pour réaliser une exécution sur le mésocentre d'Aix-Marseille. Le programme MPI/OpenMP peut lui par contre prendre en charge cette taille de maillage sans problème sur 4, 8 et 16 nœuds de calcul.

Le Tableau 4.17 montre les performances de la boucle en temps sur 4, 8, 16 et 32 nœuds pour cette taille de maillage. A nouveau, 10 itérations temporelles sont effectuées. Dans ce tableau nous avons pris pour référence les résultats sur 4 nœuds de calcul. On constate une belle évolution des performances en augmentant le nombre de nœuds utilisés, efficace à 78.5% sur 32 nœuds par rapport aux 4 noeuds de référence, soit un speedup de 6.3. Cette bonne performance s'explique par la combinaison de deux éléments : les temps de communications ont un bon comportement et diminuent lorsque l'on augmente le nombre de nœuds, d'autre part la proportion des communications par rapport aux temps de calcul reste entre 20% et 50% du temps total.

TABLE 4.17 – Execution times, speedups and efficiency. Version `mpi-optim`. Counter time loop (without initialization and IOs). Case size : 1024x1024x512.

NCU	NTH	NP	Time μs	Speedup	Eff %	Total %
48	3	16	27639508.00	1.00	100	100.0
96	3	32	15425293.00	1.79	89.5	100.0
192	3	64	8292095.00	3.33	83.3	100.0
384	3	128	4389362.00	6.30	78.5	100.0

En effet, le Tableau 4.18, qui contient les mesures pour les communications ainsi que les FFT dans la direction z avec le calcul de diffusion montre lui aussi une belle évolution de l'efficacité de la parallélisation MPI. Contrairement à la taille de maillage précédente (voir le Tableau 4.15), nous sommes ici dans une situation dans laquelle le temp de référence inclue déjà des temps de communications (alors que précédemment le temps de référence sur 1 cœur n'avait pas ce surcoût). Ceci explique en partie la bonne efficacité observée dans ce tableau.

TABLE 4.18 – Execution times, speedups and efficiency. Version `mpi-optim`. Counter fft 1D Z et diffusion. Case size : 1024x1024x512.

NCU	NTH	NP	Time μs	Speedup	Eff %	Total %
48	3	16	11560291.00	1.00	100	41.8
96	3	32	6873224.00	1.68	84	44.6
192	3	64	3700161.00	3.12	78	44.6
384	3	128	2099775.00	5.51	68.9	47.8

La même taille de maillage a été traitée sur la machine Helios. Le Tableau 4.19 montre les résultats pour la boucle en temps, et le Tableau 4.20 montre les résultats pour les communications, les FFT dans la direction z et la diffusion. Bien que l'efficacité souffre d'une baisse de 10% pour les deux tableaux présentés, les résultats restent proche par rapport à la machine précédente. On note que le nombre de threads change en fonction du nombre de nœuds dans les deux tableaux. En effet, le meilleur déploiement possible NTH / NCU par nœud varie en fonction du nombre de nœuds utilisés sur la machine Helios.

TABLE 4.19 – Execution times, speedups and efficiency. Version `mpi-optim`. Counter time loop (without initialization and IOs). Case size : 1024x1024x512. Helios supercomputer.

NCU	NTH	NP	Time μs	Speedup	Eff %	Total %
64	8	8	23543254.00	1.00	100	100.0
128	8	16	13246874.00	1.78	89	100.0
256	8	32	7374608.00	3.19	79.8	100.0
512	16	32	4221915.00	5.58	69.8	100.0

TABLE 4.20 – Execution times, speedups and efficiency. Version `mpi-optim`. Counter Diffusion + fft 1D Z. Case size : 1024x1024x512. Helios supercomputer.

NCU	NTH	NP	Time μs	Speedup	Eff %	Total %
64	8	8	10670114.00	1.00	100	45.3
128	8	16	6598600.00	1.62	81	49.8
256	8	32	4096757.00	2.60	65	55.6
512	16	32	2248265.00	4.75	59.4	53.3

4.6 Conclusion

Dans ce chapitre, nous avons mis en place une stratégie de parallélisation hybride MPI / OpenMP pour une équation de type advection diffusion proche de l'équation de pression du code Emedge3D. Cette étude a permis de résoudre la limitation par la bande passante mémoire qui bridait les performances dans le Chapitre 3, plus particulièrement pour la partie non linéaire. En effet, le passage sur architecture à mémoire distribuée par l'utilisation de la bibliothèque MPI couplé à la mise en place d'un parallélisme à grain plus fin pour la partie OpenMP (en particulier pour les transformées de Fourier) ont permis l'augmentation de la bande passante mémoire et donc des performances, jusqu'à un speedup de 82 sur 192 unités de calcul pour un maillage $(256, 256, 128)$ sur la machine d'Aix-Marseille Université. D'autre part, le surcoût en temps de calcul entraîné par les communications supplémentaires nécessaires à l'algorithme reste une fraction du temps total de calcul et affiche une évolution en efficacité bonne lors de l'ajout de nœuds de calcul. Aussi, cette parallélisation permet d'atteindre des tailles de maillage plus conséquentes sur des machines à mémoire distribuées, ce qui est souhaité par les physiciens utilisateurs du code Emedge3D. Les performances pour le maillage $(1024, 1024, 512)$ ont permis d'atteindre une efficacité de 78.5 % sur 384 unités de calcul par rapport aux 48 de référence, ce qui montre un bon passage à l'échelle des communications. Notons de plus que le cas n'a pas pu être lancé sur un seul noeud, le volume de données à traiter étant trop élevé pour tenir en mémoire vive. En terme de temps d'exécution, sur 384 unités de calcul, on passe de 2.8 à 0.4 secondes par pas de temps pour ce maillage.

En perspective à cette étude, la possibilité de coupler cette parallélisation à un schéma de type semi-implicite permettrait de diminuer le nombre d'itérations temporelles requises pour obtenir une solution à un temps final donné. Aux vues des conclusions du Chapitre 2, nous pouvons supposer un facteur au

moins 10 supplémentaire dans le cas des maillages que nous avons considéré. Ensuite, un recouvrement des temps de communication par du calcul permettrait de réduire la proportion de temps passé dans les communications. Enfin, une intégration de la stratégie de parallélisation présentée dans ce chapitre dans Emedge3D permettrait d'atteindre de meilleures performances et d'exécuter le programme sur des grands maillages.

Chapitre 5

Conclusion

Au travers d'un code de simulation dédié à la physique des plasmas, nous avons mis en évidence comment de nouveaux schémas numériques autorisent une augmentation de la taille des pas de temps d'un facteur 10 à 100 et de réduire ainsi les temps de calculs. D'autre part, en mettant en œuvre des méthodes de calcul haute-performance nous avons réduit drastiquement les temps d'exécution. Ceci nous a permis de constater des scalabilités de plus de 81 sur 192 cœurs avec une parallélisation hybride MPI/OpenMP, et d'accéder à des tailles de maillage jusqu'alors inaccessible sur un seul nœud de calcul.

Plus précisément, l'étude des méthodes numériques pour la diffusion au Chapitre 2 a permis de mettre en avant l'efficacité des méthodes de type semi-implicites en temps avec des méthodes d'ordre élevé en espace. Plusieurs méthodes ont été comparées et évaluées en fonction des critères imposés par les problèmes numériques que pose la diffusion dans Emedge3D. Ces critères sont la pollution de l'erreur dans la direction perpendiculaire aux lignes de champs provenant de la diffusion parallèle (aussi appelée diffusion perpendiculaire numérique), ainsi que le caractère anisotrope et non linéaire de la diffusion. Il apparaît que la méthode semi-implicite de pénalisation couplée à une intégration en temps de type Additive Runge-Kutta se révèle particulièrement performante pour les critères qui nous intéressent, surtout lorsqu'elle est associée à des schémas spatiaux d'ordre 4. Ceci permet de garantir une précision spatio-temporelle intéressante tout en utilisant un pas de temps raisonnable. Plusieurs extensions à ce travail sont possibles. D'abord, au niveau du choix du paramètre de pénalisation λ : afin de suivre au mieux l'anisotropie de l'opérateur étudié, il semble intéressant de choisir λ comme une fonction dépendant de la variable spatiale. D'autres choix plus élaborés peuvent être faits pour approcher au mieux la matrice de diffusion ; cependant, un bon compromis précision/efficacité doit être trouvé puisque ce nouvel opérateur $\nabla \cdot (\lambda \nabla T^{n+1})$ devra être inversé. Un second point important serait d'étendre les schémas au cas non

périodique afin d'aborder des cas test encore plus réalistes d'un point de vue de la physique. Afin de ne pas briser l'ordre en espace, des techniques doivent être utilisées comme celles proposées dans [57]. Enfin, l'aspect calcul haute performance serait intéressant à approfondir en étudiant plus finement les coûts de calcul de la méthode de pénalisation ARK, afin de mesurer le bénéfice effectif sur le temps d'exécution. Des extensions moins immédiates peuvent aussi être envisagées. Plusieurs schémas ont récemment été publiés [43, 47, 9, 56] possédant la propriété Asymptotic Preserving. Ces schémas garantissent une précision uniforme indépendamment du rapport d'anisotropie $\chi_\perp/\chi_\parallel$. Cependant, ces schémas ne sont pas d'ordre élevé en espace ou en temps et il parait intéressant de combiner les techniques développées dans ce chapitre aux reformulations du problème initial proposées dans les travaux [43, 47, 9, 56]. Enfin, l'analyse numérique des schémas de type ARK serait intéressante à mener ; en effet, les propriétés de L-stabilité ou A-stabilité sont des notions importantes lorsque le rapport d'anisotropie tend vers zéro. Une étude comparative approfondie permettrait de confronter ces propriétés sur des cas test pertinents.

Dans les Chapitres 3 et 4 sont proposées des solutions de parallélisation pour le code Emedge3D. Dans un premier temps, le Chapitre 3 expose une stratégie d'optimisation et de parallélisation pour architecture à mémoire partagée. Nous avons rencontré une limitation imposée par la bande passante mémoire, que nous avons réussi à surmonter dans la partie linéaire du code, en mettant en place une technique d'optimisation appelée tiling notamment. En ce qui concerne la partie non linéaire, les performances restaient limitées par la bande passante mémoire, majoritairement à cause du traitement des transformées de Fourier et des transpositions. C'est pourquoi le Chapitre 4 étudie une parallélisation sur architecture à mémoire distribuée, afin d'augmenter la bande passante mémoire en ajoutant des nœuds de calcul. Nous avons mené cette étude sur une équation d'advection diffusion proche de l'équation de pression d'Emedge3D, étant donné les lourds changements que supposaient cette parallélisation dans le code original. Nous avons mis en place un parallélisme distribué via MPI impliquant un changement d'axe de parallélisation et donc de distribution des données pour le calcul des transformées de Fourier. Nous avons associé cette parallélisation distribuée à une parallélisation OpenMP à grain plus fin que dans l'étude du Chapitre 3 afin de pouvoir bénéficier de plus d'effets cache, notamment en effectuant le calcul des transformées de Fourier en deux étapes 1D plutôt qu'en une seule étape 2D. Il ressort de cette étude de bonnes performances en terme de scalabilité des temps de calcul lors de l'ajout de nœuds MPI due à l'augmentation de la bande passante mémoire disponible. De plus, cette étude nous a permis d'atteindre des maillages de plus grande taille, non atteignable sur un unique nœud de calcul. Ceci nous a permis d'atteindre un speedup de 6.3 (speedup idéal de 8) pour une efficacité

de 78.5% sur 384 unités de calcul par rapport aux 48 cœurs de référence pour un maillage de taille $(1024, 1024, 512)$. Comme perspective, une étude couplée parallélisation hybride MPI / OpenMP et schéma temporel semi-implicite est intéressante, surtout en utilisant la méthode de pénalisation / ARK associée à un ordre élevé en espace.

En perspectives pour le code Emedge3D, nous proposons l'intégration de méthodes de types semi-implicites en temps, couplée à l'utilisation d'un ordre élevé (ordre 4) pour la résolution numérique des opérateurs en espace. La stratégie de parallélisation présentée au Chapitre 4 devrait elle aussi être intégrée. Il en résulterait l'association des gains du point de vue numérique et du point de vue algorithmique, alliant l'utilisation d'un pas de temps plus élevé que dans le cas de méthodes explicites et une parallélisation hybride MPI / OpenMP.

A plus long terme, un autre type schéma temporel Asymptotic Preserving pourrait être mis en place pour Emedge3D, permettant de dériver le modèle électrostatique d'Emedge3D à partir du modèle électromagnétique par un passage à la limite. En effet, le modèle pour Emedge3D peut d'écrire, avec $J = \nabla_\perp^2 \psi$, $W = \nabla_\perp^2 \phi$, $\nabla_\parallel = \partial_z - \{\varepsilon\psi + \psi_0, \cdot\}$:

$$\begin{aligned}
\partial_t W + \{\phi, W\} &= -C_1 \nabla_\parallel (J + J_0) - Gp + \nabla_\perp \cdot (\nu(x) \nabla_\perp W) && (5.0.1)\\
\partial_t p + \{\phi, p\} &= C_2 \nabla_\parallel (J + J_0) + G(\delta_c \phi - \Gamma p) + \nabla_\perp \cdot (\xi_\perp(x, y) \nabla_\perp p) \\
&\quad + \xi_\parallel(z) \nabla_\parallel^2 p + S(x) && (5.0.2)\\
\partial_t \psi &= -\frac{1}{\varepsilon}(\nabla_\parallel \phi - J) + \Gamma \nabla_\parallel p && (5.0.3)
\end{aligned}$$

Lorsque $\varepsilon \to 0$, formellement, l'Equation (5.0.3) fournit, la relation : $J = \nabla_{\parallel 0} \phi$ avec $\nabla_{\parallel 0} = \partial_z + \{\psi_0, \cdot\}$, car $\nabla_\parallel \to \nabla_{\parallel 0}$ et les termes $\partial_t \psi$ et $\Gamma \nabla_\parallel p$ deviennent négligeable . En remplaçant J par $\nabla_{\parallel 0} \phi$ et $\nabla_\parallel$ par $\nabla_{\parallel 0}$ dans l'Equation (5.0.1), on obtient alors :

$$\partial_t W + \{\phi, W\} = -C_1 \nabla_{\parallel 0}^2 \phi - C_1 \nabla_{\parallel 0} J_0 - Gp + \nabla_\perp \cdot (\nu(x) \nabla_\perp W). \quad (5.0.4)$$

qui, couplé à (5.0.2) correspond à la limite électrostatique du modèle (5.0.1)-(5.0.3)-(5.0.2). Une stratégie du type micro-macro [3, 35, 11] peut être mise en place. La fonction ψ y est décomposée sous la forme $\psi = [\nabla_\perp^2]^{-1} \nabla_\parallel \phi + g$. Un schéma semi-implicite en temps peut être obtenu qui satisfait les propriétés suivantes : *(i)* le schéma est stable par rapport à ε ; *(ii)* le schéma dégénère en un schéma consistant avec le modèle électrostatique.

A long terme également, la méthode des volumes finis (voir [57]) utilisée dans le Chapitre 2 est assez générale et peut s'appliquer sur un grand nombre d'équations aux dérivées partielles de type hyperbolique et parabolique (advection/diffusion), possiblement non linéaire. Ceci recouvre beaucoup d'applications comme les équations d'Euler ou Navier-Stokes, les équations de la MHD

ou de Maxwell, voire les équations cinétiques. La mise en place de l'ordre 4 permet de capturer dans ces différents contextes les fines structures créées par la solution. L'extension à l'ordre plus élevé et la prise en compte de conditions aux bords non périodique (Dirichlet ou Neumann) sont aussi possibles. Ainsi, ce type de schéma peut s'utiliser sur une grande variété d'applications physiques.

Enfin, concernant la méthode de pénalisation et son extension à l'ordre élevé grâce aux schémas ARK, de nombreuses applications sont aussi envisageables. En effet, ce type d'approche est particulièrement bien adapté pour des problèmes où différentes échelles en temps sont présentes (coexistence d'un terme raide et d'un terme non raide dans l'équation). Le pas de temps n'est pas contraint par l'échelle temporelle la plus petite et l'utilisateur choisit alors lui-même la valeur du pas de temps et donc la précision voulue. On peut citer en plus des problèmes d'advection-diffusion traités dans cette thèse, les équations cinétiques collisionnelles en suivant l'esprit de [18]. Un autre intérêt de cette méthode de pénalisation réside dans le traitement implicite de terme non linéaire intervenant dans l'équation étudiée ; en effet, ceci nécessite habituellement des méthodes de point fixe ou de Newton assez coûteuse. L'approche de pénalisation permet d'inverser des opérateurs plus simples et possiblement constants en temps, permettant un pré-stockage efficace. Le couplage avec les schémas ARK permet d'augmenter la précision temporelle à pas de temps fixé, dont on a vu le grand intérêt.

Annexes

Annexe A

Complément au Chapitre 2

A.1 Relations pour la discrétisation spatiale

Dans cette partie, on rappelle certaines relations utiles pour la dérivation des schémas numériques. On considère les fonctions de la variable réelle $f := f(x)$ et $h := h(x)$ et la fonction à deux variables $g := g(x, y)$. Un maillage uniforme générique est utilisé : $z_{k+1/2} = z_{k-1/2} + \Delta z$, pour $k \in \mathbb{Z}$. On note f_k la quantité moyennée sur une maille

$$f_k = \frac{1}{\Delta z} \int_{z_{k-1/2}}^{z_{k+1/2}} f(z) dz.$$

On rappelle les notations introduites plus haut $C_i^x = [x_{i-1/2}, x_{i+1/2}]$ et $C_j^y = [y_{j-1/2}, y_{j+1/2}]$, avec Δx et Δy le pas du maillage dans les directions x et y de telle sorte que les moyennes sur les cellules s'écrivent

$$g_{i,j} = \frac{1}{\Delta x \Delta y} \int_{C_i^x} \int_{C_j^y} g(x, y) dx dy.$$

La première propriété permet d'exprimer l'intégrale du produit de deux fonctions en le produit d'intégrales.

Proposition A.1.1 *Considérons $I = \int_{z_{k-1/2}}^{z_{k+1/2}} f(z)h(z)dz$. Une approximation d'ordre deux de I donne*

$$\frac{1}{\Delta z} \int_{z_{k-1/2}}^{z_{k+1/2}} f(z)h(z)dz = \frac{1}{\Delta z} \int_{z_{k-1/2}}^{z_{k+1/2}} f(z)dz \frac{1}{\Delta z} \int_{z_{k-1/2}}^{z_{k+1/2}} h(z)dz + \mathcal{O}(\Delta z^2)$$

alors qu'une approximation d'ordre quatre de I donne

$$\frac{1}{\Delta z}\int_{z_{k-1/2}}^{z_{k+1/2}} f(z)h(z)dz = \frac{1}{\Delta z}\int_{z_{k-1/2}}^{z_{k+1/2}} f(z)dz \frac{1}{\Delta z}\int_{z_{k-1/2}}^{z_{k+1/2}} h(z)dz$$
$$+\frac{1}{48\Delta z^2}\left(\int_{z_{k+1/2}}^{z_{k+3/2}} f(z)dz - \int_{z_{k-3/2}}^{z_{k-1/2}} f(z)dz\right)\left(\int_{z_{k+1/2}}^{z_{k+3/2}} h(z)dz - \int_{z_{k-3/2}}^{z_{k-1/2}} h(z)dz\right)$$
$$+\mathcal{O}(\Delta z^4).$$

La proposition suivante permet d'exprimer la valeur moyenne sur une face en fonction des valeurs moyennes sur une cellule, jusqu'à l'ordre quatre.

Proposition A.1.2 *Une approximation d'ordre deux de la moyenne sur une face f en fonction de la moyenne sur une cellule f_k est*

$$f(z_{k+1/2}) = \frac{1}{2}\left[f_{k+1} + f_k\right] + \mathcal{O}(\Delta z^2).$$

alors qu'une approximation d'ordre quatre donne

$$f(z_{k+1/2}) = \frac{7}{12}\left[f_{k+1} + f_k\right] - \frac{1}{12}\left[f_{k+2} + f_{k-1}\right] + \mathcal{O}(\Delta z^4).$$

Ainsi, on déduit, pour une fonction $g = g(x, y)$,

$$\begin{aligned}\int_{C_j^y} g(x_{i+1/2}, y)dy &= \frac{1}{2}(g_{i+1,j} + g_{i,j}) + \mathcal{O}(\Delta x^2), \quad \textit{pour l'ordre deux} \\ &= \frac{7}{12}\left(g_{i+1,j} + g_{i,j}\right) - \frac{1}{12}\left(g_{i+2,j} + g_{i-1,j}\right) + \mathcal{O}(\Delta x^4), \\ &\quad \textit{pour l'ordre quatre}\end{aligned}$$

et

$$\begin{aligned}\int_{C_i^x} g(x, y_{j+1/2})dx &= \frac{1}{2}(g_{i,j+1} + g_{i,j}) + \mathcal{O}(\Delta y^2), \quad \textit{pour l'ordre deux} \\ &= \frac{7}{12}\left(g_{i,j+1} + g_{i,j}\right) - \frac{1}{12}\left(g_{i,j+2} + g_{i,j-1}\right) + \mathcal{O}(\Delta y^4), \\ &\quad \textit{pour l'ordre quatre.}\end{aligned}$$

La proposition suivante permet d'exprimer la moyenne sur une face de la dérivée en fonction de la moyenne sur une cellule, jusqu'à l'ordre quatre.

Proposition A.1.3 *Pour une fonction f, l'approximation d'ordre deux s'écrit*

$$f'(z_{k+1/2}) = \frac{1}{\Delta z}(f_{k+1} - f_k) + \mathcal{O}(\Delta z^2).$$

et l'approximation d'ordre quatre est

$$f'(z_{k+1/2}) = \frac{5}{4\Delta z}\left[f_{k+1} - f_k\right] - \frac{1}{12\Delta z}\left[f_{k+2} - f_{k-1}\right] + \mathcal{O}(\Delta z^4).$$

Ainsi, on déduit, pour une fonction $g = g(x, y)$,

$$\begin{aligned}\int_{C_j^y} \partial_x g(x_{i+1/2}, y)dy &= \frac{1}{\Delta x}(g_{i+1,j} - g_{i,j}) + \mathcal{O}(\Delta x^2), \quad \textit{pour l'ordre deux} \\ &= \frac{5}{4\Delta x}(g_{i+1,j} - g_{i,j}) - \frac{1}{12\Delta x}(g_{i+2,j} - g_{i-1,j}) + \mathcal{O}(\Delta x^4), \\ &\quad \textit{pour l'ordre quatre}\end{aligned}$$

et

$$\begin{aligned}\int_{C_i^x} \partial_y g(x, y_{j+1/2})dx &= \frac{1}{\Delta y}(g_{i,j+1} - g_{i,j}) + \mathcal{O}(\Delta y^2), \quad \textit{pour l'ordre deux} \\ &= \frac{5}{4\Delta y}(g_{i,j+1} - g_{i,j}) - \frac{1}{12\Delta y}(g_{i,j+2} - g_{i,j-1}) + \mathcal{O}(\Delta y^4), \\ &\quad \textit{pour l'ordre quatre.}\end{aligned}$$

Enfin, des relations utiles pour l'approximation des dérivées croisées sont présentées.

Proposition A.1.4 *Pour une fonction $g = g(x, y)$, on a*

$$\int_{C_i^x} \partial_x g(x, y_{j+1/2})dx = g(x_{i+1/2}, y_{j+1/2}) - g(x_{i-1/2}, y_{j+1/2}) = (D_y D_x g)_{i,j},$$

et

$$\int_{C_j^y} \partial_y g(x_{i+1/2}, y)dy = g(x_{i+1/2}, y_{j+1/2}) - g(x_{i+1/2}, y_{j-1/2}) = (D_x D_y g)_{i,j}.$$

On utilise les notations

$$\begin{aligned}(D_x g)_{i,j} &= \frac{1}{\Delta x}(g_{i+1,j} - g_{i,j}) + \mathcal{O}(\Delta x^2), \quad \textit{pour l'ordre deux} \\ &= \frac{7}{12\Delta x}(g_{i+1,j} + g_{i,j}) - \frac{1}{12\Delta x}(g_{i+2,j} + g_{i-1,j}) + \mathcal{O}(\Delta x^4), \\ &\quad \textit{pour l'ordre quatre.}\end{aligned}$$

et

$$\begin{aligned}(D_y g)_{i,j} &= \frac{1}{\Delta y}(g_{i,j+1} - g_{i,j}) + \mathcal{O}(\Delta y^2), \quad \textit{pour l'ordre deux} \\ &= \frac{7}{12\Delta y}(g_{i,j+1} + g_{i,j}) - \frac{1}{12\Delta y}(g_{i,j+2} + g_{i,j-1}) + \mathcal{O}(\Delta y^4), \\ &\quad \textit{pour l'ordre quatre,}\end{aligned}$$

ainsi

$$\begin{aligned}(D_x D_y g)_{i,j} &= \frac{1}{4\Delta x \Delta y}(g_{i+1,j+1} - g_{i+1,j} + g_{i,j+1} - g_{i,j}) + \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta y^2), \\ &\quad \textit{pour l'ordre deux} \end{aligned} \tag{A.1.1}$$

$$\begin{aligned}(D_x D_y g)_{i,j} &= \frac{7}{12\Delta x \Delta y}\left[\frac{7}{12}(g_{i+1,j+1} + g_{i+1,j}) - \frac{1}{12}(g_{i+1,j+2} + g_{i+1,j-1})\right] \\ &+ \frac{7}{12\Delta x \Delta y}\left[\frac{7}{12}(g_{i,j+1} + g_{i,j}) - \frac{1}{12}(g_{i,j+2} + g_{i,j-1})\right] \\ &- \frac{1}{12\Delta x \Delta y}\left[\frac{7}{12}(g_{i+2,j+1} + g_{i+2,j}) - \frac{1}{12}(g_{i+2,j+2} + g_{i+2,j-1})\right] \\ &- \frac{1}{12\Delta x \Delta y}\left[\frac{7}{12}(g_{i-1,j+1} + g_{i-1,j}) - \frac{1}{12}(g_{i-1,j+2} + g_{i-1,j-1})\right] \\ &+ \mathcal{O}(\Delta x^4) + \mathcal{O}(\Delta y^4), \quad \textit{pour l'ordre quatre.}\end{aligned} \tag{A.1.2}$$

A.2 Analyse de stabilité linéaire : ordre quatre

Dans cette partie, les détails des calculs de l'analyse de stabilité linéaire des deux intégrateurs en temps (SH et ARK ordre un uniquement) sont présentés.

A.2.1 Schéma semi-implicite (SH)

On écrit le schéma numérique d'ordre quatre en espace couplé au schéma semi-implicite. La première étape du schéma s'écrit alors (avec les notations introduites précédemment)

$$T^{\star}_{i,j} = T^{n}_{i,j} + \frac{\Delta t}{12\Delta x^2} b_{xx}(D_{xx}T^{\star})_{i,j} + \Delta t b_{xy}(D_x D_y T^{n})_{i,j},$$

et la seconde étape s'écrit

$$T^{n+1}_{i,j} = T^{\star}_{i,j} + \frac{\Delta t}{12\Delta y^2} b_{yy}(D_{yy}T^{n+1})_{i,j} + \Delta t b_{xy}(D_x D_y T^{\star})_{i,j}.$$

avec

$$(D_{xx}T)_{i,j} = [-T_{i+2,j} + 16T_{i+1,j} - 30T_{i,j} + 16T_{i-1,j} - T_{i-2,j}], \quad \text{(A.2.3)}$$
$$(D_{yy}T)_{i,j} = [-T_{i,j+2} + 16T_{i,j+1} - 30T_{i,j} + 16T_{i,j-1} - T_{i,j-2}], \quad \text{(A.2.4)}$$

et $(D_X D_y T)_{i,j}$ donnée par (A.1.2).

La stabilité "à la Von Neumann" suggère d'injecter une solution type onde plane $T(t,x,y) = r(t)\exp^{-\mathrm{i}(k_x x + k_y y)}$ dans le schéma. Le facteur d'amplification r_1 pour la première étape (avec ncfl$= 4\Delta t/\Delta x^2$) se calcule comme suit :

$$\begin{aligned} r_1(t) &= \frac{1 - \frac{\text{ncfl}}{144} b_{xy}[\sin(2k_y\Delta y) - 8\sin(k_y\Delta y)][\sin(2k_x\Delta x) - 8\sin(k_x\Delta x)]}{1 + \frac{\text{ncfl}}{48} b_{xx}[2\cos(2k_x\Delta x) - 32\cos(k_x\Delta x) + 30]} \\ &= \frac{1 - \frac{\text{ncfl}}{144} b_{xy}[\sin(2k_y\Delta y) - 8\sin(k_y\Delta y)][\sin(2k_x\Delta x) - 8\sin(k_x\Delta x)]}{1 + \frac{\text{ncfl}}{48} b_{xx}[4(\cos(k_x\Delta x) - 7)(\cos(k_x\Delta x) - 1)]}, \end{aligned}$$

alors que le facteur d'amplification r_2 pour la deuxième étape s'écrit

$$r_2(t) = \frac{1 - \frac{\text{ncfl}}{144} b_{xy}[\sin(2k_y\Delta y) - 8\sin(k_y\Delta y)][\sin(2k_x\Delta x) - 8\sin(k_x\Delta x)]}{1 + \frac{\text{ncfl}}{48} b_{yy}[2\cos(2k_y\Delta y) - 32\cos(k_y\Delta y) + 30]}.$$

Le facteur d'amplification total $r = r_1 r_2$ est tracé sur la Figure 2.3 pour différentes valeurs de ncfl.

A.2.2 Schéma Additive Runge Kutta

On écrit le schéma d'ordre quatre couplé à la technique de pénalisation. Le schéma numérique s'écrit alors

$$\begin{aligned} T_{i,j}^{n+1} &= T_{i,j}^n + \frac{\Delta t}{12\Delta x^2}(b_{xx} - \lambda)(D_{xx}T^n)_{i,j} + 2\Delta t b_{xy}(D_x D_y T^n)_{i,j} \\ &\quad + \frac{\Delta t}{12\Delta y^2}(b_{yy} - \lambda)(D_{xx}T^n)_{i,j} + \frac{\Delta t\lambda}{\Delta x^2}(D_{xx}T^{n+1})_{i,j} + \frac{\Delta t\lambda}{\Delta y^2}(D_{yy}T^{n+1})_{i,j}, \end{aligned}$$

où $(D_x xT)_{i,j}, (D_x xT)_{i,j}$ sont donnés par (A.2.3) et (A.2.4), et $(D_x D_y T^n)_{i,j}$ est donné par (A.1.2).

En suivant l'analyse de Von Neumann, on injecte la solution de type onde plane $T(t,x,y) = r(t)\exp^{-\mathrm{i}(k_x x + k_y y)}$ dans le schéma. On obtient alors le facteur

d'amplification $r = r_1/r_2$ (avec ncfl$= 4\Delta t/\Delta x^2$) avec

$$\begin{aligned} r_1 = 1 + \frac{\text{ncfl }\lambda}{2}(&-[\cos(2k_x\Delta x) + \cos(2k_y\Delta y)] \\ &+16[\cos(k_x\Delta x) + \cos(k_y\Delta y)] - 30) \\ -\frac{\text{ncfl}}{2}(&b_{xx}[-\cos(2k_x\Delta x) + 16\cos(k_x\Delta x) - 15] \\ &+b_{yy}[\cos(2k_y\Delta y) + 16\cos(k_y\Delta y) - 15]) \\ -\frac{2\text{ncfl }b_{xy}}{9}(&\sin(2k_y\Delta y) - 8\sin(k_y\Delta y)][\sin(2k_x\Delta x) - 8\sin(k_x\Delta x)]) \end{aligned}$$

et

$$r_2 = 1+\frac{\text{ncfl }\lambda}{2}\Big(-[\cos(2k_x\Delta x)+\cos(2k_y\Delta y)]+16[\cos(k_x\Delta x)+\cos(k_y\Delta y)]-30\Big).$$

Le facteur d'amplification $r = r_1/r_2$ est tracé sur la Figure 2.5 pour différentes valeurs de ncfl.

Annexe B

Complément au Chapitre 3

Cette annexe donne deux exemples de code C, permettant le calcul des opérateurs de diffusion perpendiculaire et des crochets de Poisson (méthode d'Arakawa), issus du code Emedge3D. Ces deux morceaux de code sont de bons éléments représentatifs du type d'opération spatiales à effectuer sur les structures 3D représentant les inconnues du modèle physique à résoudre.

B.1 Code C pour la diffusion perpendiculaire

La fonction qui suit résout l'opérateur de diffusion perpendiculaire $\nabla_\perp^2 = \partial_x^2 + \partial_y^2$ dans l'espace semi-spectral. Les types des tableaux sont définis comme suit :

```
typedef double *const *const *const *const pcnstField3Dc;
typedef double const *const *const *const *const pcnstField3Dc;

/**
*\fn RBM_codes addDiffusion(pcnstField3Dc f1, pcnstField3Dc_cnst f2,
  sLaplacianOp const *const sLaplacian, sFieldSize const *const sFs)
*\brief applique a un champ donne l'operateur de diffusion
  perpendiculaire f1=f1+D(f2)
*\param f1 : champ ou le resultat sera stocke
*\param f2 : champ d'entree
*\param sLaplacian : structure contenant les informations sur la
  diffusion
*\param sFs : taille des champs
*\return : RBM_OK
*/
RBM_codes addDiffusion ( pcnstField3Dc f1, pcnstField3Dc_cnst f2,
sLaplacianOp const *const sLaplacian, sFieldSize const *const sFs )
```

```
{
  RBM_codes rbm_er=RBM_OK;
  size_t n=0;
  size_t const iSizeX = sFs->iSize1-1;

  for ( n=0;n<sFs->iSize3;n++ )
  {
    size_t m=0;

    for ( m=0;m<sFs->iSize2;m++ )
    {
      size_t x=0;

      pcnstField1D_cnst f2r_m = f2[0][n][m];
      pcnstField1D_cnst f2i_m = f2[1][n][m];

      pcnstField1D      f1r   = f1[0][n][m];
      pcnstField1D      f1i   = f1[1][n][m];

      pcnstField1D_cnst Coeff_m = sLaplacian->ppdCnst[m];

      for ( x=1;x<iSizeX;x++ )
      {
        f1r[x] += sLaplacian->pdCnstGradm[x]*f2r_m[x-1] +
        Coeff_m[x] * f2r_m[x] + sLaplacian->pdCnstGradp[x]*f2r_m[x+1];
        f1i[x] += sLaplacian->pdCnstGradm[x]*f2i_m[x-1] +
        Coeff_m[x] * f2i_m[x] + sLaplacian->pdCnstGradp[x]*f2i_m[x+1];
      }
    }
  }
  return rbm_er;
}
```

B.2 Code C pour la méthode d'Arakawa

La fonction qui suit résout l'opérateur de crochet de Poisson $\{.,.\}$ par la méthode d'Arakawa dans l'espace des variables spatiales physiques.

```
/**
*\fn void ArakawaMultiplication_base(double const *const re1,double
 const *const re2,double *const result, sFieldSize const*const sSizeReal,
 sFieldDiscret const*const sStepReal, sGeomOp const*const sGOP)
*\brief calcule le crochet de Poisson suivant l'algorithme d'Arakawa
```

```
*\param re1 : fonction 1
*\param re2 : fonction 2
*\param result : variable qui va contenir le resultat
*\param fftp : informations supplementaires necessaires en interne
*\return : le buffer ainsi modifie ou NULL en cas d'erreur
*/
void ArakawaMultiplication_base(double const *const re1,double
const *const re2,double *const result, sFieldSize const*const sSizeReal,
sFieldDiscret const*const sStepReal, sGeomOp const*const sGOP)
{
    static const char FCNAME[] = "ArakawaMultiplication";
    size_t step1,step2,step3,step4,step5,step6,step7,step8,step9;
    size_t k,l,m;
    size_t const iSizeX=sSizeReal->iSize1;
    size_t const iSizeY=sSizeReal->iSize2;
    size_t const iSizeZ=sSizeReal->iSize3;
    size_t const iStepYZ = (iSizeY)*(iSizeZ);
    double const dFactor = -1./(12.*sStepReal->dDx1*sStepReal->dDx2);

    for (l=1;l<iSizeX-1;l++)
    {
        for(k=0;k<iSizeZ;k++)
        {
            size_t const iStep = iSizeY*(k+l*iSizeZ);
            size_t const iStepP = iStep+iStepYZ;
            size_t const iStepM = iStep-iStepYZ;
            double const dCnst = dFactor*sGOP->pdInvRadius[l];
            for(m=1;m<iSizeY-1;m++)
            {
                step1 = iStep+m-1;  /*pos l,m-1*/
                step2 = step1+1;    /*pos l,m*/
                step3 = step2+1;    /*pos l,m+1*/

                step4 = iStepM+m-1; /*pos l-1,m-1*/
                step5 = step4+1;    /*pos l-1,m*/
                step6 = step5+1;    /*pos l-1,m+1*/

                step7 = iStepP+m-1; /*pos l+1,m-1*/
                step8 = step7+1;    /*pos l+1,m*/
                step9 = step8+1;    /*pos l+1,m+1*/

                result[step2]  = (re2[step1]+re2[step7]-re2[step3]-
                 re2[step9])*(re1[step8]+re1[step2]);
                result[step2] -= (re2[step4]+re2[step1]-re2[step6]-
```

```
     re2[step3])*(re1[step2]+re1[step5]);
    result[step2] += (re2[step8]+re2[step9]-re2[step5]-
     re2[step6])*(re1[step3]+re1[step2]);
    result[step2] -= (re2[step7]+re2[step8]-re2[step4]-
     re2[step5])*(re1[step2]+re1[step1]);
    result[step2] += (re2[step8]-re2[step3])
                    *(re1[step9]+re1[step2]);
    result[step2] -= (re2[step1]-re2[step5])
                    *(re1[step2]+re1[step4]);
    result[step2] += (re2[step3]-re2[step5])
                    *(re1[step6]+re1[step2]);
    result[step2] -= (re2[step8]-re2[step1])
                    *(re1[step2]+re1[step7]);
    result[step2] *= dCnst;
}
/* poisson bracket in the case of m=0 => m=-1 <=> m=Y-1*/
step1 = iStep+iSizeY-1;  /*pos l,m-1*/
step2 = iStep+0;         /*pos l,m*/
step3 = iStep+0+1;       /*pos l,m+1*/
step4 = iStepM+iSizeY-1; /*pos l-1,m-1*/
step5 = iStepM+0;        /*pos l-1,m*/
step6 = iStepM+0+1;      /*pos l-1,m+1*/
step7 = iStepP+iSizeY-1; /*pos l+1,m-1*/
step8 = iStepP+0;        /*pos l+1,m*/
step9 = iStepP+0+1;      /*pos l+1,m+1*/
result[step2]  = (re2[step1]+re2[step7]-re2[step3]-
 re2[step9])*(re1[step8]+re1[step2]);
result[step2] -= (re2[step4]+re2[step1]-re2[step6]-
 re2[step3])*(re1[step2]+re1[step5]);
result[step2] += (re2[step8]+re2[step9]-re2[step5]-
 re2[step6])*(re1[step3]+re1[step2]);
result[step2] -= (re2[step7]+re2[step8]-re2[step4]-
 re2[step5])*(re1[step2]+re1[step1]);
result[step2] += (re2[step8]-re2[step3])
                *(re1[step9]+re1[step2]);
result[step2] -= (re2[step1]-re2[step5])
                *(re1[step2]+re1[step4]);
result[step2] += (re2[step3]-re2[step5])
                *(re1[step6]+re1[step2]);
result[step2] -= (re2[step8]-re2[step1])
                *(re1[step2]+re1[step7]);
result[step2] *= dCnst;

/* poisson bracket in the case of m=Y-1 => m=Y <=> m=0*/
```

```
            m=iSizeY-1;
            step1 = iStep+m-1;   /*pos l,m-1*/
            step2 = step1+1;      /*pos l,m*/
            step3 = iStep+0;     /*pos l,m+1*/
            step4 = iStepM+m-1;  /*pos l-1,m-1*/
            step5 = step4+1;     /*pos l-1,m*/
            step6 = iStepM+0;    /*pos l-1,m+1*/
            step7 = iStepP+m-1;  /*pos l+1,m-1*/
            step8 = step7+1;     /*pos l+1,m*/
            step9 = iStepP+0;    /*pos l+1,m+1*/
            result[step2]  = (re2[step1]+re2[step7]-re2[step3]-
             re2[step9])*(re1[step8]+re1[step2]);
            result[step2] -= (re2[step4]+re2[step1]-re2[step6]-
             re2[step3])*(re1[step2]+re1[step5]);
            result[step2] += (re2[step8]+re2[step9]-re2[step5]-
             re2[step6])*(re1[step3]+re1[step2]);
            result[step2] -= (re2[step7]+re2[step8]-re2[step4]-
             re2[step5])*(re1[step2]+re1[step1]);
            result[step2] += (re2[step8]-re2[step3])
                             *(re1[step9]+re1[step2]);
            result[step2] -= (re2[step1]-re2[step5])
                             *(re1[step2]+re1[step4]);
            result[step2] += (re2[step3]-re2[step5])
                             *(re1[step6]+re1[step2]);
            result[step2] -= (re2[step8]-re2[step1])
                             *(re1[step2]+re1[step7]);
            result[step2] *= dCnst;
        }
    }
}
```

Annexe C

Liste des publications

Congrès internationaux avec comité de sélection et avec actes

[1] Imen Fassi, Philippe Clauss, Matthieu Kuhn and Yosr Slama, Multifor for Multicore, IMPACT 2013, Third International Workshop on Polyhedral Compilation Techniques, Jan 2013, Berlin, Germany. Epubli, Proceedings of the 3rd International Workshop on Polyhedral Compilation Techniques, pp. 37-44.

[2] Matthieu Kuhn, Guillaume Latu, Stéphane Genaud, Nicolas Crouseilles, Optimization and parallelization of Emedge3D on shared memory architecture, SYNASC 2013, HPC Workshop, Sep 2013, Timisoara, Romania. Publi, SYNASC 2013, 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 503 - 510.

Article soumis

[3] Nicolas Crouseilles, Matthieu Kuhn, Guillaume Latu, Comparison of numerical solvers for anisotropic diffusion equations arising in plasma physics.

Rapports de recherche

[4] Matthieu Kuhn, Guillaume Latu, Stéphane Genaud, Nicolas Crouseilles, Optimization and parallelization of Emedge3D on shared memory architecture, archive ouverte HAL -inria, référence interne RR-8336.

[5] Nicolas Crouseilles, Matthieu Kuhn, Guillaume Latu, Comparison of numerical solvers for anisotropic diffusion equations arising in plasma physics, archive ouverte HAL -inria, référence interne RR-8560.

[6] Alain Ketterlin, Matthieu Kuhn, Stéphane Genaud, Philippe Clauss, Loop-based Modeling of Parallel Communication Traces, archive ouverte HAL -inria, référence interne RR-8562.

Mémoire de master

[7] Matthieu Kuhn, Implémentation de la résolution des équations d'Euler sur multi-GPU via CUDA et MPI, Master CSSI, Université de Strasbourg.

Article en cours de préparation

[8] Matthieu Kuhn, Guillaume Latu, Nicolas Crouseilles, Hybrid MPI/OpenMP parallelization for an advection-diffusion problem arising in plasma physics.

Annexe D

Publications

Dans cette annexe sont présentés deux travaux effectués en collaboration avec des membres de l'équipe de recherche ICPS du laboratoire ICube dont j'ai fait parti. Ces travaux sont sans rapport direct avec le travail de thèse présenté dans ce manuscrit.

D.1 Multifor for multicore

Dans ce travail, une nouvelle structure de contrôle est proposée, qui s'appelle *multifor*. Elle permet de tirer pleinement avantage du modèle polyédrique, ainsi que d'exprimer plus facilement du parallélisme au sein des nids de boucles.

Ma contribution dans ce travail concerne en particulier la partie 5, concernant l'inversion des polynômes d'Ehrhart dits de "ranking", basés sur le modèle polyédrique. Maîtriser cette technique permet d'exécuter n'importe quel corps de boucle dans une autre boucle, sans casser l'ordre d'exécution des instructions.

Des tentatives de mise en œuvre de la structure *multifor* ont été effectuées, notamment en ce qui concerne la méthode d'Arakawa (voir [1]) et des algorithmes plus simples pour la diffusion. Lorsque le corps de boucle reste relativement simple (dans le cas d'un laplacien par exemple), il est possible de tirer un net bénéfice de la structure de contrôle présentée dans le papier qui suit (facteur d'accélération 2). Néanmoins, le stencil pour la méthode d'Arakawa étant plus compliqué, ce travail nécessite d'être approfondi pour pouvoir être présenté plus en détail.

Multifor for Multicore

Imèn Fassi
Dpt of Computer Science
Faculty of Sciences of Tunis
University El Manar
1060 Tunis, Tunisia
fassi.imen@gmail.com

Philippe Clauss
Team CAMUS, INRIA
University of Strasbourg
boulevard S. Brant
67400 Illkirch, France
philippe.clauss@inria.fr

Matthieu Kuhn
Team ICPS, LSIIT lab.
University of Strasbourg
boulevard S. Brant
67400 Illkirch, France
kuhn@unistra.fr

Yosr Slama
Dpt of Computer Science
Faculty of Sciences of Tunis
University El Manar
1060 Tunis, Tunisia
yosr.slama@gmail.com

ABSTRACT

We propose a new programming control structure called "multifor", allowing to take advantage of parallelization models that were not naturally attainable with the polytope model before. In a multifor-loop, several loops whose bodies are run simultaneously can be defined. Respective iteration domains are mapped onto each other according to a run frequency – the grain – and a relative position – the offset –. Execution models like dataflow, stencil computations or MapReduce can be represented onto one referential iteration domain, while still exhibiting traditional polyhedral code analysis and transformation opportunities. Moreover, this construct provides ways to naturally exploit hybrid parallelization models, thus significantly improving parallelization opportunities in the multicore era. Traditional polyhedral software tools are used to generate the corresponding code. Additionally, a promising perspective related to non-linear mapping of iteration spaces is also presented, yielding to run a loop nest inside any other one by solving the problem of inverting "ranking Ehrhart polynomials".

Categories and Subject Descriptors

D.3.4 [**Programming Languages**]: Processors

General Terms

Performance

Keywords

programming control structure, parallel programming, polytope model

1. INTRODUCTION

We have definitely entered a new era in programming. Parallelism is everywhere, from the many-core processor architectures that are from now on fitting mainstream computers, to the software applications that are now mixing intensive computations, specialized routines, network communication and multi-threading. Many researches focus in proposing new languages supposed to facilitate programming inside this complex environment [8, 10, 11, 13], or in proposing hardware or software support like transactional memory systems [6, 14, 15], supposed to prevent incorrect computations while still providing good performance. However, all these proposals face intractable issues. Most proposed languages imply to change drastically programmers habits and have weak chances to be adopted by the software industry [4]. Moreover, even if they offer interesting constructions to express parallel tasks, they are not solving the fundamental problem of correct and efficient parallelism extraction, which induces dependency analysis, data locality optimization, task grain adjustment, etc. Performance is also strongly dependent of their implementation, i.e. of their compilers or runtime systems. On the other hand, hardware or software support does not either result in hiding parallelization complexity to the user. And overall, these mechanisms are of high complexity by themselves, which mostly often make them unrealistic for a real usage [3].

Nevertheless, parallel programming has already a long history, where gradual extensions have been proposed. Some of them were pretty successful and are still current. For example, directive-based languages, as OpenMP [2], are extensions to mainstream languages. The use of their instructions is not mandatory when inserted in a source code, and they can be discovered and adopted progressively by any developer. They are not breaking the programming habits, while offering efficient parallelization opportunities. Although they are not solving either the fundamental complexity of parallelization, they nicely open the door of high performance computing to anyone.

At the same time, a lot of relevant transformation techniques have been discovered in order to exhibit parallelism or to optimize code, particularly on loops, as software pipelining, loop unrolling, tiling, skewing, etc [1, 16, 12]. These are applied either by experienced programmers, or automatically by compilers or optimization tools. In the parallelism era, compilers and runtime systems have to accelerate their progresses in automatic parallelization, but at the same time, programmers have to be brought to become, at least, who we called "experienced programmers" ten or twenty years ago.

We argue that a good way to achieve such an emancipation to parallel programming is to gradually extend mainstream languages with new programming control structures

that are derived from already existing ones. Our idea is that many well-known optimizing and parallelizing code transformations should now be applied naturally by developers, but only in their minds, while using a control structure translating their enriched algorithmic reasonings. The existence of such control structures will condition them to enlarge their way of reasoning while programming. In the same way that it is currently natural to express the repetition of code blocks by using loops, or to abstract parametrized code by using functions, it should now be natural to bring closer instructions that are using the same operands, or to arrange code snippets in vertical and horizontal directions to express simultaneity, sequencing and overlapping.

Following this idea, we propose a new control structure called *Multifor*, which can be seen as an extension of for-loops to multiple for-loops running at the same time, whose respective instructions are run sequentially inside one loop body as a traditional for-loop, but run in any interleaved order, or in parallel, between the bodies. Additionally to traditional parameters as indices, bounds and steps, we propose to introduce a grain and an offset, allowing to mix loops of different execution frequencies and of different starting positions. Such programming construction translates naturally to code transformations as loop fusion, loop pipelining or loop partitioning. Moreover, it facilitates many code optimizations as data locality improvement or parallelization. It can be seen as an extension of for-loops from "vertical" to "horizontal" programming.

The second motivation of this proposal is related to the parallel programming models that are covered by the polytope model. Traditional application of this model does not allow to naturally express parallel programming models as task parallelism, dataflow or MapReduce. We show that the multifor construct allows to schedule loop nest processes by mapping together their respective iteration domains. A multifor code can be represented geometrically by a particular union of polyhedra, each being previously dilated, compressed or translated, either entirely or partially, according to transformation factors defined by constants or affine functions.

This new programming structure implies interesting implementation challenges for a compiler, from its front-end to its backend. We show that, as it is already the case with for-loops, the polytope model is quite well adapted to analyze, optimize and translate multifor constructs into efficient code.

Finally, as a promising perspective, we also propose a non-linear mapping of the iteration domains guided by the ranks of the iterations. This approach opens the possibility of mapping any iteration domain onto any other domain without being constrained by their shapes. It leads to solve the general problem of executing any loop nest by any other loop nest of the same trip count. Mathematically speaking, the general solution to this problem is based on inverting "ranking Ehrhart polynomials" [5, 9].

The paper is organized as follows. In the next section, syntax and semantics of multifor loops are described, illustrated with a few examples of multifor headers and graphical representations. We also highlight the code parallelization and transformation schemes that are possible with multifor-loops. In Section 3, we discuss the main issues related to the implementation of multifor constructs and their corresponding code generation. Several real and representative code examples are presented in Section 4, highlighting the interesting contributions of this new control structure. The promising perspective of non-linear iteration space mapping is the topic of Section 5, where a solution for inverting ranking Ehrhart polynomials is proposed. Finally, conclusions and further perspectives are given in Section 6.

2. SYNTAX AND SEMANTICS

In this paper, we describe the initial syntax and semantics for the multifor construct. However, they can be extended in many ways in the future. We first present the case of one unique multifor construct, as the case of nested multifor-loops present some specificities which are presented afterwards.

2.1 Non-nested multifor-loops

The multifor syntax is defined by:

$$
\begin{array}{l}
\textbf{multifor}\ (\quad index_1 = expr, [index_2 = expr, ...]\ ; \\
\qquad\qquad\ \ index_1 < expr, [index_2 < expr, ...]\ ; \\
\qquad\qquad\ \ index_1 += cst, [index_2 += cst, ...]\ ; \\
\qquad\qquad\ \ grain_1, [grain_2, ...]\ ; \\
\qquad\qquad\ \ offset_1, [offset_2, ...]\ )\ \ \{ \\
\qquad\qquad\ \ prefix : \{statements\} \\
\}
\end{array}
$$

where [...] denotes optional arguments, $index_i$ denotes the indices of the loops composing the multifor, $expr$ denotes affine arithmetic expressions on enclosing loop indices, or constants, cst denotes an integer constant, $grain$ and $offset$ are positive integers, $grain \geq 1, offset \geq 0$, and $prefix$ is a positive integer associating each statement to a given for-loop composing the multifor-loop, according to the order in which they are defined (0 for the first loop, 1 for the second loop, etc.). Without loss of generality, we consider in the following that the index steps, cst, always equal one, since the general case can be easily deduced.

Each for-loop composing the multifor behaves as a traditional for-loop, but all are mapped on a same global "virtual referential" domain, which can also be seen as a template. The way iterations of the for-loops are mapped is defined by their respective offset and grain. The grain defines the frequency in which the associated loop has to run, relatively to the referential. For instance, if the grain equals 2, then one iteration of the associated loop will run over 2 iterations of the referential. The offset defines the gap between the first iteration of the referential and the first iteration of the associated loop. For instance, if the offset equals 3, then the first iteration of the associated loop will run at the fourth iteration of the referential loop.

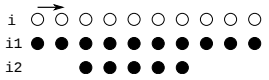
The size and shape of the referential is deduced from the for-loops composing the multifor-loop. Geometrically, it is defined as the disjoint union of the for-loop domains, where each domain has been previously shifted according to its offset and dilated according to its grain. The disjoint union is the union of adjacent convex domains, each being scanned by a referential for-loop. The relative positions of the iterations of the for-loops composing the multifor-loop inside the referential depends of the overlapping of their respective domains. It means that on domains where only one for-loop iterations are run, the grain becomes a compression factor. In general, the greatest common divisor of the grains of all the for-loops overlapping on a same referential domain

is used as the factor for compressing the points of this referential domain, according to the lexicographic order. On domains where several for-loops iterations are run, these are run in interleaved fashion, or simultaneously.

Let us illustrate this definition with a few examples. Consider the following multifor-loop header:

multifor $(i_1 = 0, i_2 = 10; i_1 < 10, i_2 < 15; i_1{+}{+}, i_2{+}{+}; 1, 1; 0, 2)$

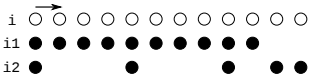
In this example, the offset of index i_1 is zero, and the one of index i_2 is 2. Thus, the first iteration of the i_1-loop will run immediatly, while the first iteration of the i_2-loop will run at the 3^{rd} iteration of the multifor, but with $i_2 = 0$. This behavior is illustrated by the figure below:



Notice that the index values have no effect on the relative positions of the for-loops bodies, which are uniquely determined by the grain and the offset. Another example is:

multifor $(i_1 = 0, i_2 = 10; i_1 < 10, i_2 < 15; i_1{+}{+}, i_2{+}{+}; 1, 4; 0, 0)$

Now, the i_1-grain is 1 and the i_2-grain is 4. In such a case, for one iteration of the i_2-loop, four iterations of the i_1-loop will be run on the domain on which they overlap. The second domain is compressed by a factor of 4, since only the i_2-loop is run, as it is illustrated below:



2.2 Nested multifor-loops

Nested multifor-loops present some particularities and specific semantics has to be described. Without loss of generality, let us consider two nested multifor-loops composed of two for-loop nests:

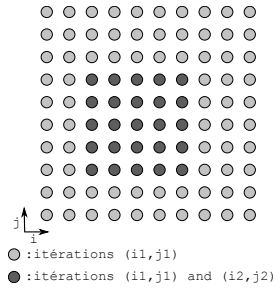
$$
\begin{array}{l}
\textbf{multifor}\ (\ \ index_1 = expr, index_2 = expr;\\
\qquad\qquad\ index_1 < expr, index_2 < expr;\\
\qquad\qquad\ index_1 += cst, index_2 += cst;\\
\qquad\qquad\ grain_1, grain_2;\\
\qquad\qquad\ offset_1, offset_2\ )\ \{\\
\qquad\qquad\ prefix : \{statements\}\\
\quad \textbf{multifor}\ (\ \ index_3 = expr, index_4 = expr;\\
\qquad\qquad\ index_3 < expr, index_4 < expr;\\
\qquad\qquad\ index_3 += cst, index_4 += cst;\\
\qquad\qquad\ grain_3, grain_4;\\
\qquad\qquad\ offset_3, offset_4\ )\ \{\\
\qquad\qquad\ prefix : \{statements\}\\
\quad \}\\
\qquad\qquad\ prefix : \{statements\}\\
\}
\end{array}
$$

Such a nest behaves as two for-loop nests, $(index_1, index_3)$ and $(index_2, index_4)$ respectively, running simultaneously in the same way as it is for one unique multifor-loop. The grain of the inner multifor-loop introduces a delay for the associated for-loop, since the same reasoning as with the non-nested case is applied at each loop depth. The lower and upper bounds are affine functions of the enclosing loop indices of the same for-loop[1]. Let us consider some examples of nested multifor headers.

[1]Notice that this restriction could be evicted for some amazing extensions.

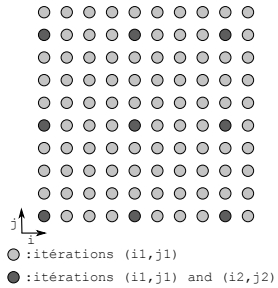
multifor $(i_1 = 0, i_2 = 0; i_1 < 10, i_2 < 5; i_1 + +, i_2 + +; 1, 1; 0, 2)$
multifor $(j_1 = 0, j_2 = 0; j_1 < 10, j_2 < 5; j_1 + +, j_2 + +; 1, 1; 0, 2)$

The second for-loop nest has a 2-offset at each loop depth. Hence it is delayed in each dimension of the referential domain:



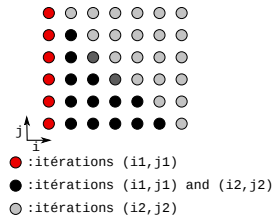
multifor $(i_1 = 0, i_2 = 0; i_1 < 10, i_2 < 3; i_1 + +, i_2 + +; 1, 4; 0, 0)$
multifor $(j_1 = 0, j_2 = 0; j_1 < 10, j_2 < 3; j_1 + +, j_2 + +; 1, 4; 0, 0)$

The second for-loop nest has a 4-grain at each loop depth. Hence its iterations are spaced by 4 in each dimension of the referential domain:



multifor $(i_1 = 0, i_2 = 0; i_1 < 6, i_2 < 6; i_1 + +, i_2 + +; 1, 1; 0, 1)$
multifor $(j_1 = 0, j_2 = 0; j_1 < 6 - i_1, j_2 < 6; j_1 + +, j_2 + +; 1, 1; 0, 0)$

In this example, the upper bound of the inner loop of the first loop nest is an affine function.



2.3 Multifor-loop parallelization and code transformations

The multifor construct exhibits a straightforward parallelization strategy which is to run, at each iteration, the loop bodies of the defined for-loops in parallel. This model of parallelization provides new opportunities to the polytope model, since it enables the expression of parallel programming models that were unattainable before, as dataflow computing or fixed-depth MapReduce, as it will be shown on code examples in Section 4.

Nevertheless, the multifor construct still allows OpenMP-like loop parallelization for each for-loop of the multifor, thus providing hybrid parallelization strategies.

Moreover, each for-loop, or for-loop nest, of a multifor, can be transformed using any well-known polyhedral transformation. However, in the context of a multifor, these transformations may be guided by the interactions between the for-loops, in order to achieve better performance or better data locality for instance. Another opportunity is the transformation of imperfect loop nests into multifor-loop nests of perfectly nested loops.

3. IMPLEMENTATION ISSUES

3.1 Reference domain

Consider one multifor-loop level. The referential for-loops cadencing the multifor execution have the constraint of scanning a sufficient number of iterations. Let us denote by f the number of for-loops. By computing the disjoint union of all for-loops iteration domains, we obtain a set of adjacent domains D_i on which some of the f loops overlap. Let us denote by $lb_i, ub_i, grain_i$ and $offset_i, i = 1..f$ the parameters characterizing each for-loop in the multifor header. Let us set $nlb_i = offset_i$ and $nub_i = (ub_i - lb_i + 1) \times grain_i + offset_i$, which define the lower and upper bounds of each loop in the referential domain, since the computation of nlb_i consists in translating the domain and the computation of nub_i in dilating the domain by a factor which equals the grain. The disjoint union of D_i's is computed using these latter bounds. The initial index value of the referential domain is $MIN_{i=1..f}(nlb_i)$. Hence, the total number of iterations in the referential domain, before compression, is:

$$MAX_{i=1..f}(nub_i) - MIN_{i=1..f}(nlb_i) + 1$$

In order to generate the referential for-loops for each domain D_i, the last step consists in compressing each D_i by a factor defined by $lcm(grain_j)$, for all loops j overlapping on D_i.

More generally for any multifor-loop nest, the computation of the referential domain is performed in three steps. First, each iteration domain associated to one for-loop nest composing a multifor-loop nest is translated from the origin according to its offsets, and dilated according to its grains in every dimension. Notice that values actually taken by the indices of the for-loop nests are not defining their positions in the referential domain. Second, a disjoint union is computed and resulting in a union of adjacent convex domains D_i. Third, each D_i may be compressed according to the greatest common divisor of the grains of the associated for-loop nests, and according to the lexicographic order.

3.2 Code generation

When considering sequential code, there are two dual ways to generate the code corresponding to a multifor-loop nest. A first way is to generate loop nests scanning the referential domains through a minimal set of convex domains, and to insert guards in their bodies in order to execute the convenient instructions at each iteration. The number of these guards can be optimized by computing their common sub-domains. The second way is to scan each D_i using a dedicated loop nest with a constant loop body, without guards.

Both solutions can be generated automatically using polytope model tools like PolyLib or CLooG, and by inserting phases to compute the translated and dilated domains, or to compress parts of the resulting referential domains.

If for-loops of given depth of a multifor-loop nest have to be run in parallel, each for-loop has to be run in a separated thread and all threads have to be synchronized at the multifor-loop completion. Notice that this could be enriched by providing OpenMP-like options as NOWAIT.

```
for (i = 0; i < K; i++)
 for (j = 0; j < N; j++)
  a[i][j] = ReadImage();
for (i = 1; i < K - 1; i++)
 for (j = 1; j < N - 1; j++){
  Sbl[i][j] = Sobel(a[i - 1][j - 1], a[i][j - 1], a[i + 1][j - 1],
                    a[i - 1][j], a[i][j], a[i + 1][j],
                    a[i - 1][j + 1], a[i][j + 1], a[i + 1][j + 1]);
  WriteImage(Sbl[i][j]);
}
```

Figure 1: Sobel edge detection code

```
multifor (i1 = 0, i2 = 1; i1 < K, i2 < K - 1;
          i1++, i2++; 1, 1; 0, 3)
 multifor (j1 = 0, j2 = 1; j1 < N, j2 < N - 1;
           j1++, j2++; 1, 1; 0, 3) {
  0 : a[i1][j1] = ReadImage();
  1 : {Sbl[i2][j2] = Sobel(a[i2 - 1][j2 - 1], a[i2][j2 - 1],
                           a[i2 + 1][j2 - 1], a[i2 - 1][j2],
                           a[i2][j2], a[i2 + 1][j2],
                           a[i2 - 1][j2 + 1], a[i2][j2 + 1],
                           a[i2 + 1][j2 + 1]);
       WriteImage(Sbl[i2][j2]); }
 }
```

Figure 2: Sobel edge detection multifor code

Original indices of the multifor (i_1, i_2, j_1, ...) have to be retrieved at the beginning of each loop body, by being computed from the referential loop indices. These computations consists in subtracting offsets, or in adding modulos of the referential loop indices relatively to grains, or in multiplying by grains in case of compressed domains.

4. EXAMPLES

Sobel edge detection: We first consider the code for performing Sobel edge detection of an image shown in Figure 1. The first loop nest of this program reads the input image, while the second loop nest performs the actual edge detection and writes out the output image.

Note that nine neighboring elements have to be read before its resulting pixel can be computed and written. Hence both loop nests can be naturally overlapped by writing the code using the multifor construct exhibiting a data-flow model of computation, shown in Figure 2. The associated referential domain is shown in Figure 3.

Red-Black Gauss-Seidel: The second example is the Red-Black Gauss-Seidel algorithm composed of two phases. The first phase consists in updating the red elements of a grid, which are one point over two in the i and j directions of the grid, starting from the first bottom left corner, using their North-South-East-West (NSEW) neighbors, which are black elements. The second phase consists obviously in updating the black elements from the red ones. For a 2D $N \times N$ problem, the usual code, is of the form shown in Figure 4 (the border elements initialization has been omitted).

On the iteration domain and at each phase, a different

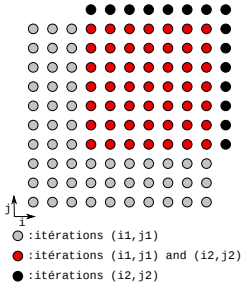


Figure 3: Sobel edge detection referential domain

```
// Red phase
for (i = 1; i < N − 1; i++)
 for (j = 1; j < N − 1; j++)
  if ((i + j) % 2 == 0)
    u[i][j] = f(u[i][j + 1], u[i][j − 1], u[i − 1][j], u[i + 1][j]);
// Black phase
for (i = 1; i < N − 1; i++)
 for (j = 1; j < N − 1; j++)
  if ((i + j) % 2 == 1)
    u[i][j] = f(u[i][j + 1], u[i][j − 1], u[i − 1][j], u[i + 1][j]);
```

Figure 4: Red-Black Gauss-Seidel code

lattice of iterations is active. Moreover, the NSEW dependencies prevent any linear parallel schedule. This code can be translated into a multifor-loop nest where the red and black phases each yield two for-loop nests with convenient grains and offsets as shown in Figure 5.

The referential initial domain of the multifor code is represented in Figure 6 on the left, also showing the transformed dependency vectors which are now allowing a linear schedule. On the right in Figure 6, the final iteration domains, after compression, are represented.

This example shows that the multifor construct allows to exploit different kind of parallelism – the so-called wavefront parallelism in this case – since the traditional parallelization consists in parallelizing each of the phases, and to execute each phase one after the other. The multifor strategy can be preferable to improve data locality and thus improve the

```
multifor (i0 = 1, i1 = 2, i2 = 1, i3 = 2; i0 < N − 1, i1 < N − 1,
          i2 < N − 1, i3 < N − 1; i0+ = 2, i1+ = 2, i2+ = 2,
          i3+ = 2; 2, 2, 2, 2; 0, 1, 0, 1)
 multifor (j0 = 1, j1 = 2, j2 = 2, j3 = 1; j0 < N − 1, j1 < N − 1,
           j2 < N − 1, j3 < N − 1; j0+ = 2, j1+ = 2, j2+ = 2,
           j3+ = 2; 2, 2, 2, 2; 0, 1, 1, 0) {
 0 : u[i0][j0] =
     f(u[i0][j0 + 1], u[i0][j0 − 1], u[i0 − 1][j0], u[i0 + 1][j0]);
 1 : u[i1][j1] =
     f(u[i1][j1 + 1], u[i1][j1 − 1], u[i1 − 1][j1], u[i1 + 1][j1]);
 2 : u[i2][j2] =
     f(u[i2][j2 + 1], u[i2][j2 − 1], u[i2 − 1][j2], u[i2 + 1][j2]);
 3 : u[i3][j3] =
     f(u[i3][j3 + 1], u[i3][j3 − 1], u[i3 − 1][j3], u[i3 + 1][j3]);
}
```

Figure 5: Red-Black Gauss-Seidel multifor code

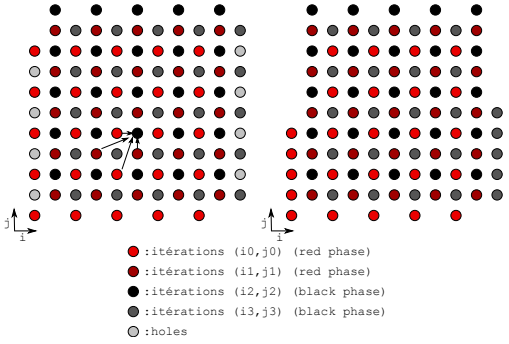


Figure 6: Red-Black Gauss-Seidel referential domain

```
for (j = 1; y < N − 1; j+ = 2)
 u[i][j] = f(u[i][j + 1], u[i][j − 1], u[i − 1][j], u[i + 1][j]);
for (i = 2; x < N − 2; i+ = 2) {
 for (j = 2; j < N − 1; j+ = 2) {
  u[i][j] = f(u[i][j + 1], u[i][j − 1], u[i − 1][j], u[i + 1][j]);
  u[i][j + 1] = f(u[i][j + 2], u[i][j],
                  u[i − 1][j + 1], u[i + 1][j + 1]); }
 for (j = 1; j < N − 1; j+ = 2) {
  u[i + 1][j] = f(u[i + 1][j + 1], u[i + 1][j − 1],
                  u[i][j], u[i + 2][j]);
  u[i + 1][j + 1] = f(u[i + 1][j + 2], u[i + 1][j],
                      u[i][j + 1], u[i + 2][j + 1]); } }
for (j = 2; y < N − 1; j+ = 2) {
 u[N − 2][j] = f(u[N − 2][j + 1], u[N − 2][j − 1],
                 u[N − 3][j], u[N − 1][j]); }
```

Figure 7: Red-Black Gauss-Seidel generated code

resulting execution time. Here, some parallelism within the red points and within the black points can still be exploited, but also parallelism between red and black points. As an example, we show as a source code the sequential code that could be generated from this multifor-loop nest in Figure 7.

Notice also that more generally, when dependencies allow it, such a decomposition of a loop-nest computation into separated lattices, and expressed as a multifor-loop nest, provides another parallelization strategy that may be often quite interesting due to data locality issues.

Matrix product by blocks: The third example is an algorithm to compute the product of two matrices $n \times n$, $(A \times B = C)$, by partitioning the matrices into uniform blocks. The matrix product is then carried out block by block. We split the two matrices A and B as follows:

- matrix A is divided into two matrices A_1 and A_2 whose dimension is $n/2 \times n$.
- matrix B is divided into two matrices B_1 and B_2 whose dimension is $n \times n/2$.

The product $A \times B = C$ translates to four products: $A_1 * B_1 = C_1, A_1 * B_2 = C_2, A_2 * B_1 = C_3$ and $A_2 * B_2 = C_4$, i.e.,

$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \times \begin{pmatrix} B_1 & B_2 \end{pmatrix} = \begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix}$$

```
multifor (i1 = 0, i2 = 0, i3 = n/2, i4 = n/2;
          i1 < n/2, i2 < n/2, i3 < n, i4 < n;
          i1 ++, i2 ++, i3 ++, i4 ++;
          1, 1, 1, 1 ; 0, 0, 0, 0 ) {
  multifor (j1 = 0, j2 = n/2, j3 = 0, j4 = n/2;
            j1 < n/2, j2 < n, j3 < n/2, j4 < n;
            j1 ++, j2 ++, j3 ++, j4 ++;
            1, 1, 1, 1 ; 0, 0, 0, 0 ) {
    0 : c[i1][j1] = 0;
    1 : c[i2][j2] = 0;
    2 : c[i3][j3] = 0;
    3 : c[i4][j4] = 0;
      multifor (k1 = 0, k2 = 0, k3 = 0, k4 = 0;
                k1 < n, k2 < n, k3 < n, k4 < n;
                k1 ++, k2 ++, k3 ++, k4 ++;
                1, 1, 1, 1 ; 0, 0, 0, 0 ) {
        0 : c[i1][j1] = c[i1][j1] + a[i1][k1] × b[k1][j1];
        1 : c[i2][j2] = c[i2][j2] + a[i2][k2] × b[k2][j2];
        2 : c[i3][j3] = c[i3][j3] + a[i3][k3] × b[k3][j3];
        3 : c[i4][j4] = c[i4][j4] + a[i4][k4] × b[k4][j4];
        }
    }
}
```

Figure 8: Multifor matrix product code

The dimension of matrices C_1, C_2, C_3 and C_4 is $(n/2 \times n/2)$. These four products might be performed simultaneously and can be naturally expressed using the multifor structure as shown in figure 8.

Geometrically, the multifor iteration domain is a $(n/2 \times n/2 \times n)$ rectangle parallelepiped where each point is associated to four iterations of the four included loop-nests. This execution scheme corresponds to the MapReduce strategy since each for-loop nest computes a $n/2 \times n/2$ sub-block (map step), and the combination of all sub-blocks forms the resulting matrix C (reduce step).

Steganography: The fourth example is the decoding phase of a steganography code where an hidden image is extracted from an enclosing one. It is assumed that the upper left pixel of the hidden image is hidden within the upper left pixel of the enclosing image; $HWidth$ and $HHeight$ are the width and the height of the hidden image ; $EWidth$ and $EHeight$ are the width and the height of the enclosing image ; $EImage$ is the image hiding another image ; $HImage$ is the extracted output image that was hidden ; $MImage$ is the output enclosing image hiding no more the image that was hidden in $EImage$. The proposed multifor code version is composed of four simultaneous for-loop nests, the first being dedicated to the extraction of the hidden image, the second to the extraction of the part of the enclosing image which is hiding the hidden image, the third and the fourth being dedicated to copy the pixels directly to the retrieved enclosing image. Since the union of the third and fourth domain is not convex, two loop-nests are necessary to scan it. Notice that we introduce a shortcut in the syntax such that similar loop bodies can be instantiated differently depending on their associated loop-nest. The multifor code is shown in Figure 4 and the referential iteration domain in Figure 9. Notice that full parallelism is exhibited with this code.

Secret key cryptosystem: The fifth example is a classic secret key cryptosystem that manipulates binary words. It

```
RGBApixel decode_hidden(i, j)
{
 RGBApixel Pixel1 = ∗EImage(i, j);
 RGBApixel Pixel2;
 Pixel2.Red = Pixel1.Red%2;
 Pixel2.Green = Pixel1.Green%2;
 Pixel2.Blue = Pixel1.Blue%2;
 Pixel2.Alpha = Pixel1.Alpha%2;
 returnPixel2;
}

RGBApixel decode_main(i, j)
{
 RGBApixel Pixel1 = ∗EImage(i, j);
 RGBApixel Pixel2;
 Pixel2.Red = Pixel1.Red − Pixel1.Red%2;
 Pixel2.Green = Pixel1.Green − Pixel1.Green%2;
 Pixel2.Blue = Pixel1.Blue − Pixel1.Blue%2;
 Pixel2.Alpha = Pixel1.Alpha − Pixel1.Alpha%2;
 returnPixel2;
}

multifor (i1 = 0, i2 = 0; i3 = 0, i4 = HWidth; i1 < HWidth,
          i2 < HWidth, i3 < HWidth, i4 < EWidth;
          i1 ++, i2 ++, i3 ++, i4 ++; 1, 1, 1, 1; 0, 0, 0, 0)
 multifor (j1 = 0, j2 = 0, j3 = HHeight, j4 = 0; j1 < HHeight,
          j2 < HHeight, j3 < EHeight, j4 < EHeight;
          j1 ++, j2 ++, j3 ++, j4 ++; 1, 1, 1, 1; 0, 0, 0, 0)
{
 0 : // Retrieve the hidden image
     ∗ HImage(i1, j1) = decode_hidden(i1, j1);
 1 : // Retrieve the enclosing image
     ∗ MImage(i2, j2) = decode_main(i2, j2);
 [2, 3] : // Retrieve the enclosing image
     ∗ MImage([i3, i4], [j3, j4]) = ∗EImage([i3, i4], [j3, j4]);
}
```

Figure 9: Multifor steganography code for the decoding phase

proceeds by splitting a message m into blocks of constant size. These cryptosystems are characterized by the length of each block, the operating mode and the encryption system of each block. Each cipher mode comprises:

1. Cutting in many blocks $m_1, ..., m_k$ the plain text message m;
2. Encrypting the blocks m_i resulting in the encrypted blocks $c_1, ..., c_k$;
3. Concatenating the blocks $c_1, ..., c_k$ to construct the encrypted message c.

Each block is encrypted through the product of two cryptosystems T_1 and T_2. It is classically computed using a loop of the form:

```
for (i = 0; i < k; i ++) {
    c[i] = Encrypt(m[i], T1);
    c[i] = Encrypt(c[i], T2); }
```

Suppose the encryption of each block by a given cryptosytem consumes one unit of time. The time required to encrypt the entire message using this loop is $2 \times k$. Let us write this code using a multifor structure:

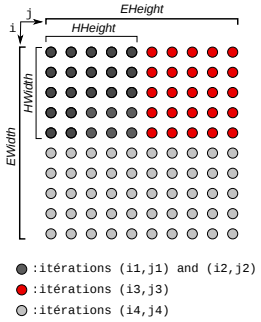


Figure 10: Referential domain for the multifor steganography code

$$\begin{array}{l}\textbf{multifor } (i_1 = 0, i_2 = 0; i_1 < k, i_2 < k;\\ \qquad i_1++, i_2++; 1, 1; 0, 1)\ \{\\ 0 : c[i_1] = Encrypt(m[i_1], T_1);\\ 1 : c[i_2] = Encrypt(c[i_2], T_2);\}\end{array}$$

This form allows to take advantage of a pipeline scheme where two successive blocks are encrypted in parallel respectively by the cryptosystems T_1 and T_2. Thus, the time required to encrypt the message is $k + 1$.

5. A PROMISING PERSPECTIVE: NON-LINEAR MAPPING

Among the numerous possible extensions, an important one is to map iteration spaces together following a non-linear fashion, such that their shapes has no influence in the mapping. In general, this would allow to execute iterations of any loop nest by any other one of the same trip count, and thus to enlarge significantly the way iterations of different loops can be mapped together. Hence a multifor construct could express quite different computations in a concise way, and augment the number of optimization and parallelization opportunities. Notice that loop nests with different trip counts can also be handled by splitting the largest nest such that one of the resulting nest has the convenient trip count.

Our idea is based on ranking Ehrhart polynomials. It has been shown in previous works dealing with spatial data locality optimization, that it is possible to compute an Ehrhart polynomial associated to a loop nest giving the rank of an iteration [5, 9]. These polynomials have specific properties as being necessarily monotonically increasing according to the lexicographic order of the loop indices, and also defining a bijection between iteration points and the interval of strictly positive integers between one and the total iteration count of the loop nest.

Since ranking Ehrhart polynomials define such bijections, the ability of inverting them would provide a way to retrieve the loop indices corresponding to the rank of an iteration. Hence at any iteration of a loop nest, it would be possible to compute, from the rank, the values of loop indices that would have been reached while running another nest. Thus, any nest could be run by another one, and any iteration space could be mapped onto another one, following the rank of the iterations. Hence this Section deals with the problem of inverting ranking Ehrhart polynomials.

Before proposing a general resolution, we first present a 2-dimensional example.

5.1 2-dimensional example

Consider the two loop nests in listings (1) and (2), where $instruction_k(i_1, i_2)$ denotes the instruction block computed at iteration (i_1, i_2). These loops have as respective ranking Ehrhart polynomials:

$$P_1(i,j) = \frac{i(i-1)}{2} + j \text{ and } P_2(i',j') = i'M_2 + j'$$

$$\begin{array}{l}\textbf{for } (i = 1, i < N, i++)\\ \quad \textbf{for } (j = 0, j < i, j++)\\ \qquad instructions_1(i,j);\end{array} \quad (1)$$

$$\begin{array}{l}\textbf{for } (i' = 0, i' < M_1, i'++)\\ \quad \textbf{for } (j' = 0, j' < M_2, j'++)\\ \qquad instructions_2(i',j');\end{array} \quad (2)$$

Taking the assumptions that both nests have the same iteration count and that there is no dependency between $instructions_1$ and $instructions_2$, we could merge the two former loop nests and write (3).

$$\begin{array}{l}\textbf{for } (i' = 0, i' < M_1, i'++)\\ \quad \textbf{for } (j' = 0, j' < M_2, j'++)\ \{\\ \qquad instructions_1(\mathbf{i},\mathbf{j});\\ \qquad instructions_2(i',j');\ \}\end{array} \quad (3)$$

However, we need to express indices (i, j) as a function of (i', j') in order to preserve the execution order of the block $instructions_1$. More precisely, for each iteration number K in loop nest (3), we want to execute the K^{th} iteration of loop nest (1). This is why we must invert the ranking Ehrhart polynomial P_1, to compute $(i, j) = P_1^{-1}(P_2(i', j'))$.

For any rank K, we have to find a couple of indices (i_0, j_0) such that $P_1(i_0, j_0) = K$. The main idea is to cut the 2-dimensional problem into two one-dimensional problems.

Let us define $Q_1(i) = P_1(i, 0) = \frac{i(i-1)}{2}$. As ranking Ehrhart polynomials are (strictly) increasing, the following relation holds:

$$Q_1(i_0) = P_1(i_0, 0) \leq P_1(i_0, j_0) = K \leq P_1(i_0 + 1, 0) = Q_1(i_0 + 1)$$

And, for the same reason, we know that index i_0 is unique on $\mathbb{N}_+$. Let us now consider polynomial Q_1 as a polynomial over $\mathbb{R}$. By continuity of Q_1 over $\mathbb{R}$, there exists $\alpha \in [0, 1[$ such that $Q_1(i_0 + \alpha) = K$. This shows that the equation $Q_1(x) = K$ has at least one real solution. So we have to find x such that:

$$Q_1(x) = K \Leftrightarrow Q_1(x) - K = 0 \Leftrightarrow \frac{x(x-1)}{2} - K = 0$$

Obviously, this last equation has two real roots:

$$x_1 = \frac{1}{2} - \sqrt{\frac{1+8K}{4}}, x_2 = \frac{1}{2} + \sqrt{\frac{1+8K}{4}}$$

To select the convenient root, we notice that for all $K > 0$, $x_1 \leq 0$. As $i_0 \geq 1$ (according to the loop bounds in (1)), $i_0 + \alpha = x_2$, and thus: $i_0 = \lfloor x_2 \rfloor$. We can now replace i_0 by its value in $P_1(i_0, j_0)$:

$$P(i_0, j_0) = \frac{1}{2}\left(\left\lfloor \frac{1}{2} + \sqrt{\frac{1+8K}{4}} \right\rfloor\right)\left(\left\lfloor \frac{1}{2} + \sqrt{\frac{1+8K}{4}} \right\rfloor - 1\right) + j_0$$

and finally deduce j_0:

$$j_0 = K - \frac{1}{2}\left(\left\lfloor \frac{1}{2} + \sqrt{\frac{1+8K}{4}} \right\rfloor\right)\left(\left\lfloor \frac{1}{2} + \sqrt{\frac{1+8K}{4}} \right\rfloor - 1\right)$$

The resulting code is shown in listing 4.

```
K = 0;
for (i' = 0, i' < M1, i' ++)
   for (j' = 0, j' < M2, j' ++) {
      K ++;
      i = floor(sqrt((1 + 8 * K)/4) + 1/2);
      j = K − i * (i − 1)/2;
      instructions1(i, j);
      instructions2(i', j'); }
```
(4)

We now present the general case, which can be easily deduced from the 2-dimensional case.

5.2 General case

Without any loss of generality, we assume all loop indices lower bounds equal 0. We consider the N-dimensional ranking Ehrhart polynomial $P(i, j, k, ...)$, and for each K, we seek the tuple $(i_0, j_0, k_0, ...)$ such that $P(i_0, j_0, k_0, ...) = K$.

Similarly to the previous example, we start with the outermost loop index i_0. We define $Q_i(i) = P(i, 0, 0, ..., 0)$ and solve $Q_i(x) - K = 0$. Here is the only issue that differs from the 2D case: as we can't state that Q_i is monotonically increasing on $\mathbb{R}_+$, we have to find a criterion to select the root giving the sought index.

First, we obviously eliminate complex and negative solutions, since $i_0 \in \mathbb{N}_+$, and consider $n \leq N$ positive real roots $\{x_1, .., x_n\}$. As we know that i_0 is unique, a way to select the convenient root is to check which $x \in \{x_1, .., x_n\}$ satisfies :

$$Q_i(\lfloor x \rfloor = i_0) \leq Q_i(x) \leq Q_i(\lceil x \rceil = i_0 + 1)$$

However, this strategy is only applicable at runtime, and may add non negligible time overhead. A compile-time solution is to check if Q_i is monotonically increasing on $\mathbb{R}_+$ by examining its derivative. If so, any root in $\{x_1, .., x_n\}$ is suitable.

Once i_0 has been found, the process starts again with $Q_j(j) = P(i_0, j, 0, ..., 0)$, and so on until all indices have been computed.

6. CONCLUSION

We have proposed a new programming control structure called "multifor" and showed that it allows the polytope model to handle programming models that were not attainable directly before. Important related theoretical studies have still to be conducted, as dependency analysis between the for-loops composing a multifor-loop, or optimizing code transformations that considers interactions between the for-loops.

Many interesting extensions can also be studied as making header parameters, or instructions, dependent of several for-loop indices composing the same multifor-loop level, or defining non-invariant grains and offsets, or introducing conditionals on the effective run of the for-loops, etc. In this paper, we showed that it may be possible to handle non-linear mapping of iteration spaces using inverted ranking Ehrhart polynomials.

The multifor structure can also be used as a representation model for some interacting mechanisms, as concurrent memory accesses, as it is done for sequential codes in [7].

We are planning to implement multifor structures in the Clang/LLVM compiler as an extension to C/C++.

7. REFERENCES

[1] U. Banerjee. *Loop Transformations for Restructuring Compilers - The Foundations*. Kluwer Academic Publishers, 1993. ISBN 0-7923-9318-X.

[2] O. A. R. Board. Openmp application program interface, version 3.1, 2011.

[3] C. Cascaval, C. Blundell, M. Michael, H. W. Cain, P. Wu, S. Chiras, and S. Chatterjee. Software transactional memory: Why is it only a research toy? *Queue*, 6(5):46–58, Sept. 2008.

[4] I. Christadler, G. Erbacci, and A. D. Simpson. Facing the multicore-challenge ii. chapter Performance and productivity of new programming languages, pages 24–35. Springer-Verlag, Berlin, Heidelberg, 2012.

[5] P. Clauss and B. Meister. Automatic memory layout transformations to optimize spatial locality in parameterized loop nests. *SIGARCH Comput. Archit. News*, 28(1):11–19, Mar. 2000.

[6] M. Herlihy, V. Luchangco, M. Moir, and W. N. Scherer, III. Software transactional memory for dynamic-sized data structures. In *Proc. of the 22nd annual symp. on Principles of distributed computing*, PODC '03, pages 92–101. ACM, 2003.

[7] A. Ketterlin and P. Clauss. Prediction and trace compression of data access addresses through nested loop recognition. In *6th annual IEEE/ACM int. symp. on Code generation and optimization*, pages 94–103, Boston, United States, Apr. 2008. ACM.

[8] C. E. Leiserson. The cilk++ concurrency platform. In *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pages 522–527, New York, NY, USA, 2009. ACM.

[9] V. Loechner, B. Meister, and P. Clauss. Precise data locality optimization of nested loops. *J. Supercomput.*, 21(1):37–76, Jan. 2002.

[10] S. Marlow, P. Maier, H.-W. Loidl, M. K. Aswad, and P. Trinder. Seq no more: Better strategies for parallel haskell. In *Proceedings of the 3rd ACM SIGPLAN symposium on Haskell*, pages 91–102, Baltimore, MD, United States, Sept. 2010. ACM Press.

[11] M. Odersky, L. Spoon, and B. Venners. *Programming in Scala*. Artima Series. Artima Press, 2011.

[12] L.-N. Pouchet, U. Bondhugula, C. Bastoul, A. Cohen, J. Ramanujam, P. Sadayappan, and N. Vasilache. Loop transformations: convexity, pruning and optimization. In *Proc. of the 38th annual ACM SIGPLAN-SIGACT symp. on Principles of programming languages*, POPL '11, pages 549–562, New York, NY, USA, 2011. ACM.

[13] K. F. Sagonas. Using static analysis to detect type errors and concurrency defects in erlang programs. In *FLOPS*, pages 13–18, 2010.

[14] B. Saha, A.-R. Adl-Tabatabai, R. L. Hudson, C. C. Minh, and B. Hertzberg. Mcrt-stm: a high performance software transactional memory system for a multi-core runtime. In *Proc. of the 11th ACM SIGPLAN symp. on Principles and practice of parallel programming*, PPoPP '06, pages 187–197, New York, NY, USA, 2006. ACM.

[15] N. Shavit and D. Touitou. Software transactional memory. In *Proc. of the 14th annual ACM symp. on Principles of distributed computing*, PODC '95, pages 204–213, New York, NY, USA, 1995. ACM.

[16] M. J. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

D.2 Loop-based Modeling of Parallel Communication Traces

Pour analyser un programme MPI, les traces de communication sont un outil central. Les programmes devenant de plus en plus lourds, stocker la totalité des communications entre les processus et être capable de les restituer de manière logique devient un enjeu majeur. Le papier qui suit propose une méthode de compression de traces MPI par un algorithme incrémental (NLR), ainsi qu'une manière de représenter les traces basée sur des horloges logiques. Cette méthode analyse et reconstruit les traces sous formes de boucles. Cela permet un rendu clair et synthétique d'une trace de programme MPI. Cette méthode est appliquée sur les NAS parallel benchmarks.

Loop-Nest Recognition for the Extraction of Communication Patterns and the Compression of Message-Passing Parallel Traces

Alain Ketterlin Stéphane Genaud Matthieu Kuhn
University of Strasbourg
Pôle API, Blvd S. Brant, F-67400 Illkirch
Email: {ketterlin,genaud,kuhn}@unistra.fr

***Abstract*—Communication traces are central to the understanding and performance analysis of message-passing parallel programs. Visualizing such traces helps the programmer to quickly understand the detailed chronology of the events by exhibiting the geometric communication patterns between processes. It is often also the start of more detailed performance studies. However, the representation of large traces with a high level of detail may obfuscate essential communication structures.**

This paper introduces a combination of two algorithms that extracts a formal model of the communication scheme of a distributed program from a collection of local traces. The first, node-local part is a fast and unobtrusive extractor of repetitive behavior. The second, system-wide part is a pairwise merger, aligning the patterns from two (sets of) nodes. The overall process models repetitive behavior as a graph of nested loops. The final result is thus both extremely compact and highly expressive: it can be replayed directly, and its direct graphical rendering provides a novel visualization technique, highlighting non-trivial topological properties. The underlying formalism is simple and general enough to provide a foundation for more elaborate empirical studies of distributed systems.

***Index Terms*—Message Passing Interface; Program Analysis; Trace compression;**

I. Introduction

The ever-growing scale and complexity of parallel systems, that will commonly have thousands to hundreds of thousands of cores will enable to run more and more massively parallel applications. In order to design, develop and tune such applications, a large variety of tools is required. The code tuning phase requires tools to monitor and analyze the program behavior on an actual execution platform. We can distinguish between two complementary approaches for this task: profiling and tracing. Profiling aims to record aggregated data that reflects specific aspects of the program behavior. It is generally designed to have the lowest possible intrusiveness in order to collect accurate measurements. In contrast, tracing addresses the need to record all communication events to understand how the concurrent processes interact all along the program execution. Tracing is challenging because it involves huge amounts of records.

Post-mortem examination of those records reveals how each process behaved, and shows the causes of a potential loss of performance. In this paper, we focus on how the numerous time-based events contained in communication traces can be translated to some higher-level information which summarizes the communication scheme of an application. This approach has also been tried through profiling. For example [1] use clustering on data collected by processor performance counters to identify groups of processes with similar behavior, or computation phases with similar characteristics. However, communication traces are central to the analysis and understanding of message-passing parallel programs. The need to understand the program at different levels of details is witnessed by the compelling usage of visualization tools like Jumpshot [2]. However, the exhaustive log of events displayed by visualization tools often yields complicated and varied communication patterns, which obfuscates the essential logic of the program.

In previous work, we have developed a method to model a series of numbers as a sequence of loop nests [3]. In this paper, we adapt this algorithm to the modeling of message-passing parallel traces. We show that the loop nest recognition technique usually correctly identifies the time-steps of typical executions. The resulting loops also provide short, up-to-the-point visualizations of complete executions, showing repeated communication patterns once and labeling them with their number of occurrences. Our method essentially abstracts the timing of the communication events, using logical time reconstructed from causal dependencies between events.

Our loop nest recognition algorithm (NLR), is also able to infer loops containing affine descriptions, e.g., representing a message as $(i \rightarrow i+1)$, where i is the counter of an enclosing loop. This capability further abstracts the actual execution, highlighting the underlying logical topology. It also provides for much shorter descriptions of structured exchange patterns, as commonly found in numerical applications.

This capability of summarizing communication events can be further used for trace compression. We thereby address the challenge of the huge size of trace files. To summarize, the present work makes the following contributions:

- A methodology to abstract repeated communication patterns in message-passing programs. This abstract representation can be used in several ways to understand the stucture of a parallel program's communications.
- A novel visualization approach, based on our abstraction methodology, which provides a compact yet exact and comprehensive view of the communication events. In

comparison to most related works based on the analysis or/and visualization of time-based events, our visualization offers an event graph view of repeated patterns, on a logical time-scale.

The paper is structured as follows. First, we present the related work and how our own work compares to other approaches to trace analysis. We then describe the technical side of the work, based on nested loop recognition. Section IV explains how traces are linearized before global behavior can be modeled. Section V describes how visualization of large-scale regular patterns can be built from the output of NLR, and illustrates the results on most significant NAS Parallel as well as on a real application kernel (Sweep3d). Finally, section VI discusses the compression with a quantitative evaluation on the benchmarks, before we conclude.

II. Related Work

A. Parallel Trace Modeling

There are basically two approaches to trace processing. The first one, profiling, includes timing information and/or aggregates of quantities; see, e.g., [1]. The second is more qualitative in nature, focusing on building an abstract model of the program behavior. This paper addresses a specific form of the latter approach, in the particular case of parallel and distributed systems. The goal is to build a model of all communications between processes. In that field many, if not all, studies have been focusing on detecting exact repetitions in a trace represented as a word on a finite alphabet. For instance, ScalaTrace [4], [5] includes an incremental algorithm not unlike our NLR algorithm. Krishnamoorthy and Agarwal [6] use variations of Sequitur [7] to build one or more grammars from a trace. Xu *et al* [8] use a variation of the Crochemore algorithm [9] to locate the repetitions. In all cases, the trace is made of discrete symbols taken in some specific finite alphabet.

The main novelty of our work is that NLR is able to infer loop nests where any expression is an affine function of its enclosing loop counters. This raises the level of abstraction, and provides a basis for compact representations of large traces. In a related approach, Xu and colleagues [10] have studied *trace logicalization*, where the underlying topology is explicitly extracted and used to formulate a unified trace. We feel that building an affine loop nest representing a parallel trace could help in highlighting topological properties of the communication scheme.

B. ScalaTrace

ScalaTrace [4], [5] is a system for deterministic compression and replay of parallel communication traces. With goals similar to ours, it adopts an approach which is almost perfectly orthogonal to our approach. ScalaTrace follows the *model-then-merge* (see Section IV): individual traces undergo loop recognition, and at the end of the run are sent to a centralized component which merges all models to produce a global trace, a kind of iterative multiple sequence alignment. Besides this high-level difference, ScalaTrace's *regular section descriptors* (RSD) and *power RSDs* are essentially nested loops in our constant model, i.e., without explicit affine function. ScalaTrace also has some knowledge of MPI primitives, and encodes message destinations as offsets from the source, which is essentially a restricted form of affine functions.

The major difference between the two systems is how they both attempt to build a trace that mixes events originating in different processes, following exact opposite approaches. ScalaTrace does this at merge-time, by interlacing terms from individual traces. Our system does this at trace-merging time, by interlacing individual events. In both cases, the problem is one of scheduling, with essentially the same challenge: finding a balanced, regular interlacing is difficult in the general case.

It is also interesting to note that NLR could be used as a drop-in replacement for the modeling of individual traces in ScalaTrace. This would bring the expressive power of affine functions, but would also complicate the merging process. Perhaps more surprising is the fact that NLR can also be used at the merging component. Let us consider the CG benchmark on 16 processors. Every individual trace follows the same pattern, e.g., for process number 0:

```
      val sync , MPI_Bcast
      for i0 = 0 to 31 do
T0,1 :    val 0 send 2 1
T0,2 :    val 2 recv 0 1
T0,3 :    val 0 send 1 2
T0,4 :    val 1 recv 0 2   ...
      done
```

Labels uniquely identify tuples coming from various traces. From the individual process loops one could apply ScalaTrace's interlacing process, which synchronizes on loop boundaries and collective operations, to obtain a merged trace like:

```
      val sync , MPI_Bcast
      for i0 = 0 to 31 do
T0,1 :    val 0 send 2 1
T1,1 :    val 1 send 3 1
          ...
T14,1 :   val 14 send 12 1
T15,1 :   val 15 send 13 1
T0,2 :    val 2 recv 0 1    ...
      done
```

Here, successive `val` lines come from successive individual process traces, in a simple round-robin manner.

The crucial point is that NLR can be applied again, recursively, on the interlaced sequence of tuples, and it would produce a loop nest whose outermost index iterates over the process number space. In our example, the result is:

```
val sync , MPI_Bcast
for i0 = 0 to 31 do
  for p = 0 to 3 do
    val (0+4*p) send (2+4*p) 1
    val (1+4*p) send (3+4*p) 1
    val (2+4*p) send (0+4*p) 1
    val (3+4*p) send (1+4*p) 1
  done ...
done
```

The inner loop index is named `p` to highlight the fact that this level of the loop represents the dimension of processes, as opposed to the loop on `i0` which represents the dimension of "time", i.e., event number. This does not mean `p` *is* the process

number, simply that the range of the affine functions on `p` is the set of processes.

III. Nested Loops Recognition

A. Basic Definitions

The NLR algorithm [3] manipulates sequences of *terms*. A term is a symbolic construction whose structure is governed by the following grammar:

Term → `val` *Function+*
| `for` *id* `= 0 to` *Bound* `do` *Term+* `done`

Let us assume for a moment that *Function* represents symbols taken in a finite alphabet Σ, and that *Bound* represents natural numbers. Then the simplest terms are formed by the keyword `val` followed by a vector of symbols. For instance, `val P0 send P1` is a simple term, called a *tuple* in the following, carrying three values with no specific semantics.

NLR forms loops that denote sequences of tuples. A loop is introduced with the keyword `for` and has a counter, whose name is taken in the set $\{\mathtt{i}_0, \mathtt{i}_1, \ldots\}$. All loops built by NLR are normalized, i.e., they iterate from zero to their upper bound (included) with step 1 (the notation `= 0 to` is nothing but syntactic sugar). The body of a loop is a non-empty sequence of terms: a loop can contain an arbitrary interleaving of sub-loops and tuples, down to an arbitrary depth. We will assume that a loop L_d contained inside d other loops uses a counter which is distinct from those used by its enclosing loops. Here is an example loop:

```
for i0 = 0 to 19 do
    val P0 send P1
    for i1 = 0 to 10 do
        val P1 send P2
        val P1 recv P2
    done
    val P0 recv P1
done
```

B. Loop Recognition Strategy

An input trace is made of a sequence of tuples. NLR will try to turn successive occurrences of identical groups of terms into loops. It works by detecting a repetition as soon as three identical occurrences of a group of terms appears (why it looks for three occurrences, rather than two or four, will be explained below). For instance, given the following input trace:

```
val P0 send P1
val P1 send P2
val P0 send P1
val P1 send P2
val P0 send P1
val P1 send P2
```

NLR will immediately replaces these 3 groups of 2 terms with:

```
for i0 = 0 to 2 do
    val P0 send P1
    val P1 send P2
done
```

It is important to remember that NLR seeks repetition of *terms*. Therefore, the following 6 terms:

```
for i1 = 0 to 9 do val P0 send P1 done
val P1 send P2
```

REDUCTION(S)
– – S is a stack of N terms $S[0] \ldots S[N-1]$
– – $S[i, j[$ denotes $S[i]S[i+1] \ldots S[j-1]$
for i = 1 to $\min(N, K)$ do
  if 3 divides i then
    let $d = \frac{i}{3}$
    if $S[N-3d, N-2d[\approx S[N-2d, N-d[\approx S[N-d, N[$, then
      let $t_i = S[N-3d+i]$, $\forall 0 \leq i < d$
      pop $3d$ terms from S
      push onto S the term
        `for` i_x`=0 to 2 do` $t_0, \ldots t_{d-1}$ `done`
      return true
  if $S[N-i-1]$ is a loop
      whose body is $S[N-i, N[$ then
    increment the bound of $S[N-i-1]$
    pop i terms from S
    return true
return false

Fig. 1. Reducing a stack of terms

```
for i1 = 0 to 9 do val P0 send P1 done
val P1 send P2
for i1 = 0 to 9 do val P0 send P1 done
val P1 send P2
```

are also turned into a depth-two loop nest.

The algorithm uses a second basic operation: prolonging a loop to cover terms forming its next iteration. Whenever a loop L is followed by a group of term which is identical to the body of L, this group of term is removed and the upper bound of L is incremented by 1.

Given an input trace made of tuples, the algorithm will try to repeatedly apply its two basic "reduction" operators (turning three groups into a loop, and extending a loop to cover its next iteration). NLR is designed to work incrementally: the input trace is read only once, and every input tuple is pushed onto a stack when first read. Every time the stack is changed, NLR examines an increasingly longer upper portion of the stack until it finds something to reduce. Reduction here means looking at the last N terms of the stack, and testing whether they form three successive groups, or testing whether the N^{th} term is a loop that can incorporate the $N-1$ terms above it. This test is done for values of N ranging from 1 to K, a user-settable parameter. The stack-reduction algorithm is summarized on Figure 1. Given this routine, the NLR algorithm simply pushes every incoming tuple on the stack, and repeatedly reduces its stack as long as possible.

C. Handling Numbers

If NLR were restricted to symbolic values, i.e., functions, it would be no more than a not-so-subtle repetition detection algorithm. Its major strength appears when one wants to include numeric data in traces, by making input tuples combine symbols and numbers, like the following:

```
val 0 send 1
val 0 send 3
val 0 send 5
```

By slightly relaxing the use of *term-equality*, NLR is able to handle numbers and produce nested loops where places that carried numeric data in input tuples are occupied by affine functions involving loop counters. It uses linear interpolation to add monomials to the functions at the time a new loop is build.

Whenever NLR examines a sequence of three groups of terms, it must perform two tests:

1) check whether the three groups of terms are isomorphic, i.e., they all three share exactly the same structures;
2) if the first test was successful, check for each and every numeric place whether the triple of functions found there can be interpolated into a new function.

For instance, the three terms shown in the previous paragraph are isomorphic (they are all made of a single tuple carrying a number, a symbol, and another number). Checking whether the symbols match is considered part of the isomorphism test. What remains is the set of numeric places (two, in this case). The first numeric place shows values `0`, `0`, and `0`, which can be interpolated as `0+0*i`. The second numeric place can be interpolated as `1+2*i`. This sequence of three terms can then be turned into the following loop:

```
for i = 0 to 2 do
    val 0+0*i send 1+2*i
done
```

In the same vein, if the next incoming tuple is `val 0 send 7` It is immediate to verify that the loop can be extended.

This linear interpolation, which is restricted to integers, is performed for all functions appearing in the three groups of terms. If it fails, no loop is build. When the three candidate groups of terms are made of nested loops, NLR has to interpolate complete affine functions that may involve counters of intermediate loops. For instance, the stack may at some point contain the following terms:

```
for i1 = 0 to 10 do
    ... val  0 + 3*i1 + 5*i2 ... done
for i1 = 0 to 15 do
    ... val 12 + 3*i1 + 5*i2 ... done
for i1 = 0 to 10 do
    ... val 24 + 3*i1 + 5*i2 ... done
```

Using our affine model, functions involving loop counters must match exactly on all monomials, and only the constant can be interpolated. In this example, the resulting loop is:

```
for i0 = 0 to 2 do
    for i1 = 0 to 10+5*i0 do
        ... val 0+12*i0 + 3*i1 + 5*i2 ...
    done
done
```

Loop bounds are no different than other functions, and are subjected to the same interpolation, i.e., a loop bound may be an affine function of the enclosing loop indexes. To avoid complications with name conflicts (due to the fact that a single term can contain sub-loops of differing depths), the counters are actually named after the depth at which they appear in a term, in a way very similar to De Bruijn indexes, i.e., the outer loop has counter i_0, first level-loops have i_1, etc.

```
for i0 = 0 to 31
  for i1 = 0 to 3
    for i2 = 0 to 3
      val 8*i1+i2 send 4+8*i1+i2
    done
    for i2 = 0 to 3
      val 4+8*i1+i2 send 8*i1+i2
    done
  done [...]
  for i1 = 0 to 3
    for i2 = 0 to 3
      val 2*i1+8*i2 recv 8*i1+2*i2
      val 1+2*i1+8*i2 recv 1+8*i1+2*i2
    done
  done [...]
done
```

Fig. 2. An example output from NLR on a CG parallel trace

Note that numbers can be treated like symbols, in which case there is no need for interpolation: we call this the *constant* model (we will use the constant model below, in Section V). What we just described is called the *affine* model. Other models could be used, e.g., degree-2 polynomials. However, in such cases, the algorithm must be adapted to provide enough groups of terms at loop-construction time, e.g., at least four terms for degree-2 polynomials. The reason why we use three groups of terms for the affine model is that the first two are needed to infer an affine function (and this works for any couple of numbers), and the third is here to check whether there actually is a non-trivial linearity.

Figure 2 contains an example of the output of NLR when run on a trace of the CG program from the NPB suite. The tuples in the input represented basic events like sending a message and receiving at the other end, with process numbers attached. It is interesting to note that NLR seems to have discovered interesting patterns (4*4 grids). Note that the loops that appear on the figure represent 1024 events. The next sections enter into the details of preparing such a trace for NLR, and using its results to help program understanding, or for trace compression.

Overall, the worst case complexity of the algorithm is $O(NK^2)$, were K is a parameter defining the maximum size of a loop body, and N the size of the input. In practice, the algorithm is very fast as long as its stack remains small, i.e., as long as stack reductions can happen frequently. More details are given in [3].

IV. Trace Linearization

Our goal is to apply the loop recognition algorithm to traces of MPI programs, or any distributed program made of a fixed number of processes that communicate by sending messages to each other. In addition, MPI programs synchronize through collective operations. Every process produces during its execution a sequential trace containing all its communication actions, along maybe local events describing the computations it performs. Tools devoted to the acquisition of traces are

commonly found, and the one we used in this work is TAU [11]. This section describes the trace format we use, and explains how we combine individual process traces into a single trace.

A. Trace Format

Since we are interested in discovering communication patterns in a distributed program execution trace, we will use traces containing the following types of events:

- *src* **send** *dst tag*, which indicates that process *src* has emitted towards *dst* a message labeled with *tag*;
- *src* **recv** *dst tag*, which happens when *dst* actually receives the message from *src*;
- **sync** *name*, signaling that all processes participate in a collective synchronization operation labeled with *name*.

Our trace format is directly inspired by the MPI model. Each message is sent with an associated tag, and similarly tagged messages between two processes are delivered in order. Regarding collective operations, we assume that, for a given execution, the traces are coherent, i.e., all processes perform the same collective operations in the same order. The name of the collective operations are purely decorative. Our trace format does not handle collective operations involving a subset of the processes: this could be added without much difficulty but is not considered here to simplify the explanations. In the same way, trace entries describing local events can be added trivially, but we do not present any application of local events in this paper. Note that the traces do not contain any timing information, since we focus on the number and order of events only. In the rest of this paper, the input data set is made of a collection of N sequential traces.

B. Merging Individual Traces

When modeling the behavior of a distributed system, there are two basic approaches:

1) *model-then-merge*, which consists in performing some processing on individual process traces, and later merge the individual results into a global model;
2) *merge-then-model*, which first merges the individual process traces, and then performs a global analysis of the result.

Both approaches are perfectly valid, and each of them presents specific difficulties. We have chosen the second solution: individual process traces are first merged into a global trace. NLR will run on this global trace, and produce loop nests. We will come back to the comparison of the two approaches below, in Section II.

Each individual trace contains the events that happened in a particular process. The merging procedure must ensure that the global trace remains coherent, with receives happening after the corresponding sends, and guarantee proper synchronization on collective operations. This amounts to simulate the execution of the system, and output a sequential scheduling of the various events. Any scheduling technique can be used, without restriction. For instance, one could maintain a queue of blocked processes, a list of messages in flight, and then handle events of non-blocked processes in some specific order. This last point is the most difficult one: any scheduling technique will face situations where arbitrary decisions have to be made. Making decisions in such cases introduces a bias in the merging process that may have an influence on the following modeling phase.

We have chosen a strategy that favors the diversity of event origins in the trace, rather than trying to keep events from the same process close together. Since our goal is to model the behavior of distributed programs, we expect to have extracts of the process states at all scales. This is achieved by alternating phases of sends and receives. During each phase, the maximal set of concurrent events of the corresponding type is emitted.

V. Program understanding

Our first application of loop recognition is the detection of regular behaviors in communications between processes, and the visualization of large-scale regular structure. This section describes the process that goes from an input trace to graphs like the ones shown below. The first step is to run NLR on a linearized trace. Since we target a visual rendering of the trace, we need to maintain explicit events, i.e., events with process numbers. This precludes the use of affine functions. In this section, we restrict NLR to use its constant model: no interpolation is performed on process numbers or tags. Theoretically, it could happen that loop bounds be generated as affine functions of the enclosing loop counters. However, we have found no case where this happened in our data set.

Here is an example of the result of NLR on a trace of the CG program from the NPB suite (class S). The main loop has the following shape:

```
for i0 = 0 to 13 [...]
  for i1 = 0 to 25 [...]
    for i2 = 0 to 2 [...] done
  done
done
```

One can see that repetitions can adopt a complex structure in some cases, even though time is the only dimension that is subject to iteration.

Reading a loop nest like the one shown in the previous paragraph is certainly not the best way to get an understanding of the overall communication scheme of a program. To guide analysis, it is however possible to produce a graphical rendering of the loop nests produced by NLR. We have developed a small tool that takes the output of NLR and produces time-lines, showing messages as arrows between the time-lines, and representing loops as boxes around the part of the trace they enclose. The body of a loop is represented only once, and labeled with a number of iterations. To visualize messages in a coherent way, the tool maintains a Lamport clock for each process. The result of NLR is traversed term after term. A **send** event gets a horizontal position equal to its clock. A **recv** is positioned in the same way, and leads to the drawing of an arc between its position and the position of the corresponding **send**. The start of a loop requires synchronization of all process clocks. Its body is then scanned and drawn. The end of the loop also synchronizes clocks and draws a box around the body. The loop structure given above is drawn like that:

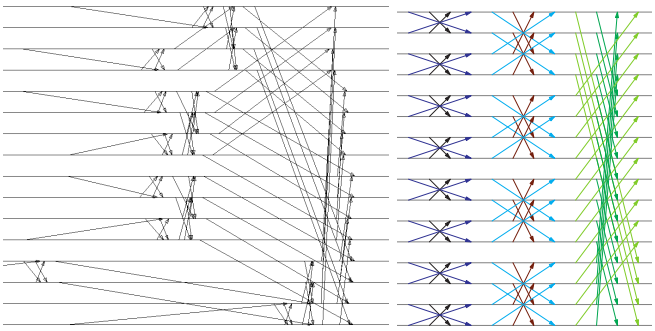
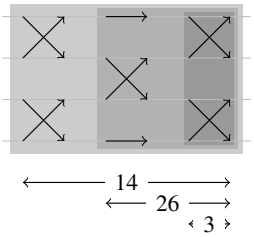


Fig. 3. One communication pattern in the MG benchmark 16 processors, as shown by Jumpshot (left) and NLR (right)



However, it is very difficult to draw messages as simple arrows in the general case. If there are several occurrences of identical **`send`** events, deciding on which is responsible for a specific **`recv`** event is a difficult question, that requires considering the enclosing loop bounds of all events involved. The same holds for a single **`recv`** event with multiple matching **`send`**s. We have found nothing simpler that using Ehrhart polynomials [12] to solve this problem. Fortunately, none of the programs we have traced has shown pathological behavior.

To illustrate how our visualization method differs from traditionnal tools, Figure 3 compares a communication pattern extracted from the MG benchmark, as displayed by Jumpshot [2] on the left, and as drawn by our tool on the right. We show further some sample outputs produced by our tool on Figures 4 to 7. These examples are LU, BT, SP, MG and CG from NPB [13]. EP, DT, FT, and IS have trivial traces because they contain mostly collectives, and are thus omitted. The apparent geometry of the communication patterns in the figures do correspond to the description of the benchmarks below.

- LU: The LU decomposition computes a block-lower and block-upper triangular approximate factorization of a finite difference discretization of the 3-D compressible Navier-Stokes equation.
- BT and SP: The block tridiagonal and scalar pentadiagonal benchmarks (resp.) solve multiple, independent systems of non-diagonally dominant, block tridiagonal and scalar pentadioganal equations (resp.). BT and SP mainly differ by their communication to computation ratio.
- MG: The Multigrid kernel is a multi-grid algorithm aimed at computing an approximate solution to a discrete Poisson problem on a 3-D grid with periodic boundary conditions. This benchmarks involves short and long distance regular communications.
- CG: The conjugate gradient method computes an approximation to the smallest eigenvalue of a symmetric positive definite sparse matrix. This kernel uses unstructured matrix vector multiplications, involving unstructured grid computations with irregular long distance communications.

Figures 8 and 9 illustrate that the abstract representation of the communication scheme simplifies visual comparisons. In this case, we have a straight overview of how the communications evolve when doubling the number of processes. We also present present the communication of sweep3d [14], [15], which is the kernel of a real application aimed at computing a neutron transport problem.

VI. Parallel Trace Compression

Applying NLR to a trace produces a sequence of loop nests. This helps producing a graphical rendering of communication patterns, with NLR voluntarily restricted to its *constant model*, where loops are built from repeating *identical* sequences of terms. However, NLR was originally designed to produce affine functions (see Figure 2). This results in a more expressive trace model, and much more compact representations of regular patterns. It also lets NLR build affine combinations of loop counters to represent the numerical parameters attached to events (process numbers and tags). This leads to shorter loops that can, for example, iterate over a range of process numbers (or even tag values). In previous work, we have shown that NLR was an excellent compression algorithm for memory address trace, providing lossless compression and deterministic replay. We are now going to evaluate its ability to compress parallel communication traces.

We have used the NPB programs BT, CG, LU, MG, and SP to evaluate the compression performance of NLR. EP, DT, FT, and IS, as in section V, are not evaluated because of their simple traces and corresponding loops. All programs have been run on input data of size C, with a number of processes of 4, 8, 16, 32, and 64 (when applicable). Input traces where kept as text files, as are the resulting loop nests. The compression rates are computed as ratio between the file sizes.

Results are shown in Table I. The table displays, for each benchmark at each of its available process number, the compression ratio along with the number of events. The BT program has an approximately constant compression rate: the explanation is that BT does not contain any affine loop, and is basically made of one single big time-based loop. Since the trace cannot be compressed along another dimension, trace size and compressed size increase linearly with the number of processes, and their ratio remains constant. CG is in the same situation for the same reason. SP has a similar communication pattern, but some loop bounds increase with process number and therefore cover more events for higher number of processes. LU shows increasing compression ratios because it is, basically, a constant size compressed form. MG sees a dramatic drop in compression ratio when run with more than 8 processes, because at that size some processes remain inactive for some

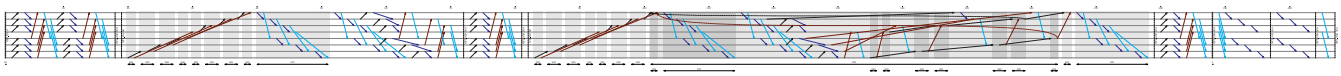


Fig. 4. LU class C on 8 processors (complete execution)

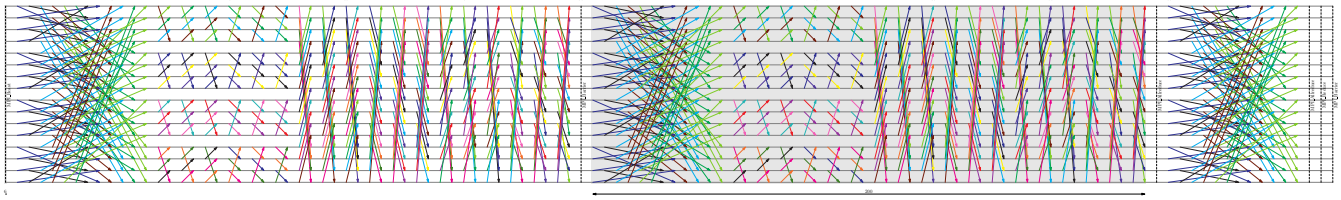


Fig. 5. BT class C on 16 processors (complete execution)

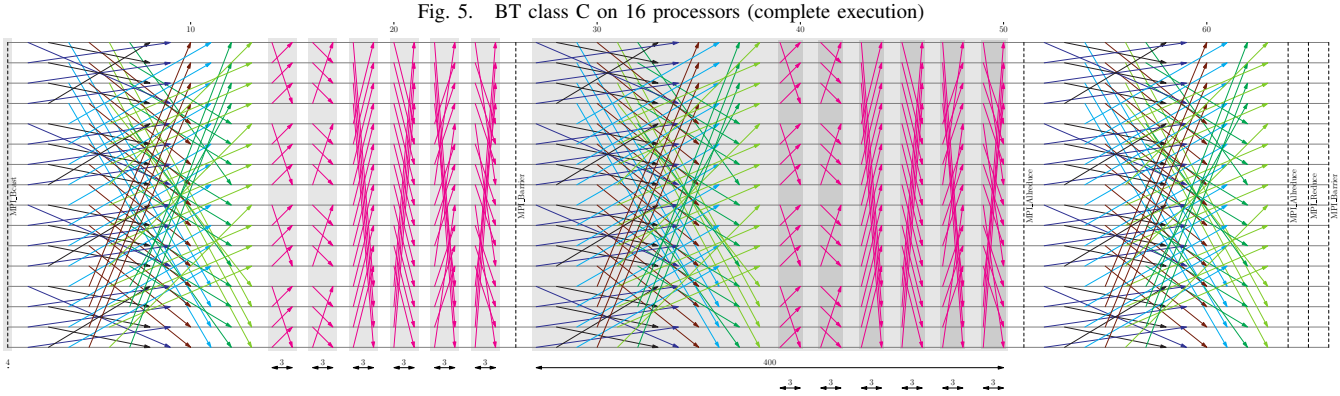


Fig. 6. SP class C on 16 processors (complete execution)

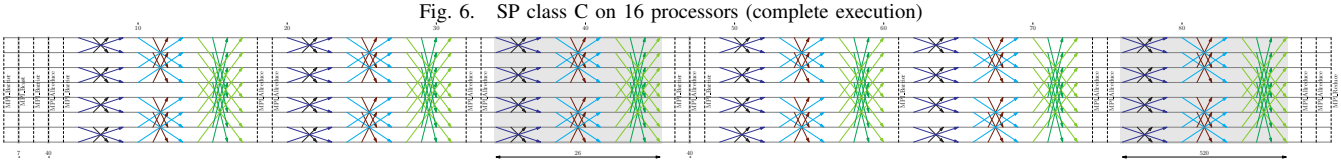


Fig. 7. MG class C on 8 processors (complete execution)

Program	4 proc.		8 proc.		16 proc.			32 proc.		64 proc.	
	Events	Ratio	Events	Ratio	Events	Ratio	*Constant*	Events	Ratio	Events	Ratio
bt	19355	47.85	-	-	154571	55.00	*55.00*	-	-	1235723	58.03
cg	63843	249.04	223747	238.78	447491	249.79	*78.43*	1279235	250.23	-	-
lu	646662	1299.73	1616618	1494.11	3879838	1763.32	*1495.29*	8406278	3947.23	17019867	8906.49
mg	26502	52.36	52902	53.16	102342	20.66	*10.60*	201894	21.68	402342	21.90
sp	38553	91.75	-	-	308169	187.70	*142.65*	-	-	2464521	378.30

TABLE I
COMPRESSION RATIOS (NPB, SIZE C)

periods while others perform intense communication with neighbors: this has an impact on the efficacy of our trace-merging strategy. We have no space to describe the techniques we have designed to overcome this problem: in the meantime this sensitivity will remain a defect of our approach. In all cases, the processing time was never above one minute. We consider this is efficient enough not to require a detailed listing. As noted above, the results shown in Table I have been obtained by running NLR with the affine model. An interesting question is: how much gain in compression does the affine model provide, compared to the constant model. The table also shows the compression ratio achieved by using the constant model with 16 processes: in all cases the ratio is dramatically lower than in the affine case. The expressive power of affine functions definitely increases compression performance.

VII. CONCLUSION

This paper has described the application of a sequential affine loop recognition technique to communication traces of parallel programs. The NLR algorithm is an efficient, incremental loop recognition engine, that is fed with the results of merging all individual traces into a unique, global trace containing all communication events performed by all processes. This produces a sequence of loop nests that, when run in the obvious way, reproduce all message exchanges and collective operations.

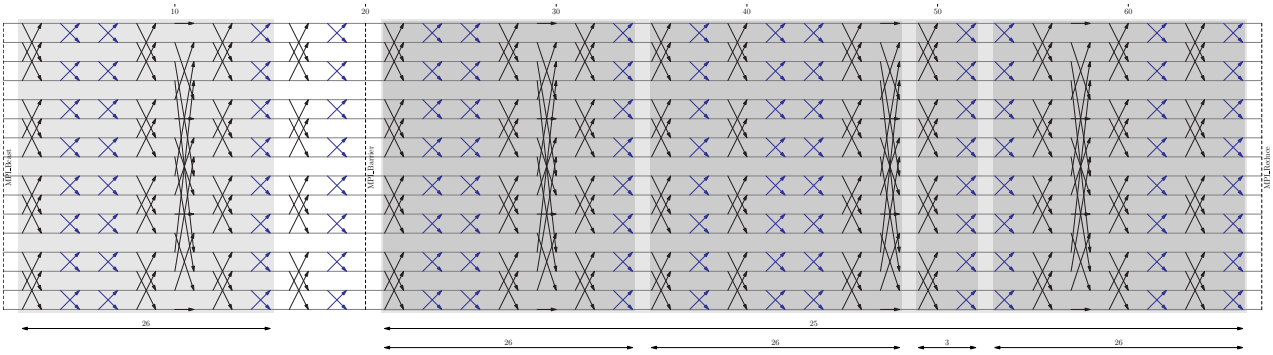


Fig. 8. CG class C on 16 processors (complete execution)

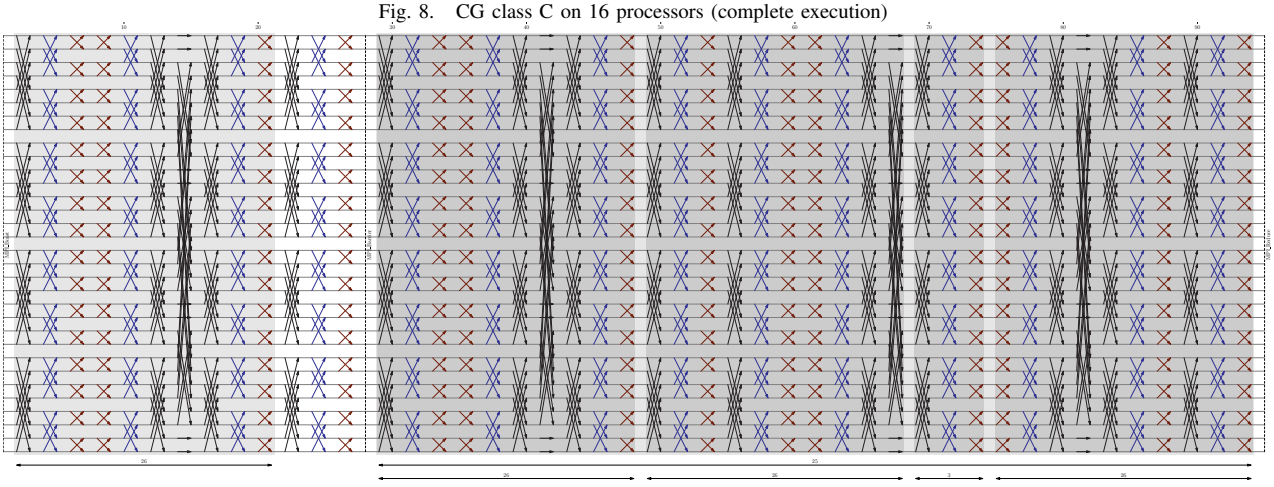


Fig. 9. CG class C on 32 processors (complete execution)

We have described two main applications of the technique. The first is the visualization of short, abstract communication patterns. The second is the compression of the initial trace.

There are several ways in which this work can be extended. As we have seen, using affine functions instead of symbols leads to shorter compressed traces, and provides an analytical model of the program. However, Figure 6 clearly shows that some regular patterns are not yet captured. The typical "walls" of messages that appear on this figure cannot be represented by a linear function. We plan on extending our technique to use modular arithmetic, because this appears to be a major element in characterizing communication patterns. The second research direction is about the predictive ability of loop nests. Earlier work of ours [3] has shown that NLR produces loop nests that can predict future behavior at a long distance. Since loops are formed early, expecting more iterations of the same loop is a reasonable guess. Exploring the potential application of this predictive power is one of our short-term research goal.

REFERENCES

[1] J. Gonzalez, J. Gimenez, and J. Labarta, "Automatic detection of parallel applications computation phases," in *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy*, May 2009, pp. 1–11.

[2] C.-F. E. Wu, A. Bolmarcich, M. Snir, D. Wootton, F. Parpia, A. Chan, E. L. Lusk, and W. Gropp, "From trace generation to visualization: A performance framework for distributed parallel systems," in *Proc. of SC2000: High Performance Networking and Computing*, Nov. 2000.

[3] A. Ketterlin and P. Clauss, "Prediction and trace compression of data access addresses through nested loop recognition," in *6th International Symposium on Code Generation and Optimization (CGO 2008), Boston, MA, USA.*, M. L. Soffa and E. Duesterwald, Eds., Apr. 2008, pp. 94–103.

[4] M. Noeth, F. Mueller, M. Schulz, and B. R. de Supinski, "Scalable compression and replay of communication traces in massively parallel environments," in *21th International Parallel and Distributed Processing Symposium (IPDPS07)*. IEEE, Mar. 2007, pp. 1–11.

[5] M. Noeth, P. Ratn, F. Mueller, M. Schulz, and B. R. de Supinski, "Scalatrace: Scalable compression and replay of communication traces for high-performance computing," *J. Parallel Distrib. Comput.*, vol. 69, no. 8, pp. 696–710, 2009.

[6] S. Krishnamoorthy and K. Agarwal, "Scalable communication trace compression," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 408–417, 2010.

[7] C. G. Nevill-Manning and I. H. Witten, "Identifying hierarchical structure in sequences: A linear-time algorithm," *Journal of Artificial Intelligence Research*, vol. 7, no. 1, pp. 67–82, 1997.

[8] Q. Xu, J. Subhlok, and N. Hammen, "Efficient discovery of loop nests in execution traces," in *MASCOTS 2010, 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Miami, Florida, USA*, Aug. 2010, pp. 193–202.

[9] M. Crochemore, "An optimal algorithm for computing the repetitions in a word," *Inf. Process. Lett.*, vol. 12, no. 5, pp. 244–250, 1981.

[10] Q. Xu, J. Subhlok, R. Zheng, and S. Voss, "Logicalization of communication traces from parallel execution," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC 2009), Austin, TX, USA*. IEEE, Oct. 2009, pp. 34–43.

[11] S. Shende and A. D. Malony, "The tau parallel performance system,"

International Journal of High Performance Computing Applications, vol. 20, no. 2, pp. 287–311, 2006.

[12] P. Clauss, "Counting solutions to linear and nonlinear constraints through Ehrhart polynomials: Applications to analyse and transform scientific programs," in *Proceedings of the 10th International Conference on Supercomputing (ICS 96)*, May 1996.

[13] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The NAS Paralell Benchmarks," NASA Ames Research Center, Tech. Rep. RNR-94-007, Mar. 1994.

[14] "Asci sweep3d v2.2b: 3-dimensional discrete ordinates neutron transport benchmark," Los Alamos National Laboratory, Tech. Rep., Dec. 1995. [Online]. Available: http://wwwc3.lanl.gov/pal/software/sweep3d/

[15] B. J. N. Wylie, M. Geimer, B. Mohr, D. Böhme, Z. Szebenyi, and F. Wolf, "Large-scale performance analysis of sweep3d with the scalasca toolset," *Parallel Processing Letters*, vol. 20, no. 4, pp. 397–414, 2010.

Bibliographie

[1] A. Arakwa. Computational design for long-term numerical integration of the equations of fluid motion : Two-dimensional incompressible flow. part i. *Journal of Computational Physics*, 1(1) :119 – 143, 1966.

[2] P. J. Basser and D. K. Jones. Diffusion-tensor mri : theory, experimental design and data analysis–a technical review. *NMR in Biomedicine*, 15(7-8) :456–467, 2002.

[3] M. Bennoune, M. Lemou, and L. Mieussens. Uniformly stable numerical schemes for the boltzmann equation preserving the compressible navier–stokes asymptotics. *Journal of Computational Physics*, 227(8) :3781 – 3803, 2008.

[4] B. Berkowitz. Characterizing flow and transport in fractured geological media : A review. *Advances in water resources*, 25(8) :861–884, 2002.

[5] P. Beyer. *Turbulence et transport dans les plasmas chauds magnétisés*. Habilitation à diriger des recherches, Université de Provence, 2004.

[6] P. Beyer, S. Benkadda, G. Fuhr-Chaudier, X. Garbet, P. Ghendrih, and Y. Sarazin. Nonlinear dynamics of transport barrier relaxations in tokamak edge plasmas. *Phys. Rev. Lett.*, 94 :105001, Mar 2005.

[7] O. A. R. Board. Openmp application program interface, version 3.1. 2011.

[8] S. I. Braginskii. *Transport Processes in Plasma*.

[9] L. Chacón, D. Del-Castillo-Negrete, and C. D. Hauck. An asymptotic-preserving semi-lagrangian algorithm for the time-dependent anisotropic heat transport equation. *Journal of Computational Physics*, 272(0) :719 – 746, 2014.

[10] F. Chen. *Introduction to Plasma Physics and Controlled Fusion*. Number vol. 1 in Introduction to Plasma Physics and Controlled Fusion. Springer, 1984.

[11] N. Crouseilles, M. Lemou, et al. An asymptotic preserving scheme based on a micro-macro decomposition for collisional vlasov equations : diffusion and high-field scaling limits. *Kinetic and related models*, 4(2) :441–477, 2011.

[12] M. Crouzeix. Une méthode multipas implicite-explicite pour l'approximation des équations d'évolution paraboliques. *Numerische Mathematik*, 35(3) :257–276, 1980.

[13] P. A. Crouzeix, M. Raviart. Conforming and nonconforming finite element methods for solving the stationary stokes equations i. *ESAIM : Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 7(R3) :33–75, 1973.

[14] J. L. Delcroix. *Physique des plasmas*. Number vol. 2 in Monographies Dunod. Dunod, 1966.

[15] A. Ern. *Aide-mémoire des éléments finis*. Dunod, 2005.

[16] D. F. ESCANDE. Plasma thermonucléaire confiné magnétiquement : un système complexe. *Images de la physique*, pages 39–44, 2005.

[17] R. Eymard, T. Gallouët, and R. Herbin. Finite volume methods. *Handbook of numerical analysis*, 7 :713–1018, 2000.

[18] F. Filbet and S. Jin. A class of asymptotic-preserving schemes for kinetic equations and related problems with stiff sources. *Journal of Computational Physics*, 229(20) :7625–7648, 2010.

[19] F. Filbet, C. Negulescu, and C. Yang. Numerical study of a nonlinear heat equation for plasma physics. *International Journal of Computer Mathematics*, 89(8) :1060–1082, 2012.

[20] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Computers*, C-21(9) :948–960, September 1972.

[21] M. P. Forum. Mpi : A message-passing interface standard. Technical report, Knoxville, TN, USA, 1994.

[22] M. Frigo and S. Johnson. The design and implementation of fftw3. *Proceedings of the IEEE*, 93(2) :216–231, 2005.

[23] G. Fuhr, P. Beyer, S. Benkadda, and X. Garbet. Evidence from numerical simulations of transport-barrier relaxations in tokamak edge plasmas in the presence of electromagnetic fluctuations. *Phys. Rev. Lett.*, 101 :195001, Nov 2008.

[24] G. Fuhr-Chaudier. *Effet d'une perturbation électromagnétique sur la turbulence et le transport dans les plasmas chauds magnétisés*. PhD thesis, Université de Provence, 2006.

[25] S. Günter, K. Lackner, and C. Tichmann. Finite element and higher order difference formulations for modelling heat transport in magnetised plasmas. *Journal of Computational Physics*, 226(2) :2306–2316, 2007.

[26] S. Günter, Q. Yu, J. Krüger, and K. Lackner. Modelling of heat transport in magnetised plasmas using non-aligned coordinates. *Journal of Computational Physics*, 209(1) :354–370, 2005.

[27] T. Görler, X. Lapillonne, S. Brunner, T. Dannert, F. Jenko, F. Merz, and D. Told. The global version of the gyrokinetic turbulence code {GENE}. *Journal of Computational Physics*, 230(18) :7053 – 7071, 2011.

[28] E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration : Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer series in computational mathematics. Springer, 2002.

[29] E. Hairer, C. Lubich, and G. Wanner. Geometric numerical integration illustrated by the störmer–verlet method. *Acta Numerica*, 12 :399–450, 2003.

[30] E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration : structure-preserving algorithms for ordinary differential equations*, volume 31. Springer, 2006.

[31] A. Harms. *Principles of Fusion Energy*. Sunil Sachdev, 2002.

[32] R. Hazeltine and J. Meiss. *Plasma Confinement*. Dover Books on Physics Series. Dover Publications, 2003.

[33] C. A. Kennedy and M. H. Carpenter. Additive runge-kutta schemes for convection-diffusion-reaction equations. 2001.

[34] M. Lemou and L. Mieussens. Implicit schemes for the fokker–planck–landau equation. *SIAM Journal on Scientific Computing*, 27(3) :809–830, 2005.

[35] M. Lemou and L. Mieussens. A new asymptotic preserving scheme based on micro-macro formulation for linear kinetic equations in the diffusion limit. *SIAM Journal on Scientific Computing*, 31(1) :334–368, 2008.

[36] R. J. LeVeque. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

[37] H. Liu and J. Zou. Some new additive runge–kutta methods and their applications. *Journal of Computational and Applied Mathematics*, 190(1) :74–98, 2006.

[38] A. Lozinski, J. Narski, and C. Negulescu. Highly anisotropic temperature balance equation and its asymptotic-preserving resolution. *arXiv preprint arXiv :1203.6739*, 2012.

[39] C. Lubich and A. Ostermann. Linearly implicit time discretization of non-linear parabolic equations. *IMA journal of numerical analysis*, 15(4) :555–583, 1995.

[40] J. MacCalpin. The stream benchmark page. *http ://www.cs.virginia.edu/stream/*.

[41] S. Maeyama, T. Watanabe, Y. Idmura, M. Nakata, M. Nunami, and A. Ishizawa. Computation-communication overlap techniques for parallel spectral calculations in gyrokinetic vlasov simulations. 8(1403150), 2013.

[42] P. McCorquodale and P. Colella. A high-order finite-volume method for conservation laws on locally refined grids. *Communications in Applied Mathematics and Computational Science*, 6(1) :1–25, 2011.

[43] A. Mentrelli and C. Negulescu. Asymptotic-preserving scheme for highly anisotropic non-linear diffusion equations. *J. Comput. Phys.*, 231(24) :8229–8245, Oct. 2012.

[44] A. Monnier, G. Fuhr, P. Beyer, S. Benkadda, and X. Garbet. Penetration of resonant magnetic perturbations at the tokamak edge. In *38th EPS Conference on Plasma Physics*, 2011.

[45] P. Mucci, S. Browne, C. Deane, and G. Ho. Papi : A portable interface to hardware performance counters. In *Proc. Department of Defense HPCMP Users Group Conference*, 1999.

[46] M. Muraglia, O. Agullo, S. Benkadda, X. Garbet, P. Beyer, and A. Sen. Nonlinear dynamics of magnetic islands imbedded in small-scale turbulence. *Phys. Rev. Lett.*, 103 :145001, Sep 2009.

[47] J. Narski and M. Ottaviani. Asymptotic preserving scheme for strongly anisotropic parabolic equations for arbitrary anisotropy direction. *arXiv preprint arXiv :1303.5219*, 2013.

[48] J. M. Nordbotten, I. Aavatsmark, and G. T. Eigestad. Monotonicity of control volume methods. *Numer. Math.*, 106(2) :255–288, Mar. 2007.

[49] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(7) :629–639, July 1990.

[50] D. Post. Iter : Physics basis. In *Plasma Science, 1990. IEEE Conference Record-Abstracts., 1990 IEEE International Conference on*, page 109. IEEE, 1990.

[51] J. M. Rax. *Physique des plasmas - Cours et applications : Cours et applications*. Physique. Dunod, 2005.

[52] P. Sharma and G. W. Hammett. Preserving monotonicity in anisotropic diffusion. *Journal of Computational Physics*, 227(1) :123 – 142, 2007.

[53] P. Sharma and G. W. Hammett. A fast semi-implicit method for anisotropic diffusion. *J. Comput. Phys.*, 230(12) :4899–4909, June 2011.

[54] E. Sonnendrücker. Approximation numérique des équations de vlasov-maxwell. Notes de cours de Master, Université de Strasbourg, 2010.

[55] P. Tamain, P. Ghendrih, E. Tsitrone, V. Grandgirard, X. Garbet, Y. Sarazin, E. Serre, G. Ciraolo, and G. Chiavassa. Tokam-3d : A 3d fluid code for transport and turbulence in the edge plasma of tokamaks. *Journal of Computational Physics*, 229(2) :361 – 378, 2010.

[56] B. Van Es, B. Koren, and H. J. De Blank. Finite-difference schemes for anisotropic diffusion. *Journal of Computational Physics*, 272(0) :526 – 549, 2014.

[57] Q. Zhang, H. Johansen, and P. Colella. A fourth-order accurate finite-volume method with structured adaptive mesh refinement for solving the advection-diffusion equation. *SIAM J. Sci. Comput.*, 34(2) :179–201, Apr. 2012.

Matthieu Kuhn

Calcul parallèle et méthodes numériques pour la simulation de plasmas de bords

Résumé

L'amélioration du code Emedge3D (code de bord électromagnétique) est abordée sous plusieurs axes. Premier axe, des innovations sur les méthodes numériques ont été mises en oeuvre. L'avantage des méthodes de type semi-implicite est décrit, leur stabilité inconditionnelle permet l'augmentation du pas de temps, et donc la diminution du nombre d'itérations temporelles requises pour une simulation. Les avantages de la montée en ordre en espace et en temps sont détaillés. Deuxième axe, des réponses sont proposées pour la parallélisation du code. Le cadre de cette étude est proche du problème général d'advection-diffusion non linéaire. Les parties coûteuses ont tout d'abord été optimisées séquentiellement puis fait l'objet d'une parallélisation OpenMP. Pour la partie du code la plus sensible aux contraintes de bande passante mémoire, une solution parallèle MPI sur machine à mémoire distribuée est décrite et analysée. Une bonne extensibilité est observée jusque 384 cœurs. Cette thèse s'inscrit dans le projet interdisciplinaire ANR E2T2 (CEA/IRFM, Université Aix-Marseille/PIIM, Université Strasbourg/Icube).

Abstract

The main goal of this work is to significantly reduce the computational cost of the scientific application Emedge3D, simulating the edge of tokamaks. Improvements to this code are made on two axes. First, innovations on numerical methods have been implemented. The advantage of semi-implicit time schemes are described. Their inconditional stability allows to consider larger timestep values, and hence to lower the number of temporal iteration required for a simulation. The benefits of a high order (time and space) are also presented. Second, solutions to the parallelization of the code are proposed. This study addresses the more general non linear advection-diffusion problem. The hot spots of the application have been sequentially optimized and parallelized with OpenMP. Then, a hybrid MPI OpenMP parallel algorithm for the memory bound part of the code is described and analyzed. Good scalings are observed up to 384 cores. This Ph. D. thesis is part of the interdisciplinary project ANR E2T2 (CEA/IRFM, University of Aix-Marseille/PIIM, University of Strasbourg/ICube).