Université de Toulouse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Présentée et soutenue par :
Jonathan Claustre
le lundi 17 décembre 2012

Titre :

Modèle particulaire 2D et 3D sur GPU pour plasma froid magnétisé: Application à un filtre magnétique

---

École doctorale et discipline ou spécialité :
ED GEET : Ingénierie des PLASMAS

Unité de recherche :
U.M.R CNRS - 5213

Directeur(s) de Thèse :

Jean Pierre BOEUF, Directeur CNRS LAPLACE - UPS
Mathias PAULIN, Professeur IRIT - UPS

Jury :
M. Minea TIBERIU, Professeur Université Paris XI Orsay (Rapporteur)

M. Xavier GRANIER, Chargé de recherche INRIA Bordeaux (Rapporteur)

Mme Claudia NEGULESCU, Professeur IMT - UPS (Examinateur)

M. Stanimir KOLEV, Professeur Faculty of physics, Sofia University (Examinateur)

Abstract

The PIC MCC method is a very powerful tool to study plasmas (we focus here on low temperature plasmas) since it can provide the space and time evolution of the charged particle velocity distribution functions under the effect of self-consistent fields and collisions. In an electrostatic problem, the method consists of following the trajectories of a representative number of charged particles, electrons and ions, in phase space, and to describe the collective interaction of the particles by solving Poisson's equation as the particles move. In a low temperature plasma, the trajectories in phase space are determined by the self-consistent electric field and by collisions with neutral atoms or molecules and, for large enough plasma densities, by collisions between charged particles. The computational cost of the method is very high in terms of CPU and memory resources, especially when multidimensional conditions must be taken into account and when steady state regimes are studied. This is because of the constraints (at least in explicit PIC simulations) on the time step (smaller than a fraction of the plasma period and inverse of the electron gyro frequency), on the grid spacing (on the order of the Debye length), and on the number of particles per Debye length in the simulation (larger than a few tens). The PIC MCC algorithm can be parallelized on CPU clusters (the treatment of particle trajectories is easy to parallelize, but the parallelization of Poisson's equation is less straightforward).

The emergence of GPGPU (General Purpose computing on Graphics Processing Unit) in scientific computing opens the way to low cost massively parallel simulations by using the large number of processors of a graphics card to perform elementary calculations (e.g. computation of electron trajectories) in parallel. A number of numerical tools for GPU computing have been developed in the last 10 years. Furthermore, NVIDIA developed a programming environment called CUDA (Compute Unified Device Architecture, [1]) that allows to create efficient GPU codes. PIC modeling using GPU or a combination of GPU and CPU has been reported by several authors, however PIC models with Monte Carlo Collisions (MCC) on GPU is an expanding area. To the best of our knowledge this work first reports results using a full GPU based implementation of 2D PIC-MCC model focussed on low temperature magnetized plasma. Tracking of particles in PIC simulations involving no creation or loss of charged particles (e.g. periodic boundary conditions, no ionization)

is straightforward. However, we need special reordering strategy when charged particle creation or loss is taken into account (e.g. ionization, absorption, attachment etc.).

This thesis highlights the strategies which can be used in GPU PIC-MCC models to overcome the difficulties with particle reordering during particle creation and loss. The aim of this work is to propose PIC MCC algorithms to be implemented on GPUs, to measure the efficiency of these algorithms (parallelization) and compare them with calculations on a single CPU, and to illustrate the method with an example of plasma simulation in a low temperature magnetized plasma. Our purpose is to describe the detailed features of the CUDA code that has been developed and to give an overview of the possibilities and constraints of programming a PIC MCC algorithm on a GPU, and to provide an estimate of the gain in computation time that can be obtained with respect to a standard CPU simulation. The discussion is focused on 2D simulations. The method we have developed has however already been implemented for 3D problems.

The manuscript is organized as follows. Chapter I gives a state of art of CPU and GPU architectures and an overview of GPU computing and of the CUDA environment. The basic principles of PIC MCC simulations are presented in chapter II. Our implementation of the PIC MCC algorithms (particle position updating, charge density assignment, Poisson solver, field interpolation, Monte Carlo collisions, generation of Maxwellian distributions of particles) is described also in this chapter. Chapter III presents simulation results for the example of a low temperature magnetized plasma under conditions similar to those of a negative ion source for neutral beam injection in fusion plasmas.

We discuss in the chapter II the computation times of different parts of the simulation and the total computation time as a function of parameters such as the number of particles or the number of grid cells. In the Chapter III, we discuss about the physics of a magnetic filter for the negative ion sources and a theoretical analysis of the electronic transport through the magnetic barrier is shown.

Finally, 3D simulations are used to compare results with 2D simulations, but a more detailled analysis still have to be done!

Remerciements

Avant d'aller plus en anvant dans le sujet de cette thèse, j'aimerai profiter des quelques lignes qui vont suivre pour remercier les personnes que j'ai pu rencontrer et cotoyer pendant ces trois années de thèse à Toulouse et sans qui ma vie toulousaine n'aurait pas été aussi parfaite !

Il va de soit que je ne citerai pas toutes ces personnes sans exception (au risque d'oublier certaines personnes) mais qu'un remerciement un peu plus général semble être plus indiqué !

Dans un premier temps, un grand merci à Jean Pierre Boeuf d'avoir fait le nécessaire pour me donner la chance de faire cette thèse au sein du groupe et de m'avoir encadrer et guider tout au long de celle-ci. C'était un grand plaisir pour moi de t'avoir comme directeur et un plaisir encore plus grand de t'écouter parler des plasmas qui semblent si simple et si passionnant après chaque entrevue !

Merci à Mathias Paulin de m'avoir intégré au sein de Vortex et de m'avoir montré un peu à quoi ressemble le rendu graphique. Merci également à Olivier, Antony, Mathieu, Samir, philippe, Marie, Mathilde et Monia pour avoir été à mes petits soins lorsque j'ai eu besoin d'un petit coup de pouce en programmation et pour ces royales sorties du CGE (Cassoulet Great Event) !

Un grand merci à Laurent Garrigues pour m'avoir aiguillé à mes débuts dans le monde des plasmas ainsi que dans le groupe GREPHE et merci de m'avoir donné cette chance de venir à Toulouse !

Un grand merci à tout le groupe GREPHE, Leanne Pitchford, Gerjan Hagelaar, Freddy Gaboriau, Thierry Callegari, Laurent Liard et Kremena Makasheva pour ne cité qu'eux et à Gwenael Fubiani pour sa disponibilité et ses discussions toujours très intéressantes même lorsqu'il ne s'agit pas de plasma !

Merci à tous les thésards du LAPLACE pour leur dynamisme, les diners du monde, pour toutes ces sorties enrichissantes qui ont su agrémenter ces trois années. Merci à mes collègues et amis Nicolas, Nordine, Guoqiang, Benoît, Benjamin, Pierre, Amine, Elisa, Juslan, Mohamed, Mustapha, Karina, José, Raja, Christopher, Tommy, Romain, Romain (bis), Marc, Estelle, F-X, Shérif, Agnès et Jean. Un gros gros merci à Yu et Philippe, mes camarades de bureau et aujourd'hui bien plus, avec qui j'ai pu partager et apprendre

beaucoup et sans qui cette thèse n'aurait pas été pareil. J'en garderai un excellent souvenir, de la bonne humeur, de la motivation et des heures passées en mer (mobil comme immobile d'ailleurs !). Surtout ne changez rien !

Merci à Bhaskar Chaudhury et Stanimir Kolev pour leur aide, leur gentillesse et surtout cette zénitude. C'est vraiment un régal de travailler avec vous !

Merci également aux membres du jury, Tiberiu Minea, Xavier Granier, Claudia Negulescu et Stanimir Kolev, d'avoir participer à la finalisation de ce projet et d'avoir accepter d'examiner ce travail.

Un avant-dernier remerciement à mes colocs Tatiana, Camille, Ben et Aurore (et le tout dernier arrivant, Siho). Merci d'avoir apporté chaque jour son lot de bonne humeur et de "bonnes énergies", de m'avoir si bien enseigné l'art de la tizane, de l'alimentation biologique et sans gluten (bon courage Ben !), merci encore pour tous ces moments avec vous que je ne peux résumé en quelques lignes.

Enfin, ces dernières lignes, je les dédie à toute ma famille que je n'ai que trop peu vu pendant ces trois dernières années mais qui est restée présente et m'a accompagné dans mes pensées et qui m'a donné la force d'aller jusqu'au bout.

Je souhaîte à tous ceux qui voudrait faire une thèse, de pouvoir la vivre et d'avoir autant de plaisir que moi j'en ai eu.

Table des matières

Table des figures

Introduction

To see a world in a grain of sand
And a heaven in a wild flower,
Hold infinity in the palm of your hand
And eternity in an hour.
William Blake, Auguries of Innocence

Le plasma, désigné comme le quatrième état de la matière est par sa nature présent dans de nombreux phénomènes physiques naturels tels que les étoiles, les aurores boréales ou encore les éclairs et constitue à 99% l'espace interstellaire (observable) [2]. Le plasma est également présent dans de nombreuses applications industrielles qui sont orientées vers la production d'énergie (appelé plasma chaud), les applications dans le domaine de la métallurgie (soudure, découpe, projection thermique, ...) (appelé plasma thermique) ou encore dans le domaine de la micro électronique, du traitement de surface, de l'éclairage, ... (appelé plasma froid) [1].

La nature relativement complexe de ces gaz ionisés donne une place très importante à la simulation et au développement de modèles permettant la compréhension du transport des particules et de leurs interactions avec les champs électromagnétiques et joue ainsi un rôle essentiel dans le développement de la théorie.

La méthode Particle-In-Cell (PIC) est une description cinétique du plasma. Les premières simulations PIC ont été réalisées dans les années 1960 par Buneman [3, 4] et Dawson [5] et elles simulaient le mouvement de 100 à 1000 particules et leurs interactions. Aujourd'hui, les codes PIC sont capables de simuler 10^5 à 10^{10} particules et représentent un outil très performant pour l'étude cinétique des plasmas. Ces codes sont utilisés dans la plupart des domaines de la physique des plasmas, i.e. des plasmas de "laboratoires" aux plasmas "spatiaux". De plus, les codes particulaires permettent de réduire au minimum le nombre de suppositions pour la description des modèles physiques. Cependant, cet avantage coûte très cher en temps de calcul et certaines simulations peuvent durer plus de 10^4 heures

1. Des détails supplémentaires sur les différentes applications dans le domaine des plasmas sont donnés sur le site : http ://www.plasmas.org/

sur les CPU actuels. C'est pourquoi, les codes PIC demandent un niveau d'optimisation important et sont essentiellement conçus pour des applications spécifiques.

C'est dans le cadre de l'optimisation du temps de calcul des codes PIC, appliqués à la simulation de plasmas froids magnétisés que les études ont été réalisées au cours de cette thèse.

De plus, le développement du GPGPU (General Purpose on GPU), permettant l'utilisation de la carte graphique pour du calcul parallèle intensif et permettant également de décharger le CPU de ces calculs, fournit aujourd'hui de nombreux outils numériques et permet d'utiliser de manière plus accessible, le GPU. De même, NVidia a développé un environnement de programmation appelé CUDA (pour "Compute Unified Device Architecture"), ouvrant à un très large public le calcul intensif sur GPU.

Aux vues des bénéfices propres à la carte graphique, il a été décidé d'utiliser ces nouvelles architectures de calcul pour l'implémentation du code PIC Monte Carlo Collision (MCC) 2D et 3D.

Le premier chapitre de cette thèse est consacré à la description de l'architecture des GPU et de l'environnement de programmation CUDA. Cependant, afin d'avoir une approche simple de ces architectures et de mieux comprendre leurs évolutions, la première partie fait une rétrospective des microprocesseurs (CPU), de leurs fonctionnement et des limitations actuelles notamment duent aux limites technologiques et physiques.

Le deuxième chapitre décrit en détail les différents modules du code particulaire et met en avant les avantages et les désavantages de la méthode PIC et de sa parallélisation sur le GPU. L'intégralité du code PIC MCC 2D-3D présenté dans ce chapitre a été développé dans le but d'effectuer l'ensemble des calculs sur le GPU. De plus, celui-ci a été optimisé afin d'obtenir des performances de calcul aussi hautes que possible. Une comparaison des temps de calcul a ainsi été réalisée pour un cas de simulation identique sur CPU et GPU.

Enfin, le dernier chapitre présente une étude sur le transport des particules chargées dans le plasma à travers un filtre magnétique. Les résultats présentés sont tous issus des simulations du code PIC MCC présenté dans le chapitre précédent. Cette étude est réalisée dans le cadre de la modélisation de la source d'ions négatifs de l'injecteur de neutres pour le projet ITER. Une étude dans une géométrie bidimensionnelle et tridimensionnelle est présentée et montre l'importance des parois et des champs associés sur le transports électronique.

Chapitre 1

Généralités sur les nouvelles architectures de processeurs

1.1 Bref historique des microprocesseurs

La cristallisation du silicium pure (à 99.9%) permet la fabrication de barreaux de silicium à partir desquels sont découpés des tranches extrêmement fines ($\sim$ 7 millimètres), puis polies, appelées *galettes* ou *wafer*. Sur ces tranches de silicium, on grave la matrice du microprocesseur, puis l'ensemble des transistors qui composent le microprocesseur.

En 1971, un microprocesseur comptait 2300 transistors. Aujourd'hui, le nombre de transistors dans les microprocesseurs est de l'ordre de 800×10^6, soit un facteur $\times 347800$ plus important comparativement au nombre de transistors qui pouvait être gravé 40 ans plus tôt. De plus, la fréquence de calcul du tout premier microprocesseur, l'Intel 4004® à 4 bits, fonctionnait à 108 kHz. Aujourd'hui, la fréquence d'exécution de ces microprocesseurs atteint 3 GHz, c'est à dire que l'on a multiplié par $\times 30000$ la vitesse d'exécution des microprocesseurs.

En 1973, deux ans après la sortie du premier microprocesseur Intel, François Gernelle et André Truong Tong Ti inventent le *MICRAL*, le premier micro ordinateur, i.e. ordinateur fonctionnant à partir d'un microprocesseur (Réfs. [6, 7]).

Afin de mieux comprendre les diverses problématiques actuelles liées à la vitesse de calcul, au nombre limité de processeurs, à l'apparition des cartes graphiques et aux architectures multiprocesseurs, il est intéressant de revenir un petit peu en arrière et de voir en détail l'évolution de ces processeurs qui ont aujourd'hui une place prépondérante dans les appareils du quotidien, dans la recherche et dans la compréhension de phénomènes grâce à la simulation numérique.

1.1.1 Des débuts à nos jours

Le premier circuit intégré ("IC" pour *Integrated Circuit*) a été inventé en 1958 par J. Kilby de la société *Texas Instrument* travaillant pour un projet militaire sur l'innovation de la fabrication de transistors, notamment sur la connexion des composants de galettes de germanium. L'année suivante un nouveau procédé de fabrication sur des tranches de silicium voit le jour et permet dans les dix années qui suivent, une évolution majeure dans la fabrication des circuits intégrés.

En 1968, pressentant un futur propice dans le développement des circuits intégrés, R. Noyce, G. Moore et A. Grove se réunissent pour fonder une nouvelle entreprise dans le but de produire des circuits intégrés à traitement centralisé, aujourd'hui appelés CPU (Central Processing Units). Cette entreprise voit le jour sous le nom d'Intel (pour "Integrated Electronics") et lance avec l'entreprise japonaise *Busicom* une nouvelle génération de calculateurs programmables connus sous le nom de *Program Data Processor 8* (PDP-8). La mise au point de ces calculateurs permet la conception des premiers "micro ordinateurs sur puce". le terme de "micro processeur" apparaîtra seulement à partir de 1972. A la suite du succès d'Intel dans la fabrication de ces micro processeurs et les performances de ceux-ci, d'autres compagnies telles que *Motorola* ou encore *Zilog*, vont commencer à produire des puces de plus en plus compétitives.

Depuis l'apparition du premier micro processeur, un rythme d'évolution très rapide s'est installé. Les avancées successives de l'électronique et des techniques de fabrications, concernant notamment les progrès dans le domaine de la conduction des matériaux semi-conducteurs (e.g. avec le dopage électronique), vont permettre d'améliorer la fiabilité, les performances et la miniaturisation des processeurs. Ce rythme d'évolution s'est maintenu jusqu'à aujourd'hui faisant apparaître de nombreux modèles de plus en plus puissant et de plus en plus complexes. Une "loi" tenant compte de l'évolution technologique des circuits intégrés et de l'évolution de l'architecture du matériel informatique a été énoncée par G. Moore en 1965 (Réf. [8]) et est considérée comme étant encore valable actuellement. Cette loi prédit que le nombre de transistors sur une puce de silicium double tous les deux ans et à coût constant. La Fig. 1.1 montre l'évolution du nombre de transistors gravés sur les microprocesseurs entre 1970 et 2000. La flèche grise correspond à l'évolution prédite par la loi de Moore et suit étonnamment bien l'évolution des microprocesseurs représentée par des carrés pleins (bleus et noirs).

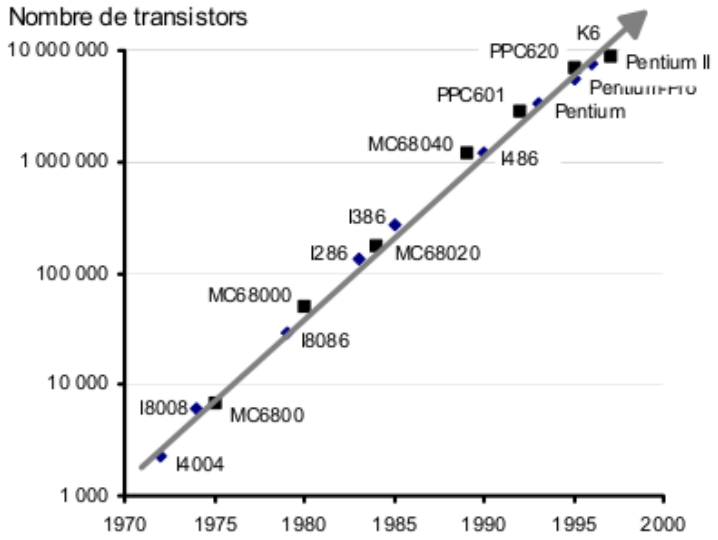


FIGURE 1.1 – Évolution du nombre de transistors gravés sur un microprocesseur entre les années 1970 et 2000. La flèche grise correspond à l'évolution prédite par la loi de Moore du nombre de transistors disponibles sur les microprocesseurs (Réf. [9]). Les carrés bleus correspondent aux microprocesseurs Intel et les carrés noirs à ceux produits par Motorola-IBM. L'échelle sur l'axe des ordonnées est une échelle logarithmique.

L'évolution des performances des microprocesseurs peut être interprétée à partir de deux phénomènes. Le premier concerne l'augmentation de la rapidité d'exécution des composants de base, qui est intrinsèquement lié à la diminution de la taille des motifs technologiques. Des transistors plus petits ont pour conséquence des processeurs plus rapides. Autrement dit, plus les transistors sont petits, plus la fréquence de transmission des signaux électriques peut être élevée (grâce à l'augmentation de la fréquence de l'horloge). Le deuxième phénomène est lié à l'amélioration de l'architecture des machines. Ces architectures reposent sur le principe que tout calcul complexe peut être décomposé en une suite d'opérations plus simples pouvant être exécutées automatiquement. Ainsi ces améliorations sont effectuées par simplification du décodage et de l'exécution des instructions (RISC et CISC), ou encore par l'exécution simultanée de plusieurs instructions (parallélisme), par la prédiction de branchements ou enfin par l'exécution spéculative. Ces différentes méthodes permettent donc de réduire le temps d'exécution des instructions. La demande croissante de performances se fait de plus en plus forte et est le moteur principal de cette évolution.

La miniaturisation a des intérêts économiques majeurs. La diminution de la taille des transistors permet d'en graver beaucoup plus sur une galette de silicium et permet une diminution du coût des transistors. De plus, des transistors plus petits sont moins gourmands en énergie. Ceci caractérise bien, en plus des performances technologiques apportées, un profit économique croissant. Cependant, un problème important se pose actuellement et

voit une limitation liée à la taille des transistors ainsi que leurs fonctionnements pour des longueurs de l'ordre du nanomètre.

La barre des 100 nanomètres a été franchie en 2002. Depuis une dizaine d'années, les chercheurs doivent rivaliser d'ingéniosité pour ajuster les impératifs industriels et les contraintes inhérentes à l'échelle nanométrique. D'après de récentes études en nanomatériaux, la limite acceptable sera atteinte entre 3 ou 2 nanomètres et devrait nous conduire jusqu'en 2030 (Réf. [10, 11, 12]). Les transistors se rapprochant à grand pas de l'échelle de l'atome, c'est une toute nouvelle électronique qui est actuellement en cours d'élaboration dans les laboratoires. Plusieurs travaux de recherches ont lieu afin de découvrir un remplacement au silicium. De nouveaux composants, tel que le graphène, sont déjà considérés comme d'inévitables remplaçants. Alors qu'à faible épaisseur, la plupart des matériaux cessent de conduire le courant électrique, le graphène reste le seul matériau connu actuellement et aussi bon conducteur que le cuivre. La mobilité des électrons du graphène est près de 50 à 500 fois plus élevée que dans le silicium. Un tout premier transistor au graphène a vu le jour en 2007, mis au point par les chercheurs André Geim et Konstantin Novodelov, prix Nobel de physique 2010. En 2008, ces chercheurs ont élaboré le plus petit transistor jamais conçu. La taille de ce transistor est équivalente à une épaisseur d'un atome et de 10 atomes de long et ouvre la voie à une nouvelle technologie et aux développements de nouveaux systèmes électroniques.

1.2 Fonctionnement général du CPU

Le microprocesseur, aussi appelé CPU (Central Processing Units) est l'élément principal d'un ordinateur. Le microprocesseur doit être en mesure de lire les instructions en mémoire, de les décoder, puis de les exécuter. Ces instructions "machine" sont traitées les unes après les autres et cette suite d'instructions est appelée communément un programme. Chaque modèle de microprocesseur lit des instructions spécifiques à sa conception sous forme d'un langage de base que l'on appelle assembleur. Ce langage de programmation est complexe et spécifique à la machine et est codé en hexadécimal.

D'une manière générale, un microprocesseur est constitué de deux unités fonctionnelles séparées et de registres. Ces trois éléments sont la base de la composition logique des CPU et c'est par elles que les instructions vont être traitées. Les deux unités fonctionnelles sont l'**UAL** (Unité Arithmétique et Logique [1]) et l'**UC** pour l'unité de commande ou de contrôle (cf. Fig. 1.2).

1. en anglais : Arithmetic Logic Unit (ALU)

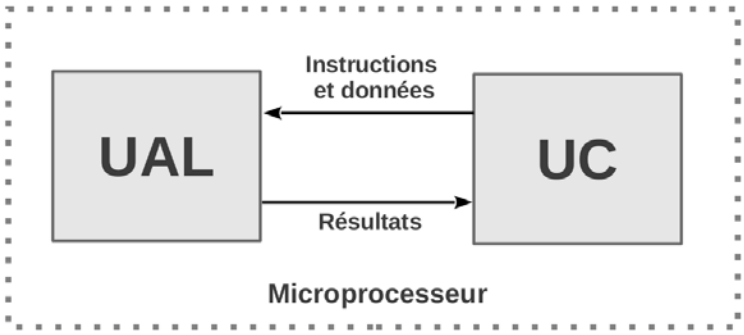


FIGURE 1.2 – Représentation des deux unités fondamentales d'un microprocesseur. L'Unité Arithmétique et Logique (UAL) effectuent des opérations simples sur les données (additions, soustractions, multiplications, ...) et l'Unité de Contrôle (UC) séquence le déroulement de l'instruction.

L'unité arithmétique et logique a pour fonction d'effectuer des opérations arithmétiques et logiques. L'UAL est constituée de circuits logiques telles que les additionneurs, les soustracteurs, les multiplicateurs, les diviseurs et les comparateurs logiques (e.g. OU, ET, ...). Les microprocesseurs modernes peuvent maintenant effectuer des calculs plus complexes sur de très larges volumes de données de types flottantes (*float* en anglais). Ceci est fait par l'ajout d'une unité de calcul en virgule flottante noté généralement FPU pour *Floating Point Unit*.

L'unité de commande est la partie du microprocesseur qui commande et contrôle le fonctionnement du système. Ses circuits génèrent les signaux nécessaires à l'exécution de chaque instruction d'un programme. Les principales tâches que l'UC effectue sont :

- La lecture de l'instruction.
- Le décodage.
- L'exécution.
- La préparation de l'instruction suivante.

L'ensemble du dispositif permettant la coordination des tâches pour l'exécution des opérations spécifiées dans les instructions d'un programme est constitué de :

- Un compteur ordinal, qui est un registre contenant l'adresse de la case mémoire où est stockée l'instruction suivante à chercher.
- Un registre d'instructions , qui contient l'instruction à exécuter.
- Un décodeur de code d'opérations, qui détermine quelle opération doit être effectuée parmi le jeu d'instructions.
- Un séquenceur, qui génère les signaux de commandes pour piloter les autres entités du microprocesseur et synchroniser ce dernier avec la mémoire.

Enfin, le jeu d'instructions, i.e. l'ensemble des instructions de base que le microprocesseur peut exécuter, peut être différent d'une famille de microprocesseurs à une autre et seules des exigences de simplicité, de rapidité, ou encore d'universalité sont requises. Le nombre d'instructions peut également varier et est compris entre environ 50 et 350 instructions. Ce nombre dépend du type d'organisation choisie et est réparti entre deux catégories appelées **CISC** pour *Complex Instruction Set Computer* et **RISC** pour *Reduced Instruction Set Computer*. Les processeurs CISC possèdent un jeu étendu d'instructions complexes contrairement au processeurs RISC qui possèdent un jeu d'instructions réduit où chaque instruction effectue une seule opération élémentaire. D'une manière générale, les différentes instructions qui sont exécutées par le microprocesseur peuvent être classées par catégorie comme par exemple, le transfert de données entre le microprocesseur et la mémoire, les opérations (arithmétiques et logiques), le contrôle des séquences ou encore les "entrées/sorties" entre le microprocesseur et les périphériques.

Les autres éléments important du microprocesseur sont les registres. Les registres sont des mémoires internes au microprocesseur et le temps d'accès à ce type de mémoires est très rapide. En général, ce temps d'accès varie comme l'inverse de la fréquence de l'horloge. Plusieurs registres sont disponibles sur le microprocesseur et leur nombre peu fortement varier d'une architecture à une autre (de 10 à plus de 100). De façon générale, plus un CPU possède de registres, plus il est performant. La taille d'un registre se mesure en "bit" (*Binary Information*) et lorsque l'on parle d'un microprocesseur 64 bits, cela signifie que le microprocesseur est composé de registres de taille 64 bits. Les registres peuvent être utilisés pour différentes fonctions sur le microprocesseur. Les registres généraux contiennent les données et résultats des opérations réalisées par le microprocesseur et sont aussi appelés "accumulateurs". Mais d'autres fonctions plus spécifiques peuvent être attribuées à ces registres comme le registre d'instructions ou encore le compteur ordinal (explicité plus haut).

Finalement, le fonctionnement d'un microprocesseur peut être vu comme la répétition d'un cycle dans lequel il effectue séquentiellement différentes tâches (lecture, décodage, exécution d'opérations, ...) pour l'exécution d'une instruction avant de passer à la suivante.

1.2.1 Organisation de la mémoire

Nous venons de voir l'importance de l'unité centrale de calcul (CPU). Nous allons voir à présent que la mémoire tient une part importante dans l'efficacité du traitement des instructions du CPU. L'unité centrale reçoit des informations via des périphériques d'entrée (clavier, lecteur de code barre, souris, ...), les traite et envoie les résultats sur des périphériques de sortie (écran, imprimante, haut-parleurs, ...). Pour ces traitements, l'unité centrale a besoin d'un espace de stockage, la mémoire centrale.

Ces structures permettent le stockage des informations de façon permanente comme dans le cas des mémoires ROM (*Read Only Memory*) qui sont accédées seulement en lecture, ou de façon temporaire telles que les mémoires RAM (*Random Acces Memory*), les registres ou encore les mémoires caches, qui peuvent être accédées en lecture et en écriture.

Les informations dont a besoin le CPU (les programmes, les données) pour exécuter ses instructions sont stockées dans la mémoire principale, la DRAM (*Dynamic Random Acces Memory*). Cependant, la fréquence des microprocesseurs a évolué extrêmement rapidement et atteint aujourd'hui des fréquences de l'ordre du GHz. En contrepartie, les temps d'accès à la mémoire DRAM n'ont pas connu la même évolution et bien que ces temps ont été réduits à environ 50 nanosecondes (soit une réduction d'un facteur 3 en $\sim$ 20 ans), le microprocesseur passe la majeure partie de son temps à attendre pour accéder aux données, ce qui réduit significativement la performance des machines. A titre d'exemple, les mémoires actuelles ont une fréquence de l'ordre de 100 MHz et les CPU de l'ordre de 3 GHz. Ainsi, pour un temps de latence de 50 ns, le nombre de cycles d'horloge est de l'ordre de 150 cycles pour l'accès aux informations en mémoire !

La méthode utilisée pour réduire les délais d'attente des informations stockées en mémoire vive (mémoire DRAM) est l'ajout d'un autre type de mémoire (de type SRAM, pour *Static Random Acces Memory*), appelée mémoire cache (également appelée *antémémoire* ou encore *mémoire tampon*) et dont les temps d'accès sont nettement inférieurs à ceux de la mémoire DRAM (de l'ordre de la nanoseconde). Les deux principes à la base de l'idée du cache, sont les principes de localité temporelle et spatiale. La localité temporelle suppose que si une adresse mémoire est utilisée, alors elle le sera certainement à nouveau dans un futur proche. La localité spatiale suppose que si une adresse mémoire est utilisée, alors les adresses proches le seront sûrement dans un futur proche. La solution consiste donc à inclure ce type de mémoire à proximité du microprocesseur (cf. Fig. 1.3) et d'y stocker temporairement les principales données devant être traitées par le microprocesseur.

Toutefois, même si cette mémoire (SRAM) peut être jusqu'à 10 fois plus rapide que la DRAM, sa capacité de stockage est cependant très limitée. Une des causes de cette limitation est principalement due au coût de production de celle-ci. Afin d'avoir un système utilisant ce type de mémoire mais à moindre coût, plusieurs niveaux de mémoires caches sont utilisés (de 2 à 3 niveaux) dont le temps d'accès et la capacité de stockage augmente en fonction du niveau dans la hiérarchie de la mémoire. Il peut y avoir de nombreux caches pour un seul microprocesseur, chacun pouvant se spécialiser dans un jeu d'instructions particulier.

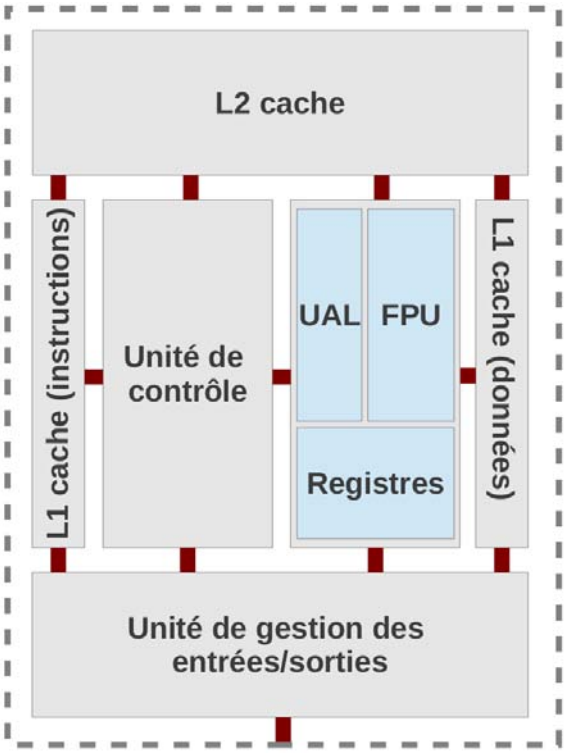


FIGURE 1.3 – Représentation simplifiée de l'architecture du microprocesseur et de la mémoire cache associée. Le microprocesseur est représenté par une unité de contrôle (UC), une unité de calcul (composé d'une UAL pour *unité arithmétique et logique*, d'une FPU pour *floating point unit* et de registres), par des espaces mémoires caches de niveau 1 et 2 (L1 et L2) et d'une unité de gestion des entrées et sorties vers la mémoire et/ou vers les coprocesseurs.

Il existe deux façons d'organiser la mémoire, en séparant celle-ci en deux zones distinctes, permettant ainsi d'optimiser les accès à la mémoire. Une zone de programmes (e.g. les instructions) et une zone de données (e.g. les valeurs numériques). Les architectures les plus courantes sont : l'architecture de Von Neumann qui est la plus souvent utilisée pour la mémoire DRAM et qui consiste à stocker les instructions et les données dans le même emplacement mémoire. L'architecture Harvard, souvent utilisée pour la mémoire SRAM, consiste à séparer dans deux mémoires distinctes les instructions et les données comme ceci est représenté sur la Fig. 1.3 dans le cas de la mémoire cache de niveau 1. Cette dernière architecture est très rapide et elle est très utilisée mais possède cependant une structure interne très complexe. Si l'on reprend l'exemple cité plus haut avec cette fois-ci un temps d'accès à la mémoire cache de l'ordre de 5 ns, la latence est réduite à une quinzaine de cycles d'horloge.

En résumé, la mémoire est caractérisée par sa capacité, représentant le volume global de stockage des informations et son temps d'accès qui correspond à l'intervalle de temps entre la demande de lecture/écriture et la disponibilité de la donnée. Cependant, le microprocesseur opère à des fréquences beaucoup plus importantes que la mémoire ($\sim$ GHz pour le CPU contre du MHz pour la mémoire) ce qui devient vite limitant au niveau des performances d'exécution du microprocesseur. C'est pourquoi plusieurs méthodes ont été développées pour palier à ces limitations telles que l'utilisation d'architectures de mémoire différentes (Von Neuman ou Harvard) ou encore l'utilisation de mémoires (mémoires caches) avec des capacités moins importante mais beaucoup plus rapides d'accès. D'autres solutions existent concernant l'amélioration des performances des microprocesseurs, comme avec l'amélioration des "bus" qui servent à transférer les informations entre le CPU et la mémoire ou l'utilisation de "ponts", appelés *North Bridge* et *South bridge* qui permettent l'optimisation de la gestion des transferts de données. La liste des exemples présentés ici n'est évidemment pas exhaustive mais reflète un point essentiel de la difficulté de plus en plus importante à améliorer les performances des microprocesseurs et de trouver un compromis entre la forte évolution des architectures microprocesseurs et celle plus faible des architectures mémoires.

1.3 Petite histoire des GPU

Contrairement au CPU, le GPU (*Graphical Processing Unit*) a fait son apparition bien plus tardivement. Il faudra attendre les années 1990 (voir la fin des années 90) pour voir apparaître les premières cartes graphiques sur le marché, destinées au grand public. Les toutes premières cartes informatiques additionnelles destinées aux calculs graphiques ont vu le jour au début des années 1980. Ces cartes misent à disposition par la société *Matrox* (fondée en 1976) et un peu plus tard par la société *ATI* (fondée en 1985), avaient pour fonction de transmettre les images produites par l'ordinateur sur l'écran, bien que l'affichage de la 2D et de la 3D qui n'en était qu'à ses prémisses, utilisait massivement la puissance des CPU. C'est la société *3DFX* avec la sortie de la carte *"Voodoo"* au milieu des années 1990, proposant de véritables cartes d'accélérations 3D à un prix abordable, qui marquera le début des cartes graphiques. A partir de cette même période, fortement poussées par l'industrie du jeu vidéo, plusieurs industriels tels que *NVIDIA, ATI, Tseng Labs, 3Dlabs* et bien d'autres, se lancent dans le développement de ces cartes grand public prenant en charge toute la partie liée à l'affichage et au calcul de rendu graphique. Aujourd'hui, deux principaux constructeurs de cartes graphiques ont subsisté et sont : ATI, racheté par la société AMD (spécialisée dans la fabrication de microprocesseurs) et la société NVIDIA. La concurrence et le succès dans le milieu du jeu vidéo pousse ces deux sociétés à améliorer

sans cesse leur cartes graphiques. A l'origine, le développement de ces cartes avait pour but de décharger le microprocesseur des calculs liés aux graphismes mais elles sont maintenant devenues un des éléments constitutifs des plus important dans le choix d'un ordinateur, dans un cadre tant personnel que professionnel. Aujourd'hui les performances de calculs d'un ordinateur ne reposent plus seulement sur les performances des microprocesseurs mais également sur les performances des processeurs graphiques.

Les motivations actuelles sur l'utilisation des cartes graphiques comme processeurs de calculs dans le domaine de la recherche et de la modélisation, s'expliquent en plusieurs points. Depuis quelques années, les CPU commencent à montrer leurs limites technologiques en terme d'architecture et de vitesse. Les CPU se sont orientés depuis quelques années vers des architectures multi-coeurs (plusieurs coeurs de calcul au sein d'un même microprocesseur), ce qui leur permet encore de fournir une puissance de calcul toujours plus élevée. Mais cette architecture a une limite qui est liée au temps de latence relativement long lors du transfert des informations entre la mémoire et le microprocesseur. Autrement dit, la bande passante ou la quantité d'informations transférées par seconde, n'est pas suffisante et est un facteur très limitant pour les performances des CPU. La puissance brute de calcul proposée par les GPU a largement dépassé depuis quelques années celle affichée par les CPU les plus performants comme le montre la Fig. 1.4. C'est à partir de 2003 que l'on peut voir la progression (en terme de GB/s) des cartes graphiques (ici, les cartes NVIDIA) par rapport à l'évolution des CPU. Aujourd'hui, les architectures des cartes graphiques des gammes *GeForceGTX* montrent une bande passante pouvant être jusqu'à quatre fois supérieures par rapport aux architectures des microprocesseurs les plus récents (cf. Fig. 1.4).

La forte progression des GPU est grandement due à l'architecture parallèle hautement spécialisée et optimisée pour les opérations graphiques. De plus, le regroupement possible des GPU en ferme de calcul (cluster) démultiplie encore cette puissance de calcul. Enfin, le faible coût de ces cartes permet d'assurer une grande accessibilité.

1.3.1 Le GPGPU

Les constructeurs de cartes graphiques ont continué leurs recherches et le développement des GPU afin de les rendre ouvert à tous et programmables. C'est à partir de 2002 qu'une importante avancée dans la flexibilité de programmation des cartes graphiques est apparue.

La communauté scientifique commence à s'intéresser de plus en plus à ces cartes devenues plus polyvalentes et les premières applications numériques, traditionnellement réalisées par les CPU, voient le jour. Cette utilisation pour faire des GPU des super calculateurs, est nommée *GPGPU* [13] pour *"General-Purpose computation on GPU"* ou *Programmation*

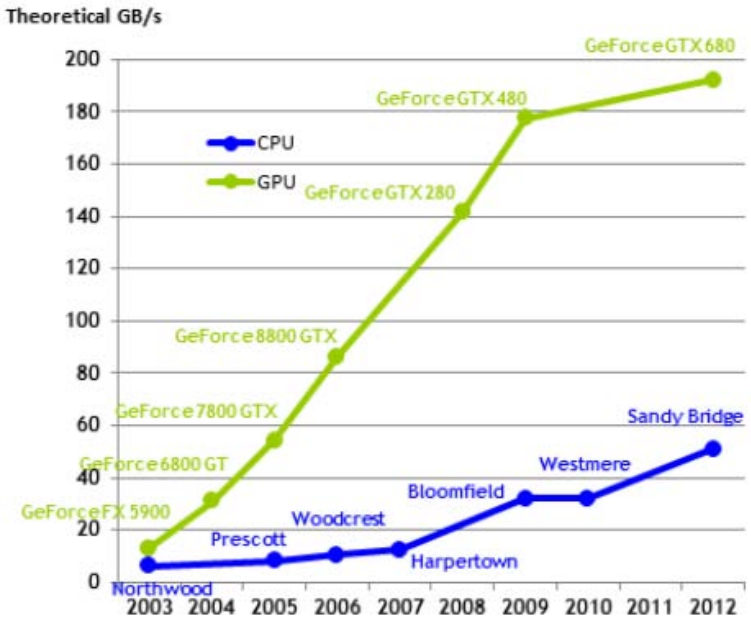


FIGURE 1.4 – Comparaison de la bande passante mémoire (en GB/s) pour le CPU (*INTEL*) et le GPU (*NVIDIA*) entre 2003 et 2012. A partir de 2003 les GPU commencent à se démarquer et vont prendre un net ascendant sur les CPU dans les années suivantes (Réf. [1]).

générique sur GPU. Cette technologie permet d'utiliser les GPU pour décharger les CPU lors de calculs lourds comme la simulation physique, l'encodage vidéo, le rendu graphique, ..., etc. Autrement dit, l'intérêt principal de la programmation GPGPU est de pouvoir, à faible coût, utiliser les GPU ou des clusters de GPU, permettant d'accélérer fortement l'ensemble des calculs, avec des gains pouvant aller jusqu'à plus de 100%.

C'est à partir de cette technologie que chaque constructeur a créé sa propre solution de développement. De nouveaux langages de programmation utilisés dans le cadre GPGPU et spécialisés dans le calcul généraliste, i.e. permettant d'utiliser les capacités du GPU seulement comme solution de calcul, ont fait leur apparition. Deux langages existent actuellement : Le langage CUDA (*Computer Unified Device Architecture*), dont nous parlerons dans la suite, est un langage développé par NVIDIA et est spécialement conçu pour le développement sur des cartes graphiques propriétaires. Le langage OpenCL (Réf. [14, 15, 16, 17]), est un langage plus récent et est adapté à tous les types de cartes graphiques (le développement de OpenCL est largement soutenu par ATI/AMD). Malgré des débuts un peu incertains, un engouement croissant de la communauté scientifique est à noter depuis maintenant une dizaine d'années et encourage grandement les améliorations et les avancées des cartes graphiques et du langage de programmation.

Nous allons dans ce qui suit présenter l'architecture des cartes graphiques (cartes gra-

FIGURE 1.5 – Photographie de la dernière carte dédiée au calcul intensif parallèle de NVIDIA : carte NVIDIA Tesla C2050 ®.

phiques NVIDIA) tout en faisant le lien avec celle des microprocesseurs (présentée dans la Section 1.2 et 1.2.1). Nous verrons également l'importance de l'organisation de la mémoire sur les GPU avant de présenter le langage de programmation utilisé pour le développement du code PIC MCC ; code qui sera présenté dans le chapitre suivant.

1.4 Architecture des GPU NVIDIA

Les GPU sont des processeurs portés par une carte annexe, prenant à leur charge le traitement des images et des données 3D, ainsi que l'affichage. Elles sont "architecturalement" prévues pour un traitement massivement parallèle des données et peuvent contenir actuellement jusqu'à 500 processeurs (appelé également *stream processor cores*). C'est le cas de la gamme Tesla (Fig. 1.5), proposée par NVIDIA, qui associe jusqu'à quatre GPU et 16 Go de mémoire dans un unique châssis, portant à 4 TFLOPS [2] (4000 GFLOPS) la puissance de calcul de ce super calculateur. Plusieurs calculateurs de ce type peuvent de plus être associés par un réseau classique. les cartes graphiques permettent aujourd'hui une programmation flexible et avancée, en proposant de plus en plus de contrôles sur les caractéristiques toujours plus nombreuses, telles que l'utilisation de types de données (en particulier le calcul flottant sur 32bits), de structures et d'instructions (branchements conditionnels) de plus en plus complexes.

Dans son ensemble, l'architecture du GPU dans sa version "Tesla" est basée sur un jeu d'instructions qualifié de *scalaire*. Cette architecture représentée Fig. 1.6 montre la disposition des processeurs pour la carte NVIDIA GeForce 8800. Celle-ci est composée de 128 coeurs de calcul, appelés coeurs SP (SP pour *Stream Processor*). Ces coeurs SP sont repartis dans les 16 *Streaming Multiprocessors* (SM), eux même répartis dans 8 unités d'instructions indépendantes appelées *Texture/Processor clusters* (TPC). Une analogie

2. FLOPS : **FL**oating point **O**perations **P**er **S**econd. Représente le nombre d'opérations à virgule flottante par seconde.

peut être faite entre l'architecture du microprocesseur présentée Fig. 1.3 et l'architecture du multiprocesseur ou "unité multi-coeurs" (SM) Fig. 1.6. L'unité SFU et SP correspondent aux unités de calculs en référence aux unités de calculs UAL (cf. Section 1.2.1). Chaque unité multi-coeurs SM possède également une unité de contrôle représentée par une unité SMC pour *SM controller* et une unité GM (pour *geometry controller*) et une mémoire cache de niveau 1 pour les instructions et pour les données.

Les différentes "micro-architectures" de l'architecture Tesla ont suivi des évolutions progressives. La stratégie généralement suivie consiste à introduire une nouvelle révision majeure dans le "haut-de-gamme" pour ensuite réutiliser sa micro-architecture dans le milieu et entrée de gamme en réduisant progressivement le nombre de TPC et de partitions mémoire.

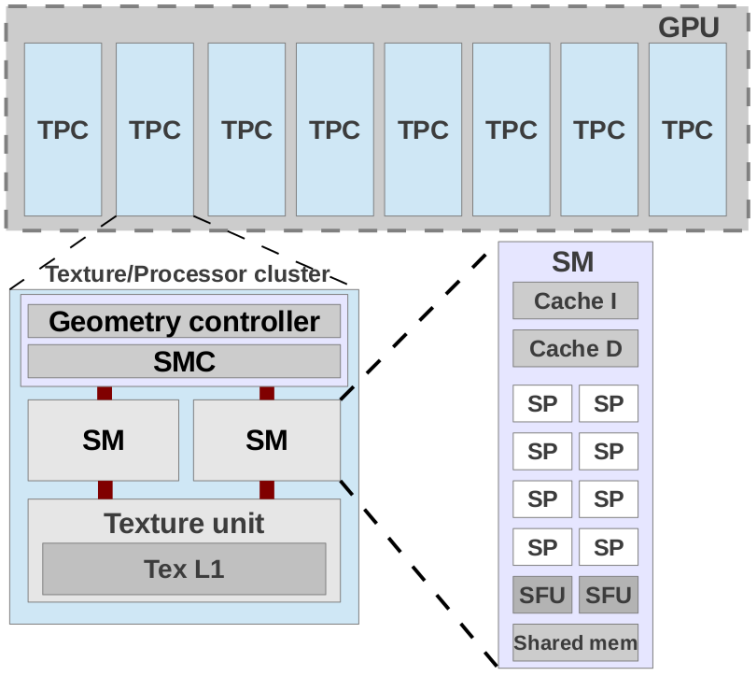


FIGURE 1.6 – Représentation simplifiée de l'architecture des cartes graphiques NVIDIA Geforce 8800. La carte graphique est composée de 8 unités de traitement (coeurs) TPC. Chaque TPC contient une unité de contrôle (SMC et geometry controller), une unité de texture et son espace mémoire cache (Tex L1) et de deux unités multi-coeurs (SM). Chaque unité multi-coeurs est composé de 8 coeurs (SP), d'une mémoire partagée et de mémoires caches (I pour instructions et D pour données) ainsi que deux unités de calculs (SFU pour *special function unit*).

Chaque TPC contient une unité de contrôle (SMC et GM), deux SM (Streaming Multiprocessor) et une unité de texture (cf. Fig. 1.6). Le multiprocesseur SM correspond à l'unité d'exécution des programmes en parallèles et comme le montre la Fig. 1.6, il est composé de huit coeurs SP, de deux unités de fonctions spéciales (SFU), de deux mémoires cache de

niveau L1 (pour les instructions et pour les données) et d'une mémoire partagée (*Shared memory*).

Les coeurs SP correspondent aux unités de calculs arithmétiques (UAL, cf. Section 1.2) et sont les processeurs "primaires". Ils effectuent des opérations scalaires fondamentales (de type entier et flottant) d'addition, de multiplication et de multiplication-addition (MAD). La fréquence de calcul de ces processeurs est de l'ordre de 1.5 GHz.

Les unités SFU sont utilisées par les coeurs (stream processors) comme unités de calculs pour les fonctions transcendantales (cosinus, sinus, exponentiel, logarithme, racine carré, ...) et possèdent des fonctions dédiées entièrement à l'évaluation précise d'interpolations de coordonnées de texture, de couleurs et de profondeur.

L'unité de texture est une unité séparée des Streaming Multiprocessor et contient une zone de mémoire cache permettant de traiter des instructions sur les coordonnées de textures.

L'espace mémoire, au niveau de l'architecture Tesla, est hétérogène. Cet espace mémoire est divisé en espaces distincts adressés explicitement. Chaque espace est accessible au moyen d'instructions distinctes, comme nous le verrons dans la Section 1.5. La décision de placement des données ne peut donc être faite qu'au moment de la compilation. En particulier, le compilateur doit être capable d'inférer statiquement l'espace mémoire désignée par chaque pointeur. Ces espaces comprennent :

- La mémoire constante
- La mémoire texture
- La mémoire locale
- La mémoire globale
- La mémoire partagée

Cette distinction des espaces mémoires permet de séparer nettement les zones mémoires en lecture seule (constante, texture), locale à un thread (locale), en lecture/écriture (globale), à latence courte et déterministe (partagée). La latence de chaque type de mémoire pouvant être estimée précisément de manière statique.

Cette architecture parallèle, composées d'un grand nombre d'unités de calculs, exploitent fortement les modes de fonctionnement SIMD (*Single Instruction Multiple Data*) / MIMD (*Multiple Instructions Multiple Data*) [18], autorisant l'exécution simultanée de plusieurs parties de code. Toutefois, L'exécution des programmes repose sur la notion de thread, similaire à celle de threads sur CPU, i.e. processus légers pouvant s'exécuter en parallèle. Lors de l'exécution d'un programme, plusieurs groupes de threads vont être lancés en parallèle afin d'effectuer des opérations sur un large ensemble de données. On parle alors d'architecture SIMT (*Single Instruction Multiple Threads*). Les multiprocesseurs (SM)

gèrent les threads par groupe de 32 threads, appelés *warps*[3].

Chaque unité multi-coeurs gère un ensemble de 24 warps, soit un total de 728 threads. Ces warps sont attribués à chaque coeur SP, où chacun de ces threads exécutent, indépendamment des autres, ses propres instructions. Ce type d'exécution est efficace et performant lorsque les 32 threads d'un même warp utilisent le même chemin d'exécution.

Comme ceci est représenté sur la Fig. 1.7, la mémoire est divisée en plusieurs modules mémoires ayant chacun le même espace. Ces espaces sont appelés "banques" et peuvent être accédés simultanément. Toute lecture ou écriture de N données doit être répartie dans N banques mémoires distinctes. Cependant, si deux requêtes se retrouvent dans la même banque mémoire, il y a ce qu'on appelle "un conflit de banque" et les accès sont sérialisés (traités les uns après les autres).

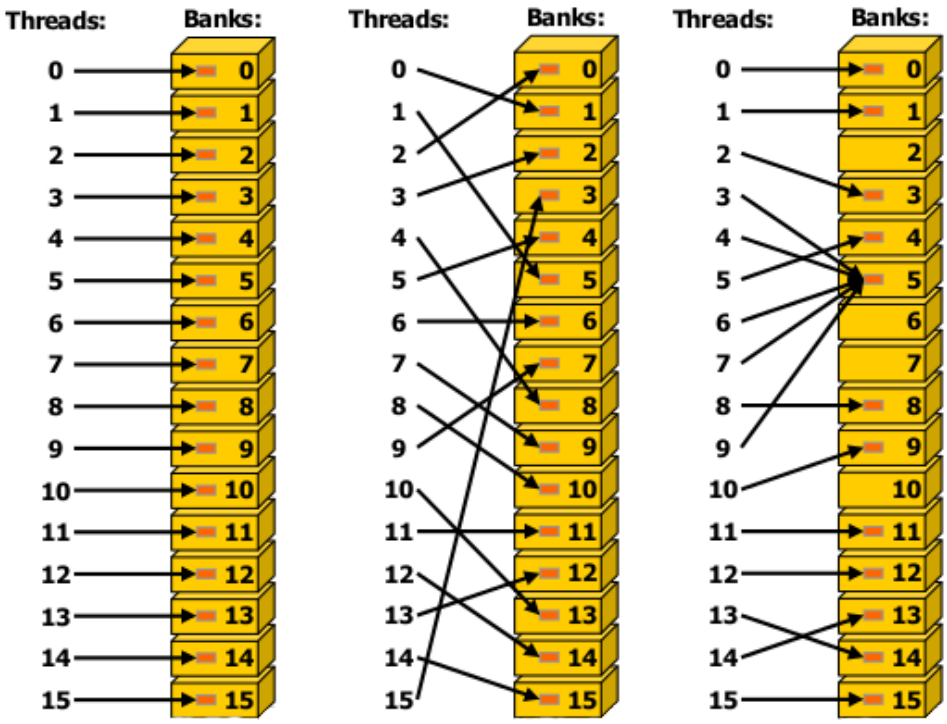


FIGURE 1.7 – Accès dans la mémoire aux banques mémoires lors de l'exécution des threads d'un demi-warp (16 threads). (a) Sans conflits d'accès aux banques mémoires, (b) avec conflits lors de l'accès aux banques mémoires (Réf. [1]). Lorsque plusieurs threads accèdent à la même banque, les accès sont sérialisés.

Les banques sont organisées de telles sorte que chacune d'entre elles possèdent un espace mémoire de 32 *bits*. De même, chaque banque mémoire a une bande passante de 32 *bits* sur deux cycles d'horloge. Une requête en mémoire, pour un warp, est divisée en deux

3. L'origine de ce terme provient des ensembles de fils parallèles sur les machines à tisser.

processus. La première moitié du warp est envoyée à l'exécution (les 16 premiers threads) lors du premier cycle d'horloge, et la seconde moitié lors du second cycle d'horloge. Il faut noter que la fréquence du multiprocesseur correspond à la moitié de la fréquence des coeurs, soit $\sim$ 750 MHz. Alors, pour un cycle horloge du multiprocesseur, les coeurs ont effectué deux cycles, soit le traitement de deux jeux de 16 threads. La conséquence et l'avantage de cette procédure est qu'il n'y a pas de conflits de banques entre les threads appartenant à la première moitié du warp et ceux appartenant à la deuxième moitié. Cette figure (Fig.1.7) montre différents exemples d'accès à la mémoire où chaque thread accède à la mémoire. D'autres cas d'accès à la mémoire sont répertoriés dans la documentation de NVIDIA (Réf. [1]).

Finalement, le mode de fonctionnement SIMT permet l'exécution d'une instruction par plusieurs threads indépendants, en parallèle. De façon générale, il est possible de s'abstraire des paramètres d'exécutions tels que les warps. Cependant pour atteindre des performances non-négligeables dans l'exécution du code, il est important de tenir compte de ces caractéristiques. De manière analogue, le rôle des différents niveaux de mémoires caches peuvent être ignorés. Toutefois, lors de l'écriture des algorithmes sur le GPU, il est important pour atteindre de bonnes performances et une bonne optimisation de ceux-ci, de considérer cette hiérarchie mémoire dans la structure du code.

Plusieurs langages de haut niveau existent et permettent l'utilisation et l'exécution des instructions sur le matériel spécifique aux cartes graphiques. Les langages de plus haut niveau disponibles sur GPU se répartissent selon deux grandes catégories : les *Shading Languages*, dont les instructions sont principalement destinées aux calculs graphiques, et ceux dont l'objectif est le calcul générique. Nous allons développer dans la section suivante le langage haut niveau CUDA, qui est le langage proposé par le constructeur NVIDIA permettant d'exploiter la puissance de calcul des GPU appartenant à cette même famille.

1.5 Compute Unified Device Architecture

Le langage CUDA pour *Compute Unified Device Architecture* est mis à disposition du grand public depuis 2007. CUDA est un langage de programmation proche du C/C++ permettant d'exploiter les capacités du GPU et ses ressources matérielles, en particulier en ce qui concerne la gestion de la mémoire et l'organisation des traitements. Il propose sa propre API (*Application Programming Interface* ou interface de programmation) haut niveau, consistant en quelques extensions au langage C et dont la prise en main ne nécessite pas de connaissances approfondies dans le domaine graphique tout en permettant de s'abstraire du fonctionnement de l'architecture physique du GPU.

Le principe de traitement d'un problème sur une architecture hautement parallèle est

de décomposer ce problème en plusieurs plus petits problèmes qui peuvent être résolus en parallèle. Ainsi, la partition d'un large tableau de données est réalisée par sa décomposition en plusieurs blocs. Chaque bloc est exécuté de façon indépendante en parallèle et les éléments de chaque bloc sont exécutés coopérativement en parallèle. La Fig. 1.8 montre la décomposition d'un ensemble de données en une grille de 2×2 blocs qui sont décomposés eux même en 4×4 éléments.

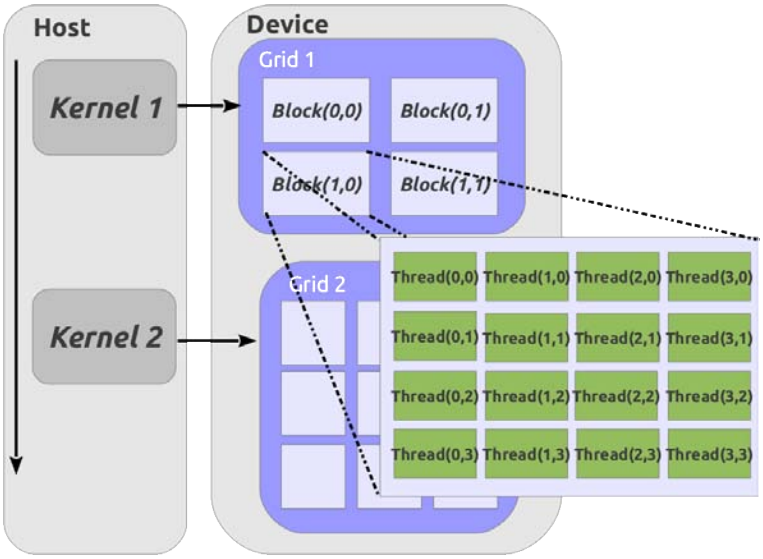


FIGURE 1.8 – Décomposition des données en grille de blocs, où chaque bloc contient un certain nombre d'éléments exécutés en parallèle. Le terme *Host* définit le CPU et le terme *Device* correspond au GPU. La dimension utilisée pour la définition de la grille est bidimensionnelle au niveau des blocs et des threads.

Tous les threads contenus dans une grille sont envoyés à l'exécution par un *kernel*, qui peut être défini comme une simple fonction ou un programme. Afin de gérer un grand nombre de threads concurrents pouvant coopérer entre eux, l'architecture des cartes graphiques a introduit des ensembles de coopérations de threads (*cooperative thread array*, CTA), appelés également des blocs de threads ou *threads blocks* dans la terminologie CUDA.

Un bloc de threads est donc un ensemble de threads concurrents qui exécutent la même tâche et qui permet une coopération entre les threads d'un même bloc. Un bloc peut contenir de 1 à 512 threads (jusqu'à 1024 threads sur les cartes graphiques plus récentes). Chaque thread dispose de son propre et unique identifiant (TID pour *thread ID*), numéroté de 0 à m (m étant un entier naturel). Il est possible également de définir la dimension des blocs en 1D, 2D ou 3D. De même les TID peuvent être indicés en 1D, 2D ou encore 3D. Les threads d'un bloc peuvent partager des informations et des données dans la mémoire globale ou la mémoire partagée. A partir de ces indices (TID), un bloc peut sélectionner ces threads pour effectuer une opération spécifique et/ou partager les données en mémoire.

Enfin, chaque unité multi-coeurs (SM) peut exécuter jusqu'à 8 blocs simultanément, le nombre de blocs exécutés en parallèle dépendant de la demande et des ressources de chaque bloc. Finalement, comme représenté schématiquement sur la Fig. 1.9, on observe plusieurs niveaux de granularité de parallélisation.

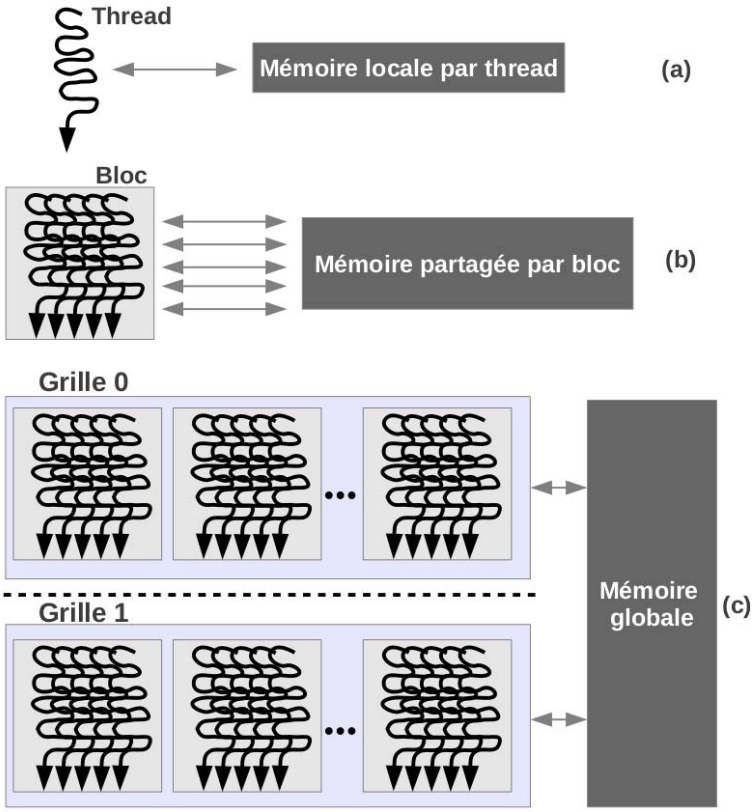


FIGURE 1.9 – Niveaux de granularité d'exécutions parallèles et hiérarchie mémoire : (a) Le thread et la mémoire privée local par thread, (b) le bloc de threads et la mémoire partagée par bloc, (c) les grilles de blocs et la mémoire globale.

Ces trois niveaux sont composés par :

- les threads : Ils effectuent les opérations sur les éléments sélectionnés et définis par leurs TID.
- Les blocs : Ils traitent les opérations à partir des résultats des threads de ce même bloc, sélectionnés par leurs indices de bloc (CTA ID). Les threads communiquant dans un bloc utilisent des instructions rapides de synchronisation, ce qui permet aux threads de lire les données écrites par d'autres threads du même bloc avant d'écrire en mémoire partagée ou en mémoire globale.
- Les grilles : Elles traitent les opérations à partir des résultats de plusieurs blocs. Les grilles dépendantes séquentiellement utilisent des barrières globales de synchronisation inter-grilles pour assurer un ordre global de lecture/écriture.

La même analogie peut être faite pour les niveaux de mémoires en lecture/écriture qui sont définis dans ce qui suit :

- La mémoire locale : Chaque thread actif à accès à une mémoire privée locale pour adresser temporairement des variables.
- La mémoire partagée : Chaque bloc actif a une mémoire partagée associée pour accéder aux données partagées par les threads de ce bloc.
- La mémoire globale : Les grilles traitées séquentiellement communiquent et partagent l'ensemble des données dans la mémoire globale.

En résumé, chaque processeur (coeurs SP), exécutant un thread, a un accès physique à plusieurs endroits mémoriels. Les threads d'un même bloc disposent de registres dont l'accès est directe (équivalent à un cycle d'horloge). Ces threads, au sein d'un même bloc, peuvent partager des données *via* la mémoire partagée dont l'accès est rapide, de l'ordre de $3-4$ cycles d'horloge, bien que l'espace mémoire soit relativement restreint ($\sim$ 48 Ko pour les cartes les plus récentes). Deux autres mémoires à accès rapides sont disponibles sur chaque multiprocesseur et les données dans ces mémoires sont cachées. La mémoire constante et la mémoire texture peuvent être accédées par tous les threads mais en lecture seule. Les temps d'accès à ces mémoires sont de l'ordre de la dizaine de cycles d'horloge. Enfin, La mémoire globale peut être accédée en lecture et écriture et est accessible par tous les threads. Bien que l'espace mémoire de la mémoire globale soit le plus important (de l'ordre de 3 Go pour les cartes Tesla C2050), les temps de latence sont cependant élevés et correspondent à environ $400-600$ cycles d'horloge.

Le programme CUDA exécute séquentiellement le code sur le CPU et traite parallèlement les kernels à travers un jeu de threads parallèles sur le GPU. Ainsi, l'application CUDA gère le GPU comme une unité de calcul (appelée *device* ou *périphérique* dans la terminologie CUDA) qui se comporte comme un coprocesseur du CPU (appelée *host* ou *hôte* dans la terminologie CUDA) avec son propre système de mémoire.

Le mode de programmation CUDA est similaire dans le style au modèle de programmation SPMD (*Single Program Multiple Data*), c'est à dire qu'il exprime explicitement le parallélisme et chaque kernel exécute un nombre fixe de threads. De plus, chaque appelle de kernels crée dynamiquement une nouvelle grille dans laquelle le nombre exacte de bloc de threads et le nombre de threads vont pouvoir être utilisés pour l'application des instructions du kernel.

Une syntaxe supplémentaire a été ajoutée comme extension du langage de programmation C/C++. Cette syntaxe s'apparente à la déclaration de mots clefs spécifiques tels que : **__global__** en entrée de fonction et qui permet de définir un kernel qui sera exé-

cuté sur le GPU, **__device__** pour définir des variables globales ou encore **__shared__** pour définir des variables de la mémoire partagées. La syntaxe d'un kernel est simplement équivalente à une fonction C pour un thread séquentiel. La définition de variables telles que **threadIdx.(x,y,z)** et **blockIdx.(x,y,z)** permet de fournir respectivement l'indice du thread (TID) appartenant au bloc d'indice **blockIdx.(x,y,z)**. L'indice du bloc correspond au numéro du bloc dans la grille. Ainsi, chaque thread possède un marqueur qui le représente (TID) et qui est utilisé pour travailler sur les adresses mémoires des tableaux ou des variables affectées sur le GPU (cf. Fig. 1.10).

Pour une liste de données disposées linéairement, les indices de chaque élément de cette liste peuvent être définis comme suit :

$$\text{Indice} = \text{threadIdx.x} + (\text{blockIdx.x} \times \text{blockDim.x})$$

Où les variables *threadIdx.x* et *blockIdx.x* sont représentées sur la Fig. 1.10 et où *blockDim.x* correspond à la taille d'un bloc, i.e. au nombre de threads contenus dans chaque bloc. Pour exploiter les performances du GPU, il est essentiel de répartir les tâches sur les grilles de blocs dont la taille doit être adaptée au problème traité afin d'optimiser l'utilisation des unités de calculs.

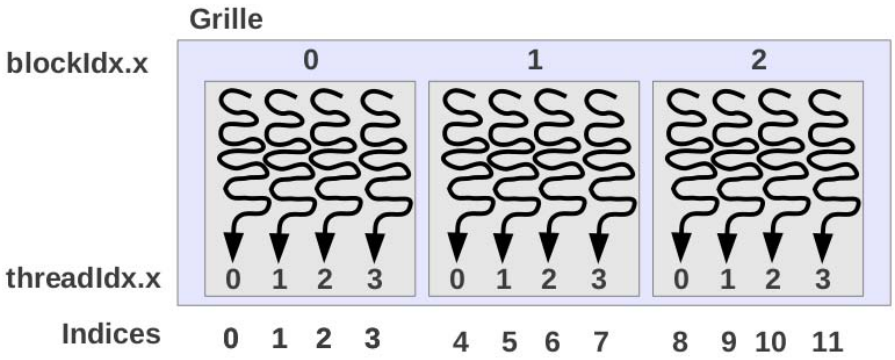


FIGURE 1.10 – Représentation des indices des threads et des blocs d'une grille dans un tableau en CUDA. *blockIdx.x* représente l'indice des blocs (1D) dans la grille et *threadIdx.x* correspond aux indices des threads dans chaque bloc. La dimension d'un bloc est caractérisée par le nombre de threads qu'il contient, soit ici $blockDim.x = 4$. L'indice des éléments est défini par la relation : indice = threadIdx.x + blockIdx.x × blockDim.x.

En résumé, pour exécuter des noyaux de calculs sur GPU (kernels), on doit systématiquement ajouter les instances **__global__** ou **__device__**. Lors de l'appel de ce noyau, on doit définir le nombre de blocs et le nombre de threads par bloc que l'on veux lancer et qui dépend du problème étudié. Les données sont réparties entre tous les threads, et sont regroupées dans des blocs au sein d'une grille. Les listing 1.1 et 1.2 montrent un exemple

simple d'un programme séquentiel en C et son équivalent dans le langage de programmation CUDA C. Le programme séquentiel C utilise deux boucles imbriquées pour itérer sur les indices de chaque tableau et calculer la somme de deux matrices de tailles $N \times N$, soit l'opération : **c[idx] = a[idx] + b[idx]**. Le programme CUDA C n'a pas de boucles. Il utilise les threads en parallèles pour calculer les mêmes indices des tableaux en parallèles et où chaque thread calcul seulement une opération de somme.

Listing 1.1– Programme C : Somme des éléments des tableaux a et b

```
void Sum (float* a, float* b, float* c, int N)
{
  int i, j, idx;
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      idx = i + j * N;
      c[idx] = a[idx] + b[idx];
    }
  }
}

void main ()
{
  ...
  Sum (a, b, c, N);
  ...
}
```

Listing 1.2– Programme CUDA C : Somme des éléments des tableaux a et b

```
__global__ void Sum (float* a, float* b, float* c, int N)
{
  int i = threadIdx.x + (blockIdx.x * blockDim.x);
  int j = threadIdx.y + (blockIdx.y * blockDim.y);
  int idx = i + j*N;

  if (i < N && j < N)
  {
    c[idx] = a[idx] + b[idx];
  }
}
```

```
13  void main ()
    {
15    ...
      dim3 dimBlock(blocksize, blocksize); /*Nombre de threads/bloc*/
17    dim3 dimGrid(N/dimBlock.x,N/dimBlock.y);/*' ' de blocs/grille*/

19    Sum <<<dimGrid, dimBlock>>> (a, b, c, N);
      ...
21  }
```

1.6 Conclusion

Tout au long de ce chapitre, nous avons constaté l'évolution importante des puissances de calculs et des microprocesseurs depuis leurs apparitions en 1971. La demande toujours croissante de ces technologies, dans un cadre professionnel tant que personnel, a fait naître un marché économique de grande ampleur où la concurrence entre les diverses sociétés stimule celles-ci dans la recherche, le développement et l'innovation de ces outils de hautes technologies.

Aujourd'hui, la communauté scientifique emploie largement ces puissances de calculs dans le cadre de simulations et de modélisations, dans le but d'une meilleure compréhension des phénomènes physiques. Cependant, les méthodes numériques utilisées demandent pour la plupart des temps de calculs et des ressources importantes. Ainsi, pour accroître significativement la puissance de calcul, des outils de parallélisation sont utilisés avec la mise en réseau de plusieurs CPU.

On a vu apparaître depuis maintenant une vingtaine d'années des processeurs spécialement dédiés à l'affichage et permettant de décharger le CPU des calculs liés aux graphismes et paraissant particulièrement bien adaptés pour les jeux vidéos. Ces processeurs graphiques ont révélé une capacité importante de calcul et pour un coût nettement inférieur par rapport aux autres solutions de supercalculateur. L'engouement récent de la communauté scientifique a donné naissance à un nouveau marché et une nouvelle utilisation des cartes graphiques.

L'architecture des GPU permet une utilisation en parallèle de centaines de processeurs et donc une parallélisation au niveau des données importante et pouvant être très efficace. Cette architecture reste cependant encore peu détaillée et son fonctionnement relativement complexe. C'est pourquoi, afin de mieux appréhender l'architecture GPU, il est intéressant de voir et de comprendre le fonctionnement des CPU, tout en mettant en avant les différences et les limitations de chacun.

En raison de l'engouement et au nouvel intérêt porté aux cartes graphiques, les deux grands constructeurs de cartes ATI et NVIDIA ont développé et mis à disposition un langage de programmation destiné aux calculs généralistes. Les deux langages de programmation développés sont OpenCl et CUDA.

OpenCl est un langage globalement similaire à celui de CUDA, bien que plus récent. Lors du commencement de la thèse, le choix fut fait d'utiliser le langage CUDA, et par conséquent les cartes NVIDIA, car celui-ci était bien plus développé et présenté des outils et des librairies permettant une meilleure utilisation de la carte graphique.

Ce langage, présenté précédemment dans la Section 1.5, est emprunté du langage C/C++ avec une syntaxe additionnelle permettant de définir à la compilation quelles fonctions et quelles instructions doivent être gérées par le GPU. De plus, la hiérarchie de la mémoire sur les cartes graphiques permet d'optimiser certaines instructions par l'utilisation de mémoires caches, telles que la mémoire texture ou la mémoire constante, ou encore la mémoire partagée, qui est la mémoire présente sur chaque multiprocesseur, de faible capacité de stockage mais ayant des temps d'accès très rapides.

Finalement, nous allons voir dans les différentes parties qui vont suivre, l'utilisation de ces cartes graphiques pour la simulation de plasmas froids. Un modèle 2D-3D a été développé et est présenté dans le chapitre suivant. On montre les capacités liées à cette architecture, avec par exemple un gain de temps de calcul important, mais aussi les limitations notamment au niveau de l'espace mémoire. Nous constaterons également les contraintes liées à la parallélisation elle-même. Enfin, nous verrons les résultats des simulations effectuées dans le cadre du projet ITER et de la modélisation de la source d'ions négatifs où une étude sur le transport électronique à travers une barrière magnétique sera présentée. Cette étude a été réalisée à partir du modèle à deux et trois dimensions et permet de mettre en valeur certaines différences observées lors du passage d'une dimension à une autre.

L'évolution continue des cartes graphiques, leur faible coût et leur capacité de calcul en font aujourd'hui un nouvel élément de calcul très performant voué à être encore amélioré et ouvrant peut être d'autres possibilités dans le domaine de la visualisation, de la parallélisation et de la modélisation.

Chapitre 2

Méthode PIC et Parallélisation sur GPU

2.1 La méthode Particle-In-Cell

La méthode Particle-In-Cell (PIC) est un très bon outil pour l'étude des plasmas et pour la simulation de trajectoires de particules chargées. Cette méthode est couramment utilisée en physique des plasmas de fusion, mais également dans l'étude des différents phénomènes intervenant dans les propulseurs plasmas (Réfs. [19, 20]), pour la simulation de la source d'ion dans le cadre du projet ITER (Réfs. [21, 22, 23, 24]), dans l'étude des phénomènes de l'environnement spatial tels que les vents solaires, les environnements atmosphériques (magnétosphère, ionosphère, ...), les interactions laser-plasmas ... (Réfs. [25, 26, 27, 28, 29, 30, 31]). Cette liste n'est évidemment pas exhaustive et pour plus de détails sur cette méthode, deux ouvrages essentiels sur la méthode PIC sont également recommandés aux lecteurs (Réfs. [32, 33]).

Les particules chargées, représentées dans un plasma par les électrons (espèce chargée négativement) et les ions (chargée positivement), interagissent les unes avec les autres par attraction pour les espèces ayant des charges différentes et par répulsion pour des mêmes charges. La force qui s'exerce entre ces charges est la force de Coulomb et est donnée par :

$$\mathbf{F}_{12} = \frac{1}{4\pi\epsilon_0}\frac{q_1 q_2}{r^2}\mathbf{r}_{12} \tag{2.1}$$

Où $\mathbf{F}_{12}$ est la force exercée par la particule de charge q_1 sur la particule de charge q_2, séparées par une distance r et ϵ_0 la permittivité du vide. Néanmoins, la simulation de plasma requiert un nombre de particules évoluant dans le système très important, de l'ordre du million de particules. Cela signifie que si l'on veut calculer les forces agissant

sur toutes les particules, pour un nombre total n de particules, on serait amené à effectuer n^2 opérations pour chaque itération dans le temps. Soit pour un nombre de particule $n = 1 \times 10^6$, le nombre d'opérations revient à 1×10^{12}. Sachant qu'une simulation demande généralement plusieurs milliers de pas en temps, le nombre et le temps de calcul augmentent considérablement et deviennent non envisageables sur les ordinateurs actuels.

La méthode PIC simule également des particules chargées discrètes représentant les électrons et les ions. Cependant, la force agissant sur les particules est obtenue cette fois-ci par le calcul du champ électrique et magnétique résultant des champs appliqués et de la présence des charges et des courants au sein du plasma. Cette force est la force de Lorentz et est donnée par :

$$\begin{aligned} F(\mathbf{r}) &= F_{electrique} + F_{magnetique} \\ &= q(E(\mathbf{r}) + \mathbf{v} \times B(\mathbf{r})) \end{aligned} \tag{2.2}$$

avec q la charge des particules, $\mathbf{v}$ et $\mathbf{r}$ sont respectivement les vitesses et les positions des particules dans le plan cartésien (x, y, z). Le calcul des forces à partir de la relation ci-dessus (cf. Eq. (2.2)) réduit le nombre d'opérations à n opérations. Ceci rend cette méthode plus "attractive" en terme de calcul par rapport à la méthode précédente bien que les temps de calcul restent relativement élevés dans le cas de densité élevées et/ou de tailles de systèmes assez larges comme nous le verrons dans la partie concernant les limites de validité du modèle PIC.

Il est important de noter que les densités de plasma atteignent assez vite des densités supérieures à $10^{16} - 10^{18}$ m^{-3} (de l'ordre de 10^{18} m^{-3} dans le cas de la source d'ions négatifs pour ITER) et donc un nombre *réel* d'électrons et d'ions très élevé (environ 10^{18} pour la source d'ions négatifs et pour **chaque** espèce chargée). C'est la raison pour laquelle, le nombre de particules simulé dans la méthode PIC, ne correspond pas au nombre réel de particule. On appelle *macro-particules*, les particules discrètes de la simulation qui sont définies comme un regroupement de particules *réelles* chargées de la même espèce. A ces macro-particules on affecte un poids équivalent à :

$$W_{macro} = \frac{D_{plasma} \times V_{syst}}{N_{macro}} \tag{2.3}$$

Où W_{macro} correspond au poids des macro-particules, D_{plasma} est la densité de plasma initiale, V_{syst} est le volume du domaine de simulation et N_{macro} est le nombre total de macro-particules initial.

Le mouvement des macro-particules est gouverné par les lois de la dynamique de Newton

couplées aux interactions du champ électrostatique et du champ magnétique. Ces équations sont données par :

$$m\frac{d\mathbf{v}}{dt} = F(\mathbf{r}) \ , \quad \frac{d\mathbf{r}}{dt} = \mathbf{v} \tag{2.4}$$

Où m est la masse des particules et $F(\mathbf{r})$ et la force de Lorentz agissant sur les particules (cf. Eq. (2.2)).

Ce système d'équations est résolu à chaque itération dans le temps connaissant la vitesse et la position de chaque particule. Cependant, dans cette thèse nous nous intéressons au cas purement électrostatique et supposons que le champ magnétique créé par les courants de particules est négligeable. Toutefois, un champ magnétique extérieur peut être appliqué. Ainsi, le calcul des forces sur le maillage (discrétisation spatiale du domaine de simulation) se restreint au calcul du champ électrostatique défini par :

$$\mathbf{E} = -\nabla\phi \tag{2.5}$$

avec ϕ, le potentiel électrique qui est donné par l'équation de Poisson : $\nabla^2\phi = -\frac{\rho}{\epsilon_0}$, où ϵ_0 est la permittivité dans le vide et ρ correspond à la densité de charges décrite comme $\rho = e(Z_i n_i - n_e)$. Ici, les indices i, e dénotent les ions et les électrons respectivement, n la densité de particules chargées, avec e la charge électronique et Z_i le nombre de charges ioniques.

Plusieurs étapes sont donc nécessaires au calcul des positions et des vitesses, à chaque pas de temps. Ces différentes étapes sont représentées Fig. 2.1. Nous verrons de façon plus explicite dans ce qui va suivre que ce schéma requiert certaines conditions pour rester valide.

La masse des électrons est beaucoup plus petite que celle des ions et dans le référentiel des ions, le déplacement des électrons semble instantané. C'est pourquoi, pour la résolution du mouvement des particules, les temps caractéristiques de la simulation sont définis à partir de la fréquence électronique ω_p qui conduit à des pas en temps très petits (de l'ordre de 10^{-10} secondes, pour des densités de 10^{14} m^{-3}). Ainsi, les temps de calculs deviennent vite très importants, ajouté au fait qu'un grand nombre de particules est nécessaire et qu'un maillage du domaine de simulation doit être suffisamment fin pour réduire au maximum toute erreur numérique.

Nous proposons de voir dans les sections suivantes, pour chacune des parties représentées Fig. 2.1, une définition générale des algorithmes utilisés et une implémentation pour la parallélisation sur GPU en essayant de mettre le plus possible en avant les avantages et les

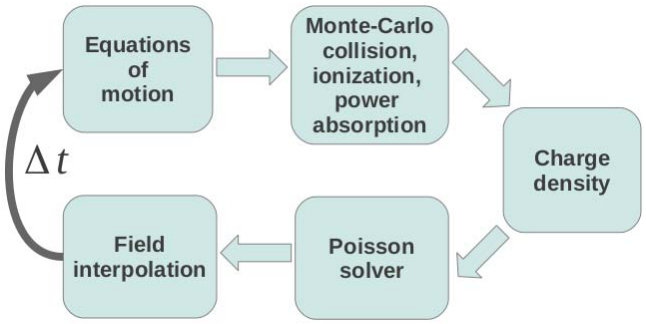


FIGURE 2.1 – Schéma représentatif d'un cycle PIC

inconvénients inhérents aux différentes parties du code PIC MCC.

Enfin, cette description est principalement détaillée pour le code PIC MCC 2D. Concernant les algorithmes du code PIC MCC 3D, nous mettrons seulement en avant les points qu'il semble important de décrire ou les problèmes rencontrés lors du passage à la 3D. D'une façon générale, nous insisterons sur les outils utilisés qui peuvent changer de la 2D à la 3D, notamment en lien avec la limitation de la mémoire sur le GPU et la gestion des blocks et des threads en parallèle.

2.2 Le modèle PIC et le GPU : Pas à pas...

2.2.1 Introduction

L'implémentation de codes PIC parallélisés sur GPU est relativement récente et certains travaux ont été entrepris par plusieurs groupes de recherches (cf. Réfs. [34, 35, 36, 37, 38]). Parmi ces travaux, plusieurs approches ont été proposées concernant notamment la gestion de la mémoire, la gestion de threads ou encore, en proposant de nouveaux algorithmes spécifiques à la parallélisation sur GPU. Ces études ont été réalisées dans le but de définir au mieux les algorithmes et les méthodes de parallélisation les plus efficaces possible pour les différentes parties de la simulation PIC ou pour des outils utilisés dans ce genre de modèle, tel que pour la résolution de l'équation de Poisson (Réfs. [39, 40]).

C'est dans ce contexte général de parallélisation et d'optimisation du code PIC MCC sur GPU à 2D et 3D que s'inscrit la majeure partie des travaux réalisés au long de cette thèse.

Comme dans ce qui a été énoncé précédemment, la gestion de la mémoire tient une part très importante dans l'implémentation et l'optimisation sur le GPU. Nous avons pu constater dans la section 1.5 que les temps d'accès à la mémoire RAM ou à la mémoire partagée (*shared memory*) sont très différents et que certaines caractéristiques spécifiques

du GPU, telle que la mémoire texture, peuvent être utilisées entre autre pour l'exécution d'interpolations linéaires. Un autre aspect important pour une utilisation efficace et intensive des capacités du GPU sont l'indépendance des données. Autrement dit, le fait que le résultat d'un thread ne dépende pas de celui d'un autre thread. Cet aspect est un aspect majeur car il représente la différence même entre le CPU et le GPU et impose une programmation que l'on pourrait appelée de *"prudente"*, c'est à dire que certains algorithmes communément utilisés sur CPU ne sont plus valable sur GPU comme nous le verrons dans la Section 2.4.

De l'analyse des algorithmes et de la dépendance des données dépendra le niveau de parallélisation et l'orientation [1] des différentes méthodes proposées pour l'implémentation du code PIC MCC sur GPU. Nous verrons à partir de la section 2.3 que l'indépendance des données est naturellement satisfaite dans certaines parties du code PIC car chaque particule présente évolue indépendemment des autres, mais nous verrons également que des dispositions doivent être prises en compte afin de prévenir la mauvaise gestion de la mémoire et la corruption de données, qui sont principalement dues à l'écriture simultanée de différents threads dans une même banque mémoire (qu'on appelle aussi *"race conditions"*).

L'adaptation et l'optimisation des algorithmes sur les architectures GPU peuvent être opérées de plusieurs manières. Après avoir mis en revue et décrit les différentes méthodes et algorithmes existant, je propose dans les parties qui suivent, une adaptation des algorithmes pour chaque routine présente dans la méthode PIC MCC appliquée aux plasmas à basse température.

2.2.2 Remarques sur les algorithmes actuels

Sur GPU, il est important de comprendre que la structure du code PIC, la façon dont il va être implémenté ou encore les algorithmes qui vont être utilisés sont entièrement définis et caractérisés par la méthode employée pour le calcul de la densité de charges sur les noeuds de la grille. Ceci vient principalement du fait que la parallélisation du calcul des charges sur la grille, i.e. l'ajout des contributions de chaque particule sur les noeuds de la cellule où elle se trouve, est le point faible majeur de la parallélisation de ce type de code, l'efficacité et l'utilisation intensive des architectures GPU peuvent en être fortement réduites.

Ce point est développé par G. Stantchev et al.[41] dans la méthode pour l'optimisation de l'interpolation des charges sur la grille, et est discuté de manière plus détaillée dans la section 2.4. Cette méthode est la première à avoir été proposée mais elle reste néanmoins complexe à mettre en oeuvre dans le cadre de son optimisation. Elle met en avant le fait que les algorithmes classiquement utilisés ne sont plus du tout avantageux pour la parallélisation

1. Dans notre cas, une parallélisation au niveau des particules ou au niveau des cellules/noeuds

sur GPU et que celles paraissant les mieux adaptées pour ce type d'architecture requièrent l'utilisation d'un algorithme de tri qui reste très coûteux et difficilement optimisable.

Basé sur ces travaux, P. Mertmann et al [35] proposent une description d'un code PIC 1D dont l'organisation des particules au sein des cellules évite l'utilisation d'un tri de toutes les particules à chaque pas de temps. Les cellules, appelées *SuperCells*, sont un regroupement de plusieurs cellules du maillage, adjacentes les unes par rapport aux autres. Les particules sont regroupées par ordre d'appartenance aux *SuperCells*. A chaque pas en temps, on vérifie si la particule a changé de cellule ou non. L'avantage dans un tel cas est que l'on gère seulement les particules qui changent de cellule et/ou de superCell à chaque pas de temps. Cet avantage reste valable si l'on suppose que la dynamique des particules modélisées dans les codes PIC vérifie que la majorité d'entre elles restent dans la cellule et que seule une petite partie dérive vers les cellules adjacentes. Le désavantage de cette méthode est qu'elle permet difficilement le passage d'une simulation 1D à une simulation 2D et qu'elle ne permet pas d'avoir des distributions de densités fortement inhomogènes dans le domaine de simulation.

Dans leur article [34], P. Abreu et al décrivent un code particulaire 2D, où contrairement à P. Mertmann et al., toutes les parties du code PIC sont parallélisées sur la carte graphique. La méthode employée pour le calcul des charges sur les noeuds de la grille, appelée calcul *"PseudoAtomic"*, utilise les opérations atomiques (cf. Annexe A.1). Autrement dit, les opérations sont exécutées sans interférences entre les threads tout en évitant des corruptions de mémoires (*race conditions*). Le terme *pseudo* vient du fait que pour éviter d'appeler plusieurs fois les fonctions atomiques (deux appels de fonctions atomiques par particules et par cellules), un algorithme de tri est utilisé. Ce tri est réalisé de manière à ranger les particules et dans le but de minimiser les chances que deux particules indexées consécutivement dans le tableau n'appartiennent pas à la même cellule.

Comme expliqué précédemment, l'algorithme de tri est très coûteux, et l'appel aux fonctions atomiques peut l'être également (cf. Annexe A.1). C'est pourquoi, l'efficacité de la parallélisation du code PIC se heurte une fois de plus au problème de l'interpolation des charges sur la grille. Dans son article, P. Abreu montre que près de 70% du temps de calcul est passé dans le tri et le calcul des charges sur les noeuds de la grille.

Finalement, dans le but de pouvoir passer relativement facilement d'un code 2D à un code 3D, il semblait préférable de garder au maximum une convention de programmation classique pour les différentes parties du code PIC MCC. Afin d'éviter le plus possible les transferts de données entre le CPU et le GPU, il paraissait également judicieux et dans la limite du possible, d'implémenter toutes les parties sur le GPU. Lors de l'implémentation du code PIC MCC et encore actuellement, une des méthodes qui semble la plus propice et

avec laquelle on peut gagner en efficacité de calcul en ce qui concerne l'interpolation des charges sur les noeuds du maillage, est la méthode proposée par G. Stantchev et al. [41].

Dans ce qui suit, je présente les algorithmes utilisés pour l'implémentation du code PIC. De plus, je montre certaines différences entre les méthodes déjà proposées et celles que j'utilise. Enfin, des comparaisons de temps de calcul entre la simulation sur CPU et sur GPU ont été effectuées et sont présentées dans la dernière partie de ce chapitre.

2.3 Transport des particules

Dans les simulations PIC, la discrétisation des équations du mouvement est basée sur la résolution des équations différentielles ordinaires de *Newton-Lorentz*. L'algorithme le plus couramment utilisé et l'un des plus simples est celui connu sous le nom de *saute-mouton* (en anglais, *leap-frog*). Ainsi, la position et la vitesse des particules sont intégrées sur un pas en temps en suivant le schéma Fig. 2.2 :

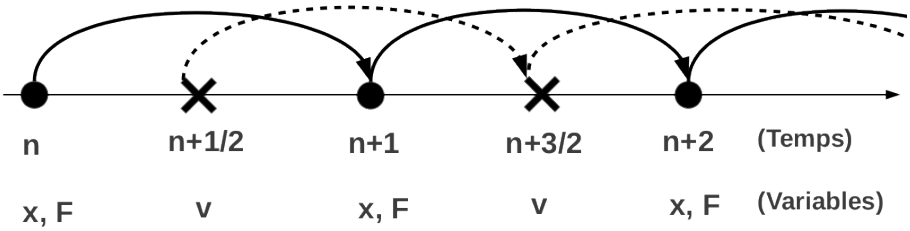


FIGURE 2.2 – Schéma représentatif de l'algorithme saute-mouton (leap-frog). La discrétisation en temps est échelonnée à chaque demi pas en temps avec le calcul de la force et des positions des particules (F,x) et le calcul des vitesses (v).

La position des particules et le champ électrostatique sont définis au pas de temps t_i et sont utilisés pour calculer la force de Lorentz (cf. Eq. 2.2) agissant sur les particules. La force électrostatique est interpolée à la position de la particule à partir des valeurs obtenues sur la grille. L'interpolation linéaire requiert les deux points du maillage les plus proches par rapport à la position de la particule, dans chaque direction. Nous verrons dans la section concernant l'interpolation des champs électrostatiques et magnétiques (cf. Section 2.6), que les points nécessaires pour cette interpolation passent de deux points en 1D à quatre points en 2D et enfin à huit points en 3D.

L'avancement des particules au pas de temps t_{i+1} se fait en connaissant les vitesses calculées au pas de temps $t_{i+\frac{1}{2}}$. Le schéma utilisé pour le calcul des vitesses est appelé algorithme de Boris (Réfs. [32, 42]). C'est un schéma numérique centré en temps, appliqué dans de nombreuses simulations PIC [33]. Il se trouve être particulièrement bien adapté pour les codes PIC. Cette technique sépare les effets du champ électrostatique et du champ

magnétique en plusieurs étapes, au temps $t_{i-\frac{1}{2}}$. Ces étapes consistent en une demie accélération due au champ électrostatique, une rotation due au champ magnétique et une deuxième demie accélération due au champ électrostatique.

En considérant les vitesses calculées au pas de temps $t_{i+\frac{1}{2}}$, les nouvelles positions des particules peuvent être calculées. Le schéma saute-mouton est donné par les relations suivantes :

$$\begin{aligned} x_{t_{i+1}} &= x_{t_i} + v_{t_{i+\frac{1}{2}}} \delta t \\ v_{t_{i+\frac{1}{2}}} &= v_{t_{i-\frac{1}{2}}} + \delta t \; F_{t_i} \end{aligned} \tag{2.6}$$

Où F_{t_i} est la force interpolée à la position de la particule au début du pas de temps t_i. x et v sont respectivement la position et la vitesse de la particule et δt correspond à l'intervalle de temps compris entre t_i et t_{i+1}.

2.3.1 Implémentation sur GPU

Les cartes graphiques actuelles permettent de lancer en parallèle un nombre relativement important de threads. La solution la plus simple et la plus évidente au premier abord, pour la parallélisation des particules, est d'affecter un thread par particule. Ceci signifie qu'il est possible d'avoir autant de threads que de particules pour le calcul des nouvelles positions à chaque pas de temps. Cependant, il existe d'autres possibilités, telle que celle utilisée dans l'article de P. Abreu (Réf. [34]), où l'on définit des groupements de particules et où chaque groupe est administré par un seul thread. Ceci peut avoir un avantage dans le cas d'un nombre de particules élevé et peut permettre de mieux gérer l'occupation mémoire, l'utilisation des registres (mémoire à accès directe) ou encore la bande passante.

Enfin, dans le but d'obtenir une occupation maximale du GPU, il a été estimé que le nombre de threads par block est de l'ordre de 64 ou de 128 threads. Ceci est également corroboré par plusieurs travaux de simulations PIC sur GPU (cf. Réfs. [34, 35, 36]). Ce nombre reste néanmoins dépendant de l'usage de la mémoire, du type de carte graphique utilisé et de la variation du nombre total de particule lors de la simulation. Ces nombres ne doivent donc pas être considérés comme invariants et dépendent sensiblement du problème et des outils utilisés.

Dans certaines simulations, lorsqu'il n'y a pas de créations ou de pertes de particules chargées (e.g. dans le cas de conditions limites périodiques et où les réactions d'ionisation n'interviennent pas), chaque particule possède son propre indice durant toute la simulation et leur suivi, i.e la façon d'ordonner les particules dans le tableau, ne pose pas de problème significatif.

Au contraire, lorsqu'il y a création de particules chargées (e.g. lors de collisions ionisantes) ou des pertes (e.g. absorptions par les électrodes ou les parois du diélectrique, processus d'attachement, ...), les indices des particules dans le tableau doivent être réorganisés à chaque pas de temps. Les pertes de particules induisent une diminution du nombre total de particules et font apparaître des "trous" dans les tableaux de positions et de vitesses, i.e. des emplacements mémoire du tableau se libèrent. Le but de la méthode présentée ici est de réordonner les particules dans le tableau de telle sorte que les particules toujours présentes dans le système soient rangées de façon consécutive. Dans le cas des créations de particules, la méthode consiste juste à déposer les nouvelles particules à partir du dernier élément de la liste des positions et des vitesses et est présentée dans la section 2.7. Dans notre cas, la perte des particules à lieu seulement au niveau des parois du domaine de simulation. La méthode suivante peut être utilisée de la même façon pour d'autres types de pertes.

Lors du réarrangement en parallèle des particules dans les tableaux (de positions et de vitesses), à chaque pas en temps et afin d'éviter l'attribution d'un même indice à deux particules différentes, on utilise des opérations de préfixages couramment appelées *"Prefix Sum"* ou encore *"Scan"* (cf. Réf. [43]). Cette méthode est décrite dans ce qui suit.

Après que les particules aient été déplacées au cours d'un pas de temps δt donné, les positions de chacune d'entre elles dans le domaine de simulation sont vérifiées puis une valeur indiquant si la particule réside toujours dans le système ou non est inscrite dans un tableau que l'on nomme ici *"pfx"*. Une valeur de 1 est assignée à la particule si celle-ci est toujours dans le domaine de simulation, sinon on lui assigne la valeur nulle.

$$\mathrm{pfx}[i] = [a_0, a_1, \ldots, a_{n-1}] \text{ avec } \begin{cases} a_i = 0 \text{ ou } 1 \text{ et } i = 0, \ldots, n \\ n = \text{nombre total de particules} \end{cases} \tag{2.7}$$

Les valeurs du tableau de préfixes (*pfx*) sont, dans une seconde étape, utilisées afin d'effectuer la somme de ces préfixes. Cette étape consiste à incrémenter la valeur du i^{eme} élément du tableau par la valeur de l'élément $i-1$. Ainsi, si la particule réside dans le domaine de simulation, la valeur est incrémentée de 1, sinon elle ne change pas. Cette somme peut être schématisée par la relation donnée ci-dessous :

$$\mathrm{pfxSum}[i] = [a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1})] \text{ avec } i = 0, \ldots, n$$

Où $\oplus$ correspond à l'opérateur d'addition binaire (opérateur d'addition en Mathématique). NVIDIA fournit un algorithme de *scan* optimisé qui peut être utilisé pour réaliser

cette opération appelée *"Prefix Sum"* (cf. Réf. [43]).

Comme présenté dans la figure 2.3, par l'opération de préfixage puis la somme des préfixes (e.g. opération de *scan*), les nouveaux indices des particules se déduisent facilement du tableau *pfx* et *pfxsum*.

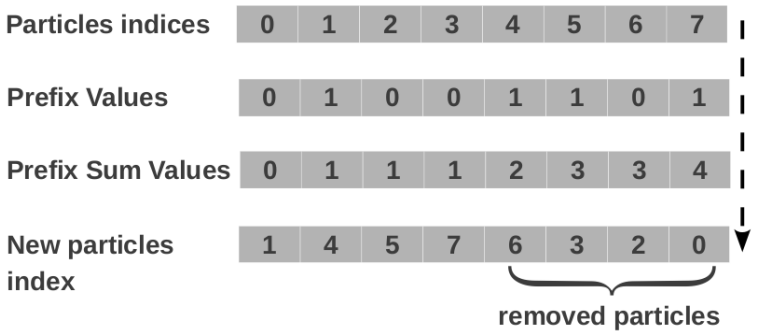


FIGURE 2.3 – Illustration de l'opération de réarrangement des particules (les 0s dans le tableau de Prefix correspondent aux particules situées à l'extérieur du domaine de simulation).

La nouvelle position des particules dans le tableau, i.e. le nouvel indice de la particule i, correspond à la valeur de l'élément i du tableau *pfxSum*. Ceci n'est vrai que pour une valeur non nulle de l'élément i du tableau *pfx*.

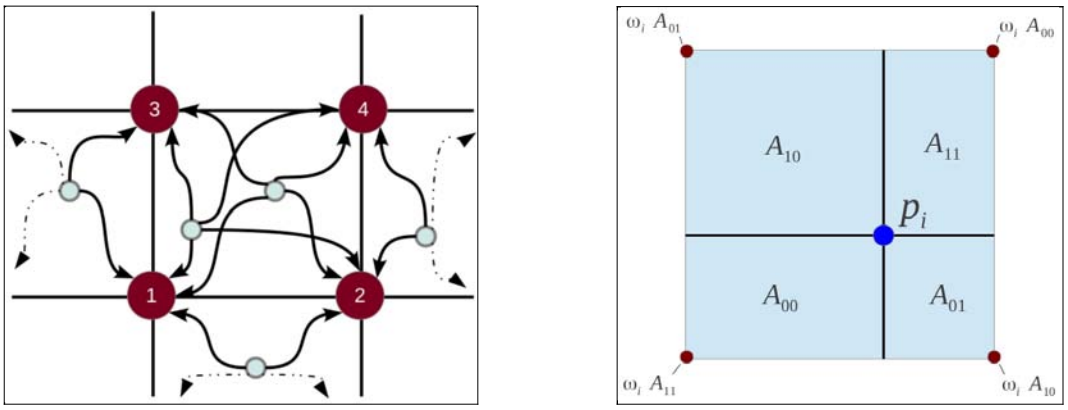
Listing 2.1– Kernel : changement d'indices

```
1
for (chaque particule Pi)
3 {
    int index = pfxSum[i];   /* Nouvel indice                        */
5   if (pfx[i] == 1)         /* pour les particules dont la valeur */
    {                        /* de pfx[i] = 1                        */
7     NewPos[index-1] = pos[i];
      NewVel[index-1] = vel[i];
9   }
  }
11 Npart = pfxSum[n-1] /* Nombre de particules restant */
                       /* avec (n-1) = dernier element de la liste */
```

Finalement, toutes les particules quittant le domaine peuvent être retirées du tableau de particules ou indexées à la fin de celui-ci. De plus le nombre de particules restant est également déduit par la lecture du dernier élément du tableau *pfxsum*. Un algorithme simplifié est présenté List. 2.1.

2.4 Densité de charge

Dans les simulations PIC, les particules chargées doivent être assignées sur une grille à chaque itération en temps dans le but de calculer la distribution spatiale de charge. Une grille rectilinéaire est généralement utilisée afin d'obtenir une représentation discrétisée du champ sur le domaine de simulation par le biais de la résolution de l'équation de Poisson. La stratégie couramment employée dans les simulations PIC consiste à interpoler la charge des particules sur les noeuds de la grille. Au pas de temps δt, connaissant la position de chaque particule, on en déduit la cellule dans laquelle elle se trouve. La charge de chaque particule dans une cellule est distribuée sur les quatre noeuds (en 2D) définissant la cellule (schématiquement représenté Fig. 2.4(a)), en tenant compte de la distance entre la particule et les noeuds.



a) b)

FIGURE 2.4 – (a) Assignement de charges sur les noeuds. Une particule dans une cellule contribue à la densité aux 4 noeuds définissant la cellule (en 2D), (b) Illustration de l'interpolation linéaire de la densité par la formule des *"aires opposées"*. Chaque A_{ij} multiplié par le coefficient de poids w_i correspond à la contribution de la particule p_i pour la fonction de densité de particule au noeud sur la diagonale opposée.

2.4.1 Méthode d'interpolation

De manière générale, l'opération d'interpolation d'un ensemble de particules $\{p_1, p_2, \ldots, p_{N_T}\}$ sur la grille correspond au calcul de la fonction de distribution de la densité sur une grille rectilinéaire uniforme, dont la valeur de la fonction de distribution de densité f à chaque noeud n_s est donnée par :

$$f(n_s) = \sum_{i=1}^{N_T} w_i K(n_s, p_i) \tag{2.8}$$

Où **s** est un indice multiple[2], w_i sont des coefficients de poids des particules et K correspondent à la fonction d'interpolation, i.e. à un facteur de forme.

À l'ordre 0 le facteur de forme K à deux dimensions est défini comme :

$$K^0((0,0),(x,y)) = \begin{cases} 1, & \text{si } |x| < \frac{1}{2}, \text{ et } |y| < \frac{1}{2} \\ 0, & \text{sinon} \end{cases} \tag{2.9}$$

où l'on assigne la charge de la particule au noeud le plus proche. Cette méthode est plus généralement appelée *Nearest Grid Point (NGP).*

À l'ordre 1 le facteur de forme K à deux dimensions correspond à une interpolation linéaire définit comme :

$$K^1((0,0),(x,y)) = \begin{cases} (1-|x|)(1-|y|), & \text{si } |x| < 1, \text{ et } |y| < 1 \\ 0, & \text{sinon} \end{cases} \tag{2.10}$$

Dans la plupart des applications, K est une fonction d'interpolation linéaire qui, à une dimension, est représentée par une fonction *"chapeau"*. L'utilisation de ce type d'interpolation linéaire implique que pour chaque noeud n_s, seules les particules contenues dans des cellules incidentes à n_s verront leurs contributions sommées dans l'Eq. (2.11) :

$$f(n_s) = \sum_{p_i \in \mathcal{P}(n_s)} w_i K(n_s, p_i) \tag{2.11}$$

Où $\mathcal{P}(n_s)$ représente l'ensemble des particules contenues dans la cellule incidente au noeud n_s. Cette expression peut être interprétée géométriquement par la mesure Euclidienne du volume *opposé* au noeud n_s par rapport à la particule p_i. Soit à deux dimensions, la contribution de la particule p_i peut être calculée par la formule des *aires opposées* Eq. (2.29) et représentée schématiquement Fig. 2.4(b).

2. Soit un domaine de dimension d, s est définit comme : $s = \{s_1, s_2, \ldots, s_d\}$

$$
\begin{aligned}
n_{i,j} &= \frac{w_i}{V_{X,Y}} \underbrace{(X_{i+1} - x)\,(Y_{j+1} - y)}_{A_{11}} \\
n_{i+1,j} &= \frac{w_i}{V_{X,Y}} \underbrace{(x - X_i)\,(Y_{j+1} - y)}_{A_{10}} \\
n_{i,j+1} &= \frac{w_i}{V_{X,Y}} \underbrace{(X_{i+1} - x)\,(y - Y_j)}_{A_{01}} \\
n_{i+1,j+1} &= \frac{w_i}{V_{X,Y}} \underbrace{(x - X_i)\,(y - Y_j)}_{A_{00}}
\end{aligned} \tag{2.12}
$$

Où $V_{x,y}(= \Delta_X \Delta_Y)$ est le volume de la cellule , (X, Y) sont les coordonnées de la cellule et (x, y) les coordonnées de la particule p_i.

2.4.2 Implémentation sur GPU

La stratégie la plus utilisée pour l'affectation des charges sur les noeuds est celle où pour chaque particule, on affecte le poids correspondant aux noeuds de la cellule à laquelle elle appartient. Soit $\mathcal{N}(p_i)$ l'ensemble des noeuds de la cellule dans laquelle la particule p_i se trouve. Le pseudo code est donné dans le List. 2.2.

Listing 2.2– Kernel : Affectation des charges sur les noeuds : strategie 1

```
/* Initialisation de f a 0 */
for (chaque noeud n_s ∈ domaine de simulation)
{
  f(n_s) ⟵ 0
}
/* Boucle sur toutes les particules p_i */
for (chaque particule p_i ∈ domaine de simulation)
{
  /* trouver N(p_i) */
  get_nodes();
  for (chaque noeud n_s ∈ N(p_i))
  {
    f(n_s) = f(n_s) + w_i K(n_s, p_i);
  }
}
```

Bien que cette stratégie ne soit pas la plus efficace en terme du nombre d'opérations à effectuer [3], elle est cependant facile d'approche et l'ensemble des noeuds $\mathcal{N}(p_i)$ peut être calculé dynamiquement à partir des coordonnées des particules.

En parallèle, lorsque ces opérations (cf. List. 2.2) sont effectuées, il est possible que la charge de deux particules différentes (p_i, p_j) soit affectée simultanément par deux threads concurrents sur le même noeud. Deux inconvénients majeurs dans cette méthode apparaissent lors de la parallélisation sur GPU. Premièrement, comme décrit précédemment, il peut y avoir une redondance des données entre les noeuds n_s des ensembles $\mathcal{N}(p_i)$ et $\mathcal{N}(p_j)$ pouvant donc donner place à des collisions en mémoire (*race conditions*). Deuxièmement sur les GPU, l'accès aléatoire en mémoire (*random-memory access* en anglais) se révèle être un sérieux problème pour le gain de performance. Autrement dit, pour chaque indice de particule et chaque indice de noeud, les éléments des ensembles $\mathcal{N}(p_i)$ et $\mathcal{P}(n_s)$ ne seront (en principe) pas placés de façon contiguës en mémoire. On va dès lors autoriser des accès non-coalescents qui augmentent le temps de latence pour les opérations de lecture/écriture en mémoire globale.

Une solution directe peut être apportée par l'utilisation des opérations atomiques interdisant les collisions mémoires. Cependant, du fait de la sérialisation, ce type d'opération réduit de façon non négligeable les performances de temps de calcul (cf. Annexe. A.1).

Une approche développée par G. Stantchev et al.[41] décrit un algorithme efficace d'interpolation sur une grille, spécifiquement conçu pour la parallélisation sur GPU. Contrairement à la stratégie montrée précédemment (cf. List. 2.2) où les opérations sont effectuées à partir des ensembles $\mathcal{N}(p_i)$ en itérant sur les particules p_i, cette autre approche se base sur les ensembles $\mathcal{P}(n_s)$ où l'itération se fait sur les noeuds. Un pseudo code décrit la structure de l'algorithme List. 2.3.

Listing 2.3– Kernel : Affectation des charges sur les noeuds : strategie 2

```
1  /* Boucle sur les noeuds n_s */
   for (chaque noeud n_s ∈ grille de simulation)
3  {
     /* trouver P(n_s) */
5    get_particles();
     /* Initialisation de f */
7    f(n_s) ⟵ 0
     for (chaque particule p_i ∈ P(n_s))
9    {
       f(n_s) = f(n_s) + w_i K(n_s, p_i);
```

3. Soit N_T le nombre total de particule et d la dimension du système, le nombre d'opérations est : $\mathcal{O}((2^d + 1)N_T)$

```
11      }
    }
```

L'avantage de cette méthode vient du fait que le nombre d'opérations [4] est plus petit que dans la méthode précédente. Cependant, trouver l'ensemble des particules $\mathcal{P}(n_s)$ incidentes aux noeuds n_s peut s'avérer complexe et très onéreux. De plus, la mémoire étant limitée sur les cartes graphiques, il est fort désavantageux d'organiser les particules en des ensembles $\mathcal{P}(n_s)$. Cette organisation fait apparaître des redondances au niveau des données sur les particules (en 2D une particule est associée à quatre noeuds différents, contre huit en 3D) et augmente donc de $\mathcal{O}(2^2 N_T)$ l'utilisation de l'espace mémoire.

Considérons maintenant un ensemble $\mathcal{P}(c_s)$ correspondant à l'ensemble des particules p_i contenu dans la cellule c_s (d'indice s) au lieu de l'ensemble $\mathcal{P}(n_s)$. Contrairement aux noeuds, une particule ne peut être associée qu'à une seule cellule. Cette astuce permet ainsi d'éviter la redondance de donnée sur les particules. En outre, si l'on choisit de prendre chaque cellule indépendemment les unes des autres, on multiplie par quatre le nombre de noeuds (en considérant une grille bidimensionnelle, un noeud est adjacent à quatre cellules) ce qui est nettement moins contraignant pour l'espace mémoire utilisé que dans le cas présenté List. 2.3.

De plus, il est possible de s'affranchir de l'inefficacité à trouver les particules de l'ensemble $\mathcal{P}(c_s)$ en organisant les particules de telle façon que pour chaque cellule c_s l'ensemble $\mathcal{P}(c_s)$ occupe un espace contigu en mémoire. Ceci est effectué par une opération de tri sur les particules, i.e. en déterminant pour chaque cellule c_s l'ensemble des particules associé. Divers algorithmes optimisés de tri basés sur l'algorithme *radix sort*, ont été développés sur GPU (cf. Réfs. [44, 45, 46]). Cependant, faire un tri sur toutes les particules à chaque pas de temps du code PIC est très coûteux et augmente considérablement le temps d'exécution du code. Les résultats de temps calcul pour les différentes parties du code PIC sont présentés dans la section 2.9. La partie concernant l'interpolation des contributions des particules sur la grille est de l'ordre de 40% du temps d'exécution global. En fonction du nombre N_T de particules à trier, l'algorithme de tri peut être jusqu'à plusieurs ordres de grandeurs supérieur au reste du temps de calcul de la densité. Une stratégie peut être utilisée pour éviter de trier toutes les particules à chaque pas en temps. Considérant qu'un ensemble restreint de particules traverse une cellule durant le temps δt, il est possible de faire un tri partiel que sur les particules qui ont changé de cellules.

Finalement, l'idée principale de cette méthode est de considérer la cellule et les noeuds associés de façon indépendante par rapport aux cellules voisines et de calculer la contribution des particules de l'ensemble $\mathcal{P}(c_s)$ aux noeuds. La valeur finale aux noeuds de la grille

4. Soit N_T le nombre total de particule, d la dimension du système et k le nombre de noeuds, le nombre d'opérations est de l'ordre de : $\mathcal{O}(2^d N_T + k)$ dans le cas où $k << N_T$.

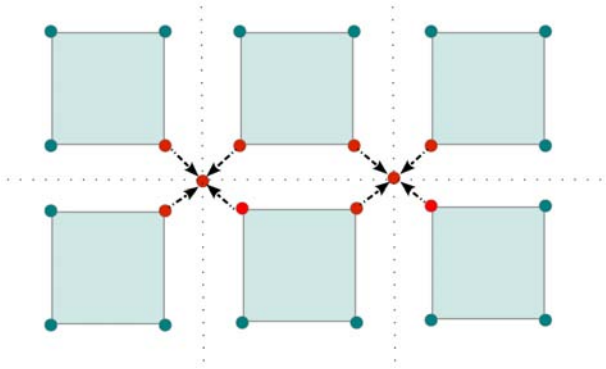


FIGURE 2.5 – Illustration de l'accumulation des valeurs aux noeuds des cellules adjacentes[41].

est obtenue en ajoutant la valeur de la densité des noeuds des cellules adjacentes. Cette étape est illustrée Fig. 2.5.

Nous noterons que cette méthode est également employée dans la version 3D du code PIC MCC. Le passage d'une dimension à l'autre est directe et ne requiert pas d'outils supplémentaires. Cependant, le nombre de particules dans un système à trois dimensions augmente très rapidement et l'algorithme de tri devient de plus en plus contraignant. Ces contraintes interviennent sur le nombre d'éléments à trier à chaque pas de temps et sur la redondance des noeuds pour le calcul des contributions de chaque particule. Autrement dit, le tri des particules affecte d'autant plus les temps de calculs que le nombre de particules devient très élevé dans les simulations en 3D, dans lequel l'espace mémoire utilisé est plus important sachant qu'une particule contribue aux huit noeuds adjacents de la cellule au lieu de quatre noeuds à deux dimensions.

2.5 Résolution de l'équation de Poisson

2.5.1 Méthodes de résolutions

Nous avons vu précédemment, dans la section 2.1, les équations régissant la dynamique des particules (cf. Eq. (2.2)) où $F(r)$ représente la force électromagnétique pour un ensemble de particules chargées de densité de charge ρ. Dans le cas où l'on considère un champ magnétique statique, le calcul du champ électrostatique se réduit à la résolution de l'équation de Poisson.

À partir des équations de Maxwell, un champ électrique peut être généré par des charges.

$$\nabla . E(r) = \frac{\rho}{\epsilon_0} \tag{2.13}$$

Nous pouvons écrire le champ électrique en fonction du potentiel :

$$\vec{E} = -\nabla\phi \tag{2.14}$$

Enfin, la combinaison de ces équations (Eq. (2.13) et 2.14), nous conduit à l'équation de Poisson :

$$\nabla^2\phi = -\frac{\rho}{\epsilon_0} \tag{2.15}$$

Où la forme dérivée à deux dimensions s'écrit :

$$\frac{d^2\phi(x,y)}{dx^2} + \frac{d^2\phi(x,y)}{dy^2} = -\frac{\rho(x,y)}{\epsilon_0} \tag{2.16}$$

Les conditions limites sont définies aux parois (soit $x = 0$, $x = L$, et $y = 0$, $y = L$ où (x,y) sont des coordonnées cartésiennes et L la taille du système) du domaine de simulation. À partir d'un potentiel aux frontières (ici, $\phi = 0$) et une distribution de densité de charges, le potentiel est entièrement défini par l'équation de Poisson.

Pour un maillage régulier de l'espace (bidimensionnel dans cet exemple), le laplacien de l'Eq. (2.16) est approximé par la formule des différences finies suivante :

$$\frac{\phi_{i-1,j} - 2\phi_{i,j} + \phi_{i+1,j}}{\Delta^2} + \frac{\phi_{i,j-1} - 2\phi_{i,j} + \phi_{i,j+1}}{\Delta^2} = -\frac{\rho_{i,j}}{\epsilon_0} \tag{2.17}$$

Où l'on note Δ le pas d'espace de la grille et

$$\phi_{i,j} = \phi(x_i, y_j) \text{ avec } \begin{cases} x_i = x_0 + i\Delta & i = 0, 1, \ldots, N \\ y_j = y_0 + j\Delta & j = 0, 1, \ldots, N \end{cases}$$

L'Éq. (2.17) peut s'écrire plus simplement :

$$\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i,j+1} - 4\phi_{i,j} = -\frac{\rho_{i,j}}{\epsilon_0}\Delta^2$$

ou

$$\phi_{i,j} = \frac{1}{4}\left(\frac{\rho_{i,j}}{\epsilon_0}\Delta^2 + (\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i,j+1})\right) \tag{2.18}$$

Finalement, ce système d'équations linéaires peut s'exprimer sous une forme matricielle dont la structure est la suivante :

$$A\phi = \rho \tag{2.19}$$

Où la matrice A est une matrice tridiagonale, ϕ et ρ sont des vecteurs à N^2 composantes (dans le cas d'un système bidimensionnel composé de N points de grille dans chaque direction).

Plusieurs méthodes numériques existent pour la résolution de ces équations. De façon générale, on peut classer ces méthodes de résolution parmi trois familles : Les méthodes matricielles directes [5], les méthodes de Fourier [6] et les méthodes de relaxation.

De manière plus générale, les **méthodes matricielles** directes, par un nombre fini d'opérations, consistent à déterminer les valeurs de la fonctions ϕ de l'Eq. (2.19) en calculant la matrice inverse de A. Le point faible de cette méthode repose sur la taille de la matrice que l'on doit inverser. Si l'on considère un maillage bidimensionnel de taille $N \times N$, la matrice A est une matrice carrée de taille $N^2 \times N^2$. Dans ce cas, la résolution de ce système linéaire nécessite une structure matricielle simple pour que le calcul de son inverse soit effectué dans des temps de calcul raisonnables.

Les **méthodes de Fourier** sont très performantes pour la résolution de ce type de systèmes linéaires. Alors que les méthodes matricielles requièrent un nombre d'opérations de l'ordre de N^2 (en deux dimensions), les méthodes de Fourier telles que les FFT[7] requièrent quand à elles $NlogN$ opérations. La transformée de Fourier de l'Eq. (2.20) est obtenue à partir des transformées de Fourier discrètes de ϕ et de ρ dans les directions Ox et Oy.

$$\overline{\phi}_{mn}(e^{\frac{2i\pi m}{N}} + e^{\frac{-2i\pi m}{N}} + e^{\frac{2i\pi n}{N}} + e^{\frac{-2i\pi n}{N}} - 4) = \overline{\rho}_{mn}\Delta^2 \quad \text{avec} \quad \begin{cases} m = 0, 1, \ldots, N \\ n = 0, 1, \ldots, N \end{cases} \tag{2.20}$$

Où $\overline{\phi}$ et $\overline{\rho}$ sont les composantes de Fourier de ϕ et ρ.

À partir de la relation 2.20, et après calcul des composantes $\overline{\phi}$, on calcul la transformée de Fourier inverse de $\overline{\phi}$ afin d'obtenir ϕ. La transformée de Fourier obtenue a une solution satisfaisant des conditions aux limites périodiques. Dans le cas de conditions aux frontières plus complexes, les méthodes de Fourier peuvent se révéler un peu plus difficiles d'approche.

Les **méthodes de relaxation** consistent à générer une solution approchée et à améliorer cette solution de façon itérative. Dans cette méthode, la matrice A définie dans l'Eq. (2.19) est décomposée comme $A = E - F$, où E est une matrice inversible et F le reste. En remplaçant A dans l'Eq. (2.19), on peut écrire :

5. Aussi appelées "méthodes directes".
6. Aussi appelées "méthodes spectrales".
7. Fast Fourier Transform

$$E\phi^r = F\phi^{r-1} + \rho \tag{2.21}$$

Où r correspond au terme d'itération. La convergence de la méthode est obtenue quand la différence entre les deux valeurs à l'itération r et $r-1$ de la fonction est inférieure à ϵ, soit : $\| \phi^r - \phi^{r-1} \| < \epsilon$.

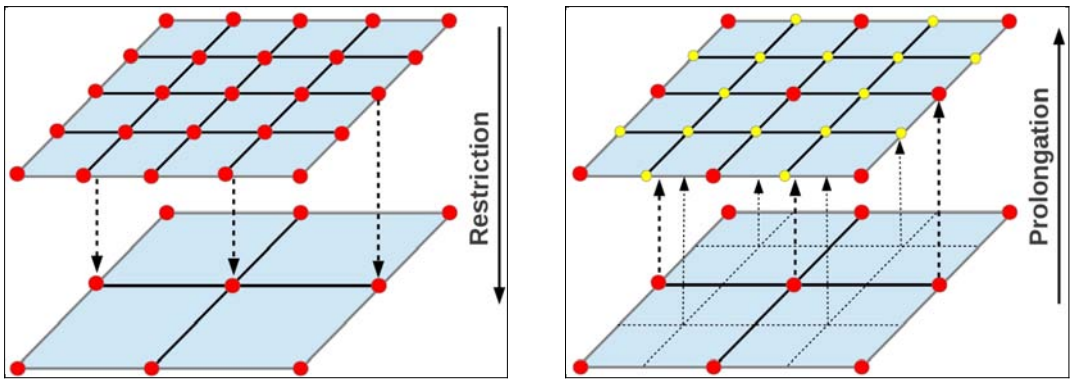
Il a été démontré que des méthodes itératives telles que Gauss-Seidel ou SOR, possèdent des propriétés de *lissage*. Ces propriétés font de celles-ci, des méthodes très efficaces pour l'élimination des hautes fréquences ou des composantes oscillatoires de l'erreur. A l'inverse, elles sont très peu efficaces pour l'élimination des basses fréquences ou des composantes *lisses* de l'erreur.

2.5.2 Méthode multigrille

Les méthodes multigrille sont maintenant étudiées depuis plus d'une trentaine d'années et une introduction sur les principes de bases des algorithmes multigrilles est donnée dans plusieurs ouvrages de références (cf. Réfs. [47, 48, 49, 50, 51, 52, 53]).

La méthode multigrille tient compte des avantages des schémas de relaxation et l'améliore par le biais du calcul de l'erreur sur des maillages de plus en plus grossiers. W.L. Briggs montre (cf. Réf. [47]) qu'après élimination des hautes fréquences de l'erreur, lors du passage d'une grille fine (N points) à une grille plus grossière ($N/4$ points à 2D et $N/8$ en 3D), les modes lisses de l'erreur sur la grille fine deviennent oscillatoires sur la grille grossière. La relaxation sur une grille grossière a donc deux principaux avantages : le nombre de points de grille est divisé par deux donc le nombre d'inconnus également, et la convergence est plus rapide du fait d'un mode de l'erreur plus oscillant. Le taux de convergence augmente avec la diminution du nombre de noeuds.

Deux autres processus importants dont il faut tenir compte dans la méthode multigrille sont les mécanismes de transfert des informations d'une grille à une autre. Ces processus sont des processus d'interpolations et sont appelés *"restriction"* pour le passage d'une grille fine à une grille grossière et *"prolongation"* pour le passage d'une grille grossière à une grille fine. Ces deux passages sont représentés de façon schématique Figs. 2.6.



a) b)

FIGURE 2.6 – Schéma simplifié (a) d'une restriction de grille, (b) de prolongation de grille.

Il existe plusieurs schémas de multigrille, tels que les multigrilles *V-cycle*, *W-cycle* ou encore le schéma appelé *Full MultiGrid (FMG)* et sont détaillés dans Réf. [47]. Chacun de ces schémas diffère par son efficacité de convergence en fonction du problème donné.

Le schéma multigrille implémenté ici est un *V-cycle* avec des conditions limites de Dirichlet, qui spécifie les valeurs des points aux frontières (ici $\phi = 0$). Un schéma simplifié du *V-cycle* est représenté sur la figure suivante (Fig. 2.7).

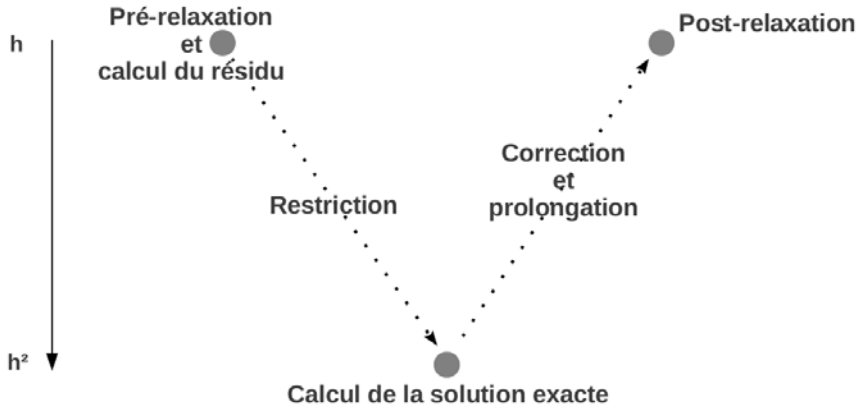


FIGURE 2.7 – Schéma simplifié d'un cycle-V pour la méthode multigrille. La relaxation et le calcul de l'erreur se font sur la grille fine (noté h) et sont suivis d'une restriction sur une grille plus grossière (noté h^2). Après le calcul de la solution sur la grille la plus grossière, on corrige l'erreur et on prolonge vers la grille la plus fine avant de relaxer au niveau h.

La première étape du *V-cycle* est le calcul du résidu de la grille la plus fine à la grille la plus grossière. La restriction est utilisée pour l'interpolation des valeurs calculées sur la grille fine vers la grille moins fine (cf. Fig. 2.6(a)). Sur la grille la plus grossière, la solution exacte est calculée avant d'être interpolée dans une deuxième étape, de la grille moins fine vers la grille plus fine.

L'algorithme utilisé pour la méthode de relaxation est la méthode itérative *Red-Black* Gauss-Seidel (Réf. [54]). L'avantage de cette méthode vient du fait qu'elle est hautement parallélisable. Un noeud sur deux de la grille est mis à jour. Le principe de cette méthode est représenté sur la Fig. 2.8.Les noeuds figurant en rouge sont calculés dans un premier temps à partir des valeurs aux noeuds en noir puis dans un second temps, on calcul les valeurs aux points noirs en accédant à ceux correspondant aux points rouges.

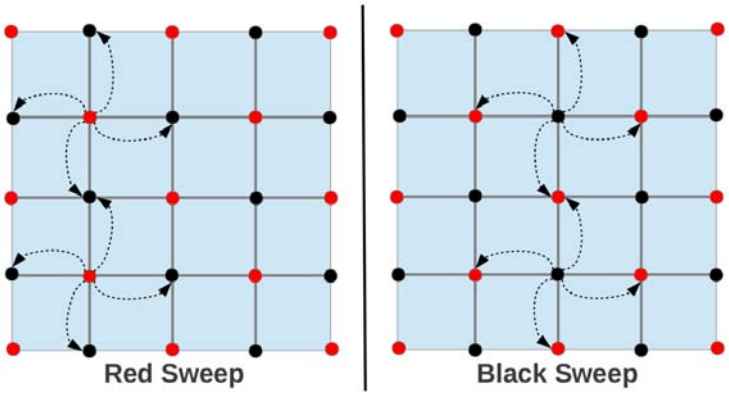


FIGURE 2.8 – Schéma représentatif d'un balayage sur les points rouges (Red sweep) et sur les points noirs (Black sweep).

2.5.3 Implémentation sur GPU

La méthode multigrille est composée de **quatre** parties majeures assemblées entre elles : La relaxation, le calcul du résidu, la prolongation et la restriction. Les routines de **restriction** et de **prolongation** tiennent le rôle important du transfert des données d'une grille à une autre. Pour un problème à deux dimensions, la prolongation d'une grille $2h$ (de dimension $(\frac{N}{2}-1)\times(\frac{N}{2}-1)$) à une grille h (de dimension $N\times N$), les composantes en chaque noeud, notées v^h sont données par :

$$\begin{aligned} v^h_{2i,2j} &= v^{2h}_{i,j} \\ v^h_{2i+1,2j} &= \frac{1}{2}(v^{2h}_{i,j}+v^{2h}_{i+1,j}) \\ v^h_{2i,2j+1} &= \frac{1}{2}(v^{2h}_{i,j}+v^{2h}_{i,j+1}) \\ v^h_{2i+1,2j+1} &= \frac{1}{4}(v^{2h}_{i,j}+v^{2h}_{i+1,j}+v^{2h}_{i,j+1}+v^{2h}_{i+1,j+1}) \quad \text{avec} \quad i,j=0,1,\ldots,\frac{N}{2}-1 \end{aligned} \tag{2.22}$$

Où h correspond à la grille fine et $2h$ à la grille moins fine. D'après l'équation (2.22), on constate que la composante $v^{2h}_{i,j}$ est accédée quatre fois en lecture, $v^{2h}_{i+1,j}$ et $v^{2h}_{i,j+1}$deux fois. De même, la restriction d'une grille h à une grille $2h$ des composantes v^{2h} est donnée

par la relation suivante :

$$\begin{aligned} v_{i,j}^{2h} = \frac{1}{16}[v_{2i-1,2j-1}^{h} &+ v_{2i-1,2j+1}^{h} + v_{2i+1,2j-1}^{h} + v_{2i+1,2j+1}^{h} \\ &+ 2(v_{2i,2j-1}^{h} + v_{2i-1,2j}^{h} + v_{2i+1,2j}^{h} + v_{2i,2j+1}^{h}) \\ &+ 4v_{2i,2j}^{h}] \quad \text{avec} \;\; i,j = 0,1,\ldots,\frac{N}{2}-1 \end{aligned} \tag{2.23}$$

Dans le cas de la restriction, chaque composante $v_{2i,2j}^{h}$ est accédée au moins deux fois en lecture.

Le calcul du **résidu** définit comme : $r = \rho - Av$, où v est une solution approchée de ϕ peut être écrit simplement :

$$r_{i,j} = \rho_{i,j} - \left(\frac{v_{i+1,j} + v_{i,j+1} + v_{i-1,j} + v_{i,j-1} - 4v_{i,j}}{\Delta^2}\right) \tag{2.24}$$

Finalement, le calcul du résidu (Eq. (2.24)) et de la solution approchée (Eq. (2.18)) montre que chaque composante est lue quatre fois lors du balayage sur les noeuds.

Dans le cadre de la résolution de l'équation de Poisson, seule la valeur en chaque noeud du maillage est utilisée. Dans ce cas, la parallélisation est faite au niveau de la grille, i.e. le nombre de threads à l'exécution correspond au nombre de noeuds de la grille. Cela signifie que le nombre de threads lancé à l'exécution est nettement inférieur à celui du transport (Section 2.3) qui est proportionnel au nombre de particules. Cependant et contrairement au cas du transport, où chaque particule évolue indépendamment des autres, la valeur calculée en un noeud de la grille dépend explicitement des valeurs aux noeuds voisins (4 noeuds en 2D, 6 noeuds en 3D). Ceci est visible à partir des Eqs. (2.18), (2.22), (2.23), (2.24) à travers les indices des noeuds $i \pm 1$ et $j \pm 1$.

Dans ces quatre parties (relaxation, résidu, restriction et prolongation), Il est possible d'optimiser les temps d'accès à la mémoire et d'éviter les écritures concurrentes entre les threads. Une méthode simple et efficace consiste à utiliser la mémoire texture comme une mémoire cache. C'est à dire que les valeurs aux noeuds sont copiées avant l'appel du noyau de calcul dans la mémoire texture. Une texture de référence (*texture binding*) est définie au préalable et est utilisée par le noyau pour accéder et lire les données situées dans la mémoire texture (cf. Réf. [1] (*texture fetching*)). Ainsi, la lecture des données dans cette mémoire réduit significativement la latence due aux accès répétés à la mémoire. Ceci permet donc un gain de temps de calcul important.

Enfin, la méthode itérative de relaxation Red-Black Gauss-Seidel est réalisée en deux étapes, qui sont illustrées Fig. 2.8. La première étape (appelée également *Red Sweep*)

consiste à lire les valeurs sur les noeuds impairs (indiquées par les cercles pleins noirs) pour calculer la nouvelle valeur sur les noeuds pairs (indiquées par les cercles pleins rouges). De la même façon, lors de la deuxième étape (appelée *Black Sweep*), on lit la valeur sur les noeuds pairs pour calculer la nouvelle valeur sur les noeuds impairs. Cette méthode permet d'éviter les collisions en mémoire lors de la mise à jours des valeurs sur les noeuds et permet également une utilisation de la mémoire texture et donc un accès aux données plus rapide qu'à partir de la mémoire globale.

Plusieurs tests de performances de calculs ont été réalisés entre le multigrille sur CPU et celui sur GPU. A partir des temps de calculs, on montre un gain de temps pouvant être très important pour un nombre de noeuds élevé (cf. Section 2.9). De plus, il a été vérifié un large bénéfice dû à l'utilisation de la mémoire texture.

Nous noterons que l'utilisation de la mémoire texture doit être manipulée avec précaution lors de l'implémentation du code PIC en 3D. La mémoire requise étant plus importante, la taille de la mémoire texture peut être insuffisante pour des simulations supérieures à 256^3 cellules.

2.6 Interpolation des champs

Les valeurs des champs (**E** et **B**) sont obtenues spatialement sur une grille discrète de points. Dans notre cas, en supposant que le courant généré dans le plasma est suffisamment faible, le champ magnétique auto induit peut être négligé. Alors, à partir de l'équation de Maxwell-Faraday (Eq. (2.25)), le champ vectoriel **E** à rotationnel nul dérive d'un gradient de potentiel scalaire (Eq. (2.26)).

$$\begin{aligned} \nabla \times \mathbf{E} &= -\frac{\partial \mathbf{B}}{\partial t} \\ \text{devient :} \quad \nabla \times \mathbf{E} &= 0 \end{aligned} \tag{2.25}$$

$$\mathbf{E} = -\nabla\phi \quad \text{ou} \quad E_{(x,y,z)} = -\left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z}\right) \tag{2.26}$$

Dans un système à deux dimensions, la résolution des équations différentielles définies Eq. (2.26) est obtenue par la formule des différences finies et permet de connaître la valeur du champ électrostatique en chaque noeud du maillage (Eqs. (2.27)) :

$$\begin{aligned} E_{x_{i,j}} &= \frac{\phi_{i-1,j} - \phi_{i+1,j}}{2\Delta x} \\ E_{y_{i,j}} &= \frac{\phi_{i,j-1} - \phi_{i,j+1}}{2\Delta y} \end{aligned} \tag{2.27}$$

Où les indices (i, j) correspondent aux points discrets de la grille illustrés sur la figure 2.9. La résolution des équations du mouvement définies dans l'Eq. (2.6) est obtenue au pas de temps δt en connaissant la force $F_{(x,y)}$ s'exerçant sur la particule de coordonnées (x, y). Autrement dit, les nouvelles vitesses des particules peuvent être calculées à partir des valeurs du champ électrostatique interpolées aux positions des particules.

Le schéma d'interpolation de la force est traité de façon similaire à une interpolation linéaire (cf. Section 2.4). Le champ vectoriel dans la cellule est donné par les valeurs aux noeuds $n_{i,j}$ de cette cellule. La force électrostatique s'exerçant sur la particule p de coordonnées (x, y) est donnée par :

$$F_{(x,y)} = \sum_{n_{i,j}} E_{i,j} K(n_{i,j}, p_{x,y}) \tag{2.28}$$

Où K est la fonction d'interpolation linéaire définie à la section 2.4.1 et $E_{i,j}$ correspondant à la valeur du champ électrique calculée sur le noeud i, j. Soit à deux dimensions, la force interpolée aux coordonnées (x, y) est donnée par :

$$\begin{aligned} F_{(x,y)} = \frac{E_{i,j}}{V_{X,Y}} \underbrace{(X_{i+1} - x)(Y_{j+1} - y)}_{A_{11}} &+ \frac{E_{i+1,j}}{V_{X,Y}} \underbrace{(x - X_i)(Y_{j+1} - y)}_{A_{10}} + \\ \frac{E_{i,j+1}}{V_{X,Y}} \underbrace{(X_{i+1} - x)(y - Y_j)}_{A_{01}} &+ \frac{E_{i+1,j+1}}{V_{X,Y}} \underbrace{(x - X_i)(y - Y_j)}_{A_{00}} \end{aligned} \tag{2.29}$$

Où $V_{x,y} (= \Delta_X \Delta_Y)$ est le volume de la cellule, (X_i, Y_j) sont les coordonnées de la cellule aux noeuds (i, j) et (x, y) les coordonnées de la particule (cf. Fig. 2.9). Les aires $A_{00}, A_{01}, A_{10}, A_{11}$ sont définies dans la section 2.4.1, Fig. 2.4(b).

2.6.1 Implémentation sur GPU

Les cartes graphiques actuelles présentent l'avantage, en plus d'être massivement parallèles, de posséder une structure matérielle intégrée permettant la réalisation d'interpolation linéaire de manière très efficace. La mémoire texture permet de calculer efficacement ces

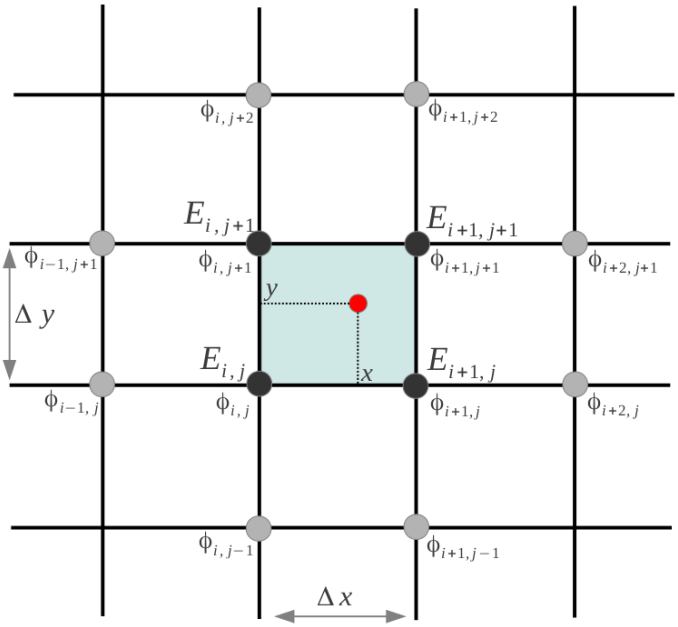


FIGURE 2.9 – Discrétisation de la grille pour le calcul du champ **E** et interpolation à la position de la particule.

interpolations linéaires. Le principe d'utilisation de la mémoire texture est semblable à celui expliqué dans la section précédente (cf. Section 2.5.3) mais requiert cette fois-ci, l'utilisation de tableaux appelés *cudaArray*. L'utilisation de ces tableaux permet une gestion des éléments dans la mémoire, optimisée pour l'accès à la mémoire texture.

Ainsi, connaissant le champ électrique en chaque noeud de la grille (cf. Eqs. 2.27), on copie ces valeurs dans un tableau *cudaArray*. Ce tableau de données est ensuite placé (lié) en mémoire texture (*texture binding*) à partir de laquelle on va pouvoir accéder *via* une texture de référence, et lire les données.

Une fonction CUDA permet de calculer directement la valeur interpolée aux coordonnées de la particule (fonction *tex2D* pour l'interpolation en 2D, *tex3D* en 3D. cf. [1]). Finalement, connaissant les coordonnées (x, y) de la particule, le nombre de noeuds (N) dans chaque direction et la taille du domaine de simulation (L), le champ électrique interpolé à la position de la particule est donnée par :

$$E_{(x,y)} = \text{tex2D}\left(texRef, \left[\frac{N_x - 1}{L_x}x + 0.5\right], \left[\frac{N_y - 1}{L_y}y + 0.5\right]\right) \tag{2.30}$$

Où $texRef$ est la texture de référence et où les valeurs de x, y sont comprises dans l'intervalle $[0, L_{x,y}]$ et pour un nombre de noeuds dans chaque direction, compris dans l'intervalle $[0, N_{x,y} - 1]$ (cf. [1](*Table Lookup*)). De plus, cet outil d'interpolation linéaire

peut être également exécuté dans le cas d'un système à trois dimensions avec la fonction *text3D*.

L'utilisation de cet outil permet un gain non négligeable de temps de calcul. P. Abreu et al. [34] observent un facteur $\times 3$ en performance de temps de calcul comparé à une interpolation linéaire qui n'utilise pas la mémoire texture et ses fonctions d'interpolations.

2.7 Monte Carlo collisions (MCC)

Le modèle Monte Carlo est une modèle probabiliste cherchant à reproduire les effets de collisions. Connaissant l'énergie cinétique $E_{cin}(= \frac{1}{2}mv^2)$ d'une particule chargée donnée et la vitesse relative v_r de celle-ci par rapport à la particule cible, il est possible de déduire une fréquence de collision $\nu_{coll} = n_{cible}\sigma_T v_r$ ainsi qu'une probabilité de collision dans un temps δt, donnée par $P(t) = \nu_{coll}\delta t$. Ici, n_{cible} est la densité de gaz neutre, v est la vitesse de la particule incidente (ici, l'électron), σ_T est la section efficace de collision totale. σ_T est fonction de l'énergie de l'électron ($Ecin$) et est la somme des sections efficaces de chaque type de collision, telles que les collisions élastiques, l'excitation, l'ionisation ou encore la dissociation. En d'autre termes, on note

$$\sigma_T = \sum_{k=1}^{K} \sigma_k \tag{2.31}$$

Où σ_k est la section efficace correspondant au k^{ieme} type de collision et K est le nombre total de processus collisionnels. Dans le modèle, seuls les processus de collisions élastiques, les processus d'ionisation et d'excitation sont pris en compte.

Une méthode de collision Monte Carlo, basée sur la technique des collisions nulles (Réfs. [55, 56, 57]), a été développée et implémentée sur GPU pour effectuer des collisions entre des particules chargées et les neutres. Seules les collisions électrons-neutres sont prises en compte dans le modèle. Cette approche de la méthode des collisions nulles est une méthode numérique "forte" permettant un gain de temps de calcul très important.

La méthode "classique" Monte Carlo requiert à chaque pas en temps un test pour chaque particule si une collision à lieu. Ainsi, il faut calculer l'énergie cinétique pour chacune d'entre elle. Ce procédé est évidemment très coûteux en terme de ressources et de temps de calcul. On peut éviter de tester toutes les particules en ajoutant une fréquence de collision "nulle" constante que l'on peut définir par :

$$\sigma_{max} = max(n_{cible})max(\sigma_T v_r) \tag{2.32}$$

En d'autre termes, un autre processus collisionnel est introduit, dont la fréquence de collision ajoutée à ν_{coll} est égale à une constante (cf. Fig. 2.10). Pour une fréquence totale de collision ν_T (incluant la fréquence de collision nulle) et en supposant un nombre total N d'électrons dans la simulation, le nombre total d'électrons subissant une collision dans le pas de temps δt est donné par la relation $(N\nu_t\delta t)$. Seules les $(N\nu_t\delta t)$ particules chargées faisant des collisions sont choisies de façon aléatoire parmi les N particules et sont traitées.

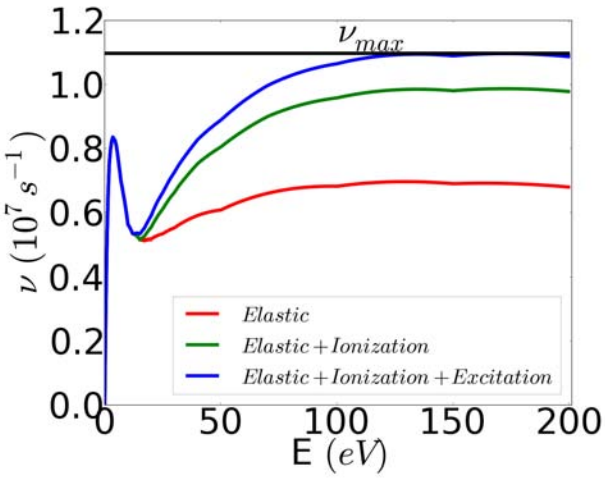


FIGURE 2.10 – Ajout d'un processus collisionnel de collisions nulles dans le cas de section efficaces de $H2$, avec ν_{max} correspondant à la somme des maximum des différentes sections efficaces d'ionisation, d'excitation et de collisions élastiques.

La source de plasma est un système très complexe à modéliser et la prise en compte de tous les phénomènes entrant en jeu pour la description de celui-ci tels que la cinétique du gaz, la chimie, la dynamique des différentes espèces et leur transport, les interactions plasma-surface, le transport du plasma à travers le champ magnétique, etc, est une tâche ardue voir impossible à inclure dans un seul modèle. Une meilleure approche est de considérer séparément les différents aspects du problème. Ainsi, le modèle réalisé tente de décrire le transport du plasma à travers un filtre magnétique et la chimie complexe du plasma ou encore la production et l'extraction des ions négatifs ne sont pas décrits en détails et un modèle de collisions simplifié est utilisé. Seules les collisions électrons-neutres sont prises en compte dans le modèle et incluent les collisions élastiques, les processus d'ionisation et d'excitation. La Fig. 2.11 montre les sections efficaces de collisions utilisées dans le modèle et sont représentées par une section efficace de collision élastique, une section efficace de collision inélastique (e.g. excitation) et une section efficace d'ionisation.

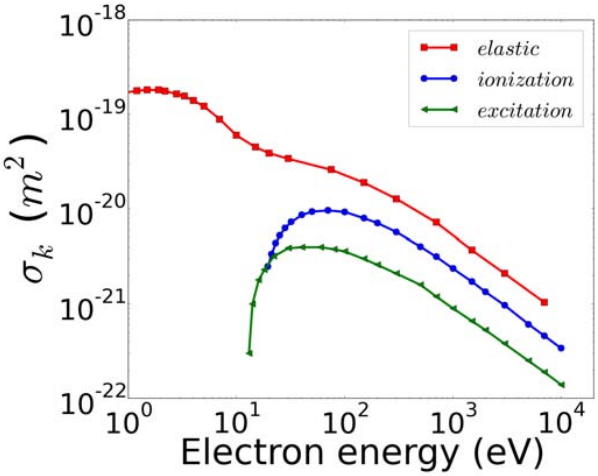


FIGURE 2.11 – Sections efficaces de collisions des électrons avec le dihydrogène (H_2). La courbe (et carrés) rouge représente la section efficace de collision élastique, en bleu (et cercles), la section efficace d'ionisation avec une énergie seuil de 15.4 eV et la courbe (et triangles) verte représente la section efficace de collisions inélastique avec une énergie seuil de 12.4 eV[58].

Enfin, les collisions Coulombiennes (collisions électrons-ions) et les collisions ions-neutres ne sont pas modélisées dans ce cas particulier. Cependant, il peut être montré que ces collisions n'influencent pas ou de façon négligeable le transport électronique à travers la barrière magnétique. Ceci sera vu un peu plus en détail dans la section 3.3 du chapitre suivant.

2.7.1 Collisions élastiques

Dans le cas de collisions élastiques entre particules chargées et neutres, la particule incidente est diffusée de façon isotrope et l'énergie, suite à la collision, est répartie entre les particules. Le processus de collision peut être représenté comme

$$e_1(v) + A(V) \longrightarrow e_1(v^*) + A(V^*) \tag{2.33}$$

Où e_1 et A dénotent respectivement les espèces chargées et neutres, v et V sont les vitesses respectives de l'électron incident et du neutre et v^* et V^* sont les vitesses après la collision. Les neutres sont supposés avoir une distribution en vitesse Maxwellienne et sont maintenus uniformément dans l'espace comme un fond gazeux à basse température. Lorsqu'une collision a lieu, la répartition des vitesses entre les particules après la collision peut être déterminée à partir de l'équation de conservation de la quantité de mouvement Eq. (2.34a) et est donnée par l' Eq. (2.34b).

$$m\vec{v_e} + M\vec{V_A} = m\vec{v_e}^* + M\vec{V_A}^* \tag{2.34a}$$

$$\vec{v_e^*} - \vec{V_A^*} = (||\vec{v_e} - \vec{V_A}||) \cdot \vec{R} \tag{2.34b}$$

Où m et M sont respectivement les masses de l'électron et de la molécule, v^* et V^* représentent les vitesses des particules concernées après la collision et $\vec{R}$ un vecteur de directions aléatoires "unitaire". Soit la vitesse de l'électron après collision (cf. détails en Annexe B.1) :

$$\boxed{\vec{v_e^*} = \frac{1}{m+M}\left(m\vec{v_e} + M\left(\vec{V_A} + ||\vec{v_e} - \vec{V_A}||\right) \cdot \vec{R}\right)} \tag{2.35}$$

2.7.2 Ionisation et Excitation

Lorsqu'une collision ionisante a lieu entre un électron et un neutre, l'électron incident possède suffisamment d'énergie pour extraire un autre électron de la molécule. Cette collision peut être représentée par :

$$e_1(v) + A(V) \longrightarrow e_1(v') + A^+(V') + e_2(v'') \tag{2.36}$$

Où e_1 et A sont respectivement l'électron et la molécule neutre présentés précédemment (cf. 2.7.2). A^+ et e_2 sont respectivement l'ion créé avec une nouvelle vitesse V' et l'électron extrait avec sa vitesse correspondante v''. De la même façon, on écrit l'équation de conservation de la quantité de mouvement

$$m\vec{v_{e_1}} + M\vec{V_A} = m\vec{v'_{e_1}} + (M-m)\vec{V'_{A^+}} + m\vec{v''_{e_2}} \tag{2.37}$$

En supposant que les particules sont diffusées de manière isotrope, les nouvelles vitesses des particules peuvent être déterminées à partir de l'Eq. (2.37) et sont données pour l'électron incident et extrait par :

$$\boxed{\tilde{\vec{v'_{e_1}}} = \vec{R}\tilde{v'_{e_1}} \qquad \tilde{\vec{v''_{e_2}}} = \vec{\dot{R}}\tilde{v''_{e_2}}} \tag{2.38}$$

Où $\vec{R}$ et $\vec{\dot{R}}$ sont deux vecteurs de directions aléatoires différents et ($\tilde{}$) dénote la vitesse à laquelle la vitesse du centre de masse v_{cm} ($= \frac{m\vec{v_{e_1}} + M\vec{V_A}}{m+M}$) est soustraite (cf. Annexe B.1).

Le rapport des masses entre l'ion et l'électron étant très grand, on peut supposer que la quantité de mouvement de l'électron incident est très inférieure à la quantité de mouvement de l'atome neutre. En d'autre termes, on fait l'approximation qu'après la collision, le neutre devient un ion et donc que la vitesse de l'ion créée est définie par $V'_{A^+} = V_A$.

Les processus d'excitation sont traités dans le modèle de la même façon que les processus d'ionisation. On considère le schéma suivant :

$$e_1(v) + A(V) \longrightarrow e_1(v') + A^*(V') \tag{2.39}$$

Où cependant aucun processus de création n'a lieu. Lors de ces processus collisionnels, l'électron incident perd une énergie équivalente à l'énergie seuil d'excitation (de 12.4eV dans le cas du gaz $H2$) et est diffusé de façon isotrope. Il est à noter que dans ce modèle, on ne suit pas l'atome excité.

2.7.3 Implémentation sur GPU

À chaque pas en temps, nous devons connaître la probabilité de collision des particules sélectionnées aléatoirement. Cette probabilité est calculée à partir de la section efficace de collision dépendant de l'énergie de la particule incidente. La méthode utilisée pour le calcul de la section efficace nécessite un pré-traitement des données. Les sections efficaces sont définies dans un fichier externe, comprenant les différents processus de collisions (élastiques, ionisations, excitations, ..., etc) pour différents types de gaz.

Il est possible d'utiliser les fonctions d'interpolations sur le GPU afin de calculer à chaque pas de temps la section efficace correspondante à l'énergie de la particule (cf. Section 2.6.1). Cependant, ce type d'interpolation nécessite une discrétisation des valeurs de sections efficaces à des intervalles réguliers d'énergie. Avant de lancer le code PIC, les valeurs des sections efficaces utilisées sont interpolées sur le CPU puis sont copiées sur la mémoire du GPU. Ces valeurs sont allouées en mémoire dans un emplacement optimisé pour le texture fetching, appelé *cudaArray* (cf. Section 2.6.1), pour toute la durée de la simulation.

Un autre point important pour la méthode de collision Monte Carlo concerne le tirage aléatoire. Une librairie CUDA, appelée *CURAND*[59] est maintenant disponible pour la génération de nombres aléatoires. Elle permet une génération efficace des séquences de nombres pseudo-aléatoires (*pseudorandom*) et quasi-aléatoires (*quasirandom*). Il existe deux possibilités pour générer les nombres aléatoires sur le GPU. La première possibilité est de générer une séquence de nombres aléatoires qui sont par la suite stockés dans la

mémoire globale du GPU. Ces nombres pouvant être utilisés lors de l'appel d'un noyau. La deuxième solution est l'utilisation de fonctions définies sur le GPU, autorisant la génération immédiate de nombres aléatoires pouvant être utilisés directement par le noyau sans avoir besoin au préalable de les écrire et de les lire à partir de la mémoire globale du GPU. Il est possible d'utiliser la première méthode de génération de nombre aléatoire, en tirant une séquence de nombres aléatoires équivalente au nombre maximum de particules pouvant faire une collision ($= N\nu_t\delta t$). Néanmoins, afin d'éviter de tirer des nombres aléatoires qui ne seront pas utilisés, il est préférable d'avoir recourt à la deuxième méthode, qui semble être la plus adaptée dans ce cas.

Le traitement des collisions élastiques et des processus d'excitation ne pose pas de problèmes majeurs pour l'implémentation sur GPU. Les indices des particules sont tirés aléatoirement. Les particules étant indépendantes les unes des autres, le calcul de la vitesse *post*-collision est direct. Un inconvénient qui peut être une source d'erreur de la méthode de collisions nulles est que l'on peut sélectionner deux fois la même particule. Cependant, la probabilité de telles occurrences est très faible devant le nombre de tirages aléatoires effectués et le nombre important de particules dans le système, et peut donc être négligée.

En revanche, le traitement des processus d'ionisation est un peu plus complexe que les processus de collisions élastiques. Dû à la création de particules et au caractère aléatoire des tirages, il nous impose certaines contraintes au niveau de la parallélisation, notamment concernant l'accès à la mémoire et l'arrangement dans le tableau des particules de celles nouvellement créées. Soit N_s le nombre de particules sélectionnées pouvant faire une collision, il n'est en aucun cas possible de connaître à l'avance le nombre d'ionisations qui auront lieu dans le temps δt sur les N_s particules sélectionnées. Chaque création de paires de particules donne lieu à l'ajout de nouvelles positions et de nouvelles vitesses dans les tableaux correspondant. En parallèle, plusieurs processus d'ionisation peuvent avoir lieu au même moment. Considérant ces contraintes et afin d'éviter des collisions en mémoire lors de l'écriture des nouvelles paires de particules en mémoire par des threads différents, la fonction atomique *atomicAdd* (Réf. [1]) est utilisée permettant l'incrément d'un compteur. Chaque thread récupère le résultat de son opération d'incrémentation et l'utilise pour stocker, à partir du dernier élément des tableaux de positions et de vitesses, la nouvelle paire de particules chargées (cf. List. 2.4). De plus, le résultat final du compteur permet de connaître le nombre de particules créées pendant δt. Bien que ce type d'opération puisse s'avérer coûteux en terme d'exécution due à la sérialisation des opérations d'incrémentation, il reste néanmoins négligeable car le nombre d'ionisations ayant lieu à chaque δt est relativement faible. Ceci a comme conséquence que le temps d'exécution de la routine MCC est nettement négligeable comparé aux autres parties du code PIC (cf. section 2.9), et est de l'ordre de quelques pourcents ($\sim 2\%$) du temps de calcul global.

Listing 2.4– Kernel : Création de particules et définition des indices

```
/* thread index definition */
int i = get_Thread_Index();

for (Pour chaque particules P_i ∈ N_T particules)
{
  /* Draw particle index */
  int indx      = get_rand() * N_T;
  float Ek      = get_Particle_Kinetic_Energy(P[indx]);
  float Col_sec = get_interpolated_Cross_Section(Ek);
  float Freq    = get_Collision_Frequency(Col_sec);
  if (get_rand() < Freq){ // if (true), a collision occurs
    int inc     = atomicAdd(counter, 1);
    /* incident particle */
    vel[indx]   = get_new_velocity();
    /* ejected particle */
    vel[N_T+inc] = get_new_velocity();
  }
}
/* Mise a jour du nombre total de particules N_T */
N_T = N_T + counter;
```

2.8 Chauffage des électrons

Dans le but d'obtenir les caractéristiques d'un plasma auto-consistant, les électrons sont sélectionnés aléatoirement pour être "chauffés" à l'intérieur d'une région du domaine de simulation. Dans cette zone de "chauffage", on suppose l'absorption d'une puissance externe injectée. L'énergie totale des électrons dans la zone de chauffage, i.e. l'énergie cinétique totale des électrons ajoutée à l'énergie externe absorbée dans un temps δt, est convertie en "température effective". Les électrons chauffés ont alors leurs vitesses remplacées par une nouvelle vitesse calculée à partir d'une distribution Maxwellienne à la température définie précédemment (Les détails du chauffage des électrons sont donnés dans la Section 3.3).

La méthode de préfixage des particules est utilisée pour le chauffage des électrons (cf. Section 2.3.1). Les électrons présents dans la zone de chauffage sont sélectionnés (une valeur de 1 leur est assignée si ils sont dans la zone de chauffage, 0 sinon) et pour chaque particule affectée d'une valeur non nulle, on écrit en mémoire, dans un tableau temporaire, la norme de la vitesse. Deux opérations de réductions (Réf. [60]) sont utilisées pour le calcul du

nombre total d'électrons dans la zone de chauffage et le calcul de la température moyenne d'agitation thermique.

La partie concernant le chauffage des électrons est relativement coûteuse car elle demande à chaque itération en temps, de faire un test sur la position de toutes les particules, ainsi que deux opérations de réductions sur un nombre important de particules. Ces opérations ne sont pas avantageuses car elle ne permettent pas l'utilisation intensive et efficace des capacités du GPU. Ceci sera constaté dans la Section 2.9, où le temps global de calcul dans cette partie est de l'ordre de 20%.

2.9 Résultats et temps de calcul

L'algorithme PIC MCC décrit précédemment a pu être appliqué à un cas test qui décrit le transport des particules d'un plasma à basse température à travers un filtre magnétique. Les conditions de ces simulations sont données plus en détails par la suite dans la Section 3.3.

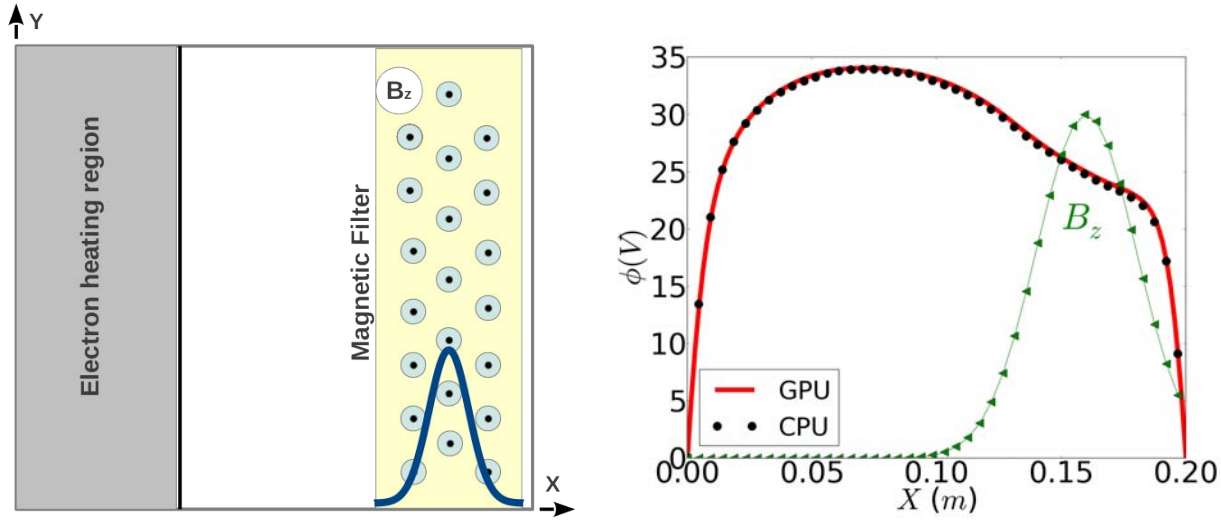
Les résultats discutés dans cette partie montrent principalement des comparaisons de performances entre le CPU et le GPU pour un cas de simulation type PIC MCC de plasma à basse température à **deux dimensions**. De plus, lors de l'implémentation du code PIC MCC, la carte graphique utilisée ne possédait seulement qu'une seule unité arithmétique en double précision pour chaque SM (Stream Multiprocessors). Ceci signifie que l'utilisation de la double précision pouvait réduire les performances jusqu'à $\times 9$ par rapport aux calculs en simple précision. C'est pourquoi, afin d'utiliser au maximum les capacités de la carte graphique, il a été décidé d'utiliser exclusivement le calcul en simple précision.

La géométrie de la source de plasma est 2D rectangulaire. Les électrons du plasma sont chauffés dans la zone de chauffage située sur la gauche de la source. Cette zone correspond environ à $\frac{1}{3}$ de la surface du domaine de simulation. Le filtre magnétique est situé sur la partie droite de la source (cf. Fig. 2.12 (a)).

Les comparaisons de performances entre la version PIC MCC sur GPU et la version sur CPU, ont été réalisées pour différents scénarii. La géométrie, les dimensions physiques et les conditions initiales restent inchangées dans chacun de ces cas. De plus, le pas en temps δt, la fréquence de chauffage, la pression, la densité initiale des différentes espèces et le champ magnétique externe restent constants. Seul le nombre de noeuds du maillage et le nombre de macro particules par cellule varient et sont choisis afin de satisfaire au mieux les critères de validité numérique associés aux codes PIC. Les résultats des simulations dans chaque cas sont obtenus après des temps de calculs équivalent à 30 microsecondes d'évolution de la décharge, pour laquelle les conditions d'état stationnaire du plasma sont atteintes.

Les différents tests ont été réalisés sur le processeur 2.2 GHz 64-bit quad core AMD

opteron avec le compilateur Intel Fortran 10.1 pour la version sur CPU et sur une carte graphique NVIDIA Tesla C2050 avec le compilateur NVCC 4.1 pour la version sur GPU. L'option standard d'optimisation $O2$ a été utilisée pour la compilation des deux codes. Comme il a été décrit précédemment, le code implémenté sur GPU utilise la simple précision contrairement au code sur CPU qui est entièrement implémenté en double précision. Un très bon accord est obtenu entre les résultats issus de la simulation pour le code PIC MCC sur CPU en double précision et la version en simple précision sur GPU. Un de ces résultats est représenté sur la Fig. 2.12 (b) est montre le potentiel plasma intégré le long de l'axe y. La courbe pleine en rouge correspond aux résultats issus de la simulation sur GPU et les cercles pleins en noir à ceux du CPU.



(a) (b)

FIGURE 2.12 – (a) Domaine de simulation (2D Cartésien). la densité de gaz (H_2) dans la chambre est de 5×10^{19} m^{-3}. les électrons sont chauffés de façon uniforme dans la région grisée (La puissance absorbée est de 10 W/m). Le champ magnétique maximum est de 3 mT. (b) Comparaison du potentiel plasma pour le cas test entre la version du code PIC MCC sur CPU représenté par les cercles noirs et la version sur GPU en rouge. Le profil du champ magnétique est représenté en vert (triangles).

Ce potentiel est obtenu dans les conditions de la Fig. 2.12 (a), où une moyenne du potentiel plasma a été effectué au cours des dix dernières microsecondes de la simulation de la décharge, i.e. à partir du moment où l'on considère avoir atteint l'état stationnaire du plasma. Cette moyenne est réalisée à partir d'un échantillonnage dans le temps (ici, toutes les 0.02 microsecondes) des grandeurs caractéristiques du plasma (potentiel, température, densité, flux) et permet ainsi d'avoir des résultats moins bruités.

Typiquement, le code séquentiel PIC sur CPU prend 80% du temps de calcul en simple précision par rapport au temps de calcul en double précision. Nous noterons que ce facteur

TABLE 2.1 – **Taille de la grille constante**

Grid Points	Initial Total Particles	Initial PPC	CPU time (Hrs)	GPU time (Hrs)	Ratio ($\frac{\text{CPU}}{\text{GPU}}$)
128^2	3.27×10^5	20	5.50	.48	11.45
	6.55×10^5	40	10.25	.80	12.81
	1.31×10^6	80	19.8	1.41	14.04
	2.62×10^6	160	38.9	2.67	14.57
	5.24×10^6	320	77.7	5.11	15.20

n'est pas pris en compte dans les performances montrées dans les tables suivantes.

Dans les deux premiers cas, le nombre de particules par cellule (PPC) varie : Premièrement, en variant le nombre total initial de particules et en gardant un maillage constant (Table 2.1) et deuxièmement, en faisant varier le nombre de noeuds du maillage et en gardant le nombre total initial de particules constant (Table 2.2). Concernant le troisième cas (Table 2.3), le nombre de particules par cellule reste constant et la résolution du système augmente linéairement avec l'augmentation du nombre de noeuds et donc celui du nombre total initial de particules. Le nombre de cellules dans chaque direction et le nombre de particules varient d'un facteur 2 entre les différents cas.

La variation du temps de calcul de l'algorithme PIC dans les deux codes (CPU et GPU) reste approximativement linéaire dans les deux cas du nombre de particules et du nombre de noeuds constants. Cependant, en comparant les temps de calcul de la Table 2.1 et de la Table 2.2, on constate que le temps de calcul est déterminé par le nombre total de particules. Sur la Table 2.1, si l'on augmente le nombre de particules d'un facteur 2, on peut voir que pour les deux versions du code PIC (CPU et GPU), le temps total de la simulation est approximativement multiplié par 2 également et que cette évolution reste quasi linéaire. Ce qui explique la faible évolution du rapport de temps total entre le GPU et le CPU. Sur la Table 2.2, on constate une évolution plus importante du ratio (CPU/GPU) pour un nombre de noeuds de plus en plus important. Si l'on observe les temps de calcul du CPU, on peut voir que contrairement au GPU, l'augmentation de la résolution de la grille affecte de plus en plus les performances de calculs. Le rapport de temps total entre la grille 64×64 et 512×512 et de l'ordre de 1.1 pour le GPU contre 1.8 pour le CPU et explique le gain de temps important avec le GPU pour les simulations utilisant des maillages plus fins. Enfin, la Table 2.3 montre clairement que pour la simulation de systèmes de hautes résolutions, ici avec un nombre de particules important et une définition de grille élevée, le code parallélisé sur GPU obtient de très bonnes performances.

TABLE 2.2 – **Nombre total de particules constant**

Grid Points	Initial Total Particles	Initial PPC	CPU time (Hrs)	GPU time (Hrs)	Ratio ($\frac{\text{CPU}}{\text{GPU}}$)
64^2		1280	73.35	5.06	14.5
128^2		320	77.7	5.14	15.11
256^2	5.24×10^6	80	86.65	5.35	16.19
512^2		20	131	5.83	22.46

TABLE 2.3 – **Nombre de PPC constant**

Grid Points	Initial Total Particles	Initial PPC	CPU time (Hrs)	GPU time (Hrs)	Ratio ($\frac{\text{CPU}}{\text{GPU}}$)
64^2	8.19×10^4		1.85	.20	9.25
128^2	3.27×10^5		5.50	.48	11.45
256^2	1.31×10^6	20	19.75	1.48	13.34
512^2	5.24×10^6		131	5.83	22.46

Afin de comprendre plus en détails les différentes évolutions du temps de calcul dans le code PIC parallélisé sur GPU, il est intéressant d'observer le temps global passé dans chaque module du code. La Fig. 2.13 montre ces résultats pour un cas particulier d'une grille de 128×128 noeuds avec 40 particules par cellules. La partie du code concernant l'interpolation des charges sur les noeuds prend environ 40% du temps total d'exécution dans tous les cas de simulations. Ceci confirme bien que l'interpolation des charges sur la grille est la tâche la plus coûteuse du code PIC sur GPU. L'implémentation de cette partie n'est pas complètement optimisée et on peut estimer à environ plus de 20%, le gain en temps de calcul qui pourrait être obtenu par l'amélioration de l'algorithme de tri des particules comme il a été suggéré par G. Stantchev et al. [41].

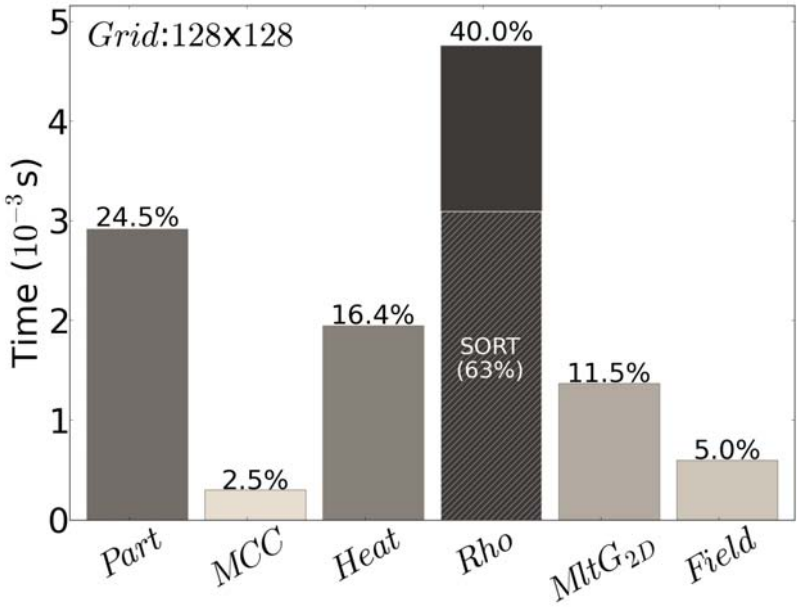


FIGURE 2.13 – Temps passé dans les différents modules du code PIC MCC sur GPU dans le cas de simulations avec 40 particules par cellules et une grille de 128×128 noeuds. *Part* est le module de transport des particules, *MCC* est le module de collisions, *Heat* est le module d'injection de puissance, *Rho* est le module d'interpolation des charges sur la grille, $MltG_{2D}$ est le solveur de l'équation de Poisson et *Field* correspond au module d'interpolation du champ électrique. La majeure partie du temps (63%) dans le module de *Rho* est passé dans le tri des particules.

Le module du calcul du transport des particules (noté *Part* sur la Fig. 2.13) est la deuxième plus importante partie en terme de temps de calcul et peut être expliqué par le très grand nombre de particules évoluant dans les simulations. Bien qu'un nombre relativement important de tirages aléatoires est effectué dans le module de collisions Monte Carlo (*MCC*), le temps passé reste négligeable contrairement au module de chauffage des électrons (*Heat*) qui correspond à environ 16% du temps total. Ceci met en évidence l'effet important des opérations atomiques sur le temps de calcul. Enfin, le solveur de l'équation de Poisson ($MltG_{2D}$) montre des résultats encourageant dans les différentes simulations (ici, le solveur prend seulement 10%) du temps total d'exécution du code). La Fig. 2.14 confirme le fait que les solveurs de type multigrille peuvent être efficacement parallélisés sur GPU. De plus, on constate que les performances du multigrille sur GPU deviennent importantes pour des maillages élevés, jusqu'à un gain de temps de calcul $\times 45$ supérieur par rapport au CPU pour une grille de 2049×2049. Ces constatations pourront également être faites dans le cas de simulations 3D où le solveur multigrille ne prend que 10% du temps de calcul.

Finalement, une corrélation peut être observée entre les résultats obtenus Table 2.2 et

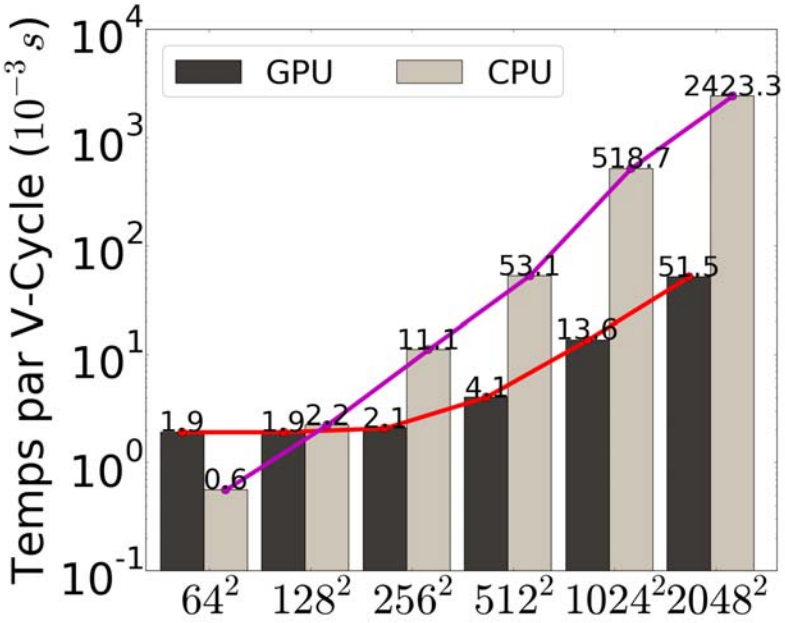


FIGURE 2.14 – Comparaison du temps de calcul entre le CPU et le GPU pour la résolution d'une solution particulière de l'équation de Poisson en fonction de la résolution de la grille à deux dimensions. Chaque V-cycle du multigrille est exécuté pour un nombre maximum de niveau de grille et accompli 8 itérations dans la routine de *pré* et de *post* relaxation. Ces résultats ont été obtenus avec l'utilisation de la carte graphique NVIDIA Tesla C2050.

les résultats de la Fig. 2.14 concernant l'évolution du rapport des temps d'exécution entre le CPU et le GPU. On remarque notamment que le fort gain de temps entre les deux cas de simulations, avec un maillage de 256×256 et 512×512 respectivement sur la Table 2.2 est également observé sur la Fig. 2.14 pour les mêmes tailles de grilles. Ceci confirme une fois de plus, le fort potentiel des cartes graphiques pour des simulations de hautes résolutions.

Nous pouvons voir à partir des résultats des temps de calcul que, lors de l'augmentation (linéaire) de la résolution de la simulation, soit en augmentant le nombre total de particules, soit par l'augmentation du nombre de noeuds ou encore en augmentant les deux, les performances du GPU augmentent de façon significative. Typiquement, on observe un gain de temps de l'ordre de $\times 12$ à $\times 22$ avec le code parallélisé sur GPU par rapport au temps de calcul sur CPU, dans les cas présentés ici. Ceci montre que le GPU a de meilleures performances pour des problèmes ayant de plus hautes résolutions ce qui est très encourageant pour le développement de simulations en 3D. Cependant, l'espace mémoire disponible sur les cartes graphiques actuelles ne permet pas des simulations requérant un nombre élevé de particules et une résolution de grille relativement fine. Bien qu'il existe des solutions pour palier au problème de la limitation de la mémoire, comme certaines fonctions permettant aux threads d'accéder à la mémoire du CPU (appelé *Zero copy*, Réf. [1]), les transferts de données entre le CPU et le GPU ou encore les lectures sur la mémoire du CPU peuvent

augmenter considérablement les temps de latences et doivent être utilisés avec précaution.

Les temps présentés sur la Fig. 2.15 montrent en pourcentage le temps de calcul passé dans chaque module du code PIC MCC dans le cas d'une simulation en 3D. On peut constater, en comparaison avec les temps présentés sur la Fig. 2.13 une forte similitude et le caractère dominant de la routine de calcul de la densité de charge (*rho*). Ceci renforce bien l'intérêt d'optimiser cette partie du code mais montre également le fort potentiel de la parallélisation en ce qui concerne les autres modules du code. Enfin, dans le cas des simulations 3D avec une grille comportant 64^3 cellules et 25 particules par cellule, soit $\sim 6.5 \times 10^6$ macroparticules, le temps total de calcul pour un temps d'évolution du plasma de 50μs est de l'ordre de 12 heures de calcul. Ainsi, on peut estimer un gain de temps supérieur à $\times 20$ le temps de calcul d'un code PIC 3D sur CPU.

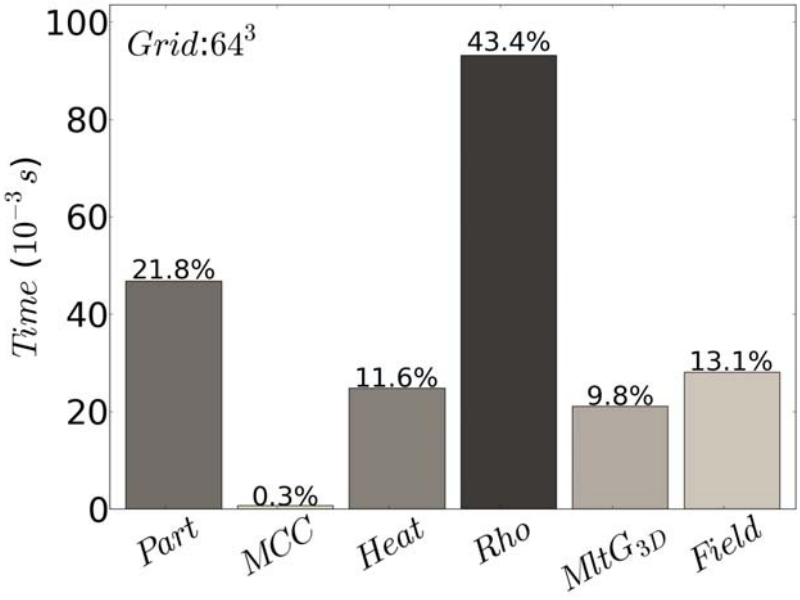


FIGURE 2.15 – Temps passé dans les différents modules du code PIC MCC 3D sur GPU dans le cas de simulations avec 25 particules par cellules et une grille de 64^3 noeuds. *Part* est le module de transport des particules, *MCC* est le module de collisions, *Heat* est le module d'injection de puissance, *Rho* est le module d'interpolation des charges sur la grille, $MltG_{3D}$ est le solveur de l'équation de Poisson et *Field* correspond au module d'interpolation du champ électrique.

Une autre forme de parallélisation plus simple et plus directe peut être utilisée pour la parallélisation de codes PIC. Ceci peut être fait par la parallélisation des boucles sur les particules à l'aide des librairies OpenMP (Réf. [61]) et en utilisant les différents coeurs disponibles sur le CPU. Différents niveaux de mémoires sont également disponibles (environs 3 niveaux sur les CPU actuels), mais la capacité de ceux-ci devient inversement propor-

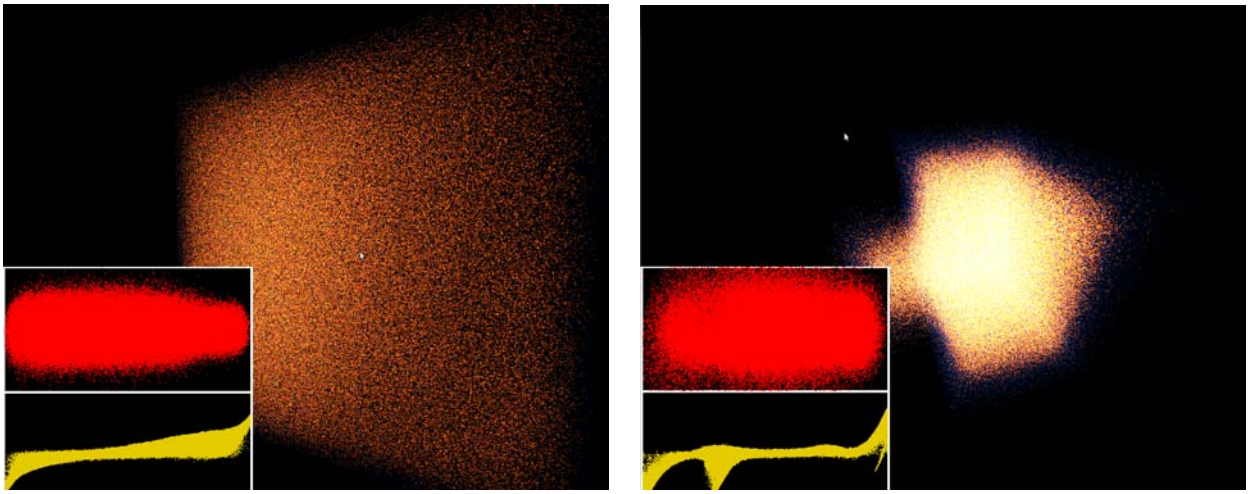
tionnelle à la vitesse d'accès à ces mémoires. Une version du code PIC a été parallélisée avec OpenMP et montre un gain de temps de $\times 3$ sur 4 processeurs. La littérature concernant la parallélisation avec OpenMP relate des gains de temps importants, cependant ces performances déclinent rapidement lorsque la taille des données atteint la taille limite de la mémoire cache. Cette limitation vient du fait que la bande passante mémoire devient insuffisante pour les simulations qui nécessitent un grand nombre d'accès à la mémoire. Dans cette situation, les GPU peuvent être une bonne alternative.

2.10 OpenGL ou la visualisation en temps réel

Comme dans le cas de nombreuses simulations numériques, la visualisation de certaines grandeurs caractéristiques et des résultats joue un rôle très important dans les simulations PIC.

Le code particulaire ayant été implémenté entièrement sur le GPU, il peut être avantageux d'utiliser ces processeurs pour la visualisation des données. De plus, le transfert des informations entre le GPU et le CPU peut être particulièrement coûteux en temps de calcul ; Ceci est dû au fait que les temps de latence sont longs et que la multiplication de ces différents accès en mémoire peut réduire significativement les performances de calcul. Ainsi, les données étant directement disponibles dans la mémoire vidéo, leurs visualisations *via* le GPU est bénéfique en terme d'utilisation des ressources et en gain de temps au niveau de l'accès des données en mémoire. Les cartes graphiques rendent alors possibles les visualisations en temps réel des données et c'est dans ce sens qu'il a été développé un code avec le langage OpenGL entièrement intégré dans le code PIC MCC implémenté en CUDA. Dans le cas des codes particulaires, les données liées aux particules telles que les positions et les vitesses de chacune d'entre elles font partie des données fondamentales pouvant être visualisées. Les images de la Fig. 2.16 montrent à un instant donné la fenêtre de visualisation OpenGL des particules dans un espace 2D dans le cas de la Fig. 2.16 (a) et 3D dans le cas de la Fig. 2.16 (b). De plus, l'encadré situé dans le coin bas à gauche des fenêtres de visualisation représente l'espace des phases (x, v_x) des électrons (coin haut de l'encadré, en rouge) et des ions (coin bas de l'encadré, en jaune). La simulation représentée sur l'image de Fig. 2.16 (a) correspond à une simulation de la source de plasma dans les conditions de la Fig. 2.12 où les électrons sont représentés par les points oranges et les ions par les points bleus (très peu visible sur l'image). La partie gauche de la source (où la densité de particules est plus importante) correspond à la zone de chauffage des électrons et la partie droite (où la densité semble plus faible) correspond à la zone de champ magnétique. La simulation montrée sur l'image de la Fig. 2.16 représente une simulation 3D dans la configuration de la source d'ions négatifs pour ITER. Sur la partie gauche de la source,

on distingue le "driver" où les électrons sont chauffés avant d'être ralentis dans la barrière magnétique à droite de la source.



(a) (b)

FIGURE 2.16 – Visualisation en temps réel des particules dans un espace en 2D (dans les conditions de la Fig. 2.12) (a) et 3D (dans la configuration de la source d'ions négatifs d'ITER) (b) grâce à l'interopérabilité de OpenGL et de CUDA. L'encadré situé dans le coin bas à gauche des fenêtre de visualisation représente l'espace des phases (x, v_x) des électrons (en haut, en rouge) et des ions (en bas, en jaune).

La visualisation de l'évolution des particules n'offre cependant pas une solution adaptée pour la visualisation de grandeurs macroscopiques. Il est néanmoins possible d'implémenter en OpenGL des outils permettant la projection de la grille et donc la visualisation de la densité, du flux, de la température, ..., etc, en temps réel ou dans le cadre de diagnostics des résultats.

2.11 Conclusion

Un modèle 3D Particle-In-Cell électrostatique avec les collisions Monte Carlo pour les plasmas à basse température a été implémenté pour opérer sur les cartes graphiques avec l'environnement CUDA. Le principe de la parallélisation sur GPU de chaque module du code est décrit dans les sections précédentes.

Les performances du code de simulation et le temps de calcul passé dans chaque module du code PIC sont comparés avec ceux d'un code PIC MCC similaire et opérant de façon séquentiel sur le CPU. Nous avons vu que l'implémentation du code sur GPU fait apparaître un gain de temps de calcul comprit entre 10 et 20 comparé à l'implémentation sur CPU et que la variation du rapport de temps d'exécution dépend de la résolution de la simulation.

Ces valeurs sont données pour une simulation en double précision sur le CPU et en simple précision sur le GPU. Nous noterons également qu'un très bon accord entre les résultats des deux codes a été obtenu.

Les meilleures performances du GPU comparées au CPU sont obtenues pour un grand nombre de particules et/ou un grand nombre de noeuds. Une attention toute particulière doit être portée aux simulations de plasmas à basse température dans des conditions réalistes car des particules sont générées (e.g. ionisation) et d'autres sont perdues (ici, aux parois) à chaque pas en temps. Une technique simple et rapide d'indexation a été implémentée et permet de réordonner les particules à chaque pas de temps. Le module le plus gourmand pour le GPU en terme de temps de calcul est l'assignement des charges sur les noeuds de la grille. Ce module prend à lui seul 40% du temps d'exécution total. Cependant, cette partie peut être optimisée et le gain de temps estimé est de l'ordre de 20% en utilisant un algorithme de tri plus sophistiqué. Le transport des particules est de l'ordre de 24% du temps de calcul total alors que le temps passé dans le module des collisions est de moins de 5%. Le solveur de l'équation de Poisson (ici, le multigrille) peut être efficacement implémenté sur GPU et correspond seulement à environ 10% du temps de calcul.

En conclusion, nous dirons que les cartes graphiques fournissent un moyen peu coûteux pour exécuter de manière intensive des simulations PIC MCC, avec un gain supérieur à un facteur 20 en temps de calcul comparé à ceux obtenus avec des processeurs CPU récents. Les simulations PIC MCC 3D peuvent être exécutées sur les GPU bien que la limitation de l'espace mémoire ne permet toujours pas l'exécution de simulations 3D avec un grand nombre de noeuds et de particules.

Chapitre 3

Modélisation du filtre magnétique et étude du transport électronique

3.1 Introduction

3.1.1 Introduction générale :

Le terme de plasma a été introduit dans les années 1920-1930 par le physicien I. Langmuir (Nobel de physique en 1932). A cette époque, plusieurs travaux de recherches étaient principalement dirigés vers la compréhension de plusieurs phénomènes tels que : les oscillations collectives de gaz ionisés dans les tubes à décharge (I.Langmuir et L.Tonks) et le phénomène d'écrantage électrique dans les électrolytes (P.Debye et E. Hückel), qui joue un rôle fondamental dans les plasmas, où encore, la propagation des ondes radio à grande distance dans l'ionosphère. Ce rôle fut élucidé par le physicien H. Alfvén (prix Nobel de physique en 1970) qui développa en 1942 la théorie des ondes magnétohydrodynamiques plus connue aujourd'hui sous le nom d'*ondes d'Alfvén*.

C'est à partir des années 1950 que les études sur les plasmas connaissent une envolée majeure. Les premiers satellites permettent l'exploration et les mesures *in situ* des différentes couches atmosphériques et de la magnétosphère terrestre qui vont mettre au grand jour des phénomènes encore inconnus tels que les vents solaires (flux de plasma éjecté du Soleil et ayant une vitesse de l'ordre de $300 - 800$ km/s) et les ceintures de Van Allen, constituées de particules chargées piégées dans le champ magnétique issu du dipôle magnétique terrestre. D'un autre coté, Les USA, la grande Bretagne et l'Union Soviétique vont démarrer simultanément des recherches en physique des plasmas basées sur la fusion thermonucléaire dans un cadre militaire.

Les années 1960 voient d'importants efforts de recherches dirigés vers l'utilisation des plasmas pour la propulsion spatiale. Les propulseurs à plasma utilisent les forces électro-

statiques ou électromagnétique afin d'accélérer les ions du plasma à des vitesses élevées. La fonction principale de ces propulseurs est avant tout la correction d'orbite et les contrôles d'attitude des satellites. Cependant, on commence à les utiliser pour des voyages interplanétaires (missions *Deep Space 1* [62]). Ils sont considérés plus sérieusement pour de futurs projets de navigation spatiale.

Les années 1980 et 1990 voient de nouvelles applications et de nouvelles études apparaître. Ce que l'on appelle aujourd'hui le *"Plasma processing"* est une utilisation des plasmas pour la fabrication de très petits et complexes circuits intégrés utilisés dans les appareils électroniques modernes. Les études liées aux "plasmas de poussières" (en anglais, *dusty plasmas*) démarrent également dans ces années là. Le principe repose sur l'immersion de grains de poussières dans un plasma qui deviennent électriquement chargés et se comportent ensuite comme une espèce supplémentaire de particules chargées. Ces grains étant beaucoup plus massifs que les ions et les électrons du plasma, des comportements collectifs complètement différents peuvent avoir lieu, conduisant à une nouvelle physique.

En plus des activités énoncées ci-dessus, des recherches dans le cadre de l'industrie, sont également menées sur des sujets comme les arcs électriques, les torches à plasmas, les laser à décharge, ..., etc.

International Thermonuclear Experimental Reactor Après la fin de la guerre froide (à partir de 1958), au vue des faibles avancements dans le domaine de la fusion thermonucléaire, les USA, l'Union Soviétique ainsi que la Grande Bretagne commencèrent à mettre en commun leurs efforts de recherche. Aujourd'hui, de nombreux pays tels que la France, l'Allemagne, l'Inde, le Japon participent activement à la recherche sur la fusion.

Peu d'avancées sur la fusion eurent lieu au cours des années 1960, et il fallut attendre la première configuration Russe du *tokamak* avant d'avoir des résultats encourageants. Dans les années $1970 - 1980$, plusieurs tokamaks ont été construits avec des performances grandissantes. La fin des années 1970 confirma les propriétés de confinement des configurations de type tokamak. Un chauffage s'est alors avéré nécessaire et permit d'atteindre des températures de l'ordre de plusieurs KeV.

En 2006 à Paris, un accord international **ITER**, structurant les efforts de recherche de l'Europe, des États-Unis, de la Chine, de l'Inde, de la Russie, de la Corée du Sud et du Japon, marque l'aboutissement de cinquante années de recherches sur la fusion thermonucléaire dans les tokamaks et le début du premier projet de coopération scientifique à l'échelle mondiale.

Les températures à l'intérieur du tokamak ITER doivent atteindre 150 millions de degrés Celsius, soit dix fois la température au coeur du soleil. Ces conditions de température doivent être réunies pour que le gaz situé à l'intérieur du tokamak atteigne un état plasma et que les réactions de fusion aient lieu. Une fois que ces processus sont établis, le plasma

chaud doit être maintenu à ces températures de façon contrôlée et dans le but d'en extraire l'énergie.

Trois dispositifs externes de chauffage sont mis en place et travaillent de concert pour fournir une puissance de chauffage de 50 MW afin d'atteindre les températures nécessaires à la fusion. Le chauffage du plasma du tokamak est réalisé principalement par l'injection de faisceaux de neutres (également appelé NBI pour *Neutral Beam Injection*) mais également par le biais de deux sources d'ondes électromagnétiques de hautes fréquences telles que le chauffage Ohmique et le chauffage radio fréquence. Ces dispositifs sont représentés sur la Fig. 3.1.

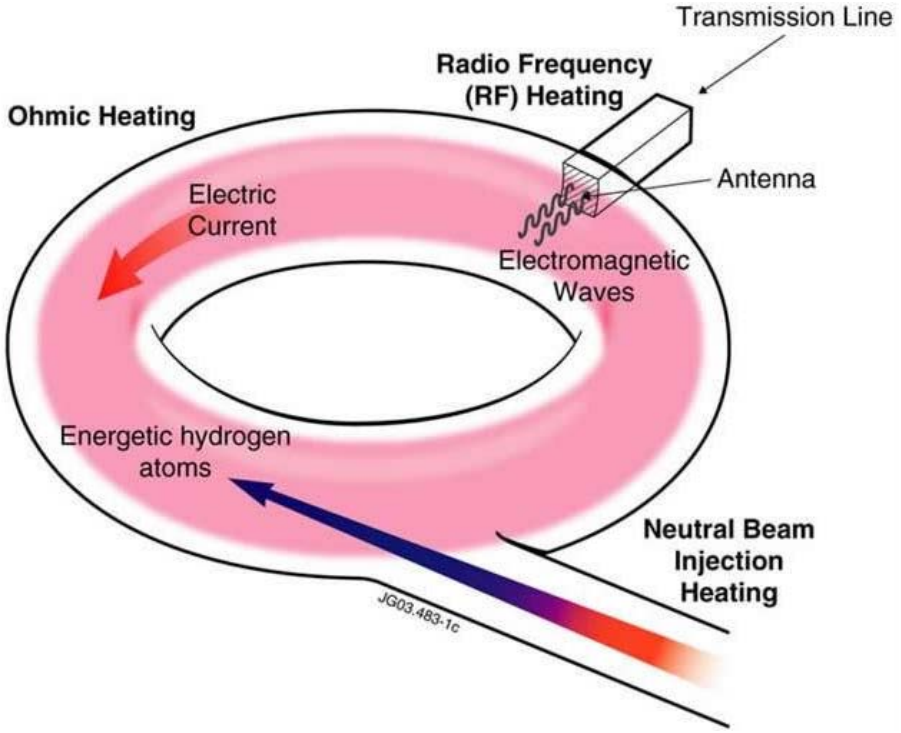


FIGURE 3.1 – Schéma représentant les trois dispositifs de chauffage de la source de plasma du tokamak d'ITER. Le chauffage est réalisé par injection de faisceaux de neutres (NBI) et par deux sources d'ondes électromagnétiques (ohmique et radio fréquence) hautes fréquences (Sources Réf. [63]).

L'injecteur de neutres pour ITER (en anglais, *neutral beam injection*), joue un rôle essentiel dans le chauffage du plasma de fusion de ITER. L'injection de neutres est la principale méthode produisant de hautes températures et de grandes performances de chauffage dans la configuration des tokamaks.

Cette méthode de chauffage consiste en l'injection de faisceaux de très hautes énergies d'atomes neutres de deutérium, dans le coeur du plasma de fusion. Le faisceau d'atome pénètre le plasma de tokamak en transférant l'énergie cinétique des particules par collision

avec le plasma augmentant ainsi la température générale. Dans le cadre du projet ITER, deux faisceaux d'ions négatifs de l'ordre du MeV, correspondant à un courant total de 40 A vont être neutralisés et injectés dans le tokamak (Réfs. [64, 65, 66]).

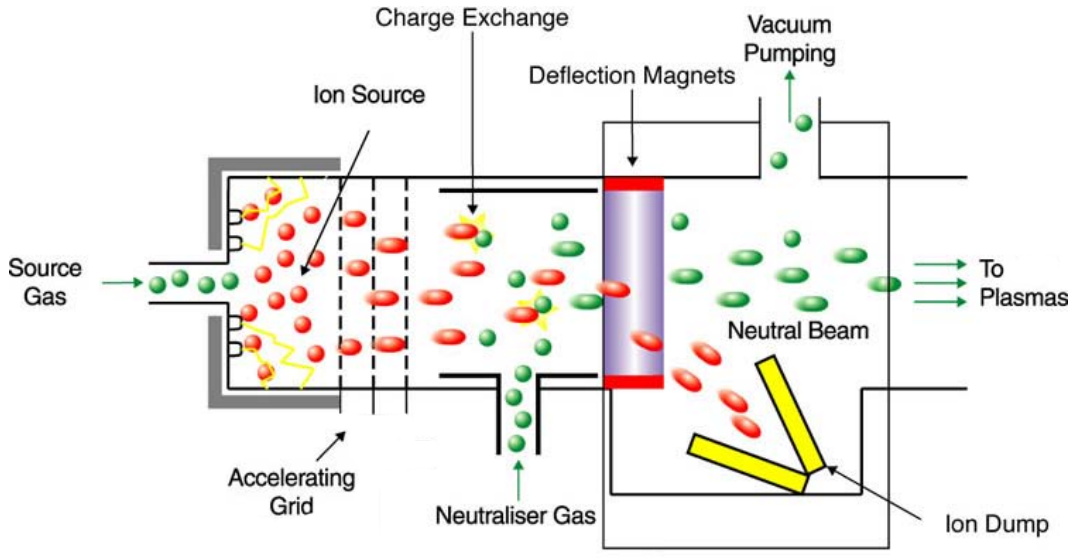


FIGURE 3.2 – Principe de fonctionnement de l'injecteur de neutres. (Sources Réf. [67])

Seuls les atomes ayant une charge négative ou positive peuvent être accélérés par le champ électrique. Une fois les ions extraits, le processus inverse de neutralisation doit être opéré avant l'injection des particules dans le plasma du tokamak. Dans le cas contraire, les particules électriquement chargées seraient défléchies par le champ magnétique qui permet le confinement du plasma dans le tokamak. Ainsi, après extraction et accélération des ions issus de la source d'ions, ceux-ci vont être neutralisés lors de leur passage dans une chambre contenant du gaz et leur permettant de récupérer un électron s'il s'agit d'ions positifs, ou de perdre un électron s'il s'agit d'ions négatifs, avant d'être injecté dans le plasma.

Les très grandes échelles imposées par la construction du tokamak ITER impliquent de nouvelles conditions concernant la méthode d'injection. Les neutres sont injectés à des très hautes énergies afin de les faire pénétrer suffisamment à l'intérieur du plasma. Cependant, à ces hautes énergies, il est plus difficile de neutraliser les espèces chargées positivement que les espèces chargées négativement. C'est pourquoi il a été décidé d'utiliser une source d'ions négatifs pour l'injecteur de neutres apportant toutefois des contraintes sur la séparation des électrons et des ions possédant la même charge.

Les ordres de grandeurs de la source d'ions sont : une densité de plasma de $10^{18} - 10^{19}$ m^{-3} et une température électronique de $2 - 20$ eV.

Le "design" de la source d'ion négatif pour l'injecteur de neutres consiste en une zone

de chauffage, appelée *driver*, une zone d'expansion du plasma, un filtre magnétique et une grille d'extraction. Cette géométrie est représentée sur la Fig. 3.3 (voir également Réfs. [65, 66, 68, 69]).

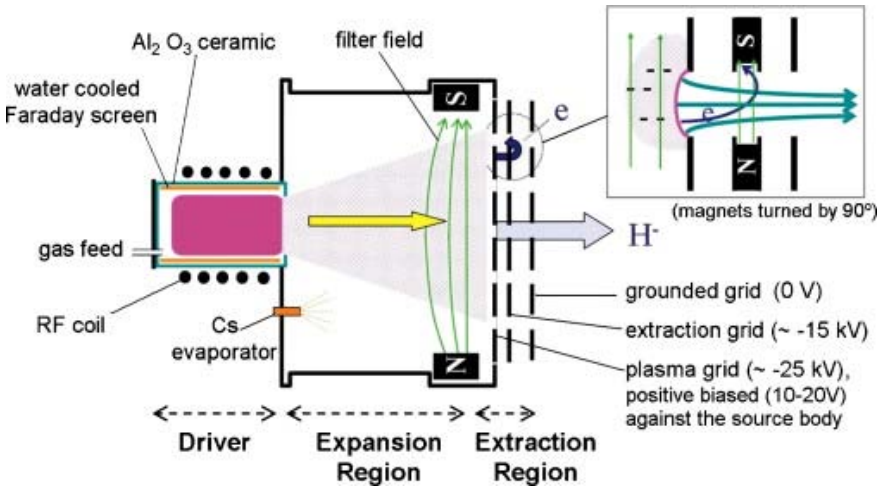


FIGURE 3.3 – Schéma de la source d'ion négatif pour le projet ITER (Source [69]).

Le plasma est généré dans le driver où les électrons sont chauffés par couplage radio fréquence inductif avant de s'étendre dans la zone d'expansion et de rencontrer le filtre magnétique. Ce filtre joue un rôle majeur dans l'extraction des charges, et est utilisé pour ralentir les électrons (par la diminution de la température) et de minimiser le flux d'électrons vers la grille d'extraction pour extraire en majorité les ions négatifs. Cependant, le filtre magnétique ne limite pas suffisamment le courant d'électrons vers la grille d'extraction, et un autre champ magnétique est appliqué directement au niveau de la grille plasma pour déflecter les électrons vers une grille extractrice.

3.1.2 Problématique sur le transport des particules à travers le filtre magnétique

Dans ce qui suit, nous allons nous intéresser principalement au transport des électrons à travers le filtre magnétique. Plusieurs études similaires ont été effectuées en 1D et plus récemment en 2D (cf. [23, 70, 24, 22]) et montrent un comportement très différents d'une dimension à l'autre. A 1D, le transport des électrons à travers la barrière n'est possible qu'avec l'intervention de processus collisionnels (en majorité, due aux collisions Coulombiennes). En 2D, les forces électromagnétiques associées à la présence des parois font apparaître des dérives électroniques qui deviennent les principaux responsables de ce transport. Il devient alors intéressant de se demander :

- Que devient le rôle des collisions dans le transport électronique à travers le filtre et y a t-il d'autre phénomènes intervenant dans ce transport ?
- Quelle est l'influence du filtre magnétique sur le flux des particules et peut-on le quantifier ?
- Quels sont les effets du filtre magnétique sur les paramètres caractéristiques du plasma (potentiel, densité, température, pression) ?
- Une réelle différence dans le transport apparaît entre les simulations 1D et 2D due aux parois dans le cas 2D. Quelle peut être l'effet des parois dans la troisième dimension et change t-il significativement le transport des électrons à travers la barrière magnétique ?

Après un rappel sur quelques généralités de la théorie de la physique des plasmas, nous allons définir à partir des résultats du modèle PIC MCC 2D3V, quels sont les effets du champ magnétique sur le plasma. Dans un premier temps nous verrons comment évolue le plasma et la densité de courant électronique pour différentes tensions appliquées à une paroi de la source, avant dans un second temps, d'ajouter le filtre magnétique et d'analyser les comportements et ces effets sur le plasma et le transport des électrons. Enfin, nous terminerons ce chapitre par une étude et une comparaison des phénomènes observés en 2D et ceux obtenus à partir des résultats du code PIC MCC 3D. La simulation 3D permettra de mettre en évidence les comportements collectifs des électrons dus aux parois dans les trois dimensions.

3.2 Bref rappels

3.2.1 Approche cinétique

Le *"milieu"* plasma est un milieu complexe à décrire en raison du fort couplage entre les particules chargées et les champs électriques et magnétiques appliqués et induits.

Ce milieu peut être décrit par la théorie cinétique basée sur les solutions de l'équation de Boltzmann que l'on présente dans ce qui suit.

À un temps t donné, chaque particule peut être repérée par une position $\mathbf{r} = (x, y, z)$ et une vitesse $\mathbf{v} = (v_x, v_y, v_z)$ spécifique. On peut alors caractériser un grand nombre de particules en spécifiant une densité de particules en chaque point $\mathbf{r}, \mathbf{v}$ de l'espace des phases. La fonction permettant de décrire cette densité de particules (pour une espèce donnée) dans l'espace des phases est appelée *fonction de distribution* et est notée $f(\mathbf{r}, \mathbf{v}, t)$. Alors l'interprétation de $f(\mathbf{r}, \mathbf{v}, t)d^3rd^3v$ est le nombre de particules à l'intérieur du volume d^3rd^3v aux coordonnées $(\mathbf{r}, \mathbf{v})$ et au temps t. Au cours du temps, le mouvement des particules et l'accélération due aux forces macroscopiques induisent un flux entrant et sortant du volume de l'espace des phases d^3rd^3v. L'évolution temporelle de f donne une description détaillée

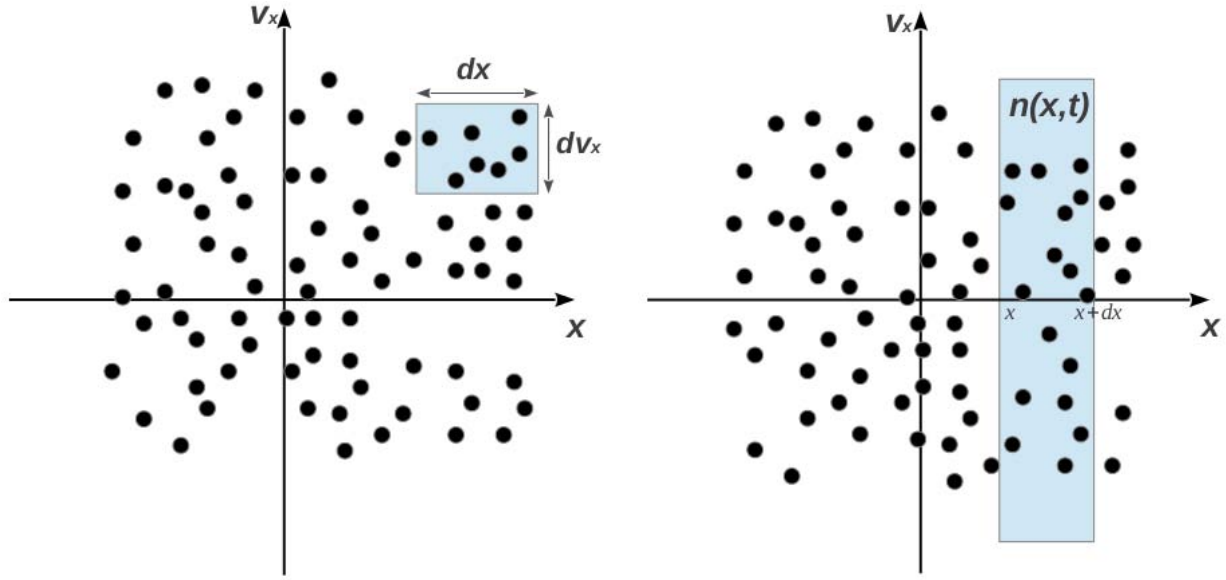


FIGURE 3.4 – a) Illustration à une dimension de la boîte de volume $dxdv_x$ dans l'espace des phases, b) moyenne sur les vitesses de l'espace des phases et obtention du moment d'ordre 0 de l'équation de Boltzmann.

du système mais ne permet pas de suivre chaque particule individuellement.

De manière générale, la variation du nombre de particules à l'intérieur de la boite de volume d^3rd^3v peut être définie comme $\frac{\partial f(\mathbf{r},\mathbf{v},t)}{\partial t}d^3rd^3v$. L'illustration de la définition de f dans l'espace des phases projetée sur (x, v_x) à une dimension est représentée Fig. 3.4(a).

Si l'on restreint f au plan (x, v_x), on peut écrire :

- Le flux entrant par la face du bas de la boîte est : $f(x, v_x, t)a_x(x, v_x, t)dx$.
- Le flux sortant par la face du haut de la boîte est : $-f(x, v_x+dv_x, t)a_x(x, v_x+dv_x, t)dx$.
- Le flux entrant par la face gauche de la boîte est : $f(x, v_x, t)v_xdv_x$.
- Le flux sortant par la face droite de la boîte est : $-f(x + dx, v_x, t)v_xdv_x$.

Où $a(x, v_x, t) = dv_x/dt$ est l'accélération de la particule. Alors, à une dimension, la fonction de distribution peut être dérivée comme suit :

$$\begin{aligned}\frac{\partial f(x,v_x,t)}{\partial t}dxdv_x = & \quad \big[f(x,v_x,t)a_x(x,v_x,t) - f(x,v_x+dv_x,t)a_x(x,v_x+dv_x,t)\big]dx \\ + & \quad \big[f(x,v_x,t)v_x - f(x+dx,v_x,t)v_x)\big]dv_x \end{aligned} \tag{3.1}$$

En divisant par $dxdv_x$, on obtient :

$$\frac{\partial f(x, v_x, t)}{\partial t} = -\frac{\partial}{\partial v_x}(f a_x) - \frac{\partial}{\partial x}(f v_x) \tag{3.2}$$

Où x et v_x sont des quantités indépendantes dans l'espace des phases. On peut donc écrire :

$$\frac{\partial}{\partial x}(f v_x) = v_x \frac{\partial}{\partial x} f$$

De plus, l'accélération de la particule est donnée par la force de Lorentz :

$$\mathbf{a} = \frac{q}{m}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \tag{3.3}$$

et le terme $[v \times B]_x = v_y B_z - v_z B_y$ est indépendant de la composante x, on peut donc écrire également :

$$\frac{\partial}{\partial v_x}(f a_x) = a_x \frac{\partial}{\partial v_x}(f)$$

Finalement, à partir de ces propriétés de commutation, l'Eq. (3.2) peut se réécrire :

$$\frac{\partial f(x, v_x, t)}{\partial t} = -a_x \frac{\partial}{\partial v_x} f - v_x \frac{\partial}{\partial x} f$$

La généralisation à trois dimensions donne :

$$\frac{\partial f(\mathbf{r}, \mathbf{v}, t)}{\partial t} + \mathbf{v} \cdot \nabla_r f + \mathbf{a} \cdot \nabla_v f = 0 \tag{3.4}$$

avec $\nabla_r = (\frac{\partial}{\partial x}\mathbf{e}_x + \frac{\partial}{\partial y}\mathbf{e}_y + \frac{\partial}{\partial z}\mathbf{e}_z)$, $\nabla_v = (\frac{\partial}{\partial v_x}\mathbf{e}_x + \frac{\partial}{\partial v_y}\mathbf{e}_y + \frac{\partial}{\partial v_z}\mathbf{e}_z)$. L'Eq. (3.4) est appelée *équation de Vlasov*.

Cependant, nous devons considérer en plus du flux de particules entrant et sortant du volume $dxdv_x$ de l'espace des phases (cf. Fig. 3.4(a)), les collisions binaires entre les particules. Ceci peut être vu comme des apparitions ou des disparitions de particules dans le volume dus à l'échelle de temps très petite de ces collisions. De telles collisions affectent seulement la vitesse des particules et non la position. Ainsi l'ajout d'un terme de création et de perte doit être pris en compte dans le membre de droite de l'Eq. (3.4). L'équation

que l'on obtient alors est appelée *équation de Boltzmann* (cf. Eq. (3.5)).

$$\frac{\partial f(\mathbf{r},\mathbf{v},t)}{\partial t} + \mathbf{v}\cdot\nabla_r f + \mathbf{a}\cdot\nabla_v f = \frac{\partial f}{\partial t}\Big|_{col} \tag{3.5}$$

La complexité de ces équations régissant la dynamique des particules peut être grandement simplifiée en moyennant la fonction de distribution sur l'espace des vitesses afin d'obtenir des équations ne dépendant que de la position et du temps.

Soit le nombre de particules représentées dans la bande bleue Fig. 3.4(b) comprises entre x et $x+dx$, alors ce nombre est équivalent à la densité de particules $n(x,t)$ au point x et au temps t. On s'aperçoit alors que l'on peut obtenir des quantités moyennées, i.e. des quantités macroscopiques telles que la densité $n(\mathbf{r},t)$, la vitesse moyenne $u(\mathbf{r},t)$ ou encore l'énergie moyenne $\varepsilon(\mathbf{r},t)$, à partir de ce que l'on appelle les *moments en vitesse* de la fonction de distribution. On généralise ces *"moments"* en prenant l'intégrale sur l'espace des vitesses à trois dimensions, soit pour la densité :

$$n(\mathbf{r},t) = \int f(\mathbf{r},\mathbf{v},t)d^3v \tag{3.6}$$

et le flux :

$$n(\mathbf{r},t)\mathbf{u}(\mathbf{r},t) = \int \mathbf{v} f(\mathbf{r},\mathbf{v},t)d^3v \tag{3.7}$$

Où $\mathbf{u}(\mathbf{r},t)$ correspond à la vitesse moyenne des particules.

Conservation de la masse Nous allons maintenant relier l'équation de Boltzmann (Eq. (3.5)) aux quantités macroscopiques $n(\mathbf{r},t), \mathbf{u}(\mathbf{r},t)$ en intégrant l'Eq.(3.5) sur l'espace des vitesses. La première et la plus simple des procédures nous conduit au *moment d'ordre* 0 de l'équation de Boltzmann et est donnée par :

$$\int \Big[\frac{\partial f(\mathbf{r},\mathbf{v},t)}{\partial t} + \mathbf{v}\cdot\nabla_r f + \mathbf{a}\cdot\nabla_v f\Big] d^3v = \int \frac{\partial f}{\partial t}\Big|_{col} d^3v \tag{3.8}$$

soit, l'équation de continuité ou de conservation de la masse :

$$\boxed{\frac{\partial n}{\partial t} + \nabla\cdot(n\mathbf{u}) = S} \tag{3.9}$$

Lors de la résolution de cette intégration (Eq. (3.8)), il faut prendre en compte, le fait que les variables $\mathbf{r}, \mathbf{v}$ et t sont des variables indépendantes. De plus, on peut montrer que le troisième terme du membre de gauche de l'équation (3.8) est négligeable en montrant que l'intégrale sur la surface de la fonction de distribution $f \to 0$ lorsque $v \to \infty$. Enfin, l'intégration du membre de droite de l'équation (3.8) est nulle dans le cas où les collisions ne modifient pas le nombre de particules. Dans le cas de processus de création ou de perte de particules ce terme n'est plus nul. Dans le cas des plasmas à basse pression les processus de création sont grandement dus à l'ionisation lors de collisions électrons-neutres et les processus de perte telle que la recombinaison sont souvent négligeables. Si de plus, l'ionisation par impact électronique direct des atomes ou des molécules est dominante, le terme de droite de l'équation (3.9) peut s'écrire :

$$S = \nu_i n_e \tag{3.10}$$

avec ν_i la fréquence d'ionisation et n_e la densité électronique.

Conservation des moments Multiplions maintenant l'Eq. (3.8) par $\mathbf{v}$ en intégrant sur l'espace des vitesses. Nous obtenons alors le *moment d'ordre* 1 de l'équation de Boltzmann.

$$\int \mathbf{v}\Big[\frac{\partial f(\mathbf{r},\mathbf{v},t)}{\partial t} + \mathbf{v}\cdot\nabla_r f + \mathbf{a}\cdot\nabla_v f\Big] d^3v = \int \mathbf{v}\frac{\partial f}{\partial t}\Big|_{col} d^3v \tag{3.11}$$

La résolution de ces intégrales demande quelques détails supplémentaires pour le calcul de tenseurs. Plusieurs documents, livres présentent de façon détaillée ce calcul, Réfs. [71, 72, 73]. L'Eq. (3.11) que multiplie la masse d'une particule m, peut se réduire finalement à :

$$m\Big[\frac{\partial n\mathbf{u}}{\partial t} + n(\mathbf{u}\cdot\nabla_r\mathbf{u})\Big] = nq(\mathbf{E}+\mathbf{u}\times\mathbf{B}) - \nabla_r\widehat{\mathbf{P}} - \mathbf{R}_\alpha \tag{3.12}$$

Où le terme du membre de droite de l'Eq. (3.12) représente le taux de transfert d'impulsion par unité de volume entre l'espèce considérée et l'espèce α, soit $\mathbf{R}_\alpha = mn\nu_{en}\mathbf{u}$ (on néglige la vitesse de l'espèce α par rapport à celle de l'espèce considérée due au rapport de masse important des deux espèces). Le deuxième terme du membre de droite représente la divergence du tenseur de pression où $\widehat{\mathbf{P}}$ est défini par : $\widehat{\mathbf{P}} = m\int VVfd^3V$. Dans le cas des plasmas faiblement ionisés, l'approximation de diagonalité et d'isotropie du tenseur de pression est souvent utilisée. Cette approximation permet de réduire tous les termes non

diagonaux du tenseur (termes liés à la viscosité) $\widehat{\mathbf{P}}$ à 0 et de garder la trace des trois termes diagonaux de $\widehat{\mathbf{P}}$. Ceci simplifie le terme de tenseur de pression en terme de pression scalaire tel que : $P = \frac{m}{3}\int V \cdot V f dV$. On peut définir une "température" T de l'espèce considérée par $P = nk_bT$, soit :

$$\nabla P = \nabla(k_bTn) \tag{3.13}$$

Où k_b est la constante de Boltzmann et T la température d'agitation thermique en Kelvin.

À partir des hypothèses citées précédemment, on peut réécrire le moment d'ordre 1 de l'équation de Boltzmann :

$$\boxed{mn\Big[\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla_r\mathbf{u})\Big] = nq(\mathbf{E}+\mathbf{u}\times\mathbf{B}) - \nabla P - mn\nu_{en}\mathbf{u}} \tag{3.14}$$

3.2.2 Approche macroscopique

Nous venons de voir dans les paragraphes précédents que le mouvement des particules dans un plasma peut être entièrement caractérisé par l'équation de Boltzmann ou, moyennant des approximations, par les moments de l'équation de Boltzmann. Revenons à l'Eq. (3.14) et considérons maintenant un plasma à l'état stationnaire. À partir de cette hypothèse, on peut écrire que $\frac{\partial}{\partial_t} \to 0$. De plus, une autre hypothèse peut être considérée, mais doit être utilisée avec précaution. De façon générale, pour les plasmas collisionnels, lorsque le libre parcours moyen des particules chargées est faible devant les dimensions caractéristiques du plasma, on peut souvent négliger l'énergie moyenne dirigée devant l'énergie d'agitation thermique des électrons. Cela revient à négliger le terme d'inertie (second terme du membre de gauche de l'équation (3.14)) devant le terme de gradient de pression cinétique (second terme du membre de droite). Ainsi l'équation de quantité de mouvement (Eq. (3.14)) peut se réécrire :

$$\Gamma = n\mathbf{u} = n\mu(\mathbf{E}+\mathbf{u}\times\mathbf{B}) - \nabla(Dn) \tag{3.15}$$

Où Γ correspond au flux des particules chargées, et où μ et D sont les coefficients de transports, appelés respectivement coefficients de *mobilité* et de *diffusion*.

$$\mu = \frac{q}{m\nu} \qquad D = \frac{k_bT}{m\nu} \tag{3.16}$$

Dans les simulations présentées dans ce chapitre, le domaine de simulation est représenté par un domaine cartésien (x, y) où le champ magnétique est dirigé perpendiculairement au plan de simulation. Soit les composantes (x, y) du flux Γ et d'après l'équation (3.15), on pose pour un champ magnétique perpendiculaire au plan (x, y) : $\mathbf{B} = (0, 0, B_z)$

$$\begin{aligned}\Gamma_x &= nu_x = n\mu E_x - \nabla(Dn) + n\mu u_y B_z \\ \Gamma_y &= nu_y = n\mu E_y - \nabla(Dn) - n\mu u_x B_z\end{aligned} \tag{3.17}$$

Posons, afin de simplifier les équations : $\Gamma^*_{x,y} = n\mu E_{x,y} - \nabla(Dn)$ qui n'est autre que le terme de flux des particules chargées sans champ magnétique (forme dite "dérive-diffusion" du flux). Alors les équations (3.17) deviennent :

$$\begin{aligned}\Gamma_x &= \Gamma^*_x + n\mu\Gamma_y B_z \\ \Gamma_y &= \Gamma^*_y + n\mu\Gamma_x B_z\end{aligned}$$

En remplaçant Γ_x et Γ_y dans les deux équations précédentes et après factorisation, on obtient :

$$\begin{aligned}\Gamma_x[1 + \mu^2 B_z^2] &= \Gamma^*_x + \mu\Gamma^*_y B_z \\ \Gamma_y[1 + \mu^2 B_z^2] &= \Gamma^*_y - \mu\Gamma^*_x B_z\end{aligned}$$

Soit, finalement l'expression vectorielle du flux Γ :

$$\boxed{\Gamma = \frac{1}{(1 + h^2)}(\Gamma^* - \Gamma^* \times \mathbf{h})} \tag{3.18}$$

Où $h = \frac{\omega}{\nu}$, appelé le paramètre de Hall et avec $\omega = \frac{q\mathbf{B}}{m}$, la fréquence cyclotronique. Ce dernier paramètre est important car il permet de caractériser le transport des électrons en présence du champ magnétique.

Si la fréquence de collision est nettement supérieure à la fréquence cyclotronique, alors $h \ll 1$ est le champ magnétique a un effet mineur sur les particules chargées. *A contrario*, si $h \gg 1$ alors les particules sont fortement magnétisées. Autrement dit, lorsque $\omega \gg \nu$ comme dans le cas des particules à l'intérieur du champ magnétique, le deuxième terme de l'Eq. (3.18) (e.g. $(\Gamma^* \times \mathbf{h})/(1+h^2)$) dirigé perpendiculairement à la composante du flux, fait apparaître des termes de dérive croisée. Ces dérives sont représentées par la dérive $\mathbf{E} \times \mathbf{B}$ et

la dérive diamagnétique $\nabla(nk_bT) \times \mathbf{B}$. Dans ces conditions, l'Eq. (3.18) peut être réécrite :

$$\Gamma = n\frac{\mathbf{E} \times \mathbf{B}}{B^2} + \frac{\nabla(nk_bT) \times \mathbf{B}}{B^2} \tag{3.19}$$

Après avoir défini et caractérisé le modèle bidimensionnel utilisé pour la simulation de la source de plasma dans la section suivante, nous verrons dans le cadre de l'étude sur le transport électronique à travers une barrière magnétique que celui-ci peut être caractérisé par l'Eq. (3.19) présentée ci-dessus.

3.3 Modèle 2D de la source de plasma

Le modèle présenté ici est un un modèle à deux dimensions dans l'espace et trois dimensions dans l'espace des vitesses (2D3V). Les espèces considérées dans la simulation sont les électrons (e), les ions positifs de dihydrogène (H_2^+) et les molécules neutres de dihydrogène (H_2). Comme il a été également expliqué dans la section 2.7, les molécules neutres sont supposées avoir une distribution uniforme sur tout le domaine de simulation. Ainsi les neutres ont une densité n_{H_2} et une température T_{H_2} fixées telle que la relation $P_n = n_{H_2}k_BT_{H_2}$ soit satisfaite, où P_n est la pression du gaz de neutres et k_B la constante de Boltzmann. Les molécules neutres sont supposés également avoir une distribution en vitesses Maxwellienne. De plus, les neutres sont beaucoup moins énergétiques que les électrons et le rapport de masse électron-neutre est important, on peut dès lors supposer que la variation de la quantité de mouvement des neutres reste négligeable dans la plupart des collisions électrons-neutres.

A chaque pas de temps δt, une fréquence de collision est calculée pour chaque particule ayant une probabilité importante de faire une collision dans cet intervalle de temps. La fréquence de collision est donnée par $\nu_{coll} = n_{H_2}\sigma v_r$ où n_{H_2} est la densité de gaz H_2, σ est la section efficace de collision correspondant au type de collision (e.g. élastique, inélastique ou ionisation) et v_r est la vitesse relative et est donnée par $v_r = v_e - v_{H_2}$ où v_e et v_{H_2} sont respectivement la vitesse de l'électron incident et la vitesse de la particule cible du gaz H_2. Toutefois, $v_e \gg v_{H_2}$ permet de faire l'approximation que $v_r \approx v_e$.

Comme il a été explicité dans la section relative au modèle Monte Carlo (cf. Section 2.7), le modèle de la source de plasma présenté ici est un modèle très simplifié du point de vue des collisions et de la chimie du plasma. Ainsi, plusieurs approximations ont été faites et l'on considère ici que les collisions ions-neutres et les collisions Coulombiennes sont négligeables dans ce cas idéal de la simulation de la source de plasma. Cependant, d'autres simulations dans les mêmes conditions de la source de plasma ont été réalisées en tenant compte des collisions Coulombiennes (Réf. [21]) et montrent que l'évolution du

TABLE 3.1 – **Processus collisionnels**

N^o réaction	Processus	Références
(1)	$e + H_2 \rightarrow e + H_2$ (Collision élastique)	LXcat [58]
(2)	$e + H_2 \rightarrow e + H_2^*$ (Excitation)	LXcat [58]
(3)	$e + H_2 \rightarrow e + H_2^+ + e$ (Ionisation)	LXcat [58]

plasma et le transport des particules à travers le champ magnétique ne dépend pas de ces collisions contrairement à ce qui est montré dans le cas des simulations à 1D où les collisions Coulombiennes sont les principales responsables du transport des électrons à travers le filtre magnétique (Réf. [74]).

Les processus collisionnels pris en compte pour la simulation sont restreints aux collisions électrons-neutres et prennent en compte les processus de collisions élastiques, d'ionisation et d'excitation. Ces processus sont résumés dans la Table 3.1.

Les sections efficaces de collisions pour les différents types de collisions sont données dans la littérature et sont généralement fonction de l'énergie relative $M_r v_r^2/2$ où M_r est la masse réduite donnée par $M_r = m_p m_{H_2}/(m_p + m_{H_2})$ avec m_p la masse des électrons ou des ions et m_{H_2} la masse des molécules de gaz H_2, ou en fonction de l'énergie dans le référentiel de la particule (référentiel du laboratoire) $m_p v_p^2/2$ avec v_p la vitesse de la particule incidente. Les sections efficaces concernant les impacts électroniques avec les molécules de gaz H_2 sont représentées sur la Fig. 3.5. Les sections efficaces d'excitation et d'ionisation ont un seuil qui est fonction de l'énergie cinétique dans le référentiel du laboratoire et à partir duquel il devient possible pour la particule incidente d'exciter ou d'ioniser une molécule H_2. Dans ce cas, l'électron incident perd exactement l'énergie seuil U où sa vitesse après la collision est $v_e^* = (v_e^2 - 2eU/m_e)$. Dans le cas de l'ionisation, l'énergie cinétique restante de la particule incidente est distribuée entre l'électron incident et l'électron éjecté. De plus, la distribution de l'angle de diffusion de la particule après la collision est supposée isotrope dans le référentiel du centre de masse (cf. Annexe B.1).

Dans ce problème simplifié de transport, la géométrie du système est 2D rectangulaire non périodique dans chaque direction. La dimension du domaine de simulation est de 0.2 m $\times$ 0.2 m (cf. Fig. 3.6). La chambre de décharge est remplie de dihydrogène avec une densité de 5×10^{19} m^{-3}. Le maintien du plasma est réalisé par un *"chauffage"* des électrons. Plusieurs méthodes peuvent être employées. Il est possible de chauffer les électrons en utilisant un processus "Maxwellien" présenté par St. Kolev (Réfs. [23, 24]), qui force le système à tendre vers une distribution Maxwellienne à une température donnée. Cette méthode permet donc de maintenir le plasma à une température fixée mais requiert pour son bon fonctionnement

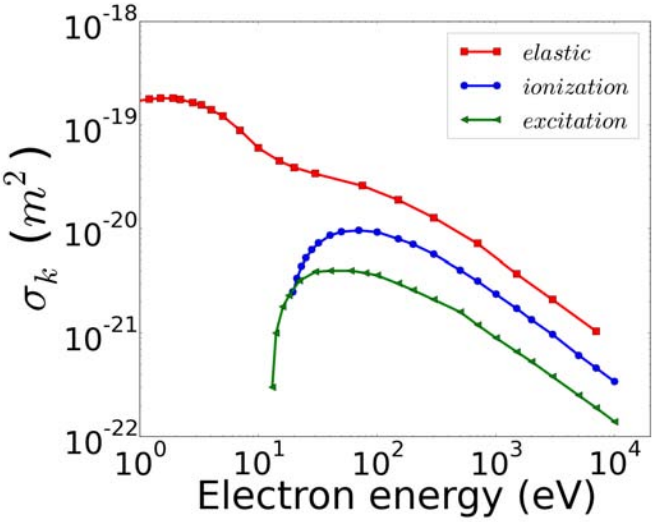


FIGURE 3.5 – Jeu de section efficaces simplifié utilisé dans les calculs pour les collisions électron-H_2. La courbe (et carrés) rouge représente la section efficace de collision élastique, en bleu (et cercles), la section efficace d'ionisation avec une énergie seuil de 15.4 eV et la courbe (et triangles) verte représente la section efficace de collisions inélastique avec une énergie seuil de 12.4 eV [58].

l'ajout d'un terme source (injection de paire de particules électron-ions). le chauffage seul, impose une température au système qui est différente de la température *auto-consistante* pour laquelle la décharge peut être maintenue dans un état stationnaire. Dans le cas où la température est trop faible, la décharge s'éteint petit à petit, et dans le cas contraire, la densité augmente indéfiniment.

La deuxième méthode, qui est celle employée dans les simulations suivantes, permet de maintenir le plasma à une température auto consistante pour laquelle la décharge atteint un état stationnaire. De façon générale, une puissance $\mathcal{P}$, fixée au préalable, va être absorbée, dans une région appelée *driver*, par les électrons résidant dans le driver. La température des électrons va s'ajuster de telle façon que les pertes de particules vers les parois soient exactement compensées par l'ionisation. La puissance est supposée être absorbée uniformément dans le driver est tend à maintenir les électrons vers une distribution en vitesse Maxwellienne.

Les électrons sont *"chauffés"* de manière aléatoire dans le driver. La région de chauffage est située dans la partie gauche de la source et dont la dimension de celle-ci équivaut à $\frac{1}{3}$ de la longueur dans la direction x contre la totalité dans la direction y (cf. Fig.3.6). La puissance absorbée est fixée à 10 W/m et la fréquence de chauffage des électrons $\nu_h = 10^8$ s^{-1}.

Ceci signifie qu'une fraction $(\nu_h \delta t)$ du nombre total d'électrons situés dans la zone de chauffage, après un intervalle de temps donné, est sélectionnée aléatoirement puis chauffée.

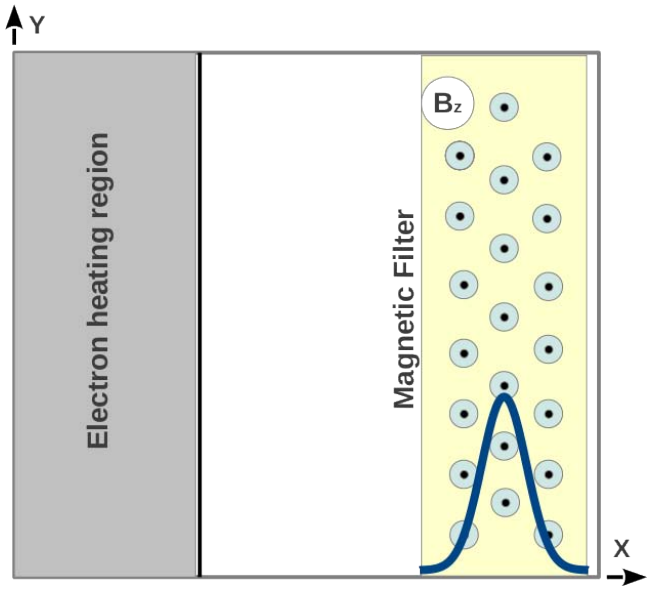


FIGURE 3.6 – Domaine de simulation (2D Cartésien). la densité de gaz (H_2) dans la chambre est de 5×10^{19} m^{-3}. les électrons sont chauffés de façon uniforme dans la région grisée (La puissance absorbée est de 10 W/m). Le champ magnétique maximum est de 3 mT.

Ainsi, l'énergie totale des électrons dans le *driver* ($\varepsilon_a = \sum_{i=1}^{N_d} \frac{1}{2} m v_i^2$) avant le chauffage est augmentée de $\mathcal{P}\delta t$ où N_d est le nombre total d'électrons dans la zone de chauffage et v_i^2 leurs vitesses respectives et où $\mathcal{P}$ correspond à la puissance totale absorbée. On peut donc résumer l'énergie totale après chauffage comme :

$$E_{tot} = \sum_{i=1}^{N_d} \frac{1}{2} m v_i^2 + \mathcal{P}\delta t \tag{3.20}$$

Une nouvelle vitesse est assignée à la fraction d'électrons sélectionnés (en moyenne, correspondant à $N_d \nu_h \delta t$) selon une distribution Maxwellienne et à une température T_h. Cette température est calculée à chaque intervalle de temps δt à partir de l'Eq. (3.20) et est définie comme suit :

$$T_h = \frac{2}{3} \frac{E_{tot}}{N_d} \tag{3.21}$$

Les résultats des simulations présentés dans ce qui suit ont été obtenus à partir des paramètres résumés dans la Table 3.2. La taille de la grille utilisée est de 128×128 cellules, avec une moyenne de 40 particules par cellules. La simulation est initialisée avec une densité uniforme $n_{e,i} = 2 \times 10^{14}$ m^{-3} représentée par 6.5×10^5 électrons et ions distribués unifor-

TABLE 3.2 – **Paramètres de la simulation**

Description	Symbole	Valeurs
Longueur du domaine	L	20 cm
Densité de neutre	n_n	5×10^{19} m^{-3}
Température du gaz	T_{H_2}	0.1 eV
Température des électrons	T_e	10 eV
Température des ions	T_i	0.026 eV
Champ Magnétique max	B_{z0}	$0 - 3 - 5 - 7 - 10$ mT
Position du filtre	x_0	16 cm
Écart type	σ_B	2 cm
Puissance injecté	$\mathcal{P}$	10 W.m^{-1}
Fréquence de chauffage	ν_h	1×10^8 s^{-1}
Bias	ϕ_b	$0 - 10 - 20 - 35 - 55$ V

mément sur tout le domaine de simulation, avec une distribution de vitesses Maxwellienne à une température de 10 eV pour les électrons et 0.026 eV pour les ions.

Les puissances et densités de plasma considérées dans ces calculs sont très inférieures à celles de la source d'ions négatifs. Le facteur d'échelle est de l'ordre de 10000. Il est pratiquement impossible de simuler les conditions réelles de la source avec un modèle particulaire explicite car les contraintes sur le pas en temps et le maillage seraient considérables. Les gaines de Debye aux parois sont donc beaucoup plus grandes dans les calculs que dans la vraie source. On peut cependant considérer que dans la mesure où le plasma reste quasi-neutre dans la plus grande partie du volume et si les gaines de Debye ne perturbent pas l'équilibre ionisation-pertes de particules chargées aux parois, les lois d'échelle sont applicables et la densité de plasma varie linéairement avec la puissance (on peut si nécessaire prendre en compte des processus non-linéaires tels les collisions Coulombiennes, en utilisant des facteurs d'échelle). Ceci a été montré dans la référence de Fubiani et al.[22], et est également discuté dans la référence Boeuf et al.[21]. Une autre manière de faire serait d'effectuer les calculs avec les "vraies" valeurs de puissance et densités de plasma, mais de multiplier, dans l'équation de Poisson, la permittivité du vide par un facteur d'échelle. Cette manière de faire est en fait parfaitement équivalente à la précédente puisqu'en pratique elle revient à diviser les densités dans l'équation de Poisson par ce facteur d'échelle.

Le but principal est de pouvoir quantifier les phénomènes intervenants avec un filtre magnétique. Pour caractériser, dans un premier temps, l'effet du champ magnétique, il est important de comprendre ce qu'il se passe sans celui-ci. Dans ce cas précis, nous analyserons dans une première partie les caractéristiques du plasma sans ce filtre magnétique avant de poursuivre lors de la deuxième partie sur les observations avec filtre magnétique.

3.3.1 Sans filtre magnétique :

La figure 3.7(b) montre la variation du profil 1D du potentiel plasma auquel on a appliqué différentes valeurs de tension (de 0V à 55V) au niveau de l'électrode droite de la source (cf. Fig. 3.7(a)) dans les conditions des Figs. 3.6 et 3.7(a) **sans** champ magnétique. On constate également que pour chaque tension appliquée, la température électronique varie très peu par rapport à la température électronique de référence (i.e pour une tension appliquée nulle, $\phi_b = 0$ V) et est de l'ordre de 9.2 eV. La différence de température un peu plus importante que l'on observe à gauche du domaine de simulation (dans l'intervalle $\sim 0 - 0.05$ m) est due au fait que les électrons sont chauffés dans cette partie. Le libre parcours moyen des électrons étant relativement grand (plusieurs centimètres) et induisant une conductivité thermique importante, la température électronique T_e reste quasi-constante entre les électrodes et ne dépend pas de la tension appliquée ϕ_b. On observe également sur la Fig. 3.7(b), que plus on augmente la tension ϕ_b, plus le potentiel plasma (ϕ_p) continue de croître. Ceci s'explique par le fait que le potentiel plasma ϕ_p est déterminé d'une part par la température des électrons et d'autre part par la tension appliquée aux parois. La température ne variant pas ou très faiblement, la variation du potentiel est déterminée par la différence de tension aux parois et donc par le rapport des flux d'électrons à ces surfaces. Alors l'introduction d'une tension positive attirant les particules chargées négativement (ici, les électrons) va accroître les pertes de ces particules à la paroi. Le potentiel va ainsi s'ajuster faisant en sorte que la quasi-neutralité soit préservée dans le système et que l'égalité du flux total d'ions et d'électrons aux parois soit assurée.

On le démontre analytiquement, en supposant la température électronique constante et en supposant que le flux total d'électrons perdus aux parois est égale au flux total d'ions aux parois, soit : $\Gamma_{tot} = \Gamma_{e_L} + \Gamma_{e_R} + \Gamma_{e_B} + \Gamma_{e_T} = \Gamma_{i_L} + \Gamma_{i_R} + \Gamma_{i_B} + \Gamma_{i_T}$, où les indices e et i représentent les électrons et les ions respectivement et Γ le flux aux parois (voir Fig. 3.7(a)). Dans le but de simplifier l'écriture et de généraliser cette expression nous noterons que le flux total intégré sur toute la surface du domaine de simulation est donné par la relation :

$$G_{tot} = A_1\Gamma_{e_1} + A_2\Gamma_{e_2} = (A_1 + A_2)\Gamma_i \tag{3.22}$$

Où A_2 correspond à la surface sur laquelle on applique une tension et Γ_{e_2} le flux électronique associé. La surface A_1 représente le reste de la surface du domaine de simulation et où Γ_{e_1} est équivalent au flux électronique à travers cette surface. Alors, $(A_1 + A_2)$ correspond à la surface totale de la source de plasma et Γ_i le flux d'ions à travers cette surface.

Pour une distribution Maxwellienne de vitesses $f(v)$ et à partir de l'Eq. (3.7), il est

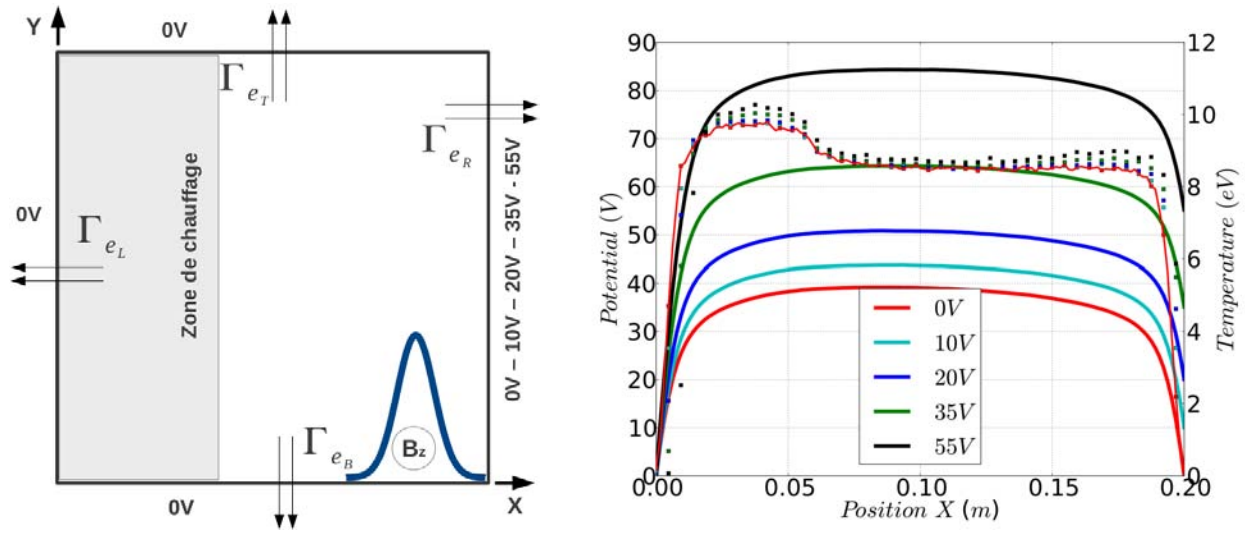


FIGURE 3.7 – (a) Schéma de la source et représentation des tensions aux électrodes et des flux $\Gamma_{e_{L,R,B,T}}$ aux parois. La tension appliquée à l'électrode e_R varie de $0-55$ V, (b) Profil 1D du potentiel plasma correspondant à cinq valeurs de tensions différentes appliquées sur l'électrode e_R. La température électronique est également représentée (ligne rouge pour une tension de 0V), les différents points de couleurs correspondent aux différentes tensions (Les profils de températures sont quasiment identiques).Le champ magnétique du filtre est nul dans ces simulations.

possible de calculer la vitesse thermique moyenne des électrons $\overline{\mathbf{v}}_e$ telle que :

$$\overline{\mathbf{v}}_e = \frac{1}{n}\int v f(v) d^3v = \left(\frac{8k_bT_e}{\pi m}\right)^{\frac{1}{2}} \tag{3.23}$$

De la même façon, le flux moyen d'électrons (ici, dans le sens positif) à travers une surface perpendiculaire à un plan (x, y) est donné par :

$$\Gamma_e = \int_{-\infty}^{+\infty} dv_x \int_{-\infty}^{+\infty} dv_y \int_0^{+\infty} v f(v) dv = \frac{1}{4} n_e \overline{\mathbf{v}}_e \tag{3.24}$$

avec n_e, la densité d'électrons. À partir de l'Eq. (3.15) en l'absence de champ magnétique et en négligeant le terme de friction $[mn_e\nu\mathbf{v}_e]$ et en posant $\mathbf{E} = -\nabla\phi$, on peut déduire la relation de Boltzmann pour les électrons :

$$n_e = n_0 \exp\left(\frac{e\phi}{k_bT_e}\right) \tag{3.25}$$

À partir des équations précédentes (Eqs. (3.24) et (3.25)), le flux d'électrons aux parois s'écrit finalement :

$$\begin{aligned} \Gamma_{e_1} &= \frac{1}{4} n_0 \overline{\mathbf{v}}_e \exp\left(-\frac{e\phi_p}{k_b T_e}\right) \\ \Gamma_{e_2} &= \frac{1}{4} n_0 \overline{\mathbf{v}}_e \exp\left(-\frac{e(\phi_p - \phi_b)}{k_b T_e}\right) \end{aligned} \tag{3.26}$$

De même, on peut montrer que le flux moyen d'ions aux parois est donné par la relation suivante :

$$\Gamma_i = n_o u_B \exp\left(-\frac{1}{2}\right) \tag{3.27}$$

Où $u_B = \sqrt{\frac{k_b T_e}{M}}$ est la vitesse de Bohm avec M la masse des ions.

En supposant l'égalité entre le flux total d'électrons et le flux total d'ions aux parois et en tenant compte des expressions du flux proposées ci-dessus (Eqs. (3.26) et (3.27)), on peut déduire facilement la valeur du potentiel plasma ϕ_p :

$$\phi_p = \frac{k_b T_e}{2e}\left(1 + \ln\left(\frac{M}{2\pi m}\right)\right) + \frac{k_b T_e}{e} \ln\left(\frac{A_1 + A_2 \exp\left[\frac{e\phi_b}{k_b T_e}\right]}{(A_1 + A_2)}\right) \tag{3.28}$$

Dans le cas de la simulation de la Fig. 3.7(a), on pose $A_1 = 3A_2$. Les valeurs théoriques du potentiel plasma (ϕ_p) en fonction de la tension appliquée à l'électrode (ϕ_b) calculées à partir de l'Eq. (3.28), sont représentées par la courbe (ligne verte) de la figure 3.8. On suppose la température électronique constante et ne dépendant pas de la tension appliquée et dans nos conditions de simulation (cf. Fig. 3.7(b)) $\frac{k_b T_e}{e} \simeq 9.2$ eV. Les valeurs numériques de ϕ_p obtenues lors des simulations effectuées dans les conditions de la Fig. 3.7(a) sans champ magnétique, sont en très bon accords avec la courbe théorique Fig. 3.8 (représentées par des cercles bleus sur la figure).

Le fait d'augmenter la tension à l'électrode permet d'attirer de plus en plus d'électrons. En l'absence d'un champ magnétique et pour une tension appliquée suffisamment importante, il est possible de collecter 100% du flux total d'électron sur cette même électrode. La Fig. 3.8 montre bien l'évolution de la fraction du flux d'électron collectée à l'électrode en fonction de la tension appliquée (ligne rouge). L'expression analytique peut être obtenue à partir de l'expression du flux d'électrons (Eq. 3.26) et est donnée par :

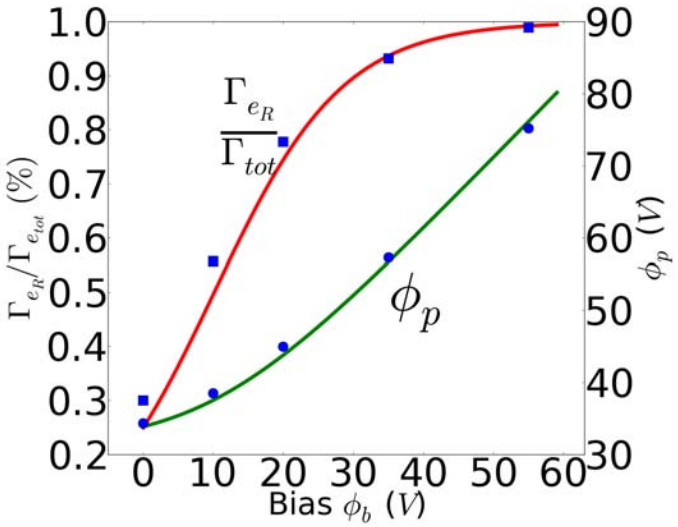


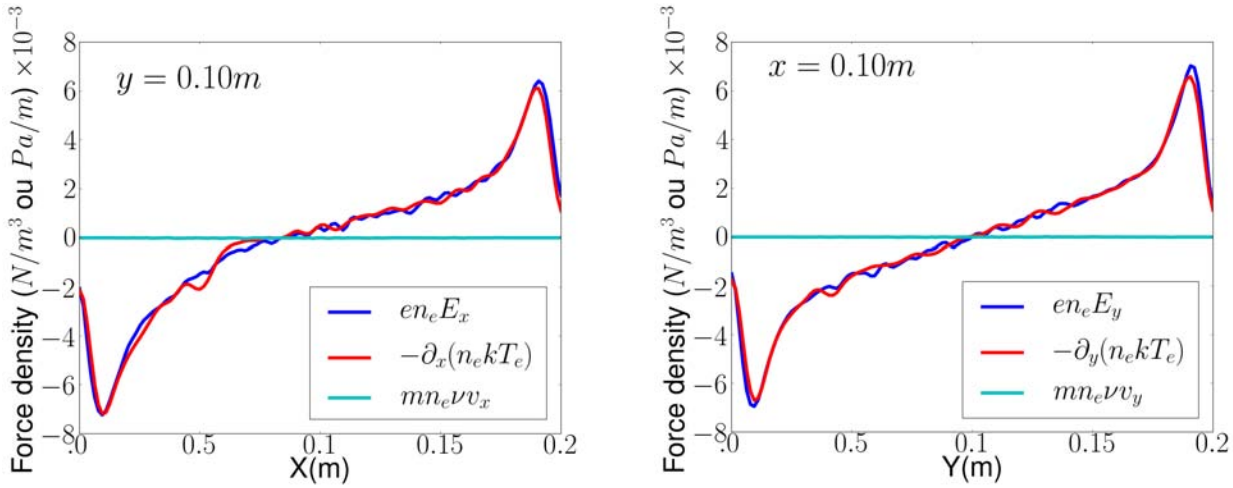
FIGURE 3.8 – Flux théorique d'électrons collectés sur l'électrode droite de la source en pourcentage en fonction de la tension appliquée (ligne rouge) et comparaison avec les résultats numériques (carrés bleus). Comparaison entre le potentiel plasma théorique (ligne verte) et numérique (cercles bleus) en fonction de la tension appliquée. Les valeurs théoriques du flux $\frac{G_{e_2}}{G_{tot}}$ et du potentiel plasma ϕ_p sont obtenues pour une température de ~ 9.2 eV.

$$\frac{G_{e_2}}{G_{tot}} = \frac{\exp\left[\frac{e\phi_b}{k_b T_e}\right]}{\frac{A_1}{A_2} + \exp\left[\frac{e\phi_b}{k_b T_e}\right]} \tag{3.29}$$

Sur la Fig. 3.8 les résultats numériques concernant la fraction de flux d'électrons collectée à l'électrode (carrés bleus) sont en bon accord avec les résultats théoriques. En présence de parois non conductrices, on montre que le flux d'électrons et le flux d'ions sont localement égaux et que la condition d'ambipolarité $\Gamma_e = \Gamma_i$ est entièrement valide dans le volume de décharge à 2D. Le flux de particules chargées en chaque point du système peut se déduire facilement de l'équation de conservation de la quantité de mouvement (Eq. (3.14)). En supposant le plasma à l'état stationnaire, sans champ magnétique et où l'on suppose le tenseur de pression diagonal et isotrope, on peut écrire cette équation (Eq. (3.14)) pour les électrons :

$$\nabla(n_e k_b T_e) = -e n_e \mathbf{E} - m n_e \nu_{en} \mathbf{v}_e \tag{3.30}$$

Où n_e, T_e sont respectivement la densité d'électrons et la température électronique, et



(a) (b)

FIGURE 3.9 – Profil 1D des termes de gradient de pression et de force électrostatique (a) dans le plan y à $x = 10$ cm, (b) dans le plan x à $y = 10$ cm sans champ magnétique et sans tension appliquée ($\phi_b = 0$).

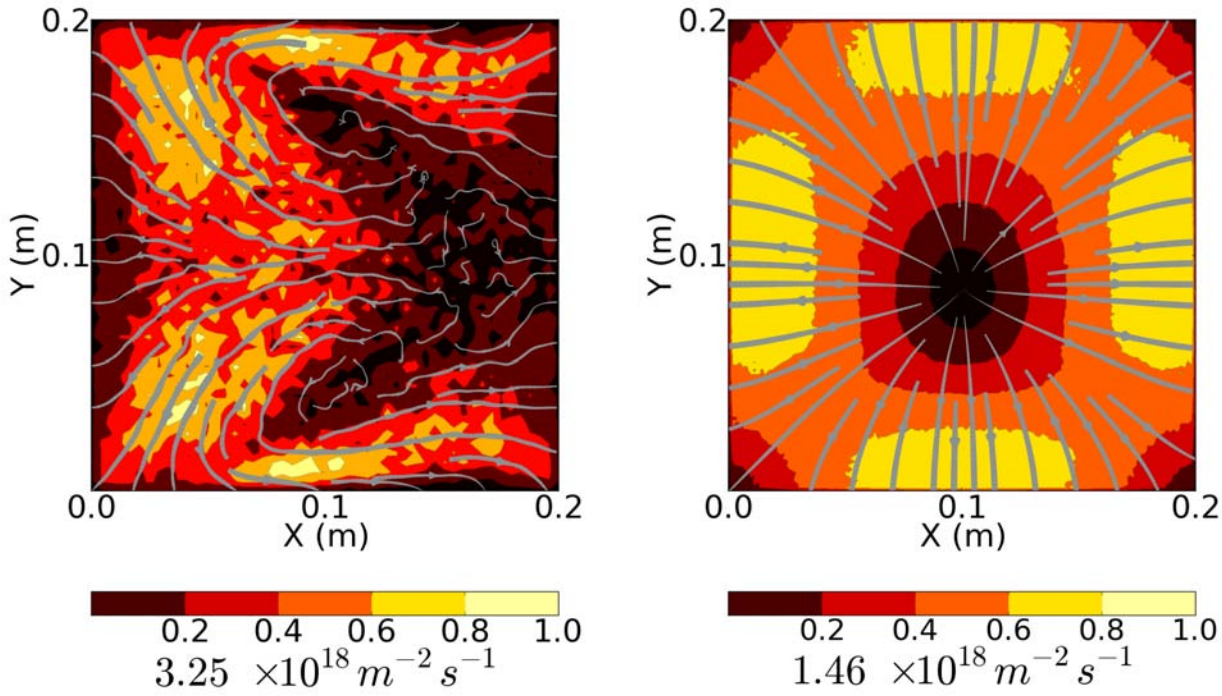
$\nu_{en}, \mathbf{v}_e$ sont la fréquence de collision électrons-neutres et la vitesse thermique moyenne des électrons. Ainsi, à partir de l'Eq. (3.30), on en déduit directement le flux électronique sans champ magnétique :

$$\Gamma_e = n_e\mathbf{v}_e = \frac{-en_e}{m\nu_{en}}\mathbf{E} - \nabla\frac{(n_ek_bT_e)}{m\nu_{en}} \tag{3.31}$$

Dans le cas d'un plasma peu collisionnel i.e. où le libre parcours moyen des électrons est grand par rapport à la taille caractéristique du système, le terme de friction $[mn_e\nu_{en}\mathbf{v}_e]$ est petit devant les termes de gradient de pression et de force électrostatique qui se compensent l'un et l'autre. La représentation dans le plan x à $y = 0.1$ m et dans le plan y à $x = 0.1$ m sur la Fig. 3.9 des profils 1D des termes de l'Eq. (3.30) permet de mettre en évidence ces hypothèses.

Toutefois, il est intéressant de constater sur la Fig. 3.10 que le flux électronique et le flux ionique présentés ne sont pas ambipolaires. Contrairement à la Fig. 3.10(b) où le flux d'ions est identique à la surface de chaque paroi, i.e. 25% du flux total d'ions est collecté sur chaque paroi de la décharge, on voit apparaître sur la Fig. 3.10(a) de façon symétrique, des vortex de courant sur le centre haut et bas de la source.

La création de ces vortex de courant peut être expliquée par le fait que dans les conditions de chauffage (cf. Figs. 3.6 et 3.7(a)), la température électronique dans cette région est légèrement supérieure au reste du domaine de simulation (cf. Fig. 3.7(b)). Les récents travaux de E.A. Bogdanov et al.[75] montrent que la présence d'une non uniformité spatiale



(a) (b)

FIGURE 3.10 – Distribution 2D moyennée du flux (a) électronique et (b) ionique (m^{-2} s^{-1}) sans champ magnétique et sans tension appliquée ($\phi_b = 0$). Les couleurs correspondent à l'intensité du flux et les lignes directionnelles à la direction du flux.

de la densité et de la température électronique peut conduire à la formation de vortex de courant d'électrons où les flux de particules chargées ne satisfont plus à la condition locale d'ambipolarité, soit : $\Gamma_e \neq \Gamma_i$ localement. L'existence de la diffusion ambipolaire et non ambipolaire est également discutée par T. Lafleur et R.W Boswell [76] dans l'étude du flux électronique et ionique à partir des simulations PIC et dans le cas de plasmas magnétisés. Ils observent que même en l'absence de champ magnétique la diffusion des particules n'est pas complètement ambipolaire.

Cependant, nous noterons que la variation de la température des électrons à gauche de la source est relativement faible et son influence sur la mobilité électronique est négligeable et demeure constante. Dans ce cas, l'équation de diffusion ambipolaire (Eq. (3.9)) reste valide en dépit de la violation de la condition locale d'ambipolarité et globalement $\Gamma_e = \Gamma_i$.

3.3.2 Avec filtre magnétique :

La diffusion des particules chargées d'un plasma dans un champ magnétique est plus complexe à cause du caractère anisotrope du mouvement des particules. Dans cette partie nous allons pouvoir caractériser les effets du filtre magnétique à partir des résultats de

simulations du modèle PIC 2D3V dans les mêmes conditions que celles des Figs. 3.6 et de la table 3.2 **avec** un champ magnétique variant de 0 a 10 mT.

Le filtre magnétique est présent dans la partie droite de la chambre de simulation et est dirigé selon la direction z ($\mathbf{B} = (0, 0, B_z)$). Ce champ est uniforme dans la direction y et suit une distribution Gaussienne dans la direction x (Eq. (3.32)), avec un maximum centré en $x_0 = 0.16$ m et une largeur à mi hauteur $\sigma_B = 0.02$ m.

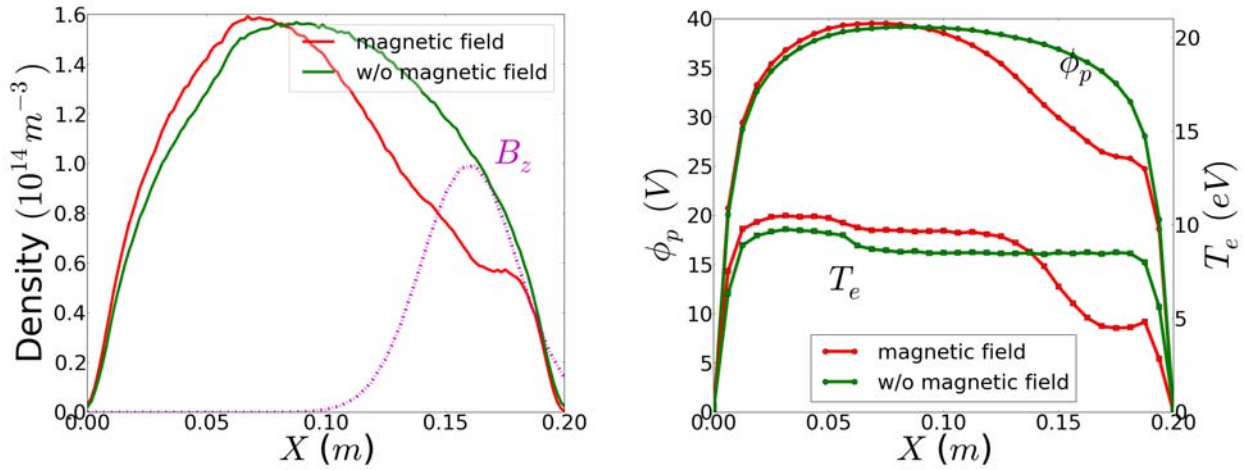
$$B_z(x) = B_{z0} \exp\left[-\frac{(x - x_0)^2}{2\sigma_B^2}\right] \tag{3.32}$$

La valeur du champ magnétique est suffisamment élevée pour magnétiser les électrons (rayon de Larmor des électrons de l'ordre de $0.3 - 0.4$ cm) mais n'affecte pas de façon significative la trajectoire des ions qui restent non magnétisés (rayon de Larmor des ions de l'ordre de 1 cm). Dans le champ magnétique le paramètre de Hall $h = \frac{\omega_c}{\nu_{en}}$ (ω_c est la fréquence cyclotronique des électrons et ν_{en} est la fréquence de collision électrons-neutres) est très grand, de l'ordre de 10^2. Dans ces conditions, les électrons dans la zone de champ magnétique sont fortement piégés et leurs temps de résidence dans la source est augmenté de façon non négligeable.

La comparaison Fig. 3.11(b) entre la température électronique sans champ magnétique (ligne verte) avec la température en présence du champ magnétique (ligne rouge) permet de mettre en évidence une forte réduction du transport d'énergie à travers le filtre dans le deuxième cas, i.e la température T_e baisse de façon significative dans la région du filtre.

Cette baisse est due au fait que les électrons ayant un temps de résidence plus long (à cause du piégeage magnétique) dans le filtre vont subir un plus grand nombre de collisions avec les neutres, induisant une perte d'énergie dans cette région. En dehors de la zone du filtre, la variation de la température électronique est définie par un équilibre opposant les pertes d'électrons aux parois et la création (ici, par des processus d'ionisation) d'électrons dans le volume de la décharge. Par sa présence, le filtre magnétique diminue le volume apparent d'ionisation. Ainsi la température électronique va s'ajuster de telle sorte que si le rapport surface sur volume augmente, la température augmente également. Cette augmentation est observée sur la figure 3.11(b). On observe également une diminution de la densité d'électrons Fig. 3.11(a) et une chute du potentiel plasma Fig. 3.11(b) à l'entrée du filtre.

En l'absence de champ magnétique, les électrons vont tendre à diffuser en dehors du plasma plus vite que les ions. Ce taux inégal de diffusion va provoquer (instantanément) une grande charge d'espace et l'apparition d'un champ électrique. Ce champ électrique va réduire le courant d'électron et augmenter le courant d'ion et maintenir ainsi une charge d'espace neutre. Ce profil axial 1D de champ électrique est représenté sur la Fig. 3.12 par



(a) (b)

FIGURE 3.11 – Distribution 1D de la densité d'électrons (a), du potentiel plasma et de la température électronique (b) pour une tension nulle à la paroi et avec un champ magnétique (courbe pointillée (a)) maximum $B_z = 3$ mT. Un facteur d'échelle d'environ 10000 doit être appliqué à la densité de plasma pour obtenir des valeurs en rapport avec la source d'ions d'ITER (densité de plasma de l'ordre de 10^{18} m^{-3}).

la courbe en rouge dans le cas de la simulation sans champ magnétique présentée dans la section 3.3.1. La diffusion résultante est une diffusion ambipolaire où le coefficient de diffusion ambipolaire pour les deux espèces chargées est le même. Cependant, en présence du champ magnétique et dans le cas où les ions ne sont pas affectés par celui-ci, la diffusion des électrons à travers ce champ va être significativement réduite au point que les ions vont cette fois-ci diffuser plus vite que les électrons à travers le filtre. Ce phénomène peut être observé sur la Fig. 3.11(b) où l'on peut voir sur la courbe du profil axial 1D du potentiel électrique dans le cas avec champ magnétique, une chute de potentiel importante à l'entrée du filtre magnétique caractérisant bien l'effet que les électrons ralentissant à mesure qu'ils approchent de la barrière magnétique voient les ions être accélérés. En supposant donc que les ions diffusent plus rapidement que les électrons à travers le filtre magnétique, un champ électrique va apparaître afin de réduire le courant d'ions à l'entrée de la barrière. Ceci peut être observé sur la Fig. 3.12 sur le profil axial 1D (courbe verte) où le champ électrique croît à l'entrée du filtre magnétique.

Il est intéressant de noter le caractère asymétrique des distributions 2D de la température des électrons, du potentiel électrique et de la densité électronique (Figs. 3.13). On peut remarquer que la distribution 2D du potentiel plasma (Fig. 3.13(a)) est étroitement liée à la distribution de la pression (Fig. 3.13(d)). En dehors et jusqu'à l'entrée du filtre magnétique les forces électrostatique et de pression se compensent et la relation de Boltzmann (Eq.(3.25)) reliant la variation de la densité en fonction du potentiel reste valable. À

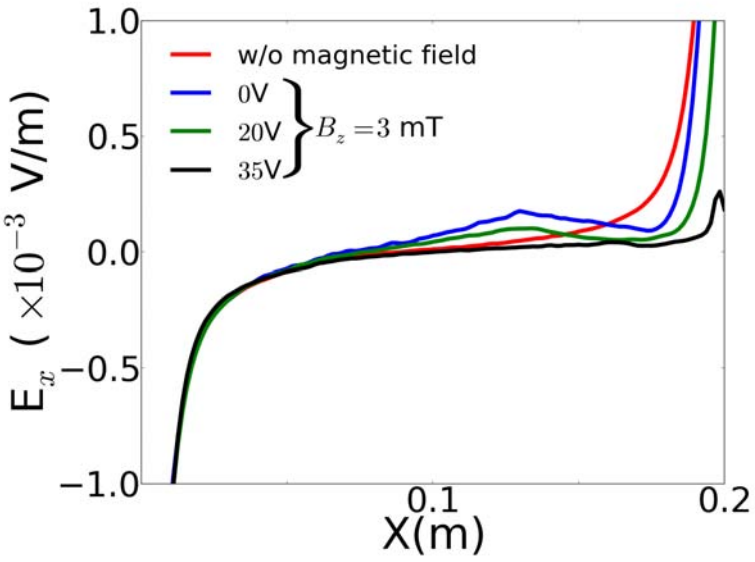


FIGURE 3.12 – Profil du champ électrique axial sans filtre magnétique et sans polarisation (courbe rouge) et avec le filtre magnétique (courbe verte) avec $B_z = 3$ mT pour 3 valeurs de la tension appliquée à l'électrode e_R.

l'entrée du filtre, les forces vont se rééquilibrer, i.e. les forces prédominantes électrostatique et de pression vont diminuer à mesure que l'interaction magnétique devient de plus en plus importante. Cette décroissance peut être également observée à travers les résultats de la simulation Figs. 3.11(b) et 3.13(a,d) et sera vérifiée dans ce qui suit par l'observation de l'évolution spatiale des termes de forces. Cependant à l'intérieur du filtre la force associée au champ magnétique joue un rôle majeur et la relation de Boltzmann n'est plus valable due au fait que la force électrostatique n'est plus entièrement balancée par le gradient de pression. Ceci s'observe *via* le plateau du potentiel de l'ordre de 26 V sur la Fig. 3.11(b)). L'asymétrie de la distribution 2D de la température électronique est liée aux effets du champ magnétique dirigé perpendiculairement au plan (x, y). Le transport des électrons à travers le filtre, nous allons le voir, est caractérisé par les dérives $\mathbf{E} \times \mathbf{B}/B^2$ dans le sens positif de l'axe x dans le haut de la zone du champ et dans le sens négatif dans le bas et est caractérisé également par la dérive diamagnétique $\nabla P \times \mathbf{B}/B^2$ le long de l'axe y dans le sens positif et permet d'expliquer l'asymétrie apparente de la distribution.

La figure 3.14(a) montre le flux des électrons dans les conditions de simulations de la Fig. 3.6 avec un champ magnétique $B_{z0} = 3$ mT. Dans le but d'une compréhension des phénomènes collectifs intervenant dans le transport des particules chargées à travers le champ magnétique (concernant notamment le transport des électrons, les ions n'étant pas magnétisés), on peut, à partir des résultats du modèle PIC, déterminer l'importance relative des termes de forces en fonction de leurs évolutions spatiales.

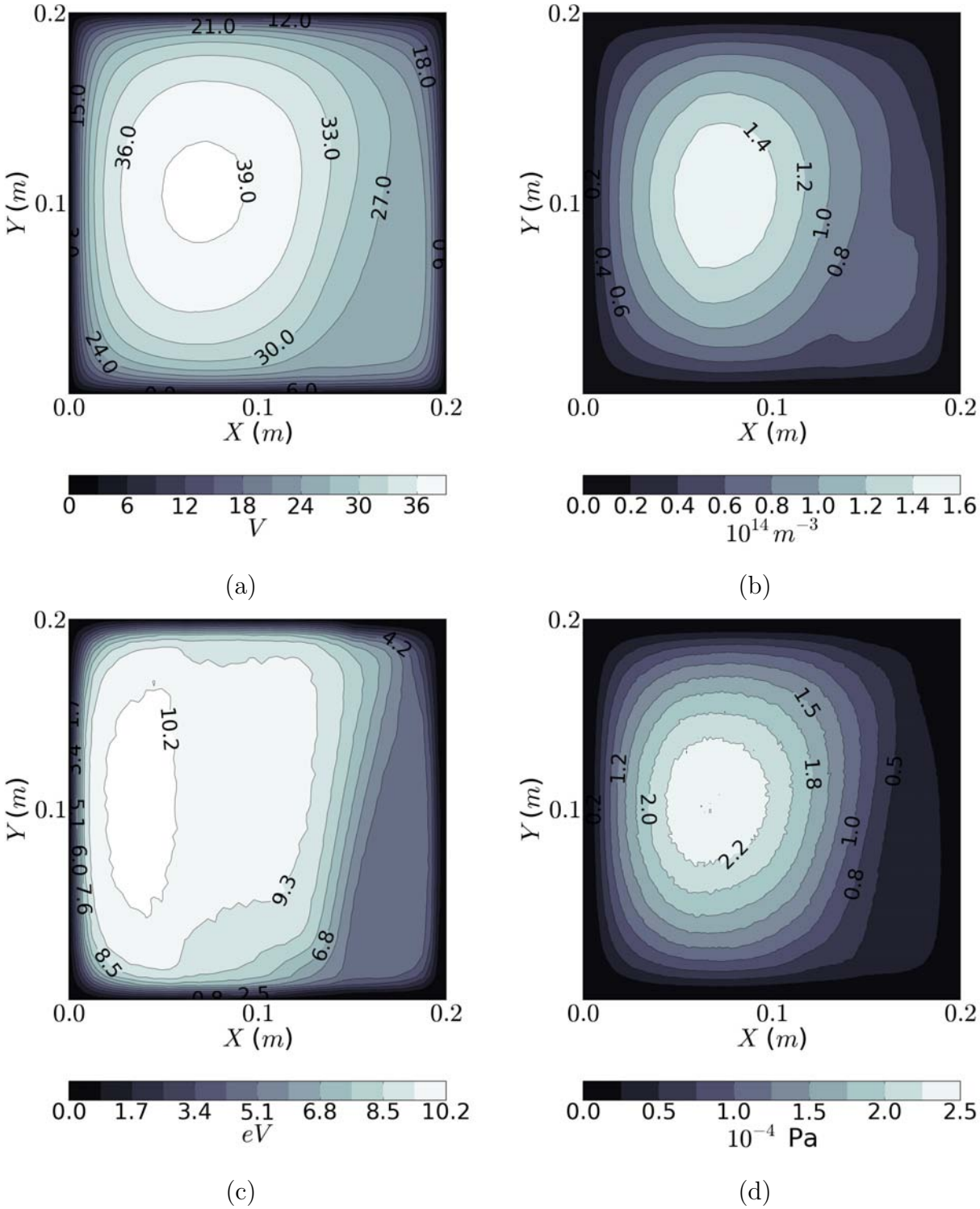


FIGURE 3.13 – (a)Distribution 2D du potentiel plasma (V), (b) de la densité d'électrons (m^{-3}), (c) de la température électronique (eV), (d) de la pression des électrons (10^{-4} Pa). Un facteur d'échelle d'environ 10000 doit être appliqué à la densité de plasma et à la pression électronique pour obtenir des valeurs en rapport avec celles de la source d'ions d'ITER (densité de plasma de l'ordre de 10^{18} m^{-3}).

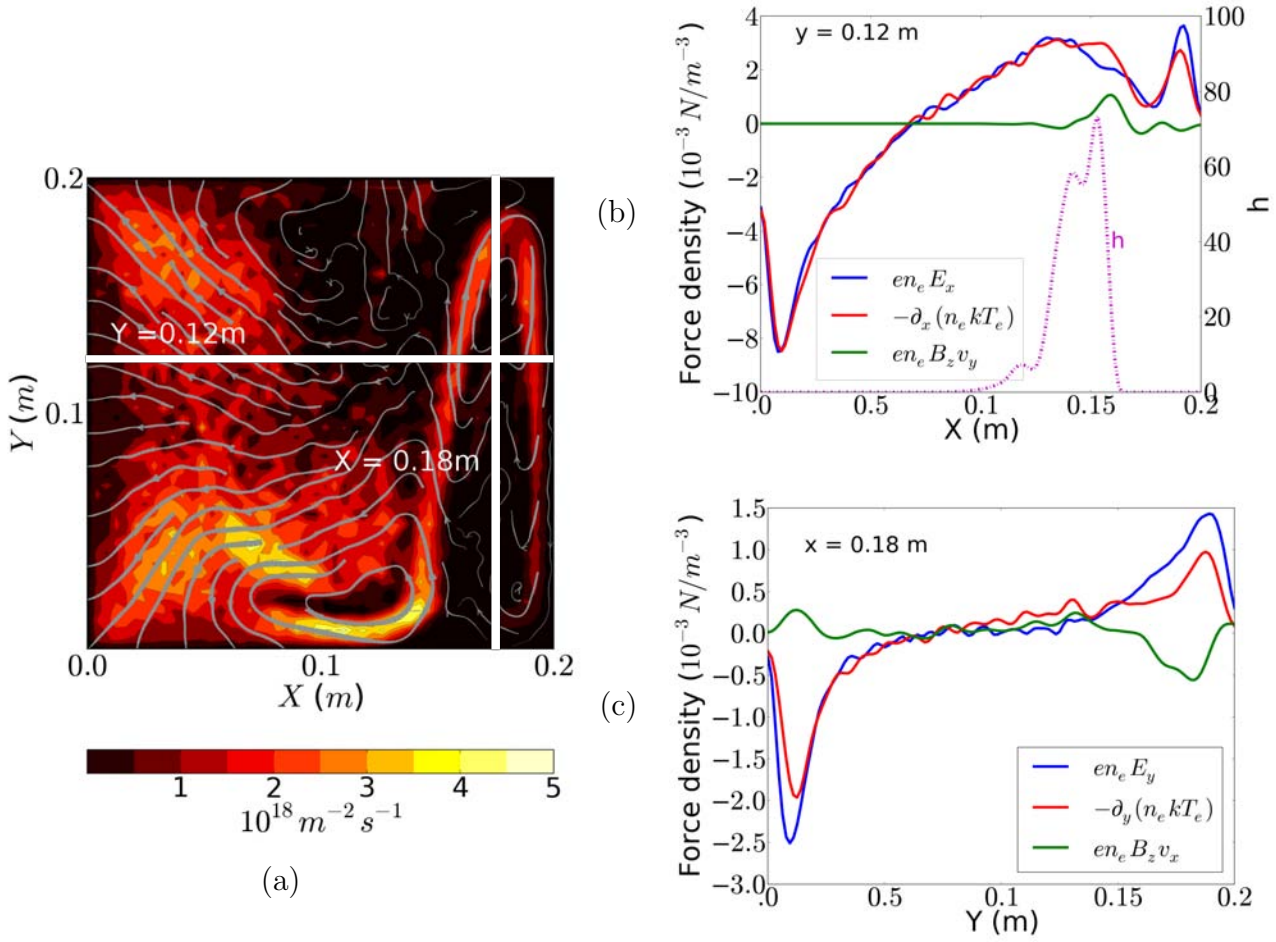


FIGURE 3.14 – (a) Distribution 2D du flux électronique (m^{-2} s^{-1}) : la couleur correspond à l'intensité du flux et les lignes directionnelles, à la direction du flux. Profil 1D des termes de forces de l'Eq. (3.33), dans le plan x à $y = 12$ cm (b) et dans le plan y à $x = 18$ cm (c). Le paramètre de Hall ($h = \frac{\omega_c}{\nu}$) est représenté sur la fig. (b) (facteur d'échelle d'environ 10000 sur les flux et forces pour obtenir des valeurs correspondant à celles de la source d'ions d'ITER).

En considérant, à deux dimensions, un champ électrique $\mathbf{E} = (E_x, E_y, 0)$ et un champ magnétique dirigé selon l'axe z ($\mathbf{B} = (0, 0, B_z)$), pour un état stationnaire où le terme d'inertie est négligeable, en supposant le tenseur de pression diagonal et isotrope, on peut écrire les composantes x et y de l'équation de conservation de la quantité de mouvement des électrons (Eq. (3.14)) telles que :

$$\begin{aligned} x : \quad & \nabla_x(n_e k_b T_e) = -en_e E_x - en_e v_y B_z - mn_e \nu_{en} v_x \\ y : \quad & \nabla_y(n_e k_b T_e) = -en_e E_y + en_e v_x B_z - mn_e \nu_{en} v_y \end{aligned} \tag{3.33}$$

De ces équations (Eqs. (3.33)), nous avons vu que l'on pouvait aisément en extraire la forme analytique du flux en tout point de la source (cf. Eqs. (3.17)) où on rappelle que la forme vectorielle du flux (cf. Eqs. (3.18)) peut être écrite comme :

$$\Gamma_e = n_e \mathbf{v}_e = \frac{1}{(1+h^2)}(\Gamma_e^* - \Gamma_e^* \times \mathbf{h}) \tag{3.34}$$

Où $h = \frac{\omega_c}{\nu}$ est le paramètre de Hall, avec ω_c et ν la fréquence cyclotronique et la fréquence de collision (ici, pour les collisions électrons-neutres) respectivement. On distingue donc deux termes dans l'Eq. (3.34) du flux à savoir : un premier terme dans la direction de la composante qui est lié au transport collisionnel sans champ magnétique ($\Gamma_e^*/(1+h^2)$) et un deuxième terme dans la direction perpendiculaire à la composante du flux lié à la dérive $\mathbf{E} \times \mathbf{B}$ et à la dérive diamagnétique $\nabla(n_e k_b T_e) \times \mathbf{B}$. Dans le cas d'un plasma peu collisionnel et en supposant que $\nu \ll \omega_c$, on fait apparaître ces termes de dérives (cf. Eq. (3.19)). On rappelle ici cette équation :

$$\Gamma_e = n_e \frac{\mathbf{E} \times \mathbf{B}}{B^2} + \frac{\nabla(n_e k_b T_e) \times \mathbf{B}}{B^2} \tag{3.35}$$

Les termes de gradient de pression, de force électrostatique et magnétique ainsi que le paramètre de Hall sont représentés Figs. 3.14(b,c) et correspondent aux coupes sur les plans x, y marqués par une ligne blanche sur la Fig. 3.14(a), avec un champ magnétique $B_{z0} = 3$ mT. Le terme de friction $[m n_e \nu_{en} \mathbf{v}]$ n'est pas représenté car il est négligeable par rapport aux autres termes de l'Eq. (3.33).

Dans la partie gauche de la décharge sur la Fig. 3.14(b), en dehors de l'influence du champ magnétique, le terme de gradient de pression compense entièrement le terme de force électrostatique et le comportement collectif des électrons est le même que celui observé en l'absence du champ magnétique (cf. Fig 3.10(a)). Dans ce cas, le transport est caractérisé par le terme ($\Gamma_e^*/(1+h^2)$) de l'Eq. (3.34) du flux.

A l'intérieur du filtre magnétique, la fréquence de collision électron-neutre devient négligeable devant la fréquence cyclotronique et le flux des électrons dans cette région lié au terme ($(\Gamma_e^* \times \mathbf{h})/(1+h^2)$) de l'Eq. (3.34) nous conduit à l'équation 3.35. Autrement dit, le flux électronique dans la zone de champ magnétique est décrit par les dérives $\mathbf{E} \times \mathbf{B}$ et par la dérive diamagnétique liée au fort gradient de pression à l'entrée du filtre.

Sur la Fig. 3.14(b), le terme de force électrostatique n'équilibre plus à lui seul le terme de gradient de pression ($\nabla_x(n_e k_b T_e)$) qui est maintenant compensé par le terme de champ magnétique ($e n_e v_y B_z$). Bien que la composante ($e n_e E_x$) ne soit pas négligeable et justifie le flux traversant axialement la barrière magnétique dans le sens négatif Fig. 3.14(a), la

composante $\left[-\frac{\nabla_x(n_e k_b T_e)}{B}\right]$ du flux Γ_y est dominante et suffit à conduire un courant diamagnétique dans la direction y. De plus nous verrons que lors de l'augmentation du flux à travers la barrière magnétique par le biais d'une tension positive à l'électrode droite de la décharge, le terme de force électrostatique devient négligeable et seul le courant diamagnétique joue un rôle majeur dans le flux des électrons à travers la barrière magnétique.

À l'inverse, sur la Fig. 3.14(c) dans le plan y à $x = 18$ cm au niveau des parois supérieures et inférieures, le terme de force électrostatique (en_eE_y) domine le terme de gradient de pression ($\nabla_y(n_e k_b T_e)$). Au niveau de la paroi supérieure le terme de force électrostatique devient significativement plus large et est maintenant compensé par le terme de champ magnétique ($en_e v_x B_z$). Le terme $\left[n_e \frac{E_y}{B}\right]$ du flux Γ_x dans l'Eq. (3.35) prédomine par rapport au terme $\left[-\frac{\nabla_y(n_e k_b T_e)}{B}\right]$ et le transport est caractérisé principalement par la dérive $\mathbf{E} \times \mathbf{B}$ des électrons dans la direction x.

Une première conclusion sur le transport des électrons à travers la barrière magnétique peut être apportée à partir des résultats discutés ci-dessus. La variation spatiale des profils 1D des termes de force permet de mettre en évidence les caractères dominants de ces forces intervenant dans le filtre magnétique. La présence de parois aux extrémités hautes et basses de la décharge participe largement aux phénomènes de transport à travers le filtre. L'existence du champ électrostatique E_y (dirigé vers la paroi supérieure et inférieure de la décharge) est responsable de la dérive $\mathbf{E} \times \mathbf{B}$ qui permet le transport axial des électrons dans la partie supérieure de la source au niveau du filtre magnétique. De plus, l'asymétrie de la distribution spatiale observée Figs. 3.13 est associée à la dérive $\mathbf{E} \times \mathbf{B}$ et peut être définie comme étant une des conséquences de cette dérive. Finalement, le filtre magnétique fait apparaître de forts gradients de densité et de température (cf. Figs. 3.13(b,d)) qui sont la conséquence d'une dérive diamagnétique dominante ($-\frac{\nabla_x(n_e k_b T_e)}{B}$) dirigée le long du plan y dans le sens positif, du bas vers le haut de la décharge (cf. Fig. 3.14(a)).

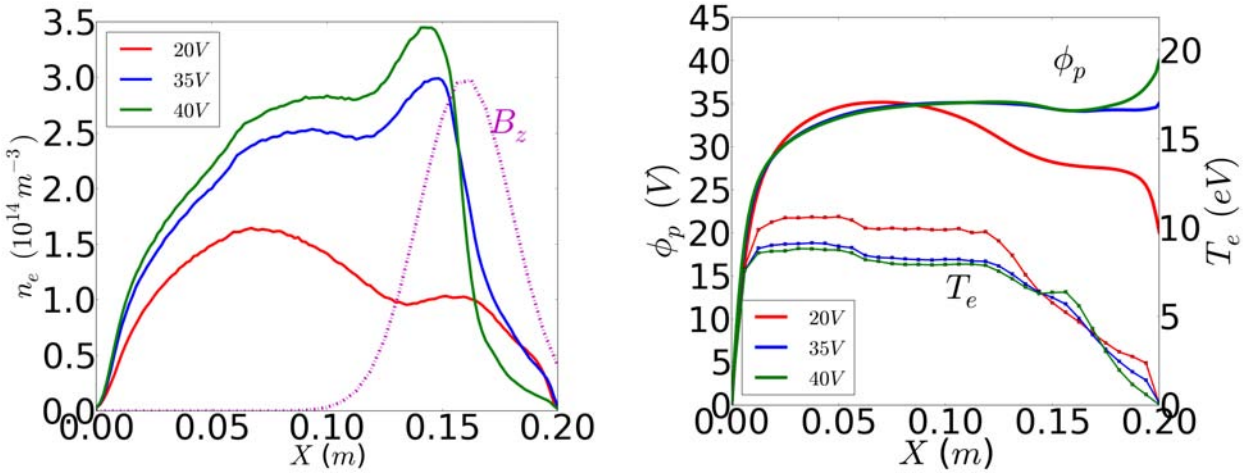
Après avoir étudié le comportement du plasma en présence du champ magnétique dirigé perpendiculairement au plan, il est intéressant de voir maintenant quelle est l'influence de la variation en intensité du champ magnétique et quelles sont les conséquences duent à l'asymétrie du plasma, sur les distributions de densités de courant d'électrons et sur le flux collecté aux parois pour une tension appliquée variant de 0 V à 55 V à l'électrode droite de la source.

3.3.3 Étude paramétrique

En premier lieu, en se référant aux études présentées dans la section 3.3.1 Fig. 3.7(b), il est intéressant de voir cette fois-ci, l'évolution du potentiel électrique et de la densité

de plasma en présence du champ magnétique, pour des tensions appliquées variant de 0 V à 55 V. Sur la Fig. 3.15(a), la présence d'un champ magnétique modifie fortement la densité électronique. Dans ces conditions, les électrons vont rester longtemps piégés le long des lignes de champ. Ceci a pour effet une accumulation des électrons et un maximum de densité dans le filtre magnétique. Ces effets sont d'autant plus importants lorsque l'on augmente la tension à l'électrode où son pouvoir d'attraction grandissant est responsable de l'accumulation dans le filtre magnétique (cf. Fig. 3.15(a)). De plus, l'asymétrie observée Fig. 3.13(b) peut être expliquée par le fait que la dérive $\mathbf{E} \times \mathbf{B}$ est dirigée vers la droite de la source au niveau de la paroi supérieure et vers l'intérieur de la source au niveau de la paroi inférieure. Dans ce cas, le temps de résidence des électrons tend à être plus petit dans la partie supérieure du filtre et plus long dans la partie inférieure. Ceci est également visible sur la Fig. 3.14 où le flux d'électrons est dirigé vers l'intérieur de la chambre dans la partie basse du filtre magnétique.

Sur la Fig. 3.15(b), la distribution spatiale 1D du potentiel plasma est représentée pour différentes valeurs de tension. On peut voir pour des faibles valeurs de tensions appliquées (0−20V), une faible chute du potentiel juste à l'entrée de la barrière magnétique. On observe également sur cette figure que la chute de potentiel située entre la zone de chauffage et le filtre magnétique tend à disparaître progressivement avec l'augmentation de la tension.



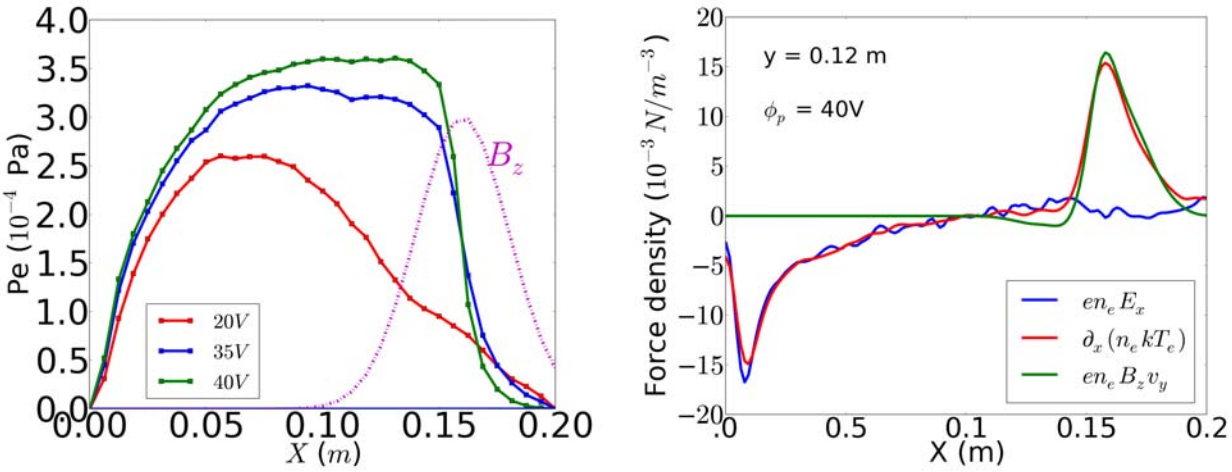
(a) (b)

FIGURE 3.15 – Distribution axiale 1D de la densité d'électrons (a), du potentiel plasma et de la température électronique (b) pour différentes tensions $(20V - 35V - 40)$ et avec un champ magnétique maximum $B_{z0} = 5\text{mT}$. Les densités sont à multiplier par un facteur d'échelle de l'ordre de 10000 pour avoir des valeurs en rapport avec celles de la source d'ions d'ITER.

Ces résultats sont corroborés par les profils de champ électrique sur la Fig. 3.12 et où la variation de ce champ à l'entrée du filtre magnétique tend à diminuer en fonction de

l'augmentation de la tension de polarisation.

La Fig. 3.16 montre l'évolution de la pression électronique en fonction de la tension appliquée (pour des valeurs comprises entre 0 et 40V). On constate que la décroissance de la pression à l'entrée du filtre tend également à disparaître avec l'augmentation de la tension appliquée pour atteindre une certaine homogénéité entre la zone située en dehors du filtre et le filtre lui-même. De plus, on peut remarquer que ce plateau est atteint lorsque la densité électronique (Fig. 3.15(a)) à l'entrée du filtre devient supérieure où égale à la densité en dehors de la zone de champ magnétique. Nous noterons que la variation de la pression induit une diminution du gradient de pression jusqu'à être quasi nul à l'entrée du filtre magnétique. Ceci peut être constaté sur les profils de variations spatiales 1D des termes de forces sur la Fig. 3.16(b) à l'entrée du filtre magnétique, dans le cas d'une tension appliquée de 40 V et en comparaison du cas présenté sur la Fig. 3.14(b) sans tension appliquée.



(a) (b)

FIGURE 3.16 – Distribution axiale 1D de la pression électronique (Pa) pour différentes valeurs de tensions ($0-40$V).

Finalement, l'évolution du potentiel plasma est étroitement liée à celle de la pression électronique Fig. 3.16(a) et le gradient de pression sur la Fig. 3.16(b) n'est plus compensé par la force électrostatique mais par le terme de champ magnétique, conduisant ainsi à un courant purement diamagnétique.

Si l'on compare l'évolution du potentiel plasma Fig. 3.7 dans le cadre d'une étude sans filtre magnétique avec celui représenté Fig. 3.15(b), on peut voir dans le cas présent, pour un champ magnétique suffisamment fort (ici, $B_{z0} = 5$mT), que le potentiel plasma (ϕ_p) en dehors de la zone du filtre ne varie plus en fonction de la tension appliquée (ϕ_b) mais reste ici quasi constant. Autrement dit, la valeur du potentiel pour une tension appliquée nulle à l'électrode est approximativement la même dans le cas où la tension devient non-nulle

et est alors déterminée par la température des électrons en dehors de la zone du filtre magnétique. Dans le cas où la tension appliquée à l'électrode droite de la source devient supérieure au potentiel plasma (voir Fig. 3.15(b)), on constate une inversion du champ électrique (cf. Fig. 3.12) à cette électrode qui a pour effet d'accélérer les espèces négatives (ici, les électrons) vers la paroi et de repousser les ions vers l'intérieur de la source.

En présence du champ magnétique, le nombre d'électrons traversant le filtre et atteignant l'électrode droite de la source décroît considérablement. De plus, à cause de la très faible gaine près de l'électrode, les électrons sont rapidement perdus à la paroi et la densité électronique dans cette région devient négligeable (cf. Fig. 3.15(a)).

Nous avons vu précédemment, dans la section 3.3.1, que le flux collecté à la paroi peut être obtenu analytiquement à partir des équations de conservation du courant et des expressions du flux aux parois pour les particules chargées (cf. Eqs. (3.26) et (3.29)). Un très bon accord entre les valeurs analytiques et numériques a pu également être observé (cf. Fig. 3.8). Cependant, en présence d'un champ magnétique, l'expression analytique du flux collecté aux parois n'est plus valide. Il devient alors intéressant de voir l'évolution de ce flux et de constater que pour un champ magnétique suffisamment important, la fraction du flux collecté à l'électrode n'augmente plus avec la tension appliquée (ϕ_b), mais atteint un seuil de saturation. Ceci met clairement en évidence le fait que le potentiel plasma (ϕ_p) n'augmente plus indéfiniment avec la tension (ϕ_b) (effet observé Fig. 3.15(b)) contrairement au cas sans champ magnétique (cf. Fig. 3.7(b)).

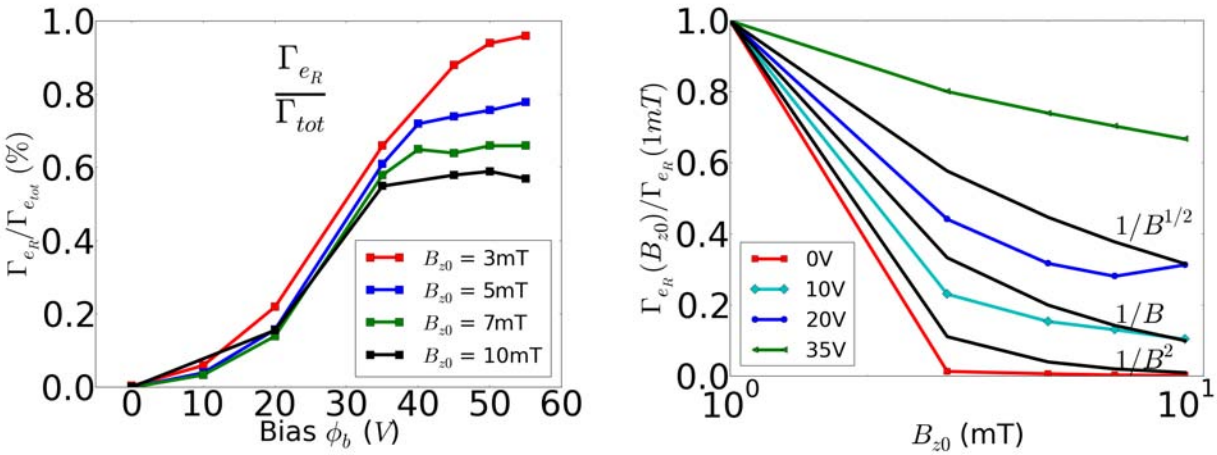
La fraction de flux d'électron $\frac{G_{e2}}{G_{tot}}$ (avec G_{tot} défini dans l'Eq. 3.23) en présence du champ magnétique est obtenue à partir des simulations numériques. Les variations du flux en fonction de la tension appliquée ϕ_b sont présentées Fig. 3.17(a) pour un champ magnétique maximum variant de 0 à 10 mT.

Il reste néanmoins très difficile de quantifier l'évolution du courant électronique en fonction de l'intensité de champ magnétique. Cependant, on constate la dépendance en $1/B$ du flux en fonction de la tension appliquée. En effet, on observe sur la Fig. 3.17(b) que le courant collecté par l'électrode, pour des tensions relativement faibles, variant de 20 à 30V, décroît en $1/B$ alors qu'elle va décroître encore plus vite pour des tensions plus faibles et décroître plus lentement pour des tensions plus forte (40V).

Cette dépendance en $1/B$ peut être décrite de manière simple. Deux considérations doivent être faites :

- On considère que la majorité des électrons sont magnétisés.
- La contribution majeure au courant électronique total collecté est supposée être due au courant de dérive $\mathbf{E} \times \mathbf{B}$ au niveau de la paroi supérieure.

En d'autre terme, le flux traversant la barrière magnétique est un flux axial et est dû de façon générale à la dérive $\mathbf{E} \times \mathbf{B}$ dans la partie supérieure de la source dans la région



(a) (b)

FIGURE 3.17 – Pourcentage du flux d'électrons collecté à la paroi de droite, (a) en fonction de la tension ϕ_b appliquée, (b) en fonctions des valeurs du champ magnétique maximum B_{z0} et pour différentes valeurs de ϕ_b. Les résultats sont donnés pour différentes valeurs du champ magnétique ($0 - 10$ mT).

du champ magnétique. A partir de l'Eq. (3.35), on peut donc écrire que le flux d'électrons collecté au niveau de la paroi à droite de la source correspond à :

$$\Gamma_{e_x} \equiv \Gamma_{e_2} \simeq n_e \frac{E_y}{B}$$

Soit :

$$G_{e_2} = A_2 \Gamma_{e_x} \tag{3.36}$$

Ceci montre bien la dépendance en $1/B$ du flux d'électrons collecté à la paroi. On peut de même montrer la linéarité entre l'évolution de la densité électronique n_e au niveau du filtre magnétique et la densité de courant collectée à l'électrode.

Enfin, il est montré (Réf. [21]) dans le cas d'études sur le transport à travers un filtre magnétique à 1D que le courant collecté varie comme $1/B^2$. Ceci vient du fait qu'à une dimension, il n'apparaît pas de champ E_y et qu'aucune dérive $\mathbf{E} \times \mathbf{B}$ n'est présente. Les électrons ne traversent la barrière magnétique qu'à partir de processus collisionnels. Il est montré que les collisions Coulombiennes jouent un rôle majeur dans ce transport électronique. Cette dépendance en $1/B^2$ peut se retrouver simplement à partir de l'équation de flux (Eq. (3.34)) pour Γ_{ex}, soit :

$$\Gamma_{e_x} = \frac{\Gamma_e^*}{h^2} \tag{3.37}$$

Où dans le filtre magnétique, le terme $[1/h^2]$ varie comme $1/B^2$.

La question qui se pose maintenant est de savoir si dans un cas en 3D d'autres phénomènes interviennent et jouent un rôle prépondérant dans le transport à travers la barrière magnétique. Si l'existence du champ électrostatique E_z peut faire apparaître d'autres dérives et comme lors du passage du 1D au 2D avec l'ajout de parois dans la direction du plan y où le transport n'est plus caractérisé par les collisions mais par les dérives magnétiques. L'ajout de parois perpendiculaires à l'axe z fait elle intervenir d'autres phénomènes ?

C'est précisément ce que nous allons étudier dans la partie suivante, grâce au modèle 3D développé au cours de cette thèse (cf. Section 3.4).

3.4 Physique du transport en trois dimensions

Dans cette partie, nous considérons un problème simplifié de transport à partir de simulations réalisées dans une géométrie 3D rectangulaire. Nous verrons dans un premier temps, comme dans le cas des simulations en 2D, un cas sans filtre magnétique, qui permettra de comparer les valeurs théoriques et analytiques du potentiel plasma et du flux collecté à la paroi sur laquelle une tension est appliquée. Dans un second temps, les résultats obtenus à partir des simulations 3D en présence de la barrière magnétique seront présentés et commentés. Enfin, l'analyse de ces résultats sera comparée avec ceux obtenus dans le cas de simulations 2D et permettra de mettre en évidence les effets éventuels dus aux parois dans la troisième dimension.

3.4.1 Modèle 3D de la source

Le modèle présenté ici est un modèle à trois dimensions dans l'espace et dans l'espace des vitesses. Les caractéristiques de la source de plasma sont similaires à celles présentées dans le cas 2D de la Section 3.3 et les mêmes considérations qui ont été faites dans la section précédente sont maintenues dans ce cas présent.

Dans le cas où l'on veut imposer la même température électronique au système que celle définie dans le cas 2D, il est important de conserver le même rapport de surface sur volume, soit S_p/V_c où S_p correspond à l'aire effective de perte des particules et où V_c correspond au volume total de création (e.g. volume d'ionisation) des particules (Réf. [71]).

Dans un cas à deux dimensions, la surface totale est représentée par le périmètre du domaine de simulation et le volume total de création par la surface de la chambre. Soit pour

TABLE 3.3 – **Paramètres de la simulation 3D**

Description	Symbole	Valeurs
Longueur du domaine	L	30 cm
Densité de neutre	n_n	5×10^{19} m^{-3}
Température du gaz	T_{H_2}	0.1 eV
Température des électrons	T_e	7.5 eV
Température des ions	T_i	0.026 eV
Champ Magnétique max	B_{z0}	$0 - 3 - 5 - 7$ mT
Position du filtre	x_0	24 cm
Écart type	σ_B	3 cm
Puissance injecté	$\mathcal{P}$	6.75 W
Fréquence de chauffage	ν_h	1×10^8 s^{-1}
Bias	ϕ_b	$0 - 10 - 20 - 30 - 40 - 50$ V

un domaine carré de longueur L ce rapport est donné par $\frac{S_p}{V_c} = \frac{4L}{L^2}$. Or à trois dimensions, ce même rapport est donné par $\frac{S_p}{V_c} = \frac{6L^2}{L^3}$. Ceci signifie qu'en trois dimensions la longueur utilisée doit être équivalente à $3L/2$ dans chaque direction si l'on veut conserver ce même ratio. De plus, la puissance injectée, de 10 W/m dans un cas 2D avec un domaine carré de dimensions 0.2×0.2 m est équivalente à 6.75 W pour un domaine 3D carré et de dimensions $0.3 \times 0.3 \times 0.3$ m.

Les résultats des simulations présentées dans ce qui suit ont été obtenus à partir des paramètres résumés dans la Table 3.3. La taille de la grille utilisée est de $64 \times 64 \times 64$ cellules, avec en moyenne 25 particules par cellule, soit $\sim 6.5 \times 10^6$ macroparticules pour chaque espèce chargée présente. La simulation est initialisée avec une densité uniforme $n_{e,i} = 8 \times 10^{13}$ m^{-3} et une température moyenne de 7.5 eV pour les électrons et de 0.026 eV pour les ions.

3.4.2 Sans filtre magnétique

Dans un premier temps, les résultats présentés ont été obtenus dans le cadre de simulations en l'absence de champ magnétique. La configuration du système est montrée sur la Fig. 3.18 et représente un domaine 3D cartésien. La zone rectangulaire bleue représente la zone de chauffage des électrons et correspond à 30% de la longueur dans la direction axiale (équivalent à 9 cm pour une longueur $L = 30$ cm) et à la totalité dans les deux autres directions. La puissance injectée dans cette région est de l'ordre de 6.75 W et est équivalente à une puissance injectée de 10 W/m pour un domaine 2D de longueur 0.2×0.2 m (cf. Section 3.3). Sur la paroi à droite de la source, une tension variant de 0 à 50 V

est appliquée et est indiquée par la face grisée sur la Fig. 3.18 et correspond à la zone d'extraction des électrons.

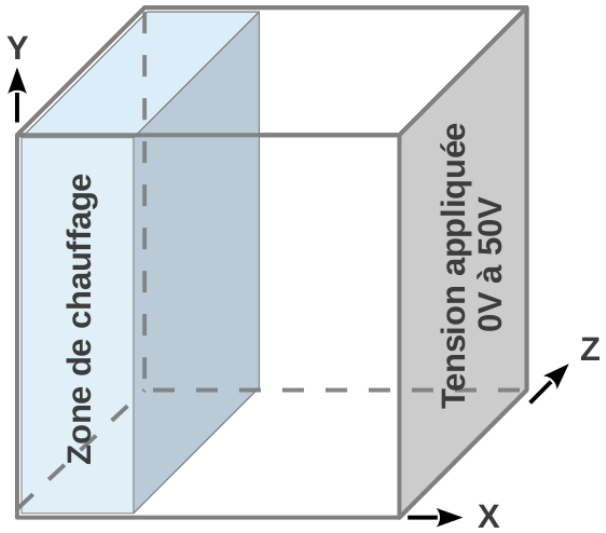


FIGURE 3.18 – Domaine de simulation (3D Cartésien). La densité de gaz (H_2) dans la chambre est de 5×10^{19} m^{-3}. les électrons sont chauffés de façon uniforme dans la région en bleue (La puissance absorbée est de 6.75 W) et correspond à 30% de la longueur dans la direction x contre la totalité dans les directions y et z.

Comme nous avons pu le voir précédemment dans la Section 3.3.1, sans champ magnétique, le potentiel plasma (ϕ_p) peut être déterminé par la température électronique ainsi que par la tension appliquée aux parois. Nous avons également pu constater que pour différentes tensions appliquées et avec une température variant très peu entre les différentes simulations, un très bon accord a été obtenu entre les valeurs analytiques et les valeurs numériques (cf. Fig. 3.8). Cette même approche a été réalisée et plusieurs simulations avec différentes tensions de polarisation ont été effectuées afin de déterminer le flux d'électrons collecté à la paroi à droite de la source et le potentiel plasma, permettant ainsi une comparaison entre les valeurs analytiques et numériques. Nous rappelons les équations générales utilisées pour le calcul du potentiel plasma ϕ_p (Eq. (3.38)) et le calcul de la fraction de flux total collecté à la paroi (G_{e_2}/G_{tot}) pour laquelle une tension de polarisation ϕ_b est appliquée (Eq. (3.39)) :

$$\phi_p = \frac{k_b T_e}{2e}\left(1 + \ln\left(\frac{M}{2\pi m}\right)\right) + \frac{k_b T_e}{e}\ln\left(\frac{A_1 + A_2 \exp\left[\frac{e\phi_b}{k_b T_e}\right]}{(A_1 + A_2)}\right) \tag{3.38}$$

$$\frac{G_{e_2}}{G_{tot}} = \frac{\exp\left[\frac{e\phi_b}{k_b T_e}\right]}{\frac{A_1}{A_2} + \exp\left[\frac{e\phi_b}{k_b T_e}\right]} \tag{3.39}$$

Où G_{tot} est le flux total collecté aux parois et est défini Eq. (3.23). A_2 correspond à la surface sur laquelle est appliquée la tension de polarisation (surface grisée Fig. 3.18) et A_1 à la surface totale restante du domaine de simulation, soit dans ce cas ci $A_1 = 5A_2$.

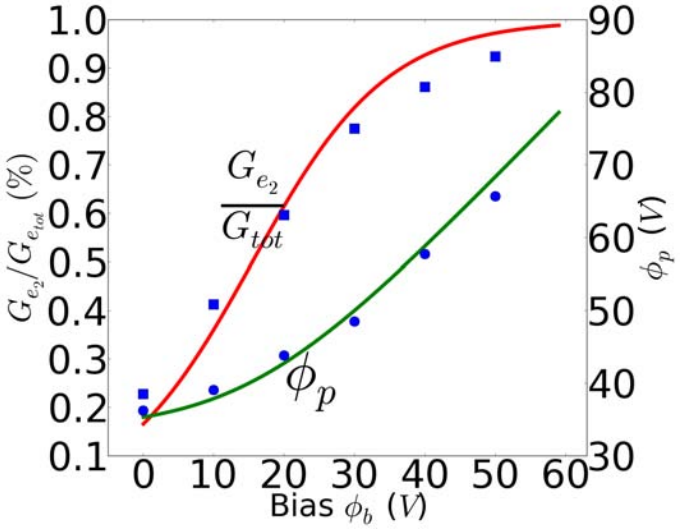
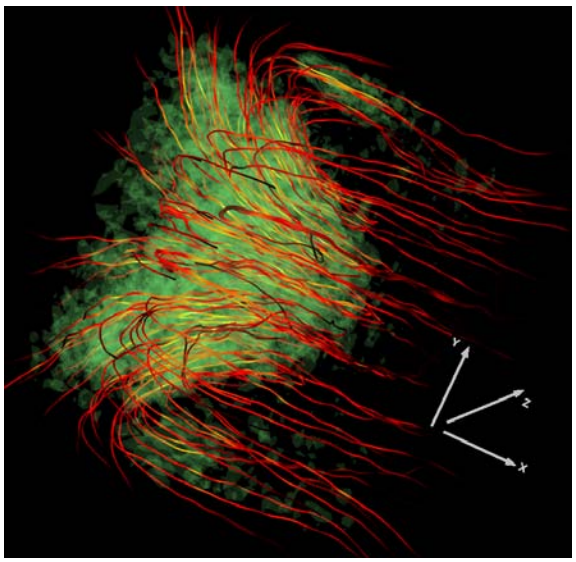


FIGURE 3.19 – Flux théorique d'électrons collectés sur l'électrode droite de la source en pourcentage en fonction de la tension appliquée (ligne rouge) et comparaison avec les résultats numériques (carrés bleus). Comparaison entre le potentiel plasma théorique (ligne verte) et numérique (cercles bleus) en fonction de la tension appliquée. Les valeurs théoriques du flux $\frac{G_{e_2}}{G_{tot}}$ et du potentiel plasma ϕ_p sont obtenues pour une température de ~ 9.5 eV.

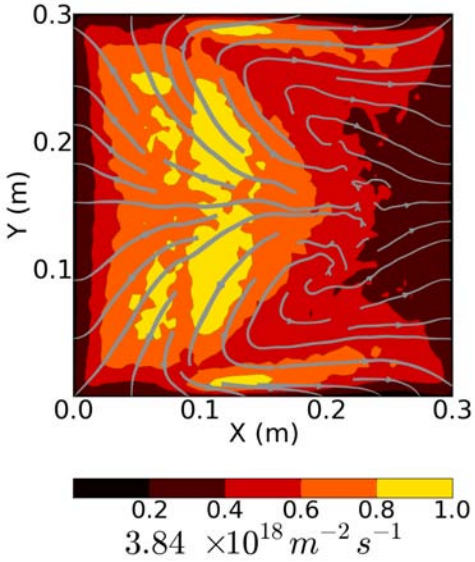
Les valeurs théoriques du potentiel plasma (ϕ_p) en fonction de la tension appliquée à l'électrode (ϕ_b) calculées à partir de l'Eq. (3.38) sont représentées par la courbe (courbe verte) de la figure 3.19. On suppose la température électronique constante et ne dépendant pas de la tension appliquée et dans nos conditions de simulation $k_b T_e/e \simeq 9.5$ eV. Les valeurs numériques de ϕ_p obtenues lors des simulations exécutées dans les conditions de la Fig. 3.18 sans champ magnétique, sont en bon accords avec la courbe théorique Fig. 3.19. De la même façon, la fraction du flux collecté à la paroi est représentée sur la Fig. 3.19 et montre les valeurs obtenues lors des simulations (carré bleus) et les valeurs théoriques obtenues à partir de l'Eq. (3.39) (courbe rouge). Enfin, les résultats obtenus étant très similaires à ceux observés dans la Section 3.3.1 dans le cas 2D sans champ magnétique

et n'apportant pas de façon évidente plus d'informations, nous n'approfondirons pas dans l'analyse de ces résultats [1]. Cependant, dans le but d'observer une analogie avec les résultats de la Section 3.3.1, la Fig. 3.20 montre le flux électronique obtenu à partir du cas 3D.



(a)

(b)



(c)

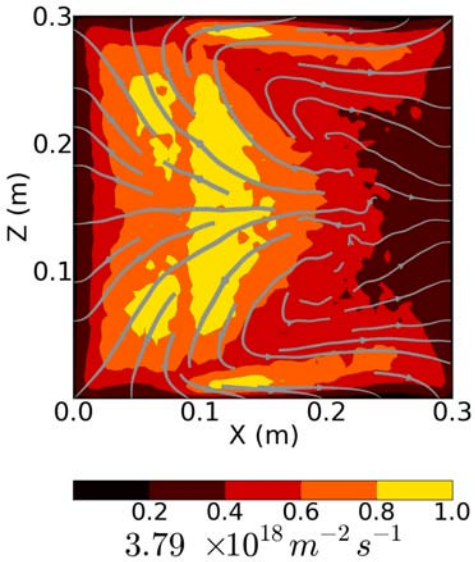


FIGURE 3.20 – (a) Distribution 3D du flux d'électrons. La zone verte représente un isocontour du flux en 3D et où les lignes rouges correspondent à la direction du flux. Profil 2D du flux électronique (m^{-2} s^{-2}) (b) dans le plan (x, y) et (c) dans le plan (x, z). Ces résultats sont obtenus à partir des simulations 3D et sont intégrés dans les directions z et y respectivement. Les couleurs correspondent à l'intensité du flux et les lignes directionnelles à la direction du flux.

1. Le lecteur pourra se référer à la section 3.3.1 pour le détails des résultats obtenus en l'absence de champ magnétique.

Le flux électronique moyen est montré à titre représentatif en 3D (Fig. 3.20(a)) dans les conditions de la Fig. 3.18, dans le plan (x, y) (Fig. 3.20(b)) et dans le plan (x, z) (Fig. 3.20(c)). On peut observer une distribution 2D similaire à celle observée dans la Section 3.3.1 sur la Fig. 3.10(a) avec l'apparition de vortex de courant et dans un cas sans tension appliquée à la paroi. De même, une symétrie du flux dans le plan parallèle à la direction axiale peut être observée comme ce qui est observé dans la Section 3.3.1, où le caractère de la distribution du flux est également discuté.

3.4.3 Avec filtre magnétique

Nous avons discuté dans la Section 3.3.2 de la diffusion des électrons à travers le filtre magnétique dans le cas de simulations à deux dimensions. Plusieurs résultats ont été montrés afin de caractériser au mieux les effets de ce filtre magnétique. Il a pu être observé que les parois dirigées perpendiculairement à l'axe y étaient responsables *via* le champ électrostatique E_y (dirigé vers l'extérieur des parois) d'une dérive électronique $\mathbf{E} \times \mathbf{B}$ dans la zone de champ magnétique, permettant ainsi aux électrons de traverser axialement la barrière magnétique. De plus, cette dérive dirigée vers la zone d'extraction au niveau de la paroi supérieure et dirigée vers l'intérieur de la source au niveau de la paroi inférieure, entraîne une asymétrie des composantes du plasma telles que dans le cas de la température électronique, de la densité ou encore du potentiel plasma.

Les résultats présentés dans ce qui suit ont pour but l'observation des effets pouvant éventuellement apparaître avec l'ajout de parois due à la troisième dimension. Autrement dit, il est intéressant d'observer si la paroi elle-même ou bien le champ électrostatique E_z dirigé vers l'extérieur de la source et perpendiculaire à la paroi dans la dimension z induisent d'autres dérives et/ou d'autres phénomènes de transport électronique.

La Fig. 3.21 montre de façon schématique le domaine de simulation de la source de plasma. Contrairement aux différents cas de simulations précédents, la zone de chauffage (e.g. le "driver") est restreinte à une plus petite zone. Les électrons sont ainsi chauffés uniformément dans cette région (région bleutée sur la figure) qui correspond à 30% de la longueur L dans la direction axiale et à 50% de la longueur dans les deux autres directions (y et z).

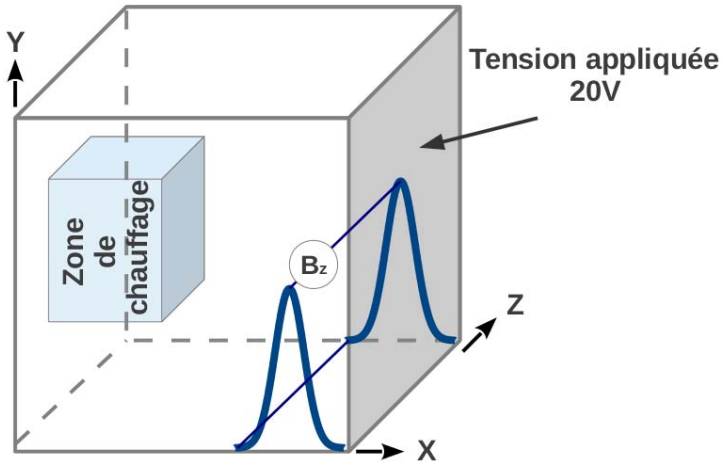


FIGURE 3.21 – Domaine de simulation (3D Cartésien). La densité de gaz (H_2) dans la chambre est de 5×10^{19} m^{-3}. les électrons sont chauffés de façon uniforme dans la région en bleu (La puissance absorbée est de 6.75 W) et correspond à 30% de la longueur dans la direction x et de 25% à 75% dans les directions y et z.

Le filtre magnétique est situé dans la partie droite de la source et est dirigé le long de la composante z, soit $\mathbf{B} = (0, 0, B_z)$. Ce champ suit une distribution Gaussienne donnée par l'Eq. (3.32) et est centré en $x_o = 0.24$ cm avec une largeur à mi hauteur $\sigma = 0.03$ m.

La Fig. 3.22 montre les résultats d'une simulation de la source de plasma avec une tension appliquée de 20 V à la paroi (cf. Fig. 3.21). Ces distributions 2D sont représentées dans le plan (x, y) et sont intégrées le long de l'axe z.

Ces résultats sont très similaires à ceux présentés Fig. 3.13. On observe une forte asymétrie dans la zone de champ magnétique entre la paroi supérieure et inférieure de la chambre concernant le potentiel plasma, la densité et la température électronique (cf. Figs. 3.21(a)-(c)). Nous noterons également que dans la région du filtre magnétique, la température électronique Fig. 3.21(c) au niveau de la paroi inférieure de la source est plus basse à l'endroit où la densité d'électrons devient plus importante Fig. 3.21(b). Ceci a pour effet de tendre vers une pression électronique Fig. 3.21(d) plus homogène entre l'intérieur de la source et la zone de champ magnétique.

Une étude récente a été réalisée par J.P Boeuf et al.[21] dans le but d'étudier les effets du filtre magnétique dans le cadre de la source d'ions négatifs pour ITER. Cette étude montre les résultats obtenus à partir d'une simulation PIC MCC à deux dimensions et avec une géométrie plus réaliste et plus proche de la configuration géométrique de la source d'ions négatifs. Il est intéressant de constater que les résultats obtenus dans le cadre de cette étude sont très proche de ceux observés ici et que pour des conditions analogues de simulation, on observe une différence de 2% à 5% sur les distributions de potentiel plasma et de la température électronique.

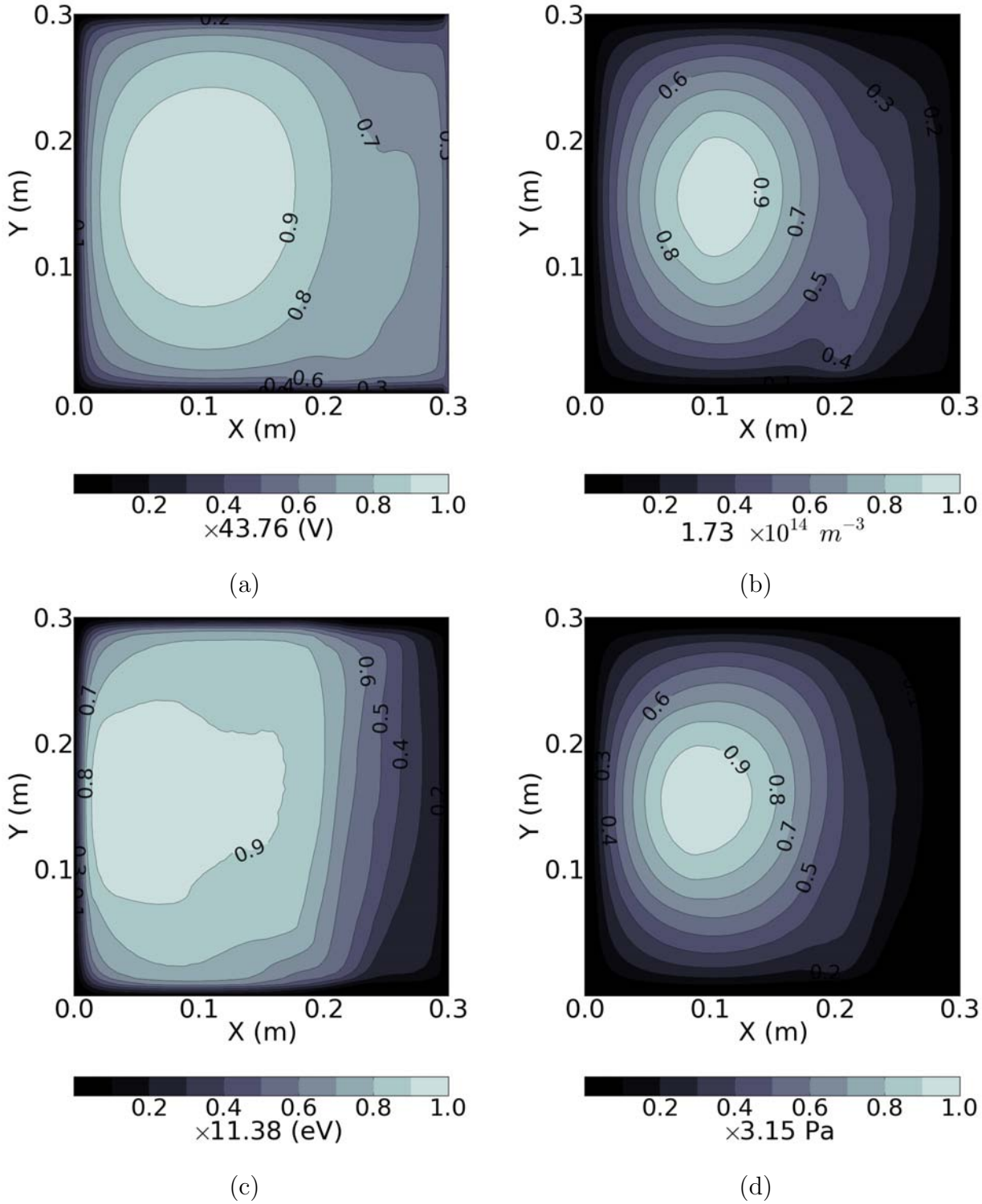


FIGURE 3.22 – (a)Distribution 2D du potentiel plasma (V), (b) de la densité d'électrons (m^{-3}), (c) de la température électronique (eV), (d) de la pression des électrons (10^{-4} Pa). Ces distributions dans le plan (x, y) sont obtenues à partir des résultats 3D intégrés dans la dimension z. Une tension de polarisation de 20 V est appliquée sur la paroi à droite de la source (cf. Fig. 3.21).

La Fig. 3.23 montre le flux électronique issu de la simulation 3D en présence du filtre magnétique. La Fig. 3.23(a) montre ce flux dans un espace tridimensionnel où les lignes représentent la direction du flux et les couleurs à l'intensité correspondante. Cette figure est montrée à titre représentatif mais permet l'observation d'une distribution uniforme du flux le long de la direction z.

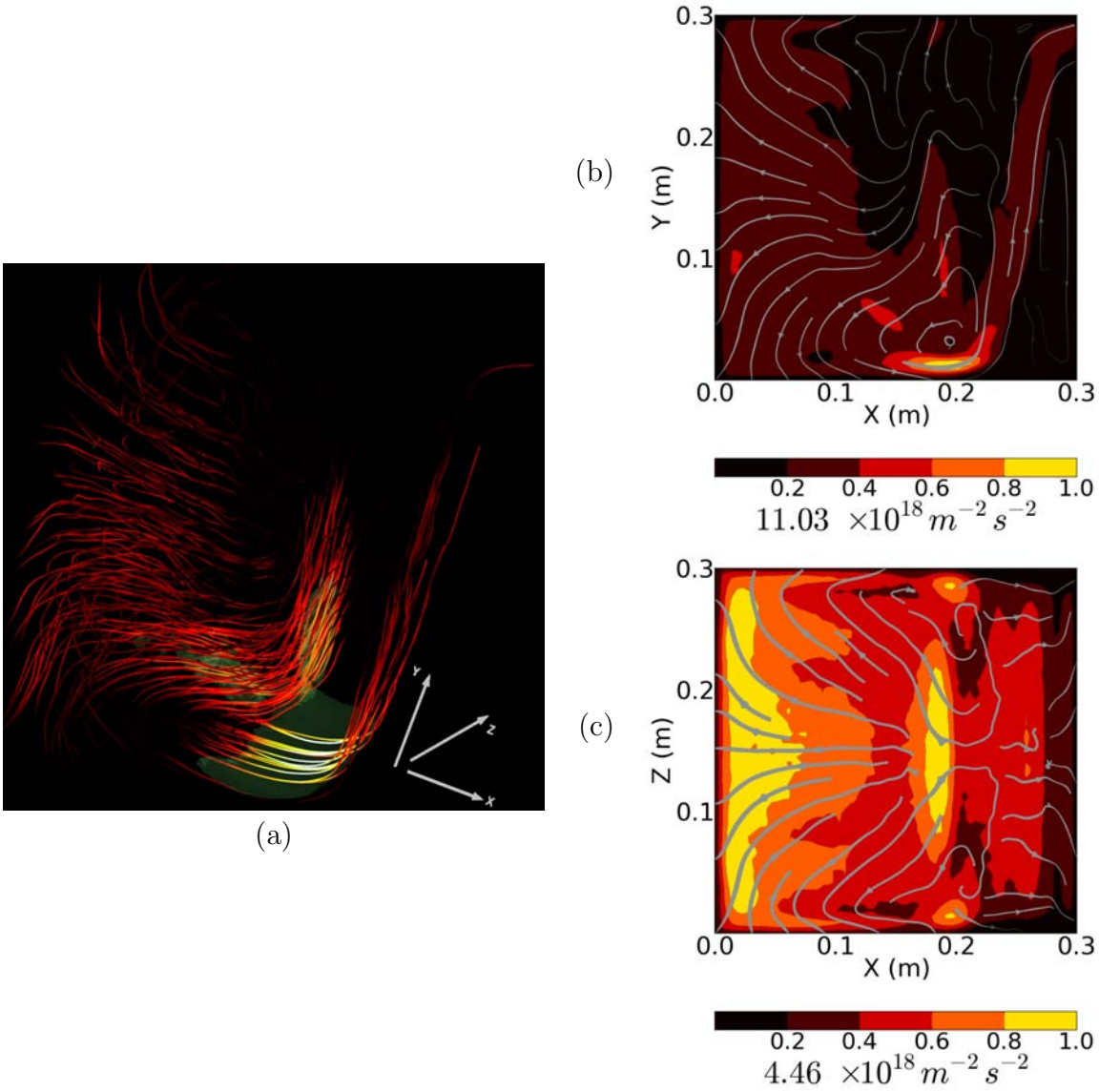


FIGURE 3.23 – (a) Distribution 3D du flux d'électron. La zone verte représente un isocontour du flux en 3D et où les lignes rouges correspondent à la direction du flux. Profil 2D du flux électronique (m^{-2} s^{-2}) (b) dans le plan (x, y) et (c) dans le plan (x, z). Ces résultats sont obtenus à partir des simulations 3D et sont intégrés dans les directions z et y respectivement. Les couleurs correspondent à l'intensité du flux et les lignes directionnelles à la direction du flux. Ces résultats sont obtenus avec un champ magnétique maximum $B_{z0} = 3$ mT et une tension de polarisation de 20 V.

Autrement dit, les parois dans la troisième direction ne montrent pas d'effets significatifs sur le transport des électrons à travers la barrière. De plus, le champ électrostatique E_z étant dirigé parallèlement au champ magnétique B_z, celui ci ne peut être responsable d'une dérive croisée supplémentaire.

La Fig. 3.23(b) montre le flux électronique dans le plan (x, y) (e.g. vue de face de la Fig. 3.23(a)) et la Fig. 3.23(b) dans le plan (x, z) (e.g. vue du dessus de la Fig. 3.23(a)). Ces deux distributions 2D sont intégrées le long de la troisième direction, soit respectivement le long de z et de y.

Une comparaison des distributions obtenues dans le cas des simulations 2D Fig. 3.10(a) et 3D Fig. 3.23(b) permet d'observer la forte similitude dans le transport des électrons dans la source de plasma. On observe dans les deux cas une forte intensité du flux électronique au niveau de la paroi inférieure à l'entrée du filtre magnétique remontant vers la paroi supérieure de la source due à la dérive diamagnétique pour ensuite, traverser le filtre axialement au niveau de la paroi supérieure grâce à la dérive $\mathbf{E} \times \mathbf{B}$. Nous noterons cependant que le flux d'électrons est directement perdu à la paroi droite de la chambre après avoir traversé la barrière magnétique sur la Fig. 3.23(b) contrairement à la Fig. 3.10(a) où le flux électronique est dirigé selon l'axe y dans le sens négatif. Ceci vient du fait que dans le cas de la simulation 2D, aucune tension de polarisation n'est appliquée sur la paroi de droite contrairement au cas présent où une tension de 20 V est appliquée.

Enfin, la Fig. 3.23(c) met en évidence une forte intensité du flux électronique située au milieu de la source de plasma en $x = 0.2$ m. Ceci est également visible sur la Fig. 3.23(b) où le flux est dirigé selon l'axe y et dans le sens négatif. Comme il a été décrit dans la Section 3.3.1, ce transport est principalement dû à la dérive $\mathbf{E} \times \mathbf{B}$ et est causé par l'apparition d'un champ électrostatique axial E_x à l'entrée du filtre magnétique et qui tend à réduire la diffusion des ions à travers la barrière.

3.5 Conclusion

Le transport électronique dans la source de plasma avec et sans filtre magnétique a pu être analysé tout au long de ce chapitre avec l'aide des simulations PIC MCC 2D-3D dans une géométrie cartésienne. L'influence de la barrière magnétique a pu être mise en évidence à partir d'une première étude des caractéristiques du plasma sans champ magnétique (Section 3.3.1) puis par la comparaison des résultats obtenus en présence du filtre magnétique (Section 3.3.2).

Un des principaux résultats de cette étude est l'observation du transport électronique axial à travers le filtre magnétique. Il est montré dans le cas 2D comme dans le cas 3D que le transport des électrons est dû à la dérive $\mathbf{E} \times \mathbf{B}$ au niveau de la paroi supérieure du domaine de simulation et confirme les observations précédemment décrites à partir de modèles fluides [77] et de modèles PIC MCC [24, 22]. La distribution 2D du flux électronique et l'analyse des différents termes de l'équation de conservation de la quantité de mouvement des électrons permet de mettre en avant les dérives diamagnétiques ($\nabla P \times \mathbf{B}$) et les dérives $\mathbf{E} \times \mathbf{B}$

responsables du transport électronique dans la source de plasma. On montre également que la dérive $\mathbf{E} \times \mathbf{B}$ dirigée vers l'intérieur de la source au niveau de la paroi inférieure à l'entrée du filtre magnétique et dirigée dans le sens contraire au niveau de la paroi supérieure est responsable de la forte asymétrie de la densité de plasma, du potentiel et de la température électronique.

Enfin, une étude paramétrique a été réalisée et montre la variation du flux électronique collecté à la paroi extractrice sur laquelle différentes valeurs de tensions ont été appliquées. Il est intéressant d'observer que le flux d'électrons traversant la barrière magnétique et considérablement plus important que le flux observé dans un cas idéal 1D où le transport est collisionnel et est contrôlé par les collisions Coulombiennes électrons-ions.

Nous rappellerons cependant que le modèle utilisé tout au long de ce chapitre est un modèle simplifié et que la chimie de l'hydrogène, la présence de différentes espèces d'ions positifs ou encore la présence des ions négatifs dans le plasma ne sont pas considérées et peuvent certainement avoir des conséquences sur les propriétés du plasma (densité, température, asymétrie, ...). De plus, des comparaisons systématiques entre les prédictions du modèle et les résultats expérimentaux concernant l'asymétrie du plasma sont nécessaires et permettraient de confirmer les hypothèses et les explications fournies à partir du modèle.

Conclusion générale

Un code Particle-In-Cell avec collisions Monte-Carlo en deux et trois dimensions a été développé dans le cadre de simulations appliquées aux plasmas magnétisés à basse température. Les algorithmes PIC MCC sont des méthodes demandant d'importantes ressources de calculs et de mémoire et ces simulations où évoluent plusieurs millions de particules peuvent atteindre des temps de calcul très longs.

Depuis les années 1990 années, les outils de parallélisation apparus avec les architectures multi-coeurs ont permis une très grande avancée dans le développement de modèles et dans la simulation de problèmes de plus en plus réalistes. Cependant, nous avons pu voir dans la Section 1.1 qu'après plus d'une trentaine d'années d'évolution qui ont vu les avancées successives de l'électronique et des techniques de fabrication de ces CPU ainsi que des performances toujours plus grandissantes, apparaissent aujourd'hui des contraintes majeures liées notamment à la fabrication de transistors de plus en plus petits.

En parallèle, poussé par l'industrie du jeu vidéo, le développement de processeurs dédiés aux traitements graphiques appelés communément cartes graphiques ou GPU, présentés dans la Section 1.3, a connu une croissance importante. Aujourd'hui, l'implantation de ce type de matériel est largement répandu et depuis une dizaine d'années les capacités de ces processeurs intéressent la communauté scientifique, en particulier pour la simulation numérique. Cette utilisation détournée de la carte graphique (appelée GPGPU pour *General Purpose computing on Graphics Processing Units*) permet un traitement massivement parallèle des données et une puissance de calcul importante à un coût réduit.

Le cadre de cette thèse s'inscrit donc dans l'utilisation de ce type d'architecture massivement parallèle pour la simulation numérique et plus particulièrement à la modélisation des particules chargées du plasma à partir de l'implémentation de l'algorithme PIC MCC.

L'implémentation de cette méthode a été présentée dans le chapitre 2. Le principe de l'implémentation sur GPU pour chaque module du code PIC MCC y est décrit et met en avant les avantages et les contraintes spécifiques à l'utilisation de ces architectures. Les performances du code de simulation et les temps de calculs passés dans chaque module sont comparés à un code PIC MCC similaire opérant sur un CPU. On montre que l'implémentation sur GPU permet un gain de temps relativement important, compris entre 10 et

20 fois plus rapide comparativement au CPU et que ce temps de calcul varie en fonction de la taille de la simulation et se révèle nettement plus performant pour des simulations comprenant un grand nombre de particules et une grille relativement fine. Ces gains de temps paraissent d'autant plus importants que les temps de calcul pour des simulations comprenant plusieurs millions de macro-particules sont très grands et peuvent prendre plusieurs jours.

La simulation de plasmas à basse température dans des conditions réalistes suppose la création de particules (ionisation) et la perte de celles-ci (aux parois) à chaque pas en temps. Dans le but de décrire ces phénomènes, une technique simple a été implémentée et est décrite dans la Section 2.3 qui permet de réordonner les indices des particules à chaque pas en temps. Il est montré également que le module concernant le calcul des charges sur la grille est le plus contraignant en terme d'implémentation sur GPU et en terme de temps de calcul. Ce module prend environ 40% du temps total d'exécution du calcul ce qui est nettement supérieur aux autres modules dont le temps d'exécution est de l'ordre de 24% pour le transport des particules et inférieur à 20% du temps de calcul pour les autres modules. Un autre constat peut être fait concernant le module de résolution de l'équation de Poisson où dans le cas de simulations 2D et 3D, ce module ne prend que environ 10% du temps de calcul contrairement à la parallélisation sur CPU qui se révèle moins efficace notamment pour des hautes résolutions de grilles. Les cartes graphiques représentent donc un bon moyen pour la parallélisation intensive des codes PIC MCC avec des gains de temps pouvant être supérieurs à 20 fois le temps de calcul de codes PIC séquentiels sur CPU. De plus, les simulations PIC MCC en 3D peuvent être réalisées sur le GPU mais l'espace mémoire disponible est encore trop limité pour permettre des simulations 3D avec un grand nombre de particules et des tailles de grilles relativement importantes sans avoir recours à la mémoire du CPU.

Toutefois, l'application du code en 2D et 3D a pu être réalisée lors de l'étude du transport électronique à travers un filtre magnétique dans le chapitre 3. Une première validation a été donnée lors de la comparaison des temps de calcul entre le code PIC MCC sur GPU et celui sur CPU. Un très bon accord entre les résultats a été trouvé bien que le code sur CPU soit en double précision contrairement au code sur GPU qui utilise la simple précision. De plus, il est montré dans le cadre de simulations sans filtre magnétique, dans la Section 3.3.1 dans le cas 2D et dans la Section 3.4.2 pour le cas 3D un bon accord entre les résultats analytiques et les résultats numériques. L'étude de la source de plasma sans le filtre magnétique permet également de caractériser l'évolution du plasma en fonction de la tension de polarisation et définir le comportement général de celui-ci. La Section 3.3.2 permet de montrer quels sont les effets dus au filtre magnétique par comparaison avec les résultats de la Section 3.3.1. Le résultat important qui est décrit ici est la confirmation d'un

transport axial à travers la barrière magnétique non collisionnel et dû à la dérive $\mathbf{E} \times \mathbf{B}$. Une analyse détaillée des différents termes de l'équation de conservation de la quantité de mouvement des électrons permet de mettre en évidence les termes dominants conduisant aux dérives diamagnétiques ($\nabla P \times \mathbf{B}$) et aux dérives $\mathbf{E} \times \mathbf{B}$. De plus, les résultats du flux électronique montrent également comment ces dérives combinées permettent le transport à travers le champ magnétique. La dérive $\mathbf{E} \times \mathbf{B}$ dirigée vers la zone d'extraction dans le haut de la source de plasma et dirigée dans le sens opposé dans le bas de la chambre conduit à la formation d'une forte asymétrie de la densité de plasma, du potentiel et de la température électronique. Enfin, la présence de parois perpendiculaires au courant diamagnétique est responsable de la dérive $\mathbf{E} \times \mathbf{B}$ et du transport à travers la barrière magnétique contrairement au cas de simulations 1D où le transport à travers le filtre magnétique est collisionnel et est contrôlé par les collisions Coulombiennes électrons-ions. Ainsi le flux total d'électrons collectés traversant la barrière en fonction de la tension appliquée à la paroi, est nettement plus importante dans le cas 2D où l'on observe une dépendance du flux en $1/B$ que dans le cas idéal 1D où la dépendance varie en $1/B^2$.

La simulation en 3D présentée dans la Section 3.4, dans des conditions similaires au cas 2D avec la présence du filtre magnétique, ne montre pas de changements significatifs du transport électronique à travers la barrière magnétique comparativement au cas 2D. Cependant, une étude plus approfondie doit être effectuée notamment pour observer la variation du flux total d'électrons collectés à travers le filtre. Il serait intéressant de voir dans les mêmes conditions de simulation que dans le cas 2D, si l'on observe la même variation du flux électronique et/ou si les parois perpendiculaires à l'axe z jouent un rôle supplémentaire.

Finalement, la parallélisation des codes PIC MCC sur les cartes graphiques se montre relativement intéressante du point de vue du temps de calcul et semble être idéale pour l'implémentation de codes 3D. Les modèles PIC semblent montrer toutefois une limitation du gain de temps de calcul (de l'ordre de $\times 20 - 30$). Le nombre relativement élevé de macro-particules et la difficulté à paralléliser efficacement le calcul des charges sur les noeuds de la grille reste le facteur limitant et ne permettra pas d'atteindre les temps de calcul beaucoup plus important comme dans le cas de simulations fluides ou encore des modèles statistiques pour la finance ou des gains de temps de plus de 100 peuvent être observés (cf. [78]).

De plus, il serait intéressant d'utiliser ces cartes pour permettre une visualisation des données et rendre les traitements et les diagnostiques des résultats efficaces notamment pour la visualisation des résultats issus des simulations en 3D qui peut se révéler complexe. Force est de constater que l'engouement de la communauté scientifique pour ces nouvelles architectures a permis dans un sens un développement important des outils lié à la programmation GPGPU pendant ces dix dernières années et est encore en pleine évolution. C'est

pourquoi, on peut penser que les architectures graphiques peuvent encore augurer, dans un avenir relativement proche, des performances nettement supérieures à celles observées actuellement et ouvrir la voie à de nouvelles méthodes de simulations.

Annexe A

Fonctions CUDA

A.1 Opération atomique

Une opération atomique est un ensemble d'actions s'exécutant sans pouvoir être interrompues. Cette notion prend toute son importance dans le cas d'applications *"multithreads"*. Les opérations atomiques sont utilisées dans le cas présent, pour prévenir les interactions entre les threads et éviter alors les écritures dans une même banque mémoire de façon simultanées [1].

Dans certaines situations, des cas très simples sur des applications *"monothread"* peuvent se révéler très complexes lors d'applications sur architectures massivement parallèles, i.e *"multithreads"*. Un des exemples qui illustre simplement cette situation est l'incrémentation d'une variable.

Dans le langage standard C-C++, l'opérateur d'incrémentation s'écrit : $i++$. L'exécution de cette opération résulte de l'ajout de 1 à la valeur i. Les trois étapes nécessaires (appelé aussi *"read-modify-write"*) peuvent être résumées comme suit :

1	Lecture de la valeur de la variable i.
2	Ajout de la valeur 1 à la valeur lue dans l'étape 1.
3	Ecriture du résultat dans l'emplacement mémoire de i.

Supposons maintenant que nous disposons de deux threads "A" et "B" et supposons également que chaque thread veut incrémenter la variable i. Chacun des threads (A et B) devra effectuer les trois étapes que nous venons de décrire. Cependant, si la lecture de la valeur de i est réalisée simultanément par le thread A et par le thread B, le résultat obtenu ne sera en aucun cas celui escompté. Plusieurs scénarii peuvent être obtenus à partir des étapes précédentes selon l'ordre d'exécution de celles-ci. Le tableau suivant (cf. Table. A.1) montre un cas éventuel.

1. couramment appelée *"race conditions"*

TABLE A.1 – Exemple de scénario pour l'incrémentation de i par les threads A et B : Exemple pour $i = 5$ initialement.

Le thread A lit la valeur de i	A lit 5.
Le thread B lit la valeur de i	B lit 5.
Le thread A ajoute 1 à la valeur de i	A exécute $i + 1 = 6$
Le thread B ajoute 1 à la valeur de i	B exécute $i + 1 = 6$
Le thread A écrit le résultat dans l'espace mémoire de i	$i = 6$
Le thread B écrit le résultat dans l'espace mémoire de i	$i = 6$

Dans l'exemple Table. A.1, l'exécution par un thread des trois étapes nécessaires (*"read-modify-write"*) ne doit pas être interrompue par un autre thread afin d'obtenir le bon résultat ($i = 7$). Autrement dit, l'exécution de ces trois opérations ne peut être scindée en de plus petites parties par d'autres threads. C'est pourquoi, les opérations qui satisfont à cette contrainte sont appelées ***"atomique"***.

Le point faible de ces opérations atomiques vient du fait que chaque appel sera sérialisé. Ceci a pour effet de réduire considérablement le temps d'exécutions.

A.2 Opérations de Réduction

La Réduction de données permet de calculer, à partir d'un flux de données entrant, un sous-flux ou un élément unique. Autrement dit, les opérations dites de réduction consistent à effectuer des opérations successives de type somme ou produit sur les éléments d'un tableau. Un exemple simple est la somme des éléments d'un tableau où l'élément d'indice i d'un tableau est additionné avec l'élément d'indice $i+1$. Soit pour un tableau T de taille n, cette opération est donnée par :

$$S = \sum_{i=0}^{n} T_i \tag{A.1}$$

Où S est un scalaire et correspond à la somme de tous les éléments de T. Cependant, une dépendance liée à la modification des variables à chaque itération peut être observée dans le cas d'un traitement séquentiel de cette somme.

Pour la parallélisation de cet algorithme sur GPU, le principe de base est de partitionner ces données en réalisant des sommes partielles. La Fig. A.1 montre un exemple de traitement en parallèle de l'opération de réduction.

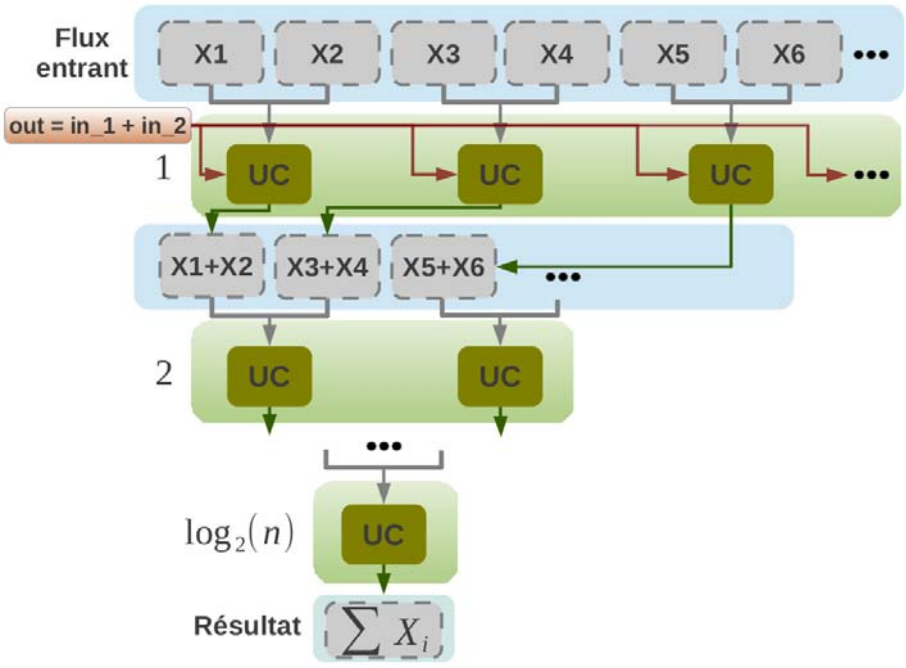


FIGURE A.1 – Exemple de Réduction : somme sur une liste de données. Les données sont placées dans un flux en entrée. A chaque itération du processus, la taille du tableau est divisée par deux. Un élément d'index i du flux de sortie contient la somme suivante de valeurs du flux d'entrée. Après $\log_2(n)$ itérations, on obtient un nouveau flux de taille 1 contenant la somme de tous les éléments initiaux.

La taille des données, initialement n, est divisée par 2, le flux de données entrant est découpé en morceaux et chacune de ces parties étant envoyée à une unité de calcul parallèle. Chaque thread va exécuter sur ces données la suite d'instructions qu'on lui a confié, puis va écrire un résultat à l'endroit idoine dans le flux sortant. Lorsqu'une unité a fini de traiter sa partie courante, une synchronisation des threads est nécessaire afin d'éviter les conflits en mémoire. Après au maximum $\log_2(n)$ itérations (dans le cas d'une dimension des données en puissance de 2), la sortie est réduite au sous-flux ou à l'élément voulu. La complexité temporelle d'une telle réduction est de l'ordre de $\mathcal{O}(np\log(n))$, avec p le nombre d'unités de calcul disponibles sur le matériel. Séquentiellement, la même réduction aurait une complexité de l'ordre de $\mathcal{O}(n)$.

Enfin, plusieurs algorithmes de réductions ont été développés sur GPU et son mis à disposition dans diverses librairies. De plus, M. Harris propose une optimisation des algorithmes de réduction spécifiques pour les architectures NVIDIA (Réf. [60]) et montre comment il est possible d'utiliser la mémoire partagée pour augmenter significativement le temps de calcul pour ce type d'opérations.

A.3 Scan (ou *prefix sum*)

Le Scan est un algorithme qui, appliqué à une séquence, en calcule une autre de même taille, dans laquelle chaque élément est la somme de tous les éléments d'indices inférieurs dans la séquence d'entrée. Soit le tableau de données S_1 de taille n :

$$S_1 = [a_0, a_1, \ldots, a_{n-1}] \text{ avec } \begin{cases} a_i = 0 \text{ ou } 1 \text{ et } i = 0, \ldots, n \\ n = \text{nombre total de particules} \end{cases}$$

Alors, l'opération de scan sur le tableau S_1 est donnée par :

$$S_2 = S_1[a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1})] \text{ avec } i = 0, \ldots, n \tag{A.2}$$

Malgré une nature apparemment séquentielle, il existe différents algorithmes parallèles efficaces pour réaliser cette opération, également appelée *prefix-sum*.

Hillis et al.[79] furent dans les premiers à proposer une implémentation parallèle de l'algorithme de scan, ou *all partial sums* sur un tableau. Un pseudo-code basé sur leur implémentation est présenté par l'algorithme List. A.1. La complexité de cet algorithme est en $\mathcal{O}(n \log_2(n))$, avec n la taille de la séquence d'entrée, ce qui est moins bon que la complexité de l'algorithme séquentiel. La parallélisation de l'algorithme introduit ici une baisse théorique de performance, compensée en pratique par l'exécution parallèle de ce code.

Listing A.1– Kernel : Algorithme naïf de scan parallèle

```
for (i = 0; i < log2(n); i++){
  for (k){ /*En parallele*/
    if (k ≥ 2^i)
    {x[k] = x[k - 2^(i-1)] + x[k]}
  }
}
```

Cependant, plusieurs améliorations ont été apportées pour l'optimisation de l'algorithme. Sengupta et al.[80] ont depuis présenté un nouvel algorithme, utilisant au mieux la parallélisation et s'exécutant en $\mathcal{O}(n)$, avec n la taille des données en entrée. Cette amélioration provient d'une décomposition du scan en deux phases. une phase de réduction pour calculer certaines sommes partielles, suivie d'une recomposition, appelée *upsweep* et une deuxième phase, appelée *downsweep*. La phase de réduction et de downsweep sont

telles qu'elles permettent d'occuper efficacement l'ensemble des processeurs, optimisant la parallélisation du calcul.

Une version implémentée en CUDA a été proposée par Harris et al.[43]. Cet algorithme présente une adaptation efficace de l'algorithme de Blelloch [81] et de Sengupta [80]. D'autres améliorations ont ensuite été proposées, telle que l'algorithme appelé *segmented scan* [82], autorisant un découpage plus arbitraire du flux de données en entrée. Cette version a été appliquée à des tris de type *quicksort* (Réf. [46]) ou à des multiplications de matrices creuses. Leur implémentation s'appuie sur certaines propriétés spécifiques à CUDA, notamment sur l'utilisation de la mémoire partagée.

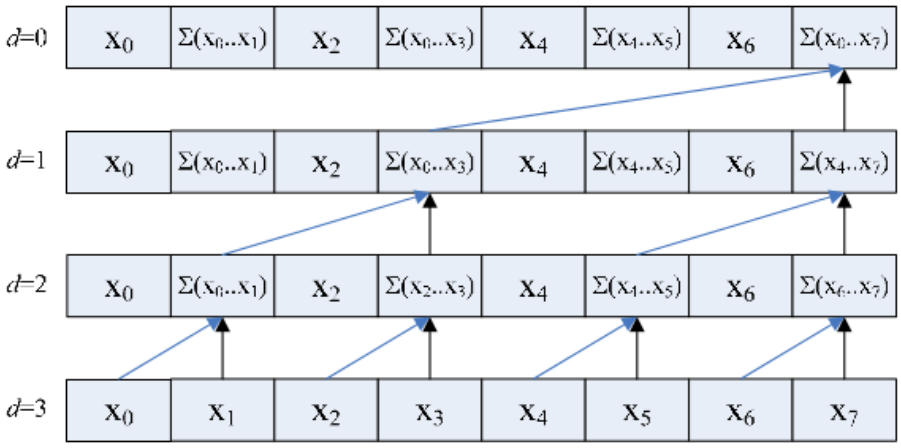


FIGURE A.2 – Phase de réduction (upsweep) (Réf. [43]).

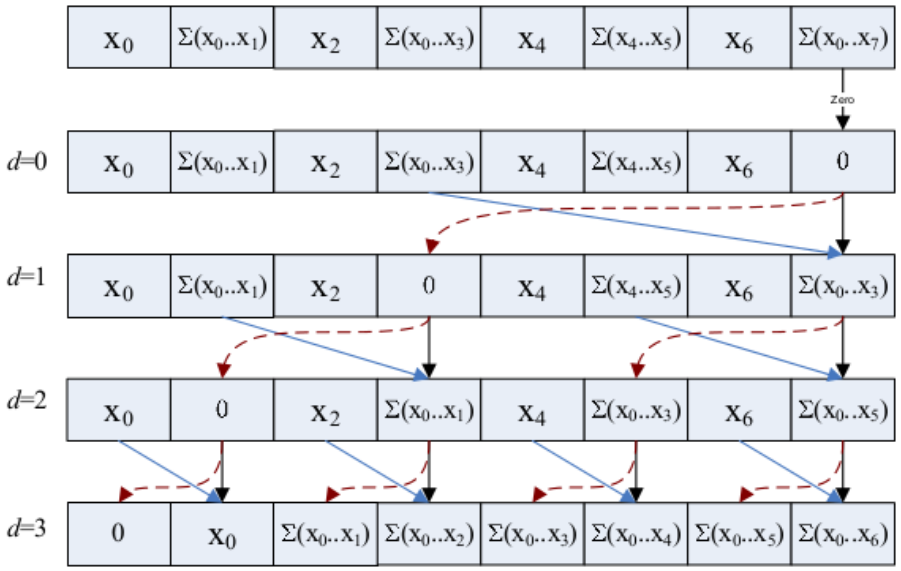


FIGURE A.3 – Phase de downsweep (Réf. [43]).

Annexe B

Méthodes numériques

B.1 Calcul des vitesses post-collisions

Dans le cas d'un processus de collision élastique, on a la réaction suivante :

$$A + B \rightarrow A + B$$

Pour une particule A incidente, dont la masse m est très petite devant celle de la particule cible B que l'on note M ($m << M$). De plus, on fait l'hypothèse que la vitesse v_A est très grande devant celle de la particule cible : $v_A >> v_B$.

Les lois de conservation de la quantité de mouvement et de l'énergie nous permettent d'écrire :

$$m\vec{v_A} + M\vec{v_B} = m\vec{v_A}^* + M\vec{v_B}^* \tag{B.1a}$$

$$\frac{1}{2}mv_A^2 + \frac{1}{2}Mv_B^2 = \frac{1}{2}mv_A^{*2} + \frac{1}{2}Mv_B^{*2} \tag{B.1b}$$

Ici, v^* représente la vitesse de la particule concernée après la collision. On introduit deux autres quantités, qui sont la vitesse du centre de masse et la vitesse relative, respectivement définies par v_{cm} et v_r (Eq. (B.2a, B.2b)) :

$$\vec{v_{cm}} = \frac{m\vec{v_A} + M\vec{v_B}}{m + M} \tag{B.2a}$$

$$\vec{v_r} = \vec{v_A} - \vec{v_B} \tag{B.2b}$$

D'après Eq. (B.2b), on a $\vec{v_B} = \vec{v_r} - \vec{v_A}$ et d'après Eq. (B.1a, B.2a), on a $m\vec{v_A} + M\vec{v_B} = \vec{v_{cm}}(m + M)$.

On remplace dans l'équation (B.1a) :

$$\begin{aligned} m\vec{v_A} + M(\vec{v_A} - \vec{v_r}) &= \vec{v_{cm}}(m+M) \\ \vec{v_A} &= \vec{v_{cm}} + \frac{M}{m+M}\vec{v_r} \end{aligned} \tag{B.3}$$

De même avec $\vec{v_B}$, on obtient :

$$\vec{v_B} = \vec{v_{cm}} - \frac{m}{m+M}\vec{v_r} \tag{B.4}$$

On élève au carré les équations (B.3) et (B.4) et on remplace les termes de v_A^2, v_B^2 dans l'équation de conservation d'énergie Eq. (B.1b).

$$\begin{aligned} mv_A^2 &= mv_{cm}^2 + 2\mu\vec{v_{cm}}\cdot\vec{v_r} + m\left(\frac{M}{m+M}\right)^2 v_r^2 \\ Mv_B^2 &= Mv_{cm}^2 - 2\mu\vec{v_{cm}}\cdot\vec{v_r} + M\left(\frac{m}{m+M}\right)^2 v_r^2 \\ mv_A^2 + Mv_B^2 &= \mu v_r^2 \quad \text{avec la masse réduite } \mu = \frac{mM}{m+M} \end{aligned} \tag{B.5}$$

On peut donc réécrire l'équation (B.1b) :

$$\frac{1}{2}\mu(\vec{v_A} - \vec{v_B})^2 = \frac{1}{2}\mu(\vec{v_A^*} - \vec{v_B^*})^2 \tag{B.6}$$

On a alors :

$$\vec{v_A^*} - \vec{v_B^*} = (\vec{v_A} - \vec{v_B})\cdot\vec{R} \tag{B.7}$$

Où $\vec{R}$ est un vecteur unitaire de directions aléatoires.

On peut déterminer ainsi la vitesse de la particule après la collision en remplaçant dans (B.1a) l'équation (B.7).

$$\begin{aligned}
\vec{v_B^*} &= \vec{v_A^*} - (\vec{v_A} - \vec{v_B}) \cdot \vec{R} \\
m\vec{v_A^*} + M(\vec{v_A^*} - (\vec{v_A} - \vec{v_B}) \cdot \vec{R}) &= m\vec{v_A} + M\vec{v_B} \\
\vec{v_A^*}(m+M) - M(\vec{v_A} - \vec{v_B}) \cdot \vec{R} &= m\vec{v_A} + M\vec{v_B} \\
\boxed{\vec{v_A^*} = \tfrac{1}{m+M}\left(m\vec{v_A} + M\left(\vec{v_B} + (\vec{v_A} - \vec{v_B}) \cdot \vec{R}\right)\right.} &
\end{aligned} \tag{B.8}$$

Dans le cas d'un processus d'ionisation, on a la réaction suivante :

$$A + B \rightarrow A + B^+ + C$$

Où A et C sont respectivement les électrons incident et éjecté et B^+ l'atome ionisé. Les équations de conservations sont :

$$\begin{aligned}
m\vec{v_A} + M\vec{v_B} &= m\vec{v_A}^* + (M-m)\vec{v_B}^* + m\vec{v_C}^* \\
\vec{v_{cm}}(m+M) + \vec{v_{cm}}(m+M) - \vec{v_{cm}}(m+M) &= m\vec{v_A}^* + (M-m)\vec{v_B}^* + m\vec{v_C}^* \\
2\vec{v_{cm}}(m+M) - \vec{v_{cm}}(m+M) &= m\vec{v_A}^* + (M-m)\vec{v_B}^* + m\vec{v_C}^*
\end{aligned}$$

$$m\underbrace{(\vec{v_A^*} - \vec{v_{cm}})}_{\tilde{\vec{v}}_A^*} + m\underbrace{(\vec{v_C^*} - \vec{v_{cm}})}_{\tilde{\vec{v}}_C^*} + (M-m)\underbrace{(\vec{v_B^*} - \vec{v_{cm}})}_{\tilde{\vec{v}}_B^*} = 0$$

soit l'équation de conservation $\boxed{m\tilde{\vec{v}}_A^* + m\tilde{\vec{v}}_C^* + (M-m)\tilde{\vec{v}}_B^* = 0}$ (B.9)

Où, v^* dénote la vitesse de la particule après la collision et ($\tilde{}$), la vitesse à laquelle la vitesse du centre de masse v_{cm} est soustraite. De même pour l'équation de conservation de l'énergie, on écrit :

$$\boxed{\tfrac{1}{2}\mu(\vec{v_A} - \vec{v_B})^2 = \tfrac{1}{2}m\tilde{v}_A^{*2} + \tfrac{1}{2}m\tilde{v}_C^{*2} + \tfrac{1}{2}(M-m)\tilde{v}_B^{*2} + E_{ion}} \tag{B.10}$$

E_{ion} est l'énergie d'ionisation. D'après l'équation (B.9) : $\tilde{\vec{v}}_B^* = -\frac{m}{M}(\tilde{\vec{v}}_A^* + \tilde{\vec{v}}_C^*)$, où $m << M$. On constate d'après l'équation précédente, que le troisième terme du membre de droite de l'équation (B.10) est négligeable devant les deux autres premiers termes du membre de droite. On peut donc réécrire cette équation :

$$\boxed{\frac{1}{2}\mu(\vec{v_A}-\vec{v_B})^2 = \frac{1}{2}m\tilde{v_A^{*2}} + \frac{1}{2}m\tilde{v_C^{*2}} + E_{ion}}$$
$$\boxed{\frac{1}{2}\mu(\vec{v_A}-\vec{v_B})^2 - E_{ion} = \Delta E} \tag{B.11}$$

ΔE est l'excès d'énergie après la collision. On divise alors ΔE en deux et on répartit l'énergie entre les deux électrons comme suit :

$$\frac{1}{2}m\tilde{v_A^{*2}} = r\Delta E \tag{B.12}$$
$$\frac{1}{2}m\tilde{v_C^{*2}} = (1-r)\Delta E \tag{B.13}$$

Où r est un nombre aléatoire compris entre 0 et 1. Enfin, les nouvelles vitesses des particules incidentes et éjectées sont mises à jour :

$$\boxed{\tilde{\vec{v}}_A^* = \vec{R}\tilde{v}_A^* \qquad \tilde{\vec{v}}_C^* = \vec{\dot{R}}\tilde{v}_C^*} \tag{B.14}$$

Et $\vec{R}$ et $\vec{\dot{R}}$ sont deux vecteurs unitaires de direction. Enfin, les vitesses dans le référentiel du laboratoire sont obtenues en ajoutant $\vec{v_{cm}}$ à $\tilde{\vec{v}}_A^*, \tilde{\vec{v}}_B^*, \tilde{\vec{v}}_C^*$.

Bibliographie

[1] NVIDIA. *"NVIDIA CUDA C Programming guide"*, 2010.

[2] J. M. Rax. *Physique des Plasmas*. Dunod, 2005.

[3] O Buneman. "Time reversible difference procedures". *J. Comput. Phys*, 1, june 1967.

[4] O Buneman and D Dunn. "Computer experiments in plasma physics". *Science Journal*, 2, july 1966.

[5] J. M. Dawson. "One dimensional plasma model". *Phys. Fluids*, 5, april 1962.

[6] François Gernelle. 2eme colloque de l'histoire de l'informatique. `http://www.ahti.fr/CNAM90.pdf`, 1990.

[7] F Gernelle. 5eme colloque de l'histoire de l'informatique. `http://www.ahti.fr/Toulousegernelle.pdf`, 1998.

[8] G. E. Moore. "Cramming more components onto integrated circuits". *Electronics*, April 1965.

[9] F. Anceau. *"La saga des microprocesseurs, la course à la puissance"*. Conservatoire national des arts et métiers, november 1998.

[10] W. J. Broad. Incredible shrinking transistor nears its ultimate limit : The laws of physics. `http://query.nytimes.com/gst/fullpage.html?res=9D05E3DB113DF937A35751C0A961958260`, 1997.

[11] R Kelsall. *Nanoscale Science and Technology*. New York :Wiley, 2005.

[12] Science Daily. Pushing the limits of computer chip miniaturization. `http://www.sciencedaily.com/releases/2008/01/080112083626.htm`, 2008.

[13] GPGPU. General-purpose computation on GPU. `http://gpgpu.org`, 2002.

[14] P. O. Jääskeläinen, C. S. Lama, P Huerta, and J. H. Takala. "OpenCL-based design methodology for application-specific processors". *IEEE*, july 2010.

[15] M Aafta. "OpenCL parallele computing on the gpu". In *SIGGRAPH'08*, pages 4,26,160,165. SigGraph, 2008.

[16] M Aafta. "The OpenCL specification". In *SIGGRAPH'08*, pages 4,26,160,165. SigGraph, 2008.

[17] N Trevett. "OpenCL - heterogeneous parallel programming". In *SIGGRAPH'08*, pages 4,26,160,165. SigGraph, 2008.

[18] M. J. Flynn. "Some computer organizations and their effectiveness". *IEEE Transactions on Computers*, c-21, september 1972.

[19] J. C. Adam, A Héron, and G Laval. "Study of stationary plasma thrusters using two-dimensional fully kinetic simulations". *Physics of Plasmas*, 11, January 2004.

[20] L Garrigues, A Heron, J. C. Adam, and J. P. Boeuf. "Hybrid and particle-in-cell models of a stationary plasma thruster". *Plasma Sources Sci. Technol*, 9, 2000.

[21] J. P. Boeuf, J Claustre, B Chaudhury, and G Fubiani. "Physics of a magnetic filter for negative ion sources : II. EXB drift through the filter in a real geometry". *Phys. of Plasmas*, 2012.

[22] G Fubiani, G. J. M. Hagelaar, J. P. Boeuf, and S Kolev. "Modeling a high power fusion plasma reactor-type ion source : Applicability of particle methods". *Phys. of Plasmas*, 19, 2012.

[23] S Kolev, G. J. M. Hagelaar, and J. P. Boeuf. "Particle-in-cell with monte carlo collision modeling of the electron and negative hydrogen ion transport across a localized transverse magnetic field". *Physics of plasmas*, 16, 2009.

[24] S Kolev, G. J. M. Hagelaar, G Fubiani, and J. P. Boeuf. "Physics of magnetic barrier in low temperature bounded plasmas - insight from particle-in-cell simulations". *Plasma Source Sci.Technol*, 21, 2012.

[25] H. X. Vu and J. U. Brackbill. "CELEST1D : an implicit, fully kinetic model for low-frequency, electromagnetic plasma simulation". *Computer Physics Communications*, 69, 1992.

[26] K Fujimoto and R. D. Sydora. "Electromagnetic particle in cell simulations on magnetic reconnection with adaptative mesh refinement". *Computer Physics Communications*, 178, 2008.

[27] A. B. Langdon. "On enforcing Gauss'law in electromagnetic particle-in-cell codes". *Computer Physics Communications*, 70, 1992.

[28] G Lapenta and S Markidis. "Particle acceleration and energy conservation in particle in cell simulations". *Phys. of Plasmas*, 18, 2011.

[29] G Lapenta. "Particle simulations of space weather". *Journal of Computational Physics*, 231, february 2012.

[30] M Drouin, L Gremillet, J. C. Adam, and A Heron. "Particle-in-cell modeling of relativistic laser-plasma interaction with the adjustable damping, direct implicit method". *Journal of Computational Physics*, 229, 2010.

[31] G Lapenta. "Adaptative multi-dimensional particle-in-cell". *Phys. of Plasmas*, 2008.

[32] K Birdsall and A. B. Langdon. *Plasma Physics via Computer Simulation*. Mc Graw-Hill, 1985.

[33] R Hockey and J Eastwood. *Computer Simulation using Particles*. London :Taylor and Francis, 1988.

[34] P Abreu, R. A. Fonseca, J. M. Pereira, et al. "PIC codes in new processors : A full relativistic PIC code in CUDA-enabled hardware with direct visualization". *IEEE Transactions on Plasma Science*, 39, 2011.

[35] P Mertmann, D Eremin, T Mussenbrock, et al. "Fine-sorting one dimensional particle-in-cell algorithm with monte-carlo collisions on a graphics processing unit". *Computer Physics Communications*, may 2011.

[36] V. K. Decyk and T. V. Singh. "Adaptable particle-in-cell algorithm for graphical processing units". *Computer Physics Communications*, november 2010.

[37] H Burau, R Widera, W Hönig, et al. "PIConGPU : A fully relativistic particle-in-cell code for a GPU cluster". *IEEE Transaction on Plasma Sci.*, 38, october 2010.

[38] J Suzuki, H Shimazu, K Fukazawa, and M Den. "Acceleration of PIC simulation with GPU". *Plasma and Fusion Research*, 6, 2011.

[39] Nolan Goodnight, Cliff Woolley, et al. "A multigrid solver for boundary value problems using programmable graphics hardware". In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '03, pages 102–111. Eurographics Association, 2003.

[40] J Bolz, I Farmer, E Grinspun, et al. "Sparse matrix solver on the GPU : Conjugate gradients and multigrid". In *ACM SIGGRAPH Courses*, volume 22(3). ACM Transactions on Graphics, july 2003.

[41] G Stantchev, W Dorland, and N Gumerov. "Fast parallel particle-to-grid interpolation for plasma PIC simulation on GPU". *Journal of Parallel and Distributed Computing*, 2008.

[42] J. P. Boris. "Relativistic plasma simulation - optimization of a hybrid code". In *Proceedings of Fourth Conf Num Sim Plasmas*, pages 3–67. Naval Res. Lab., 1970.

[43] Mark Harris. *"Parallel Prefix sum (scan) with CUDA"*. NVIDIA, 2007.

[44] N Satish, M Harris, and M Garland. "Designing efficient sorting algorithms for manycore GPUs". Technical report, NVIDIA Corporation, Sept 2008.

[45] S Le grand. "Broad-phase collision detection with CUDA". Technical report, Addison wesley, 2007.

[46] Duane Merrill and Andrew Grimshaw. "Revisiting sorting for GPGPU stream architectures". Technical Report CS2010-03, University of Virginia, Department of Computer Science, Charlottesville, VA, USA, 2010.

[47] W. L. Briggs, V.E Henson, and S. F. McCormick. *A Multigrid tutorial*. SIAM, 2000.

[48] U Trottenberg, C. W. Oosterlee, and A Schüller. *MultiGrid*. Academic Press, 2001.

[49] A Brandt. "Multi-level adaptive solutions to boundary value problems". *Math. Comput.*, 31, 1977.

[50] A Brandt, W Hackbush, and U Trottenberg. *Guide to multigrid development, Multigrid Methods*. Berlin : (Lecture notes in Mathematics) Springer, 1982.

[51] W. L. Briggs and S. F. McCormick. *Introduction, Multigrid Methods*. Philadelphia :SIAM, 1987.

[52] K Stüben and U Trottenberg. *Multigrid Methods : fundamental algorithms, model problem analysis and applications*. Berlin : (Lecture notes in Mathematics) Springer, 1982.

[53] W Hackbush. *Multi-grid Methods and applications*. Berlin : Springer, 1985.

[54] I Yavneh. "On red-black SOR smoothing in multigrid". *SIAM Journal of Sci. Comput.*, 17 :180–192, 1996.

[55] V Vahedi and M Surendra. "A monte carlo collision model for the particle-in-cell method : Applications to argon and oxygen discharges". *Computer Physics Communications*, 87 :179, 1995.

[56] Kenichi Nanbu. "Probability theory of electron–molecule, ion–molecule, molecule–molecule, and coulomb collisions for particle modeling of materials processing plasmas and gases". *IEEE Trans. on Plasma Sci.*, 28, june 2000.

[57] C. K. Birdsal. "Particle-in-cell charged-particle simulations, plus monte carlo collisions with neutral atoms, PIC-MCC". *IEEE Trans. on Plasma Sci.*, 19, april 1991.

[58] Phelps. database,. `http://www.lxcat.laplace.univ-tlse.fr`, retrieved August 13, 201.

[59] NVIDIA. *CURAND library*, 2010.

[60] Mark Harris. *"Optimizing Parallel Reduction in CUDA"*. NVIDIA Developer Technology, 2.3 edition, 2008.

[61] OpenMP. Open specifications for multi-processing. `http://openmp.org`.

[62] Marc D. Raymanq, Philip Varghese, David H. Lehman, and Leslie L. Livesay. "Results from the deep space 1 technology validation mission". *Acta Astronautica*, 2000.

[63] ITER. International thermonuclear experimental reactor. `http://www.iter.org`.

[64] B Schunke, D Bora, R Hemsworth, A Tanga, et al. "Negative ion based heating and diagnostic neutral beams for ITER". In *Negative ions, Beams and sources*, volume 1097, pages 480–490. AIP conference proceedings, 2009.

[65] E Speth and other. "Overview of the RF source development programm at IPP garching". *Nuclear Fusion*, 46 :s220, 2006.

[66] U Fantz and other. "Physical performance analysis and progress of the development of the negative ion RF source for the ITER NBI system". *Nuclear Fusion*, 49, 2009.

[67] EFDA : European Fusion Development Agreement. Fusion : Plasma heating and current drive. `https://www.efda.org/fusion/focus-on/plasma-heating-current-drive/neutral-beam-injection`.

[68] J. P. Boeuf, J Claustre, B Chaudhury, and G Fubiani. "Physics of a magnetic filter for negative ion sources : II. EXB drift through the filter in a real geometry". *accepted to Physics of Plasmas*, october 2012.

[69] M Fröschle, S Leyer, P Franzen, ch Martens, E Speth, et al. "Technical overview and first results of the half-size ITER NBI source". *Fusion Engineering and Design*, 82, 887-896 2007.

[70] J. P. Boeuf, G. J. M. Hagelaar, P Sarrailh, G Fubiani, and N Kohen. "Model of an inductively coupled negative ion source : Application for ITER type source". *Plasma Source Sci.Technol*, 20 :015002, 2011.

[71] M. A. Lieberman and A. J. Lichtenberg. *Principles of plasma Discharges and Materials Processing*. New York :Wiley, 2005.

[72] F Chen. *Introduction to plasma physics and controlled fusion*. New York :Plenum, 2008.

[73] P. M. Bellan. *Fundamentals of plasma physics*. Cambridge :Cambridge University press, 1984.

[74] J. P. Boeuf, B Chaudhury, and L Garrigues. "Physics of a magnetic filter for negative ion sources : I. collisional transport across the filter in an ideal, 1D filter". *Phys. of Plasmas*, 2012.

[75] E. A. Bogdanov, A. S. Chirtsov, and A. A. Kudryavtsev. "Fundamental nonambipolarity of electron fluxes in 2D plasmas". *Physical Review Letters*, 106, 2011.

[76] T Lafleur and R. W. Boswell. "Particle-in-cell simulations of ambipolar and nonambipolar diffusion in magnetized plasma". *Phys. of Plasmas*, 19, 2012.

[77] G. J. M. Hagelaar and N Oudini. "Plasma transport across magnetic field lines in low-temperature plasma sources". *Plasma Phys. and Control. Fusion*, 53, 2011.

[78] NVIDIA cuda. cuda zone. `http://www.nvidia.fr/object/cuda_apps_flash_new_fr.html#state=home`, 2012.

[79] W. D. Hillis and Steele Jr. G. L. "Data parallel algorithms". *Commun. ACM*, 29, 1986.

[80] S Sengupta, A. E. Lefohn, and J. D. Owens. "Work-efficient step-efficient prefix sum algorithm". In *Workshop on Edge Computing Using New Commodity Architectures*, pages 26–27, 2006.

[81] G. E. Blelloch. *Vector models for data-parallel computing.* Cambridge : MIT press, 1990.

[82] S Sengupta, M. J. Harris, Y Zhang, and J. D. Owens. "Scan primitives for gpu computing". *ACM : Graphics Hardware*, 2007.