Plateforme de calcul parallèle “ Design for Demise ”

Bastien Plazolles

Université de Toulouse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *10/01/2017* **par :**

Bastien PLAZOLLES

Plateforme de calcul parallèlle "Design for Demise"

JURY

M. Thierry GAYRAUD	Président du Jury, Professeur (UT3)	LAAS-CNRS, Toulouse
M. David DEFOUR	Maître de Conférence (HDR)	UPVD, Perpignan
M. Rafael MOLINA	Ingénieur	ESA/ESTEC, Keplerlaan 1, 2200 AG Noordwijk
Mme. Marianne BALAT-PICHELIN	Directrice de Recherche	CNRS/PROMES - UPR 8521, Font Romeu Odeillo
M. Philippe COCQUEREZ	Ingénieur	CNES, Toulouse
M. Didier EL BAZ	Chargé de Recherche (HDR)	LAAS-CNRS, Toulouse
M. Martin SPEL	Ingénieur	R.Tech, Verniolle

École doctorale et spécialité :
MITT : Domaine STIC : Sureté de logiciel et calcul de haute performance
Unité de Recherche :
Laboratoire d'Analyse et d'Architecture des Systèmes - CNRS
Directeur(s) de Thèse :
Didier EL BAZ et *Martin SPEL*
Rapporteurs :
David DEFOUR et *Rafael MOLINA*

à ma famille et mes amis. . .

Remerciements

Cette thèse a bénéficié d'un financement CIFRE de la part de l'Association Nationale de la Recherche et de la Technologie (ANRT) et a été réalisée au Laboratoire d'Analyse et d'Architecture des Systèmes en co-encadrement avec la société R.Tech sous la direction du Docteur Didier El Baz et sous la co-direction de Monsieur Martin Spel.

Je tiens à remercier le Directeur de Recherche Liviu Nicu, Directeur du LAAS-CNRS pour m'avoir acceuilli dans les locaux de son laboratoire durant ces trois années de thèse ainsi que Monsieur Martin Spel, gérant de la société R.Tech, qui m'a aussi acceuilli dans les locaux de sa société.

Je tiens à remercier tout particulièrement Monsieur Martin Spel pour m'avoir fait confiance et m'avoir proposé ce sujet de thèse et je souhaite remercier mes directeurs de thèse Didier El Baz et Martin Spel, pour m'avoir accompagné dans ces travaux et guidés par leurs conseils tout en me laissant une grande liberté dans la conduite de mes recherches. Je les remercie aussi pour toutes les opportunitées qu'ils m'ont offertes que ce soit l'organisation d'évènements internationaux (workshops SCDW à Toulouse et Lisbonne, IEEE Smart World Congress à l'Université Paul Sabatier, Toulouse) ou l'encadrement de formations professionnelles (formations NVIDIA: CUDA et Deep Learning au LAAS-CNRS, Toulouse). Je les remercie aussi pour les ouvertures qu'ils m'ont fournies vers d'autres instituts comme le CNES et l'ESA ou d'autres laboratoires dont l'Observatoire Midi-Pyrénées.

Je tiens à remercier les Docteurs Pascal Gegout du GET, Didier Gazen et Juan Escobar de l'OMP pour les nombreuses discussions que nous avons eues ainsi que l'accés privilégié au coprocesseur Intel Xeon Phi.

Je remercie Messieurs Frédéric Parienté et François Courteille de la société NVIDIA Corporation pour nos échanges et collaborations.

Je tiens à remercier également le Docteur David Defour, Maître de Conférences à l'Université de Perpignan (UPVD), ainsi que Monsieur Rafael Molina, Ingénieur à l'Agence Spatiale Européenne (ESA) qui m'ont fait l'honneur d'être rapporteurs de mes travaux de thèse et pour leur lecture attentive du manuscrit.

Je tiens à remercier également la Directrice de Recherche Marianne Balat-Pichelin du laboratoire CNRS/PROMES à Odeillo, ainsi que Monsieur Philippe Cocquerez, ingénieur au CNES à Toulouse qui m'ont fait l'honneur de faire parti de mon jury de thèse.

Je voudrai remercier les personnes avec lesquelles j'ai travaillé. Tout d'abord l'équipe de R.Tech avec laquelle ça a été un plaisir de travailler pendant 4 années: Martin Spel et Vincent Rivola pour les différents projets sur lequels nous avons travaillé ensemble, Philippe Makowski pour avoir réparé ma machine à de nombreuses reprises et toutes nos discussions, Jonathan Lafite pour nos ateliers bricolage et me rappeller périodiquement pourquoi Windows c'est le mal, Fabien Guichard pour m'avoir aidé dans mon processus de transition vers la programmation orienté objet et m'avoir plus que grandement aidé avec le site web du workshop, Didier Lavallard pour m'avoir fait prolonger l'expérience ballon par procuration et me rappeller péridodiquement pourquoi Java c'est le mal, Fabien Tachon et Didier Hermann les derniers arrivés à R.Tech que j'ai trés peu vu au final mais qu'il a été un plaisir de rencontrer. Ensuite l'équipe Calcul Distribué et Asynchronisme du LAAS-CNRS: Didier El Baz évidemment, Bilal Fakhi pour m'avoir fait découvrir le monde passionnant des communications réseaux P2P, Li Zhu et Jia Luo avec qui ça a été un plaisir de partager mon bureau et de découvrir des éléments de culture chinoise.

Sur une note plus personnelle je voudrai ensuite remercier ma famille qui m'a toujours soutenu. Tout particulièrement je voudrais remercier ma mère et mon frère, qui m'ont supporté dans tous les sens du terme, dans les bons moments ainsi que dans les moins bons, pleins de doutes et d'inquiétudes, où je ne suis pas forcément la personne la plus facile à vivre au monde. Pour celà un trés grand merci à vous deux.

Ensuite je voudrai remercier tous mes amis, pour me prendre pour plus intelligent que ce que je suis, tout en me maintenant les pieds sur Terre. Je ne vais pas faire la liste ici, principalement parce que ça risquerait de faire doubler le nombre de pages de ce manuscrit, mais je ne doute pas un instant qu'ils se reconnaitront; et puis ça permet à ceux que ne figureraient pas dans la liste de penser qu'ils en font parti...

Je voudrai aussi remercier ces personnes qui ont guidé mon parcours: Mrs Peired, père et fils, qui les premiers m'ont fait découvrir les merveilles de la physique, M. Ghirardi dont la passion à enseigner les sciences physiques a fini de me convaincre que c'était ce que je voulais faire plus tard: des sciences. Ensuite Arnaud Lepadellec pour le citer: "m'avoir tenu à l'oeil" parce qu'il m'avait repéré en cours, Roser Pello et Didier Barret qui m'ont éncadré tour à tour dans mes travaux universitaires et qui m'ont redonné l'envie et l'énergie lorsque les choses semblaient plus que mal engagées pour moi, Olivier Godet avec qui j'ai, une année durant, étudié les trous noirs, et quel trou noir HLX-1, me permettant d'assouvir ma passion. Enfin *last but not least*, bien au contraire, Dominique Toublanc qui est devenu pour moi bien plus qu'un ensignant.

Et pour finir je voudrai tout particulièrement remercier toutes ces personnes qui n'ont jamais eu confiance en moi et qui n'ont eu de cesse de me répéter que je n'étais bon qu'à entasser des cartons pour un géant du meuble en kit suédois.

Table des matières

Liste des figures

Liste des tableaux

Liste des pseudo-codes

Chapitre 1

Introduction

La question de la gestion des débris de satellites n'est pas une problématique nouvelle même si ce n'est que récemment que les différentes agences spatiales ont commencé à considérer cette dernière comme très importante.
Au milieu des années 70 une série d'études a été effectué à la NASA [14], [9]. Elles dressent un bilan de la situation depuis 1957, date du lancement du premier engin spatial : Sputnik, sans pour autant évoquer les risques liés aux collisions avec ceux-ci en orbite ou bien la menace au sol qu'ils peuvent représenter lors de leur rentrées atmoshériques. On estime à l'époque la masse totale de débris liés aux activités spatiales humaines à environs 900 tonnes de matériaux (cf. Figure 1.1 qui trace l'évolution de la masse de débris en orbite autour de la Terre de 1957 à nos jours). C'est en 1978 que pour la première fois la question des débris est identifiée comme étant problématique. Kessler dans son étude [41] pointe en effet l'augmentation exponentielle du nombre de débris en orbite à cause des effets de cascade: deux débris entrant en collision et se divisant pour engendrer davantage de débris. Même si ces nouveaux débris sont plus petits, ils n'en restent pas moins dangereux puisqu'on considère qu'une collision à 10 km/s avec un objet de 1 cm dégage autant d'énergie que l'explosion d'une grenade à main. Dans son article Kessler [41] commence à émettre les premières recommandations concernant la gestion des débris mais sans que cela ne soit pris en considération par aucune agence spatiale. À ce moment là on estime à 1400 tonnes, la quantité de débris en orbite.
Dans les années 80 plusieurs études sont effectuées [89], [76] et [44] sur les risques liés aux débris spatiaux. En Septembre 1988 l'Agence Spatiale Européenne (ESA) édite le standard ESA-PSS-01-40 qui est le premier standard contenant, entre autre, toute une série de règles visant à réguler la problématique des débris. Le problème majeur de ce standard est d'être beaucoup trop théorique et relativement impossible à appliquer dans la pratique. On en est alors à 2700 tonnes de débris en orbite. Malgré son aspect trop théorique et impratiquable ce premier standard marque le début d'un changement de positionnement des instances envers la question des débris spatiaux. Ainsi en 1995 la

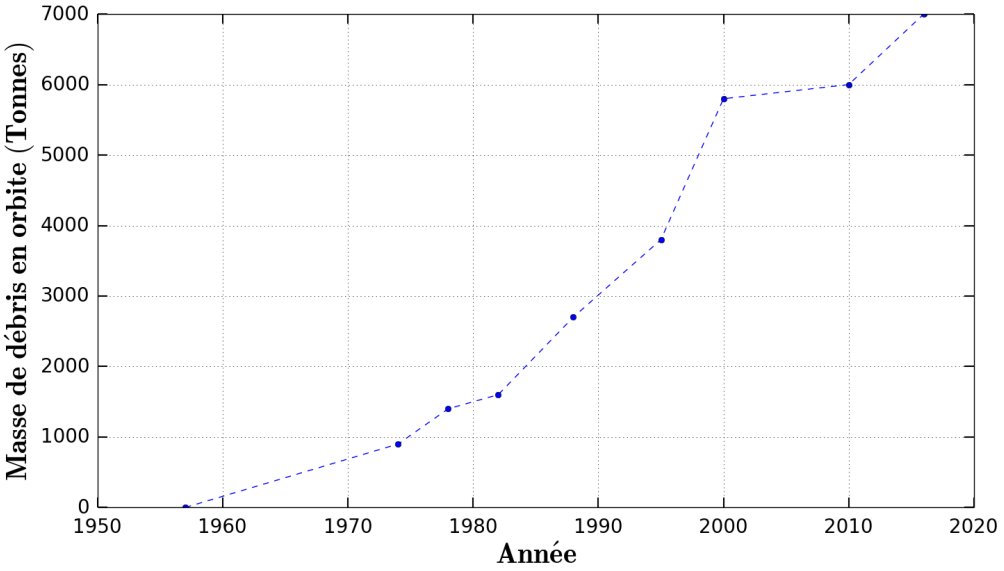

FIGURE 1.1: Évolution de la masse totale de débris en orbite de 1957 à nos jours

NASA éditera le "Guidelines and Assessment Procedures for Limiting Orbital Debris" qui, à la façon du standard édité par l'ESA en 1988, regroupe un ensemble de régles et de principes à appliquer par la NASA pour limiter la production de débris en orbite. Dans le courant des années 2000 on voit aussi apparaitre les premiers standards nationaux dont la Loi sur les Opérations Spatiales (LOS) Française (2008), qui identifie l'agence spatiale française (le CNES) comme responsable de la question des débris et impose, entre autre, qu'un engin ne puisse être envoyé, depuis le territoire français, en orbite que si l'on est en mesure de garantir une maîtrise de sa fin de vie, notamment en garantissant que la probabilité qu'aprés une rentrée atmosphérique il puisse faire une victime soit inférieure à 2.10^{-5} dans le cas d'une rentrée contrôlée ou inférieure à 10^{-4} dans le cas d'une rentrée non contrôlée.

Aujourd'hui aprés environ soixantes années d'opérations spatiales internationales, on estime que près de 4900 fusées ont été tirées, mettant en orbite environ 6600 satellites dont 3600 sont encore en orbite et seulement moins d'un tiers est encore en opération (source ESA). Ainsi on estime que la masse totale des satellites en orbite hors service représente 6300 tonnes tandis qu'on estime à 700 000 le nombre total de débris d'un diamètre supérieur à 1 cm en orbite. Les débris les plus gros sont traqués et catalogués par le US Space Surveillance Network qui suit approximativement 17 500 objets:

- objets de dimension supérieure à 5-10 cm en orbite basse (<2000 km d'altitude)
- objets de dimension supérieure à 0.3 m en orbite géostationnaire (36 000 km)

L'ESA se sert des données issues de ce réseau de surveillance pour contrôler les trajectoires de ses satellites et anticiper les risques de collisions.

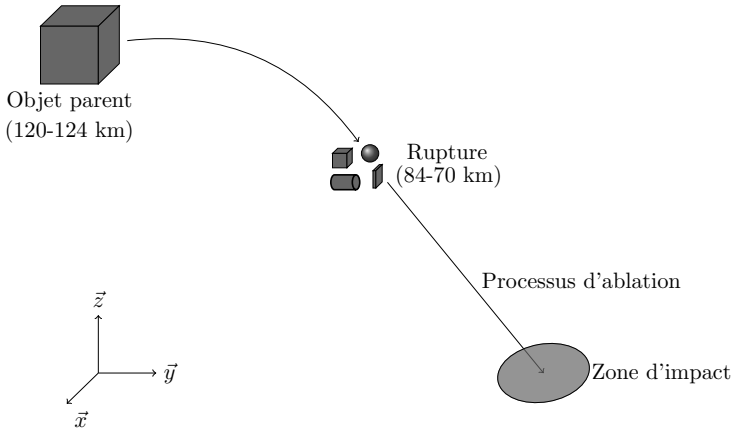

FIGURE 1.2: Représentation de la modélisation orientée objet de la rentrée atmosphérique d'un satellite

En 2010 l'ESA définit une nouvelle norme pour la gestion des débris: la norme ECSS-U-AS10C/ISO 24113. Cette norme, qui a pour vocation de remplacer celle de 1988, définit entre autre le terme "Design for Demise" : principe de conception ayant pour vocation de garantir que l'objet conçu ne pose aucun danger pour les populations au sol suite à une rentrée atmosphérique, en utilisant des matériaux et des géométries qui favorisent que l'objet se désintègre complètement durant la phase de rentrée de telle façon que plus rien ne subsiste et n'heurte le sol.
Afin de pouvoir respecter ces différents standards visant à maîtriser la question des débris, les différentes agences spatiales, mais aussi des industriels, ont développé des outils de simulation de rentrée atmosphérique de débris pour pouvoir déterminer les zones de dommages au sol que peuvent engendrer les débris en retombant ainsi que l'état de ces débris au sol (masse restante, énergie cinétique à l'impact, etc..) et les coordonnées du lieu d'impact. À notre connaissance, à ce jour on dénombre dix logiciels de simulation de rentrée atmosphérique publiés dans la litérature pouvant être regroupés en deux catégories: les logiciels orientés objets et les logiciels orientés véhicules spatiaux.

Les logiciels orientés objets considèrent le satellite comme un assemblage de formes géométriques simples (boite, sphère, cylindre et plaque) qui représentent les constituants critiques du satellite, contenus dans un objet parent qui représente la structure initiale du satellite. Chaque forme géométrique dispose d'une gamme de mouvement autorisée qui peut varier selon le logiciel. En terme de modélisation, typiquement l'objet parent effectue la rentrée atmosphérique jusqu'à atteindre une altitude critique (entre 70 et 84 km d'altitude) où il se décompose en parties élémentaires comme on peut le voir sur la Figure 1.2. Selon le logiciel l'objet parent pourra être équipé de panneaux solaires qui

seront détruits automatiquement à l'altitude critique de fragmentation. Ici la fragmentation de l'objet parent est ainsi prescrite, l'altitude de rupture étant déterminée de façon empirique. Aprés la rupture de l'objet parent chaque élément du satellite est simulé indépendamment des autres jusqu'à ce qu'il se désintègre ou qu'il n'atteigne le sol [83]. La trajectoire d'un objet est calculée à partir des équations du mouvement à 3 degrés de liberté. Dans la plupart de ces outils la modélisation thermique des objets est basée sur l'approche de la masse localisée, qui suppose une distribution uniforme de la température dans l'objet (pas de gradient de température dans la structure de l'objet). L'intérêt de cette modélisation simplifiée est de permettre d'obtenir des résultats numériques en des temps de calculs assez courts. Sur les dix logiciels de rentrée atmosphérique publiés, huit sont des logiciels orientés objets:

- DAS (NASA) [58] : DAS est le premier logiciel dédié à l'analyse de la rentrée de satellite publié. Le mouvement du satellite est propagé avec le propagateur "PROP3D". Les composants du satellites contenus dans l'objet parent sont libérés à l'altitude fixe de 78 km à une température fixe aussi de 300 K. Lors de la rentrée si la température de l'objet devient supérieure à la température de fusion du matériaux le composant, l'objet est intégralement détruit.

- ORSAT (NASA) [25] : le mouvement du satellite est basé sur l'intégration des équations du mouvement à 3 degrés de liberté de Vinh [86]. ORSAT considère pour chaque forme géométrique uniquement des orientations stables ou tournant autour d'un axe donnée. ORSAT permet de prendre en considération l'ablation partielle d'un objet. ORSAT prend aussi en considération les échauffements par oxidation du matériau.

- ORSAT-J (NASDA) [39] : ORSAT-J est une version modifiée du logiciel ORSAT développée par l'Agence au Développement Spatial du Japon (NASDA)

- DRAPS (Chine) [88] : la trajectoire d'un objet est modélisée par un modèle ballistique à 3 degrés de liberté. Le modèle thermique peut être 0D ou 1D. DRAPS gère plus de formes géométriques que les autres logiciels existants. La rupture de l'objet parent n'est pas uniquement déterminée par une altitude de référence mais dépend aussi de la température: si celle-ci dépasse la température de fusion avant l'altitude de rupture, les enfants sont alors libérés. DRAPS est le seul logiciel qui implémente la possibilité d'effectuer des simulations de Monte-Carlo simples en prenant en considération les incertitudes sur les conditions initiales.

- DEBRISK (CNES) [55] : dans DEBRISK la rentrée atmosphérique est simulée en propageant une trajectoire à 3 degrés de liberté via la bibliothèque Sirius en considérant les forces gravitationnelles et aérodynamiques. Le modèle thermique

prend en considération les transferts convectifs, radiatifs ainsi que l'oxydation du matériau. Pour ce qui est de l'ablation partielle, elle s'effectue de façon uniforme sur toute la surface de l'objet une fois que la température de fusion du matériaux est atteinte.

- DRAMA-SESAM (ESA) [2] : ce logiciel orienté objet a été dévelopé par la société HTG pour le compte de l'agence spaciale européenne. La dynamique de vol est modélisée par une trajectoire à 3 degrés de liberté propagée via un intégrateur de Runge-Kutta en prenant en considération les forces gravitationnelles et aérodynamiques. Le modèle thermique considère les charges thermiques sur une masse ponctuelle ainsi que le re-rayonnement. L'ablation partielle s'effectue de façon uniforme sur la surface de l'objet lorsque la température de fusion est atteinte.

- DEBRIS (DEIMOS) [59] : DEBRIS est un logiciel interne développé par la société DEIMOS dans le cadre d'un ensemble d'outils dédiés à l'étude de la rentrée planétaire. Les objectifs de DEBRIS sont: d'estimer l'empreinte au sol, l'analyse de la survivabilité des débris et l'analyse de risque. Dans DEBRIS la fragmentation de l'engin spatial est modélisée comme étant un évènement dépendant des paramètres de la trajectoire (ex.: les efforts thermo-mécaniques sur la structure). Aprés la rupture de l'engin, les débris sont traités comme des objets indépendants. La thermique et la dynamique sont découplées dans la modélisation. La dynamique de vol est modélisée par une trajectoire à 3 degrés de liberté propagée via un intégrateur de Runge-Kutta à pas variable en prenant en considération les forces gravitationnelles et aérodynamiques. Le modèle aérothermodynamique se base sur le calcul au point d'arrêt par la méthode de Detra-Kemp-Riddel.

- Spacecraft Aerothermal Model, SAM (Fluid Gravity et Belstead Research) [8] : la trajectoire de l'objet modélisé est calculée à partir des équations du mouvement à 3 degrés de liberté. Pour ce qui est de la modélisation thermique SAM utilise des corrélations pour le calcul thermique de chaque forme simple. Ces corrélations sont établies à partir de calcul CFD.

Les logiciels orientés engins spatiaux quant à eux, au lieu de représenter l'engin comme un assemblage de formes géométrique simples, utilisent une représentation 3D maillée aussi fidèle que possible à la structure de l'objet réel. La trajectoire du débris est calculée à partir des équations du mouvement à 6 degrés de liberté afin de prendre en considération les mouvement de la structure autour de son centre de gravité. Les forces aéro et aérothermodynamiques sont ainsi calculées par la méthode des éléments finis sur la structure maillée du débris et non sur des formes simplifiées. L'analyse thermique est basée sur une modélisation du transfert thermique 2 ou 3D. Dans ces outils la fragmentation n'est pas imposée à une altitude critique, mais l'ablation et la fragmentation

des composants de l'engin spatial dépendent des contraintes aérothermodynamiques qui s'exercent sur la structure durant la rentrée atmosphérique. En cas de fragmentation chaque fragment est ensuite suivi indépendament jusqu'à ce qu'il atteigne le sol ou ne se désintègre. Cette modélisation plus fidèle qui permet d'obtenir des résultats plus réalistes s'effectue au prix d'un temps de calcul plus important. À ce jour deux logiciels orientés engins spatiaux ont été publiés:

- SCARAB (ESA) [21] : ce logiciel a été développé par la société HTG pour le compte de l'ESA et est le premier logiciel orienté vaisseau spatial publié. SCARAB est doté de son propre module de conception d'engin basé sur l'assemblage de primitives géométriques pannelisées (structure de maillage propre à SCARAB). La trajectoire de la structure est calculée en intégrant les équations du mouvement à 6 degrés de liberté qui décrivent le changement de moment dû aux forces extérieures et au moment de torsion. L'ablation est effectuée en retirant un panneau lorsque la température de celui-ci a atteint la température de fusion de son matériau. La fragmentation s'effectue soit par ablation, soit par étude des contraintes exercées sur des plans de fracture prédéfinis.

- PAMPERO (CNES) [6] : Pampero effectue ses calculs sur un maillage téthrahédrique du satellite ou du débris à étudier, maillage qui aura été généré au préalable par un outil tierce. Afin de déterminer sur quelles cellules du maillage appliquer les forces de pression aéro et aérothermodynamiques, Pampero se base sur la méthode des facteurs de vue. L'ablation s'effectue en retirant des cellules lorsque la température de ces dernières est supérieure à la température de fusion. Les propriétés géométriques de l'objet (rayon de courbure, etc...) sont alors recalculées.

Le principal défaut de tous ces outils c'est d'être déterministe. Or le phénomène de la rentrée atmosphérique est un système extrêmement chaotique. La moindre modification d'une condition initiale peut radicalement changer le résultat d'une simulation qui est en plus binaire avec ces logiciels: soit l'objet arrive au sol à une position donnée, soit il se désintègre durant la rentrée. Cette approche déterministe empêche ainsi de prendre en considération les incertitudes provenant des simplifications effectuées sur la modélisation des phénomènes aéro et aérothermodynamiques en particulier dans les logiciels orientés objets. Mais ce problème concerne aussi les logiciels orientés engin spatiaux qui malgré une modélisation physique plus réaliste, plus couteuse en temps de calcul, ne prennent pas en considération les incertitudes sur les propriétés des matériaux, qui sont mal caractérisés dans les conditions de température et de pression rencontrées durant la phase de rentrée, ni celles sur les conditions initiales de la rentrée atmosphérique de l'engin étudié (position, vitesse, angle d'incidence, etc...) pouvant rendre le résultat de la simulation discutable.

C'est pour cette raison que dans le cadre de ces travaux de thèse nous avons décidé

de développer un nouveau logiciel de rentrée atmosphérique de débris en adoptant une démarche unique. Nous avons pris le parti de développer un logiciel orienté objet, utilisant des cœfficients aéro et aérothermodynamique pré-calculés par des méthodes issues de la modélisation orientée engins spatiaux. L'autre grande particularité de cet outil est la prise en considération des incertitudes sur différents paramètres de modélisation (conditions initiales, propriétés de matériaux, cœfficients aérothermodynamique) afin d'effectuer de façon native des analyses de Monte-Carlo en utilisant la puissance des accélérateurs de calculs de type carte graphique ou Intel Xeon Phi afin de maîtriser les temps de calcul.

Notons qu'il existe relativement peu d'études faisant la comparaison entre les mérites des GPUs et des Xeon Phi pour une application donnée et en particulier pour des applications industrielles ou du monde réel. Seules deux études ont été menées visant à comparer les performances entre les GPUs et les Xeons Phi considérant des problèmes d'algèbre linéaires [75] ou d'analyse d'image [82].

Le plan que nous avons adopté dans ce travail est le suivant:

Au Chapitre 2 nous présentons des généralités sur les méthodes de Monte-Carlo et nous détaillons la méthode de Taguchi.

Au Chapitre 3 nous présentons les accélérateurs de calculs de type Graphics Processing Units (GPU) ainsi que Intel Xeon Phi.

Au Chapitre 4 nous présentons une première étude visant à valider l'utilisation des accélérateurs de calculs dans des conditions opérationnelles. Nous nous intéressons plus particulièrement au cas de la dérive descente de ballons stratosphériques et nous présentons des résultats expérimentaux pour des campagnes effectuées par le Centre National d'Étude Spatiale.

Au Chapitre 5 nous considérons le cas de la rentrée atmosphérique des engins spatiaux et nous détaillons l'outil Calima. Calima permet d'effectuer des analyses statistiques, en parallèle sur accélérateur de calcul, sur le problème de la rentrée atmosphérique en prenant en considération les incertitudes sur les conditions initiales et certains paramètres de modélisation.

La conclusion générale et les perspectives sont présentées ensuite.

Chapitre 2

Outils d'analyse statistique

Dans le présent chapitre nous allons présenter les bases théoriques sur lesquelles reposent l'ensemble des outils d'analyse statistique utilisés durant ces travaux de thèse.

2.1 Méthode de Monte-Carlo

2.1.1 Historique

L'essor et la popularité de la méthode de Monte-Carlo est étroitement lié au développement de l'informatique.
À l'origine de la méthode de Monte-Carlo on trouve tout d'abord une ancienne méthode mathématique nommée Échantillonnage Statistique. Cette ancienne méthode requiert d'échantillonner aléatoirement un grand nombre de valeurs de façon à ce que l'échantillon obtenu soit représentatif du système dont il est issu, et il s'avère difficile pour un individu d'effectuer une telle tâche. Du fait de sa lourdeur et de sa pénibilité, cette méthode finie par tomber en désuétude au début du 20ème siècle.
En 1946 Stan Ulman, John von Neuman et Enrico Fermi assistent à la démonstration du premier ordinateur mécanique ENIAC. Ils réalisent alors très rapidement le potentiel de l'appareil qui permet l'automatisation de la génération de nombres aléatoires et des calculs, rendant alors à nouveau pertinente la méthode d'échantillonnage statistique [46].
En 1947, avec leur équipe, ils traitent le problème de la diffusion de neutron dans les matériaux fissibles avec une méthode fondée sur l'échantillonnage statistique et qu'ils baptisent méthode de Monte-Carlo.
Ainsi grâce à sa facilité d'utilisation et à l'augmentation de la puissance de calcul des machines modernes, les méthodes de Monte-Carlo ont permis de résoudre une grande diversité de problèmes et sont utilisées dans de nombreux domaines : physique [45], résolution de l'équation de Boltzmann [10], biologie (docking moléculaire [84]), calcul d'intégrale (résolution du problème de l'aiguille de Buffon), etc....

2.1.2 Définition

2.1.2.1 Méthode de Monte-Carlo et pseudo-Monte-Carlo

Le terme méthode de Monte-Carlo désigne une technique permettant d'approximer la solution d'un problème mathématique en étudiant la distribution de variables aléatoires [16].
Ainsi soit une fonction $h(X)$. On définit son espérance $E_f[h(X)]$ tel que :

$$E_f[h(X)] = \int_{\chi} h(x)f(x)dx, \tag{2.1}$$

Où χ est l'ensemble continue de valeurs de x et f la densité de probabilité de x dans χ. Nous notons X les variables aléatoires échantillonées dans χ et $P(X)$ la probabilité de tirer X dans l'ensemble χ :

$$P(X) = \int_{\chi} f(x)dx, \tag{2.2}$$

Le principe de la méthode de Monte-Carlo pour calculer (2.1) consiste à générer un échantillon de valeurs $(X_1, X_2, ..., X_N)$ dans χ suivant la fonction de densité f et de calculer la moyenne empirique $\overline{h}_n$:

$$\overline{h}_N = \frac{1}{N}\sum_{j=1}^{N} h(x_j), \tag{2.3}$$

D'après la théorie des Grands Nombres, si l'on tire N valeurs X_i, N suffisamment grand, $\overline{h}_N$ converge vers $E_f[h(X)]$. On peut alors calculer la variance asymptotique de cette approximation selon :

$$var(\overline{h}_N) := v_N = \frac{1}{N}\int_{\chi} (h(x) - E_f[h(X)])^2 f(x)dx, \tag{2.4}$$

qui peut aussi être estimée à partir de l'échantillon $(X_1, X_2, ..., X_N)$ via :

$$v_N = \frac{1}{N}\sum_{j=1}^{N} [h(x_j) - \overline{h}_N]^2, \tag{2.5}$$

De plus, d'après le théorème de la Limite Centrée, pour N grand, on a :

$$Z := \frac{\overline{h}_N - E_f[h(X)]}{\sqrt{v_N}} \equiv \mathcal{N}, \tag{2.6}$$

Z suit ainsi la loi de Gauss (la loi normale centrée réduite), permettant alors de construire des intervalles de confiances encadrant l'erreur commise en remplaçant (2.1) par (2.3), à

partir du calcul de la variance. Typiquement suivant la loi normale pour avoir un niveau de risque d'erreur sur l'estimation de la valeur moyenne de 1%, la valeur moyenne est majorée par $3 * v_N$.
Dans le cas pratique, c'est à dire la résolution d'un problème numérique sur ordinateur, on va utiliser un générateur de nombre pseudo-aléatoire, afin de générer les valeurs présentées ci-dessus. En toute rigueur on parle alors de méthode pseudo-Monte-Carlo, bien que l'on trouve aussi dans la littérature le terme méthode de Monte-Carlo [16]. Dans ce manuscrit afin d'éviter toute lourdeur, le terme méthode de Monte-Carlo sera utilisé pour désigner une méthode pseudo-Monte-Carlo.

2.1.2.2 Méthode quasi-Monte-Carlo

La méthode de Monte-Carlo est une méthode particulièrement robuste puisque sa précision ne dépend pas de la dimension du problème traité [15]. Par exemple, un calcul d'intégrale par méthode de Monte-Carlo converge à un taux $O(N^{1/2})$, qui est indépendant de la dimension de l'intégrale. La contrepartie de cette robustesse est la lenteur à laquelle la méthode de Monte-Carlo va converger, puisqu'en effet une augmentation d'un facteur 4 du nombre de calculs effectués n'apporte qu'une amélioration d'un facteur 2 en précision.
Une des alternatives permettant d'optimiser le nombre de simulations à calculer, donc le temps de calcul, sans sacrifier la précision consiste à jouer sur la distribution des nombres aléatoires. Au lieu d'utiliser des séquences de nombres aléatoires ou pseudo-aléatoire, les méthodes de type quasi-Monte-Carlo reposent sur une alternative déterministe : les séquences de nombres quasi-aléatoires. Ces séquences ont pour objectif d'engendrer des séquences de nombres uniformément répartis sur un espace donné afin d'éliminer les phénomènes d'agglomérats de points que l'on peut observer avec les générateurs de nombres pseudo-aléatoires (cf. Figure 2.1) [15]. Parmi les séquences de nombres le plus couramment utilisés lors d'analyse quasi-Monte-Carlo citons les séquence de Sobol [77], de Halton [34] ou bien encore de Faure [29].

Contrairement à la méthode de Monte-Carlo classique qui mesure l'erreur effectuée en remplaçant (2.1) par (2.3) par le calcul de l'erreur quadratique moyenne, on ne peut mesurer directement cette erreur avec la méthode quasi-Monte-Carlo. À la place on va borner l'erreur faite lors de cette approximation par l'inégalité de Koksma-Hlawka [15] :
Soit l'intégrale :

$$I[f] = \int_{I^a} f(x)dx, \tag{2.7}$$

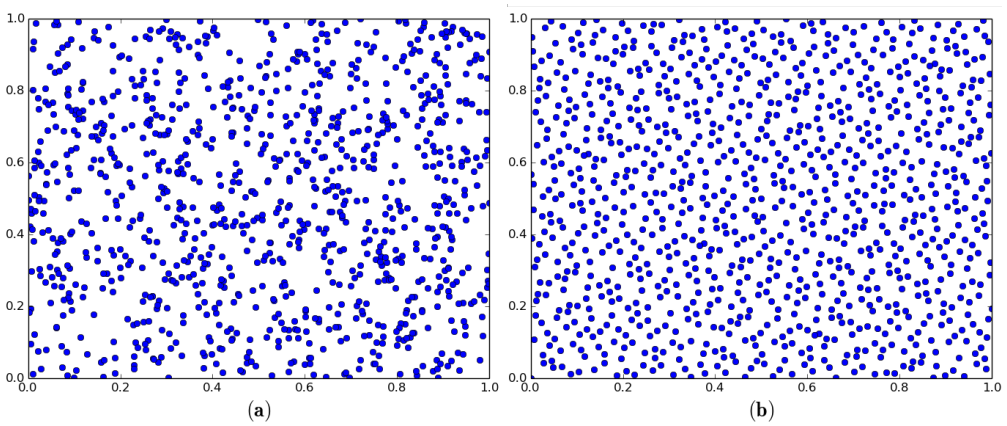

FIGURE 2.1: Projection 2D de 1000 points engendrés dans l'intervalle [0,1][0,1] par une séquence (a) pseudo aléatoires, (b) de Sobol (2,3)

Et son approximation par la méthode quasi-Monte-Carlo :

$$I_N[f] = \frac{1}{N}\sum_{n=1}^{N} f(x_n), \tag{2.8}$$

On définit alors la quadrature de l'erreur commise en substituant (2.7) par (2.8) par :

$$\epsilon[f] = |I[f] - I_N[f]|, \tag{2.9}$$

Soit $V[f]$ la variation de la fonction f, à plusieurs dimensions au sens de Hardy-Krause :

$$V[f] = \int_{I^a} |\frac{\partial^d f}{\partial t_1 ... \partial t_d}| dt_1 ... dt_d + \sum_{i=1}^{d} V[f^i], \tag{2.10}$$

On peut alors définir l'inégalité de Koksma-Hlawka permettant de borner l'erreur d'intégration $\epsilon[f]$ commise par la méthode pseudo-Monté-Carlo pour une fonction f a variation bornée et une séquence de nombres x_n donnée :

$$\epsilon[f] \leqslant V[f] D_N^*, \tag{2.11}$$

avec D_N^* la discrépance de x_n :

$$D_N^* = \sup_{J \subset [0,1]} \frac{\#x_n \in J}{N} - m(J), \tag{2.12}$$

où $m(J)$ est le volume du sous ensemble J rectangulaire.
À noter que l'erreur commise par l'utilisation d'une séquence pseudo-aléatoire est plus faible que celle effectuée lors de l'utilisation d'une méthode classique de Monte-Carlo. À noter aussi que le taux de convergence de la méthode quasi-Monte-Carlo est $O((logN)^k N^{-1})$,

où k est la dimension du problème.
Toutefois de par l'utilisation des séquences de nombre quasi-aléatoires, il n'existe pas de base théoriques permettant d'estimer la précision de la méthode quasi-Monte-Carlo. En d'autre terme il n'existe pas d'outils mathématique permettant d'évaluer si l'analyse a convergé contrairement à la méthode de Monte-Carlo classique où l'on peut appliquer le théorème de la limite centrale. Ensuite de par les corrélations existantes entre les point (pour assurer l'uniformité de la distribution) les méthodes quasi-Monte-Carlo sont faites pour des calculs d'intégrales et ne sont pas directement applicables à l'analyse de simulations numériques tels que celles auxquelles nous nous intéressons dans le cadre de ces travaux [15].
Bien que le taux de convergence de la méthode quasi-Monte-Carlo soit plus rapide que celui de la méthode classique, notons que cela ne signifie pas que l'on obtiendra une meilleure précision sur la détermination du résultat, par rapport à la méthode classique, à cause des corrélations entre les valeurs aléatoires. Enfin notons que si k devient trop grand, le taux de convergence de la méthode quasi-Monte-Carlo sera alors plus lent que celui de la méthode de Monte-Carlo classique.

2.1.3 Mise en œuvre pratique de la méthode de Monte-Carlo

2.1.3.1 Mise en œuvre

Selon le domaine d'application et selon le problème traité on trouve un large éventail de méthodes de Monte-Carlo et d'algorithmes différents (Monte-Carlo, quasi-Monte-Carlo, Monte-Carlo par Chaîne de Markov, algorithme de Metropolis-Hasting, etc...). Toutefois toutes ces méthodes ont en commun de suivre le même processus lors de leurs mises en œuvre [17] :

- Définition du domaine des conditions initiales : pour chaque variable aléatoire X_j du modèle on définit l'ensemble des valeurs x_i qui lui sont accessibles ainsi que la loi de distribution $P_{X_j}(x_i)$ qui lui est associée.
- Génération aléatoire de N jeux de conditions initiales à partir des informations définies à l'étape précédente.
- Calcul des N simulations à partir des jeux de conditions initiales définis précédemment : le calcul de chaque simulation est alors déterministe.
- Agrégation des résultats des N simulations et détermination de la moyenne empirique et des autres propriétés statistiques de la grandeur que l'on souhaite analyser.

Dans le cadre des travaux présentés dans ce manuscrit, lorsqu'il sera question de méthode de Monte-Carlo, sauf indication contraire explicite, cela fera référence à la génération

de N jeux de conditions initiales, les valeurs pour chaque paramètre étant tirées via un générateur de nombre pseudo-aléatoire selon une loi de densité donnée (le plus souvent Gaussienne) de façon indépendante : c'est à dire soit deux variables X_A et X_B, la probabilité de tirer la valeur j pour X_B sachant que l'on a obtenu la valeur i pour X_A :

$$P(X_B = j|X_A = i) = P(X_B = j), \tag{2.13}$$

2.1.3.2 Exploitation des résultats

Nous allons faire ici un inventaire non exhaustif des outils à notre disposition pour extraire les résultats d'une analyse de Monte-Carlo.

Convergence de la méthode de Monte-Carlo Lors d'une analyse de Monte-Carlo il est impératif de s'assurer de la convergence des résultats de l'analyse avant de procéder à leur exploitation.
On dit qu'il y a convergence d'une analyse statistique par la méthode de Monte-Carlo lorsqu'on a effectué suffisamment de tirages aléatoires (que N est suffisamment grand) pour que (2.6) soit vérifié. Concrètement la convergence d'une analyse de Monte-Carlo garantit que deux analyses de Monte-Carlo consécutives avec des graines différentes appliquées à un même problème donneront des résultats identiques à l'erreur près. Or il n'est pas aisé a priori de savoir à partir de quelle valeur N est suffisamment grand sachant que cela varie pour chaque modèle, en fonction du nombre de paramètres perturbés et de leurs lois de distributions. La tâche est d'autant moins aisée que l'on ne connaît pas a priori la loi de distribution que suivent les résultats des simulations.
Il existe une méthode permettant d'estimer N à partir des résultats d'un ensemble de simulations aléatoires. D'après le théorème de la limite centrale on a pour une distribution f et N suffisamment grand :

$$\epsilon_N[f] \approx \sigma[f]N^{-1/2}v, \tag{2.14}$$

avec v une variable aléatoire normale, $\sigma[f]$ l'écart type de f et $\epsilon_N[f]$ l'erreur.
En prenant la réciproque du théorème de la limite centrale on peut réécrire (2.14) pour obtenir la valeur N à partir de laquelle on a une certaine valeur d'erreur avec un niveau de confiance c donné :

$$N = \epsilon^2\sigma^2 s(c), \tag{2.15}$$

avec s la fonction de confiance (qui est une fonction tabulée) pour une variable normale telle que :

$$c = erf(\frac{s(c)}{\sqrt{(2)}}), \tag{2.16}$$

Typiquement pour un niveau de confiance de 95%, s vaut 2.
L'écart type de la distribution des résultats n'étant pas connu il est possible de l'estimer à partir des résultats d'un échantillon de M simulations aléatoires [70],[15]. On calcule ainsi M simulations générées aléatoirement à partir des propriétés statistique de l'analyse de Monte-Carlo que l'on souhaite effectuer. On a alors l'erreur quadratique moyenne théorique $\widetilde{\epsilon}_M$:

$$\widetilde{\epsilon}_M = (\frac{1}{M}\sum_{j=1}^{M}(f(x_j) - \overline{f}_N)^2)^{1/2}, \tag{2.17}$$

Ainsi on définit la variance empirique moyenne $\widetilde{\sigma}_M$:

$$\widetilde{\sigma}_M = \sqrt{M}\widetilde{\epsilon}_M, \tag{2.18}$$

On fait alors l'approximation que $\sigma \approx \widetilde{\sigma}_M$ d'où :

$$N \approx \epsilon^{-2} M \epsilon_M^2 s(c), \tag{2.19}$$

Détermination des moments statistiques D'aprés sa définition même, une analyse par la méthode de Monte-Carlo nous permet de déterminer la valeur moyenne d'un résultat d'un modèle à conditions initiales perturbées, sans connaître sa loi de distribution théorique. De la même façon il est aussi possible de définir les autres moments statistiques de la distribution du résultat étudié :

- Sa variance $V(X)$, moment d'ordre 2 : permet de caractériser la dispersion des résultats autour de la valeur moyenne. Elle se calcule comme suit :

$$V(X) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{h}_N)^2, \tag{2.20}$$

  avec x_i le résultat de la i-ème simulation, $\overline{h}_N$ la valeur moyenne des résultats, et N le nombre total de simulation. Grâce à la variance on peut calculer l'écart type σ_X de la distribution :

$$\sigma_X = \sqrt{V(X)}, \tag{2.21}$$

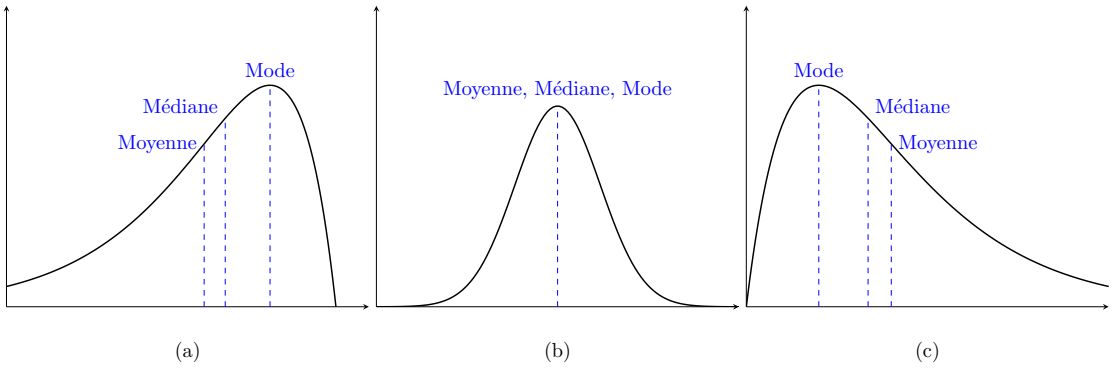

FIGURE 2.2: Allure des distributions en fonction du signe du skewness : (a) négatif; (b) nul; (c) positif

- Son coefficient d'asymétrie ou "skewness" γ, moment d'ordre 3 : permet de caractériser l'asymétrie de la distribution par rapport à une distribution gaussienne de référence centrée sur la moyenne de la distribution. Il se calcule comme suit :

$$\gamma_X = \frac{1}{N}\sum_{i=1}^{N}(\frac{x_i - \overline{h}_N}{\sigma_X})^3, \tag{2.22}$$

  Lorsque le skewness est négatif, la queue de la distribution est plus étalée vers la gauche de la moyenne, et lorsqu'il est positif, la queue de la distribution est plus étalée vers la droite de la moyenne (cf. Figure 2.2). En d'autres termes, une distribution avec un skewness négatif a une moyenne qui est à gauche de la valeur médiane, tandis qu'une distribution avec un skewness positif a se moyenne décalée à droite de sa médiane.

- Le coefficient d'aplatissement de Pearson ou "kurtosis" β, moment d'ordre 4 : permet de caractériser l'aplatissement de la distribution. Il se calcule comme suit :

$$\beta_X = \frac{1}{N}\sum_{i=1}^{N}(\frac{x_i - \overline{h}_N}{\sigma_X})^4, \tag{2.23}$$

  Un kurtosis de 3 indique que la distribution est Normale. Si il est inférieur à 3 on dit que la distribution est platykurtic, soit plus "plate" qu'une distribution Normale (exemple, une distribution uniforme, type créneau, ou une loi du demi-cercle). Enfin s'il est supérieur à 3 on dit que la distribution est leptokurtic, soit plus "piquée" qu'une distribution Normale (exemple distribution de Laplace). La Figure 2.3, présente l'allure d'une distribution de Laplace (leptokurtic), normale et du demi-cercle (platykurtic). Dans certains ouvrages on trouve aussi le kurtosis

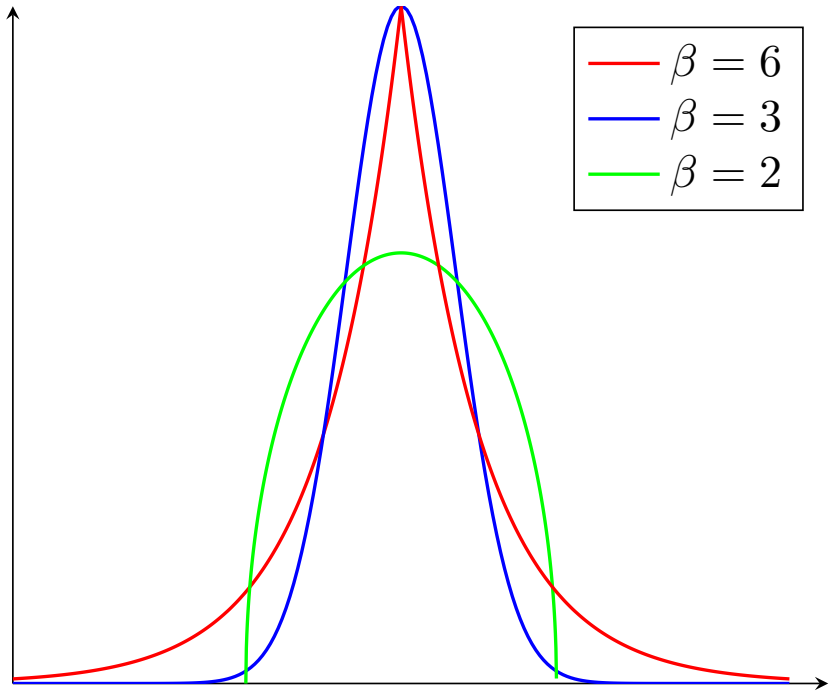

FIGURE 2.3: Distributions avec différents kurtosis : de Laplace (rouge), normale (bleu), du demi-cercle (vert)

normalisé qui vaut 0 pour une distribution normale, est négatif pour un distribution patykurtic et est positif pour une distribution leptokurtic.

Détermination des zones de confiances À partir du calcul de la variance on peut définir les intervalles de confiance à 1, 2 ou 3 σ, soit l'erreur commise en remplaçant (2.1) par (2.3) pour un nombre N de simulations donné, c'est à dire la probabilité que la valeur moyenne théorique se trouve bien dans les intervalles définies respectivement à 68, 95 et 99, 7%. Mais on peut aller plus loin, en définissant des ellipses de zones de confiance lorsque l'on considère la distribution couplée de deux variables. Ceci est particulièrement intéressant lorsque l'on cherche par exemple à déterminer la zone de retombée d'un débris de satellite [27].

Pour ce faire on se réfère aux travaux de [35] et de [71] nous permettant de déterminer les paramètres d'une ellipse de confiance pour une distribution de résultats quelconque en 2 dimensions dans le plan de mesure ($\alpha = 0$).
Ainsi considérons N points tels que chaque point i a pour coordonnées le couple (x_i, y_i). On définit :

$$\overline{x} = \frac{\sum_{i=1}^{N} x_i}{N}, \tag{2.24}$$

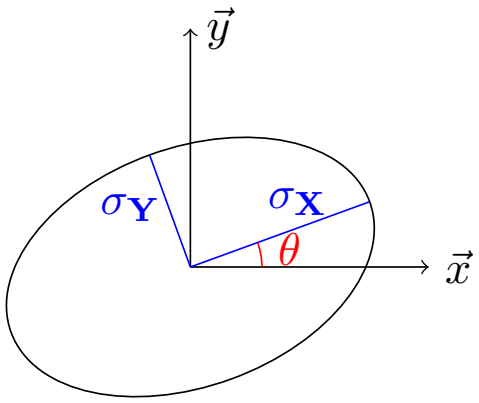

FIGURE 2.4: Paramètres de l'ellipse de confiance

La position moyenne des points projetés sur l'axe horizontal.

$$\overline{y} = \frac{\sum_{i=1}^{N} y_i}{N}, \tag{2.25}$$

La position moyenne des points projetés sur l'axe vertical.

$$\sigma_x^2 = \frac{\sum_{i=1}^{N} (x_i - \overline{x})^2}{N-1}, \tag{2.26}$$

La variance des points sur l'axe horizontal.

$$\sigma_y^2 = \frac{\sum_{i=1}^{N} (y_i - \overline{y})^2}{N-1}, \tag{2.27}$$

La variance des points sur l'axe vertical.
On définit alors r le coefficient de corrélation entre la distribution des points selon l'axe vertical et horizontal tel que :

$$r = \frac{\sum_{i=1}^{N} (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{N} (x_i - \overline{x})^2 \sum_{i=1}^{N} (y_i - \overline{y})^2}}, \tag{2.28}$$

Pour calculer σ_X et σ_Y respectivement le demi-grand axe et le demi-petit axe de l'ellipse de confiance on définit d'abord :

$$A = \sigma_x^2 + 2 * r * \sigma_x * \sigma_y + \sigma_y^2, \tag{2.29}$$

$$B = 2 * (1 - r^2)^{1/2} * \sigma_x * \sigma_y, \tag{2.30}$$

Ainsi on peut calculer les paramètres de ellipse de confiance à 1 sigma :

$$\sigma_X = \sqrt{0,5 * [A + (A^2 - B^2)^{1/2}]}, \tag{2.31}$$

$$\sigma_Y = \sqrt{0,5 * [A - (A^2 - B^2)^{1/2}]}, \tag{2.32}$$

Trivialement on obtient les paramètres des ellipses de confiance à 2 et 3 σ : $2 * \sigma_{X/Y}$ et $3 * \sigma_{X/Y}$ respectivement.
On peut ensuite définir θ , l'angle entre le demi-grand axe σ_X et l'axe horizontal. Soit D tel que :

$$D = -r * \sigma_x * \sigma_y, \tag{2.33}$$

D'où :

$$\theta = \arctan(\frac{2D}{\sigma_x^2 - \sigma_y^2 * \sqrt{(\sigma_x^2 - \sigma_y^2)^2 + 4D^2}}), \tag{2.34}$$

2.1.3.3 Limitation

Comme nous venons de le voir la principale limitation de la méthode de Monte-Carlo vient du temps passé à calculer les N simulations qui sont engendrées et ce malgré les facilités de calculs apportées par les ordinateurs modernes : fréquence d'horloge élevée, possibilité de parallélisation des calculs, vectorisation, etc...
Afin de contourner ce problème il existe plusieurs alternatives visant à optimiser les calculs effectués :
Une première alternative réside dans l'optimisation de la méthode de génération des nombres aléatoires. On l'a vu à la sous-section 2.1.2.2 c'est ce que fait la méthode quasi-Monte-Carlo. Toutefois, comme expliqué dans cette sous-section, le caractère déterministe des séquences de nombres qui sont engendrées et les corrélations qui sont imposées entre ces nombres font que cette solution n'est pas applicable directement à l'analyse de simulations numériques, celle-ci étant conçue à l'origine pour le calcul

d'intégrale.
Une autre solution consiste à optimiser l'exploration des conditions initiales en ne considérant qu'un ensemble de simulations fournissant un résultat "représentatif". C'est ce que permettent de faire les méthodes des plans d'expérience. Ces méthodes proposent des critères permettant de choisir pour chaque variable aléatoire 2 ou 3 valeurs dites représentatives. Une fois ces valeurs sélectionnées chaque méthode propose ensuite une table qui permet de générer un nombre restreint de simulations correspondant à des combinaisons uniques des valeurs représentatives. Le grand intérêt de ces méthodes de plan d'expérience est de ne générer qu'un nombre limité de simulations dont le nombre est proportionnel au nombre de paramètres. En revanche l'inconvénient de ces méthodes est de ne fournir que des informations limitées par rapport à ce que peut fournir une analyse de Monte-Carlo. Par exemple on ne peut pas effectuer de recherche d'optimum avec ces méthodes qui soient aussi précise que ce que l'on obtient avec une analyse de Monte-Carlo et un test de χ^2. À titre d'illustration on peut citer deux méthodes de plan d'expérience :

- La méthode de Plackett-Burman : elle permet d'estimer quelles variables ne sont pas négligeables lors de la mesure d'un résultat. Cette méthode utilise comme valeurs représentatives la valeur nominale de la variable et un extremum de sa distribution. Afin de définir les combinaisons de valeurs des variables, la méthode de Plackett-Burmann utilise les matrices d'Hadamar [62].

- La méthode de Box-Behnken : elle génére une séquence de plan d'expérience permettant de définir la réponse de surface avec un polynome du second degré permettant d'obtenir la réponse optimale d'un système. Cette méthode utilise de 3 à 16 valeurs représentatives par variables, et selon le système utilise une table conçue spécialement pour le système [12].

Une autre solution consiste à réduire le nombre de paramètres que l'on perturbe afin de ne considérer que ceux ayant une influence notable sur la dispersion du résultat que l'on souhaite analyser afin d'obtenir une convergence plus rapide du système. Pour ce faire on peut utiliser les méthodes des plans d'expérience, la méthode de Plackett-Burman évoquée plus haut pouvant convenir par exemple.
Dans le cadre des travaux présentés ici nous avons pris le parti de chercher à optimiser nos simulations de Monte-Carlo en réduisant le nombre de paramètres à perturber pour ne garder que les paramètres ayant une influence notable sur la dispersion des résultats à analyser. Pour ce faire nous n'utilisons pas la méthode de Plackett-Burman, mais la méthode de Taguchi qui fournit tout un arsenal d'outils permettant d'automatiser la détection des paramètres d'influence et de fournir un niveau de confiance sur le résultat obtenu.

2.2 Méthode de Taguchi

En 1950 Genichi Taguchi, alors membre de l'Institut des Mathématiques Statistiques du Ministère de l'Éducation Japonais, prend la la tête de la section du développement de produits de télécommunication au Electric Communications Laboratory de la compagnie de téléphonie japonaise : Nippon Telegraph and Telephone (NTT). Il est alors chargé de trouver un moyen de produire des produits d'excellente qualité malgré la situation du pays (manque de matériaux de qualité, d'infrastructures de pointe et d'ingénieurs qualifiés). De plus la direction d'ECL décide d'entrer en compétition avec le Bell Laboratories pour le développement d'un système de communication crossbar. Durant les six années que dura la compétition entre les deux équipes Taguchi mis en pratique un ensemble de processus de production et d'optimisation posant ainsi les bases de ce qui est connu aujourd'hui sous l'appellation de "Robust Design". Parmi cet ensemble d'outil et de méthodes figure une méthode d'analyse statistique se basant sur le concept de "Conception de l'Experience" (Design of Experiment, DE) développé par Ronald A. Fisher dans les années 1920 [81]. Cette méthode appelée méthode de Taguchi se différencie des autres méthodes de plan d'expérience par les trois points suivants [31] :

- l'identification d'un ensemble de Tables Orthogonales ou Orthogonal Array (OA) à utiliser et qui s'appliquent à de nombreuses situations.
- une méthode standard pour l'analyse des résultats : l'Analyse de la Variance ou Analysis of Variance (ANOVA).
- un outil graphique appelé "graphe linéaire" permettant de représenter l'interaction entre les paires de colonnes.

Dans le présent chapitre nous allons ainsi détailler les différents éléments de la méthode de Taguchi. Dans la première sous-section nous présentons les Tables Orthogonales. Dans la seconde sous-section nous détaillons la méthode de l'Analyse de la Variance. Dans la troisième sous-section nous présentons les "graphes linéaires". Enfin dans la dernière sous-section nous présentons une des nombreuses applications de la méthode de Taguchi, à savoir son utilisation pour réduire le nombre de dimensions d'un système lors de la préparation d'une analyse par la méthode de Monte-Carlo.

2.2.1 Introduction aux Tables Orthogonales

Le principe de la Conception de l'Expérience a été développé par Sir Ronald Fisher en Angleterre dans les années 1920 pour la rationalisation de l'expérimentation agricole. Le but de la Conception de l'Expérience est de déterminer une relation entre les valeurs possibles d'un paramètre d'un modèle et la réponse de ce dernier. Dans le cas où l'on considère plusieurs paramètres, on parle alors de plan factoriel d'expérience ou de

TABLE 2.1: Expérience Factorielle Complète avec deux paramètres à deux niveaux

Expérience	A	B
1	1	1
2	1	2
3	2	1
4	2	2

matrice de plan d'expérience. Le grand intérêt de la méthode de Taguchi est d'étendre le concept de la méthode du plan factoriel d'expérience pour considérer les incertitudes sur les paramètres du modèles en se basant sur l'utilisation des Tables Orthogonales.

2.2.1.1 Expérience Factorielle

Soit un modèle avec un nombre de paramètres initiaux donnés, chacun pouvant prendre un certain nombre de valeurs. Pour chaque paramètre initial, on associe chacune des valeurs possibles à un niveau dans la table d'expérience factorielle. Par la suite on identifie les paramètres initiaux en fonction du nombre de valeurs accessibles. Ainsi pour un paramètre pouvant prendre 3 valeurs différentes on parlera de paramètre à 3 niveaux. L'expérience factorielle complète consiste alors à considérer toutes les combinaisons possibles de niveaux entre ces paramètres initiaux.
Comme on peut le voir dans le Tableau 2.1, avec deux paramètres à deux niveaux on obtient 4 combinaisons de paramètres (4 expériences différentes). Si l'on considère à présent 3 paramètres à 2 niveaux on passe à 8 combinaisons (2^3) et ainsi de suite, le nombre de combinaisons possibles étant ainsi exponentiel avec le nombre de paramètres considérés. Se pose alors le problème de l'exploration de ces combinaisons. Dans le cas de modèles numériques complexes, où la résolution d'une seule combinaison de paramètres peut s'avérer coûteuse en terme de temps de calcul, on comprend aisément qu'il peut s'avérer très compliqué d'explorer l'ensemble des combinaisons possibles. La solution consiste alors à sélectionner un nombre limité d'expériences en essayant de produire un maximum d'information. On parle alors d'Expérience Factorielle Fractionnelle. Il existe de nombreux tests pour construire une Expérience Factorielle Fractionnnelle. Le lecteur intéréssé par le sujet pourra se référer à [73] pour une présentation détaillée d'un certain nombre d'entre elles.
Taguchi [80] a basé sa méthode d'Expérience Factorielle Fractionnelle sur l'utilisation des Tables Orthogonales. L'intéret majeur des OA est de permettre d'évaluer plusieurs paramètres avec un nombre restreint de simulations. Ainsi par exemple pour un système avec 13 paramètres ayant chacun 3 niveaux possibles il faut résoudre 3^{13} simulations avec la méthode de l'Expérience Factorielle Complète, contre seulement 27 simulations avec la méthode de l'Expérience Factorielle Partielle de Taguchi (qu'on appellera par la suite méthode de Taguchi).

TABLE 2.2: Table Orthogonale Standard L_9

Exériences	A	B	C	D
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

2.2.1.2 Les Tables Orthogonales et le principe d'orthogonalité

Les tables orthogonales sont des matrices construites à partir des Carrés Latins, et qui sont définies par 3 paramètres :

- Le nombre de niveaux : le nombre de valeurs que peut prendre un élément de la table.
- Le nombre de paramètres : le nombre de colonnes de la table, chaque paramètre étant assigné à une colonne.
- Le nombre d'observations : le nombre de ligne dans la table, chaque ligne correspondant à une expérience (c'est à dire une combinaison de paramètres initiaux).

Une table orthogonale est nommée comme suit : L_X ou L_{X_Y}, X étant le nombre d'observations et Y le nombre de niveaux de la table.
Taguchi a ainsi tabulés 18 Tables Orthogonales standards (cf. [80] et [61]), telle que L_9 présentée Table 2.2, qui est une table orthogonale à 3 niveaux, pour 4 paramètres (ici identifiés A, B C et D) et qui fourni les indications pour engendrer 9 simulations orthogonales.
Dans le cas des tables orthogonales le terme d'orthogonalité doit se comprendre au sens d'orthogonalité combinatoire. Pour qu'un ensemble de combinaisons soit considéré comme orthogonal il faut que pour chaque combinaison de paire de paramètres, toutes les combinaisons de niveaux des paramètres apparaissent le même nombre de fois. On appelle aussi cette orthogonalité la propriété d'équilibrage [61].
Illustrons cette propriété à partir d'un exemple : soit 3 paramètres A, B et C ayant 3 niveaux chacun. On peut assigner chacun de ces paramètres à une colonne de la Table Orthogonale L_9 : A à la première colonne, B à la seconde et C à a troisième.
Pour chaque combinaison qui figure dans la table orthogonale on la marque sur la Figure 2.5 par un point. On obtient ainsi neuf points, un par ligne de la table. En observant cette figure on constate que chaque plan contient alors exactement trois points.

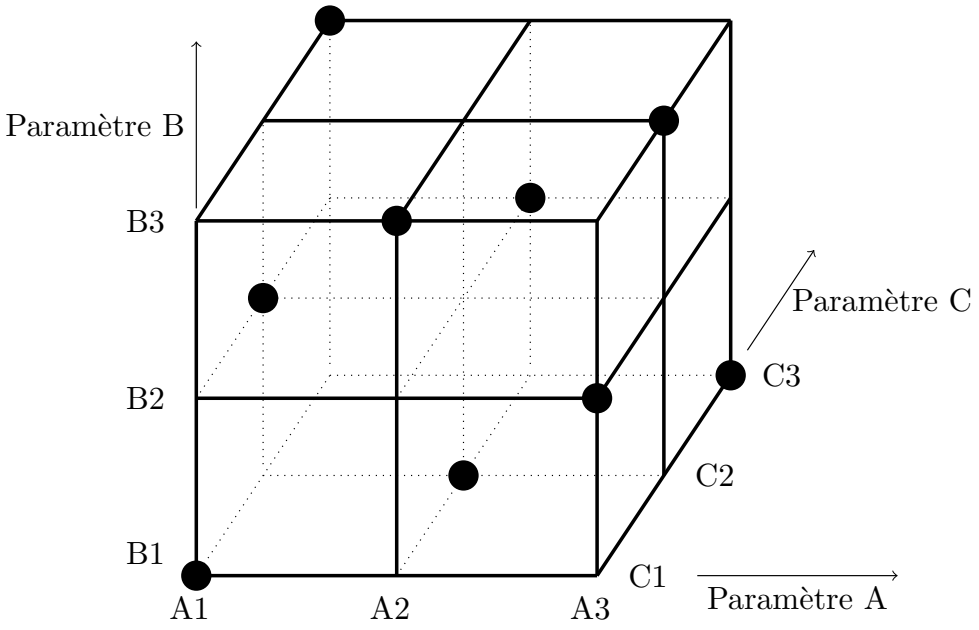

FIGURE 2.5: Illustration du principe d'orthogonalité dans les Tables Orthogonales

Grâce à la propriété d'orthogonalité, on obtient ainsi le nombre minimal de combinaisons nécessaires pour trois paramètres à trois niveaux pour obtenir l'information concernant l'influence de ces paramètres sur la variation du résultat [31]. On vérifie aussi que cette propriété d'orthogonalité ne dépend pas non plus du nombre de colonnes considérées. Ainsi on n'est pas obligé d'avoir quatre paramètres dans notre modèle pour utiliser la table L_9, on peut avoir uniquement deux ou trois paramètres a étudier. La seule restriction qui s'applique est l'utilisation de toutes les lignes de la table orthogonale sans quoi l'orthogonalité ne serait pas respectée [61].
Les Tables Orthogonales sont déjà établies et le lecteur pourra en trouver quelques unes en Annexe A et d'autres dans [81]. De plus il pourra trouver une méthode pour engendrer automatiquement des tables à trois niveaux à partir des deux carrés latins à trois niveaux dans [31].

2.2.1.3 Utilisation des tables orthogonales dans le cadre de la méthode de Taguchi

Afin de construire le plan d'expérience factorielle, la méthode de Taguchi impose d'identifier dans un premier temps le nombre de paramètres que nous souhaitons étudier afin de sélectionner la table orthogonale la plus appropriée pour l'étude, c'est à dire celle qui permet de faire le moins de simulations pour le nombre de paramètres à étudier souhaité. Pour cette étape il faut aussi choisir le nombre de niveaux à considérer pour chaque paramètre. Le choix d'utiliser des paramètres à deux niveaux permet d'obtenir moins de simulations à calculer. L'utilisation de paramètres à trois niveaux permet d'estimer la variance du résultat avec plus de précision.

Une fois cette première étape effectuée, se pose ensuite la question de la détermination des valeurs représentatives et de l'association de chaque valeur représentative à un niveau pour chacun des paramètres traités.
Taguchi recommande que pour un paramètre avec une valeur moyenne μ et une tolérance δ autour de ce dernier, si l'on décide de ne considérer que deux niveaux, de prendre comme valeurs représentatives $\mu \pm \delta$ et de prendre μ et $\mu \pm \delta$ dans le cas où l'on considère un paramètre à trois niveaux.
Dans le cas d'un paramètre avec une incertitude normale, Phadke [61] définie alors les valeurs de δ à utiliser afin d'optimiser l'exploration de l'espace des paramètres initiaux :

- pour une variable avec une distribution gaussienne à deux niveaux de moyenne μ et d'écart type σ, on prend $\delta = \sigma$. On obtient ainsi les deux valeurs suivantes pour nos deux niveaux : $\mu + \sigma$ et $\mu - \sigma$.

- pour une variable avec une distribution gaussienne à trois niveaux de moyenne μ et d'écart type σ, on prend $\delta = \sqrt{3/2}\sigma$. On obtient alors les trois valeurs suivantes pour nos trois niveaux :$\mu - \sqrt{3/2}\sigma$; μ et $\mu + \sqrt{3/2}\sigma$.

L'attribution de ces valeurs à un niveau donné n'est soumise à aucune règle, si ce n'est que cette attribution doit rester cohérente entre les paramètre d'une même étude. De façon générale il est d'usage d'attribuer au niveau le plus bas la valeur associé $\mu - \delta$ et au niveau le plus élevé $\mu + \delta$.
Une fois les valeurs associées aux niveaux il faut ensuite attribuer chaque paramètre à une colonne de la table orthogonale. La méthode de Taguchi est robuste à l'attribution des colonnes. Autrement dit la conclusion de la méthode de Taguchi sera toujours la même, même si on modifie l'ordre d'attribution des paramètres, ou si l'on a des colonnes vides [80].
Une fois que l'on a effectué ces 5 étapes : identification des paramètres, choix du nombre de niveaux, choix de la Table Orthogonale, calcul des valeurs représentatives et assignation de chaque paramètre à une colonne de la Table Orthogonale, on peut alors effectuer les simulations à partir des informations contenues dans chaque ligne de la table, et ensuite les analyser à partir de la méthode d'Analyse de la Variance.

2.2.2 Introduction à l'Analyse de la Variance

Comme nous venons de le voir la méthode de Taguchi se base sur exploration partielle de l'espace des paramètres grâce à l'utilisation des Tables Orthogonales. Du fait de cette exploration partielle, l'analyse des résultats ne doit pas se contenter de fournir une analyse statistique du résultat mais doit aussi inclure une évaluation du niveau de confiance que l'on peut avoir dans le résultat obtenu. C'est exactement ce que permet de faire la méthode de l'Analyse de la Variance (Analysis of Variance, ANOVA). Cette

technique permet de déterminer la variabilité d'un résultat, comment cette variabilité est influencée par les fluctuations des paramètres du modèle, et quel niveau de confiance on peut attribuer au résultat précédent.
La méthode ANOVA nécessite de calculer diverses grandeurs qui sont ensuite organisées sous forme de tableaux standards : les Tables d'Analyse de la Variance. Avant de détailler les différentes grandeurs calculées pour une ANOVA, voici un résumé des notations qui employées dans la suite de cette sous-section :

- SS = Somme des Carrés (Square Sum)
- SS' = Pure Somme des Carrés (Pure Square Sum)
- f = Degrés de Liberté
- V = Variance
- F = Ratio de la Variance
- P = Pourcentage de contribution d'un paramètre
- CF = Facteur de Correction
- N = Nombre d'expériences

2.2.2.1 Facteur de correction

Le facteur de correction est le carré de la somme des résultats divisé par le nombre total d'expériences.

$$CF = \frac{(\sum_{i=1}^{N} y_i)^2}{N}, \tag{2.35}$$

avec y_i le résultat de la i-ème simulation et N le nombre total de simulations.

2.2.2.2 Somme des Carrés

La somme des carrés permet de mesurer la déviation de la valeur moyenne d'une donnée. Plus la somme des carrés pour un paramètre donné est proche de la somme des carrés totale, plus le paramètre considéré aura de l'influence sur la variation du résultat.
Dans le cadre de l'Analyse de la variance appliqué à la méthode de Taguchi on définit les trois types de somme des carrés suivants :

La somme des carrés totale

$$SS_T = \sum_{i=1}^{N} y_i^2 - CF, \tag{2.36}$$

avec y_i le résultat de la i-ème simulation, N le nombre total de simulations, et CF le facteur de correction.

La somme des carrés d'un paramètre Soit A un des paramètres d'entré du modèle qui est perturbé par la méthode de Taguchi. On définit la somme des carrés du paramètre A tel que :

$$SS_A = \frac{\sum_{i=1}^{k_A} A_i^2}{n_A} - CF, \tag{2.37}$$

avec k_A le nombre de valeurs différentes que peut prendre le paramètre A (dans le cadre de la méthode de Taguchi généralement 3), A_i la somme des résultats obtenus en utilisant la i-ème valeur du paramètre A et n_A le nombre de simulations effectuées avec une valeur donnée du paramètre A. Ce coefficient dépend de la table orthogonale utilisée mais pour une table donnée est identique d'une valeur à l'autre.

La somme des carrés de l'erreur La somme des carrés de l'erreur quand à elle est calculée en soustrayant à la somme des carrés totale, la somme des carrés de tous les paramètres.

$$SS_e = SS_T - \sum_{i=1}^{M} SS_i, \tag{2.38}$$

avec M le nombre de paramètres d'entrée du modèle qui ont été utilisés par la méthode de Taguchi.

2.2.2.3 Degrés de liberté

En statistique les degrés de liberté sont associés à chaque partie d'information qui peut être estimée à partir de la donnée. Dans le cadre de l'analyse de la variance on distingue trois types de degrés de liberté :

- degrés de liberté total : il s'agit du nombre de degrés de liberté d'un résultat du modèle. Il est associé au nombre de simulations indépendantes définies par la matrice d'expérience. On le calcule via l'équation suivante :
  $f_T = N - 1$; N étant le nombre total de simulations dans le cas de la méthode de Taguchi, puisque dans les tables orthogonales chaque simulation est indépendante.

- degrés de liberté du paramètre A : Soit A un des paramètres du modèle perturbé via la méthode de Taguchi. Son nombre de degrés de liberté est alors égal au nombre de niveaux accessibles par A (L_A) moins un, car les valeurs de chaque niveau dans la méthode de Taguchi sont liés entre elles [61] :
  $f_A = L_A - 1$; L_A étant généralement égal à trois dans le cadre de la méthode de Taguchi.

- degrés de liberté de l'erreur : de par l'orthogonalité de la matrice d'expérience dans le cadre de la méthode de Taguchi on a la définition suivante du degrés de liberté de l'erreur :
  $f_e = f_T - \sum_{i=1}^{M} f_i$; f_i étant le nombre de degrés de liberté du i-ème paramètre et M le nombre de paramètres considérés par la méthode de Taguchi.

2.2.2.4 Variance

On appelle variance V_A d'un paramètre A, la division de sa somme des carrés par son nombre de degrés de liberté. Ainsi :

$$V_i = \frac{SS_i}{f_i}, \tag{2.39}$$

i pouvant désigner soit un des paramètres du modèle, soit l'erreur.

2.2.2.5 Ratio de la Variance

Le ratio de la variance F d'un paramètre A est le rapport entre la variance de ce paramètre et la variance de l'erreur. Cette valeur est aussi appelée F-test en référence à Sir Ronald Fisher qui a inventé la méthode d'analyse de la variance.

$$F_i = V_i / V_e, \tag{2.40}$$

i pouvant être un des paramètre, ou l'erreur.
La distribution de ce paramètre suit une distribution statistique connue sous le nom de F-distribution. Ces distributions sont bien connues et tabulées [74], [78]. On peut ainsi utiliser ce paramètre pour le comparer avec les valeurs figurant dans les F-tables pour un niveau de confiance donné. Si la valeur de F calculée est supérieure à la valeur figurant dans la table pour le niveau de confiance donné, on pourra affirmer que le paramètre influe bien la variation du résultat avec le niveau de confiance testé. Les colonnes de la F-table correspondent au nombre de degrés de liberté du paramètre que l'on considère et les lignes correspondent au nombre de degrés de liberté du terme d'erreur.
Ceci n'est évidemment valable que pour les paramètres du modèle et non pour l'erreur, puisque son ratio de la variance est toujours égal à 1.

2.2.2.6 Pure somme des carrés

La pure somme des carrés d'un paramètre A est le résultat de la différence entre la somme des carrés de ce paramètre et le nombre de degrés de liberté de ce paramètre multiplié par la variance de l'erreur.

$$SS'_A = SS_A - f_A * V_e, \tag{2.41}$$

2.2.2.7 Contribution du paramètre

Le dernier paramètre de la table d'analyse de la variance est le Pourcentage de contribution du paramètre P. Il indique (en pourcentage) l'influence qu'a un paramètre sur la variation du résultat analysé, et permet de classer les différents paramètres de celui ayant le plus d'influence sur cette variation à celui qui en a le moins. À noter que certains auteurs [61] utilisent le paramètre F pour effectuer le même type de classification. On calcule P pour le paramètre i selon l'équation suivante :

$$P_i = 100 * SS'_i/SS_T, \tag{2.42}$$

2.2.2.8 Exploitation de l'Analyse de la Variance

Comme on vient de le voir l'Analyse de la Variance nécessite le calcul d'un grand nombre de paramètres. Cependant parmi ces paramètres seulement deux sont vraiment pertinents pour l'analyse des résultats : le ratio de la variance F et le pourcentage de contribution P.
Le paramètre P permet ainsi de discriminer les paramètres qui ont une influence notable sur la variation du résultat de ceux qui ont une influence négligeable et le paramètre F nous permet d'estimer le niveau de confiance que l'on a sur ce résultat.

2.2.3 Introduction aux graphes linéaires

Les graphes linéaires sont des outils graphiques dévellopés par Dr. Taguchi afin de faciliter l'assignation des interactions entre paramètres. De la même façon que les tables orthogonales, les graphes linéaires sont prédéfinis et propre à chaque table. Le lecteur pourra trouver les différents graphes linéaires dans les Annexes de [81], [74] et [61].
Dans ces graphes on assigne un paramètre à un point du graphe, tandis que le trait reliant deux points représente l'interaction entre les deux paramètres. La Figure 2.6 présente les graphes linéaires associés à la table orthogonale L_{27}. Considérons un modèle avec cinq paramètres : A, B, C, D et E. Si on alloue A à la colonne 1, B à la colonne 2, C à la colonne 5, D à la colonne 9 et E à la colonne 10, d'après le graphe linéaire présenté Figure 2.6 (a) on note que les interactions entre A et B pourront être analysé à partir des colonnes 3 et 4, les interactions entre A et C à partir des colonnes 6 et 7, tandis que

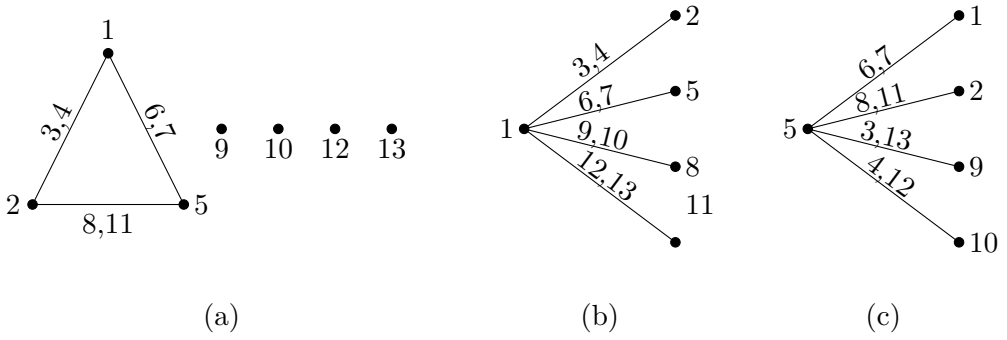

FIGURE 2.6: Graphes linéaires de la table orthogonale L_{27}

les interactions entre B et C à partir des colonnes 8 et 11. D et E quand à eux sont alloués sur des sommet déconnectés sur le graphe (respectivement colonnes 9 et 10) il n'est donc pas possible avec cette disposition d'analyser des interactions avec d'autres paramètres.

Enfin les graphes linéaires peuvent être utilisés afin de combiner des colonnes des tables orthogonales pour calculer de nouvelles tables avec plus de niveaux (mais moins de colonnes). Ce procédé sortant du cadre de l'utilisation des tables orthogonales dans ces travaux de thèse nous renvoyons le lecteur vers [81] pour plus de détails.

2.2.4 Application à l'étude d'une rentrée atmosphérique de satellite

2.2.4.1 Présentation du cas test

Voici à présent un exemple d'application de la méthode de Taguchi pour l'analyse de la rentrée atmosphérique d'un satellite simple. Dans le cadre de la modélisation de la rentrée atmosphérique d'un satellite un grand nombre de paramètres souffrent d'incertitudes et ils est particulièrement intéressant de pouvoir déterminer, parmi ces paramètres, quels sont ceux qui ont le plus d'influence sur la dispersion des résultat d'une simulation de rentrée atmosphérique. Les sources des incertitudes sur ces paramètres sont diverses. Dans le cas des propriétés des matériaux, celles-ci sont mal définies dans les conditions rencontrées durant la rentrée atmosphérique et ne prennent pas en compte les modifications dues à l'oxidation des matériaux [7]. Quand aux autres conditions initiales de la modélisation ces incertitudes viennent du fait que la rentrée est non contrôlée. Le présent cas a été défini en partenariat avec le CNES dans le cadre d'un contrat passé avec R.Tech [DBK-CT-LOG-0371-CNES]. Nous allons ici chercher à déterminer, parmi un ensemble de paramètres sur lesquels nous avons des incertitudes, quels sont ceux qui ont une grande influence sur le paramètre de survivabilité, P_S, qui est un paramètre permettant d'évaluer le taux d'ablation d'un objet en fonction de son altitude finale. Il vaut -1 dans le cas d'un objet arrivant intact au sol, et +1 dans le cas d'une ablation

TABLE 2.3: Conditions initiales pour l'analyse de Taguchi et leurs incertitudes associées

	Valeur nominale	1σ
Demi grand axe[m]	6488136.46	333.33
Argument du Périgée [°]	0.0	0.33
Angle d'inclinaison [°]	0.0	0.33
Ascencion Droite du Nœud Ascendant (Raan) [°]	0.0	0.33
Anomalie [°]	0.0	0.033
Epaisseur de la paroi de la sphère [mm]	2.0	0.167
Altitude de fragmentation [m]	78000.0	666.67
Température de naissance [K]	300.0	16.67
Densité du Cuivre $[kg/m^3]$	8930.0	297.67
Chaleur de Fusion du Cuivre [J/kg]	205363.0	6845.43
Température de Fusion du Cuivre [K]	1357.0	45.23
Cp du Cuivre [J/kg/K]	385.0	12.83
Emissivité du Cuivre	1.0	0.03

totale instantanée. Le paramètre de survivabilité est calculé via l'équation suivante :

$$P_S = \frac{Altitude_{finale}}{Altitude_{fragmentation}} - \frac{Masse_{Sol}}{Masse_{naissance}}, \tag{2.43}$$

Pour cet exemple les simulations sont effectuées en utilisant l'outil de référence du CNES : Debrisk.

On considère ainsi un satellite cubique de 300kg, de 1m de côté en Titane, sur une trajectoire d'inclinaison nulle avec une altitude initiale de 110 km. Le satellite contient une sphère creuse en Cuivre, de 0,25m de rayon extérieur. Dans le cadre d'une simulation Debrisk, on considère que le satellite cubique est totalement détruit lorsqu'il atteint l'altitude de fragmentation pour libérer la sphère. Dans notre analyse nous ne considérerons que l'état final de la sphère.

La Table 2.3 contient la liste des conditions initiales sur lesquelles nous avons une incertitude ainsi que la valeur nominale (moyenne) de ces conditions initiales et leur écart type (incertitude à 1 σ). Nous avons ainsi une liste de 13 paramètres (conditions initiales avec des incertitudes), et suivant les recommandations de [61], nous allons considérer 3 niveaux pour chaque paramètre. Lorsque l'on regarde les liste des tables orthogonales qui figure dans [81] on constate que la table L_{81} est la mieux adapté à notre problème, car étant la seule table à 3 niveaux disposant de suffisament de colones par rapport au nombre de paramètres à considérer.

La table L_{81}, est une table qui peut supporter jusqu'à 40 paramètres à 3 niveaux et fournissant les conditions initiales pour 81 simulations.

On associe les niveaux de la table aux valeurs des paramètres, selon les recommandations de Phadke [61] comme suit :

$1 \rightarrow$ valeur nominale $-\sqrt{3/2}\sigma$

TABLE 2.4: Table d'Analyse de la Variance pour le paramètre de survivabilité

	f	SS	V	F	SS'	P
Demi-grand axe	2	$1.022e-2$	$5.011e-3$	$3.949e+13$	$1.002e-2$	0.307
Argument du périgé	2	$1.868e-3$	$9.342e-4$	$7.363e+12$	$1.868e-3$	0.057
Angle d'inclinaison	2	$7.474e-3$	$3.737e-3$	$2.945e+13$	$7.474e-3$	0.229
Raan	2	$4.119e-3$	$2.059e-3$	$1.623e+13$	$4.118e-3$	0.126
Anomalie	2	$5.399e-4$	$2.7e-4$	$2.128e+12$	$5.4e-4$	0.017
Épaisseur de la paroi	2	$9.274e-2$	$4.637e-2$	$3.654e+14$	$9.274e-2$	2.844
Température initiale de la paroi	2	$1.148e-3$	$5.742e-4$	$4.526e+12$	$1.148e-3$	0.035
Altitude de fragmentation	2	$2.193e-2$	$1.097e-2$	$8.642e+13$	$2.193e-2$	0.672
Densité du cuivre	2	$1.61e-2$	$8.052e-3$	$6.346e+13$	$1.61e-2$	0.494
Chaleur de fusion	2	$1.243e-2$	$6.214e-3$	$4.897e+13$	$1.243e-2$	0.381
Température de fusion	2	2.907	1.454	$1.146e+16$	2.907	89.148
Cp du Cuivre	2	$1.418e-2$	$7.091e-3$	$5.588e+13$	$1.418e-2$	0.435
Emissivité du Cuivre	2	$1.713e-1$	$8.567e-2$	$6.752e+14$	$1.713e-1$	5.254
Erreur	56	$-7.105e-15$	$-1.268e-16$	1.0		

$2 \rightarrow$ valeur nominale

$3 \rightarrow$ valeur nominale $+\sqrt{3/2}\sigma$

Ensuite on associe chaque paramètre à une colonne. Comme vu précédemment l'ordre d'attribution des paramètres est sans conséquence et de la même façon il n'est pas obligatoire que toutes les colonnes de la table orthogonale soient attribuées [81], [61]. On attribue ainsi les paramètres dans l'ordre dans lequel ils sont présentés ci dessus : le demi-grand axe étant associé à la première colonne jusqu'à l'émissivité qui est associée à la 13ème colonne.

Chaque ligne nous fourni ainsi les conditions initiales à utiliser par paramètre pour une simulation. On effectue ensuite nos 81 simulations avec notre simulateur de rentrée atmosphérique de satellite (Debrisk), puis on analyse les résultats des 81 simulations.

2.2.4.2 Résultat et analyse

On obtient ainsi la table d'Analyse de la Variance pour le paramètre de survivabilité présenté Table 2.4. Une fois la table d'ANOVA créée il s'agit à présent d'en extraire l'information qui nous intéresse. Comme expliqué précédemment, on se concentre sur la colonne P qui nous donne la contribution, en pourcentage, de chaque paramètre à la dispersion du résultat.

Dans cet exemple on constate que le paramètre qui a le plus d'influence sur la dispersion du paramètre de survivabilité (c'est à dire qui a la plus grande contribution) est la température de fusion du Cuivre avec une contribution de 89%. De la même façon on observe que trois paramètres ont une influence négligeable sur la dispersion du paramètre de survivabilité, avec une contribution inférieure à 0.1% : l'argument du périgée , l'anomalie et la température initiale de la paroi. Lorsqu'on considère la F-table à 99.5% on trouve que pour $f_{P_S} = 2$ et $f_e = 56$ on a $F = 6.0664$. On constate que toutes les valeurs de F calculées dans la table d'ANOVA sont supérieures à cette valeur,

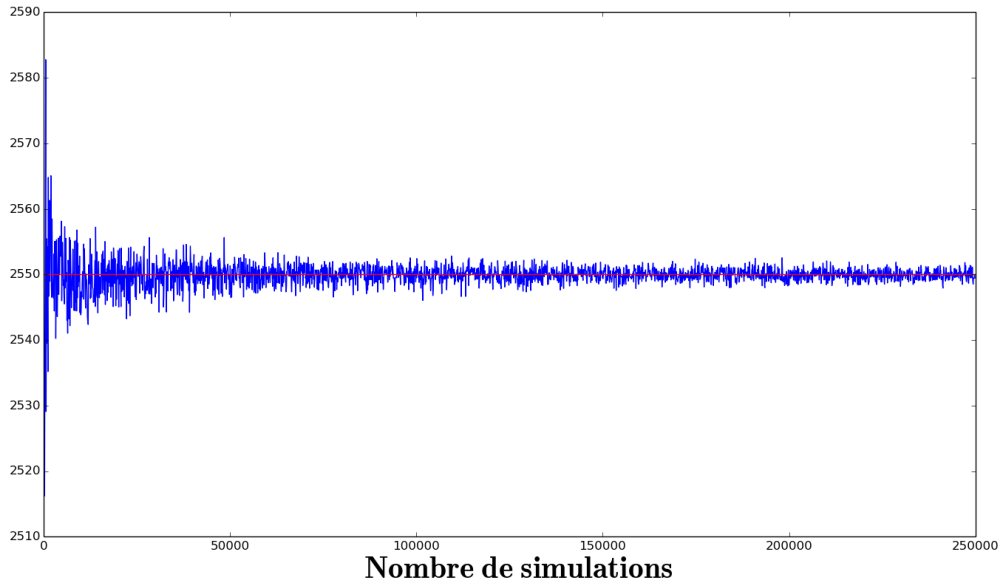

FIGURE 2.7: Évolution de la valeur moyenne de D en fonction du nombre de simulations (courbe bleu), valeur théorique moyenne de D (courbe rouge)

ce qui confère un niveau de confiance supérieur à 99.5% dans les résultats mentionnés précédemment.
Grâce à notre analyse de Taguchi nous pouvons alors réduire notre nombre de paramètres dans l'optique d'une analyse de Monte-Carlo en retirant les 3 paramètres identifiés comme ayant le moins d'influence sur la variation du paramètre de survivabilité.

2.2.4.3 Analyse des propriétés de la méthode de Taguchi appliquée à la méthode de Monte-Carlo

Dans cette sous-section nous allons vérifier sur un cas numériquement simple et purement théorique que, lorsque nous utilisons la méthode de Taguchi pour réduire le nombres de paramètres à perturber dans le cadre de l'analyse de Monte-Carlo (en ne gardant que les paramètres les plus influents), on ne modifie pas les propriétés statistiques du système.

Présentation du problème Considérons trois variables aléatoires : A, B et C, suivant une loi Gaussienne (normale, centrée) de moyenne 50 et d'écart type 5. On considère D tel que :

$$D = A * B + C, \tag{2.44}$$

Pour ce système on a la valeur moyenne théorique de D qui vaut 2 550.

Première analyse de Monte-Carlo Nous allons à présent effectuer une première analyse de Monte-Carlo sur ce système. Dans un premier temps nous allons effectuer

TABLE 2.5: Bilan de la première analyse de Monte-Carlo

Min	Max	Moyenne	Sigma	Skewness	Kurtosis
1249, 05	4318, 39	2550, 23	354, 59	0, 21	3, 06

TABLE 2.6: Configuration de l'analyse de Taguchi

Expériences	A	B	NA	C
1	43,88	43,88	NA	43,88
2	43,88	50,0	NA	50,0
3	43,88	56,12	NA	56,12
4	50,0	43,88	NA	56,12
5	50,0	50,0	NA	43,88
6	50,0	56,12	NA	50,0
7	56,12	43,88	NA	50,0
8	56,12	50,0	NA	56,12
9	56,12	56,12	NA	43,88

une première série de tests afin de contrôler la convergence de ce système. En effet, le reste de la démonstration n'aurait pas de sens si nous considérions un système de Monte-Carlo qui n'aurait pas convergé, c'est à dire dont la moyenne serait très proche de la moyenne théorique (c'est à dire lorsque l'équation (2.3) est valide). On considère donc sur la Figure 2.7 l'évolution de la valeur moyenne de D en fonction du nombre de simulations. Comme on peut le voir, à partir de 75 000 simulations on a convergence du résultat, avec une fluctuation de la valeur moyenne de D entre 2 547 et 2 553 environs. On effectue alors un premier run de Monte-Carlo, en calculant 100 000 simulations (pour se placer dans la zone où la méthode de Monte-Carlo a convergée). On trouve ainsi les résultats présentés Table 2.5 pour l'ensemble de ces calculs. Notre modèle étant linéaire on vérifie bien que l'on obtient une dispersion Gaussienne des résultats (Kurtosis = 3,06), quasiment centrée sur la valeur moyenne 2 550,23 (Skewness = 0,21) et d'écart type 354,59, confirmant la convergence de l'analyse de Monte-Carlo.

Analyse de Taguchi On effectue à présent notre analyse de Taguchi sur le système, afin de déterminer parmi A, B et C quels sont les paramètres qui ont le plus d'influence sur la dispersion de D, et quels sont ceux qui ont une influence négligeable.
Notre système étant composé de 3 variables nous utiliserons la table orthogonales L_9, cf. Table 2.2.
On considère les 3 niveaux suivants pour les 3 paramètres : $50 - \sqrt{1.55}; 50; 50 + \sqrt{1.55}$
De façon arbitraire on choisit d'attribuer le paramètre A à la première colone, B à la seconde et D à la quatrième colonne. On obtient ainsi les configuration présentées dans le tableau 2.6 pour les 9 simulations. De cette façon on obtient les résultats suivants pour D :
Simulation 1 : D = 1 969.3344

TABLE 2.7: Table d'analyse de la Variance pour D

	f	SS	V	F	SS'	P
A	2	561816,0	280908,0	300,3652	559945.557	49,64
B	2	561816,0	280908,0	300,3652	559945.557	49,64
C	2	1655,0649	827,5325	0,8849	-215,378	-0,02
Erreur	2	2805,6642	935,2214	1,0		

TABLE 2.8: Bilan de la seconde analyse de Monte-Carlo

Min	Max	Moyenne	Sigma	Skewness	Kurtosis
1266,38	4371,07	2550,13	355,32	0,22	3,09

Simulation 2 : D = 2 244

Simulation 3 : D = 2 518.6656

Simulation 4 : D = 2 250.12

Simulation 5 : D = 2 543.88

Simulation 6 : D = 2 856

Simulation 7 : D = 2 512.5456

Simulation 8 : D = 2 862.12

Simulation 9 : D = 3 193.3344

Dans un premier temps on vérifie bien que la valeur moyenne des 9 simulations calculées avec la méthode de Taguchi est égale à la valeur moyenne théorique de D : 2 550.

On peut à présent établir la table d'ANOVA qui est présentée Table 2.7 pour ce système. On trouve ainsi que les paramètres qui ont le plus d'influence sur la dispersion de D sont A et B; C ayant lui une contribution négligeable.

Seconde analyse de Monte-Carlo Pour cette seconde analyse de Monte-Carlo on se base donc sur les résultats de l'analyse de Taguchi et on ne considère plus que les perturbations sur A et B, et on fixe C à 50,0.

On commence par faire un premier test de convergence, comme pour la première analyse. Comme on peut le voir sur la Figure 2.8 on atteint la convergence aux alentours de 75 000 simulations. Dans notre cas, notre système étant très simple et composé de peu de variables, on ne constate aucun gain significatif sur le nombre de simulations à effectuer pour y parvenir.

On effectue alors un nouveau run de Monte-Carlo, de 100 000 simulations comme précédemment 2.2.4.3. On trouve alors les résultats présentés Table 2.8.

On vérifie bien avec cet exemple simple que la méthode de Taguchi n'a eu pratiquement aucun impact sur les résultats de la méthode de Monte-Carlo et que l'on a conservé les propriétés statistique du système malgré le fait que l'on ait négligé les perturbations des paramètres les moins influents.

On confirme ainsi que lorsque l'on considère des simulations de Monte-Carlo convergées,

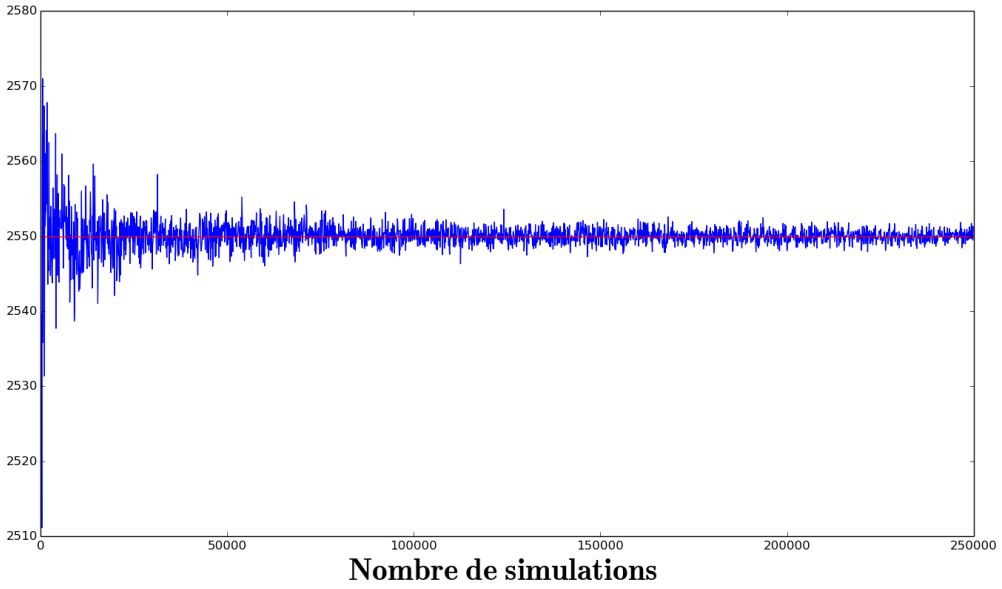

FIGURE 2.8: Évolution de la valeur moyenne de D en fonction du nombre de simulations (courbe bleu), valeur théorique moyenne de D (courbe rouge)

il n'y a pas de perte d'information en ignorant les perturbations sur les paramètres identifiés comme étant les moins influents par la méthode de Taguchi.

2.3 Conclusion

Dans ce chapitre nous avons présenté les différents outils d'analyse statistique mis en œuvre dans le cadre de ces travaux de thèse. Tout d'abord nous avons introduits la méthode de Monte-Carlo en présentant successivement les méthodes quasi et pseudo Monte-Carlo. Nous avons ensuite définis précisément la méthode d'analyse statistique que l'on nomme analyse de Monte-Carlo dans le cadre de ces travaux. Nous avons ensuite discuté des différentes informations qu'il était possible d'extraire d'une analyse de Monte-Carlo ainsi que des limitations de cette dernière. Nous avons alors introduit une autre méthode d'analyse statistique : la méthode de Taguchi. Cette méthode vise à identifier, parmi l'ensemble des paramètres perturbés d'un modèle, quels sont ceux qui ont le plus d'influence sur la dispersion d'un résultat et ceux qui en ont le moins. Après avoir présenté les bases théoriques de l'analyse de Taguchi (utilisation de Table Orthogonale, de la table d'ANOVA), nous avons illustré son utilisation avec un premier cas de rentrée atmosphérique de satellite défini par le CNES. Enfin nous avons illustré la combinaison de l'utilisation de la méthode de Monte-Carlo et de celle de Taguchi sur un cas théorique, pouvant être résolu de façon analytique, en montrant que l'utilisation de la méthode de Taguchi en préambule de la méthode de Monte-Carlo ne dégrade pas l'information que l'on peut extraire de celle-ci.

Chapitre 3

Les accélérateurs de calculs parallèles

Au cours de la dernière décénie les accélérateurs de calculs sont devenus des éléments incontournable dans l'univers du calcul haute performance. Dans ce présent chapitre nous allons présenter les deux principaux types d'accélérateurs de calculs parallèles que sont les cartes graphiques (Graphics Processing Units, GPU) et les coprocesseurs Intel Xeon Phi.

3.1 Introduction

Comme mis en évidence au Chapitre 2, les simulations de Monte-Carlo requièrent la résolution d'un nombre important de simulations. En fonction de la complexité du modèle numérique à traiter, la résolution de toutes ces simulations peut devenir prohibitive en terme de temps de calcul. Considérons l'exemple de la rentrée atmosphérique d'un satellite simple constitué d'une boite de dimension 1x1x1 m contenant une sphère de 0,5 m de diamètre traité à la sous-section 2.2.4. La résolution d'une simulation de ce système avec le logiciel Debrisk s'effectue en 8 s sur un CPU AMD FX-4100 Quad-Core. Mais effectuer une analyse de Monte-Carlo sur ce système en réalisant 1 000 000 simulations correspond à trois mois de calculs si ces derniers se font de façon séquentielle. Il est possible de descendre jusqu'à 23 jours de calcul si on tire partit des 4 cœurs de la machine pour effectuer les calculs en parallèle. Au vu de ces temps de calcul et de la simplicité du modèle traité, on comprend les problèmes que va poser le temps de calcul lorsqu'il faudra traiter un satellite plus complexe. Une solution serait alors d'utiliser des CPU plus performants avec plus de cœurs de calculs, ce qui nous permettrait d'effectuer plus de calculs en parallèle. Toutefois aujourd'hui les CPUs les plus performants contiennent au maximum une vingtaine de cœurs ce qui nous limite dans notre capacité à paralléliser les calculs. Il est alors possible de réaliser les calculs en parallèle sur un

cluster. En principe cette solution est en effet tout à fait envisageable puisqu'elle permet une parallélisation importante grâce à la quantité importante de CPUs multi-cœurs constituant un cluster.
Toutefois cette solution n'est pas forcément une bonne solution car elle peut se révéler trop contraignante dans certains cas, en particulier dans le cas de l'utilisation des outils d'analyse de Monte-Carlo en condition opérationnelle. Typiquement l'utilisation d'un cluster est soumise à la possibilité pour l'utilisateur de s'y connecter ainsi qu'à la disponibilité des ressources du cluster. Or dans le cadre de l'analyse en temps réel de la rentrée atmosphérique d'un satellite, ou bien dans celui de l'aide à la décision pour la séparation des ballons stratosphérique ces deux conditions sont potentiellement prohibitives, puisqu'on veut un résultat le plus rapidement possible ce qui n'est pas forcément possible si les ressources du cluster sont toutes réservées par d'autres utilisateurs. Dans le cas de la problématique de la détermination des zones probables d'atterrissages des ballons stratosphériques l'équipe opérationnelle se trouve souvent dans des régions reculées (Kiruna en Suède, Timins au Canada, ...) et n'a pas forcément la capacité matérielle d'établir une connexion avec un cluster distant.
C'est pour ces raisons que nous avons pris le parti pour notre plate forme de calcul parallèle "Design for Demise" de chercher à tirer parti des nouveaux accélérateurs de calculs parallèles que sont les cartes graphiques (Graphics Processing Units, GPU) et les cartes accélératrices Intel Xeon Phi. Dans le reste de cette étude nous nous concentrerons sur les cartes graphiques du fondeur NVIDIA qui avec la création de l'environnement de programmation CUDA a permis l'utilisation de ses cartes graphiques à des fins autres que le traitement et l'affichage d'images sur un écran d'ordinateur.
Ces accélérateurs sont constitués d'un grand nombre d'unités de calculs (4 992 cœurs CUDA pour la GPU NVIDIA K80 et 61 cœurs doté d'une unité de calcul vectoriel SIMD 512-bit pour le Xéon Phi) permettent une parallélisation importante des calculs. De plus ces cartes se connectent directement sur une unité centrale classique via le port PCI-Express, l'utilisateur peut ainsi avoir toujours à sa disposition une carte pour effectuer une analyse statistique au moment où il en a besoin et qui se révèle de plus économes en terme de consommation électrique.

3.2 Présentation des accélérateurs de calculs

3.2.1 Architectures

3.2.1.1 GPU

Le milieu des années 90 a vu l'émergence d'un nouveau composant matériel pour les ordinateurs de bureau ayant pour vocation l'affichage sur un écran, en temps réel, d'images

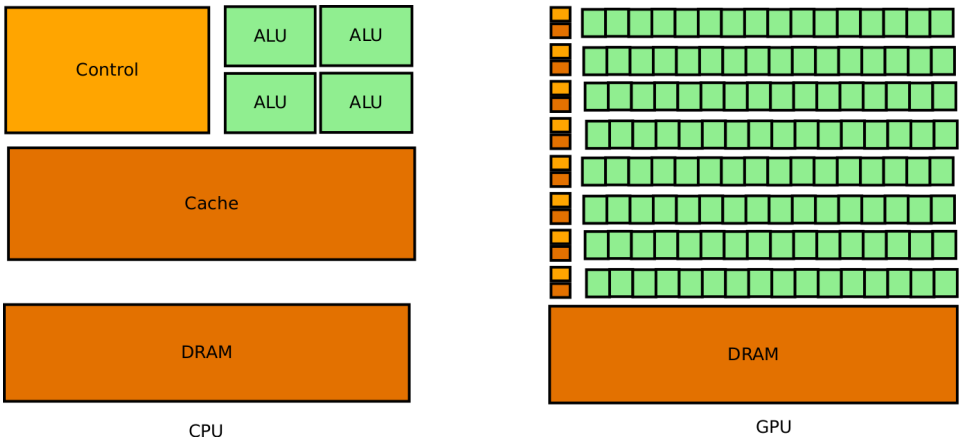

FIGURE 3.1: Répartition schématique des transistors sur CPU et GPU

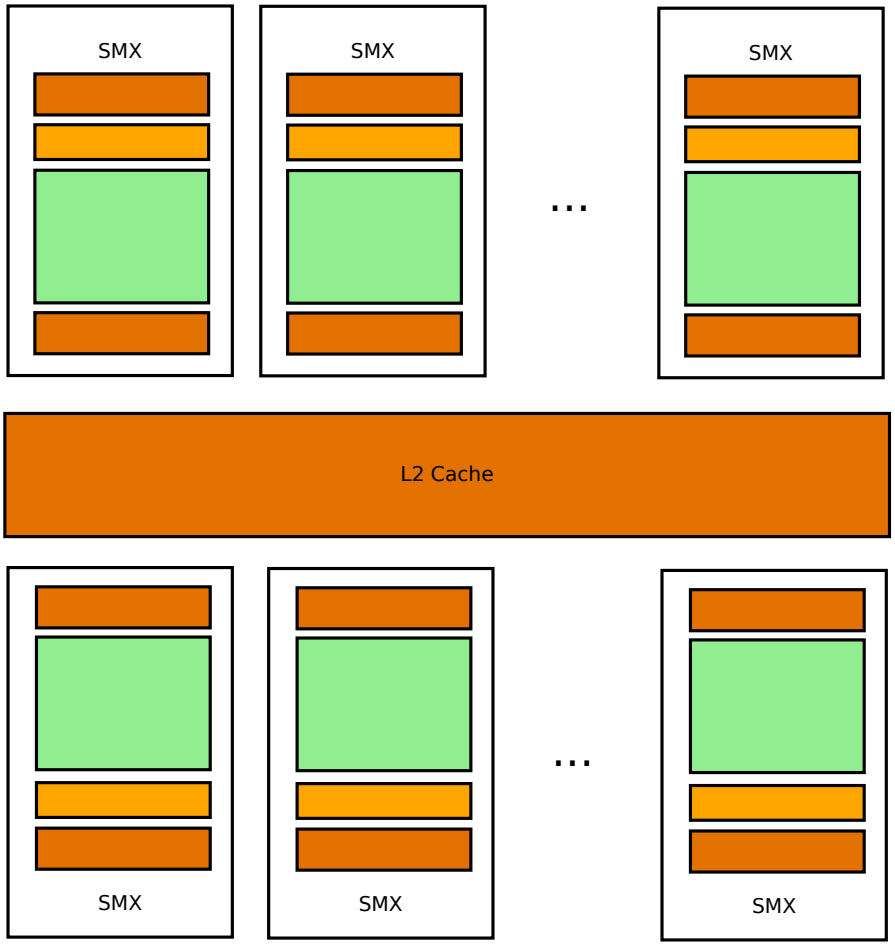

FIGURE 3.2: Architecture schématique d'une carte graphique de la famille Kepler

ou de vidéos 2D, tâche qui était jusqu'alors généralement effectuée par l'unité de traitement centrale (Central Processing Unit, CPU) de la machine. Or de par sa conception ce dernier était très peu performant pour cette tâche ce qui explique l'apparition et la rapide démocratisation d'un composant dédié : les cartes graphiques.

Avec le temps la demande en matière de rendu et de visualisation a évolué pour se

TABLE 3.1: Description des caractéristiques des GPUs

GPU	Architecture	SM	Cœurs CUDA	Cœurs CUDA D.P.	Frequence d'horloge (GHz)	Mémoire totale (GB)	Bande passante mémoire (GB/s)	Puissance Max (W)
GTX 760	Kepler	6	1152	0	0.98	2.05	192.2	170
K20	Kepler	15	2496	624	0.705	5	208	225
K40	Kepler	15	2880	960	0,745	12	288	235
K80	Kepler	26	4992	1664	0.560	24	480	300
K5000	Kepler	8	1536	512	0,71	4	173	122
Jetson TX1	Maxwell	NA	256	0	1.91	4	25.6	15
P100	Pascal	60 SMP Streaming Multiprocessor Pascal	3584	1792	1.5	16	720	300

tourner du traitement de graphisme 2D vers les graphismes 3D en haute définition notamment sous l'impulsion du domaine du jeu vidéo. Cette évolution a ainsi poussé les GPUs à évoluer pour devenir progressivement des environnement multi-cœur, hautement parallèles avec une bande passante mémoire très élevée.

En raison de cette spécialisation l'architecture des GPUs est radicalement différente de celle des CPUs, comme on peut le voir sur la Figure 3.1 qui présente de façon schématique la répartition des transistors sur les deux architectures. Le CPU est spécialisé pour gérer plusieurs tâches complexes et différentes en parallèle qui réclament beaucoup de données. Cela se traduit en terme d'architecture par peu d'Unité Arithmétique et Logique (Arithmetic-Logic Unit, ALU) mais un cache important pour permettre un accès rapide aux données ainsi qu'une unité de contrôle complexe permettant une gestion optimale des données en cache ainsi qu'une occupation maximale des ALUs.

À l'inverse les GPUs sont spécialisés dans les calculs rapides et hautement parallèles comme la détermination des valeurs de chaque pixel d'un écran. Ils possèdent donc beaucoup plus de transistors dédiés au traitement des données qu'à la mise en cache de ces dernières ou au contrôle des flux d'instructions.

La Figure 3.2 présente de façon schématique l'architecture générale d'une carte graphique de la famille Kepler du fabricant NVIDIA. Comme on peut le voir celle-ci s'organise avec un ensemble de Streaming Multiprocesseur (SMX), contenant chacun un ensemble d'unités de calculs, agencés autour d'un bloc de mémoire cache L2 (GDDR5, Graphics Double Data Rate version 5) qui va contenir des données qui seront accessibles à toutes les unités de calcul de la carte. Chacune des ces unités de calcul, que l'on nomme cœur CUDA, exécute un des threads (un des processus parallèles), chaque cœur exécutant la même instruction mais sur une donnée différente selon le modèle SIMD, pour instruction unique, données multiples (Single Instruction, Multiple Data). Dans le cadre de la programation sur GPU on parle aussi de modèle SIMT pour instruction unique, threads multiple (Single Instruction, Multiple Threads). Chacun de ces cœurs est capable d'effectuer les opérations arithmétiques de base, selon le type de cœur en double précision (DP CUDA core) ou bien en simple précision, virgule flottante ou entier (CUDA core), cf. Figure 3.3. En plus de ces cœurs de base, les cartes graphiques disposent aussi d'unités de chargement et d'écriture des données en mémoire (LD/ST) et d'unités dédiées au traitement des opérations à virgule flottantes complexes (opération

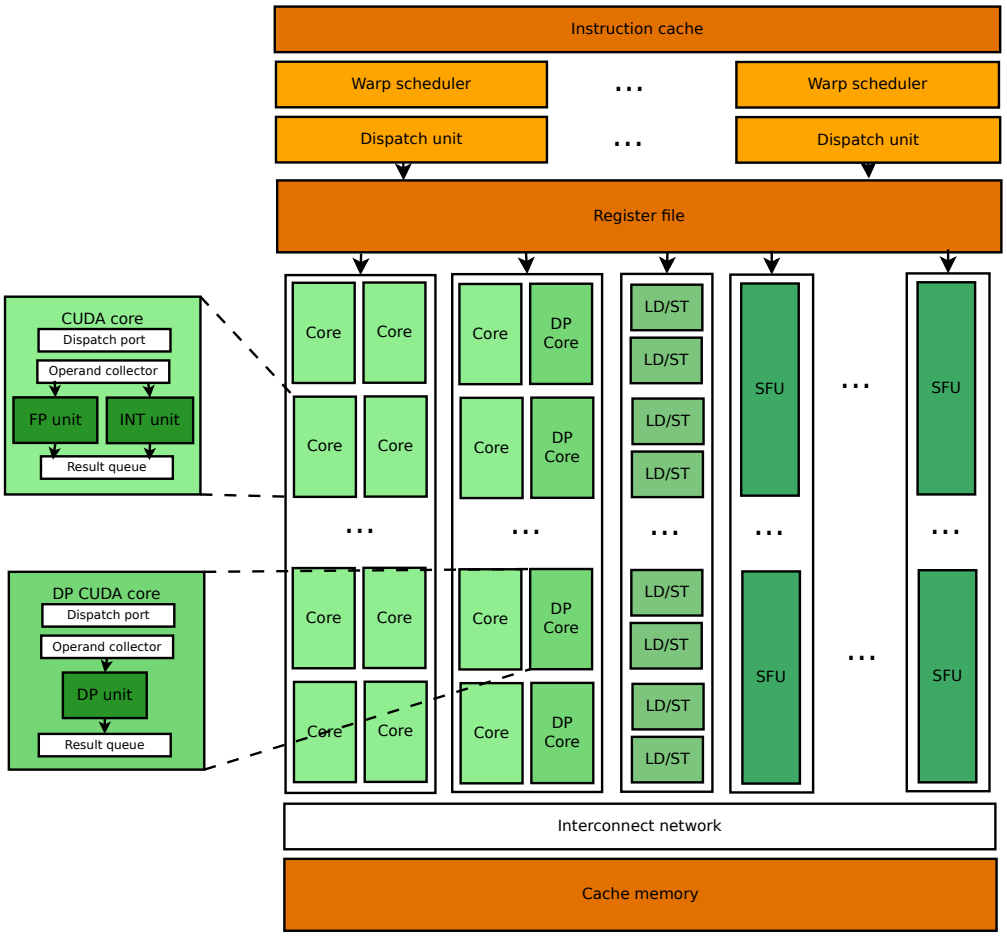

FIGURE 3.3: Schéma d'un streaming multiprocesseur

réciproque, sinus, cosinus, etc...) : les unités de fonction spéciales (SFU). Ces unités sont donc regroupées en sous ensemble au sein des SMXs. On note aussi une mémoire cache qui est uniquement accessible aux cœurs du SMX et qui se compose d'une mémoire cache de type L1 et d'une mémoire de texture (réservé pour des variables définies comme globales et statiques). Les SMXs gèrent ainsi la distribution des threads sur les cœurs de calcul en regroupant les threads par paquet de 32 que l'on nomme warps. Au sein d'un warp toutes les threads vont effectuer la même instruction mais chacune sur une donnée différente. Cette tâche est assurée par les Warp scheduler. Cette gestion en warp des threads permet une meilleure souplesse puisque deux warps distincts peuvent exécuter des instructions différentes. Néanmoins si il y a divergence des threads au niveau d'un warp, cela va provoquer une sérialisation des calculs pour ce warp entraînant une perte de performance significative de l'application. Le nombre de SMX, de cœurs CUDA, le ratio entre les cœurs doubles et simples précisions, etc... dépend du type de carte et de son architecture. Le passage à l'échelle (scalabilité) de la composition des cartes permet ainsi à NVIDIA d'utiliser la même architecture de base pour développer des cartes graphiques spécialisées pour des domaines aussi variés que le jeux vidéo (gamme

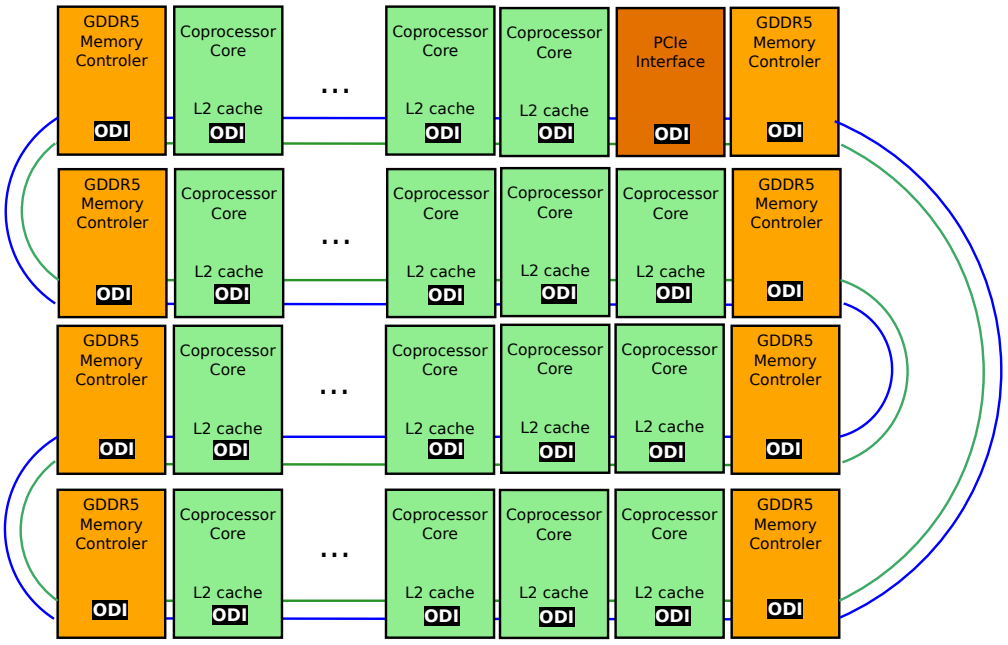

FIGURE 3.4: Vue schématique de l'architecture de la puce de l'Intel Xeon Phi

GeForce), le graphisme professionnel (gamme Quadro), le calcul haute performance et les data centers (gamme Tesla) ainsi que le deep learning et le calcul embarqué (carte Jetson TX1). Le Tableau 3.1 présente ainsi les quantités des différents composants pour divers GPUs dont notamment les cartes NVIDIA K40 (carte spécialisée pour le calcul haute performance) et K5000 (carte spécialisée pour le graphisme professionel) qui ont été utilisés dans le cadre de ces travaux de thèse. Enfin le lecteur pourra trouver en Annexe B.1 la roadmap pour la gamme de produit NVIDIA.

3.2.1.2 Intel Xeon Phi

Au milieu des années 2000, suite à l'arrêt de son projet de carte graphique basé sur les architectures multi-cœur Larrabee, la société Intel a lancé le développement de sa propre gamme de coprocesseurs dédiée au calcul haute performance (Hight Performance Computing, HPC) basée sur ces mêmes architectures multi-cœurs [68]. L'objectif derrière le développement de cette gamme d'accélérateurs de calculs est alors la création d'un coprocesseur parallèle qui puisse se programmer à partir des langages de programmation parallèle déjà existant tels que OpenMP ou MPI afin que les différents codes qui tournaient jusqu'alors sur des clusters ou des machine multi-cœurs puissent s'exécuter sur ces accélérateurs sans qu'il ne soit nécessaire de les réécrire intégralement.
Après plusieurs années de développement la société Intel a commercialisé en 2013 sa propre gamme d'accélérateur de calculs dédié au HPC : les coprocesseurs Intel Xeon Phi. Ce dernier se présente sous la forme d'un coprocesseur qui, comme les cartes graphiques,

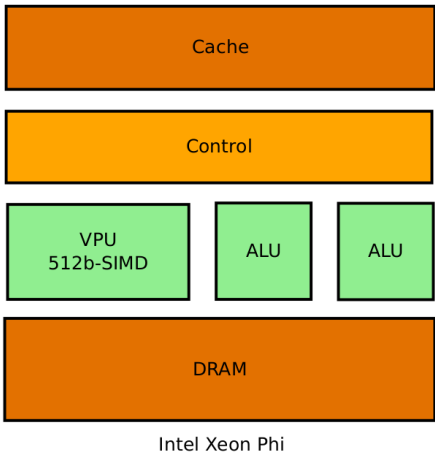

FIGURE 3.5: Répartition schématique des transistors sur un cœur d'Intel Xeon Phi

se connecte sur l'unité centrale via une connexion PCI-Express. Cette carte est constituée d'une puce de silicium contenant jusqu'à 61 cœurs, d'une puce mémoire GDDR5 qui peut s'assimiler à un cache L2 distant relié par 16 canaux à la puce principale et un système de contrôle et de management (SMC). Cette unité SMC, équivalente aux puces que l'on peut trouver sur chaque ligne d'alimentation sur les GPUs, est chargée de surveiller le bon fonctionnement de la carte (tension, température, etc...) et d'ajuster en temps réel les "fonctions vitales" de la carte (ajustement des ventilateurs, gestion de l'alimentation, de la fréquence des cœurs, etc...) [37].

La Figure 3.4 présente une vue schématique de l'organisation des cœurs dans la puce en silicium. Les cœurs (coprocessor core) sont ainsi organisés sur un anneau bidirectionnel (représenté par les traits vert et bleu sur la figure pour chaque direction) selon un processus d'interconnexion On-Die (ODI), c'est à dire que la connexion est gravée directement dans le silicium. Chaque cœur est capable de traiter deux instructions par cycle et par sens de circulation de l'anneau, soit quatre instructions en simultané. Chaque cœur du Xeon Phi étant doté d'une unité vectorielle ces instructions peuvent trés bien être des instructions vectorielles. On parle ici de threads hardware. Sur cet anneau figurent aussi une unité dédiée assurant la gestion des communications avec l'hôte : l'interface PCIe, ainsi que 8 unités de gestion de la mémoire GDDR5 (GDDR5 Memory Controler), uniformément réparties sur l'anneau et qui assurent le transfert des données avec la mémoire GDDR5 de façon transparente pour le programmeur : l'accès aux données dans la mémoire GDDR5 se fait sans qu'il ne soit nécessaire d'ajouter dans le code des instructions spécifiques. Chaque cœur dispose de son propre cache L2 de 512-KB, qui est adressé par un système de répertoires labellés de telle façon qu'un cœur peut accéder à n'importe quelle donnée qui se situerait sur un autre cœur tout en assurant une cohérence de la donnée à travers les caches sans intervention logicielle. Grâce à cette structure et à ces différents composants, la puce du Xeon Phi peut être assimilée

TABLE 3.2: Description des caractéristiques des coprocesseurs Intel Xeon Phi

	Architecture	Cœurs	Fréquence (GHz)	Memoire Totale (GB)	Bande passante mémoire (GB/s)	Unité Vectorielle	Puissance Max (W)
Xeon Phi 3120P	KNC	57	1.1	6	240	AVX-512 (512-bit SIMD)	300
Xeon Phi 5110P	KNC	60	1.053	8	320	AVX-512 (512-bit SIMD)	225
Xeon Phi 7120P	KNC	61	1,24	16	352	AVX-512 (512-bit SIMD)	300
Xeon Phi 7250	KNL	68	1.4	≤ 384	115.2	2 x AVX-512 (512-bit SIMD)	215
Xeon Phi 7290F	KNL	72	1.5	≤ 384	115.2	2 x AVX-512 (512-bit SIMD)	260

à un système multiprocesseur symétrique à mémoire unifié (SMP).

Les cœurs du coprocesseur sont basés sur l'architecture des x86 Pentium 54C modifiée et améliorée par l'ajout d'unité de calcul vectoriel 512-bit SIMD (AVX-512). Ces cœurs bénéficient de la technologie de gravure des transistors à 22 nm d'Intel, qui permet d'avoir autant de cœurs x86 sur une surface comparable à celle d'un CPU multi-cœurs plus conventionnel. La Figure 3.5 présente une vue schématique de la répartition des transistors à l'intérieur d'un des cœurs du Xeon Phi. Comme on peut le constater on a une répartition qui est très similaire à celle que l'on a pour un CPU standard (Figure 3.1), ce qui se traduit par une capacité, pour chaque cœur, à traiter un ensemble de tâches complexes et différentes (à la façon d'un CPU) à l'inverse des cartes graphiques dont les cœurs ne peuvent gérer qu'une tâche à la fois.

On touche là au cœur même de la différence de conception entre les cartes graphiques et des coprocesseurs Intel Xeon Phi. Ces deux architectures ont été conçues pour traiter des problèmes à parallélisme de données. Les deux architectures ont opté pour une fréquence des cœurs de calcul plus basse que celle des processeurs classiques, pour limiter la consommation électrique. Mais là où les GPUs se basent sur un nombre de cœurs importants réalisant la même tâche en parallèle et jouent sur les warps pour obtenir une certaine granularité, les cartes Xeon Phi se reposent sur un nombre de cœurs plus limités capables de gérer des tâches différentes et utilisent leurs unités vectorielles pour massivement vectoriser chacune de ces tâches. Notons que malgré ces approches différentes, les deux cartes accélératrices sont données avec des performances pic comparables : 1,43 Tflops en calcul double précision pour la carte NVIDIA K40 et 1,21 Tflops pour la carte Intel Xeon Phi 7120P.

Terminons cette sous-section sur l'architecture du Xeon Phi avec quelques chiffres complémentaires. Le Tableau 3.2 présente les caractéristiques des Xeons Phi dont la carte 7120P qui a été utilisée pour les travaux réalisés durant ce travail de thèse. Enfin le lecteur pourra trouver en Annexe B.2 la roadmap pour les produits Xeon Phi.

3.2.2 Modèles de programmation

3.2.2.1 GPU

CUDA En Novembre 2006 la société NVIDIA lance CUDA (Computing Unified Device Architecture), afin d'offrir aux développeurs un environnement de programmation leur permettant d'utiliser les capacités de traitements parallèles des cartes graphiques de la firme pour leurs applications (calcul haute performance, jeux vidéos, traitement d'images,etc...) sans qu'ils aient besoin d'apprendre des langages complexes de programmation graphique tel que les shaders ou autres.
Le terme CUDA désigne ainsi un environnement propriétaire gratuit qui se compose en deux parties :

- un environnement d'exploitation comprenant les différents pilotes pour les cartes graphiques de la firme NVIDIA ainsi qu'une série d'utilitaires de configuration.
- un environnement de développement CUDA C contenant une série d'extension et de fonctions pour le langage C ainsi qu'un ensemble de bibliothèques dédiées au calcul sur GPU.

Les différents éléments de l'environnement de développement sont directement utilisables dans les codes C, C++ et Fortran ou via des wrappers (fonctions assurant l'interface entre le langage choisi et CUDA C) pour les codes en Java, Python et C#. Il est aussi possible d'accéder à ces éléments via des directives de programmation (ex : OpenACC, OpenCL). CUDA C permet d'écrire du code pouvant être destiné soit à destination du CPU (l'hôte) soit à la destination du ou des GPUs connectées sur la plateforme hôte. Le développeur dispose d'un ensemble d'interface de programmation (API) de plus ou moins haut niveau lui offrant différents niveaux de maîtrise sur la parallélisation de son application, selon le niveau d'accélération qu'il souhaite obtenir.
Cet environnement comprend aussi un compilateur propre NVCC, basé sur le compilateur GNU gcc, qui permet de compiler le code pour l'exécuter sur le GPU. À noter qu'il est aussi possible d'utiliser les compilateurs de PGI et de LLVM pour compiler les codes utilisant CUDA C.
Aujourd'hui encore, CUDA est un environnement en perpétuelle évolution, qui durant cette thèse aura effectué 5 changement de versions (de 5.0 à 7.5, 2 versions majeures et 3 mineures) chacune apportant son lot de nouveautés, d'optimisation de performance et d'améliorations simplifiant la programmation sur GPU (ex : introduction de la mémoire unifiée permettant en une instruction d'adresser de la mémoire sur le CPU et sur le GPU).
Notons enfin que si CUDA C ne permettait jusqu'alors d'utiliser que des cartes graphiques NVIDIA pour exécuter du code parallèle, AMD a lancé fin 2015 l'Initiative Boltzmann qui comprend un ensemble d'outils pour permettre aux codes CUDA de tourner sur ses

cartes graphiques [4], ouvrant d'avantages de possibilités et de plate formes aux codes CUDA C.

Programmation hétérogène et hiérarchisation de la mémoire Dans ce paragraphe et le reste de ce manuscrit nous aborderons uniquement l'appel direct aux fonctions CUDA C dans un code code C, C++, car il s'agit de la méthode permettant d'avoir le plus de contrôle sur la parallélisation du code sur GPU et c'est celle qui a été mise en œuvre dans le cadre de ces travaux.
Avant de commencer à discuter des principes de programmation pour GPU avec CUDA, il semble pertinent de rappeler que les cartes graphiques que nous considérons possèdent leur propre système de mémoire qui est embarqué sur la carte. Ce système de mémoire est donc indépendant du reste de la plate-forme sur laquelle la carte est branchée et l'espace disponible est dépendant de la carte considérée et ne peut être augmenté par l'ajout de barrettes mémoires ou autre. De plus ces cartes ne disposent pas de système d'exploitation propre, on ne peut pas développer un code que l'on lancerait directement sur la carte et qui tournerait intégralement sur la carte. Il est impératif de passer par le CPU pour lancer le calcul sur le GPU et de revenir sur le CPU pour récupérer les résultats : typiquement, à titre d'illustration, on ne peut pas écrire un fichier de résultat directement depuis le GPU sur le GPU ou sur le CPU.
Pour cette raison un code CUDA est composé de deux sections : une qui va s'exécuter sur le CPU et une autre qui va s'exécuter sur le GPU. Le code CUDA commence donc toujours par une partie séquentielle qui s'exécute sur le CPU et qui sert, au minimum, à l'instanciation des calculs sur le GPU : création des vecteurs de données dans la mémoire globale de la carte et transfert des données nécessaires du CPU vers le GPU. Ce processus s'effectue soit manuellement en déclarant et en allouant les données sur l'hôte, puis en déclarant leur contrepartie sur le GPU avant d'effectuer la copie des données du CPU vers le GPU, soit depuis l'arrivée de CUDA 6 automatiquement, en déclarant les données dans la mémoire unifiée, qui permet en une instruction de réserver l'espace mémoire sur le CPU et sur le GPU, et assure automatiquement le transfert des données entre le CPU et le GPU selon le besoin.
Lorsque cette étape de préparation est terminée le code séquentiel lance ensuite un kernel qui se présente sous la forme d'une fonction C classique mais qui va s'exécuter de façon parallèle sur le GPU selon une grille de calcul. On appelle ainsi chaque processus d'un kernel sur une donnée un thread. Cette définition diffère de la définition du thread en MPI, puisque dans le contexte SIMD du GPU un thread peut s'envisager comme le traitement d'une seule donnée, chaque thread traitant une donnée différente en parallèle, alors que le thread dans le contexte MPI peut être quelque chose de beaucoup plus complexe.
Cette grille de calcul se décompose ensuite en blocs qui regroupent un certain nombre de

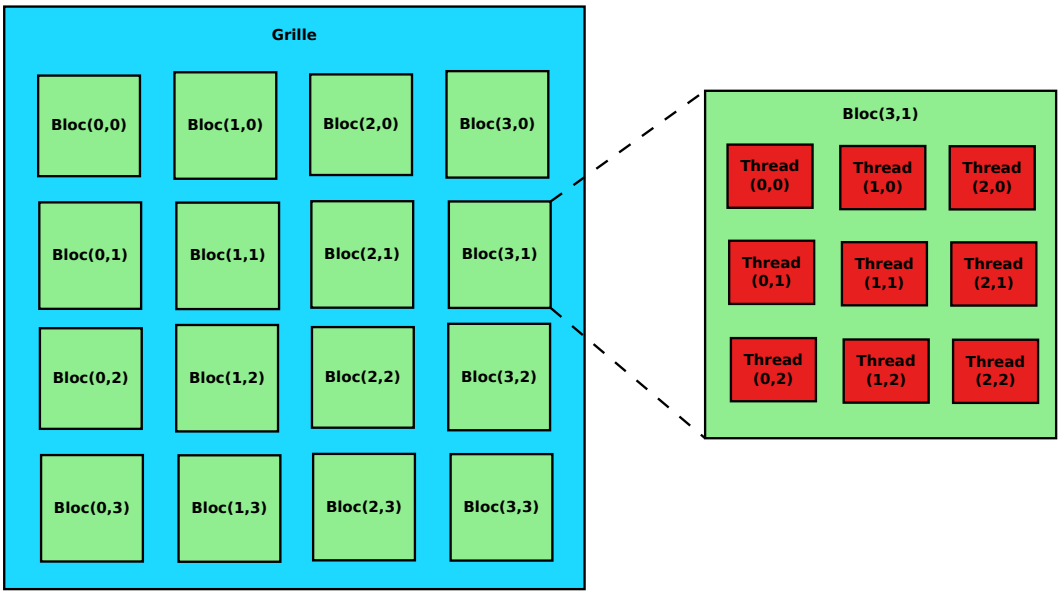

FIGURE 3.6: Hiérarchisation des threads sur une grille de calcul CUDA sur le GPU

threads. Au sein du kernel, chaque bloc dispose d'un identifiant unique, et chaque thread dispose d'un identifiant unique au niveau du bloc. La conjonction de ces deux systèmes permet de définir pour chaque thread un identifiant unique à l'échelle du kernel. La Figure 3.6 présente ainsi une vue schématique de cette hiérarchisation. Le dimensionnement de cette grille (le nombre de threads par bloc, et le nombre de blocs par grille) se définit au lancement du kernel et peut varier d'un kernel à l'autre.
Cette hiérarchisation des threads en plus de fournir une structuration des threads qui s'avère particulièrement utile pour les opérations sur les vecteurs et les matrices par exemple, est aussi directement liée avec la hiérarchisation de le mémoire. En effet on distingue trois niveaux de mémoires (cf. Figure 3.7) qui sont directement accessibles pour le programmeur :

- La mémoire globale, accessible à tous les threads de la grille ainsi qu'au CPU. Toutes les données provenant du CPU ou devant être envoyées au CPU devront se trouver dans la mémoire globale. Il s'agit de la zone mémoire la plus grosse du GPU (plusieurs GB), mais c'est aussi celle qui présente la latence la plus élevée.

- La mémoire partagée par bloc, qui permet aux threads d'un même bloc d'accéder et de modifier une même donnée, sans qu'un thread d'un autre bloc n'y ait accès.

- La mémoire locale à la thread, qui n'est accessible que par le thread. Il s'agit de la mémoire ayant la latence la plus basse, mais c'est aussi la mémoire la plus petite (quelques KB).

Pour faire le lien avec la sous-section 3.2.1.1 notons enfin que les blocs sont assignés aux SMs (selon la dimension des blocs, 2 blocs ou plus par SM).

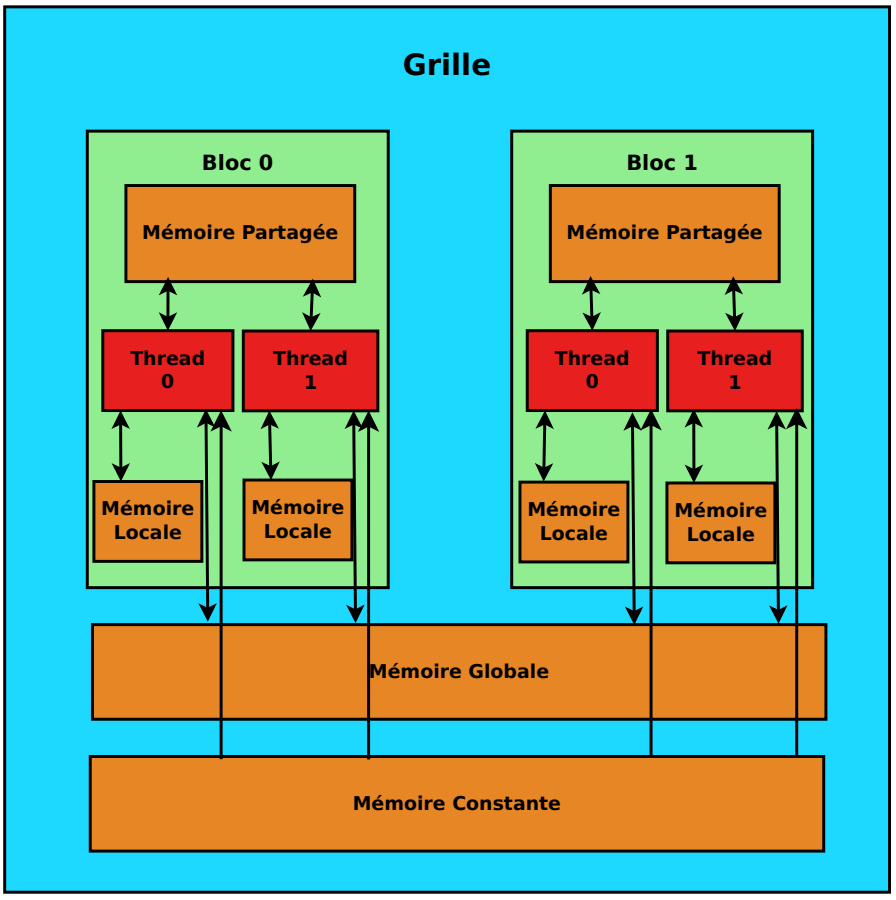

FIGURE 3.7: Hiérarchisation des accès mémoires GPU

Langages à directives Dans la sous-section précédente nous avons détaillé comment, grâce à l'environnement de programmation CUDA C, le programmeur pouvait de façon fine programmer son application de façon à tirer profit au maximum de toutes les caractéristiques des cartes graphiques en terme de gestion de mémoire etc... Toutefois un programmeur n'est pas non plus condamné à un tel degré de maîtrise de son application pour faire du calcul sur GPU. En effet, grâce aux langages à directives tels que OpenACC [57] ou OpenHMPP [26] un programmeur peut grâce à quelques directives porter un code C sur GPU. Un des avantages de ces langages est de permettre l'écriture de codes qui soient facilement portables sur différentes architectures parallèles telles que les GPUs, les coprocesseurs Intel Xeon Phi ou un cluster. Néanmoins cette portabilité se fait au détriment de la performance puisque les codes ainsi produits ne peuvent pleinement tirer profits des spécificités de chaques architectures. De plus ces languages ne permettent pas de tirer profits de certaines bibliothèques spécifiques : par exemple il est impossible d'utiliser la bibliothèque cuRAND de NVIDIA pour la génération des nombres aléatoires dans un code OpenACC. Autant de raisons qui font que ces langages à directives n'ont pas été considéré au cours de ces travaux de thèse.

3.2.2.2 Xeon Phi

Environnement de travail Intel Xeon Phi Le coprocesseur Intel Xeon Phi s'accompagne d'un environnement qui se compose de deux parties majeures : la première étant une partie logiciel se chargeant de la gestion de la carte et de ses fonctionnalités, la seconde étant un ensemble d'outils dédiés au développement et à l'exécution de codes de calcul [68].
La première partie logiciel se nomme Intel Many Integrated Core Platform Software Stack (Intel MPSS). Elle contient tous les outils nécessaires pour faire fonctionner le Xeon Phi et assurer son interaction avec la plate-forme hôte. C'est donc l'Intel MPSS qui permet d'amorcer la carte et d'émuler le comportement d'un nœud classique avec un disque dur via un RAM disque, d'assurer les communications Internet via une adresse IP virtuelle. C'est elle aussi qui se charge d'assurer le fonctionnement du noyaux Linux sur la plate-forme, permettant à un utilisateur de se connecter directement sur la carte à la façon d'un nœud classique. Elle gère enfin aussi l'adressage mémoire pour la configuration du système, les entrées-sorties et les mouvements de mémoires interne au Phi.
La suite Intel Parallel Studio consitue la seconde partie de cet environnement dédié au Xeon Phi. Elle regroupe un ensemble de composants logiciels permettant le développement et le debugage de codes sur Xeon Phi, la compilation des codes pour le Phi, avec une série de compilateurs Intel C/C++, Fortran, ainsi que des outils pour la parallélisation pour les différents langages de programmations avec des extensions spécifiques OpenMP pour le Phi, des bibliothèques dédiées au calcul (Intel MKL), et des instructions de parallélisations propriéraires telles que Intel Thread Building Block, Intel Cilk Plus à insérer dans des codes C pour automatiser la parallélisation des codes à la compilation pour le Phi.

Mode de programmation et gestion des données en mémoire

Mode de programmation Nous l'avons vu à la sous-section précédente, les cœurs du coprocesseur Intel Xeon Phi sont basés sur des cœurs Pentium, qui ont été modifiés pour correspondre aux exigences de conception de la nouvelle carte. Cette base de cœur Pentium permet aux cartes Xeon Phi de supporter les codes parallèles déjà existants, et pas seulement. Cette base permet aussi aux cartes Xeon Phi de faire tourner leur propre micro OS basé sur un noyau Linux modifié, permettant au Xeon Phi de disposer d'un système de gestion de répertoire, de protocole de communications, etc... à la façon d'une unité centrale classique. Un coprocesseur qui est branché sur une unité centrale apparaît alors depuis le poste hôte comme une machine, ou un nœud de serveur, branché sur le réseau et accessible via ssh. Ce dernier point est loin d'être anodin car il permet une grande versatilité quand à l'utilisation du Phi pour accélérer un code.
En effet contrairement aux cartes graphiques qui s'utilisent toujours en suivant le même

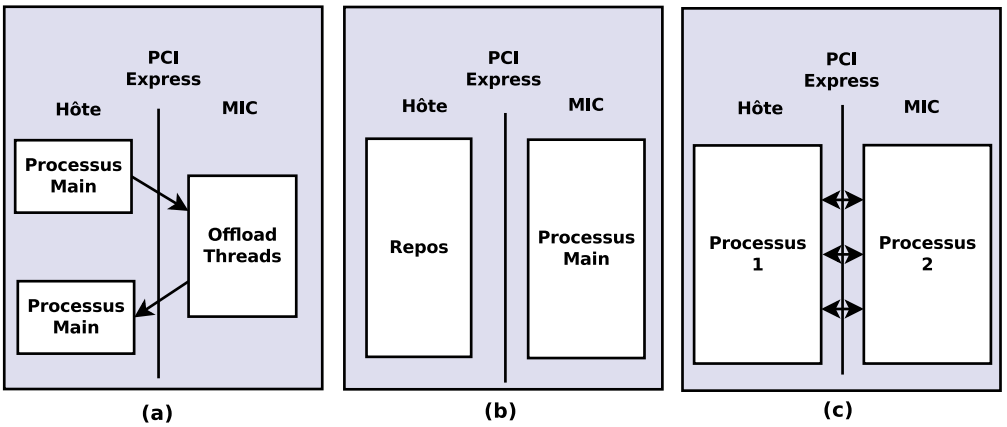

FIGURE 3.8: Modèles d'exécutions sur le Xeon Phi : (a) mode offload, (b) mode natif et (c) mode symétrique

protocole, présenté à la sous-section 3.2.2.1, a savoir le code commence de façon séquentielle sur le CPU, puis déporte tout ou parti du calcul sur le GPU, avant de rapatrier les résultats sur le CPU, on identifie trois modes de programmation pour les cartes Xeon Phi, présentés de façon schématique à la Figure 3.8 :

- Le mode offload, ou hétérogène (Figure 3.8.(a)) : ce mode est identique au mode de programmation des GPUs, avec une partie séquentielle du code qui s'exécute sur le CPU, puis tout ou partie du calcul qui est exporté sur le Phi pour être exécuté de façon parallèle. En revanche à l'inverse du GPU, il n'est pas obligatoire de rapatrier les résultats sur le CPU, le Xeon Phi disposant des composants logiciels nécessaire pour les écrire dans un fichier de résultat par exemple.

- Le mode natif (Figure 3.8.(b)) : ce mode tire avantage de la possibilité offerte par le micro-OS Linux, de se connecter sur le Phi comme on le ferait pour un nœud de cluster. Une fois connecté par ssh sur le Phi, il est alors possible de lancer directement son application sur le Phi. Les deux seules limites à l'utilisation de ce mode étant tout d'abord de compiler le code spécifiquement pour le Phi, en ajoutant l'instruction -mmic par exemple pour le compilateur Intel. Il faut ensuite que l'application tienne intégralement dans la mémoire du Phi. Le grand avantage de ce mode de programmation est de permettre au programmeur de totalement s'affranchir des communications entre le CPU et la carte via le port PCIe, communications notoirement connues pour être très pénalisantes en terme de temps car très lentes en comparaison des autres communications (au sein de la carte ou du CPU).

- Le mode symétrique (Figure 3.8.(c)) : dans ce mode l'application tourne en parallèle sur la machine hôte et sur le coprocesseur et s'apparente au fonctionnement classique que l'on a lorsqu'un code tourne sur plusieurs nœuds d'un cluster. Notons qu'ici aussi la partie du code qui doit tourner sur le Phi doit être compilée

spécifiquement pour ce dernier et le développeur doit s'assurer que cette partie tienne dans l'espace mémoire du Phi. Le défi de ce mode est de réussir à obtenir un bon équilibrage des charges entre ce qui est exécuté sur le CPU et ce qui est exécuté sur le Phi afin d'optimiser les temps de calcul et de limiter les temps d'inactivité de chaque support.

Gestion de la mémoire Comme mis en évidence à la sous-section 3.2.1.2, les cartes Intel Xeon Phi disposent elles aussi d'un système de hiérarchisation de la mémoire qui, bien que très différent sur le plan hardware de ce qui est fait sur les cartes graphiques, suit une logique similaire. Alors qu'avec CUDA et les cartes graphiques on observe une sorte de symbiose au niveau de la hiérarchisation de la mémoire entre le hardware et le software, cette hiérarchisation de la mémoire peut être reproduite via les langages de programmation MPI ou OpenMP et s'effectue de façon transparente pour le programmeur grâce à "l'intelligence" du compilateur Intel, là où la hiérarchisation doit être explicite en CUDA. Ainsi toutes variables déclarées hors des zones de code parallèles mais qui sont labellées comme partagées entres les différents processus parallèles sont transférées dans la mémoire commune GDDR5, alors que les variables déclarées au sein d'un processus parallèle et labellées comme privées, demeurent dans la mémoire L2 de chaque cœur. Ici le caractère privé ne vient pas de la localisation mémoire (comme pour le GPU), mais de la façon dont la variable est déclarée, avec l'emploi du tag private en OpenMP par exemple. Notons enfin qu'une variable déclarée au sein d'un processus parallèle et n'étant pas clairement taggée comme privée est accessible à tous les autres processus parallèles, le concept de bloc sur Xeon Phi n'existant pas, mais la donnée demeure bien dans la mémoire L2 du cœur hébergeant le processus qui la créée.

Interfaces de programmation parallèle pour le Xeon Phi À l'inverse des GPUs l'un des avantages majeurs des coprocesseurs Xeon Phi est de permettre aux programmeurs d'utiliser des interfaces de programmation parallèle déjà existantes telles qu'OpenMP, MPI ou même OpenACC ou OpenCL pour développer leur applications parallèles. Grâce à celà les applications parallèles déjà existantes pour des clusters par exemple peuvent s'éxécuter sur Xeon Phi. À noter que cela n'est pas pour autant un gage de performance optimale sur le Phi.
Pour faire le lien avec la sous-section 3.2.2.2, dans le cadre d'une application dédiée au Xeon Phi, dans les modes de programmations offload et symmétrique MPI pourra être utilisé pour assurer les communications entre le processus sur l'hôte et celui sur le coprocesseur. Chaque cœur du Phi disposant de son propre espace mémoire, le coprocesseur peut être considéré comme une machine à mémoire partagée et MPI peut être utilisé pour lancer des processus parallèle sur chaque cœur.
OpenMP peut aussi être utilisé pour dans le cadre du modèle offload ou natif pour

déployer l'application sur les cœurs du Phi, et de façon générale sera toujours utilisé afin de tirer profit des capacités de calcul vectoriel SIMD [1]. Il est ainsi envisageable de combiner MPI et OpenMP, le premier afin d'instancier les processus parallèles sur les cœurs du coprocesseur, et le second au sein des processus parallèles MPI pour tirer profit des unités vectorielles de la carte.
Pour ce qui concerne OpenACC et OpenCL, les deux peuvent être employés de la même façon que pour le GPU, avec les mêmes restrictions en terme d'exploitation sous optimales de l'architecture. Par exemple il n'existe pas d'instruction de calcul vectoriel en OpenACC.

3.3 Utilisation des accélérateurs pour la résolution d'intégrateur numérique dans un contexte Monte-Carlo

3.3.1 Les applications agréablement parallèles

Dans son ouvrage traitant du calcul parallèle Fox [32] introduit pour la première fois la notion d'applications "embarassingly parallel", qui sera renomé par la suite "pleasingly parallel" [36] afin de lever toutes ambiguïtés concernant l'utilisation du terme "embarassingly", et que l'on traduira ici par "agréablement parallèles". Ce terme désigne tout problème dont le graphe de calcul est déconnecté, cf. Figure 3.9.
Le terme de calcul déconnecté indique ici qu'il n'y a pas communication entre les tâches effectuées par chaque processus parallèle (threads). Dans la pratique ce terme désigne alors tout problème pour lequel, durant le processus de calcul de chaque processus (dans notre cas l'intégration temporelle du système durant la rentrée à titre d'exemple), les communications entre les processus sont inexistantes et confinées hors de la zone de calcul. On estime que cette gamme de problèmes représente plus de 20% des applications parallèles [36].
Ce faible besoin de communications entre les processus rend les problèmes agréablement parallèles particulièrement adaptés à l'utilisation des accélérateurs de calculs conçus pour le calcul SIMD tels que les cartes GPU ou les Xeons Phi, qui comme présenté plus haut sont conçus pour exécuter en parallèle des tâches identiques (ou différentes jusqu'à un certains degrés de granularité) sur des données différentes avec un nombre limité de communication entre ces processus. Ainsi la Figure 3.9 présente une version schématisée d'une implémentation agréablement parallèles. Sur ce schéma on peut voir une série de processus parallèles : T0, T1 et T2, chacun disposant d'une donnée unique, qui vont ensuite subir le même traitement numérique selon le mode SIMD, sans aucune communication entre les processus. À noter que ce processus peut s'effectuer de façon synchrone ou asynchrone.
Dans le cadre des travaux présentés dans ce manuscrit, comme indiqué au Chapitre

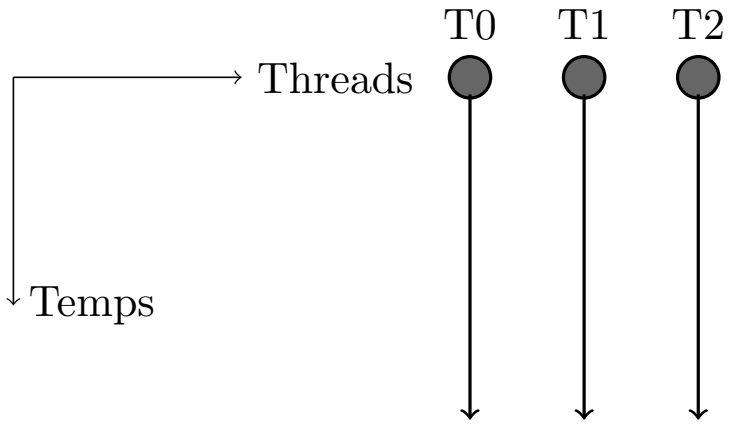

FIGURE 3.9: Illustration de problème agréablement parallèles

2, nous désignons par le nom méthode de Monte-Carlo, une méthode d'analyse statistique permettant d'évaluer la dispersion d'un résultat issu de l'intégration temporelle d'un modèle physique pour lequel nous disposons d'incertitudes sur certaines conditions initiales ou certains paramètres du modèle physique. Pour ce faire, nous créons donc une série de combinaisons de conditions initiales différentes, chaque combinaison ainsi engendrée aléatoirement définissant les conditions d'une simulation. Chaque simulation numérique est ensuite effectuée indépendamment des autres. L'ensemble des résultats de chaque simulation est alors post-traité afin de déterminer, dans le cas de la problématique qui nous occupe ici de la rentrée atmosphérique de satellites, des probabilités d'impact au sol, des zones probables de retombées dans le cas où l'engin ne serait pas détruit durant la phase de rentrée, etc...
Notre problème de Monte-Carlo ainsi posé on constate que ce dernier fait alors partie de la classe des problèmes agréablement parallèles telle que nous l'avons introduite ci dessus, et qu'il est alors tout à fait judicieux de considérer l'utilisation des accélérateurs de calculs pour le résoudre.

3.3.2 Contraintes de programmation sur accélérateur de calcul

Nous venons de le voir, les applications de type agréablement parallèles telles que celles considérées dans le cadre de ces travaux, possèdent toutes les propriétés nécessaires pour pouvoir profiter au maximum des capacités de calcul parallèle offertes par les accélérateurs de calculs de type GPU ou Xeon Phi.
Cela ne signifie pas pour autant qu'il suffit de porter une application de type agréablement parallèle sur ces accélérateurs pour obtenir de bonnes performances en terme de temps de calcul, bien au contraire. De part les spécificités hardware des accélérateurs, un portage hasardeux pourra entraîner des performances largement inférieures à celles d'une application séquentielle sur CPU, à cause des fréquences d'horloge plus basses sur les

accélérateurs, des temps d'accès mémoire entre l'accélérateur et le CPU très lents, etc... Nous allons lister ici certains des points les plus critiques à adresser afin d'optimiser les performances qu'il est possible de tirer des accélérateurs, et que l'on peut trouver dans les différents manuels consacrés à la programmation sur GPU et Xeon Phi [37], [28] et [50].

Certainement le point critique primordial à considérer lorsqu'on commence à déployer une application parallèle sur un accélérateur c'est l'occupation des cœurs de calcul de la carte. Pour les GPUs il faut ainsi s'assurer de déployer suffisamment de threads CUDA pour occuper tous les cœurs de la carte, tandis que sur le Phi en plus d'un déploiement parallèle important il faut aussi s'assurer d'utiliser pleinement les unités vectorielles de chaque cœurs. En effet, comme dit précédemment, qu'il s'agisse des cœurs CUDA ou des cœurs du Xeon Phi, ces derniers ont une fréquence d'horloge en moyenne trois fois plus basse que les CPUs actuels et on comprend bien si l'on ne sollicite pas suffisamment de cœurs sur les accélérateurs, on peut arriver à des configurations où moins d'opérations par secondes seront effectuées, par rapport à une exécution sur CPU.

En relation avec ce point, un autre point critique est la localité des données. Comme nous l'avons montré à la sous-section 3.2.1, que ce soit pour les cartes Xeon Phi ou pour les GPUs, les unités de mémoire sont dispersées sur les cartes, de telle façon que pour les deux architectures, les cœurs de calculs disposent d'une unité mémoire dédiée, en plus des unités de mémoire accessibles à tout ou partie des autres cœurs. Ces unités de mémoire sont là pour réduire au maximum le temps de latence du transfert des données entre la mémoire et l'unité de calcul. Mais pour cela il faut que les variables qui sont utilisées figurent dans ces unités de mémoire, et comme dis à la sous-section 3.2.2.2. cela passe par un effort de programmation.

Un autre point critique est la communication des données entre le CPU et l'accélérateur de calculs. Cette dernière s'effectue via le port PCI-Express et se trouve être très lente en comparaison au mouvements des données en mémoire au sein du CPU ou au sein des accélérateurs. Cette communication étant un goulot d'étranglement important lorsqu'on recherche les performances maximales, il est important de limiter le nombre de communications et d'optimiser les temps passé à effectuer des calculs sur l'accélérateur.

En général ces différentes recommandations sont illustrées sur des applications extrêmement simples à mettre en œuvre, et comportant peu d'opérations (type SAXPY, produit matrice-vecteur, etc...). C'est pour cette raison que nous présenterons au Chapitre 4 une évaluation du gain de performance que l'on peut obtenir en respectant ces points critiques sur une application concrète : la détermination des zones de retombées des ballons stratosphériques. Nous chercherons alors à analyser le rapport entre l'investissement, en temps et en travail, qu'implique le respect de ces directives et le gain en temps de calcul obtenu.

3.4 Conclusion

Dans ce chapitre nous avons procédé à une présentation des accélérateurs de calculs de type carte graphique et Xeon Phi, décrivant à la fois les architectures matérielles de ces appareils ainsi que les environnements logiciels et de programmation qui ont été développé par NVIDIA et Intel pour permettre aux programmeurs de les utiliser.
À cette occasion, nous avons mis en avant le fait que ces deux accélérateurs constituaient deux réponses différentes à une même problématique de calcul intensif : le GPU basant sa puissance de calcul sur un nombre important d'unités de calcul (les cœurs CUDA) privilégiant une approche massivement parallèle de type SIMT, tandis qu'à l'inverse le Xeon Phi base sa puissance de calcul sur ses unités vectorielles privilégiant une approche parallèle (puisqu'il supporte jusqu'à 244 processus parallèles) mais surtout massivement vectorielle avec ses unités vectorielles AVX-512 pour le calcul SIMD et qui sont à ce jour les unités vectorielles les plus grandes qui existent.
Ces accélérateurs de calculs sont une solution relativement jeune et donc susceptible d'évolutions. En effet comme évoqué, CUDA connaît périodiquement de nouvelles versions apportant chacune de nouvelles améliorations afin de faciliter la tâche du programmeur, de la même façon la technologie évolue elle aussi avec la sortie périodique de nouvelles architectures de cartes graphiques toujours plus performantes mais conservant toujours les grandes lignes présentées ici. De son côté le Phi ne connaît pas d'évolution aussi rapide en terme de programmation puisqu'il se base sur les grands langage de programmation parallèle que sont MPI et OpenMP, mais connaît des mises à jour annuelle de Intel MPSS, afin de traiter les problèmes de fonctionnement interne de la carte qui nuisent aux performances. Citons à titre d'exemple un problème rencontré au cours de nos expériences où l'on notait une explosion du temps de calcul de l'application qui après analyse s'est avérée être due à une surconsommation des ressources par les tâches systèmes pénalisant le calcul, et qui a été corrigée par l'installation d'une nouvelle MPSS.
Au niveau architecture le Phi évolue lui aussi, plus lentement que les GPUs, mais de façon plus radicale, la prochaine génération de Phi délaissant l'organisation des cœurs en double anneau au profit d'une architecture en grille [33].
Enfin nous avons mis en évidence comment les architectures de ces cartes accélératrices s'avèrent particulièrement bien adaptées à la résolution de problèmes agréablement parallèles tels que que les problèmes de Monte-Carlo que l'on cherche à traiter dans le cadre de ces travaux de thèse. Nous avons aussi pointé que bien que les architectures de ces cartes accélératrices se prêtent avantageusement à la mise en œuvre d'applications agréablement parallèles, l'obtention de bonnes performances en terme de temps de calcul passe obligatoirement par le respect de quelques règles de programmation dont nous discuterons de l'efficacité ainsi que de la difficulté à les mettre en œuvre au Chapitre 4.

Chapitre 4

Application à l'analyse de la dérive descente de ballons stratosphériques

DANS ce chapitre nous considérons la mise en œuvre d'un code de calcul parallèle sur accélérateur GPU et Xeon Phi, en considérant l'application de la retombée de ballon stratosphérique

4.1 Motivations et contexte

4.1.1 Motivations

Dans ce chapitre nous considérons un problème de trajectographie intéressant le CNES. Il s'agit de la dérive descente de ballon stratosphérique et tout particulièrement de la détermination des zones probables de retombées. Nous proposons ainsi une nouvelle méthodologie de détermination de ces zones basée sur une analyse statistique de type Monte-Carlo. Nous comparons aussi les bénéfices de l'utilisation des accélérateurs de calcul de type GPUs et Xeon Phi pour la résolution de cette analyse.
Cette étude permet de définir la faisabilité d'un outil de calcul statistique sur accélérateur et d'identifier en amont quelles peuvent être les difficultés liées à la réalisation de ce projet ainsi que les meilleures stratégies à adopter pour la conception d'un tel outil. Une telle étude se justifie par le manque de travaux dans la littérature comparant les performances d'applications accélérés sur GPUs et Xeon Phi. Les accélérateurs de calcul et notamment les GPU ont été utilisés avec grand succés à de très nombreuses classes de problèmes. Parmis les plus importantes citons l'algèbre linéaire [19],[85], le traitement de signal et de l'image ainsi que beaucoup d'autres problèmes : l'optimisation combinatoire [13], [43], la biologie, la physique [72], l'astophysique [24]... Néanmoins au moment du début de ces travaux seuls deux articles traitant de problèmes académiques (algèbre linéaires

et traitement d'images), figurent dans la litérature, avec des conclusions opposées ([82], [75]) et n'apportent que peu d'éléments utiles pour le traitement de notre application. L'intérêt de la problématique de la dérive descente des ballons stratosphérique réside dans la modélisation de ce phénomène avec les outils actuels qui apparaît comme une version simplifié des équations aérodynamiques utilisées pour la retombée des satellites. Ce qui permet d'avoir un modèle physique simple à traiter permettant de se concentrer sur les problématiques liées à la programmation sur les accélérateurs.
Cette étude de faisabilité a donc été entreprise avec trois objectifs en vue :

- Se familiariser avec les paradigmes de programmation de ces cartes accélératrices (CUDA pour les GPUs, la vectorisation SIMD pour les Xeon Phi).
- Évaluer les accélérations que l'on peut obtenir grâce à ces cartes accélératrices au vu de l'investissement en terme d'effort de programmation.
- Définir la meilleure stratégie à employer dans l'optique du développement du code de rentrée atmosphérique de satellite.

4.1.2 Contexte

Le début des années 60 voit le lancement du programme scientifique français de lancement de ballon. Trés rapidement ce programme va rejoindre le giron du CNES qui, durant les 50 années qui suivirent, devint le second plus gros lanceur de ballons stratosphériques (ballons volant dans la stratosphère, entre 15 et 50 km d'altitude) derrière les Etats Unis. Le CNES a ainsi lancé 3 600 ballons depuis environs 27 sites différents un peu partout sur le globe, depuis Kiruna (Suède) ou Timmins (Canada) jusqu'à la station McMurdo (Pôle Sud), en passant par Aire sur Adour (France), Mahé (Seychelles) ou Alice Springs (Australie).
Ce n'est pas un hasard si le CNES a autant développé son activité ballon. En effet grâce à une grande diversité de type de ballons (Ballon Stratosphérique Ouvert ou Pressurisé, Aeroclippers, Montgolfière Infra-Rouge,...) ces derniers permettent de réaliser un grand nombres de missions scientifiques très variées : météorologique (Concordiasi [67], Strapolété [69]), astrophysique (Pilot [47]) ou biologique. Les coûts d'une mission ballon sont très largement inférieurs à ceux d'un satellite, les conditions environnementales sont beaucoup plus clémentes, le matériel n'a pas à répondre à toutes les exigences de conformité qui sont imposées pour le matériel embarqué à bord de satellites, et comme à la fin de la mission, la charge scientifique est récupérée, celle-ci peut être réutilisée pour mener d'autres études, à un autre endroit. Dans le cadre de cette étude, nous ne considérerons que les ballons stratosphériques ouverts auquels nous ferons référence en tant que ballon stratosphérique. Ces ballons sont dotés d'enveloppes allant de 3 000 à 1 200 000 m3, épaisses de quelques microns (environs 0,015mm), pesant de 35 kg pour

les plus petits jusqu'à 840 kg pour les plus gros. Ces enveloppes sont ensuite remplies d'Hélium et peuvent porter des charger pouvant aller jusqu'à 3 tonnes, entre 15 et 40 km d'altitudes sur des vols durant de 7 heures à quelques jours. La durée du vol ainsi que l'altitude dépendent bien sûr de la dimension du ballon ainsi que du poids de la charge.

De façon générale on peut décomposer un vol de ballon pour une mission scientifique en 3 étapes. La première se nomme la phase de montée et correspond comme son nom l'indique à la phase de montée entre le lâché du ballon et son arrivée au plafond. On nomme plafond l'altitude à partir de laquelle le ballon atteint un équilibre hydrostatique (sa force de portance est égale à l'attraction terrestre). Vient ensuite la phase de vol au plafond, où le ballon vole à une altitude quasi-constante durant un certain temps (de quelques heures à plusieurs semaines ou mois selon les ballons). C'est en général durant cette phase que la mission scientifique s'effectue. Enfin, lorsque la mission scientifique a été effectué l'aérostat rentre alors dans la phase de fin de vol qui a pour objectif de ramener la charge scientifique au sol. Pour ce faire, on procède à la séparation de la chaîne de vol (qui contient la nacelle avec la charge utile plus les instruments scientifiques) de l'enveloppe du ballon. La chaîne de vol effectue alors une dérive descente jusqu'au sol sous un parachute, tandis que l'enveloppe est déchirée et retombe au sol après s'être plus ou moins roulée en boule.

Cette opération de séparation est loin d'être anodine, car une fois la chaîne de vol séparée de l'aérostat, les deux vont dériver dans le vent jusqu'au sol sans qu'il ne soit possible d'influencer de quelque façon que ce soit sur leurs trajectoires. C'est pour cette raison que la séparation ne peut être effectuée que si les opérateurs peuvent garantir que les deux éléments du ballon atterriront dans une zone dite sûre : avec une densité de population inférieure à une certaine valeur critique, sans voies de circulations importantes (routes, rails), ni de réseau électrique aérien ou de structures industrielles. Effectivement, comme dis plus haut, selon l'enveloppe et selon la mission, la chaîne de vol peut peser jusqu'à 3 tonnes et l'enveloppe plusieurs centaines de kilos. Il faut donc pouvoir garantir, avant d'entamer la dérive descente, qu'aucun de ces deux composants n'atterrissent dans une zone non sûre pour éviter qu'ils ne causent des dégâts matériels et/ou humains.

Notons que si à partir de la séparation les opérateurs n'ont plus aucun contrôle sur le mouvement de la chaîne de vol et de l'enveloppe, cela ne veut pas dire pour autant qu'ils disposent de moyen pour piloter précisément l'aérostat durant les phases précédentes de vol. Ils n'ont à leur disposition que deux commandes : une de dégazage et une de délestage, leur permettant de jouer avec l'altitude du ballon afin de prendre des vents, donc des directions, différentes.

4.1.2.1 Méthode CNES de détermination des zones de retombées sûres

Á l'heure actuelle pour garantir un atterrissage dans une zone sûre, la méthodologie employée par le CNES consiste à réaliser une simulation de dérive descente à partir du point de vol actuel, donnant une prédiction de la trajectoire de dérive descente de la chaîne de vol et de l'enveloppe ainsi que leurs points d'impact au sol, en utilisant un profil de vent prévisionnel. À partir du point d'impact de la chaîne de vol, les opérateurs météo du CNES déterminent une zone probable d'atterrissage que l'on nomme le quadrilatère ou polygone météo, qui est calculé via des formules qui ont été définies empiriquement en se basant sur l'expertise et la réanalyse des campagnes ballons précédentes. Si le polygone ainsi calculé ne recouvre aucune zone non sûre selon les critères évoqués plus haut, on considère alors que la séparation de la chaîne de vol et de l'enveloppe peut s'effectuer en toute sécurité. Dans le cas contraire le vol se poursuit jusqu'à ce que le ballon atteigne un point garantissant un atterrissage dans une zone sûre.

4.1.2.2 Nouvelle méthodologie

Dans le cadre de ces travaux nous proposons ainsi une nouvelle méthodologie de détermination des zones de retombées basées sur une analyse de Monte-Carlo de la dérive descente de l'enveloppe et de la chaîne de vol en prenant en considération les incertitudes sur le profil météorologique, issues de la prévision, employé pour effectuer la simulation. Au niveau de la modélisation du processus de dérive descente notre approche se conformera aux équations considérées par le CNES dans leur outil actuel. Nous proposons aussi de tirer parti des accélérateurs de calculs modernes de type GPUs ou Xeon Phi pour effectuer cette analyse.

Notre approche d'une analyse de Monte-Carlo sur accélérateurs, connectés à une unité centrale standard, est tout à fait pertinente dans le cadre des opérations ballons qui ont généralement lieu dans des lieux reculés tels que Timmins au Canada ou Kiruna en Suède, avec des moyens de calculs et des accès internets restreints (difficulté de se connecter à un cluster distant et impossibilité d'en embarquer un pour la mission pour des raisons logistiques évidentes). Un autre point à prendre en considération est l'aspect temps réel, puisque le ballon est en vol durant la simulation de Monte-Carlo. Il faut donc que cette dernière livre un résultat en des temps acceptables sous peine de quoi le résultat produit n'aura aucune valeur, l'aérostat ayant poursuivit sa trajectoire en vol et ayant dépassé de plusieurs kilomètres le point garantissant un atterrissage dans une zone sûre. Cette contrainte supplémentaire pousse donc à chercher la meilleure stratégie et la meilleure optimisation possible pour le code afin de maximiser les performances des accélérateurs de calcul.

4.2 Modélisation du phénomène de dérive descente de ballon stratosphérique

Une fois la séparation effectuée, la chaîne de vol sous parachute et l'enveloppe du ballon sont soumises aux mêmes forces (la chaîne de vol sous parachute est assimilée à une masse ponctuelle). Le mouvement vertical est ainsi dirigé par la somme des forces de gravitation et de traînée tandis que le mouvement horizontal est lui dirigé par le vent. Conformément à la modélisation en cours dans l'outil CNES de référence, on décrit le mouvement de l'enveloppe et de la chaîne de vol sous parachute dans un référentiel terrestre $(0, \vec{x}, \vec{y}, \vec{z})$ que l'on considère comme Galiléen sur l'échelle de temps de la simulation. L'axe $\vec{z}$ est orienté selon la verticale locale, les axes $\vec{x}$ et $\vec{y}$ sont quand à eux orientés respectivement dans la direction Est et Nord. Afin de modéliser numériquement ce mouvement et effectuer l'analyse de Monte-Carlo on procède selon l'algorithme présenté à la Figure 4.1. Cet algorithme présente le fonctionnement du code de dérive descente séquentiel afin d'introduire les différents éléments de la modélisation. L'implémentation parallèle sur accélérateur de ce code est présenté à la section 4.3.

4.2.1 Mise à jour des paramètres atmosphériques et du coefficient balistique

Dans un premier temps on charge un profil météorologique. Il s'agit d'une coupe verticale de l'atmosphère, échantillonnée à différentes altitudes et contenant pour chaque niveau les valeurs des différents paramètres atmosphériques : pression, température, force et direction du vent. Ces profils verticaux sont soit des prévisions issues de modèles météorologiques (prévision Météo France par exemple), soit des relevés in-situ réalisés par des sondes météorologiques qui échantillonnent toutes les x secondes ces paramètres durant leurs ascensions, lorsqu'elles sont accrochées sous des ballons sondes, ou durant leurs chutes, dans le cas des drop sondes.
Ce profil météorologique étant donc échantillonné en altitude, on doit calculer à chaque pas de temps la valeur de ces paramètres à l'altitude courante de la simulation par interpolation.
On calcule ainsi $T(z)$, $P(z)$, $U(z)$ et $V(z)$ respectivement la température, la pression, la composante zonale (le long de l'axe Est-Ouest, positive vers l'Est) et la composante méridionale (le long de l'axe Nord-Sud) du vent à l'altitude courante z via :

$$T(z) = T[n] + \frac{T[n] - T[n-1]}{z[n] - z[n-1]}(z - z[n]), \tag{4.1}$$

$$ln(P(z)) = ln(P[n]) + \frac{ln(P[n]) - ln(P[n-1])}{z[n] - z[n-1]}(z - z[n]), \tag{4.2}$$

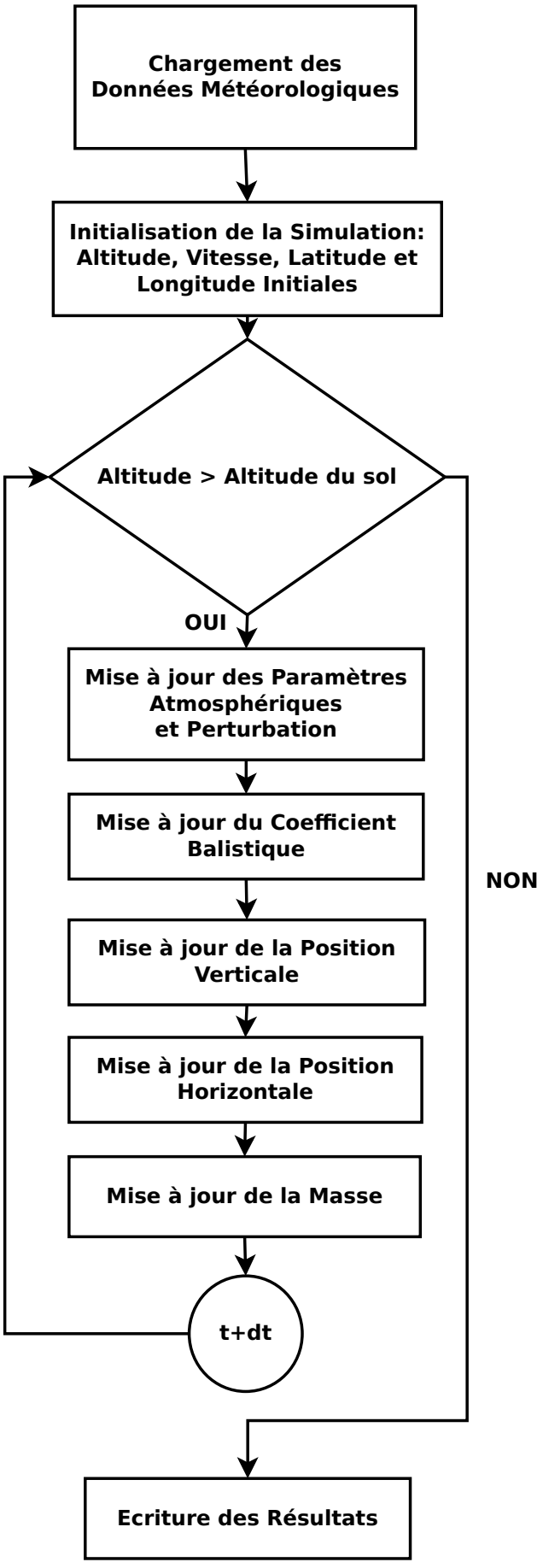

FIGURE 4.1: Algorithme séquentiel de la simulation du mouvement de dérive descente de la chaîne de vol sous parachute ou de l'enveloppe

$$U(z) = U[n] + \frac{U[n] - U[n-1]}{z[n] - z[n-1]}(z - z[n]), \tag{4.3}$$

$$V(z) = V[n] + \frac{V[n] - V[n-1]}{z[n] - z[n-1]}(z - z[n]), \tag{4.4}$$

avec n l'indice correspondant au niveau d'altitude supérieur par rapport à l'altitude courante ($z[n] > z$) et $n-1$ correspondant au niveau d'altitude inférieur ($z[n-1] < z$). Outre les paramètres météorologiques, il faut aussi mettre à jour le coefficient balistique à chaque pas de temps. Pour la chaîne de vol sous parachute, cette mise à jour correspond à l'ouverture progressive du parachute au début de la dérive descente, ainsi qu'à la réduction progressive de la masse de la chaîne de vol dûe à l'ouverture des bacs à lest au moment de la séparation. Pour l'enveloppe, cette mise à jour correspond à l'évolution de la forme de l'enveloppe déchirée au cours de la dérive descente qui passe d'un objet informe avec une traînée importante à une boule avec une traînée très faible.
Ces processus étant complexes et soumis à beaucoup de paramètres inconnus il n'existe pas de modèles permettant de donner l'évolution de ce coefficient. On utilise donc des tables d'évolutions en fonction du temps du coefficient balistique dans lesquelles on vient interpoler la valeur du coefficient à l'instant courant $B(t)$:

$$B(t) = B[n-1] + \frac{B[n] - B[n-1]}{t[n] - t[n-1]}(t - t[n-1]), \tag{4.5}$$

où n est l'indice dans la table d'évolution tel que $t[n-1] < t$ et $t[n+1] > t$.

4.2.2 Perturbation des paramètres météorologiques

Une fois les paramètres météorologiques calculés à chaque pas de temps par interpolation, la simulation applique ensuite une perturbation sur chacun de ces paramètres afin d'engendrer un profil unique pour la simulation en cours. La perturbation est donnée par l'équation (4.6) suivante :

$$Paramètre_{Perturbé} = Paramètre_{Initial} \pm \alpha_{Paramètre}\sigma_{Paramètre}, \tag{4.6}$$

avec $\alpha_{Paramètre}$ une variable aléatoire de distribution normale. Cette valeur est tirée au début de chaque simulation de l'analyse Monte-Carlo, et est différente pour chaque paramètre météorologique. De la même façon que $\alpha_{Paramètre}$, le signe de la perturbation est déterminé de façon aléatoire au début de chaque simulation de l'analyse, pour chaque paramètre. $\sigma_{Paramètre}$ correspond à l'écart type pour un des paramètres atmosphériques entre les valeurs de ce paramètre issues de la prévision météorologique et les valeurs réelles rencontrées par l'aérostat durant sa phase de montée. À l'heure actuelle, cette

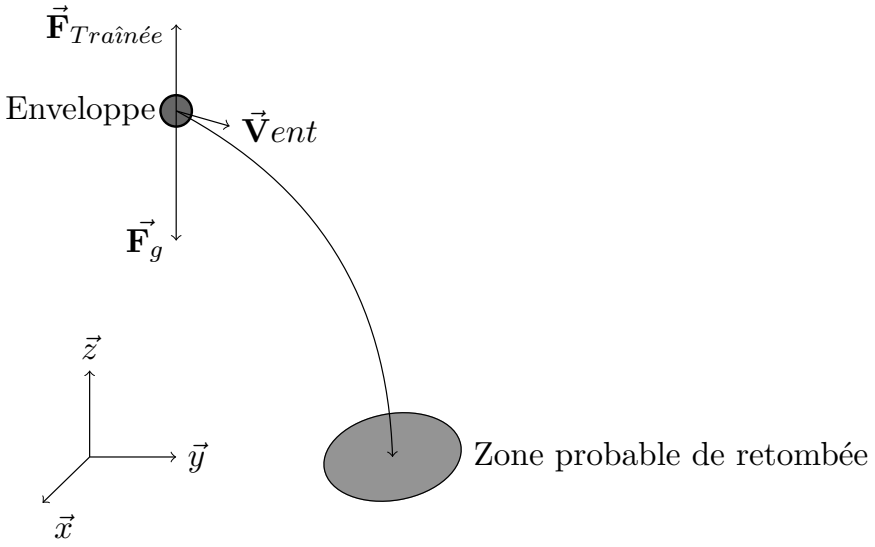

FIGURE 4.2: Schéma du mouvement de dérive descente de l'enveloppe

méthode est la seule envisagable compte tenu des données disponibles au moment de la préparation de la séparation du vol.

4.2.3 Modélisation physique de la dérive descente de l'enveloppe

Dans le modèle de la dérive descente de l'enveloppe, la position de l'enveloppe est définie dans le référentiel terrestre (cf. Figure 4.2) à chaque pas de temps t par son altitude z(t), sa latitude lat(t) et sa longitude lon(t). Dans ce système de coordonnées le mouvement de l'enveloppe peut alors être décomposé en deux composantes indépendantes : le mouvement vertical et le mouvement horizontal.

4.2.3.1 Mouvement vertical de l'enveloppe

Le mouvement vertical de l'enveloppe est le résultat de la somme de la force de gravitation $\vec{F_g}$, qui attire l'enveloppe vers le sol, et de la force de traînée $\vec{F}_{Traînée}$, cf. Figure 4.2. D'après la deuxième loi de Newton on calcule l'accélération verticale de l'enveloppe à l'instant t et à l'altitude z a(t,z), selon l'équation suivante :

$$\begin{aligned} a(t,z) = \frac{dv}{dt} &= \frac{F_{Traînée}}{m} - F_g, \\ &= \frac{-1}{2}\rho(z)v(t)|v(t)|B(t) - g, \end{aligned} \tag{4.7}$$

avec $a(t,z)$ l'accélération en $m.s^{-2}$; $v(t)$ la composante verticale de la vitesse verticale de l'enveloppe à l'instant t en $m.s^{-1}$ tel que :

$$v = \frac{dz}{dt}, \tag{4.8}$$

$B(t)$ est le coefficient balistique de l'enveloppe à l'instant t, g l'accélération gravitationnelle terrestre et $\rho(z)$ est la densité de l'atmosphère à l'altitude z, calculée à partir de la loi des gaz parfaits :

$$\rho(z) = \frac{0,029P(z)}{8,314T(z)}, \tag{4.9}$$

Dans le code, la mise à jour de la position verticale de l'enveloppe est effectué en intégrant par la méthode d'Euler du système d'équations différentielles (4.7) et (4.8). Ici l'utilisation de la méthode d'Euler pour l'intégration du système se justifie par la linéarité du modèle qui permet une intégration avec un pas de temps de 0.1s sans instabilité numérique.

4.2.3.2 Mouvement horizontal de l'enveloppe

Le mouvement horizontal de l'enveloppe durant la phase de dérive descente ne dépend que de la force et de la direction du vent. La force de Coriolis est négligée. Pour pouvoir déterminer l'équation du mouvement horizontal de l'enveloppe on définit alors U la vitesse du vent suivant sa composante zonale (positive vers l'Est) et V la vitesse du vent suivant sa composante méridionale (positive vers le Nord), en $m.s^{-1}$.
À chaque pas de temps, la position horizontale de l'enveloppe en latitude et longitude est calculé en intégrant par la méthode d'Euler les équations (4.10) et (4.11) en respectant l'hypothèse d'une Terre sphérique conformément à la modélisation implémentée dans l'outil CNES de référence :

$$lat(t+dt) = lat(t) + \arcsin(\frac{V(z)dt}{R_T + z}), \tag{4.10}$$

$$lon(t+dt) = lon(t) + \arcsin(\frac{U(z)dt}{(R_T + z)\cos(lat(t))}), \tag{4.11}$$

avec R_T le rayon terrestre en mètre et z la position verticale dans le référentiel local.

4.2.4 Modélisation de la dérive descente de la chaîne de vol sous parachute

Pour la modélisation du mouvement de la chaîne de vol sous parachute, on assimile l'ensemble à une masse ponctuelle égale à la somme des composants de la chaîne de vol et des parachutes (Figure 4.3). Dans cette étude, on néglige la liaison élastique entre la chaîne de vol et les parachutes.
Comme on peut le constater sur la Figure 4.3, dans le cas de la descente de la chaîne de vol sous parachute la composante de la force de traînée n'est plus alignée avec la force de gravitation prévenant la décomposition du mouvement en deux composantes

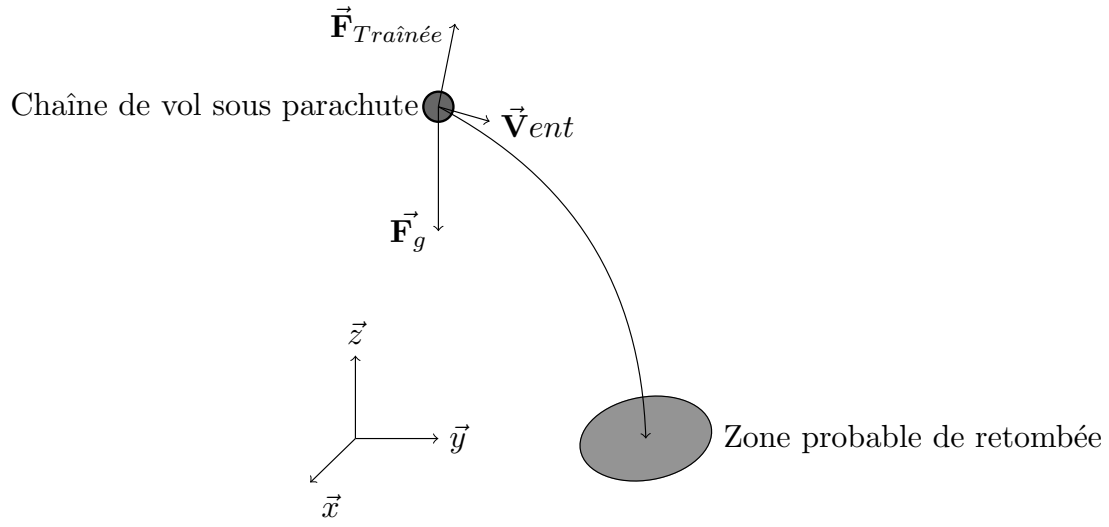

FIGURE 4.3: Schéma du mouvement de dérive descente de l'ensemble chaîne de vol sous parachute

indépendantes comme cela a été fait pour le cas de l'enveloppe.
On définit v_t la vitesse relative de l'ensemble par rapport au vent de la manière suivante :

$$v_t = [(v_x - U)\cos\alpha + (v_y - V)\sin\alpha]\sin\theta + v_z\cos\theta, \tag{4.12}$$

avec v_x, v_z et v_z la vitesse de l'ensemble dans le référentiel $(O, \vec{x}, \vec{y}, \vec{z})$ et α et θ tels que :

$$\alpha = \arctan(\frac{v_y - V}{v_x - U}), \tag{4.13}$$

$$\theta = \arctan(\frac{\sqrt{(v_x - U)^2 + (v_y - V)^2}}{-v_z}), \tag{4.14}$$

D'après la seconde loi de Newton on obtient alors l'accélération de l'ensemble chaîne de vol sous parachute à l'instant t, et à l'altitude z :

$$\vec{a}(z,t) = \frac{dv}{dt} = \frac{\vec{F}_{Traînée}}{M} + \vec{g} = \frac{-1}{2}\rho(z)\vec{v}(t)|\vec{v}(t)|\frac{C_tS}{M} + \vec{g}, \tag{4.15}$$

où v est la vitesse de l'ensemble chaîne de vol sous parachute, g est l'accélération gravitationnelle terrestre, C_tS est la surface de traînée des parachutes et M est la masse de l'ensemble.
Ainsi on calcule la position de l'ensemble dans le référentiel terrestre à chaque pas de temps en intégrant par la méthode de Runge-Kutta 4 les équations (4.16), (4.17) et

(4.18) suivantes :

$$\frac{dv_x}{dt} = \frac{d^2x}{dt^2} = \frac{F_{Traînée} \sin\theta \cos\alpha}{M}, \tag{4.16}$$

$$\frac{dv_y}{dt} = \frac{d^2y}{dt^2} = \frac{F_{Traînée} \sin\theta \sin\alpha}{M}, \tag{4.17}$$

$$\frac{dv_z}{dt} = \frac{d^2z}{dt^2} = \frac{F_{Traînée} \cos\theta}{M} - g, \tag{4.18}$$

4.2.5 Les polygones de dérive descente météo

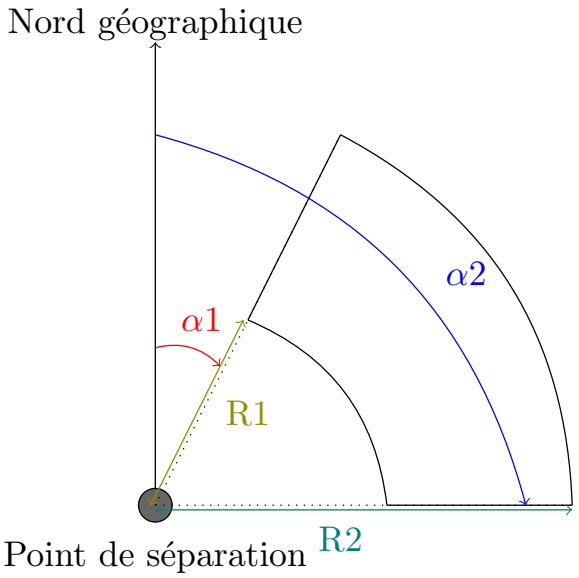

FIGURE 4.4: Schéma du polygone dérive descente météo

Comme évoqué précédemment afin d'identifier si une dérive descente va mener une chaîne de vol dans une zone sûre ou non, le CNES utilise des polygones de dérive descente météo. Il s'agit d'un quadrilatère défini à partir du point de séparation de l'enveloppe et de la chaîne de vol, d'un angle minimal α_1, d'un angle maximal α_2 par rapport au Nord géographique, d'un rayon minimal $R1$ et d'un rayon maximal $R2$. Le quadrilatère ainsi défini constitue ce que l'on appelle le polygone de dérive descente Z1 comme illustré à la Figure 4.4. Le point théorique où la chaîne de vol est supposée atterrir après sa dérive descente sous parachute se situe au centre de cette zone Z1. La zone d'atterrissage de l'enveloppe est ensuite déduite à partir de cette zone Z1 en augmentant l'ouverture des angles α_1 et α_2 , ainsi que R2, et en diminuant R1.
La zone Z1 est alors assimilée à la zone où la probabilité que la chaîne de vol atterrisse

est la plus élevée. Deux autres zones Z2 et Z3 sont définies en augmentant progressivement l'ouverture entre les angles α_1 et α_2 (réduisant α_1 et augmentant α_2) et la distance entre R1 et R2, tout en restant centré sur le point de retombée théorique.
Les formules permettant de calculer ces différentes zones font parties de l'expertise des opérateurs du CNES. Elles ont été définies de telle façon que 99,7% des points d'atterrissage réel des vols des campagnes précédentes figurent dans la zone Z3. Bien que calculés de façon empiriques, ces polygones ont ainsi une fonction de zone de confiance à 1, 2 et 3 σ.

4.3 Estimation des performances

À la sous-section 3.3.2, nous avons mis en évidence l'adéquation qui existe entre la résolution des problèmes de Monte-Carlo tels que nous les concevons dans nos expériences et l'utilisation des accélérateurs de calculs. En particulier, nous avons montré comment l'architecture de ces accélérateurs était pensée pour la mise en œuvre de problèmes agréablement parallèles. Nous avons aussi mis en avant un ensemble de points critiques qu'il fallait traiter lors du portage d'une application agréablement parallèle sur carte accélératrice afin de maximiser les performances en terme de temps de calcul. Cependant dans la littérature ces recommandations sont accompagnées de gains mais uniquement pour des applications "simples", comme par exemple l'algèbre linéaire (produit ou inversion de matrices,...).
Dans cette section nous allons évaluer les différentes performances que l'on obtient en fonction du degré de respect de ces règles de programmation pour une application issue d'un problème réel (qui n'est pas purement académique). Nous tenterons d'évaluer de façon qualitative le rapport entre le gain obtenu et l'investissement de travail nécessaire pour obtenir ces gains dans une logique de viabilité pour un industriel : "Est-ce que le gain obtenu vaut le temps que va passer un ingénieur sur le code ?" Nous complèterons cette analyse avec d'autres considérations plus techniques tels que les coûts comparés entre les différentes plate-formes (achats, consommation électriques,...).
Pour ce faire nous allons traiter exclusivement le cas de la dérive descente enveloppe celle-ci ayant moins de variables et d'opérations que la dérive descente sous parachute et étant donc la plus propice à des réécritures successives de l'algorithme et nous comparerons différentes versions des implémentations de l'algorithme présenté Figure 4.1 sur le GPU et le Xeon Phi en utilisant comme base de référence une mise en œuvre de cet algorithme sur un nœud de cluster.
Nous détaillerons les étapes successives de l'optimisation du code à partir d'une mise en œuvre de l'application sur les accélérateurs GPU et Phi, qui s'inspire des différentes librairies que l'on peut trouver à l'heure actuelle pour l'intégration de système différentiels parallèles et indépendants, que nous allons progressivement optimiser, évaluant à chaque

TABLE 4.1: Description des caractéristiques des composants de test

Nom	Architecture	Cœurs	Horloge (GHz)	Mémoire totale (MB)	Bande passante mémoire max (GB/s)	Unité vectorielle	Compute capability
K40	Kepler	2 880	0,745	12 000	288	N.A.	3,5
K80	Kepler	4 992	0,875	24 000	480 (240/GPU)	N.A.	3,7
Xeon Phi 7120P	N.A.	61	1,24	16 000	352	AVX 512 (512-bit SIMD)	N.A.
Xeon E5-2680-v2	N.A.	20	2,8	25 (cache/proc)	59,7	AVX (256-bit SIMD)	N.A.

étape le gain en temps de calcul à la lumière de l'effort de programmation mis en œuvre. Nous terminerons cette section en dressant un bilan de cette étude.

4.3.1 Protocole expérimental

Dans cette sous-section nous allons présenter l'ensemble des plate-formes de test que nous avons utilisées pour mener cette étude ainsi que leurs configurations. Nous présenterons ensuite la démarche suivie, pour l'ensemble des expériences présentées dans ce chapitre, pour mesurer les temps d'exécutions des différentes itérations du code de calcul de dérive descente.
Les données utilisées pour le calcul des simulations sont données en Annexe C.

4.3.1.1 Plate-forme de tests

Pour la réalisation de ces tests nous avons utilisé trois systèmes de calcul détaillés dans le Tableau 4.1 :

- Un nœud de cluster composé de deux Intel Xeon E5-2680 v2 à 2,8GHz chacun disposant de 10 cœurs physiques et d'un Intel Xeon Phi 7120P, avec 61 cœurs cadencés à 1,238 GHz et embarquant 16GB de mémoire.
- Une station de travail équipée d'un Intel Xeon E5640 à 2,67GHz disposant de 4 cœurs et d'une carte NVIDIA Tesla K40 avec 2 880 cœurs de calculs à 0,745GHz et 12GB de mémoire.
- Un nœud sur un cluster de la société NVIDIA, équipé d'une carte Tesla K80 avec 4 992 cœurs de calculs CUDA à 0,875GHz et 24GB de mémoire.

La mise en œuvre du code sur GPU est effectuée en utilisant CUDA 7 [49]. La mise en œuvre des codes parallèles sur Xeon et Xeon Phi est effectuée en utilisant OpenMP 4.0 et compilée avec Intel Compiler 14.0.1 de la distribution Intel Parallel Studio XE 2015.

4.3.1.2 Protocole expérimental

Les temps de calculs présentés dans les sous-sections suivantes correspondent à la moyenne des temps mesurés sur 10 instances de calculs afin de lisser les éventuelles fluctuations

TABLE 4.2: Temps de calcul pour 100 800 simulations sur GPU

Implémentation parallèle	Temps total (s)	Temps de communications (s)	Temps de calcul (s)
Parallélisme de boucle	21,49	17,39	4,1
Parallélisme de tâche	3,03	0,12	2,91
Parallélisme de tâche et gestion mémoire	2,79	0,11	2,68

liées à l'occupation de la machine hôte. Ces temps de calcul sont mesurés entre le moment où le code commence à charger la première donnée météo et celui où le résultat de la dernière simulation est traitée. Tous les calculs sont effectués en virgule flottante double précision, et pour chaque machine on utilise toujours le nombre maximum de cœurs disponibles (sur le CPU, la K40, K80 ou le Phi)

4.3.2 Évaluation des performances sur les GPUs

Dans cette sous-section nous présentons les différentes mises en œuvre de l'algorithme de dérive descente enveloppe présenté Figure 4.1, sur GPU.

4.3.2.1 L'approche parallélisme de boucle

Pour la première mise en œuvre parallèle sur GPU de l'algorithme de dérive descente, nous avons tout d'abord considéré l'utilisation d'une bibliothèque de calcul afin de démarrer avec une version nécessitant peu d'investissement en terme de développement. Nous nous sommes ainsi intéressés à la bibliothèque Odeint [38] qui est une bibliothèque en C++ pour l'intégration numérique de système d'équations différentielles. Au moment de ces travaux il s'agissait alors de la seule bibliothèque existante (et à notre connaissance c'est toujours le cas aujourd'hui) permettant l'intégration d'équation différentielle sur GPU. Les différentes fonctions d'intégrations d'équations différentielles, que l'on nomme aussi les solveurs, de cette bibliothèque ont un mode de fonctionnement simple. Le solveur différentiel est appellé depuis la section séquentielle sur CPU du code. Ce dernier accepte comme argument le système différentiel à intégrer ainsi que la liste des variables et le nombre de pas d'intégration. La bibliothèque se charge ensuite de distribuer automatiquement les calculs sur les cœurs du GPU, d'intégrer tous les calculs en restant sur le GPU tant que l'intégration n'est pas finie, puis de collecter tous les résultats pour les restituer à la fin de l'intégration. Dans le cas de notre algorithme, avec Odeint, il n'est pas possible d'effectuer tout le calcul sur le GPU, puisque à chaque pas de temps il faut mettre à jour les valeurs des paramètres météorologique et le coefficient balistique. Avec ce mode de fonctionnement cela nous oblige à transférer les informations vers le CPU entre deux itérations, adoptant ainsi un modèle de parallélisme de boucle, tel que présenté dans la Figure 4.5.
En fin de compte il s'est avéré qu'en raison de limitations techniques inhérentes à la bibliothèque Odeint, qui ne permet de passer qu'un nombre limité d'arguments au solveur,

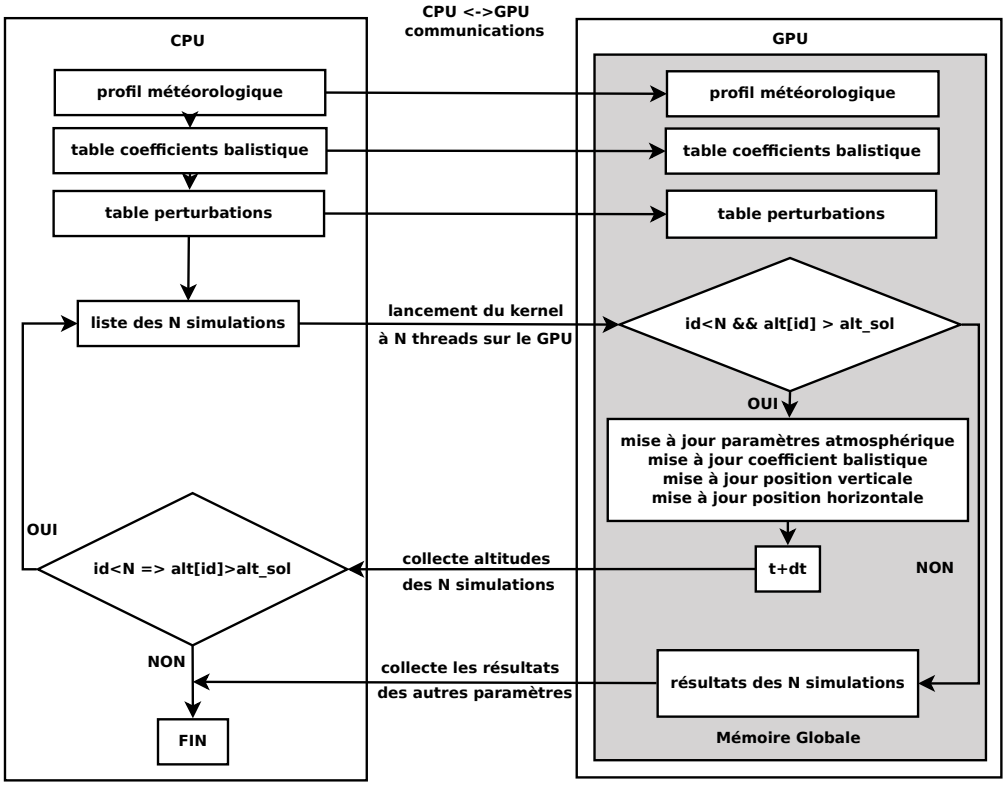

FIGURE 4.5: Algorithme a parallélisme de boucle sur le GPU

inférieur au nombre d'arguments nécessaires à l'intégration du système différentiel de la dérive descente, il n'a pas été possible utiliser Odeint pour implémenter l'algorithme de dérive descente. Nous sommes donc partis de la structure algorithmique qu'impliquait l'utilisation de cette bibliothèque, pour développer notre propre solution de solveur de système à équation différentielle. C'est ainsi qu'a été mis en œuvre l'algorithme présenté Figure 4.5.

Le code commence donc sur le CPU en chargeant le profil météorologique à utiliser pour la simulation ainsi que la table de variation temporelle du coefficient balistique, qu'il copie ensuite dans la mémoire globale de la carte graphique. Les données du profil météorologique ainsi que la table d'évolution du coefficient balistique sont alors organisées dans des vecteurs. Dans un second temps, en fonction du nombre N de simulations à effectuer, le code génère sur le CPU une table de coefficients de perturbations pour chaque grandeur météorologique et pour chaque simulations, tel que défini à la sous-section 4.2.2. On obtient ainsi une table ayant autant de colonnes que de coefficients de perturbations (chaque coefficient est assigné à une colonne et cette assignation est toujours la même) et ayant autant de lignes que de simulations à calculer (N lignes). Cette table est ensuite à son tour copiée dans la mémoire globale de la carte graphique. Le code CPU lance ensuite le kernel sur le GPU qui va effectuer une itération de la boucle 'while' sur l'altitude de l'algorithme présentée Figure 4.1. La grille de calcul de ce kernel est automatiquement dimensionnée en fonction de la carte graphique connectée

à la machine hôte de telle façon qu'un bloc contienne autant de threads que la carte le permet, et on lance autant de blocs que nécessaire pour calculer les N simulations demandées par l'utilisateur. Chaque thread traite ainsi une seule simulation, identifiée par l'identifiant du thread sur la grille de calcul *id*, défini tel que :

$$id = blockDim.x * blockIdx.x + threadIdx.x, \tag{4.19}$$

Avec $blockDim.x$ le nombre de threads par bloc, $blockIdx.x$ l'identifiant du bloc sur la grille de calcul et $threadIdx.x$ l'identifiant du thread dans le bloc.
À partir de cet identifiant, un thread donné accède à la ligne correspondante dans le tableau de perturbation, permettant d'assurer que chaque thread perturbe de façon unique les données météorologiques assurant l'unicité de la solution de chaque thread. En effet grâce à cette méthode de perturbation chaque thread utilise un profil météorologique unique.
Concrètement, un thread commence ainsi par interpoler les valeurs des paramètres atmosphériques dans le profil à partir de son altitude courante et des équations (4.1), (4.2), (4.3) et (4.4), qu'il perturbe ensuite à partir des coefficients de perturbations récupérés via la méthode présentée ci-dessus et de l'équation (4.6). Le coefficient balistique est ensuite calculé à partir de l'équation (4.5). Ces calculs sont effectués sur le GPU et non sur le CPU afin d'optimiser le temps de calcul, ce traitement se faisant de façon indépendante pour chaque simulation et se parallélisant de la même façon que le calcul de l'évolution temporelle du système. De plus en effectuant ces calculs sur le GPU on restreint la quantité de données à transférer du CPU vers le GPU à chaque itération.
Ensuite par la méthode d'Euler, le thread met à jour la position en altitude de l'enveloppe en intégrant l'équation (4.7) puis la position en latitude et en longitude en intégrant les équations (4.10) et (4.11).
Une fois que tous les threads ont effectué cette série d'opérations, l'ensemble des positions en altitude de chaque simulation est stocké dans un vecteur qui est copié sur le CPU et le kernel prend fin. L'exécution du programme se poursuit alors sur le CPU en contrôlant pour chaque simulation si l'altitude de cette dernière est supérieure à celle du sol. Tant que l'altitude d'une simulation est supérieure à l'altitude du sol (alt_{sol}), un nouveau kernel est relancé afin d'effectuer une nouvelle itération tant qu'il subsiste des simulations avec des altitudes supérieures à celle du sol. À ce stade seul le vecteur contenant les altitudes de toutes les simulations transite du GPU vers le CPU afin de limiter la quantité de données transférées. À noter qu'un thread n'intègre le système que si l'altitude de ce thread est supérieure à l'altitude du sol. Grâce à cela on ne risque pas d'avoir des simulations avec une altitude inférieure au niveau du sol lorsque toutes les simulations auront fini d'être intégrées. En effet si au cours de la dernière itération de calcul l'altitude d'une simulation passe sous le niveau du sol on conserve la valeur

précédente.
Toutes les variables de calcul (altitude, latitude, longitude, vitesse, etc...) sont stockées dans la mémoire globale de la carte graphique afin d'assurer leurs persistances d'un kernel à l'autre, puisque les données qui sont stockées dans la mémoire locale d'un thread, ou dans la mémoire partagée d'un bloc sont supprimées lors de l'interruption du kernel, tandis que les données dans la mémoire globale persistent tant que l'exécution du code sur le CPU n'est pas interrompue.
Lorsque toutes les simulations ont été effectuées jusqu'à ce que leur altitude soit inférieure ou égale à l'altitude du sol, les résultats de toutes les simulations (position latitude-longitude, vitesse d'impact, etc...) sont ensuite copiés sur le CPU où ils sont stockés dans un fichier pour post-traitement.
Cet algorithme avec un kernel n'effectuant en parallèle qu'une itération à la fois de la boucle de calcul, référencé sous le nom Parallélisme de boucle dans le Tableau 4.2, permet de calculer 100 800 simulations (3 150 warps de 32 threads) de dérive descente enveloppe en 21,49 s.

4.3.2.2 Gestion des communications sur le GPU

Lorsqu'on analyse dans le détail les performances de cette première implémentation de l'algorithme par parallélisme de boucle à l'aide de l'outil de profilage nvprof de NVIDIA, on constate que près de 81% du temps d'exécution est passé à copier des données entre le CPU et le GPU, alors que le temps de calcul effectif (celui passé à réaliser des opérations arithmétiques) représente seulement 19% du temps d'exécution, cf. Table 4.2. On est ici clairement confronté au problème de gestion des communications entre le CPU et le GPU évoqué à la sous-section 3.3.2, la question de l'occupation des cœurs de calculs étant naturellement réglée par le nombre important de simulations à effectuer. Avec cet algorithme on dénombre ainsi en moyenne 38 079 communications entre le CPU et le GPU.
Ce nombre de communications important venant de la stratégie de parallélisme de boucle mis en œuvre, nous avons ainsi retravaillé l'algorithme pour passer d'une approche parallélisme de boucle, à une approche parallélisme de tâche. Concrètement cela se traduit par le fait qu'un thread, au lieu de ne faire qu'une itération de la boucle de l'algorithme de la Figure 4.1, effectue à présent l'intégralité de la tâche c'est à dire toute la boucle 'while' sur l'altitude avant de s'interrompre. Lorsque tous les threads, c'est à dire toutes les simulations, ont fini leurs calculs, les résultats sont stockés dans la mémoire globale avant d'être transférés au CPU. Grâce à cette nouvelle approche le nombre de communications tombe à trois : deux en début de simulations pour envoyer les données initiales de simulations (météo, coefficient balistique) et une dernière en fin pour rapatrier les résultats sur le CPU.

Toujours dans cette logique de réduction des communications, on ne crée plus la grille de coefficients de perturbations sur le CPU pour la transférer sur le GPU. À la place chaque thread va engendrer ses propres coefficients en utilisant la librairie de génération de nombre aléatoire sur GPU cuRAND.
En terme d'effort de programmation pour migrer de l'approche parallélisme de boucle à parallélisme de tâche, le coût est assez réduit. En effet, cela consiste principalement à faire migrer la boucle 'while' d'intégration temporelle du CPU vers le GPU, autrement dit à déplacer 3 lignes de codes, et à retirer la boucle qui testait si toutes les simulations avaient atteintes l'altitude du sol, donc très peu de modifications de code. Le plus gros investissement étant le remplacement de la méthode de gestion des coefficients de perturbation par génération d'une table contenant tous les coefficients pour toutes les simulations sur le CPU, à une génération des coefficients par threads. Au final la solution mise en œuvre s'avère plus simple et intuitive que la première méthode, car ne demandant plus de gérer de grosses tables, en jouant sur les indices en fonction de l'identifiant de la thread et le coefficient considéré, seulement la gestion d'une graine unique par thread.
En revanche si l'on regarde le gain en temps de calcul, on constate dans la Table 4.2 que ce nouvel algorithme nommé Parallélisme de tâche, résout à présent les 100 800 simulations en 3,03 s, soit une réduction du temps d'exécution de 85,9%. Ce gain est particulièrement important surtout si l'on considère les efforts de programmation qui ont été mis en œuvre et qui n'ont au final demandé que peu de ré-écriture de code.
Si l'on regarde le détail de ce nouveau temps de calcul on note que le temps de communication des données entre le CPU et le GPU est à présent quasi négligeable ne représentant que 0,04% du temps d'exécution total. Il est intéressant de noter ensuite que le temps d'exécution du kernel a lui aussi été réduit en changeant d'algorithme, passant de 4,1s à 2,91s. Cette réduction s'explique par le fait qu'avec le nouvel algorithme on n'instancie plus qu'un unique kernel, alors qu'avec l'ancien on en instanciait 19 039, chaque instanciation ayant un coût en temps de calcul.

4.3.2.3 Gestion de la mémoire sur le GPU

La localité des données en mémoire sur le GPU n'a pas été traité jusqu'à présent, toutes les variables évoquées dans les deux précédentes mises en œuvre étant stockées dans la mémoire globale de la carte.
Dans cette sous-section nous cherchons à tirer parti de la mémoire locale afin de minimiser la latence et de maximiser la bande passante. Le kernel de la sous-section précédente est ainsi modifié de façon à ce que chaque thread effectue une copie dans sa mémoire locale du profil de donnée météorologique ainsi que de la table d'évolution du coefficient balistique au début de son exécution. De la même façon les variables

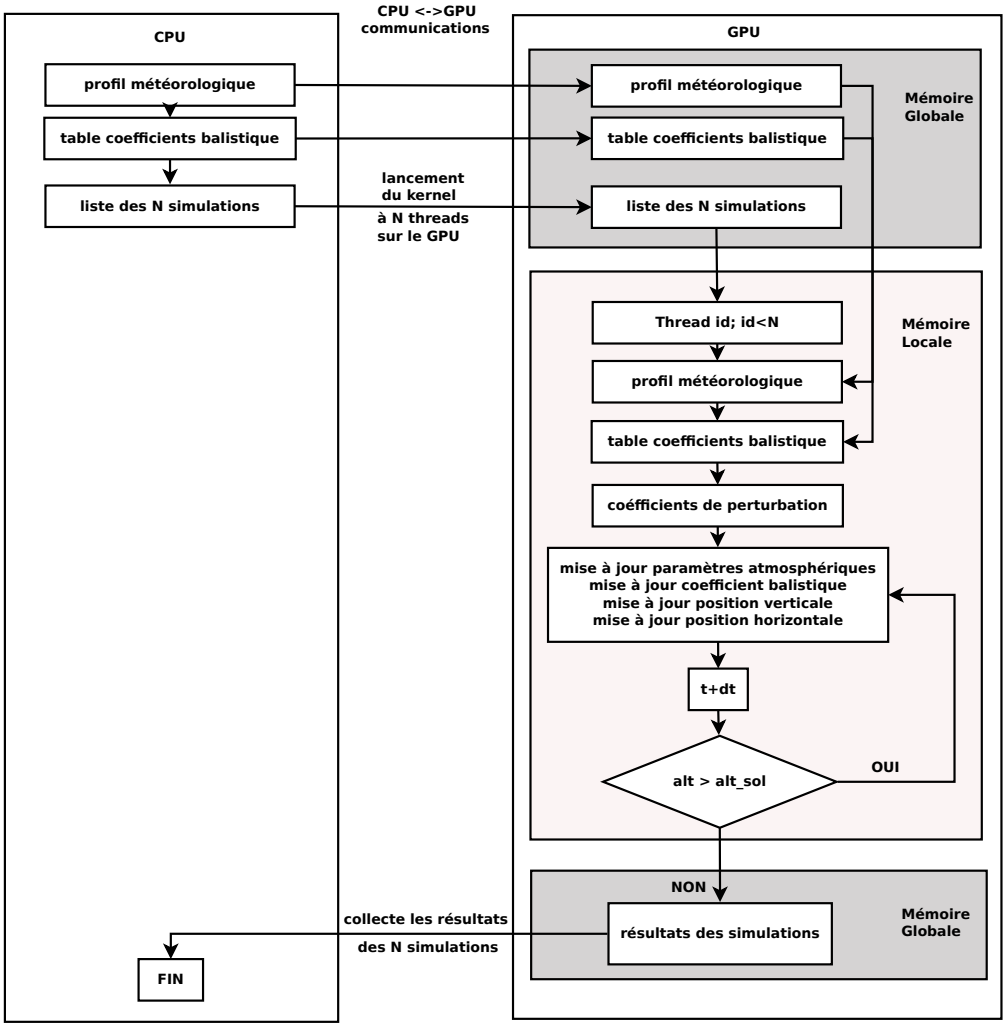

FIGURE 4.6: Algorithme GPU à parallélisme de tâche avec gestion de la mémoire

de la simulation (position, vitesse, coefficients de perturbation) sont stockées dans la mémoire locale de façon à ce que durant le calcul, un thread ne manipule aucune donnée dans la mémoire globale. Seul les résultats du dernier pas de temps de calcul de chaque thread sont stockés dans la mémoire globale pour pouvoir ensuite être copié vers le CPU. Chaque thread étant indépendant aucune variable n'est stockée dans la mémoire partagée des blocs. Afin de s'assurer que toutes ces données restent dans la mémoire locale de chaque thread nous avons utilisé la commande *cudaFuncCachePreferL1*, afin de faire passer la mémoire L1 cache de chaque processus à 48KB.

Ce nouvel algorithme présenté en détail Figure 4.6, et nommé Parallélisme de tâche et gestion mémoire dans le tableau 4.2, calcule les 100 800 simulations en 2,79 s, soit une réduction du temps de calcul de 8%.

Ce gain est assez modeste compte tenu des efforts de programmation mis en œuvre à ce stade, qui ont impliqué une réécriture de toutes les variables dans le code, la gestion de la copie des données de la mémoire globale vers la mémoire locale. Toutefois ces efforts

```
//Lancement du code sur le CPU
Initialisation des donnees sur le CPU;
Declaration : N; //Nombre de simulations total à faire

//Détection du nombre de GPUs
Declaration : nbrGPU; //Nombre de GPU sur le systeme hôte
cudaGetDeviceCount(&nbrGPU);

for chaque GPU : i <= nbrGPU do:
  //Sélection du GPU i
  cudaSetDevice(i);

  //Récuperation des caractéristiques du GPU i
  cudaDeviceProp deviceProp;
  cudaGetDeviceProperties(&deviceProp,i);

  Declaration : donnees meteo et coefficient de trainee pour le GPU i;
  Copie : donnees meteo et coefficient de trainee du CPU vers le GPU i;

  //Dimensions du kernel pour le GPU i
  Initialisation : threadsPerBlock; //On lance autant de threads par bloc que ce que la carte supporte
  Initialisation : blockPerGrid = (N/nbrGPU+threadsPerBlock-1)/threadsPerBlock;

  //Lancement du kernel sur le GPU i
  DDEnveloppeKernel<<<blockPerGrid,threadsPerBlock>>>(variables GPU i);
end for

//Synchronisation des GPU
cudaDeviceSynchronize();

//Copie des données sur le CPU et écriture des résultats
```

PSEUDO-CODE 4.1: Partie du code mis en œuvre sur le CPU dans le cas d'une mise en œuvre multi-GPU

sont à relativiser dans le contexte du développement d'un code ex-nihilo puisque alors la prise en compte de ce paramètre dès la conception du code dispense de l'étape de réécriture, seule la gestion de la copie des données entre la mémoire globale et locale sera à considérer.

Afin de nous assurer que les données qui sont utilisées pour le calcul figurent bien dans la mémoire locale de chaque thread nous considérons le "*local memory overhead*" qui est un ratio entre le trafic de données en mémoire locale et le trafic de donnée total sur la carte [53]. Nous avons alors constaté que ce ratio passait de 0,06% dans le cas l'algorithme à parallélisme de tâche à 92,79% dans le cas du nouvel algorithme à parallélisme de tâche avec gestion de la mémoire, confirmant bien que les données utilisées pour le calcul figurent dans la mémoire locale L1 de chaque threads.

4.3.2.4 Gestion Multi-GPU

Nous avons enfin considéré la problématique de la programmation sur plusieurs GPU grâce à l'utilisation de la carte NVIDIA K80. En effet cet accélérateur de calcul à 4 992 cœurs CUDA est en fait composé de deux cartes graphiques GK210 assemblées sur le même support. L'accélérateur est donc composé de deux cartes possédant chacune son propre système de mémoire devant être adressé séparément. Concrètement en terme de programmation tout se passe comme si il y avait deux cartes connectées à la plate-forme hôte.

Les deux cartes composant la K80 étant identiques, nous avons pris le parti d'allouer à

chaque carte la moitié du nombre total de calculs à effectuer. En terme de programmation du kernel, le passage à la programmation sur deux GPU n'a engendré aucune modification de code puisque du fait de la structure agréablement parallèle de l'application aucune communication de données n'est nécessaire entre les threads et donc entre les GPUs. En revanche au niveau du code CPU, il a fallu doubler tout ce qui était en lien avec le GPU. Le Pseudo code 4.1 présente une version simplifiée et généralisée à "$nbrGPU$" de l'algorithme employé sur le CPU pour gérer cette implémentation multi-GPU (dans le cas de l'utilisation de la K80: $nbrGPU = 2$). Comme on peut le voir la première étape consiste a identifier le nombre de GPUs CUDA compatibles connectés sur la plate-forme hôte "$nbrGPU$". Ensuite pour chaque GPU il faut allouer la mémoire pour les données météo et balistique, les copier et instancier le kernels et ainsi de suite. Ce processus se fait naturellement de façon asynchrone. En effet après le lancement d'un kernel, le code sur CPU passe aux instructions suivantes sans attendre que le kernel ait fini de traiter les tâches qui lui ont été confiées, ce qui permet de lancer plusieurs kernels en même temps sur un même GPU ou sur différents GPUs de façon asynchrone.
Il faut ensuite synchroniser tous les GPUs pour récupérer les résultats de tous les GPU. Cette étape consiste à bloquer l'exécution du code CPU jusqu'à ce que tous les kernels instanciés sur les différentes cartes aient terminé de s'exécuter. Une fois les données de chaque kernel copiées sur le CPU il faut ensuite les concaténer et les écrire dans un fichier de sortie. Cette tâche s'avère plus fastidieuse que compliquée puisqu'elle implique, dans le cadre de l'utilisation de la K80, de doubler toutes les variables, et de les nommer de façon à identifier sur quelle carte elles résident, puis de passer d'une carte à l'autre pour instancier les calculs et récupérer les résultats. En effet il n'est pas possible de réutiliser le même nom de variables pour tous les kernels, ceci étant considéré par le compilateur au niveau du CPU comme une redéfinition de variable.
Ainsi le code avec un algorithme à parallélisme de tâche avec gestion de la mémoire, déployé sur deux cartes calcule les 100 800 simulations en 1,4 s sur la K80.

4.3.2.5 Comportement en cas de divergence des threads

Dans le cas d'une utilisation plus générale des GPUs se pose la question des performances en cas de divergences des threads. En effet, en particulier pour la problématique de la rentrée atmosphérique des satellites on s'attend à avoir un certain degré de divergence selon le régime aérothermodynamique de chaque simulation. Afin d'éprouver la robustesse des performances à la divergence nous avons mené deux expériences, se basant sur un changement d'intégrateur numérique selon une condition donnée dans le code de dérive descente afin de créer artificiellement de la divergence. Nous avons implémenté dans la meilleure mise en œuvre de l'algorithme sur GPU en plus de l'intégrateur par la méthode d'Euler, un intégrateur de Runge-Kutta 4 (RK4).

TABLE 4.3: Temps de calcul sur la K40 en fonction de l'intégrateur numérique

Nombre de simulations	Euler	RK4	Expérience
100 800	2,79	2,91	2,96

Nous avons ensuite réalisé une expérience consistant à imposer à tous les threads pairs d'utiliser la méthode d'Euler et à tous les threads impair d'utiliser la méthode RK4, afin qu'au sein d'un même block deux threads consécutifs utilisent une méthode d'intégration différente. La Table 4.3, présente un récapitulatif de ces résultats obtenus sur la K40 pour 100 800 simulations calculées uniquement avec la méthode d'Euler, uniquement celle de RK4 et selon l'expérience décrite ci-dessus. On constate que le temps de calcul est légèrement supérieur à celui obtenu uniquement avec la méthode RK4. Mais la faible différence de temps de calcul indique que la divergence introduite dans le calcul n'a pas induit une séquentialisation des calculs comme on aurait put le craindre, indiquant une certaine résistance à la divergence des threads sur le GPU. Cette robustesse est certainement dûe à l'éfficacité des ordonanceurs.
Notons enfin que le temps de calcul avec la méthode de Runge-Kutta n'est pas quatre fois plus lente que la méthode d'Euler. Cela vient du fait que seul le temps passé à intégrer le système différentiel est multiplié par quatre tandis que le temps passé à lire les tables de données (météo et coefficient balistique) ainsi qu'à les interpoler est lui identique quel que soit l'intégrateur numérique.

4.3.2.6 Bilan GPU

Le premier constat que l'on peut dresser suite à ce travail sur le GPU, c'est que le modèle de programmation imposé par l'usage des bibliothèques, qui sont aujourd'hui développées pour le calcul parallèle sur GPU, ne permet pas une mise en œuvre optimale sur GPU des problèmes de type agréablement parallèles tel que celui que nous considérons. En effet, si cette implémentation permet d'obtenir une réduction du temps de calcul d'un facteur 50 par rapport à une version séquentielle non optimisée sur CPU de l'algorithme, cette première mise en œuvre sur GPU est deux fois plus lente qu'une implémentation parallèle et vectorisée sur CPU (cf. sous-section 4.3.4).
Ensuite grâce à une transition d'un modèle de parallélisme de boucle à un modèle de parallélisme de tâche, les performances du code ont été grandement améliorée (réduction du temps de calcul par un facteur 7). Le passage de l'un à l'autre n'a posé aucun véritable problème, simplifiant la gestion des données qui devaient être privées à chaque threads. Ensuite le passage de l'utilisation des données de la mémoire globale à la mémoire locale pour limiter les latences dans le calcul n'a pas offert un gain très conséquent, surtout si l'on considère le fait d'avoir dû rédéfinir pratiquement toutes les variables et gérer leurs copies.

TABLE 4.4: Temps de calcul pour 100 800 simulations sur Xeon Phi

Implémentation Parallèle	Temps (s)
Parallélisme de tâche	29,98
Parallélisme de tâche et vectorisation	23,06
Parallélisme de tâche avec vectorisation et gestion de la mémoire	8,02
Parallélisme de tâche avec gestion de la mémoire et vectorisation à taille fixe	6,86

Pour ce qui est de notre problème de Monte-Carlo de type agréablement parallèle, l'utilisation de plusieurs GPU ne représente aucune réelle difficulté, juste de multiplier la définition des variables et l'appel aux fonctions selon le nombre de GPU utilisés. Le gain obtenu est linéaire avec le nombre de GPU utilisé ce qui nous permet d'envisager l'utilisation de plusieurs GPU pour réduire d'avantages le temps calcul, ou d'augmenter le nombre de simulations calculées dans un temps donné, ce qui peut être une solution intéressante pour des analyses de Monte-Carlo demandant énormément de calculs pour converger.

Le véritable point problématique de la mise en oeuvre de code sur GPU, qui n'a pourtant pas été au cœur de cette étude, c'est l'absence de bibliothèque disponible pouvant être utilisée pour notre approche à parallélisme de tâche, qui comme nous venons de le montrer est la plus adaptée au traitement des problèmes de type agréablement parallèle. En effet la plupart des bibliothèques existantes sont conçues pour être appelées depuis des sections de codes sur le CPU, exporter et résoudre une partie des calculs sur le GPU, puis retourner les résultats au CPU, et ne sont pas prévues pour être appelées directement depuis un kernel, ce qui oblige le programmeur à développer sa propre solution, comme nous avons dû le faire pour l'intégrateur numérique. Ce point pose un certains nombre de questions en particulier pour ce qui est de la validation de la solution implémentée, sa maintenance, etc...

4.3.3 Évaluation des performances sur coprocesseur Xeon Phi

Dans cette sous-section nous étudions la mise en œuvre de l'algorithme de dérive descente enveloppe (Figure 4.1), sur coprocesseur Xeon Phi.

4.3.3.1 L'approche parallélisme de tâche

Au moment où nous avons entamé cette étude il n'existait pas de bibliothèques permettant l'intégration de système différentiel en parallèle sur Xeon Phi, et pour autant que nous le sachions ce n'est toujours pas le cas au moment de l'écriture de ces lignes. Nous avons donc comme pour le GPU pris le parti de mettre en œuvre notre propre solution d'intégrateur numérique de système différentiel sur Xeon Phi.

```
//Lancement du code sur un seul c\oe ur
Declaration : N; //Nombre de simulation à calculer

Entree : NPROCESS; //Nombre de processus OpenMP
Entree : nbrSimu = N/NPROCESS; //Nombre de simulations jouées par processus OpenMP

Entree : alt(m); pres(m); temp(m); ventU(m); ventV(m); //Vecteurs contenant les données du profil météo a 'm'
      lignes: altitude, pression, température, vents U et V
Entree : coef_bal(o); temps(o); //Vecteurs contenant  l'évolution temporelle du coefficient balistique en
     fonction du temps sur 'o' niveaux

Declaration : z(N); vit(N); lat(N); lon(N); //Vecteurs contenant l'altitude, la vitesse et la position en
     latitude longitude de chaque simulations à chaque itération

//Déploiement parallèle
#pragma omp shared(alt,pres,temp,ventU,ventV,coef_trainee,temps,z,vit,lat,lon)
#pragma omp for
//Boucle sur les simulations à jouer par processus OpenMP, une simulation à la fois
for chaque simulation do

  Generation : perturb(m,5); //Table contenant les coefficients de perturbation des donnees météo
  Declaration : alt_simu; pres_simu; temp_simu; ventU_simu; ventV_simu; //Variables contenant les donnees mét
     éo pour l'itération courante
  Declaration : coef_bal_simu; //Variable contenant la valeur de coefficient balistique pour l'itération
     courante

  while z[simulation]>alt_sol do
    Mise a jour donnees meteo : alt_simu, pres_simu, temp_simu, ventU_simu, ventV_simu
    Mise a jour coefficient balistique : coef_bal_simu
    Mise a jour position verticale : z[simulation],     vit[simulation]
    Mise a jour position horizontale : lat[simulation], lon[simulation]
    t+=dt
  end while
end for

Ecriture : z(N); vit(N); lat(N); lon(N); //Archivage des  résultats de simulations
```

PSEUDO-CODE 4.2: Mise en œuvre de l'algorithme à parallélisme de tâche en mode natif sur coprocesseur Xeon Phi

Comme dis à la sous-section 3.2.2.2, à l'inverse du GPU, le Xeon Phi permet à un utilisateur de se connecter directement sur la carte afin d'y stocker des données et de lancer directement un code de calcul parallèle sans l'intervention d'une plate-forme hôte, via le mode de programmation natif. Nous avons choisi de tirer avantage de ce mode de programmation car celui-ci permet de s'affranchir de toutes les problématiques de gestion de copies des données entre l'accélérateur et l'hôte. Ce postulat de départ étant adopté, il est alors apparu naturel d'adopter directement un algorithme à parallélisme de tâche. En effet le code se lançant sur le Phi, une fois déployé en parallèle sur les différents cœurs du Phi, la mise en œuvre la plus intuitive consiste à effectuer tous les calculs sur les cœurs du Phi.

Nous avons donc mis en œuvre l'algorithme présenté sur le Pseudo-code 4.2.

Comme pour le GPU, on commence par lancer un seul processus qui charge les données météorologiques et la table d'évolution du coefficient balistique. Ensuite on crée les vecteurs qui vont contenir l'ensemble des données de chaque simulation. Tous les vecteurs sont alloués en utilisant l'instruction posix_memalign(), et sont alignés sur 64 bits. On déploie ensuite l'application sur l'ensemble des cœurs du Xeon Phi via OpenMP, en lançant 244 processus parallèles. Chaque processus est chargé de calculer $nbrSimu$ simulations, soit le nombre total de simulations à effectuer divisé par le nombre de processus OpenMP. Un processus calcule ainsi ses $nbrSimu$ de façon séquentielle. Toutes les variables à l'exception des coefficients de perturbation et des valeurs intermédiaires des données météo sont stockées dans des vecteurs partagés par tous les processus OpenMP.

Cet algorithme, nommé Parallélisme de tâche dans la Table 4.4, permet de calculer 100 800 simulations en 29,98 s.

4.3.3.2 Gestion de la vectorisation sur le Phi

La mise en œuvre présentée à la sous-section précédente tire parti de tous les cœurs de calculs de la carte et du nombre maximal de threads que chaque cœur peut supporter. Néanmoins nous avons passé sous silence les capacités de calcul vectoriel du Xeon Phi. Nous détaillons ci-dessous le code qui tire profit des unités vectorielles de chaque cœur, présenté dans le Pseudo-code 4.3.

```
1  //Lancement du code sur un seul c\oe ur
   Declaration : N; //Nombre de simulation à calculer
3
   Entree : NPROCESS; //Nombre de processus OpenMP
5  Entree : nbrSimu = N/NPROCESS; //Nombre de simulations jouées par processus OpenMP

7  Entree : alt(m); pres(m); temp(m); ventU(m); ventV(m); //Vecteurs contenant les données du profil météo
       à 'm' lignes: altitude, pression, température, vents U et V
   Entree : coef_bal(o); temps(o); //Vecteurs contenant  l'évolution temporelle du coefficient balistique
       en fonction du temps sur 'o' niveaux
9
   Declaration : z(N); vit(N); lat(N); lon(N); //Vecteurs contenant l'altitude, la vitesse et la position
       en latitude, longitude de chaque simulations à chaque itération
11
   //Déploiement parallèle
13 #pragma omp shared(alt,pres,temp,ventU,ventV,coef_bal,temps,z,vit,lat,lon)
   #pragma omp for
15 //Boucle sur les simulations à jouer par processus OpenMP, une simulation à la fois
   for chaque simulation do
17   Generation : perturb(m,5); //Table contenant les coefficients de perturbation des données météo
     Declaration : vAlt_simu(v); vPres_simu(v); vTemp_simu(v); vVentU_simu(v); vVentV_simu(v); //Vecteurs
         contenant les données météo pour l'itération courante de dimension v fixe  à l'exécution
19   Declaration : vCoef_bal_simu(v); //Vecteurs contenant la valeur des coefficient balistique pour l'ité
         ration courante

21 #pragma vector aligned
   #pragma ivdep
23 #pragma simd
     for i parmi v do
25     while z[simulation[i]]>alt_sol do
         Mise a jour donnees meteo : vAlt_simu[i]; vPres_simu[i]; vTemp_simu[i]; vVentU_simu[i];
27       Mise a jour coefficient balistique : vCoef_bal_simu[i]
         Mise a jour position verticale : vZ[simulation[i]], vVit[simulation[i]]
29       Mise a jour position horizontale : vLat[simulation[i]], vLon[simulation[i]]
         t+=dt
31     end while
     end for
33 end for

35 Ecriture : z(N); vit(N); lat(N); lon(N); //Archivage des  résultats de simulations
```

PSEUDO-CODE 4.3: Mise en œuvre vectorielle de l'algorithme à parallélisme de tache en mode natif sur coprocesseur Xeon Phi

Pour réaliser cette mise en œuvre vectorielle de l'algorithme nous avons utilisé la norme SIMD d'OpenMP 4.0 [1], qui permet de vectoriser le calcul d'une boucle 'for' sur les éléments d'un ou plusieurs vecteurs.
Pour donner un exemple, considérons 3 vecteurs A, B et C de même dimension, tels que C = A+B. De façon séquentielle, on écrirait une boucle sur le nombre i d'éléments

de C, qui calculerait $C[i] = A[i] + B[i]$. Grâce aux directives SIMD d'OpenMP 4, on écrit la même boucle 'for' que pour le code séquentiel, mais grâce à l'ajout de #pragma simd, l'addition des éléments des deux vecteurs se fait dans l'unité vectorielle du cœur de calcul qui va calculer en parallèle cette opération pour tous les éléments du vecteur à condition que la dimension du vecteur soit inférieure ou égale à la dimension de l'unité vectorielle (512 bits dans le cas du Phi). Si la dimension est supérieure le calcul sera séquentialisé par éléments de 512 bits.
Nous avons donc réécrit le code dans ce sens : un processus OpenMP ne calcule plus $nbrSimu$ de façon séquentielle, mais il les calcule par vecteurs de dimension v fixé de façon à ce que $nbrSimu/v$ soit un entier et que la dimension du vecteur ainsi défini soit inférieure ou égale à 512 bits. Cette opération implique aussi de reprendre la définition de toutes variables intermédiaires de calcul (données météo et coefficient balistique de l'itération courante) qui étaient définies précédemment comme des scalaires afin qu'ils deviennent des vecteurs, pour permettre la résolution vectorielle du calcul. Tous les vecteurs sont alloués avec l'instruction *posix_memalign()* pour garantir l'alignement des données sur 64 bits.
La vectorisation est effectuée par le compilateur d'Intel à partir des directives OpenMP dans le code. Or il s'avère que le compilateur est extrêmement conservateur. Si dans le code le compilateur détecte la moindre ambiguïté concernant la potentielle dépendance d'une donnée entre deux itérations de la boucle que l'on cherche à vectoriser, ou un possible accès à un élément vectoriel non aligné, le compilateur bloquera la vectorisation. Pour éviter cet écueil nous avons rajouté les directives #pragma vector aligned et #pragma ivdep, pour garantir au compilateur que les vecteurs qui figurent dans la suite du code sont alignés, et que les calculs sont indépendants.
Cet algorithme, nommé Parallélisme de tâche et vectorisation dans la Table 4.4, permet de calculer 101 504 simulations sur 244 processus OpenMP, avec des vecteurs de 32 éléments (on vérifie bien que notre vecteur fait 512 bits) en 23,06 s.

4.3.3.3 Gestion de la mémoire sur le Phi

Malgré la mise en œuvre de la vectorisation dans l'algorithme, le gain de temps de calcul obtenu est minime, en particulier lorsqu'on considère le travail qu'a demandé la vectorisation du code. La vectorisation étant automatiquement effectuée par le compilateur nous avons voulu vérifier dans un premier temps que l'exécutable généré était bien vectorisé. Ainsi nous avons utilisé l'outil de profilage d'Intel, Vtune Amplifier, pour vérifier les compteurs des unités vectorielles. Nous avons constaté que le pourcentage d'opérations effectuées sur les unités vectorielles était maximal, validant en terme de vectorisation la mise en œuvre de la sous-section précédente.
Le faible gain obtenu avec la mise en œuvre précédente de l'algorithme provient d'un

```
//Lancement du code sur un seul c\oe ur
Declaration : N; //Nombre de simulation à calculer

Entree : NPROCESS; //Nombre de processus OpenMP
Entree : nbrSimu = N/NPROCESS; //Nombre de simulations jouées par processus OpenMP

Entree : alt(m); pres(m); temp(m); ventU(m); ventV(m); //Vecteurs contenant les données du profil météo à 'm'
      lignes: altitude, pression, température, vents U et V
Entree : coef_bal(o); temps(o); //Vecteurs contenant  l'évolution temporelle du coefficient balistique en
     fonction du temps sur 'o' niveaux

Declaration : z(N); vit(N); lat(N); lon(N); //Vecteurs contenant l'altitude, la vitesse et la position en
     latitude, longitude de chaque simulations à chaque itération

Entree : v; //Dimension des vecteurs SIMD

//Déploiement parallèle
#pragma omp shared(alt,pres,temp,ventU,ventV,coef_bal,temps,z,vit,lat,lon)
#pragma omp for
//Boucle sur les simulations à jouer par processus OpenMP, une simulation à la fois
for chaque ensemble de v simulations do

  Generation : perturb(m,5,v); //Table contenant les coefficients de perturbation des données météo

  Copie : alt,pres,temp,ventU,ventV,coef_trainee,temps; //Copie les données dans la mémoire locale à la
     thread

  Declaration : vAlt_simu(v); vPres_simu(v); vTemp_simu(v); vVentU_simu(v); vVentV_simu(v); //Vecteurs
     contenant les données météo pour l'itération courante
  Declaration : vCoef_bal_simu(v); //Vecteurs contenant la valeur des coefficients balistique pour      l'ité
     ration courante
  Declaration : vZ(v); vVit(v);vLat(v); vLon(v); //Vecteurs contenant l'altitude, la vitesse et la position
     en latitude longitude des v simulations calculées en parallèle de facon SIMD

#pragma vector aligned
#pragma ivdep
#pragma simd

  for i parmi v do

    while z[simulation[i]]>alt_sol do
      Mise a jour donnees meteo : vAlt_simu[i]; vPres_simu[i]; vTemp_simu[i]; vVentU_simu[i];
      Mise a jour coefficient balistique : vCoef_bal_simu[i]
      Mise a jour position verticale : vZ[simulation[i]], vVit[simulation[i]]
      Mise a jour position horizontale : vLat[simulation[i]], vLon[simulation[i]]
      t+=dt
    end while
  end for

  Copie : vZ, vVit, vLat, vLon dans z(N), vit(N), lat(N), lon(N)

end for

Ecriture : z(N); vit(N); lat(N); lon(N); //Archivage des  résultats de simulations
```

PSEUDO-CODE 4.4: Mise en œuvre vectorielle de l'algorithme à parallélisme de tâche en mode natif sur coprocesseur Xeon Phi prenant en considération la localité des données en mémoire

problème de latence dans l'accès aux données. En effet, on a d'un côté des unités vectorielles qui traitent beaucoup de calculs et de l'autre des données qui sont stockées dans la mémoire partagée de la carte. Ainsi avant qu'une donnée ne soit disponible sur un cœur pour être traitée par une unité vectorielle, le processus OpenMP sur le cœur doit effectuer une requête de la donnée dans un des contrôleurs mémoires et la faire transiter par l'anneau, ce qui est très lent par rapport à des accés en mémoire cache et freine la puissance de calcul des unités vectorielles.

Il apparait donc important de maîtriser la localité des données. C'est cette mise en œuvre qui est présenté dans le Pseudo-code 4.4. Ce que nous avons fait a donc été d'effectuer une copie dans la mémoire locale de chaque processus OpenMP des données météorologiques, balistiques et des variables d'états de l'enveloppe. Ces vecteurs ainsi créés sont privés au processus et sont alloués dans la mémoire L2 cache du cœur hébergeant le processus. De ce fait lors du calcul, la donnée est très rapidement

disponible dans la mémoire cache pour son traitement par l'unité vectorielle.
Cette nouvelle version de l'algorithme, nommé Parallélisme de tâche avec vectorisation et gestion de la mémoire dans la Table 4.4, permet de calculer 100 800 simulations (avec la même granularité de calcul que précédemment) en 8,02 s.
Durant le processus d'optimisation nous avons aussi constaté que si nous fixions la dimension des vecteurs utilisés à 8 (en double précision : 8 éléments * 64 bits = 512 bits) pour les calculs SIMD, au lieu qu'il s'agisse d'une variable d'entrée du code, le temps de calcul passait à 6,86 s. Ce gain provient du fait que si la taille des vecteurs n'est pas fixée au moment de la compilation, le compilateur Intel est contraint d'ajouter des fonctions de contrôle dans le code pour gérer les différentes tailles possibles de vecteur ce qui a pour effet de rendre le code compilé moins optimisé, et donc plus lent.

4.3.3.4 Bilan Xeon Phi

La première leçon que l'on peut tirer est la très forte dépendance qui existe pour le Xeon Phi entre vectorisation et maîtrise de la localité des données. En effet, comme nous l'avons montré, il est impossible d'obtenir de bonnes performances si l'on n'a pas ces deux conditions réunies. Si l'on n'implémente que la vectorisation des calculs, le temps de calcul est limité par les problèmes de latence d'accès aux données dans la mémoire globale du Phi. De la même façon si l'on ne gère que la localité des données sans utiliser la vectorisation sur le Phi, le calcul prend plus de 28 s pour 100 800 simulations, étant limité par le nombre d'opérations effectuées de façon parallèle. Il est donc capital de traiter ces deux points avec soin pour obtenir de bonnes performances et cela peut impliquer des efforts de programmation assez conséquents. En effet dans notre cas, l'algorithme de dérive descente enveloppe qui a servi de base pour ce travail, a été pensé de façon scalaire et séquentielle. Il a donc fallu repenser et réécrire toute la structure des données de façon vectorielle. À noter que certaines portions de code n'ont put être vectorisées à cause de l'utilisation de fonctions spécifiques telle que rand() par exemple.
Remarque : Un autre point qui n'apparaît pas directement dans l'étude menée ci-dessus, c'est la question de l'utilisation de bibliothèques de calcul. Il existe des bibliothèques de calcul qui sont utilisables en mode natif aussi bien qu'en mode offload pour le Phi, mais le problème ici vient du fait qu'il s'agit de bibliothèques développées par Intel et qui sont régies par des licences d'utilisation gratuites ou non, plus ou moins restrictives, ce qui est un frein à leur utilisation dans un contexte industriel.

4.3.4 Point de comparaison avec un nœud de cluster

Le nœud de cluster considéré est comme indiqué dans la Table 4.1, constitué d'un Intel Xeon E5-2680-v2, contenant 20 cœurs répartis sur deux sockets et chaque cœur dispose d'une unité vectorielle AVX-256. Comme indiqué au Chapitre 3, en terme d'architecture

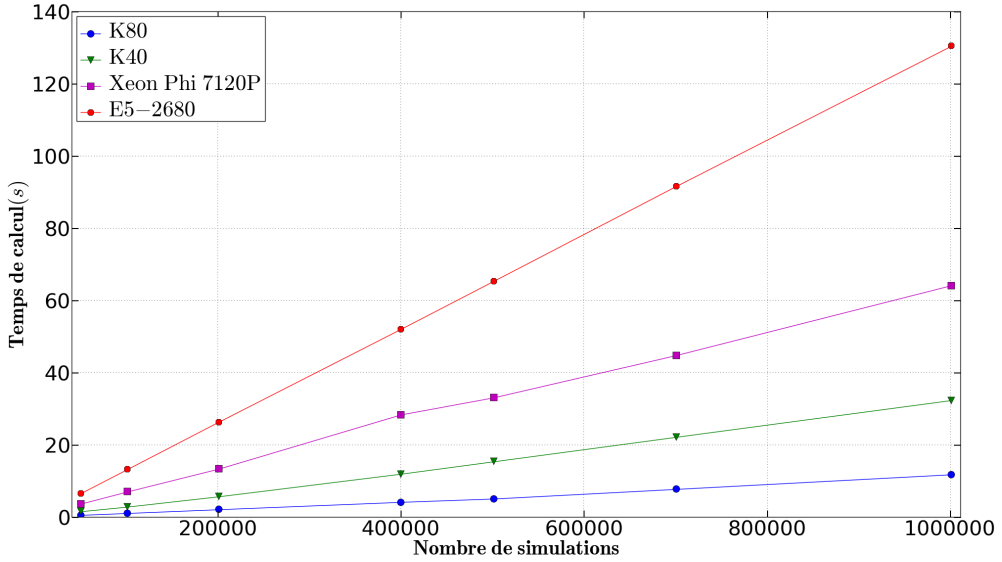

FIGURE 4.7: Temps de calculs en fonction du nombre de simulations des meilleures implémentations sur les 4 plate-forme de test

des cœurs les cartes Xeon Phi et les CPU de la gamme Xeon d'Intel sont très similaires (cf. Figure 3.1 et 3.5). La conséquence de ce point est qu'un code qui aura été optimisé pour tirer profit des capacités d'un accélérateur Xéon Phi est de fait optimisé pour être exécuté sur un processeur Intel. Nous avons donc réutilisé le code développé pour le Xeon Phi que nous avons compilé avec des options spécifiques pour le CPU (-mtune=core-avx-i -axCORE-AVX-I -mavx) à la place des instructions de compilations dédiées au Xeon Phi. L'exécutable engendré est ainsi vectorisé, parallèle et utilise la même gestion des données que la version optimale du Phi.

Le code ainsi généré permet de calculer 100 800 simulations en 13,31 s.

Si l'on compare à présent les performances des différentes mises en œuvre (GPU, Xeon Phi et Xeon) on constate que la meilleure mise en œuvre sur le Phi est deux fois plus rapide que la même mise en œuvre sur un nœud de cluster, et que la meilleure mise en œuvre sur la K40 est quatre fois plus rapide et celle sur la K80 est douze fois plus rapide que la mise en œuvre sur le nœud de cluster pour 100 800 simulations.

La Figure 4.7 présente les temps de calculs sur les différentes plate-forme de test. Comme on peut le voir, dans une perspective opérationnelle qui nécessite de réaliser un maximum de calcul en un minimum de temps avec des moyens de calcul offrant un encombrement et une consomation d'énergie limitée, les cartes graphiques (K40 et K80) ainsi que le Xeon Phi se révèlent très adapté permettant de calculer jusqu'à 1 000 000 simulations en une minute ou moins (12 s sur la K80).

E5-2680 v2	K40	K80	Xeon Phi 7120P
230	280	300	500

TABLE 4.5: Puissance électrique consommée (W)

4.3.5 Considérations pratiques

Outre les considérations de mise en œuvre de code parallèle que nous venons de traiter dans les sous-sections précédents, d'autres considérations d'ordre beaucoup plus pratique entrent en ligne de compte dans une optique industrielle. Typiquement il est crucial pour un industriel de pouvoir estimer *a priori* l'effort et le temps de travail qu'il va devoir mettre en œuvre pour obtenir un certain niveau de qualité de l'outil produit, mais pas seulement. En effet se pose aussi les questions des coûts d'achat, de maintenabilité du code et de disponibilité des solutions utilisées.

Pour ce qui est des coûts d'achat des cartes accélératrices lorsqu'on considère une carte graphique, telle que la K40, et le Xeon Phi 7120P, on est sensiblement sur la même base de prix d'achats au moment de la date de sortie aux alentours de 4 000 $, le prix baissant au fur et à mesure du temps, on les trouve aujourd'hui aux alentours de 2 000$. Si le prix d'achat de la carte est équivalent, une chose à laquelle il faut faire attention c'est la plate-forme hôte. Tandis qu'une carte NVIDIA K40 est compatible avec toute machine (tour, rack) disposant d'une alimentation suffisante, les Xeon Phi ne sont compatibles qu'avec certains modèles de plate-forme. Ainsi la carte 7120P n'est compatible qu'avec un nombre limité de racks de la gamme Intel ce qui la rend incompatible avec une utilisation telle que nous l'envisageons (moyen de calcul embarqué sur un site isolé). Seuls les coprocesseurs 5120 et 3120 (données avec des performances proches 1,2 TFLOPS pour 7120P et 1,0 TFLOPS pour 5120P) sont compatibles avec des stations de travail classique mais uniquement équipées d'un processeur Intel ce qui contraint grandement l'environnement de travail.

Ensuite en terme de consommation électrique, on note que les GPUs sont bien plus économes que le Xeon Phi, et même si ils consomment un peu plus qu'un nœud de cluster, la consommation indiquée pour les GPUs comprend aussi la consommation du système de refroidissement intégré à la carte, ce qui n'est pas le cas du nœud de cluster qui nécessite un système de refroidissement externe. En effet, en plus de la consommation du nœud, il faut considérer la consommation du système de régulation thermique du cluster qui selon la taille de ce dernier peut être très conséquente. Bien sûr à la consommation des GPUs doit aussi s'ajouter celle de la plate-forme sur la laquelle elles sont branchées, mais cette plate-forme pouvant être une unité centrale classique, cette consommation est très limitée vis a vis des solutions Phi et cluster.

Les deux accélérateurs de calculs permettent une programation orientée objet qui permet une bonne maintenabilité du code (cf. 5.3.2.3 pour la programmation orientée objet sur

GPU). Pour ce qui est de la pérennité des codes, on constate que depuis la version 2.0 de CUDA on a rétrocompatibilité entre les différentes versions de CUDA, mais rien ne garantit que ce sera toujours le cas, ce qui figerait alors les codes avec certaines versions de CUDA et certaines générations de GPU. Pour ce qui est des codes pour le Phi, ceux-ci sont basés sur OpenMP et MPI garantissant la pérennité des codes. En revanche on constate avec les nouvelles générations de coprocesseur Xeon Phi une importante modification de l'architecture des cartes pouvant altérer certaines optimisations.

4.3.6 Bilan des performances

Dans ce chapitre nous avons considéré la résolution d'un problème issu du monde réel, à savoir la détermination de zones probables de retombées d'enveloppes de ballons stratosphériques, au moyen de cartes accélératrices GPU et Xeon Phi. Nous avons mis en œuvre notre application sur les deux architectures en cherchant à identifier les points critiques à traiter pour obtenir une bonne performance dans une optique de développement industriel [63].

Pour ce qui est de la faisabilité de la mise en œuvre d'un outil d'analyse statistique sur accélérateur dans la perspective du développement du code parallèle de rentrée atmosphérique de satellites, l'étude ci-dessus démontre bien la faisabilité d'un tel projet, mettant en lumière que grâce aux spécificités des deux architectures un problème de type agréablement parallèle mis en œuvre avec un algorithme à parallélisme de tâche s'avère très performant en terme de temps de calcul. Avec des temps de calcul inférieurs à la minute pour plus de 1 000 000 de simulations, les architectures GPUs et Phi offrent une solution viable pour la réalisation d'analyses statistiques de type Monte-Carlo pour la dérive descente de ballons stratosphériques en conditions opérationnelles.

Pour ce qui est de la mise en œuvre, cette étude a mis en évidence la nécessité de repenser de façon vectorielle les algorithmes scalaires et séquentiels en plus de devoir gérer la localité des données sur Phi, à l'inverse de l'implémentation sur GPU qui requiert moins de re-conceptualisation de code et s'avère moins sensible (en terme de performance) aux divergences, puisque de toute façon dans le cas du Phi en cas de divergence la vectorisation est impossible. Typiquement il est impossible de vectoriser l'expérience menée à la sous-section 4.3.2.5.

Cette étude met aussi en lumière un point important, à savoir l'utilisation des bibliothèques de calcul existantes. En effet, comme nous l'avons mis en évidence avec Odeint qui sert à l'intégration numérique de système différentiel, la plupart des bibliothèques du marché pour le GPU ne sont pas compatibles avec une approche parallélisme de tâche car ne pouvant être appelées depuis un kernel. Pour le Xeon Phi le problème est un peu différent car la quasi-totalité des bibliothèques de calcul sont développées par Intel et leur utilisation est soumise à licence. Dans un cas comme dans l'autre, pour un

logiciel développé par un industriel pour commercialisation, le développeur est contraint de développer sa propre bibliothèque de calcul.
En terme d'investissement, nous avons aussi mis en évidence le fait que les cartes graphiques s'avéraient plus économiques, tandis qu'en terme de pérennité des codes la situation est un peu plus incertaine du côté des GPUs que du Xeon Phi.
Au vu de ces différents éléments nous avons décidé de ne conserver que la solution GPU pour la suite de ces travaux, qu'il s'agisse de l'exploitation du logiciel d'analyse de dérive descente ou bien le développement du logiciel de rentrée atmosphériques de satellite, la solution Xeon Phi se révélant trop dépendante de la capacité à vectoriser efficacement les calculs et à maîtriser la localité des données.

4.4 Résultats expérimentaux

Suite à l'étude précédente, nous nous concentrerons par la suite sur les accélérateurs de type GPU. Nous avons alors repris la mise en œuvre du code de dérive descente enveloppe sur GPU et avons ajouté le modèle de dérive descente chaîne de vol sous parachute présenté dans la sous-section 4.2.4 pour obtenir un logiciel capable d'effectuer une analyse statistique de dérive descente enveloppe ou chaîne de vol selon le modèle sélectionné. L'outil ainsi obtenu a été nommé Aquilon, en référence au dieu de la mythologie grecque des vents froids et violents du Nord.
Dans la section précédente nous nous sommes concentrés uniquement sur les performances en terme de temps de calcul de l'outil, dans ce chapitre à l'inverse nous allons nous concentrer sur les résultats expérimentaux que l'on peut obtenir avec Aquilon en réanalysant les données de la campagne ballon de Timmins de 2014, et en comparant les résultats produits par Aquilon à ceux qui ont été engendrés durant la campagne par l'outil de référence du CNES : NOSYCA.
Nous rappellons que NOSYCA est le logiciel de suivi de vol de ballon stratosphérique ouvert (BSO) utilisé par le CNES. Il permet de suivre en temps réel le vol d'un BSO, mais aussi de modéliser la trajectoire du ballon du moment du lâché jusqu'à l'arrivée au sol de l'enveloppe et de la chaîne de vol à partir de prévisions météorologiques. NOSYCA est ainsi utilisé pour simuler la dérive descente des composants du ballon afin de déterminer le polygone météo.

4.4.1 Validation d'Aquilon

Avant de considérer l'utilisation d'Aquilon pour réanalyser les différents vols de la campagne de vols ballons de Timmins et de comparer les zones de retombées ainsi déterminées par méthode de Monte-Carlo avec les polygones météo déterminés durant la campagne, il est important de vérifier la validité des résultats produits par Aquilon. Pour cette étude, la base de validation est constituée par les résultats produits par le

logiciel opérationnel du CNES, NOSYCA, durant la campagne.
Pour ce faire nous avons configuré le logiciel de dérive descente afin qu'il n'engendre qu'une simulation nominale, c'est à dire sans perturber le profil météorologique. Nous avons ensuite simulé la dérive descente sous parachute de chaque vol de la campagne en nous basant sur les conditions initiales et les données météo qui ont été utilisées pour effectuer les simulations servant à la détermination du polygone météo avec NOSYCA, lors de la campagne de Timmins 2014.
En comparant les résultats obtenus avec Aquilon et ceux de NOSYCA nous avons constaté un écart moyen de $10^{-6\circ}$ en latitude et $10^{-5\circ}$ en longitude ce qui correspond à la précision des résultats du logiciel du CNES que j'avais à ma disposition, validant ainsi les résultats d'Aquilon.

4.4.2 Amélioration de la méthode de perturbation du profil météorologique

Un des points critiques pour cette étude est la détermination des cœfficients d'incertitude pour les différents paramètres météorologiques à utiliser pour chaque vol.
Afin de générer ces coefficients on procède comme suit : tout d'abord on compare le profil de prévision pour la phase de descente avec celui qui a été mesuré par le ballon durant la phase de montée. Concrètement on parcourt le profil météo de montée, et à chaque niveau en altitude on cherche dans le profil de prévision si l'on dispose du même niveau. Si c'est le cas, on calcule alors directement la différence entre les données (pour ce niveau) du profil de montée et celles du profil de prévision. Dans le cas contraire on procède alors à une interpolation des données du profil de la prévision en utilisant les équations présentées à la sous-section 4.2.2. puis on calcule la différence entre les données interpolées et celles du profil de montée. De cette façon on construit une table d'écart entre les données réelles (qui sont mesurées durant la phase de montée) et les données issues de la prévision pour cette phase de montée (dans le cas de cette étude ces données sont engendrées par le logiciel ARPEGE de Météo France). Il est alors possible à partir de cette table de calculer pour chaque paramètre l'écart type et le biais entre les données de la prévision et celles mesurées par le ballon.
Puis chaque simulation génère son propre profil météo unique perturbé, en appliquant la formule de perturbation suivante à chaque niveau du profil :

$$Paramètre_{Perturbé} = Paramètre_{Initial} + Biais_{Paramètre} + \alpha_{Paramètre}\sigma_{Paramètre}, \quad (4.20)$$

avec :
$Biais_{Paramètre}$, la valeur du biais calculé pour le paramètre considéré
$\alpha_{Paramètre}$, un nombre aléatoire de distribution gaussienne centré en 0 et d'écart type 1, déterminé de façon aléatoire pour chaque paramètre d'une simulation donnée.

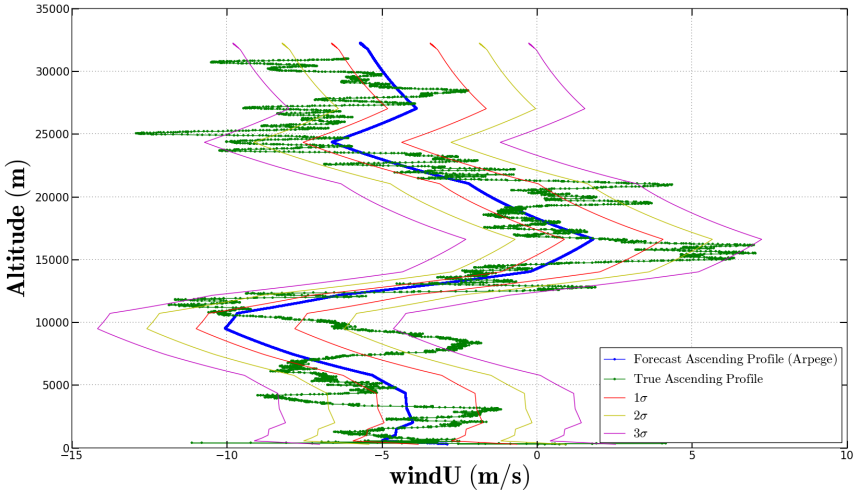

FIGURE 4.8: Profils de vents prévus perturbés à 1, 2 et 3 sigmas

$\sigma_{Paramètre}$, l'écart type calculé pour le paramètre considéré.
De cette façon on s'assure d'engendrer un ensemble de profils météo nous permettant de maximiser les incertitudes que l'on a sur le profil de prévision. Sur la Figure 4.8 on peut ainsi observer, pour le premier vol de la campagne Timmins 2014, le profil de vent prévu (courbe bleu) par rapport au profil de vent effectivement rencontré par le ballon durant sa phase de montée (courbe verte). Sur cette figure sont aussi tracés les profils à 1, 2 et 3 σ calculés par la méthode présentée ci-dessus. Le grand nombre N de simulations calculées nous assure une distribution Gaussienne représentative des profils générés.

4.4.3 Convergence des résultats

L'objectif de cette étude à présent est de pouvoir déterminer les zones de retombée probables à 1, 2 et 3 σ pour les confronter aux dimensions et aux positions des polygones météo déterminées par les opérateurs du CNES ainsi qu'à la position du point de retombée effectif. Afin de déterminer ces zones de retombées, nous utilisons la méthode de détermination des zones de confiances présentée à la sous-section 2.1.3.2. Or comme expliqué dans le Chapitre 2 sur la méthode de Monte-Carlo, pour que ce résultat soit véritablement exploitable, il faut au préalable s'assurer que notre analyse de Monte-Carlo a convergé.
Pour ce faire nous avons donc réalisé une étude de convergence sur les dimensions des petits et grands axes de la zone 1 σ du vol 1 de la campagne en augmentant progressivement le nombre de simulations. On constate sur les Figures 4.9 et 4.10, qu'à partir de 10 000 simulations on obtient la convergence des dimensions de l'ellipse de confiance, avec une variation résiduelle de l'ordre de 10^{-4} degrés lorsqu'on utilise plus de simulations.

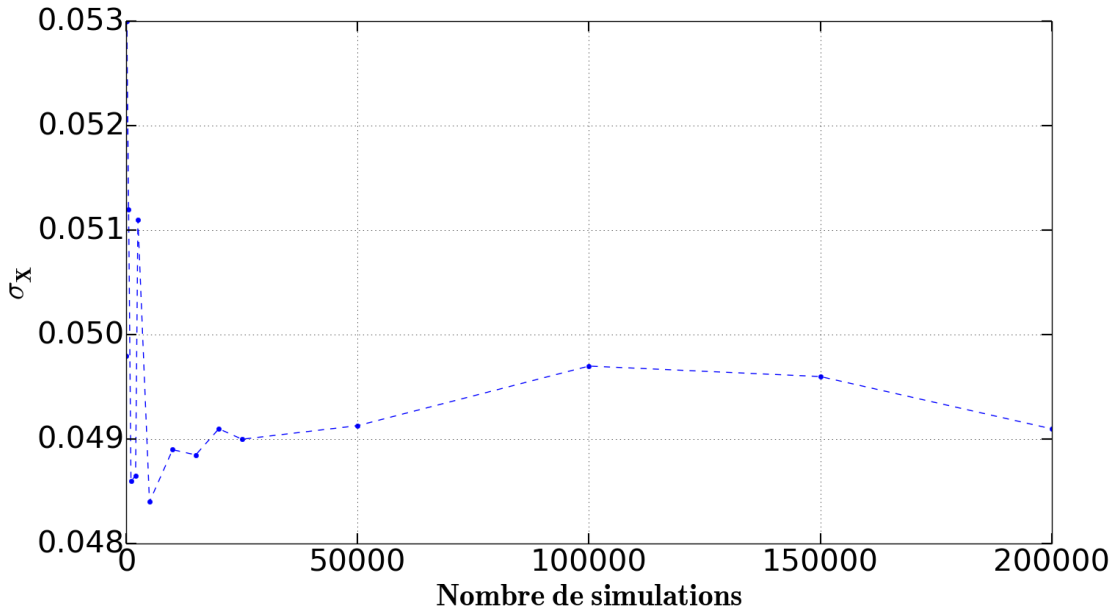

FIGURE 4.9: Evolution du demi-grand axe de la zone à 1 σ, en fonction du nombre de simulations

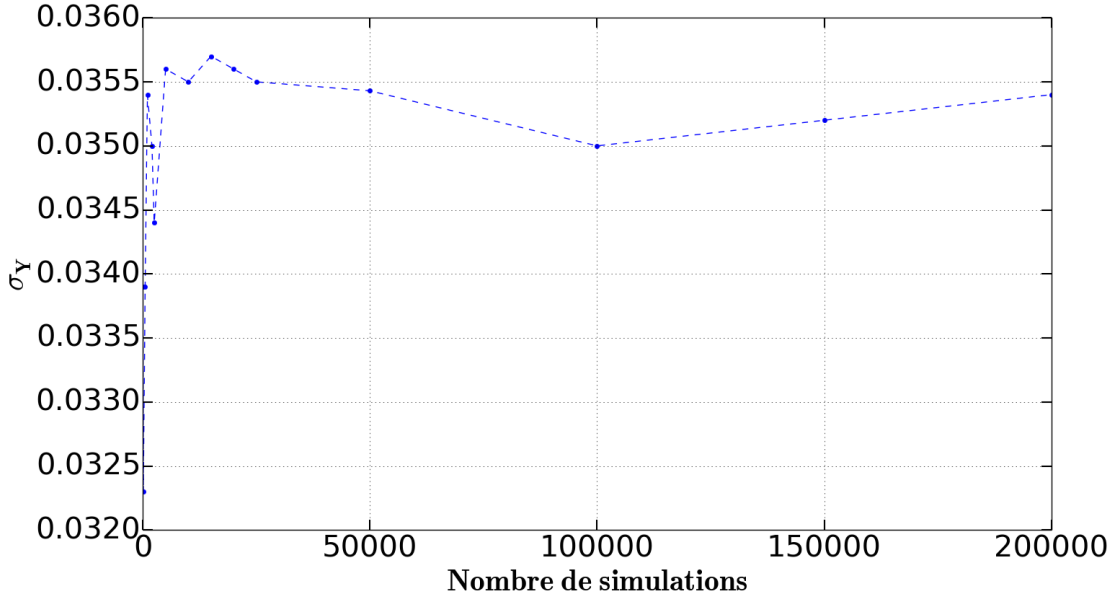

FIGURE 4.10: Evolution du demi-petit axe de la zone à 1 σ, en fonction du nombre de simulations

TABLE 4.6: Conditions initiales au moment de la séparation du Vol 1

Altitude de séparation	31 487,0m
Longitude de séparation	$-83,48805°$
Latitude de séparation	$48,0806333°$
Vitesse verticale de séparation	$1,10\ m.s^{-1}$
Masse à la séparation	614,5 kg
Masse de lest restante	146,6 kg
Masse sans lest	467,9 kg
Début vidange du bac à lest	58,0 s aprés la séparation
Surface des parachutes	$315\ m^2$

4.4.4 Réanalyse des données de la campagne ballon de Timmins en 2014

4.4.4.1 Protocole

Pour cette étude nous nous sommes concentrés sur l'analyse de la dérive descente chaîne de vol sous parachute puisqu'il s'agit de la partie la plus critique de la dérive descente. De plus dans l'outil CNES NOSYCA, la zone de retombée de l'enveloppe est déduite de la zone de retombée de la chaîne de vol.

Concernant le protocole de détermination des dimensions des zones probables de retombées, nous initialisons les analyses pour chacun des vols de la campagne à partir des paramètres des points de séparation réels, et nous employons les données du profil météo prévu utilisé lors de la campagne et qui sont perturbées à partir de la méthode détaillée à la sous-section 4.4.2. Pour chaque analyse nous réalisons ensuite 20 000 simulations afin de garantir la convergence des différents résultats (cf. 4.4.3).

4.4.4.2 Analyse des résultats

Vol 1

Contexte Le premier vol de la campagne ballon du CNES à Timmins en 2014 a eu lieu le 21 Aout 2014. Il s'agissait d'un ballon de type 100Z emportant une charge de 687 kg au décollage.

Aprés 7h06 de vol, dont 1h30 de phase de montée, le ballon a rempli sa mission scientifique et vient le moment de procéder à la séparation de la chaîne de vol et de l'enveloppe. Les paramètres initiaux de la simulation ayant menés à cette décision sont présentés Table 4.6.

Un des points intéressants du vol 1 c'est que comme on peut le voir sur la Figure 4.11, la chaîne de vol n'est pas retombée dans le premier polygone de dérive descente météo Z1 (en rouge sur la figure), qui est centré sur le point de retombée théorique issue de la simulation NOSYCA, mais à côté dans la zone Z2 (en vert, le Z3 est représenté en mauve).

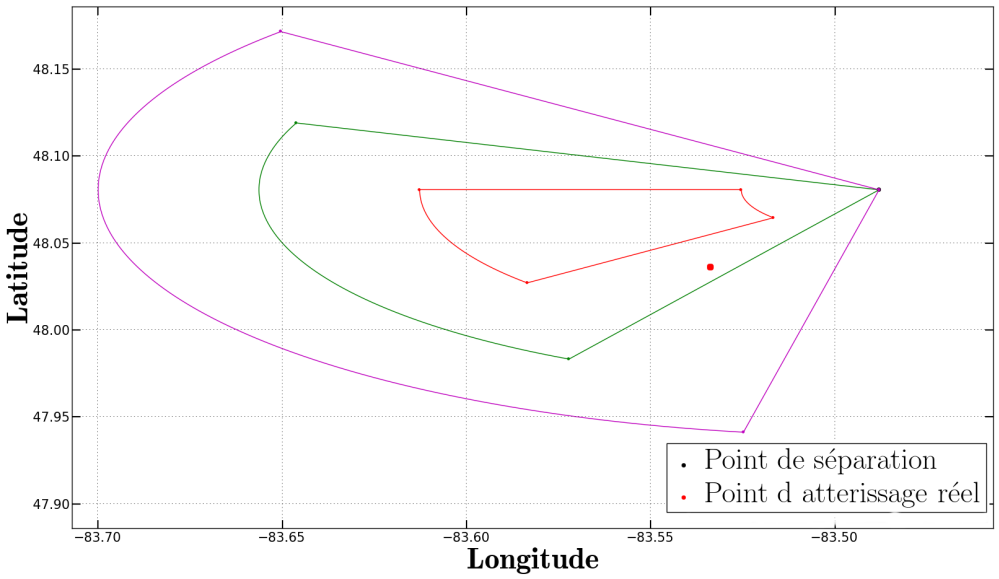

FIGURE 4.11: Point d'atterissage réel de la chaîne de vol et zone Z1 (rouge), Z2 (vert) et Z3 (mauve) définis à partir du résultat de simulation de NOSYCA

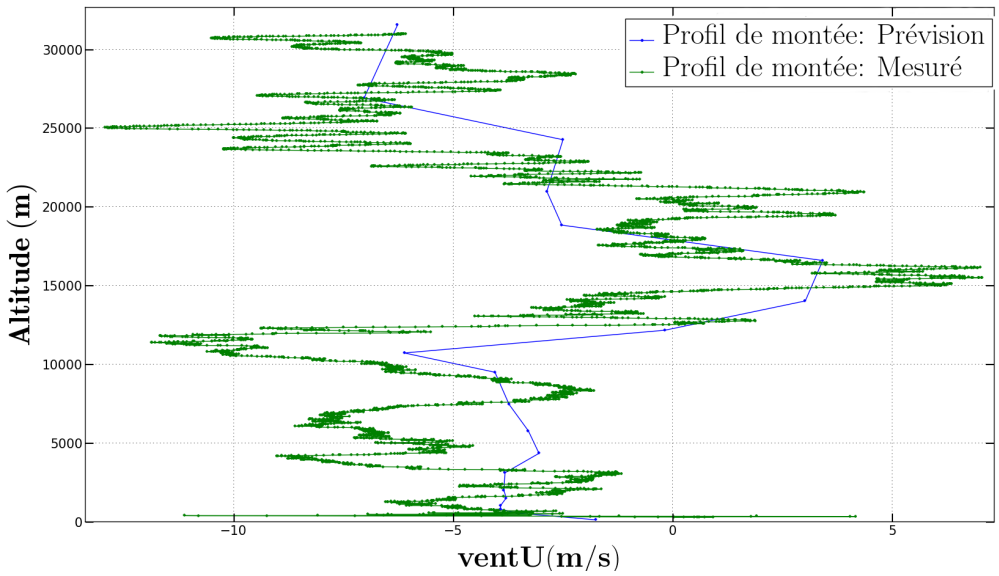

FIGURE 4.12: Profils verticaux du vent U pour la phase de montée : prévu (bleu) et effectivement observé (vert)

Perturbation du profil météorologique Afin d'estimer les paramètres d'incertitude à appliquer sur le profil météorologique prévu pour la dérive descente, nous comparons le profil météo issu de la prévision ARPEGE (généré 4-5h avant le départ du vol), avec le profil météo engendré à partir de l'analyse des données de la phase de montée du ballon selon la méthode présentée à la sous-section 4.4.2. Les Figures 4.12 et 4.13 présentent les deux profils de vents (prévus et mesurés en vol) pour les composantes U et V.

À partir de la comparaison des profils verticaux, on trouve ainsi les paramètres pour les composantes du vent présentés Table 4.7.

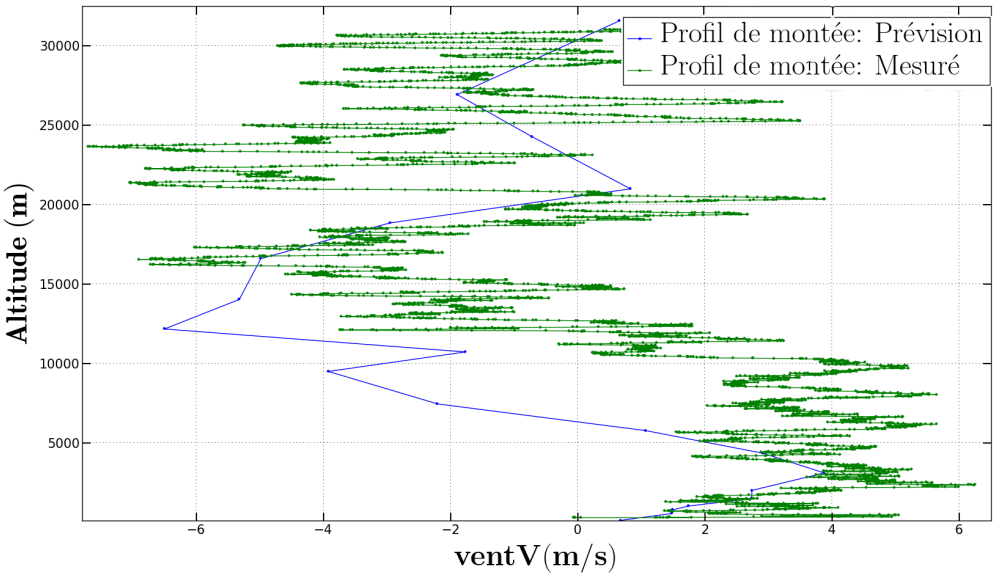

FIGURE 4.13: Profils verticaux du vent V pour la phase de montée : prévu (bleu) et effectivement observé (vert)

TABLE 4.7: Mesure de l'écart type et du biais pour les composantes du vent U et V

	Ecart type	Biais
Vent U	1,59	0,65
Vent V	1,7	-1,75

TABLE 4.8: Surfaces des zones probables de retombées calculées par NOSYCA et Aquilon

		1σ	2σ	3σ
Surface (km^2)	Aquilon	42,61	170,46	383,54
	Zone météo	27,24	109,10	259,51

Détermination de la zone de retombée A partir des conditions initiales et des perturbations présentées à la sous-section 4.4.4.2, Aquilon génère ainsi 20 000 résultats de simulations. On utilise alors la formule présentée à la sous-section 2.1.3.2 pour déterminer les paramètres des ellipses de confiance à 1, 2 et 3 σ.

Le Tableau 4.8 compare les surfaces des zones à 1, 2 et 3 σ calculées à partir des résultats produits par Aquilon, et les surfaces des polygones Z1, Z2 et Z3 déterminées pour ce vol. Le premier constat que l'on peut faire à partir de ces résultats c'est que, bien que les surfaces déduites des résultats d'Aquilon sont 1.5 fois plus grandes que celles des polygones météo, les résultats produits avec notre méthode de perturbation assez grossière restent du même ordre de grandeur que les polygones météo. Si outre la surface totale calculée, on considère à présent les dimensions caractéristiques des ellipses et du polygone on constate que la différence entre les deux résultats est encore moins tranchée. En effet le grand axe de l'ellipse est de 0,1° en longitude et le petit axe de 0,7° en latitude, tandis que la longueur du polygone est de 0,09° en longitude et son amplitude de 0,6° en

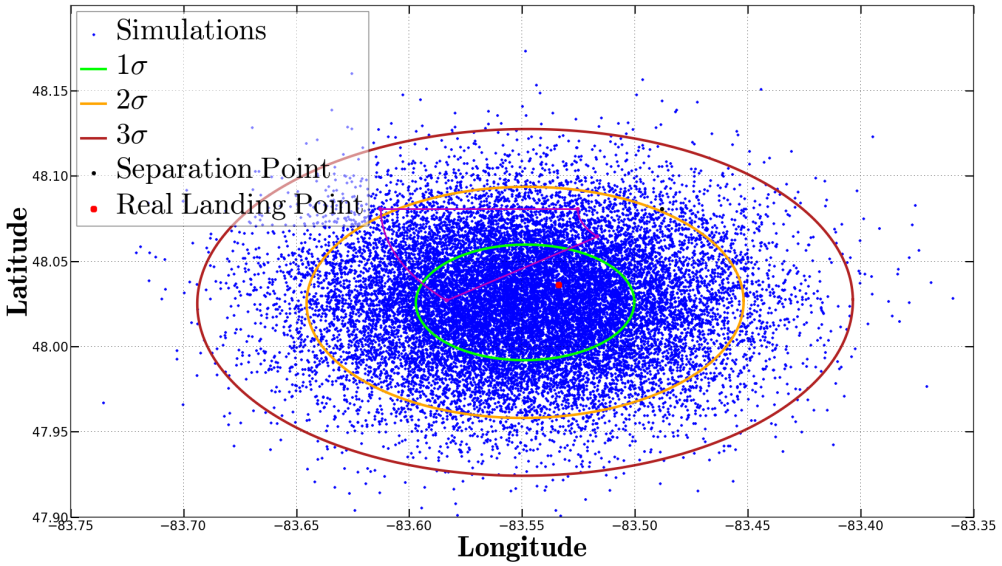

FIGURE 4.14: Résultats des simulations d'Aquilon et zones à 1, 2 et 3 σ

TABLE 4.9: Récapitulatif des surfaces des zones de retombées définis par Calima et Aquilon

—	Vol 2		Vol 3		Vol 5		Vol 6		Vol 7	
	Aquilon	NOSYCA	Aquilon	NOSYCA	Aquilon	NOSYCA	Aquilon	NOSYCA	Aquilon	NOSYCA
1σ	140,63	63,75	88,00	66,45	246,58	72,73	124,6	99,61	116,28	95,54
2σ	562,52	255,01	352,02	265,79	986,33	290,93	498,41	398,45	465,11	382,17
3σ	1265,67	573,77	792,04	598,03	2219,23	654,6	1121,43	896,51	1046,5	859,87

latitude. On vérifie ainsi que notre ellipse à 1 σ a les mêmes dimensions caractéristiques que le polygone météo et que la différence de surface constatée provient d'une différence entre les formes géométriques utilisées, l'ellipse se justifiant par la dispersion des résultats tandis que la forme du polygone est basée sur les observations des vols précédents.
De plus lorsqu'on considère la Figure 4.14 qui présente la dispersion des résultats d'Aquilon, ainsi que les zones à 1, 2 et 3 σ qui en sont déduits, on constate que le point réel de retombée se situe dans la zone à 1 σ (alors qu'il était hors du polygone Z1).

Résumé des autres vols Nous avons ensuite procédé selon la même méthode pour 5 des 6 autres vols de la campagne Timmins 2014. En effet faute de données suffisantes nous n'avons pas pu réanalyser le 4ème vol de la campagne. Les résultats de ces différentes études sont récapitulés dans le Tableau 4.9.
En vert sur ce tableau sont indiqués les résultats de simulation qui ont définis une zone à 1 σ contenant le point réel, et en orange ceux pour lesquels le point de retombée réel se trouve dans la zone à 2 σ.
En rajoutant les résultats du premier vol on constate que pour 4 des 6 vols analysés, Aquilon a défini une zone à 1 σ contenant le point de retombée réel. Les figures présentant la dispersion des résultats d'Aquilon ainsi que les zones à 1, 2 et 3 σ figurent en Annexe D.

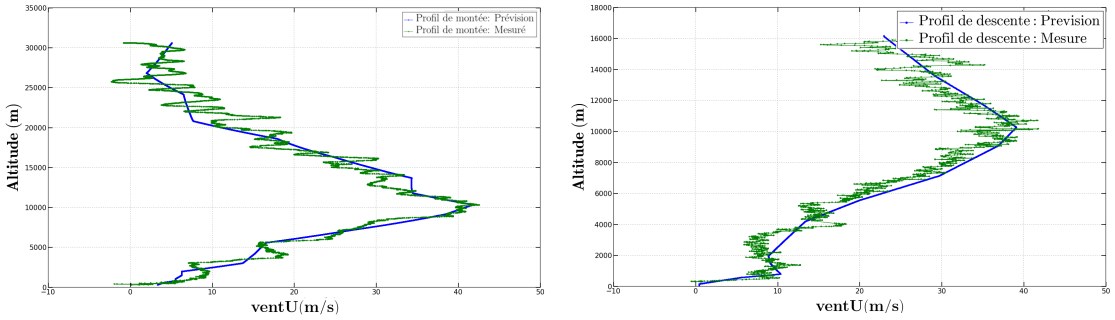

FIGURE 4.15: Comparaison du profil vertical du vent U pour le vol 6 prévu (bleu) et observé par le ballon (vert) durant la phase de montée (à gauche) et durant la phase de descente (à droite)

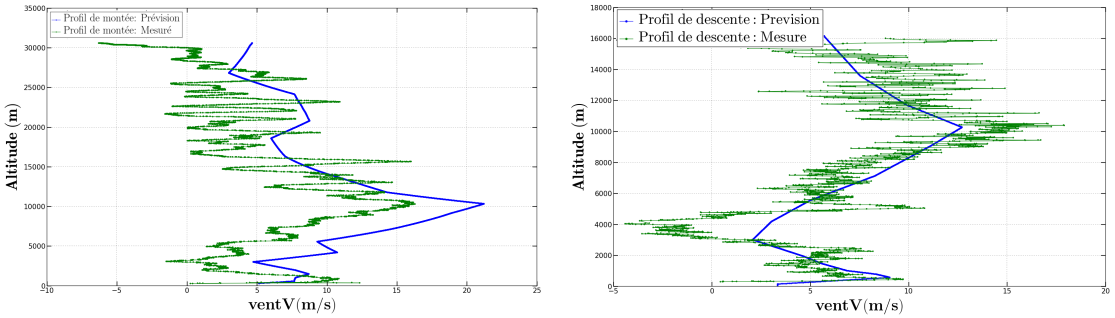

FIGURE 4.16: Comparaison du profil vertical du vent V pour le vol 6 prévu (bleu) et observé par le ballon (vert) durant la phase de montée (à gauche) et durant la phase de descente (à droite)

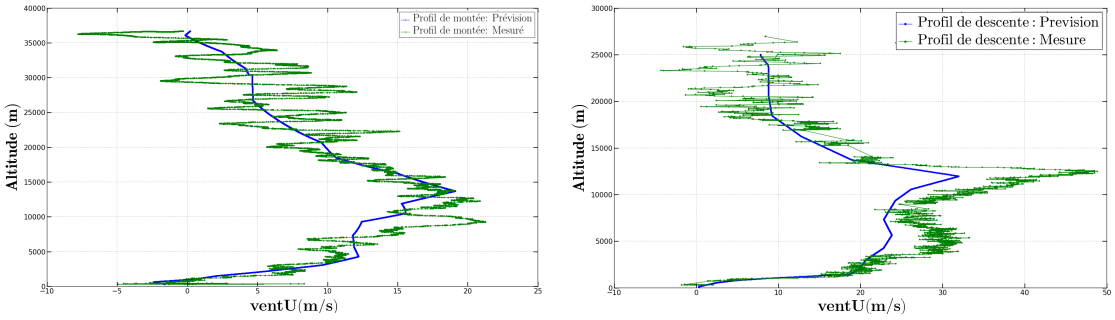

FIGURE 4.17: Comparaison du profil vertical du vent U pour le vol 7 prévu (bleu) et observé par le ballon (vert) durant la phase de montée (à gauche) et durant la phase de descente (à droite)

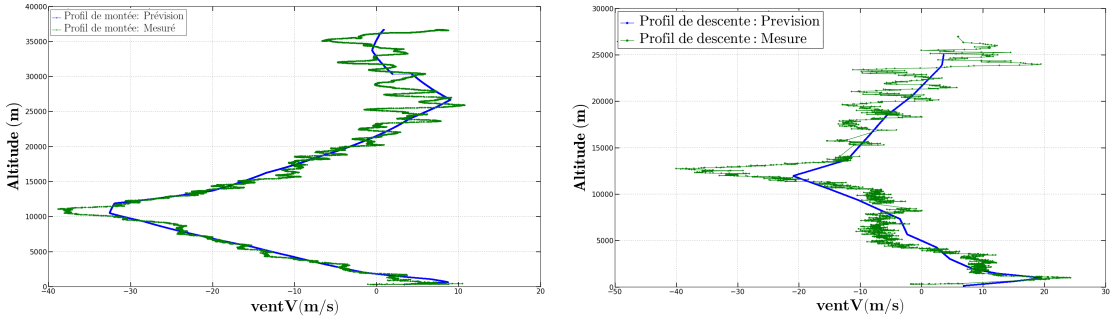

FIGURE 4.18: Comparaison du profil vertical du vent V pour le vol 7 prévu (bleu) et observé par le ballon (vert) durant la phase de montée (à gauche) et durant la phase de descente (à droite)

TABLE 4.10: Différences de biais calculées entre le profil de montée et celui de descente pour les composantes U et V du vent pour les vols de la campagne Timmins 2014

Vols	Biais U montée	Biais U descente	ΔBiais U	Biais V montée	Biais V descente	ΔBiais V
1	0,65	1,26	-0,61	-1,75	-1,17	-0,61
2	-1,9	-0,14	-1,76	-0,42	-0,78	-1,76
3	-0,87	-0,04	-0,83	0,4	-0,15	-0,83
5	-0,89	-0,31	-0,58	1,1	0,7	-0,58
6	-0,06	-1,38	1,32	-4,03	-0,56	-3,47
7	0,33	3,3	-2,97	0,39	-1,39	1,78

Nous avons ensuite analysé plus en détail les résultats des vols 6 et 7 afin d'étudier pourquoi pour ces deux vols le point de retombée se situait dans la zone à 2 σ. Nous avons ainsi comparé les profils de vents prévus et ceux effectivement observés durant la phase de montée et durant la phase de descente pour ces deux vols et nous avons constaté que les écarts entre les deux profils à la montée et à la descente étaient très importants.

Le Tableau 4.10 récapitule les biais mesurés entre les profils de prévision et réels pour les phases de montée et descente pour les composantes U et V des 6 vols de la campagne Timmins que nous avons analysées. Comme on peut le constater l'écart entre les biais calculés pour les deux phases pour les vols 6 et 7 est environ 3 m.s-1 supérieur à l'écart entre les biais des autres vols, pointant ici la limitation de la méthode naïve de génération des données de perturbations que nous avons implémenté pour cette étude. En effet avec cette méthode nous supposons que les écarts calculés entre la prévision et la réalité sont les mêmes au moment de la phase de montée et durant la phase de descente qui a lieu plusieurs heures après alors que rien ne garantit cela a priori. Et c'est ce qui s'est passé avec les vols 6 et 7. Pour le vol 6, il y a eut un écart important de vent entre la prévision et la réalité sur la phase de descente qui n'a pas été observé durant de la phase de montée (cf. Figure 4.15 et 4.16). C'est la même chose qui s'est produite pour le vol 7, sauf que là l'évènement a été observé durant la phase de montée et pas durant la phase de descente (cf. Figure 4.17 et 4.18).

Ce résultat ne remet donc nullement en question les performances d'Aquilon, mais seulement la méthode de génération des perturbations que l'on savait déjà être naïve.

4.4.5 Bilan de la première utilisation en condition quasi réelle d'Aquilon

Grâce au groupe de travail DEDALE, qui rassemble des experts et des opérationnels du CNES, ainsi que différentes personnes issues des milieux industriels et académiques, dédié à la réanalyse des vols des campagnes passées et à l'amélioration des procédures et des outils des opérationnels, nous avons pu avoir accès aux données de vols de la campagne ballon du CNES à Timmins en 2014.

Grâce à ces données nous avons pu dans un premier temps valider l'algorithme de dérive descente chaîne de vol et enveloppe implémenté dans Aquilon en reproduisant avec un

très bon accord les résultats des simulations de l'outil opérationnel CNES NOSYCA.
Nous avons ensuite appliqué une méthode de génération et de perturbation des données météo qui a été implémentée dans Aquilon afin de déterminer les zones de retombées probables à 1, 2 et 3 σ pour six des sept vols de cette campagne.
La méthode naïve de perturbation mise en œuvre dans Aquilon, s'est révélée plus précise que la méthode des polygones météo employée actuellement en réussissant à définir des zones à 1 σ contenant le point de retombé réel pour 4 des 6 vols étudiés, tandis que la méthode des polygones n'a obtenu ce résultat que pour 2 de ces 6 vols.
Néanmoins cette étude a aussi pointé deux limitations de cette nouvelle méthode. La première liée aux surfaces des zones déterminées qui sont plus grandes que celles des polygones météo. Cette limitation était attendue puisque la méthode de perturbation mise en œuvre a justement été pensée pour maximiser ces zones de retombées. Un raffinement de la méthode (par exemple en définissant des paramètres d'incertitudes par tranche d'altitude) entraînera donc automatiquement une réduction de cette surface. L'autre limitation pointée vient de l'utilisation des données de la phase de montée pour déterminer les perturbations de la phase de descente. Cette solution a été proposée car il s'agit de la seule méthode qui peut être utilisée actuellement en opération avec les données dont disposent les opérateurs au moment de la séparation. La seule façon de résoudre ce problème serait de trouver un moyen pour sonder le profil de vent réel peu de temps avant la séparation du ballon, mais cela demanderait de changer les procédures opérationnelles du CNES.

4.5 Conclusion

Dans ce chapitre nous avons présenté les résultats de l'étude portant sur la faisabilité de la réalisation d'un logiciel d'analyse de Monte-Carlo sur accélérateur de calcul, en considérant la problématique de la détermination de zone probable d'atterrissage d'enveloppes et de chaînes de vol de ballons stratosphériques.
Dans un premier temps nous nous sommes concentrés sur les problématiques liées à la programmation sur les accélérateurs de type GPU ou Intel Xeon Phi en ne traitant que le problème de la dérive descente d'enveloppe de ballon stratosphérique et en mesurant les performances en terme de temps de calcul à la lumière de considérations industrielles telles que le rapport entre investissement de programmation sur gain de temps, coûts matériel par rapport aux performances, etc... Cette étude a mis en évidence la faisabilité d'un tel projet et a mis en avant le fait que les GPUs s'avéraient être une meilleure solution pour cette mise en œuvre : travail de reconceptualisation et de codage moindre, meilleure résistance à la divergence de threads qui est inévitable dans le cadre du problème de la rentrée atmosphérique de satellites, coûts moindre et de meilleures performances en temps de calcul.

Dans un second temps nous avons considéré l'utilisation de l'outil d'analyse statistique de dérive descente de ballons stratosphériques sur GPU, baptisé Aquilon, pour la réanalyse de la dérive descente sous parachute des vols de la campagne ballon de Timmins 2014. Nous avons dans un premier temps présenté une méthode de détermination des coefficients d'incertitudes pour les paramètres météo se basant sur l'utilisation des données actuellement disponibles pour les opérateurs au moment de la prise de décision de fin de vol. Cette méthode consiste ainsi à utiliser les écarts mesurés entre le profil météorologique mesuré par le ballon durant sa phase d'ascension et le profil météorologique issue de la prévision pour ce moment du vol pour déterminer les coefficients d'incertitudes (écart-type et biais) qui sont ensuite utilisés durant la phase de dérive descente. Conscient des limitations de cette méthode nous avons aussi pris le parti de les appliquer de telle façon à maximiser la surface des zones de retombées déterminées par cette méthode.

Suite à cette étude il s'avère que sur les six vols de la campagne qui ont put être réanalysés, pour quatre de ces vols le point de retombée réel figure dans la zone à 1 σ, les deux autres se situant dans la zone à 2 σ, tandis que seulement 3 de ces vols figuraient dans la zone à 1 σ avec la méthode de référence du CNES. Si les surfaces des zones probables de retombées déterminées par Aquilon sont aujourd'hui jusqu'à 3 fois plus grandes que celles déterminées par la méthode de référence du CNES, ceci constitue une borne maximale sur la qualité des résultats que peut produire cet outil. Suite à cette étude, une collaboration est ainsi envisagée avec le CNES et Météo-France afin d'améliorer la méthode de détermination des paramètres d'incertitudes ainsi que leur application afin d'éviter les écueils pointés dans l'analyse pour les vols 6 et 7.

Chapitre 5

Analyse de rentrée atmosphérique de satellites : l'outil Calima

Au Chapitre 1 nous avons présenté un bref état de l'art sur la problématique des débris spatiaux et comment celle-ci est traitée par les différentes agences spatiales internationales. Nous avons notamment mis en évidence les limitations des méthodes déterministes des simulations numériques qui sont mises en œuvre afin de prévoir la rentrée atmosphérique d'engins spatiaux. Dans ce présent chapitre nous allons maintenant présenter comment nous avons développé un outil d'analyse statistique dédié à l'étude de la rentrée atmosphérique de satellites : Calima.

5.1 Introduction

La rentrée atmosphérique de satellite est un problème complexe dont la modélisation implique la prise en compte de différents processus physiques. Afin de modéliser le comportement des satellites durant la phase de rentrée, les outils orientés objets (qui ont notamment été présentés au Chapitre 1) utilisent des modéles approchés des phénomènes physiques mis en jeux. De façon générale on peux diviser la modélisation dans ces logiciels en deux parties, et c'est aussi ce que l'on fait dans notre logiciel de rentrée atmosphérique. La première partie est dédiée à l'intégration des équations du mouvement du satellite aprés avoir calculé les forces aérodynamiques s'exerçant sur le satellite, ces forces provenant de l'intéraction du satellite avec l'atmosphère (dans le cadre de notre logiciel c'est ce qui est fait dans le bloc Calcul Aérodynamique de la Figure 5.2). La modélisation du mouvement oblige notamment à travailler avec différents référentiels (cf. Annexe E.1) en particulier les forces aérothermodynamiques s'exercent dans le référentiel Aéro (représenté par le référentiel O'x'y'z' de la Figure 5.1) et le mouvement de l'objet lui s'exprime dans le référentiel Terrestre (représenté par le référentiel Oxyz sur la Figure 5.1). En plus du mouvement du satellite il faut aussi prendre en

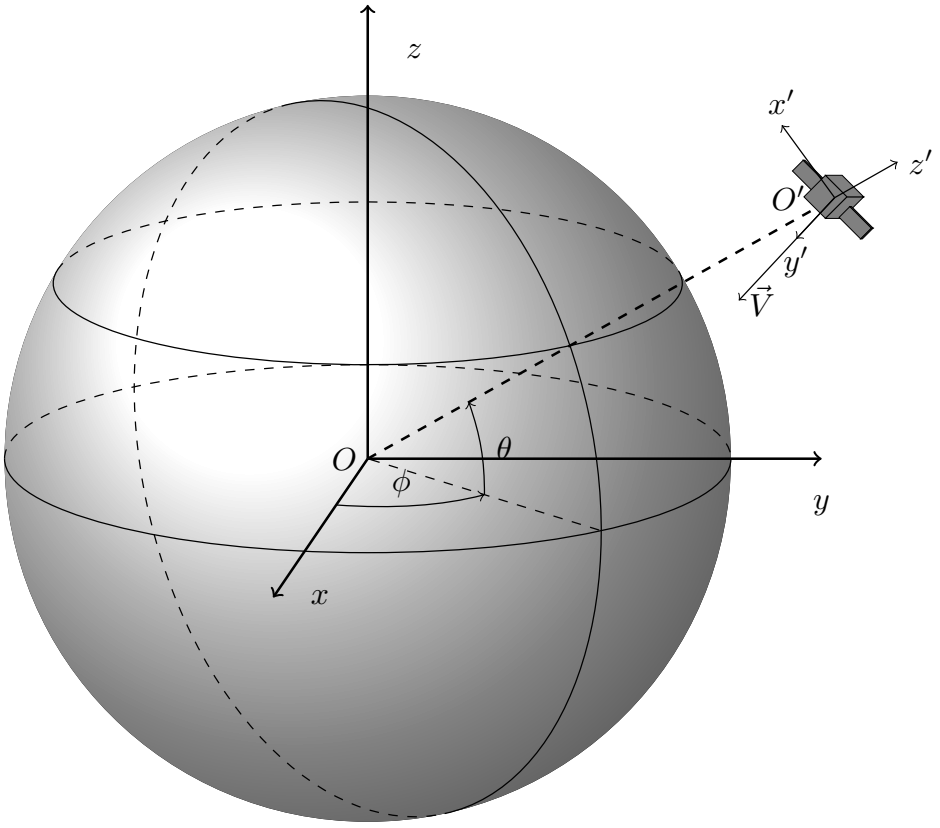

FIGURE 5.1: Présentation du référentiel Terrestre (Oxyz), en rotation uniforme autour de l'axe z, et du référentiel de la trajectoire de vol (O'x'y'z') pour un satellite durant une rentrée atmosphérique

considération les efforts thermodynamiques qu'il subit en calculant l'échauffement des matériaux et éventuellement l'ablation (la perte de masse) de la structure (dans le cadre de notre logiciel c'est ce qui est fait dans le bloc Calcul Thermodynamique de la Figure 5.2 où nous considérons les différents flux de chaleurs convectifs et radiatifs afin de modéliser l'évolution thermique des matériaux). Notre logiciel suivant une démarche orientée objet, les calculs des forces et des efforts thermodynamiques ne sont pas effectués en calculant directement les efforts sur la structure réelle (ce qui est fait dans le cadre des logiciels orientés engins spatiaux) mais en calculant ces efforts sur une sphère de dimensions équivalentes qui est ensuite corrigée par un facteur de forme. Là où dans les autres outils ces facteurs de formes sont calculés à partir de formules analytiques simplifiées déterminées de façon empirique à partir de résultats de simulations CFD ou d'expériences, nous avons pris le parti de prédéterminer ces cœfficients pour une simulation donnée en nous inspirant des méthodes employées dans les logiciels orientés véhicules spatiaux via un module dédié (module aéroCœf de la Figure 5.2). Nous avons opté pour cette approche hybride car elle nous permet de tendre vers la précision de calcul des logiciels orientés véhicules spatiaux tout en nous permettant de maîtriser les

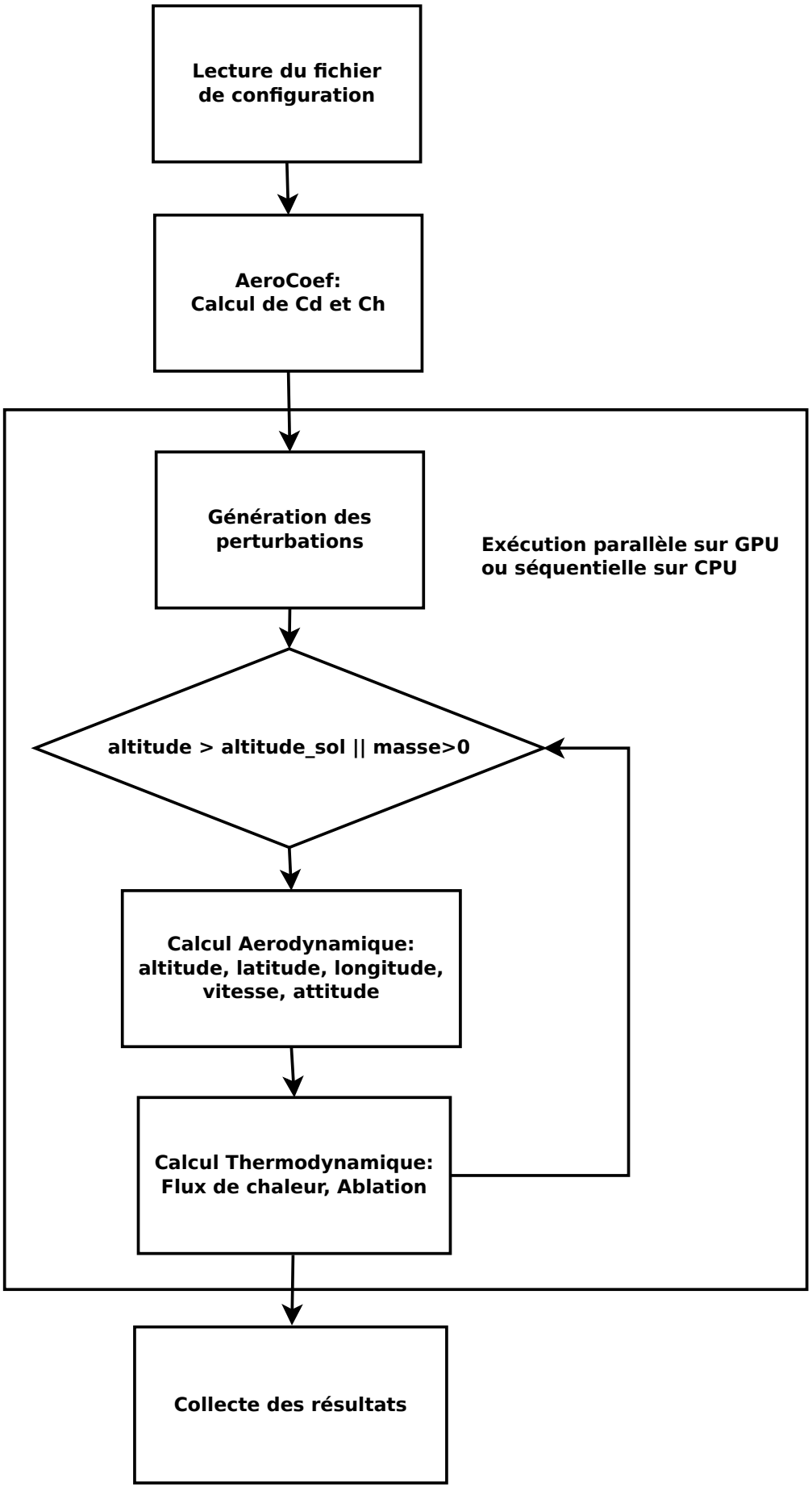

FIGURE 5.2: Algorithme de Calima

temps de calculs.
Tous les outils actuels de prévision de rentrée atmosphériques d'engin spatiaux sont de type déterministe, engendrant donc un seul résultat par simulation. Lorsqu'on considère la question de la survivabilité des objets effectuant une rentrée atmosphérique ce résultat est binaire : soit l'objet arrive au sol à un endroit donné avec une masse donnée, soit il se désintègre à une altitude donnée. Cependant une légère modification des paramètres initiaux de la simulation, des paramètres de calcul ou des modèles peut faire passer le résultat de l'un à l'autre. Tous les outils orientés objets utilisent des modélisations approchées des phénomènes physiques mis en jeux durant la phase de rentrée afin de maintenir un temps de calcul acceptable, mais qui posent la question de la vraisemblance de ces résultats. Avec l'évolution de la technologie des processeurs, la stratégie actuelle pour les nouveaux logiciels de rentrée atmosphérique consiste à mettre en œuvre des modèles physiques de plus en plus complexes et de plus en plus réalistes sur des géométries qui sont les plus réalistes possibles. Cette approche qui est celle employée par les logiciels orientés vaisseaux spatiaux demeure malgré tout coûteuse en terme de temps de calcul, les résultats obtenus sont toujours déterministes et restent toujours aussi sensibles aux conditions initiales et aux modèles.
Ainsi dans le cadre de cette thèse nous avons pris le parti d'avoir une approche radicalement différente de ce qui est fait à l'heure actuelle, puisque notre volonté est d'utiliser des modélisations physiques de type orienté objet (c'est à dire simplifiées) mais en prenant en considération toutes les incertitudes qui existent à l'heure actuelle aussi bien sur les paramètres des modèles employés que sur les conditions initiales des simulations, afin de pouvoir réaliser de façon native des analyses statistiques sur la dispersion des résultats. Les analyses statistiques étant généralement coûteuses en temps de calcul, l'outil tirera aussi profit des capacités de calcul parallèle des accelérateurs de calculs et notamment des GPUs et sera capable de mener des analyses de Taguchi (cf. Chapitre 2) afin de réduire le nombre de paramètres à considérer pour l'analyse statistique, le tout dans le but de réduire le temps de calculs de l'analyse.
Ce chapitre est ainsi dédié à la présentation de l'outil de rentrée atmosphérique de satellite accéléré sur carte graphique : Calima.

5.2 Présentation générale de Calima : démarche et cahier des charges

Comme dit en introduction, l'objectif avec Calima est d'avoir à disposition un logiciel de rentrée atmosphérique, permettant de réaliser des analyses statistiques accélérées sur GPU, en utilisant des modélisations simplifiés et en prenant en compte les incertitudes

sur les conditions initiales de simulations ainsi que sur certains des paramètres de simulation. Néanmoins, toutes les plateformes de calculs ne sont pas forcément équipées de cartes graphiques ou d'accélérateurs de calcul compatibles pour le calcul CUDA. Pour cela Calima doit aussi être à même de réaliser les mêmes calculs en utilisant le CPU à la place d'un GPU. À cette fin, et dans une logique de conception industrielle, l'ensemble de contraintes suivantes a été proposé durant le développement de l'outil constituant un cahier des charges auquel doit répondre le logiciel produit:

- Calima devra offrir la possibilité de faire les calculs sur CPU ou sur GPU en fonction du souhait de l'utilisateur et des moyens de calculs disponibles sur la plateforme utilisée afin d'optimiser au maximum son degré de portabilité.
- Calima devra produire les mêmes résultats quelque soit la plateforme de calcul considérée CPU ou GPU, que l'on considère une simulation nominale ou une analyse statistique ayant convergée.
- Pour des raisons de maintenabilité de code et pour garantir la non divergence des résultats, autant que possible le même code sera employé pour la version CPU et pour la version GPU de Calima.

Ces contraintes étant établies nous avons donc développé Calima afin que celui-ci soit modulable pour s'adapter aux besoins de l'utilisateur ainsi qu'aux moyens de calcul disponibles, le choix de la plateforme de calcul se faisant à la compilation. Ainsi le même code est compilé soit pour le CPU, soit pour le GPU générant deux exécutables différents, respectivement Calima_CPU et Calima_GPU qui utilisent les mêmes fichiers source. Nous détaillerons plus loin la mise en œuvre.

5.3 Conception d'un logiciel de rentrée atmosphérique de satellite sur GPU et CPU

5.3.1 Conception de l'ensemble des fonctionnalités

Outre la possibilité de mener des calculs statistiques de façon native, un des points au cœur de la conception de Calima est la réalisation d'un logiciel ayant une approche hybride, à savoir qu'il utilise des modèles aéro-thermodynamiques simplifiés à la façon des logiciels orientés objets, mais qui puissent aussi permettre de traiter des formes géométriques plus complexes à la façon d'un logiciel orienté engin spatiaux. C'est dans cette optique que le cœur de calcul de Calima est constitué d'un intégrateur numérique de type Runge-Kutta 4 pour intégrer les équations du mouvement à 6 degrés de liberté (6DDL) et utilisant pour le calcul des forces aéro-thermodynamiques des cœfficients pouvant soit être imposés par l'utilisateur, soit calculés sur la géométrie réelle de l'objet

simulé en amont du calcul d'intégration par un module dédié (le détail de la mise en œuvre et des modèles figurent dans les sous-sections suivantes).

5.3.1.1 Contrainte et mise en œuvre

Dans le cahier des charges figure l'obligation de modularité du code qui sera produit afin qu'il soit possible de le déployer sur une plateforme contenant des GPUs ou non, mais aussi pour que l'utilisateur ait la possibilité de choisir quelles fonctions il veut utiliser pour son calcul (utilisation du module de calcul des cœfficients, calcul aerothermodynamique ou sans thermique,...).
Afin de répondre à cette exigence nous avons utilisé les deux solutions suivantes :

- La première se trouve au niveau de la compilation. En effet, nous avons élaboré un Makefile qui permet, à partir du même code source de Calima, d'engendrer un exécutable sur CPU ou sur GPU. Selon la plateforme et les bibliothèques installées sur la machine, l'utilisateur pourra choisir de compiler soit un exécutable réalisant uniquement des simulations de rentrée atmosphérique, soit un exécutable pré-calculant les cœfficients aéro-thermodynamiques avant de réaliser les simulations.

  Pour ce faire le Makefile contient la liste de tous les codes sources du projet et accepte deux options : "compil" et "cœf". L'option "compil" accepte ainsi deux valeurs CPU ou GPU afin de désigner la plateforme pour laquelle on souhaite compiler le code. Selon la valeur sélectionnée le Makefile va ensuite modifier l'extensions des fichiers contenant le code source selon la plateforme cible : .cpp pour le CPU et .cu pour le GPU. Les fichiers sources sont ensuite compilés indépendamment avec le compilateur approprié (gcc ou nvcc) pour engendrer les objets compilés avant d'être ensuite lié au sein du même binaire exéctutable. L'option "cœf" quand à lui prend deux valeurs TRUE ou FALSE selon que l'utilisateur souhaite compiler le module de calcul des cœfficients aéro-thermodynamiques ou non. La liste des fichiers sources de ce module est alors rajouté à la chaîne de compilation décrite ci-dessus.

- La seconde solution se trouve au niveau de l'exécution. En effet, le code contient un grand nombre de paramètres de contrôle qui peuvent être configurés au moment du lancement de l'exécutable et qui permettent d'activer ou désactiver certaines fonctions tel que le pré-calcul des cœfficients (si le module est compilé), la réalisation de calcul statistiques ou non, l'utilisation des calculs thermiques...

Dans la section suivante nous allons présenter quels ont été les choix effectués pour la conception du code.

5.3.2 Conception du code

5.3.2.1 Problème de l'équivalence CPU-GPU

Le point qui a posé le plus de défis lors de la phase de conception de l'algorithme de Calima a été le respect de la règle de réutilisation maximale du code entre la version CPU et la version GPU de l'outil. En effet, par exemple sur la version CPU nous avons pris le parti d'utiliser la bibliothèque Boost afin de définir les vecteurs et les matrices, hors il est impossible d'utiliser les fonctions de Boost pour déclarer un vecteur depuis un kernel. De la même façon certaines fonctions, telles que la fonction de perturbation ne prennent pas le même nombre d'arguments selon que l'on considère la version CPU ou la version GPU. Pour le cas de la fonction de perturbation, la version GPU prend entre autre comme argument la graine spécifique à la thread courante (cf. sous-section 5.6.2 pour plus de détails sur la génération des perturbations) ce qui n'est pas le cas de la version CPU.

5.3.2.2 Mise en œuvre de la solution

Dans le cadre de la modélisation de la rentrée atmosphérique de satellite, la grande majorité des vecteurs manipulés sont des vecteurs à 3 éléments, tandis que pour les matrices se sont des matrices 3x3. Ainsi afin de pouvoir produire un code unique pouvant être à la fois compilé pour le CPU et pour le GPU, nous avons eu recours à l'utilisation d'alias, créant 2 types de variables "vector_type" et "matrix_type", qui selon le compilateur utilisé pour engendrer le binaire sont des alias des fonctions de déclaration de vecteur et de matrice sur le CPU ou le GPU. Le Pseudo-code 5.1 présente comment cela est géré au niveau du code. L'instruction pour le préprocesseur #ifdef __NVCC__ sert à identifier si le compilateur utilisé est le compilateur CUDA nvcc, ce qui signifierait que le code est compilé pour le GPU. Grâce à cela, à chaque fois que dans le code un vecteur ou une matrice est déclaré avec ces alias, lorsque le code sera compilé pour le CPU, le compilateur ira chercher dans la bibliothèque uBLAS de Boost les instruction pour déclarer la matrice ou le vecteur, tandis que si le code est compilé pour le GPU, le compilateur ira chercher dans les fichiers d'en-tête nvVector.h et nvMatrix.h les instructions de déclaration. Il est à noter que ces deux bibliothèques font à l'origine partie du SDK de NVIDIA, et servent à déclarer des vecteurs et des matrices de dimensions prédéfinies (CUDA ne permettant pas d'allocation dynamique). Les versions de ces bibliothèques utilisées dans Calima sont des versions que nous avons modifiées entre autre pour avoir des vecteurs à 3 éléments et des matrices 3x3, pour qu'elles permettent de déclarer des vecteurs et des matrices sur le GPU depuis le CPU. Aussi pour uniformiser la syntaxe d'appel aux éléments des vecteurs et des matrices avec ce qui est fait dans Boost. A titre d'exemple pour accéder au premier élément d'un vecteur A dans Boost on utilise la syntaxe suivante : A(0) tandis que dans nvVector on utilise la syntaxe suivante : A[0],

```
//Liste des bibliothèque à inclure
#include ....

//Compilation pour le GPU
#ifdef __NVCC__
//Liste des bibliothèque à inclure uniquement pour la
   version GPU
#include ...

//Définition type de donnée matrice et vecteur sur GPU
typedef nv::vec3<double> vector_type;
typedef nv::matrix3<double> matrix_type;
#else
//Définition type de donnée matrice et vecteur sur CPU
typedef ublas::vector<double> vector_type;
typedef ublas::matrix<double> matrix_type;
#endif
```

PSEUDO-CODE 5.1: Définition de variables de type vecteur et matrice sur CPU et GPU

nous avons donc corrigé ce type de divergence syntaxique en modifiant le code source des fichier d'en-tête nvVector.h et nvMatrix.h, encore une fois dans l'optique de pouvoir écrire un code unique, pouvant être compilé pour une exécution sur le CPU ou sur le GPU.

Pour les fonctions n'utilisant pas le même nombre d'arguments ou d'autres points particulers dans le code (des vecteurs à 6 éléments par exemple ou l'utilisation de certaines bibliothèques), la même gestion basée sur la détection du compilateur est mise en œuvre : la déclaration et l'appel de la fonction ou de la variable sont doublés pour permettre la double compilation.

Cette solution nous permet ainsi de maximiser la réutilisation de code entre les versions CPU et GPU avec la contrainte de produire deux exécutables distincts (un pour le CPU et un pour le GPU). Les matrices et les vecteurs ne sont pas les seuls types de variables particulières que l'on manipule dans Calima. Il y a aussi les quaternions (cf. Annexe E.2) qui sont des objets mathématiques permettant d'effectuer des changements de repères. Devant l'absence de bibliothèques permettant la création et la manipulation de quaternion sur GPU, nous avons repris la bibliothèque de quaternion de Boost, afin de la modifier pour la rendre compatible avec une utilisation sur GPU. Cela a consisté à rajouter les décorateurs __host__ __device__ devant toutes les fonctions de la bibliothèque, doubler les définitions de certaines fonctions en remplaçant les appels au vecteurs de ublas, par des appels aux vecteurs de nvVector.

A titre informatif, cette opération a été rendue possible grâce à l'indépendance du fichier d'en-tête quaternion.h au sein de Boost. En effet une telle tâche s'avère quasi-impossible par exemple pour les fonctions de déclaration des vecteurs et des matrices dans uBLAS

en raison du trop grand nombre de dépendances entre les différents fichiers d'en-tête composant la bibliothèque.

5.3.2.3 Programmation orientée objet sur GPU

L'autre point à traiter durant cette phase de conception était l'exigence de maintenabilité du code. Nous avons ainsi pris le parti d'adopter une programmation orientée objet, et utilisant autant que possible des éléments issus de différentes bibliothèques afin de s'affranchir autant que possible des problèmes de validation. Calima est ainsi articulé autour de deux classes principales :

- La première classe SimulationParameters contient la liste de tous les paramètres de configuration de la simulation : le type de calcul mené, avec quels modèles, la grille de calcul pour l'exécution sur GPU,...
- La seconde classe SimulatedObject, contient quand à elle toutes les variables et les propriétés de l'objet que l'on cherche à simuler, ainsi qu'une partie des fonctions de calcul.

Mais cette approche pose la question particulièrement épineuse de sa mise en œuvre sur GPU.

5.3.2.4 Mise en œuvre de la solution

Bien que la programmation orientée objet soit très courante sur CPU, il a pendant longtemps été pratiquement impossible de l'employer sur GPU. En effet, il était particulièrement compliqué de définir un objet sur le GPU depuis le CPU, puisqu'il fallait commencer par déclarer la structure de l'objet avant de déclarer les éléments au sein de l'objet puis de copier un à un ces éléments dans l'objet. De fait au début de mes travaux de thèses cette approche était très difficilement envisageable. C'est d'ailleurs pour cette raison que le code de dérive descente présenté précédemment est codé de façon procédurale. Mais l'introduction de la notion de mémoire unifiée entre le CPU et le GPU, juste avant d'entamer la phase de conception de Calima a changé la donne. En effet, il est maintenant possible de définir la classe Managed présentée au Pseudo-code 5.2. Il est donc possible de définir une classe héritant de celle-ci. Ainsi, lors de la déclaration d'un objet ce dernier est automatiquement déclaré dans la mémoire unifiée, ce qui dispense le programmeur des problèmes de gestion des données entre le CPU et le GPU. Au niveau de la déclaration de la classe, nous avons ainsi implémenté la même solution de gestion que présenté précédemment basé sur l'identification au niveau du préprocesseur de compilateur utilisé comme illustré sur le Pseudo-code 5.3 pour la classe SimulationParameters. Grâce à cela, la même classe peut être utilisé dans un code qui sera compilé pour le CPU ou pour le GPU.

```
class Managed
{
public:
  void *operator new(size_t len)
  {
    void *ptr;
    cudaMallocManaged(&ptr, len);
    return ptr;
  }

  void operator delete(void *ptr)
  {
    cudaFree(ptr);
  }
};
```

Pseudo-code 5.2: Classe Managed qui permet de déclarer un objet sur GPU dans la mémoire unifiée

```
#ifdef __NVCC__
class SimulationParameters: public Managed
#else
class SimulationParameters:
#endif
{
  //Définition des paramètres
};
```

Pseudo-code 5.3: Gestion de la déclaration des classes pour le CPU et le GPU

5.3.3 Entrée/sorties

Afin de pouvoir configurer tous les paramètres de calculs ainsi qu'initialiser toutes les variables pour une simulation donnée, Calima se base sur les données renseignées dans un fichier de conditions initiales. Nous avons choisi de construire ce fichier au format XML, ce dernier permettant une bonne lisibilité ainsi qu'une hiérarchisation des données. Le Pseudo-code 5.4 présente de façon succinte l'organisation du fichier de configuration de Calima. Comme on peut le voir celui-ci contient les conditions initiales sur l'objet à étudier, la liste des paramètres de configuration de Calima, les paramètres de configurations du calcul sur GPU ainsi que la liste des informations nécessaires pour le précalcul des grilles de cœfficients aérothermodynamiques. A noter qu'il est possible de rajouter pour certains de ces paramètres des attributs dans le XML contenant les informations nécessaires pour les perturber en vu d'une analyse statistique (cf. section 5.6).
En sortie, Calima produit deux fichiers. Le premier en .dat, au format Tecplot [3], contient les valeurs des facteurs de vue et des Cp de chaque élément du maillage, pour une représentation 3D de ces valeurs. Le second en .txt, contient la liste des résultats finaux de chaque simulation, chaque ligne de ce fichier correspondant à une simulation

```
<?xml version='1.0' encoding='UTF-8'?>
<init>
  <calima> Initialisation des variables et des paramètres pour la simulation de rentrée
        <fichier_materiaux>materiaux.xml</fichier_materiaux> fichier contenant les propriétées des différents
      matériaux
        <objet1>
            <object>valeur</object> Longeur de référence, surface de référence, masse et matériaux de l'objet
      simulé
            <Matrice_d'inertie> valeurs </Matrice_d'inertie> Matrice d'inertie de l'objet simulé
            <Position> valeurs </Position> altitude, latitude, longitude et azimuth
            <Mouvement> valeurs </Mouvement> vitesse, attitude
            <Coefficients> valeurs </Coefficients>   température de paroi
        </objet1>
    <parametres> valeurs </parametres> Paramètres de la simulation: pas de temps, nombre de simulations...
    <parametres_GPU> valeurs </parametres_GPU> Paramètres de la grille de calcul sur GPU
  </calima>
  <aeroCoef> Initialisation des variables et des   paramètres pour le calcul des coefficients aéro-thermo
    <rayons> valeurs </rayons> Paramètres pour le lancé des rayons, sélection du maillage, etc...
    <parametres> valeurs </parametres> Paramètres de configurations des calculs des coefficients aéro-thermo
  </aeroCoef>
</init>
```

PSEUDO-CODE 5.4: Présentation schématique du fichier de configuration de Calima

et chaque colonne correspondant à un paramètre :
Temps, nombre d'itérations, altitude finale, longitude, latitude, attitude, vitesse finale et énergie cinétique au point d'impact.

5.3.4 Présentation de l'algorithme

De façon très schématique on peut représenter Calima par trois blocs s'exécutant en séquence l'un à la suite de l'autre comme on peut le voir sur la Figure 5.2 : le premier sert à charger tous les paramètres de simulation en allant des conditions initiales de simulation jusqu'à la sélection des modèles physiques utilisés pour la modélisation de la rentrée. Le second bloc est le module de calcul des cœfficients aero-thermodynamiques, son fonctionnement sera présenté en détail à la section 5.4. Enfin le dernier bloc sert à la génération des jeux de conditions initiales et au calcul des simulations. Ce bloc s'exécute de façon séquentielle sur CPU ou bien de façon parallèle sur GPU. Si l'on regarde le détail de ce bloc présenté sur la Figure 5.2, on constate que la première étape de ce bloc consiste à engendrer les valeurs de conditions initiales à partir des données de perturbations figurant dans le fichier de configuration (cf. section 5.6). Ensuite tant que l'objet simulé n'a pas atteint le sol, ou ne s'est pas complément désintégré durant le processus de rentrée, ce bloc intègre temporellement le système, en commençant par résoudre les équations aérodynamique (cf. section 5.7) puis les efforts thermodynamiques (cf. section 5.8). Enfin les résultats de chaque simulation sont stockés dans un seul fichier. A l'heure actuelle le post-traitement ne se fait pas encore au sein de Calima mais est assuré par un script Python externe (cf. section 5.9). De façon générale dans les sous-sections suivantes nous présenterons les principaux modèles physiques ainsi que leur mise en œuvre, pour une présentation plus exhaustive des équations considérées dans chaque module nous renvoyons le lecteur à l'Annexe E

5.4 Module aeroCœf

5.4.1 Présentation du module aeroCœf

Comme dit précédemment l'un des objectifs au cœur du développement de Calima est d'avoir la possibilité d'étudier le comportement de formes complexes (par opposition aux formes simples : pavés, sphères, plaques ou cylindres) malgré une modélisation orientée objet. Ce processus passe par le calcul de cœfficients de forme aérodynamique et aérothermodynamique permettant de calculer les flux et les forces exercées sur une forme complexe à partir des flux et des forces calculés sur une forme plus simple.
Le module aeroCœf permet ainsi de calculer les cœfficients aérodynamiques et aérothermodynamiques en résolvant les équations des forces aérodynamiques et des flux de chaleur sur un maillage représentant la forme du satellite que l'on veut étudier.
Afin de réaliser ces calculs il est nécessaire de déterminer quelles sont les cellules du maillage de l'objet étudié qui sont effectivement exposées à l'écoulement de l'air sur l'objet en rentrée atmosphérique et lesquelles sont masquées. Afin de déterminer quelles sont ces cellules qui sont effectivement exposées et celles qui sont masquées, on procède au calcul d'un cœfficient appelé facteur de vue (View Factor, VF), compris entre 0 et 1 ; 0 indiquant que la cellule du maillage est totalement masquée, tandis qu'une cellule avec un facteur de vue de 1 voit totalement l'écoulement. La méthode de détermination de ces facteurs de vue consiste à tirer des rayons depuis différents points à la surface de chaque cellule du maillage dans la direction de l'écoulement et de détecter si ces rayons interceptent une autre cellule du maillage. Ces points sont déterminés aléatoirement pour chaque cellule, sauf dans le cas où l'on ne tire qu'un seul rayon ce dernier est alors lancé depuis le centre de la cellule.
La singularité du module aeroCœf par rapport aux autres outils de calcul de facteur de vue est d'utiliser la bibliothèque OptiX Prime de NVIDIA qui permet de réaliser le lancer de rayons et la détection de collision de façon parallèle sur CPU ou sur GPU.
La Figure 5.3 présente une vue schématique de l'algorithme du module de calcul des cœfficients aeroCœf.

5.4.2 Utilité des cœfficients aérothermodynamique

Dans le cœur du calcul (blocs aéro et thermodynamiques) pour des raisons de temps de calculs et de moyens mis en œuvre, Calima ne détermine pas directement les forces aéro-thermodynamiques (pression aérodynamique, cf. équation (5.11), flux de chaleur convectif, cf. Annexe E.4.3.2) sur la structure réelle de l'objet. A la place on résout ces équations sur un objet sphérique de surface et de dimensions équivalentes que l'on vient ensuite corriger par un facteur de forme.
Ainsi pour déterminer la force de traînée s'exerçant sur un objet quelconque, on calcule

la force sur la surface d'une sphère de rayon équivalent que l'on multiplie ensuite par un cœfficient de traînée. De la meme façon pour le calcul thermique, Calima considère le flux convectif au point d'arrêt pour une sphère de rayon équivalent que l'on multiplie ensuite par le cœfficient thermique. Si il est possible de connaître et donc d'imposer ces cœfficients pour des formes géométriques simple, la chose devient beaucoup plus compliquée lorsque l'on considère des formes plus complexes d'autant que ces cœfficients dépendent aussi de l'écoulement autour de la structure (sa vitesse, sa densité, sa température,...) ainsi que de l'attitude de la structure. C'est là que le module aeroCœf intervient, en permettant de pré-calculer les valeurs de ces cœfficients pour qu'ils puissent ensuite être utilisés dans le cœur de calcul de Calima.

5.4.3 OptiX Prime

La bibliothèque OptiX [51] est une bibliothèque développée par la société NVIDIA en 2008, à l'origine pour permettre le lancer de rayon dans les outils de rendu afin de modéliser l'éclairage ainsi que les jeux de lumières pour des films d'animations ou des infographies 3D [60].
OptiX Prime est un composant de la bibliothèque OptiX qui sert au lancé de rayon ainsi qu'à la détermination d'intersection de ces rayons avec des éléments constituants la scène.
Concrètement l'utilisateur commence par définir une "scène" qui s'avère être un maillage d'un objet 3D au format OBJ ainsi qu'une liste de points sources à partir desquels les rayons seront lancés ainsi que la direction de propagation de chaque rayon. OptiX lance ensuite chaque rayon et détermine pour chaque rayon si celui-ci a intersecté une ou plusieurs des cellules constituant la scène considérée ou aucun. Avec OptiX Prime ce processus peut être effectué en parallèle soit sur GPU, soit sur CPU, mais avec deux fonctions différentes. En effet l'API d'OptiX n'utilise pas en interne les mêmes fonctions pour la génération des rayons et produit ainsi un format de donné différent pour la définition des rayons selon que l'on utilise le CPU ou le GPU. Il est donc nécessaire, si l'on souhaite avoir le loisir de choisir la plateforme que l'on utilise, d'implémenter deux appels différents aux fonctions d'OptiX Prime pour la génération des rayons [51].
Enfin il est à noter que tous les calculs effectués par OptiX se font sur des variables à virgule flottante en simple précision, ce qui induit certaines imprécisions dans les calculs qui mènent à certains comportements symptomatiques dont nous discuterons plus loin. Notons que ces comportements ne sont pas propres à OptiX mais se retrouvent dans toutes les bibliothèques de lancer de rayon [18].

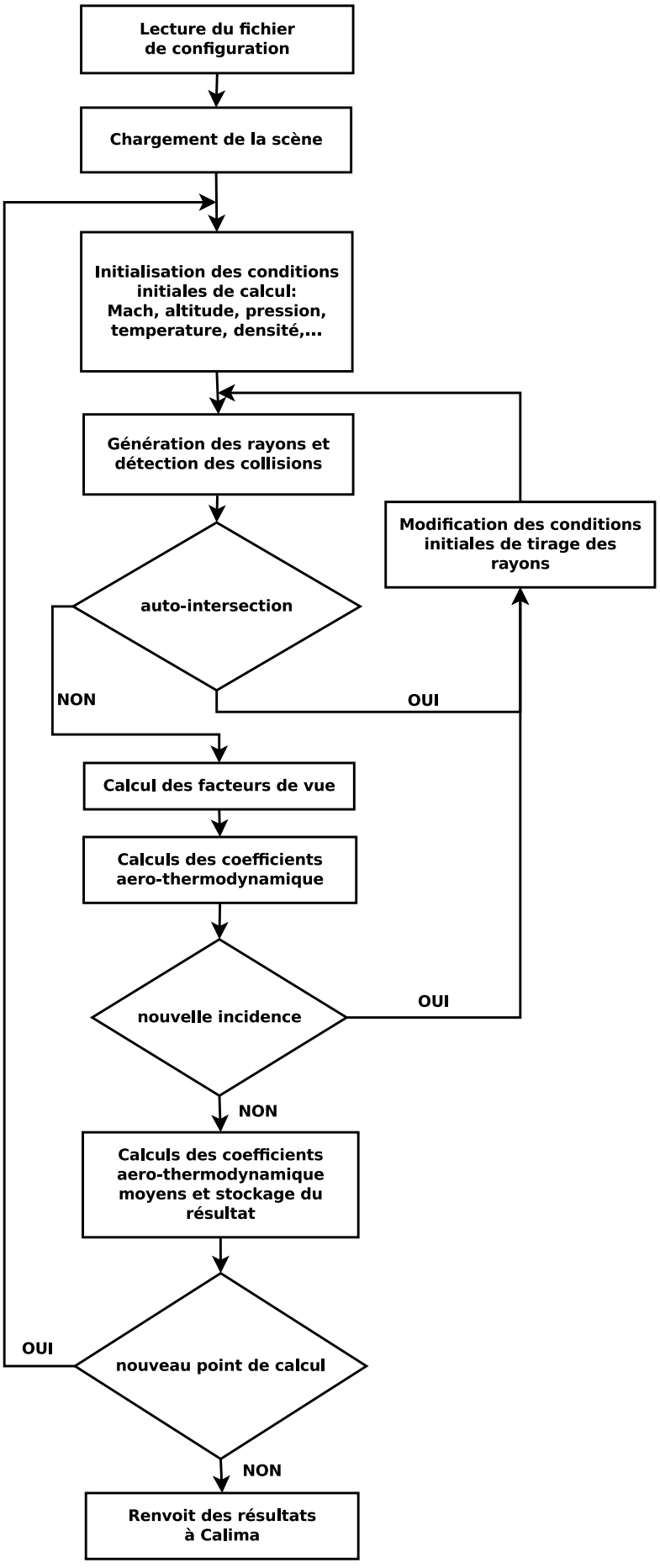

FIGURE 5.3: Algorithme de aeroCœf

5.4.4 Utilisation et compilation

5.4.4.1 Utilisation du module aeroCœf

A l'origine, nous avons développé le module aeroCœf indépendamment de Calima avant de l'intégrer au sein de celui-ci. De cette indépendance d'origine le module a conservé sa propre structure de classes, sa propre fonction d'initialisation, etc... ainsi depuis le module de calcul de Calima, aeroCœf apparaît comme une fonction qui accepte en argument le fichier de configuration des calculs et qui renvoi les listes de valeurs des cœfficients aero-thermodynamiques calculés pour différentes conditions définies dans le fichier de configuration.

5.4.4.2 Compilation du module

Comme expliqué ci-dessus (cf. sous-section 5.4.3), OptiX est une bibliothèque NVIDIA à en tête partiellement pré-compilé. En fait il existe deux niveaux dans la bibliothèque Optix, un premier niveau de fonctions qui peuvent être utilisées dans un code C ou CUDA, et un second niveau de fonctions internes à la bibliothèque, qui ne sont pas accessibles au programmeur, et qui servent à la génération et à la gestion optimale des lancés de rayons.
En terme de compilation, si l'utilisateur choisit de n'utiliser que son CPU pour effectuer le lancer des rayons, tous le fichiers contenant des appels aux fonctions OptiX peuvent être compilés avec le compilateur C gcc (il n'y a pas besoin du compilateur CUDA). En revanche, si l'utilisateur souhaite faire appel au GPU, il devra écrire le kernel faisant appel aux fonctions de générations de rayons sur GPU d'OptiX dans un fichier séparé qui sera alors dans un premier temps compilé en PTX, qui est un ersatz d'assembleur dédié au GPU [52] propre à CUDA, avec le compilateur nvcc de NVIDIA. Le fichier PTX est ensuite lié de façon classique au reste des fichiers objets du projet pour engendrer le binaire exécutable.

5.4.5 Maillages

On désigne sous le nom de maillage un ensemble de points, de côtés et de faces définissant une forme géométrique (triangle, rectangle, ...) dont l'ensemble constitue un polyèdre représentant une modélisation numérique d'un objet. La Figure 5.4, représente ainsi le maillage d'un satellite par des triangles. On voit bien que la forme du satellite est rendue à partir d'un assemblage de triangles de dimensions et d'orientations différentes. L'API d'OptiX n'accepte que les maillages constitués de triangles au format OBJ. Un fichier de maillage au format OBJ (cf. Pseudo-code 5.5) contient la liste des points constituant le maillage avec leur coordonnées dans un repère (0,x,y,z) identifiés par la lettre 'v' en début de ligne pour vertex suivi de la liste des triangles avec pour chacun la

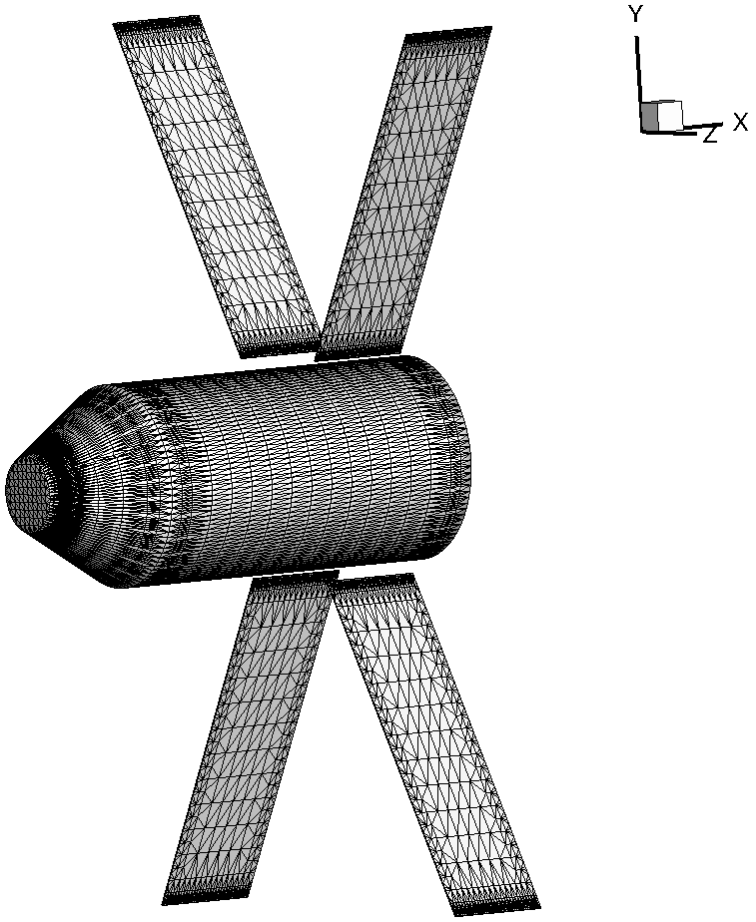

FIGURE 5.4: Exemple de maillage d'un satellite simplifié par des triangles

```
v 0.051767 0.062937 -0.000294
v 0.041727 0.086570 -0.000141
v 0.048660 0.059432 -0.000294
.
.
.
f 251 252 211
f 251 211 253
f 253 211 202
f 253 202 254
f 254 202 255
.
.
.
```

PSEUDO-CODE 5.5: Exemple de fichiers de maillage au format OBJ

liste des points (vertex) le constituant et identifiés par la lettre 'f' en début de ligne. De façon générale dans le suite du manuscrit nous utiliserons le terme cellule pour désigner un triangle, constituant le maillage.

5.4.6 Calcul des facteurs de vue

5.4.6.1 Lancé des rayons et calcul des facteurs de vue

Le module aeroCœf commence par charger le maillage de l'objet à étudier, constitué de M cellules. Afin de calculer les facteurs de vue, on lance N rayons par cellule, chacun

ayant pour origine une position aléatoire sur la cellule et étant lancé dans la direction de déplacement de l'objet (cette direction est opposée à celle de l'écoulement). L'outil lance donc les N*M rayons et identifie ceux qui sont entrés en collision avec une autre cellule du maillage. Grâce à cette méthode on obtient ainsi pour chaque cellule le nombre de rayons ayant subi une intersection $Nbr_{Intersect}$. On calcule alors pour chaque cellule son facteur de vue VF via l'équation (5.1) suivante :

$$VF = 1 - \frac{Nbr_{Intersect}}{N}, \tag{5.1}$$

Le grand intérêt de cette méthode est de permettre, grâce à l'utilisation des GPUs, de calculer l'ensemble des facteurs de vue pour un maillage donné en des temps très courts par rapport aux temps constatés dans les autres outils. De plus cette méthode permet de déterminer des facteurs de vues partiels pour les cas où seulement une partie de la cellule serait masquée, par opposition aux méthodes mises en œuvre dans d'autres outils, tel que Pampero, et qui ne lancent qu'un seul rayon, pour des raisons de temps calcul, fournissant un résultat binaire. Ce traitement de la détermination du facteur de vue permet un calcul des cœfficients aéro-thermodynamiques plus fidèle ce qui est inédit pour un logiciel orienté objet.
La Figure 5.5, représente ainsi les valeurs de facteurs vue pour le maillage de satellite déjà considéré, pour un flux dirigé selon l'axe Y ($\alpha = 0$ et $\beta = 90$). Typiquement on note bien l'influence des panneaux solaires qui masquent complètement ou partiellement certaines cellules, comme on peut le voir sur la Figure 5.6 qui présente une vue zoomé sur le corps du satellite, là où un des panneaux solaires masque tout ou partie de l'écoulement.

5.4.6.2 Gestion de la génération des rayons sur CPU et GPU

Le lancer et la résolution des intersections s'effectuent en parallèle sur le CPU et sur le GPU. De plus ce processus se fait au cœur de l'API OptiX et le programmeur n'a pas la possibilité de modifier directement cette portion de code.
En revanche il est possible pour le programmeur d'intervenir sur le processus de création des rayons afin de l'optimiser. Ce processus consiste à définir pour chaque rayon, la position de son origine sur le triangle, la distance de cette origine par rapport à la surface du triangle que l'on appelle la tolérance (cette tolérance est un paramètre critique comme nous l'expliquerons à la sous-section 5.4.6.3) et la direction dans laquelle ce rayon va se propager, dans notre cas toujours dans la direction opposée à l'écoulement.
À cause de la structure particulière des données imposée par la bibliothèque OptiX ce processus se fait uniquement de façon séquentielle lorsqu'on l'exécute sur le CPU. En revanche l'exécution sur GPU offre plus de versatilité quand à son implémentation, permettant une maîtrise complète de la façon dont ce processus est distribué sur le GPU et présentant une structure des données légèrement différentes de la version CPU.

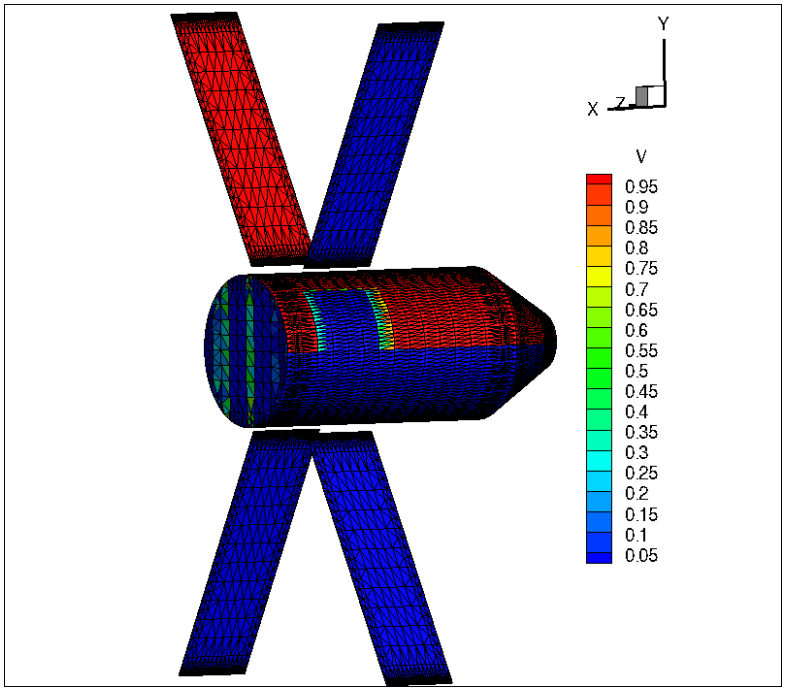

FIGURE 5.5: Facteurs de vue déterminés pour un flux dirigé le long de l'axe Y

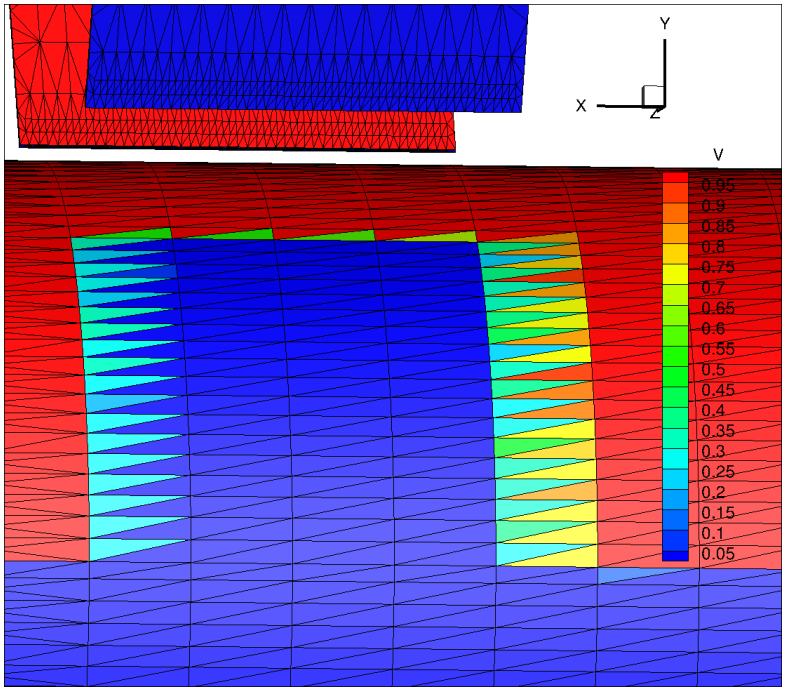

FIGURE 5.6: Facteurs de vue : Zoom sur la zone du corps du satellite masquée un des panneau solaire

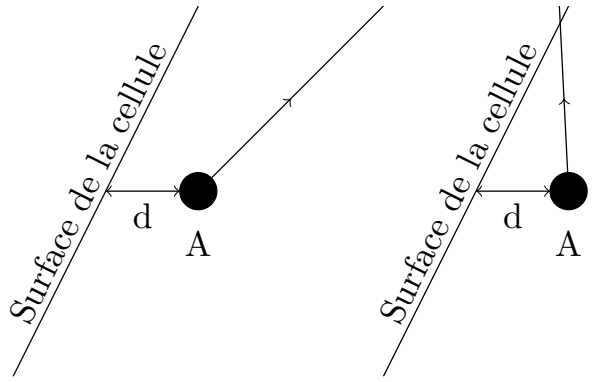

FIGURE 5.7: Principe du lancer de rayon : un rayon est lancé depuis un point source 'A' à une distance 'd' de la surface de la cellule dans une direction donnée. À gauche cas nominal, à droite cas d'auto-intersection

Nous avons ainsi mis en œuvre une méthode pour la génération des rayons consistant à allouer une cellule du maillage par thread CUDA. Le thread ainsi alloué engendre ensuite tous les rayons pour cette cellule. Cette méthode est à priori meilleure du moment que le maillage considéré contient suffisament de cellules pour occuper tous les cœurs CUDA de la carte graphique, car assurant une rentabilisation du temps passé sur le GPU et une limitation des communications CPU-GPU. En effet, en terme de communication seuls le maillage et la liste des graines pour la génération des positions aléatoires sont fournis en entrée de ce processus et la liste des rayons est récupérée en sortie. Pour ce qui est du calcul, le thread passe la majorité de son temps à générer les rayons puisqu'il n'a besoin de charger que les paramètres de la cellule qu'il traite ainsi que la graine pour la génération des positions aléatoires sur la cellule, et qu'il n'effectue cela qu'une seule fois, ces informations étant suffisantes pour permettre la génération des N rayons pour la cellule.

Ainsi lorsque l'on compare cette stratégie avec une autre consistant à faire en sorte qu'un thread ne génère qu'un rayon, on constate que notre première méthode appliquée sur un maillage constitué de 5804 triangles, en lançant 100 rayons par triangle, génère les 580 400 rayons (sur une carte NVIDIA GT630) en 0,038s, alors que la seconde ne les génère qu'en 0,078s. De plus la seconde méthode sature très vite la mémoire de la carte graphique. A noter que la même génération sur un CPU AMD FX-4100 s'effectue en 14,83s.

5.4.6.3 Gestion de l'auto-intersection

La Figure 5.7 représente le principe du lancé de rayon. Pour une cellule donnée, un rayon est lancé depuis un point, ici noté A, situé à une distance donnée 'd' de la surface de la cellule. Durant le processus d'identification des intersections, il arrive qu'un rayon généré par OptiX intersecte le triangle depuis lequel il est lancé, c'est ce l'on observe sur

le shéma de droite de la Figure 5.7. On parle alors d'auto-intersection. En effet, pour des raisons de précisions numériques, les rayons sont toujours lancés en étant décalés de la surface, et il arrive que certains rayons intersectent les triangles depuis lesquels ils sont lancés. Comme on peut le voir sur la figure de droite, Ce résultat est problématique, bien qu'il soit détecté par les outils d'OptiX, car s'il n'est pas traité il va fausser le calcul du facteur de vue. Afin d'éviter l'auto-intersection OptiX permet de jouer sur un facteur de tolérance qui va modifier la distance entre la source du rayon et la surface du triangle. Il est évidemment possible de gérer les rayons auto-intersectés en les retirant du calcul du facteur de vue, mais avec certains maillages l'expérience a montré qu'un nombre de rayons trop conséquent subissaient une auto-intersection pour pouvoir mener un calcul de facteur de vue convenable. Nous avons alors pris la décision de contourner ce problème en jouant sur le cœfficient de tolérance.
Ce cœfficient de tolérance fait partie des paramètres que l'utilisateur définit dans le fichier de configuration de Calima. Toutefois, dans le cas où la valeur fixée par l'utilisateur ne conviendrait pas, et où on détecterait une auto-intersection, l'outil va automatiquement détecter la valeur minimale de tolérance à considérer afin qu' il n'y ait pas d'auto-intersection pour le maillage et la direction du flux considérés. Il va ensuite re-générer les rayons en utilisant cette valeur de tolérance et recalculer les facteurs de vue. Il est important de noter toutefois que cette modification de la valeur de tolérance n'est pas anodine puisqu'elle peut modifier de façon importante les facteurs de vue d'un grand nombre de cellules qui n'avaient pas d'auto-intersection, impactant alors la détermination du cœfficient de traînée. De façon générale plus la valeur de ce paramètre est basse mieux c'est. Une fois les facteurs de vue recalculés le paramètres de tolérance est réinitialisé à sa valeur d'origine.

5.4.7 Calcul cœfficient aérodynamique : cœfficient de traîné C_D, cœfficient de portance C_L et cœfficient de portance latérale C_Y

5.4.7.1 Modèle

Le calcul du cœfficient de traînée pour une forme quelconque s'effectue en résolvant l'équation (5.2) :

$$C_D = -PObjet.x/(P_{dynamique} * Surface_{Référence}), \tag{5.2}$$

Le calcul du cœfficient de portance latérale pour une forme quelconque s'effectue en résolvant l'équation (5.3) :

$$C_Y = PObjet.y/(P_{dynamique} * Surface_{Référence}), \tag{5.3}$$

Le calcul du cœfficient de portance pour une forme quelconque s'effectue en résolvant l'équation (5.4) :

$$C_L = PObjet.z/(P_{dynamique} * Surface_{Référence}), \tag{5.4}$$

où $PObjet.x, y, z$ sont respectivement les composantes x, y et z de la force de pression aérodynamique exprimées dans le référentiel aérodynamique, $Surface_{Référence}$ la surface de référence de la forme considérée et $P_{dynamique}$ la pression dynamique calculée via (5.5) :

$$P_{dynamique} = 0.5 * \rho * v^2, \tag{5.5}$$

où ρ est la densité atmosphérique et v la vitesse de l'écoulement dans le référentiel de l'objet.

L'équation (5.2) permet ainsi de calculer le cœfficient de traînée pour une orientation du maillage par rapport à un écoulement donné. Or, durant une rentrée atmosphérique l'objet (satellite ou débris) ne rentre pas forcément de façon stable. En général les objets ont tendance à tourner sur eux même, soit autour d'un axe préférentiel, soit de façon aléatoire, autour de leur centre de gravité. De plus, au cours d'une rentrée atmosphérique les paramètres de l'écoulement évoluent : la vitesse change, la densité de l'atmosphère augmente en se rapprochant du sol, etc... Pour prendre en compte ces comportements nous avons mis en place dans le module aeroCœf deux processus : Pour la gestion de l'attitude du satellite, l'utilisateur a la possibilité de choisir de calculer le cœfficient de traîné selon trois attitudes : "fixe", "axe" ou "aléatoire". Dans le cas d'une attitude "fixe" on calculera ainsi les cœfficients uniquement selon la direction prescrite de l'écoulement et pour l'orientation de l'objet définie par le fichier de maillage. Dans le cas d'une attitude "axe", on supposera un objet en rotation autour d'un vecteur renseigné par l'utilisateur. Concrètement le maillage reste dans la position dans laquelle il est défini dans le fichier de maillage, et l'outil engendre différentes direction de flux autour de l'axe de rotation. Les directions ainsi engendrées sont régulièrement espacées et leur nombre dépend de la quantité d'échantillons prescrite par l'utilisateur. L'outil calculera alors une valeur pour chaque cœfficient par direction du flux échantillonné et produira les cœfficients moyens correspondant à la rotation de l'objet autour de cet axe. Dans le cas d'une attitude "aléatoire", on procède de la même manière que pour l'attitude "axe" mais en supposant une rotation autour du centre de gravité de l'objet maillé dans les 3 directions de l'espace.

Dans le cas des rotations autour d'un axe ou aléatoires, l'outil recalcule, pour chaque nouvelle orientation de l'objet, les facteurs de vu de chaque cellule.

Afin de prendre en considération l'évolution des valeurs des cœfficients durant la rentrée

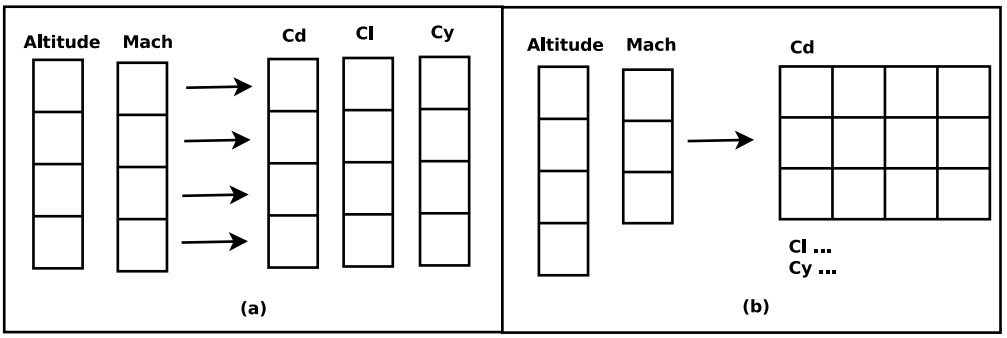

FIGURE 5.8: Mode de calcul des cœficients aérodynamiques : (a) en imposant des couples altitude/écoulement (nombre de Mach) et (b) en combinant chaque valeur d'altitude et d'écoulement

atmosphérique pour une réutilisation dans Calima, l'utilisateur a la possibilité de fournir un vecteur de vitesse d'écoulement et un autre d'altitude. Ensuite il a deux options pour relier ces deux listes :

- La première option offerte consiste à considérer les deux listes et associer élément par élément une altitude avec une vitesse d'écoulement exprimée en Mach. Ainsi la première altitude de la liste est associée au premier nombre de Mach de la liste et ainsi de suite. Pour chaque couple altitude/nombre de Mach l'outil calcule ensuite les différents cœfficients aérodynamiques qui sont stockés dans un vecteur (cf. Figure 5.8 (a)). De cette façon, lors du calcul de la rentrée atmosphérique, le cœfficient pourra être calculé à un point de simulation donné par interpolation linéaire dans ce vecteur.

- La seconde option consiste à effectuer toutes les permutations possibles entre les listes d'altitude et de vitesse d'écoulement. L'outil fournit alors une table de valeur pour chaque cœfficient, chaque valeur de cette table correspondant à un couple altitude-écoulement comme on peut le voir sur la Figure 5.8 (b). De cette façon, lors du calcul de la rentrée atmosphérique, le cœfficient pourra être calculé à un point de simulation donné par bi-interpolation linéaire dans cette table.

5.4.7.2 Mise en œuvre

Le calcul des cœfficient aérodynamiques (qui s'effectue dans le bloc "Calcul des cœfficients" de la Figure 5.3) se fait uniquement sur CPU. En effet, bien que les différentes étapes de calcul que nous allons détailler ci-dessous, requièrent d'effectuer le même calcul sur toutes les cellules du maillage considéré , elles imposent aussi de réduire fréquemment les résultats ce qui oblige à des aller-retours trop fréquents des données entre le CPU et le GPU par rapport au temps passé effectivement à calculer sur le GPU.
Le calcul des cœfficients aérodynamique commence ainsi par le calcul de la force de pression aérodynamique dans le référentiel de l'objet. Pour ce faire il faut itérer sur chaque

triangle du maillage :
Pour un triangle donné, on commence par calculer la surface de ce triangle qui voit effectivement le flux $S_{effective}$:

$$S_{effective} = S_{Triangle} * VF_{Triangle}, \tag{5.6}$$

où $VF_{Triangle}$ est le facteur de vue du triangle, et $S_{Triangle}$ la surface du triangle.
Normalement on devrait calculer la force de pression, F_P, exercée sur le triangle via l'équation suivante :

$$F_P = (P_{inf} + P_{dynamique} * Cp) * S_{effective}, \tag{5.7}$$

où P_{inf} est la pression atmosphérique, $P_{dynamique}$ la pression dynamique calculée via l'équation (5.5) et Cp le cœfficient de pression.
La valeur de Cp dépend du régime d'écoulement dans lequel on se trouve. A cette fin on calcule le nombre de Knudsen (Kn). Ce nombre sans dimension permet d'identifier le régime d'écoulement. On définit le nombre de Knudsen tel que :

$$Kn = \frac{\lambda_{air}}{L}, \tag{5.8}$$

où λ_{air} est le libre parcours moyen dans l'air (cf. Annexe (E.5), qui correspond à la distance moyenne que parcours une particule (molécule d'air dans notre cas) entre deux collisions successives, et L la longueur caractéristique de l'objet maillé que l'on étudie.
A partir de la valeur de Knudsen on distingue trois régimes d'écoulement différents, chacun avec sa propre équation pour le cœfficient de pression.
Ainsi pour un nombre de Knudsen inférieur à 0,001, on se trouve dans le régime continu et on calcule Cp par la méthode de Newton modifiée [5] :

$$Cp = Cp_{Max} * cos^2(\theta), \tag{5.9}$$

où θ est l'angle entre la normale extérieure au triangle considéré et la direction de l'écoulement et Cp_{Max} est le cœfficient de pression maximal calculé selon l'équation (E.8) en annexe.
Pour les régimes raréfiés et transitionnels, le Cp est alors calculé selon les équations présentées en Annexe (E.11) et (E.12).
Toutefois nous n'avons pas utilisé l'équation (5.7) dans l'outil. En effet, pour les triangles orientés parallèlement à l'écoulement, la méthode de génération des rayons avec OptiX, présente un comportement aléatoire amenant à la détermination, pour certains de ces triangles, d'un facteur de vue non nul. Ce phénomène est particulièrement visible sur certaine cellules de la face arrière du satellite de la Figure 5.5. Ce problème a

vraisemblablement pour origine un problème de précision numérique dans la méthode de calcul de propagation des rayons et de détection des intersections puisque l'on observe ce comportement quelque soit le support utilisé (CPU ou GPU) mais pour des rayons issus de triangles différents. A cause de ce caractère aléatoire, lorsque l'on considère la pression infini, sur deux faces opposées d'un même objet, de même dimension et parallèle à l'écoulement, la force de pression calculée sur ces deux faces sera différente car la contribution de la pression infinie sera calculée sur un nombre de triangle différents donc sur des surfaces effectives différentes, faussant le calcul.
Pour cette raison nous avons choisi d'ignorer la contribution de P_{inf} dans le calcul de la force, la contribution de la pression dynamique étant annulée grâce au Cp qui dépend du cosinus de l'angle et qui est nul dans le cas des faces parallèles à l'écoulement.
Cette hypothèse n'a aucun impact dans le cas d'un objet ayant une surface fermée puisque l'intégrale sur une surface fermée d'une constante est nulle. Dans le cas d'une surface ouverte, nos calculs sont valides pour un régime hypersonique, où $P_{inf} <<< P_{dynamique}$, et donc là aussi cette hypothèse n'a que peu d'impact sur le résultat final.
On calcule ainsi la force de pression sur le triangle via l'équation (5.10) :

$$F_P = P_{dynamique} * Cp * S_{effective}, \tag{5.10}$$

Une fois que l'on a calculé la force de pression exercée sur chaque triangle du maillage, on calcule la force de pression aérodynamique exercée sur l'ensemble de l'objet, $PObjetAero$, via l'équation 5.11 :

$$\begin{aligned} PObjetAero.x &= -\sum F_p * Normal_{Ext}.x, \\ PObjetAero.y &= -\sum F_p * Normal_{Ext}.y, \\ PObjetAero.z &= -\sum F_p * Normal_{Ext}.z \end{aligned} \tag{5.11}$$

La somme se faisant sur l'ensemble des triangles, $Normal_{Ext}$ étant la normale au triangle, orientée vers l'extérieur du maillage, et .x, .y et .z désignant les composantes selon les axes x,y et z des vecteurs.

5.4.8 Calcul cœfficient thermodynamique

5.4.8.1 Modèle

Le cœfficient thermique pour une forme quelconque est donné par l'équation (5.12) suivante :

$$C_H = \frac{q}{q_{Référence} * S_{maillage}}, \tag{5.12}$$

où q est le flux de chaleur convectif sur le maillage pour l'écoulement considéré, $S_{maillage}$ la surface du maillage et $q_{Référence}$ le flux de chaleur convectif de référence calculé sur une sphère de même surface et de rayon équivalent R_{eq} par la méthode de Detra-Kemp-Riddell (5.13) :

$$q_{Référence} = \frac{1e^4 * 11028.5}{\sqrt{R_{eq}}} \sqrt{\frac{\rho}{1.225}} (\frac{v}{7802.88})^{3.15}, \tag{5.13}$$

Le rayon équivalent est calculé via la méthode présentée Annexe E.4.3.1.
De la même façon que pour le calcul du cœfficient de traîné, l'outil aeroCœf permet de déterminer le cœfficient thermique pour différentes attitudes de l'objet et pour différentes conditions d'écoulement.

5.4.8.2 Mise en œuvre

Notons qu'un des points critiques dans (5.13) est la détermination du flux de chaleur q sur notre maillage. En effet si il est assez aisé de déterminer le flux de chaleur sur des formes géométriques simples grâce à de nombreuses études ayant fournies toutes une séries d'équations empiriques permettant de calculer les flux de chaleur sur ces objets à différents régimes [42],[23],[40],[20],... il n'existe pas de telles équations permettant de calculer directement le flux sur une forme quelconque.
Toutefois il est possible de reconstruire le flux de chaleur convectif sur un maillage, en définissant pour chaque cellule du maillage un rayon de courbure local qui permet alors d'utiliser les équations de la thermique pour les formes sphériques et ainsi de calculer le flux sur cette cellule.
La première étape du calcul du cœfficient thermique est donc la détermination des rayons de courbures locaux pour chaque triangle du maillage à partir de la démarche présentée en annexe E.4.3.1. Le calcul de ces rayons de courbure locaux est effectué en parallèle grâce à l'emploi de la bibliothèque OpenMP.
Une fois le calcul des rayons de courbures locaux effectués, le module calcule le flux de référence au point d'arrêt pour une sphère de rayon équivalent via 5.13. Ensuite, le flux de chaleur réel est calculé pour chaque cellule du maillage en fonction du régime d'écoulement (cf. Annexe E), puis ces flux par cellule sont sommés afin d'obtenir le flux

total sur la structure.
Le cœfficient thermique est ensuite calculé par l'équation (5.12).

5.4.9 Autres fonctionnalités

Le module aeroCœf a aussi été pensé afin de pouvoir fonctionner indépendamment du cœur de calcul de Calima. Ainsi aeroCœf permet de fournir des résultats complémentaires que ceux présentés ci-dessus et qui ne sont pas réutilisé dans le reste de Calima.
Il est ainsi possible d'utiliser le module pour simuler des calculs de soufflerie en imposant les conditions de température et de pression plutôt que l'altitude, mais aussi de calculer la section efficace de l'objet dans une direction donnée (qui correspond à la somme des surfaces des cellules du maillage multiplié par leur facteur de vue). Concernant cette dernière fonctionnalité, notons que le module aeroCœf permet d'obtenir de meilleurs résultats que le module CROC du logiciel DRAMA de l'Agence Spatiale Européenne. En effet par exemple, si on considère une sphère de 1m de diamètre, sa surface projeté dans une direction donnée est égale à la surface d'un disque de même diamètre, donc dans notre cas $\pi \approx 3.1415\ m^2$. Avec Calima on trouve une surface projetée de $3.1411\ m^2$, tandis qu'avec CROC on trouve une surface projetée de $3.16\ m^2$. Ensuite si l'on considère un cube de $1.0\ m$ de côté. La valeur théorique de la surface projeté d'une face est de $1\ m^2$, et c'est bien ce que l'on retrouve avec CROC et aeroCœf. En revanche si l'on considère la surface projeté du cube le long d'une arrête, la valeur théorique de cette surface est de $1.414214\ m^2$. Avec le module CROC on trouve $1.42\ m^2$ tandis qu'avec aeroCœf on retrouve la valeur théorique à savoir $1.414214\ m^2$.

5.5 Déploiement parallèle sur GPU

Suite aux résultats de l'étude de performance sur la problématique de la détermination de zone de retombée des enveloppes de ballons stratosphériques (cf. section 4.3), nous avons choisi d'implémenter un algorithme à parallélisme de tâche dans Calima. Comme expliqué à la sous-section 4.3.2.2, chaque thread considère ainsi une simulation qui lui est propre avec un jeu de conditions initiales uniques définies selon le mode de perturbation considéré (cf. section 5.6) puis effectue la simulation jusqu'à ce que l'objet atteigne le sol ou bien se désintègre. Chaque simulation étant totalement indépendante des autres, aucune communication n'est requise entre les threads. De cette façon, on limite bien le nombre de communications entre le CPU et le GPU puisqu'il n'y a que deux communications qui sont effectuées et on maximise le temps passé sur le GPU à faire du calcul.
Pour ce qui est de la gestion des données, les données des paramètres de configurations de Calima qui sont communs à tous les processus parallèles demeurent dans la mémoire unifiée afin de ne pas surcharger la mémoire locale de chaque thread, tandis que les

données de l'objet simulé sont copiées dans la mémoire locale de chaque thread. Ici cette copie locale des données est surtout pertinente en ce qui concerne la gestion des données de chaque simulation, puisque de cette façon chaque thread manipule un objet contenant toutes les variables de simulation dans sa mémoire locale.

5.6 Gestion des perturbations

La singularité majeure de Calima par rapport aux autres outils de simulation de rentrée atmosphérique est d'offrir la possibilité d'effectuer des analyses statistiques de type Taguchi, Monte-Carlo ou de sensibilité de façon native, le tout accéléré par l'utilisation des cartes graphiques. Dans cette section nous allons ainsi présenter comment ces différentes méthodes ont été mises en œuvre.

5.6.1 Étude de sensibilité

Calima permet de réaliser trois types d'analyses statistiques : les analyses de Monte-Carlo ou Taguchi, que nous avons déjà introduits en détail au Chapitre 2, mais aussi des études de sensibilité. Ces études se basent sur l'utilisation de tables d'expérience factorielle complètes (cf. sous-section 2.2.1.1), et consistent à générer aléatoirement ou à imposer un certain nombre de valeurs par paramètres à perturber, effectuer toutes les combinaisons possibles entre les valeurs de chaque paramètre, puis d'analyser les résultats obtenus afin de déterminer des corrélations entre les combinaisons de paramètres initiaux et la variation des résultats obtenus. Bien que cette méthode puisse sembler similaire à la méthode de Monte-Carlo elle est très éloignée. En effet, ce type d'analyse ne possède aucune des propriétés statistiques d'une analyse de type Monte-Carlo. Dans le cas où l'on considère un ensemble de valeurs imposées par l'utilisateur et ne suivant aucune loi de dispersion cela paraît évident, mais même dans le cas où l'on considère des valeurs aléatoires tirées selon une distribution gaussienne, le caractère systématique des combinaisons donne le même poids statistique à des combinaisons de valeurs issues des régions centre de la gaussienne, qu'à des combinaisons de valeurs issues des queues de distribution. De ce fait l'ensemble de simulations produits ne converge jamais malgré l'augmentation du nombre de simulation.
Cette méthode est alors pertinente car elle peut permettre d'établir la sensibilité d'un résultat aux variations de paramètres. Pour rappel, la méthode de Taguchi est un cas particulier d'analyse de sensibilité.

5.6.2 Modes de perturbations supportés et restrictions

Dans Calima nous avons mis en œuvre à ce jour 5 modes de perturbations regroupés en 3 catégories exclusives, et permettant de perturber jusqu'à 14 paramètres initiaux

différents, à savoir : la masse, l'altitude, la latitude, la longitude, l'azimuth, la vitesse, l'angle d'attaque, l'angle de lacet, l'angle de roulis, l'angle de la trajectoire de vol et les propriétés du matériaux : l'émissivité, la température de fusion, la chaleur de fusion et la capacité thermique.
Parmi ces trois catégories de perturbations on a donc :

- La méthode de Monte-Carlo : qui permet de perturber un paramètre soit en tirant aléatoirement des valeurs entre une borne min et une borne max selon une distribution uniforme, soit de tirer une valeur aléatoire selon une distribution gaussienne. Dans ce mode l'utilisateur impose le nombre de simulations qu'il souhaite effectuer et Calima se charge de créer les simulations.

- La méthode d'étude de sensibilité : qui permet de perturber un paramètre soit en utilisant un ensemble de valeurs imposées par l'utilisateur, soit en échantillonnant un domaine imposé par l'utilisateur. Dans ce mode l'utilisateur impose le nombre de valeurs qu'il souhaite avoir par paramètre et Calima se charge d'effectuer toutes les permutations possibles de valeurs et de calculer le nombre de simulations nécessaires.

- La méthode de Taguchi : l'utilisateur choisit les paramètres qu'il souhaite étudier ainsi que les trois niveaux correspondants. Calima se charge alors d'identifier la Table Orthogonale la plus appropriée à l'étude demandée, de distribuer les paramètres sur les colonnes de la table et de calculer les simulations définies.

De par leurs différences de fonctionnement ces catégories sont exclusives, autrement dit il n'est pas possible de perturber un paramètre avec la méthode gaussienne en même temps qu'on perturbe un autre paramètre en lui imposant un jeu de valeurs possibles. Calima identifiera ceci comme une erreur de l'utilisateur et s'interrompra en demandant à l'utilisateur de corriger cette incohérence. En revanche il est tout à fait possible de combiner les modes de perturbations d'une même catégorie : faire une analyse de Monte-Carlo en utilisant des perturbations gaussiennes pour un paramètre et uniformes pour l'autre ou une analyse de sensibilité en utilisant des valeurs imposées pour un paramètre et une variation à pas fixe pour un autre.

5.6.3 Mise en œuvre

Dans cette sous-section, nous allons considérer séparément la mise en œuvre de ces modes de perturbations, chacun ayant apporté une problématique au niveau de la mise en œuvre qui lui est propre. Dans la suite nous illustrerons les différentes mises en œuvre par des pseudo-codes. Pour simplifier la lecture de ces pseudo-codes nous ne considérerons que quatres paramètres pouvant être perturbés, respectivement $Param\_0$, $Param\_1$, $Param\_2$ et $Param\_3$.

5.6.3.1 Identifier un paramètre à perturber

Dans Calima, tous les paramètres ne peuvent être perturbés, en effet seuls ceux listés à la sous-section 5.6.2. peuvent l'être. En effet dans le contexte du calcul parallèle sur GPU il faut s'assurer que chaque thread dispose de toutes les informations nécessaires pour engendrer son jeu de condition initiale unique sans devoir communiquer avec les autres threads. Pour cela nous déclarons pour chaque variable pouvant être perturbée une variable qui contient l'information sur le mode de perturbation à utiliser, ainsi qu'un ensemble de variables qui vont contenir les données de perturbations selon le mode sélectionné : écart-type et valeur moyenne pour une perturbation gaussienne, ou valeur minimale et maximale ainsi que le nombre de valeurs à échantillonner pour une étude de sensibilité par exemple. Ainsi, ces variables de perturbation ne sont déclarés que pour les seuls paramètres listés à la sous-section 5.6.2.
Afin d'identifier quels sont les paramètres à perturber et de quelle manière les perturber, on tire profit des capacités du XML. Pour chaque paramètre à perturber on ajoute un ensemble d'attributs spécifiques qui sont lu par la procédure d'initialisation, et qui vont indiquer la méthode de perturbation ainsi que les propriétés statistiques à utiliser.

5.6.3.2 Mise en œuvre de la méthode de Monte-Carlo

Comme mis en évidence précédemment, la contrainte principale de la méthode de Monte-Carlo est d'assurer que chaque simulation utilise un ensemble de valeurs de paramètres initiaux uniques qui diffèrent d'une simulation à l'autre. De plus il faut aussi gérer le fait que le paramètre est perturbé selon une distribution uniforme ou une distribution gaussienne.
Sur la version séquentielle sur CPU l'unicité des valeurs engendrées pour chaque paramètre à chaque simulation est assurée de façon native : une seule graine est initialisée pour le générateur de nombre aléatoire en amont des calculs (cf. Pseudo-code 5.6, ligne 8), ainsi toutes les générations de nombre aléatoires s'effectuent sur la même séquence de génération assurant qu'entre deux simulations on ne génère pas la même séquence de valeur aléatoire. Le Pseudo-code 5.6, présente ainsi une version simplifiée de l'algorithme de perturbation des paramètres par la méthode de Monte-Carlo, appliquée ici à 4 paramètres. Durant le processus d'initialisation comme expliqué à la sous-section 5.6.3.1 on collecte la méthode de perturbation pour chaque paramètres que l'on stocke dans la variable perturbMode_j pour chaque paramètre. De la même façon les propriétés statistiques de perturbations sont stockées pour chaque paramètre dans la variable perturbCoef.
Sur la version parallèle sur GPU aussi bien pour la génération des graines et que pour la génération des perturbations nous avons utilisé les fonctions de la bibliothèque cuRAND développée par NVIDIA [48]. Le Pseudo-code 5.7, présente ainsi l'algorithme simplifié de

```
int main()
{
  //Initialisation
  N; //Nombre de simulations à calculer
  SimuObject; //Objet contenant tous les paramètres de la simulation
  ...

  //Initialisation du générateur de nombre aléatoire
  srand(clock());

  //Boucle sur les simulations
  for(int i=0;i<N; i++)
  {
    //Boucle sur les 4 paramètres à perturber
    for(int j=0; j<4; j++)
    {
      SimuObject.Param_j = PerturbMC(SimuObject.Param_j, SimuObject.perturbMode_j, SimuObject.
     perturbCoef_j);
    }

    //Simulation de la rentrée atmospherique avec les paramètres perturbés
    Simulation(SimuObject);
  }

  return 0;
}

//Fonction de génération des perturbations sur le CPU
double PerturbMC(Param_j, perturbMode, perturbCoef)
{
  if(perturbMode == Gaussian)
  {
    alpha = std::normal_distribution(0,1); //alpha: nombre aléatoire de moyenne 0 et        d'
     ecart type 1
    Param_j = Param_j + alpha * perturbCoef; //perturbCoef: écart type de l'incertitude sur
     Param_J
  }
  else if(perturbMode == Uniforme)
  {
    alpha = rand()/RAND_MAX; //alpha : nombre aléatoire entre 0 et 1 de distribution uniforme
    Param_j = perturbCoef[0] + alpha * (perturbCoef[1]-perturbCoef[0]); //perturbCoef[0] =
     valeur min et perturbCoef[1] = valeur max possible pour Param_j
  }

  return Param_j;
}
```

PSEUDO-CODE 5.6: Pseudo-code de la mise en œuvre de la perturbation pour la méthode de Monte-Carlo sur CPU

l'implémentation de la perturbation par la méthode de Monte-Carlo sur GPU. Comme on peut le voir la gestion du mode de perturbation et des coefficients statistiques de perturbation est identique à ce qui est fait sur CPU.

5.6.3.3 Implémenter l'analyse de sensibilité

La mise en œuvre des perturbations pour l'analyse de sensibilité s'est révélée plus complexe à réaliser que celle de la méthode de Monte-Carlo, puisqu'elle ne permet pas d'avoir une génération directe des perturbations sur le GPU. En effet pour ce type d'analyse il faut garantir que toutes les simulations du plan d'expérience soient réalisées, ce qui sous entend une certaine connaissance pour un thread donné des simulations effectuées par les autres threads parallèles. Or la mise en œuvre d'un tel réseau de communication entre les threads du GPU est compliquée à mettre en œuvre, requiert des synchronisations périodiques et est donc pénalisante vis à vis des performances de l'outil. C'est pour cette raison que nous avons pris le parti de contourner le problème en le traitant différemment : Lors de la phase d'initialisation des données sur le CPU, Calima produit pour chaque paramètre le vecteur des valeurs accessibles par ce dernier à partir des informations fournies par l'utilisateur. Ce vecteur de valeur accessible pour un paramètre est ensuite

```
int main()
{
  //Initialisation
  N; //Nombre de simulations à calculer
  SimuObject; //Objet contenant tous les paramètres de la simulation
  ...

  //Creation sur le CPU du vecteur de graines Seeds pour le générateur de nombre aléatoire
     curand via le kernel graineGenerateur
  seed = clock();
  graineGenerateur<<<N,1>>>(seed,Seeds);

  //Lancement du kernel de calcul
  KernelCalcul(SimuObject);

  //Récuperation résultats;
  return 0;
}

__global__ void KernelCalcul(objet)
{
  threadId; //identification de la thread courante
  if(threadId <N)
  {
    //Boucle sur les 4 paramètres à perturber
    for(j=0;j<4;j++)
    {
      objet.Param_j = PerturbeMC(objet.Param_j, Seeds[threadId],objet.pertubMode_j,objet.
     perturbCoef_j);
    }
    //Simulation de la rentre atmospherique avec les paramètres perturbés
    Simulation(objet);
  }
}

//Fonction de génération des perturbations sur le GPU
__device__ double PerturbeMC(Param_j,Seed,pertubMode,perturbCoef)
{
  if(perturbMode == Gaussien)
  {
    alpha = curand_normal(Seed); //alpha : nombre aléatoire de moyenne 0.0 et d'écart type 1.0
    Param_j = Param_j * alpha *perturbCoef; //perturbCoef = écart type de l'incertitude sur
     Param_j
  }
  else if(perturbMode == Uniforme)
  {
    alpha = currand_uniform(Seed);
    Param_j = perturbCoef[0] + alpha * (perturbCoef[1] - perturbCoef[0]); //perturbCoef[0] =
     valeur min et perturbCoef[1] = valeur max possibles pour Param_j
  }

  return Param_j;
}
```

PSEUDO-CODE 5.7: Pseudo-code de la mise en œuvre de la perturbation par méthode de Monte-Carlo sur GPU

```
Initialisation(); //Identifie Param_1 avec 3 valeurs possibles et Param_2 avec 2 valeurs

//Création des vecteurs de valeurs possibles pour les paramètres avec si besoin génération des
     valeurs
Param_1_vecteur = [2,5,8];
Param_2_vecteur = [3,4];

//Génération de la table de plan d'expérience
Table = Generation(len(Param_1_vecteur),len(Param_2_vecteur));

//Table = [[0,0];[0,1];[1,0];[1,1];[2,0];[2,1]]

//Déploiement parallele sur le GPU
threadId = 3;
Simulation(Param_0,Param_1[1],Param_2[0],Param_3);
```

PSEUDO-CODE 5.8: Principe de la mise en œuvre de la perturbation pour l'analyse de sensibilité sur GPU

stocké dans l'objet "SimulateObject". Ensuite, en fonction du nombre de paramètres à perturber et du nombre de valeurs accessibles pour chaque paramètre, nous construisons sur le CPU, une table où chaque colonne correspond à un paramètre à perturber et où chaque ligne correspond à une combinaison de valeurs de ces paramètres. En d'autres termes, Calima construit la table correspondant au plan d'expérience défini par l'utilisateur, chaque ligne de ce tableau correspondant à une combinaison de paramètres initiaux pour une simulation. En fonction de son identifiant, la simulation charge la ligne correspondante dans la table de plan d'expérience et utilise les index contenus dans ce vecteur pour initialiser ses paramètres perturbés. Cette implémentation est présentée de façon schématique dans le Pseudo-code 5.8.

5.6.3.4 Mise en œuvre l'analyse de Taguchi

Nous allons nous attarder ici sur l'implémentation de la première moitié de la méthode de Taguchi, à savoir l'identification de la Table Orthogonale à utiliser et la génération des simulations. Nous traiterons la seconde partie de la méthode sur le traitement des résultats dans la section 5.9 consacrée au post traitement des résultats.
La mise en œuvre de la première phase de l'analyse de Taguchi est très similaire à celle de l'analyse de sensibilité, à la différence qu'ici au lieu d'engendrer la table de plan d'expérience, il faut charger la Table Orthogonale correspondant au nombre de paramètres à perturber.
Étant donné que dans Calima l'utilisation de la méthode de Taguchi a pour vocation la préparation d'analyse de Monte-Carlo, nous avons pris le parti de n'implémenter que la possibilité d'utiliser des tables orthogonales à 3 niveaux (cf. sous-section 2.2.1). De ce fait, si l'utilisateur cherche à perturber moins de 5 paramètres ce sera la table L9 qui sera chargée, jusqu'à 13 paramètres ce sera la table L27 et au delà ce sera la table L81. L'identification de la ligne de la table orthogonale à utiliser pour engendrer le jeux de condition initiale se fait à partir de l'identifiant de la simulation.

5.7 Module Aérodynamique

Le module aérodynamique tel que présenté à la Figure 5.2 se compose de deux parties, la première sert à calculer l'ensemble des forces aérodynamiques qui s'exercent sur la structure de l'objet à l'instant courant tandis que la seconde intègre l'équation du mouvement.

5.7.1 Calcul de la trajectoire

Le mouvement orbital ou de rentrée atmosphérique d'un satellite ou d'un débris peut être décrit mathématiquement en première approximation via le principe fondamental

de la dynamique (5.14), plus communément appelé deuxième loi de Newton, qui définit la variation de quantité de mouvement linéaire d'un objet comme étant égale à la somme des forces externes qui s'exercent sur sa structure .

$$\frac{d\vec{p}}{dt} = \sum_i \vec{F_i}, \tag{5.14}$$

avec $\vec{p}$ la quantité de mouvement qui est égal à la masse de l'objet multipliée par la vitesse.

Dans le cas général, la solution de cette équation est un vecteur à 3 composantes qui représente la mouvement, dans un référentiel donné, de l'objet en l'assimilant à une masse ponctuelle. Toutefois cette modélisation n'est pas suffisante pour représenter le mouvement d'un objet réel qui du fait des différentes forces aérodynamiques s'exerçant sur sa surface va tourner autour de son centre de gravité. Il faut alors en plus considérer le système d'équation décrivant la variation de moment angulaires de l'objet.

Ainsi dans Calima la trajectoire est calculée à partir du système à 6 équations suivant (5.15) issu de [87] :

$$\left.\begin{aligned}
\frac{dr}{dt} &= V\sin\gamma,\\
\frac{dl}{dt} &= \frac{V\cos\gamma\cos\chi}{r\cos L},\\
\frac{dL}{dt} &= \frac{V\cos\gamma\sin\chi}{r},\\
\frac{dV}{dt} &= -\frac{1}{m}T + \omega^2 r\cos^2 L(\sin\gamma - \cos\gamma\tan L\sin\chi) - g\sin\gamma,\\
V\frac{d\gamma}{dt} &= \frac{1}{m}P\cos\delta - \frac{1}{m}\frac{P_l\cos\delta}{\cos\gamma} - \frac{V^2}{r}\cos\gamma + 2\omega V\cos L\cos\chi,\\
&\quad + \omega^2 r^2\cos^2 L(\cos\gamma + \sin\gamma\tan L\sin\chi) - g\cos\gamma,\\
V\frac{d\chi}{dt} &= \frac{1}{m}\frac{P\sin\delta}{\cos\gamma} + \frac{1}{m}\frac{P_l\cos\delta}{\cos\gamma} - \frac{V^2}{r}\cos\gamma\cos\chi\tan L,\\
&\quad + 2\omega V(\tan\gamma\cos L\sin\chi - \sin L) - \frac{\omega^2 r}{\cos\gamma}\sin L\cos L\cos\chi
\end{aligned}\right\} \tag{5.15}$$

Avec :

- r : l'altitude de l'objet par rapport au sol.
- l : la longitude de l'objet
- L : la latitude de l'objet
- V : la vitesse de l'objet dans le référentiel de la trajectoire de vol

- γ : l'angle de la trajectoire de vol
- χ : l'azimuth
- T : la force de trainée
- P : la force de portance
- P_l : la force de portance latérale
- g : la gravitée
- ω : la vitesse de rotation du référentiel Terrestre par rapport au référentiel Géocentré

5.7.2 Mise en œuvre

En terme de mise en œuvre, le calcul de (5.15) ne pose aucun défis majeur. Ce calcul est effectué par un ensemble de fonctions qui sont appellées depuis le kernel. Les forces de trainée, portance et portance latérale sont calculées à partir des coefficients C_D, C_Y et C_L calculés par le module aéroCoef (cf. section 5.4). Enfin le système (5.15) est intégré par un Runge-Kutta d'ordre 4 (cf. [66]).

5.8 Module Thermodynamique

Ce module sert à calculer le flux de chaleur s'exerçant sur la structure et à mettre à jour la température de la paroi de l'objet et sa masse pour le cas où la température de la paroi de l'objet atteindrait la température de fusion.

5.8.1 Calcul du flux de chaleur

Dans Calima on considère un flux de chaleur $\dot{Q}_{net}$ s'exerçant de façon homogène sur la surface de l'objet tel que :

$$\dot{Q}_{net} = (\dot{q}_{conv} + \dot{q}_{rad,Gain} + \dot{q}_{rad,Loss}) * S_{paroi}, \tag{5.16}$$

où $\dot{q}_{rad,Gain}$ désigne le flux radiatif absorbé par la structure et correspond au flux rayonné par l'environnement loin de l'objet. Il est donné par l'équation de Stefan-Boltzmann 5.17 :

$$\dot{q}_{rad,Gain} = \alpha_{paroi}\sigma T^4_{atmosphère}, \tag{5.17}$$

avec α_{paroi} le coefficient d'absorption de la paroi qui dépend du matériau, σ la constante de Stephan-Boltzmann et $T_{atmosphère}$ la température de l'atmosphère loin du choc engendré par la rentrée atmosphérique de l'objet. $\dot{q}_{rad,Loss}$ désigne le flux radiatif émis par

la surface de la structure en fonction de la température de la paroi T_{paroi} . Il est aussi défini par l'équation de Stefan-Boltzmann, toujours négative :

$$\dot{q}_{rad,Loss} = -\epsilon_{paroi}\sigma T_{paroi}^4, \tag{5.18}$$

avec ϵ_{paroi} l'émissivité du matériau constituant la paroi. Enfin $\dot{q}_{conv}$ désigne le transfert d'énergie induit par la convection forcée de l'écoulement sur la paroi, tel que :

$$\dot{q}_{conv} = C_H * \dot{q}_{ref}, \tag{5.19}$$

Avec $\dot{q}_{ref}$ le flux convectif de référence calculé au point d'arrêt sur une sphère de rayon équivalent par l'équation de Detra-Kemp-Riddel (5.13), et C_H le coefficient thermique qui est soit imposé par l'utilisateur soit issue du calcul du module aeroCoef. Dans le second cas, il peut alors être interpolé de la même façon que les coefficients aérodynamiques selon les conditions de calculs du module.

5.8.2 Mise à jour des propriétés de l'objet : température et masse

Le flux absorbé par la structure de l'objet entraine ensuite une variation de la température de celui-ci. Dans Calima, on considère une modélisation 0D de la température de la paroi, ce qui veut dire que l'on considère que la température est identique sur la paroi de l'objet et que son évolution est directement proportionnelle au flux de chaleur. On définit donc l'évolution de la température de la paroi pendant un temps Δt via l'équation (5.20) suivante :

$$\Delta Température = \frac{\dot{Q}}{mCp}\Delta t, \tag{5.20}$$

avec m la masse de la paroi et Cp la capacité thermique massique de la paroi. Lorsque la température de fusion est atteinte, l'énergie transférée à l'objet n'entraîne plus une augmentation de la température de la paroi. A la place cela entraîne une fusion du matériau qui passe alors à l'état liquide avant d'être extrait par les forces de pression et de frottement. On calcule alors la masse ablatée selon l'équation (5.21) suivante :

$$\Delta masse = \frac{-\dot{Q}}{h_{fus}}, \tag{5.21}$$

avec h_{fus} la chaleur de fusion du matériau.
Nous rappelons que dans Calima, la perte de masse n'est pas associée à une modification des dimensions de l'objet.

5.8.3 Mise en œuvre

Comme pour le module aérodynamique, la mise en œuvre de la résolution du module thermodynamique ne représente aucune réelle difficulté en terme de programmation.

5.9 Post-traitement

Pour des raisons de commodité, le postraitement des résultats n'est pas effectué directement dans l'outil. Il est réalisé par un script annexe en Python qui lit le fichier de résultat généré par Calima et fourni une analyse statistique détaillée :

- moments statistiques et détermination des régions à 1, 2 et 3 sigmas pour les analyses de Monte-Carlo ou de sensibilité
- table d'ANOVA dans le cas d'une analyse de Taguchi

5.10 Détermination des incertitudes

Comme évoqué à de nombreuses reprises dans ce mémoire, les résultats d'une analyse statistique quelle qu'elle soit (Taguchi, Monte-Carlo,...) sont extrêmement dépendants des valeurs d'incertitudes utilisées en entrée de l'analyse. Ainsi l'utilisation d'un outil tel que Calima ne peut avoir de sens que si l'on a une bonne connaissance des incertitudes que l'on a aussi bien sur les conditions initiales de simulation que sur les modèles employés. Nous allons détailler une initiative de la société R.Tech, qui a germée dans le cadre de ces travaux de thèse afin d'organiser un workshop Européen sur la question de l'étude des débris de satellite le Spacecraft Demise Workshop.

5.10.1 Organisation de la première édition du Spacecraft Demise Workshop (SCDW)

La question de la détermination des incertitudes sur les différents paramètres de la rentrée atmosphérique d'un satellite et sur les modèles employés est des plus épineuses, principalement à cause du manque de données expérimentales. Il n'est donc pas possible de comparer les données issues de la modélisation pour déterminer des valeurs d'incertitudes utilisables lors d'analyse statistiques; l'aspect multi-physique du processus de rentrée atmosphérique d'un satellite rendant la caractérisation des incertitudes d'autant plus ardues.
La plupart des codes de rentrée actuels sont basés sur des méthodes de modélisation simplifiées, souvent héritées des recherches menées dans les années 60 sur la rentrée atmosphérique des missiles nucléaires. Le problème est qu'à cette époque les marges appliquées sur les modéles ont été prises de façon à garantir la conception d'un missile

qui survive à la phase de rentrée, or dans le cas de la rentrée de débris l'objectif est inverse.

Les méthodes de calculs actuelles telles que la mécanique des fluides numériques (computational fluid dynamics ou CFD en anglais), ainsi que les codes d'analyses thermiques permettent déjà de confronter les résultats de modélisation très fine avec ceux des outils de rentrée actuels afin d'obtenir une première évaluation des incertitudes à utiliser pour compenser les approximations de ces modélisations.

C'est pourquoi avec la société R.Tech, en partenariat avec l'agence spatiale française le CNES, et l'agence spatiale européenne l'ESA, nous avons organisé dans le cadre de ces travaux de thèse la première édition du Spacecraft Demise Workshop (SCDW) qui s'est tenu en deux temps : un premier évènement qui a eut lieu le 14 Octobre 2014 au CNES à Toulouse, et un second qui a eut lieu le 6 Mars 2015 à Lisbone dans le cadre du 8$^{\text{ème}}$ Symposium Européen sur l'Aérothermodynamique pour les Véhicules Spaciaux organisé par l'ESA. La vocation première de ce workshop est d'augmenter la quantité de donnée disponible pour la validation des logiciels actuels et futurs d'étude de rentrée atmosphérique. L'autre objectif est de réunir un maximum d'acteurs industriels, académiques, membres d'agence spatiale autour d'un ensemble de cas test afin de réaliser, pour la première fois, une comparaison à grande échelle de tous les différents logiciels de rentrée atmosphérique entre eux, et de déterminer les incertitudes entre les codes dans les différentes disciplines de la rentrée atmosphérique pour ensuite pouvoir utiliser ces incertitudes dans des outils d'analyses statistiques tels que Calima.

5.10.2 Présentation des cas tests

Du fait de la diversité des disciplines impliquées dans la simulation de la rentrée de débris, nous avons pris le parti de diviser les cas tests du workshop en 3 catégories, notre but pour cette première édition étant de chercher à valider indépendamment chaque discipline impliquée dans la rentrée atmosphérique de satellite. Ces catégories sont les suivantes :

- Cas de validation aérothermodynamique : l'objectif de ces cas est la validation des modèles aérothermodynamiques employés dans les outils de rentrée. Cela inclut la validation des formules aérodynamiques utilisées dans le régime raréfié et continue en les comparant avec des résultats de modélisation CFD ou DSMC (Direct Simulation Monte-Carlo). Ces cas ont été définis pour des cylindres de dimensions différentes ainsi que des cubes pour différents régimes aérothermodynamiques.

- Cas thermiques : l'objectif de ces cas est de comparer les formules thermiques ainsi que les méthodes de résolution employées dans les outils de rentrée de débris avec des modèles plus évolués. Ces cas concernent deux types de sphères (qui sont creuses ou pleines) en considérant une modélisation thermique 0 ou 1D.

- Cas d'intégration : ces cas se concentrent sur la comparaison des trajectoires complètes des débris de satellites calculées par les différents outils afin d'évaluer comment les différents modèles mis en œuvre dans chaque outils impactent les paramètres globaux de simulation tels que l'altitude de désintégration, l'énergie d'impact, le point d'impact,... Ces cas traitent différentes formes géométriques de différents matériaux mais toujours avec les mêmes conditions initiales (position, vitesse, ...).

L'objectif durant la conception de ces cas tests était de définir un ensemble de cas tests aussi simples que possible afin qu'une majorité de personnes puissent les étudier. La définition de ces cas tests est détaillée dans le booklet du workshop [65] et dans [79].

5.10.3 Premiers résultats

L'analyse de la plus grande partie des résultats de ce workshop sortant du cadre de ces travaux de thèse, nous nous bornons à brosser un rapide portrait des calculs et des résultats obtenus dans chaque section. Nous renvoyons à la lecture de Spel et al. [79] pour une présentation plus détaillé des résultats. Nous présentons en fin de ce chapitre une comparaison des résultats obtenus avec Calima et avec les autres outils.

5.10.3.1 Aérothermodynamique

Un total de 11 contributions différentes ont été recensées pour les cas aérothermodynamiques : 6 en régime continu et 5 en régime transitionnel. La Table 5.1 présente un récapitulatif des contributeurs, des logiciels employés ainsi que du type de logiciel dont il s'agit. Comme on peut le voir à la fois des codes à haute fidélité (CFD et DMSC) ainsi que des codes simplifiés ont été employés ce qui va nous permettre de comparer les résultats de ces logiciels à haute fidélité entre eux afin d'estimer la dispersion des résultats, mais aussi de comparer ces résultats avec ceux des méthodes simplifiées afin d'évaluer la précision de ces méthodes simplifiées et leurs éventuelles restrictions.
Les codes de CFD qui ont été employés pour traiter ces cas sont tous des codes bien connus de la communauté CFD, et la plupart d'entre eux ont été développés pour des applications de rentrée non-destructive.
Bien que les cas qui ont été définis aient une géométrie simple, ils donnent lieux à des écoulement très complexes. Tout d'abord les angles saillants de ces objets engendrent des types d'ondes de chocs, qui ne sont pas souvent rencontrés dans les cas de rentrée non-destructive de par la géométrie des objets considérés qui sont généralement émoussés (capsules, etc...). Dans un second temps lorsqu'on considère le cylindre creux, on constate un écoulement très complexe à l'intérieur du cylindre ainsi que dans le sillage de ce dernier (cf. [79]).

TABLE 5.1: Liste des contributeurs aux cas aérothermodynamique

Code	Contributeurs	Type de Code
ANITA	Fluid Gravity	CFD
DsmcFoam	R.Tech	DSMC
DsmcFoamStrath	Strathclyde University	DSMC
Mistral-DSMC	R.Tech	DSMC
Mistral-CFD	R.Tech	CFD
OpenFOAM (rhoCentralFoam)	Strathclyde University	CFD
Pampero	R.Tech/CNES	Simplifié
SAM	Fluid Gravity	Simplifié
SMILE	ITAM	DSMC
SU2	R.Tech	CFD
TAU	DLR	CFD

TABLE 5.2: Liste des contributeurs aux cas d'intégration

Code	Contributeurs	Type de Code
Debrisk	R.Tech	Orienté Objet
DEBRIS	Deimos	Orienté Objet
SAM	Belstead Research	Orienté Véhicule spatial
SESAM	HTG	Orienté Objet

5.10.3.2 Thermique

Faute de participant durant la première édition du workshop nous ne disposons pas de résultats à présenter.

5.10.3.3 Intégration

Quatre contributeurs ont participé aux cas tests d'intégration. La Table 5.2 présente un récapitulatif des contributeurs ainsi que des codes employés et du type de code dont il s'agit. Trois des codes utilisés sont de type orienté objet, tandis qu'un seul est orienté véhicule spatial ce qui nous permet de pouvoir comparer entre eux les résultats de différents logiciels orientés objets et aussi de les confronter avec ceux d'un logiciel orienté véhicule spatial.
Il est à noter que dans le cadre de ce workshop, les calculs SAM ont été effectués avec une grande diversité de modèles thermodynamiques, mais que ceuls les résultats obtenus avec deux modèles différents ont été présentés dans le cadre de ce workshop.

5.10.4 Bilan et perspectives du workshop

À l'issue de cette première édition, nous avons ainsi pu établir une première base de donnée pour la vérification des codes de rentrée atmosphérique. Toutes les données collectées durant ce workshop ont été mises en libre accès à la communauté sur le site dédié du workshop : http://scdw.rtech.fr

Lorsqu'on considère les cas aérothermodynamiques on note un très bon accord entre les codes à haute fidélité (CFD et DSMC). Les méthodes simplifiées quand à elles, ont besoin d'être encore améliorée en ce qui concerne la modélisation de la distribution des flux de chaleur et des forces de pression. Il est évident que les modèles simplifiés ne seront jamais à même de capter les comportements complexes tels que les ondes de choc, les séparations de l'écoulement, ou d'autres, mais les résultats obtenus mettent en évidence le fait que ces méthodes sont beaucoup trop simplistes et divergent de façon significative des résultats des méthodes à haute fidélité. Du côté des cas d'intégration on constate que l'on obtient un très bon accord entre les résultats des différents logiciels pour tous les cas tests qui ne s'ablatent pas (de 1 à 8), montrant un bon accord sur la partie aérodynamique et la trajectographie de ces codes . Pour les cas qui s'ablatent complètement on constate des différences plus marquées en raison des différents processus thermiques considérés entre les codes, entraînant des ablations plus ou moins rapides des objets.
Cette première édition a été un succés, malgré l'absence de participants pour la section thermique du workshop, et il reste encore beaucoup de travail à mener pour finir de dépouiller toutes les données qui ont été générées à l'occasion de ce workshop, afin d'identifier plus clairement les sources de divergences entre les outils. Pour une édition prochaine, il serait intéressant de compléter ces données en considérant d'autres points d'intérêts tel que la détermination des incertitudes sur les propriétés thermiques des matériaux dans les conditions de la rentrée atmosphérique qui sont très mal connus.

5.11 Résultats expérimentaux

Nous allons à présent comparer les résultats produits par Calima et ceux des différents contributeurs du Spacecraft Demise Workshop (SCDW).

5.11.1 Aérothermodynamique

5.11.1.1 Présentation des résultats

Nous allons comparer les résultats des cas mandataires M001 à M006 ainsi que ceux des cas optionnels X007 à X015 de la section aérodynamique du SCDW. Nous ne considérons pas les cas X001 à X006 faute de données pour la comparaison. Nous rappellons les conditions expérimentales de ces cas dans les Tables 5.3 et 5.4. Pour mener cette étude nous avons utilisé les maillages grossiers pour le cube et pour les différents cylindres qui ont été fournis à l'ensemble des participants du SCDW. Dans la suite de cette sous-section nous privilégierons toujours les comparaisons entre les résultats de Calima et ceux des logiciels de calcul CFD ou DSMC. Plus particulièrement nous utiliserons les résultats de Mistral-CFD pour l'analyse détaillée des résultats des cas M001 à M006 et ceux de Mistral-DSMC pour celle des résultats des cas X007 à X015, ces derniers

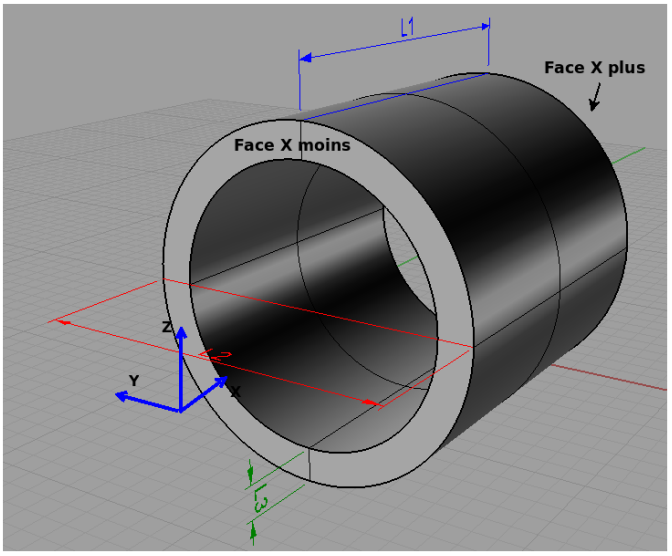

FIGURE 5.9: Définition du cylindre étudié

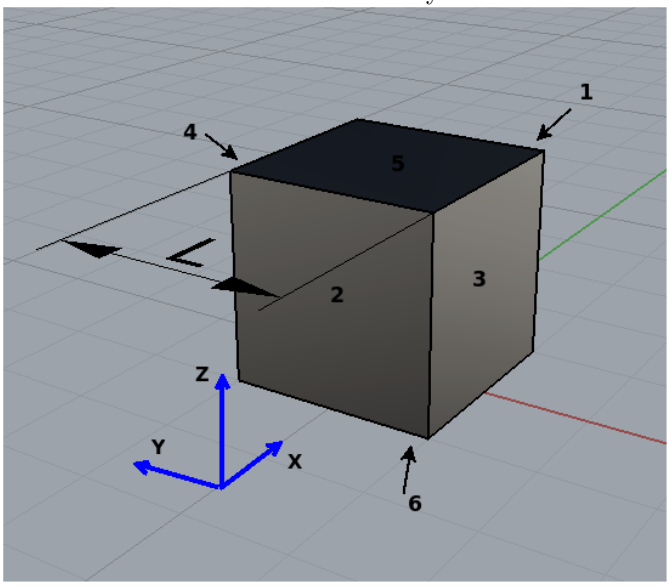

FIGURE 5.10: Définition du cube étudié

étant les plus complets. Dans un premier temps comparons les valeurs des coefficients aérodynamiques C_D et C_L. Faute de données fournies par les participants nous ne pouvons comparer les résultats de Calima qu'avec ceux de Pampero pour les cas M001 à M006. Lorsqu'on considère les valeurs de ces coefficients, présentés Table 5.5, on note un trés bon accord entre les valeurs calculées par les deux outils. Pour les cas X007 à X015 nous comparons ensuite les valeurs des coefficients C_D et C_L calculés par Calima et Mistral-DSMC (cf. Table 5.6). On constate alors une différence trés importante entre les résultats de Calima et ceux de Mistral-DSMC. Néanmoins cela s'explique facilement par le fait que dans ces conditions on sort totalement du domaine de validité de la méthode de détermination des coefficients tel que définis à la sous-section 5.4.7. Notamment la méthode actuelle de détermination ignore l'influence des forces de

TABLE 5.3: Définition des cas aérothermodynamiques

Cas N°	Forme	L1 [m]	L2 [m]	L3 [m]	Angle d'attaque [°]	Dérapage [°]	Conditions
M001	Cylindre	1.0	1.0	0.25	0	0	1
M002	Cylindre	1.0	1.0	0.25	45	0	1
M003	Cylindre	1.0	1.0	0.25	90	0	1
M004	Boite	1.0	1.0	1.0	0	0	1
M005	Boite	1.0	1.0	1.0	45	0	1
M006	Boite	1.0	1.0	1.0	45	45	1
X007	Cylindre	1.0	1.0	0.1	0	0	2
X008	Cylindre	1.0	1.0	0.5	0	0	2
X009	Cylindre	1.0	1.0	0.25	0	0	2
X010	Cylindre	1.0	1.0	0.1	45	0	2
X011	Cylindre	1.0	1.0	0.5	45	0	2
X012	Cylindre	1.0	1.0	0.25	45	0	2
X013	Cylindre	1.0	1.0	0.1	90	0	2
X014	Cylindre	1.0	1.0	0.5	90	0	2
X015	Cylindre	1.0	1.0	0.25	90	0	2

TABLE 5.4: Conditions initiales des cas aérothermodynamiques

Condition	Mach	Température [K]	Pression [Pa]	Densité [kg/m^3]	Température de paroi [K]
1	9	256.26	272.72	3.71e-3	700
2	24.64	222	7.49787e^-3	1.18e^{-07}	200

TABLE 5.5: Coefficients aerodynamiques pour les cas M001 à M006

—	M001		M002		M003		M004		M005		M006	
	Pampero	Calima	Pampero	Calima	Pampero	Calima	Pampero	Calima	Pampero	Calima	Pampero	Calima
C_D	1.078	1.078	0.907	0.907	1.22	1.22	1.83	1.83	1.294	1.294	1.104	1.104
C_L	0.0	0.0	0.145	0.145	0.0	0.0	0.0	0.0	0.0	0.0	0.189	0.155

friction ainsi que la pression atmosphérique dans le calcul de la force de pression, celle-ci n'étant plus négligeable devant la pression dynamique dans ce domaine. Nous allons à présent comparer les valeur de flux de chaleur intégré sur la surface Q exprimé en Watt, calculé par les différents outils. Comme on peut le constater sur la Figure 5.11, conformément aux conclusions du workshop, Calima qui utilise des méthodes simplifiées pour le calcul du flux, sous estime systématiquement la valeur du flux de chaleur intégré pour tout ces cas. Si l'on considère à présent l'écart en pourcentage entre les résultats de Calima et ceux des autres logiciels (cf. Table 5.8) on constate qu'en moyenne l'écart entre les résultats des codes de CFD et ceux de Calima est environ de 30%. Cette différence se comprend trés rapidement lorsqu'on considère les contributions par face de l'objet. Les Tables 5.9 et 5.10 présentent ainsi les flux de chaleur pour chaque face, les faces étant identifiées comme indiqué sur les Figures 5.9 et 5.10, pour les données de Mistral-CFD et de Calima. Ainsi on constate, lorsque l'on considère ces tables, que

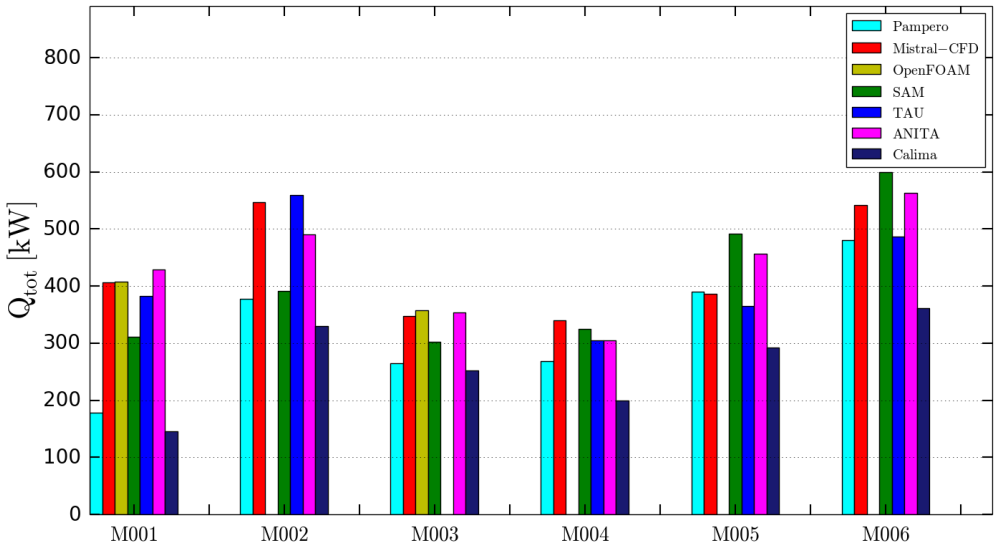

FIGURE 5.11: Flux de chaleur intégré sur la surface en kW pour les cas mandataires

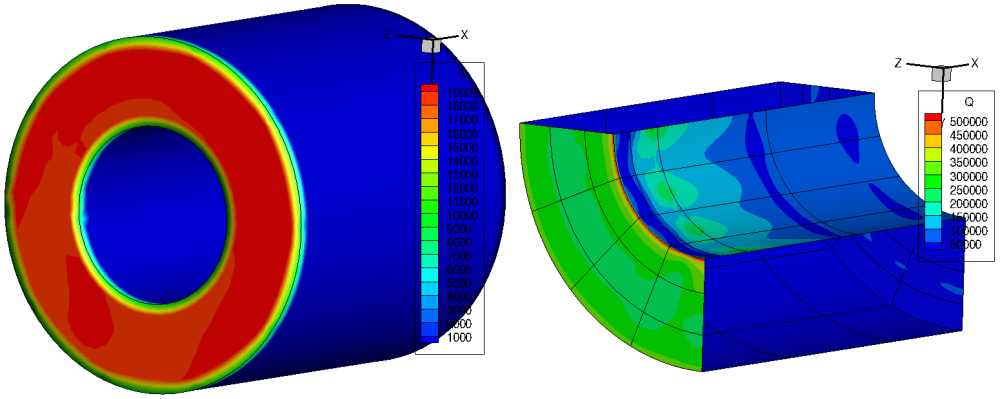

FIGURE 5.12: Flux de chaleur en W/m^2 pour le cas M001 à gauche calculé par Calima, à droite par Mistral-CFD

pour les faces directement exposées à l'écoulement on trouve un assez bon accord entre les valeurs de Mistral-CFD et celles de Calima avec un écart moyen d'une dizaine de pourcent. En revanche on note que sur les faces parallèles ou bien qui sont masquées par rapport à l'écoulement, le flux calculé par Calima est nul, tandis que celui calculé par la CFD ne l'est pas. La Figure 5.12 présente le même résultat mais sur l'objet maillé. On note, pour le résultat Mistral-CFD, que localement à l'intérieur du cylindre, le flux est aussi élevé que sur la face directement exposée au flux tandis que pour Calima le flux est nul partout ailleurs que sur la face avant.

Si l'on considère à présent les résultats obtenus sur les cas optionels (en millieu raréfié) on observe que le flux de chaleur intégré sur la surface calculé par Calima est en moyenne 20% inférieur aux flux de chaleurs calculés par les outils de simulations DSMC (cf. Figure 5.13). Comme pour les cas mandataires, lorsqu'on considère la contribution par face (cf. Table 5.7) on note un assez bon accord entre les résultats de Mistral-DSMC et ceux de Calima pour les faces qui sont directement exposées à l'écoulement, alors que pour les

TABLE 5.6: Coefficients aerodynamiques pour les cas X007 à X015

—	X007		X008		X009		X010		X011		X012		X013		X014		X015	
	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima
C_D	1.128	0.4	1.77	1.1	1.64	0.83	2.21	2.49	2.23	2.48	2.23	2.49	1.96	1.81	1.96	1.81	1.96	1.81
C_L	5.1e-5	0.0039	4.4e-5	0.006	-0.1e-5	0.001	0.24	-1.63	-0.045	-0.25	0.05	-0.79	-6.2e-5	0.22	-2.7e-5	0.68	-10.9e-5	0.5

TABLE 5.7: Flux de chaleur intégré Q [W] par face pour les cas optionels

—	X007		X008		X009		X010		X011		X012		X013		X014		X015	
	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima	Mistral-DSMC	Calima
Face X plus	55.86	0.0	15.3	0.0	31.31	0.0	142.59	0.0	195.68	0.	221.94	0.0	442.39	0.0	958.23	0.0	849.39	0.0
Face X moins	5074.07	5394.28	13698.53	12204.43	10931.34	11436.44	3928.37	3886.02	9972.13	8788.98	7802.29	7404.97	596.18	0.0	799.82	0.0	675.51	0.0
Face latérales	8304.21	0.0	2956.51	0.0	6870.54	0.0	28312.7	15463.52	13717.53	9844.32	16691.36	12090.66	19851.84	17665.62	18973.13	17625.59	19371.51	17668.39

TABLE 5.8: Écarts de flux de chaleur intégré en pourcentage avec Calima

	Pampero	Mistral-CFD	OpenFOAM	SAM	TAU	ANITA
M001	18.58	64.29	64.35	53.26	62.05	66.10
M002	12.43	39.63		15.54	41.03	32.61
M003	4.69	27.45	29.38	16.68		28.58
M004	25.80	41.35		38.64	34.62	34.58
M005	25.01	24.21		40.55	19.86	35.85
M006	24.86	33.32		39.66	25.79	35.81

TABLE 5.9: Flux de chaleur intégré Q [W] par face cas mandataires 1 à 3

—	M001		M002		M003	
	Mistral-CFD	Calima	Mistral-CFD	Calima	Mistral-CFD	Calima
Face X plus	12000.3.07	0.0	15460.29	0.0	21891.08	0.0
Face X moins	159032.96	145353.82	139541.48	115127.36	21891.08	0.0
Face latérales	232150.00	0.0	386671.96	215098.06	297943.09	252460.9

TABLE 5.10: Flux de chaleur intégré Q [W] par face cas mandataires 4 à 6

—	M004		M005		M006	
	Mistral-CFD	Calima	Mistral-CFD	Calima	Mistral-CFD	Calima
Face 1	14921.38	0.0	7690.28	0.0	9833.52	0.0
Face 2	182154.16	199404.34	156626.82	151611.04	150772.19	114838.25
Face 3	32006.51	0.0	23323.24	0.0	202182.79	156770.41
Face 4	32006.51	0.0	23323.24	0.0	5415.27	0.0
Face 5	32092.56	0.0	7019.62	0.0	10257.57	0.0
Face 6	32092.56	0.0	156511.39	140902.37	150990.88	114762.67

autres (parallèles à l'écoulement ou masquées) Calima trouve un flux de chaleur nul, alors qu'il est non nul dans les résultats de Mistral-DSMC ce qui explique la différence de flux de chaleur intégré observé précédemment. C'est ce que l'on observe sur la Figure 5.14, où l'on note bien qu'à l'intérieur du cylindre le flux est non nul dans le calcul Mistral-DSMC.

5.11.1.2 Analyse et Bilan des comparaisons des résultats de Calima et des contributions aérothermodynamique au SCDW

Pour ce qui est de la détermination des coefficients aérodynamiques C_D et C_L nous avons observé que, sur le domaine de validité de la méthode mise en œuvre, les résultats obtenus avec Calima sont pratiquement identiques à ceux obtenus avec Pampero. Faute de données fournies par les contributeurs nous n'avons pu comparer ces résultats avec des calculs CFD.

Pour ce qui est du flux de chaleur nous avons observé que ce dernier est systématiquement inférieur à celui calculé par les outils de CFD et ceux de DSMC. Une analyse des contributions par face nous a indiqué que cette différence venait du fait que la méthode de calcul actuellement mise en œuvre dans Calima ne permet pas de calculer les contributions des faces parallèles à l'écoulement du flux ou qui sont masquées, les calculs CFD

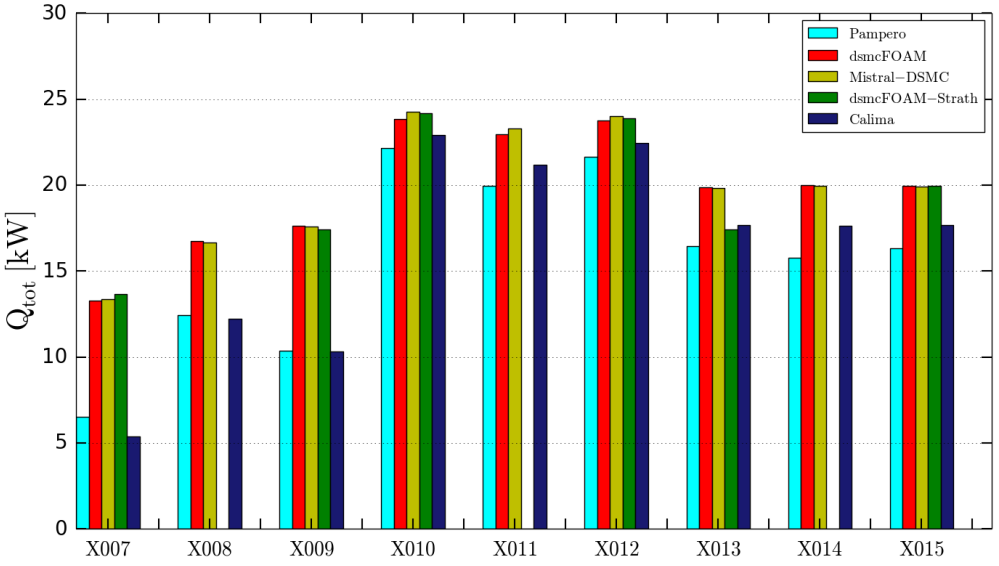

FIGURE 5.13: Flux de chaleur intégré sur la surface en kW pour les cas optionnels

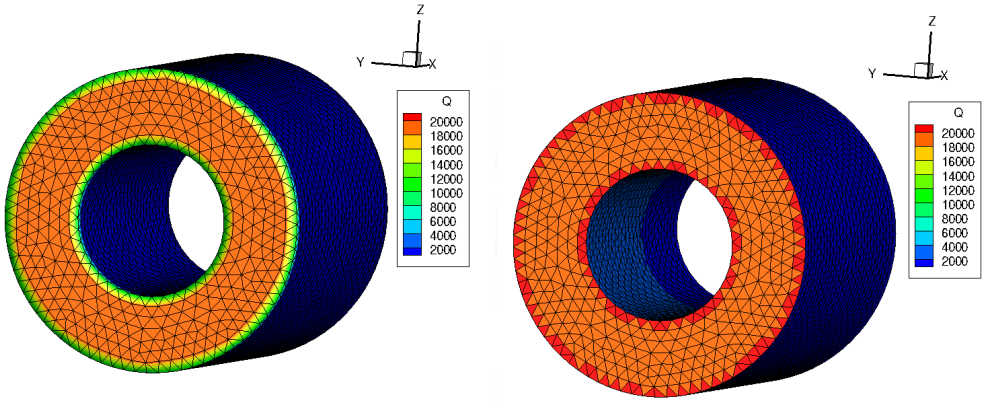

FIGURE 5.14: Flux de chaleur en W/m^2 pour le cas X009 calculé par Calima à gauche et par Mistral-DSMC à droite

et DSMC nous indiquant que ces faces contribuent au flux de chaleur total reçu par l'objet de façon non négligeable. Lorsqu'on pousse davantage l'analyse de ces résultats on constate que l'origine de cette différence de comportement vient de la détermination des coefficients de pression C_p dans Calima (cf. Annexe E.4.3.2). En effet dans Calima ces derniers sont nuls pour les faces parallèles à l'écoulement ou masquées (cf. Figure 5.15), alors que ce n'est pas le cas dans les calculs CFD et DSMC. Ainsi dans le futur pour améliorer la détermination du flux thermique dans les codes de calculs simplifiés il faudrait améliorer la méthode de calcul des coefficients de pressions. Néanmoins cela s'avère trés compliqué par les méthodes simplifiées, une solution pourrait être la prise en considération des longueurs des lignes de courant pour la définition des facteurs de vue, mais qui pose la question de la définition de ces lignes de courants. L'amélioration des calculs des coefficients de pression passera alors peut être par des méthodes CFD simplifiées.

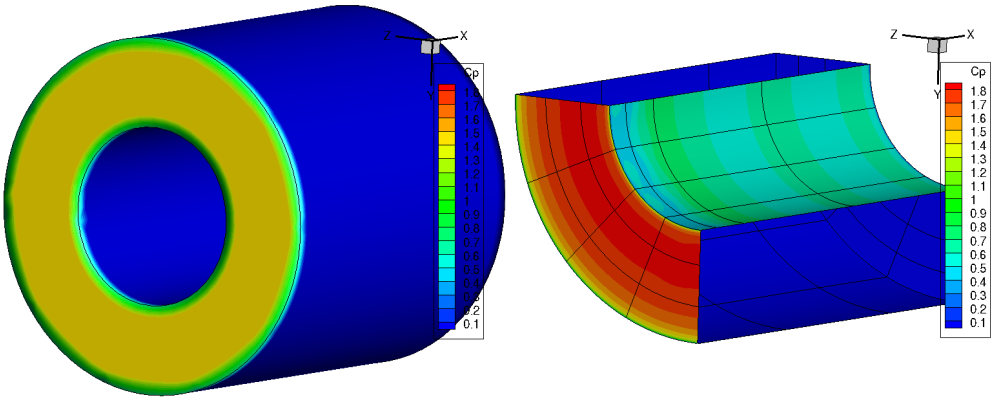

FIGURE 5.15: Coefficient de pression(Cp) pour le cas M001 à gauche calculé par Calima, à droite par Mistral-CFD

TABLE 5.11: Conditions initiales du cas Test 1 et leurs incertitudes associées

	Valeur nominale	1σ
Altitude [m]	120000.00	333.33
Vitesse [m/s]	7600	121.24
Masse [kg]	247.224	8.2
Latitude [°]	0.0	0.33
Longitude [°]	0.0	0.33
Azimuth [°]	45.0	0.33
FPA [°]	-2.612	0.33
Chaleur de Fusion du AA7075 [J/kg]	376788.0	12559.59
Température de Fusion du AA7075 [K]	830.0	27.66
Cp du AA7075 [J/kg/K]	1012.35	33.91
Emissivité du AA7075	0.141	0.004

TABLE 5.12: Points de calcul des coefficients aérothermodynamiques

Altitude	Mach
90000	26.6
70000	24.26
51609	18.82
30104	4.84
20117.9	1.11
10780	0.6
7555	0.47
5409	0.41

5.11.2 Intégration

5.11.2.1 Cas test 1

Nous allons à présent nous concentrer sur le cas test 1 de la section "Intégration" de la première édition du SCDW dont les conditions initiales sont présentées dans la première colonne de la Table 5.11. Dans ce cas on considère la rentrée atmosphérique d'une sphère de 0.5m de rayon extérieur constituée d'un alloy d'aluminium AA7075

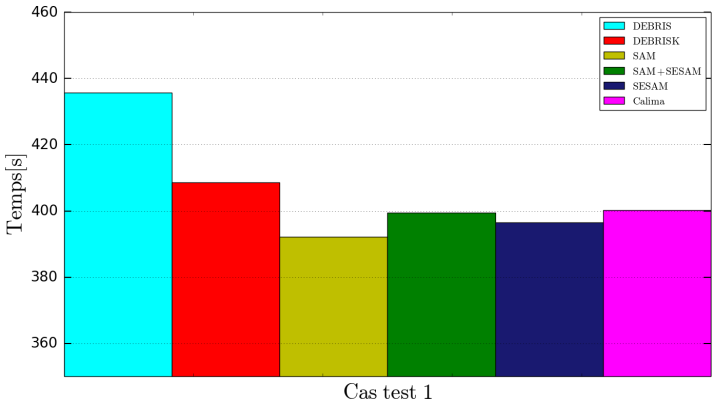

FIGURE 5.16: Durée de la rentrée atmosphérique du cas 1

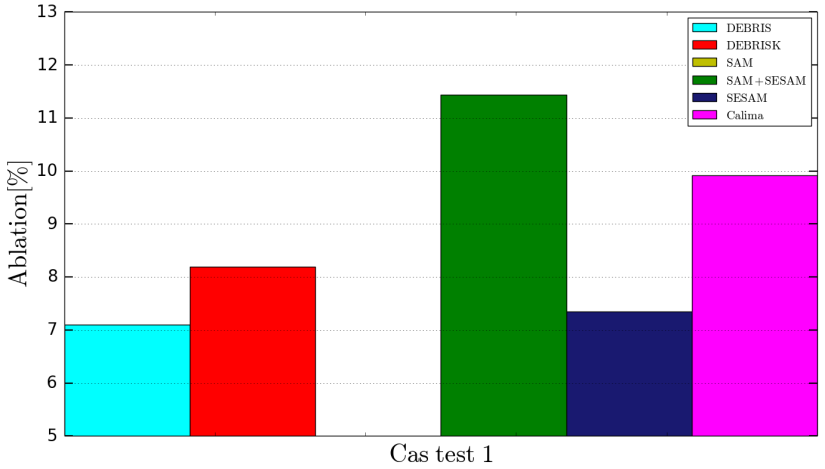

FIGURE 5.17: Taux d'ablation du cas 1

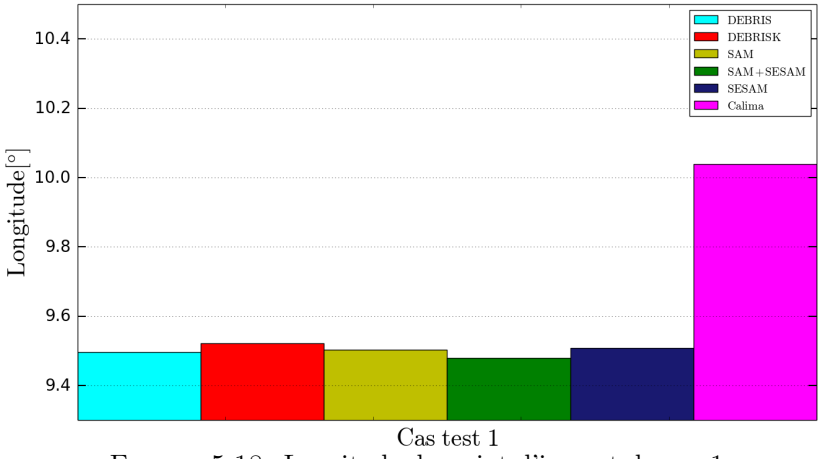

FIGURE 5.18: Longitude du point d'impact du cas 1

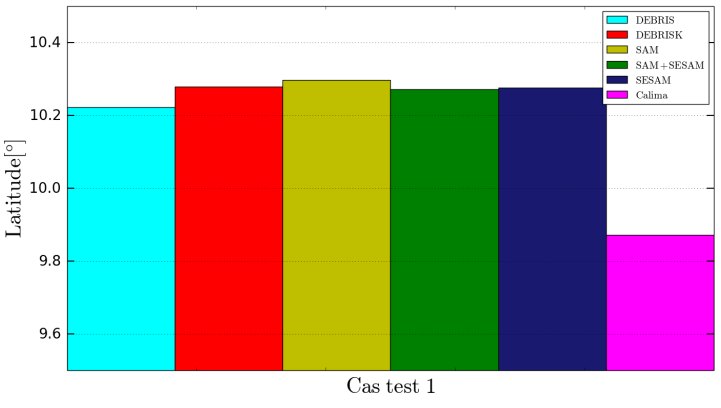

FIGURE 5.19: Latitude du point d'impact du cas 1

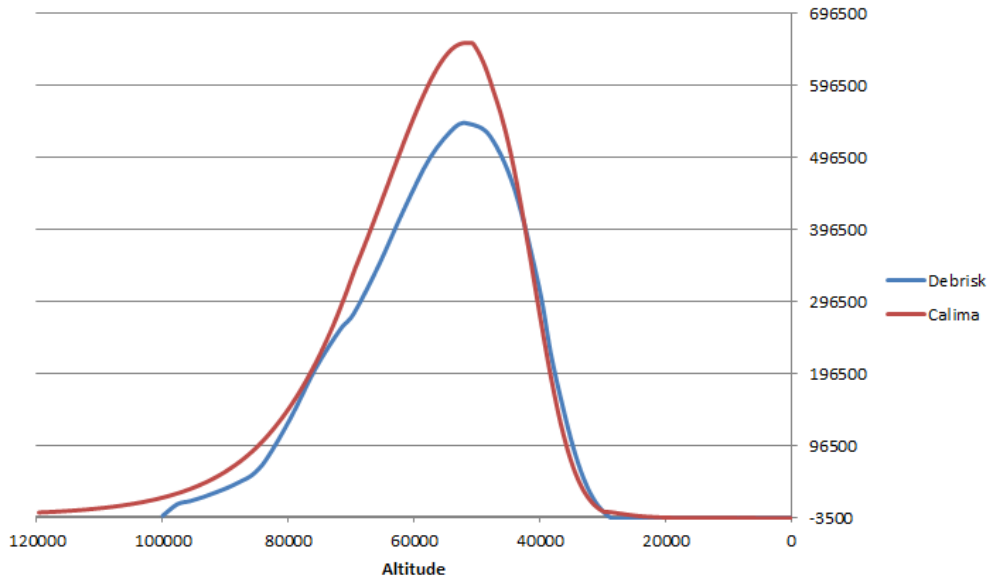

FIGURE 5.20: Évolution du flux thermique net (en W/m^2) sur la paroi en fonction de l'altitude (en m)

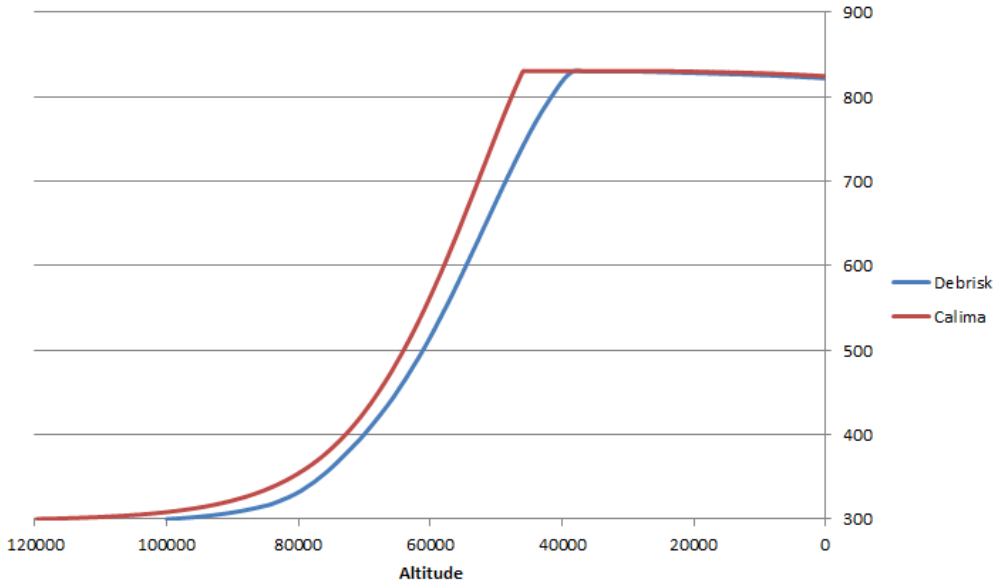

FIGURE 5.21: Évolution de la température de la paroi (en K) en fonction de l'altitude (en m)

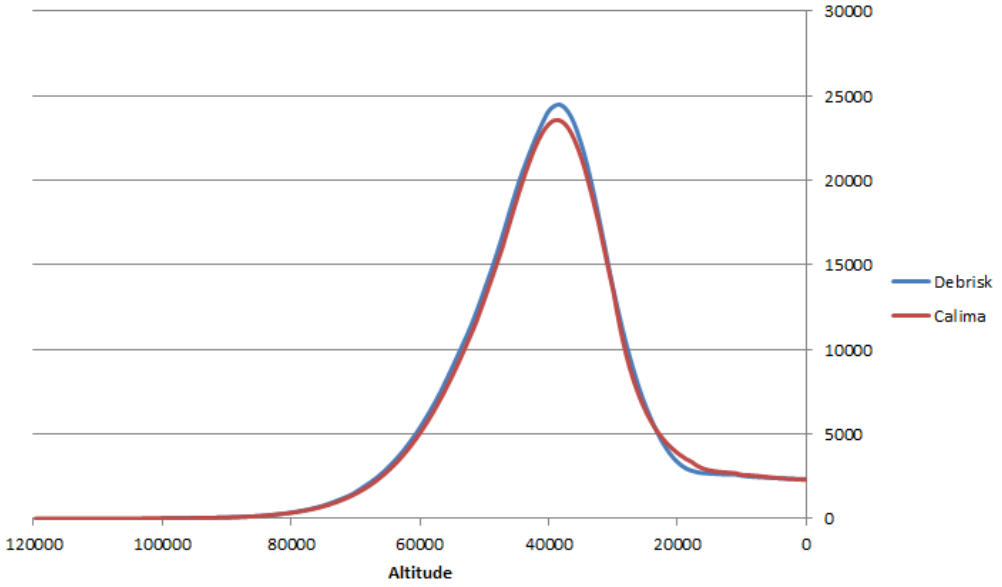

FIGURE 5.22: Évolution de la force de traînée (en N) en fonction de l'altitude (en m)

et pesant 247.224kg. Afin de mener le pré-calcul des coefficients aérodynamiques et aérothermodynamiques nous avons pris le parti de nous baser sur 8 points de vols issus de la simulation Debrisk et présentés Table 5.12 en utilisant une sphère maillée comprenant 1200 cellules et en tirant 100 rayons par cellule pour la détermination des facteurs de vue. Les Figures 5.16 à 5.19 présentent les résultats obtenus par les différent contributeurs du SCDW et Calima, respectivement pour le temps de rentrée atmosphérique (Figure 5.16), le taux d'ablation calculé en pourcentage (Figure 5.17) et la position du point d'impact en longitude (Figure 5.18) et latitude (Figure 5.19). Dans un premier temps notons que le taux d'ablation calculé par Calima est supérieur à celui calculé par les autres outils (cf. Figure 5.17) avec un taux de 9.91%. Seul SAM utilisant la base thermique de SESAM obtient un taux d'ablation supérieur. Lorsqu'on analyse plus en détail la simulation, notamment en comparant les résultats de Calima avec ceux de Debrisk, on constate que Calima calcule un flux net, qui correspond à la somme du flux convectif et des flux radiatifs, supérieur à celui de Debrisk (cf. Figure 5.20) ce qui implique une augmentation plus rapide de la température de la paroi (cf. Figure 5.21) qui va donc atteindre plus rapidement la température de fusion et ablater plus longtemps. Notons ici que ce taux d'ablation calculé par Calima est encore sous évalué par rapport à la réalité puisque nous avons vu à la sous-section 5.11.1 que Calima sous-estimait systématiquement le flux de chaleur par rapport à des calculs CFD et DSMC. Considérons ensuite la durée de rentrée, ainsi que la position finale en latitude et longitude. On constate que pour ce qui est de la durée de rentrée Calima obtient un temps de rentrée qui est cohérent avec celui des autres contributeurs du SCDW (cf. Figures 5.16). Ensuite pour ce qui est de la position du point d'impact on note un écart sur la position en latitude d'environ 0.5° (cf. Figure 5.18) et de 0.4° en latitude (cf. Figure 5.19). Cette différence de point d'impact simulé se comprend lorsqu'on considère l'évolution de la force de trainée. La Figures 5.22 représente l'évolution de la force de traînée en fonction de l'altitude calculée par Calima et Debrisk. Comme on peut le constater on a une évolution différente de la force de traînée entre les deux codes de simulation numérique. Cela vient notamment du fait que dans Debrisk les valeurs du coefficient de traînée sont calculées via des formules établies de façon empirique alors que, comme on l'a expliqué à la sous-section 5.4.7, dans Calima ce coefficient est calculé à partir du calcul des forces de pression sur une structure maillée. Faute de données dans les résultats du workshop, il n'est pas possible de faire la même comparaison pour les forces de portance et de portance latérale. La même explication s'applique pour justifier les différences observées entre Calima et les autres logiciels qui comme Debrisk se basent sur des méthodes simplifiées. Calima se basant sur une méthode issue de la modélisation orientée véhicules spatiaux, pour le calcul des coefficients aérodynamiques, il aurait été intéressant de le confronter aux résultats d'un des logiciels orientés objets de référence (SCARAB ou Pampero) or aucun de ces logiciels n'a participé à la section

intégration du workshop. On peut néanmoins citer Lips et al. 2005 [83] qui a déjà effectué une étude comparant SCARAB avec d'autres logiciels orientés objets et qui a observé des différences sur le calcul du coefficient de traînée entre les deux approches (orienté véhicule spatial et objet) mais n'a pas discuté de l'impact de ces différences sur la détermination du point d'impact.

5.11.2.2 Analyse de Taguchi

Ce qui différencie Calima des autres logiciels qui ont contribué au SCDW c'est sa capacité à effectuer différents types d'analyse statistique (DEBRIS de la société DEIMOS ne permettant que des analyses de Monte-Carlo). Nous allons présenter quels types d'information Calima permet d'obtenir en effectuant successivement, sur le cas test 1 de la section Intégration du SCDW, une analyse de type Taguchi afin de déterminer parmi un ensemble de paramètres avec des incertitudes quels sont ceux qui ont le plus d'influence sur la variation de trois paramètres : la position finale en latitude et longitude, ainsi que le paramètre de survivabilité. Nous meneront ensuite une analyse de Monte-Carlo à partir des paramètres qui auront été identifiés comme ayant le plus d'influence et analyseront la dispersion des résultats.
La première édition du SCDW n'ayant pas prévu d'effectuer d'analyse statistiques, [65] ne fourni pas les incertitudes sur les paramètres initiaux du cas 1 d'intégration. C'est pour cette raison que nous avons décidé de nous baser sur les incertitudes qui ont été définies en partenariat avec le CNES dans le cadre du [DBK-CT-LOG-0371-CNES]. La Table 5.11 présente ainsi les différents paramètres utilisés, leur valeur nominale ainsi que l'écart type à 1 σ associé. Nous avons ainsi 11 paramètres à perturber. Pour la suite de cette étude nous considèrerons toujours une incertitude Gaussienne sur les différents paramètres. Pour mener cette analyse de Taguchi nous avons employé la table orthogonale à 3 niveaux L_{27}. Les Tables 5.13, 5.14 et 5.15 présentent les tables d'ANOVA issues de l'analyse des 27 simulations respectivement pour la longitude, la latitude et le paramètre de survivabilité. Dans ces tableaux sont identifiés en vert pour chaque table le paramètre qui a le plus d'influence sur la dispersion du résultat final, en orange ceux qui ont une contribution supérieure à 1% à cette dispersion et en rouge ceux qui ont une contribution inférieure à 1%. On note que pour la longitude le paramètre qui a le plus d'influence sur la dispersion de la longitude du point d'impact est la longitude initiale avec une contribution de 99.37%. L'altitude, la latitude, la capacité thermique (Cp), l'émissivité, la température de fusion et la chaleur de fusion de AA7075 ont tous une contribution clairement négligeable, de l'ordre de 10^{-4}%. On observe le même comportement pour la latitude, avec la latitude initiale ayant le plus d'influence sur la dispersion de la latitude du point d'impact (98.99%) et l'altitude, la latitude, la capacité thermique,

TABLE 5.13: Table d'Analyse de la Variance pour la Longitude

	f	SS	V	F	SS'	P
Masse	2	5.07	2.53	$1.78e^{+2}$	5.04	0.05
Altitude	2	$9.23e^{-3}$	$4.62e^{-3}$	$3.23e^{-1}$	$1.93e^{-2}$	$1.98e^{-4}$
Latitude	2	$9.63e^{-2}$	$4.81e^{-2}$	3.38	$6.78e{-}2$	$6.97e^{-4}$
Longitude	2	$9.67e^{+3}$	$4.83e^{+3}$	$3.39e^{+5}$	$9.67e+3$	99.37
Azimuth	2	$1.49e^{+1}$	7.48	$5.25e^{+2}$	$1.49e^{+1}$	0.15
Vitesse	2	5.23	2.62	$1.84e^{+2}$	5.21	0.05
FPA	2	$3.60e^{+1}$	$1.80e^{+1}$	$1.26e^{+3}$	$3.6e^{+1}$	0.37
Cp de AA7075	2	$4.46e^{-2}$	$2.23e^{-2}$	1.57	$1.61e^{-2}$	$1.66e^{-4}$
Emissivité de AA7075	2	$9.01e^{-3}$	$4.50e^{-3}$	$3.16e^{-1}$	$1.95e^{-2}$	$2.0e^{-4}$
Température de fusion de AA7075	2	$8.58e^{-3}$	$4.29e^{-3}$	$3.01e^{-1}$	$1.99e^{-2}$	$2.05e^{-4}$
Chaleur de fusion de AA7075	2	$3.07e^{-3}$	$1.54e^{-3}$	$1.08e^{-1}$	$2.54e^{-2}$	$2.61e^{-4}$
Erreur	4	$5.7e^{-2}$	$1.42e^{-2}$	1.0		

TABLE 5.14: Table d'Analyse de la Variance pour la Latitude

	f	SS	V	F	SS'	P
Masse	2	2.36	1.18	$9.59e^{+2}$	2.36	0.02
Altitude	2	$4.67e^{-3}$	$2.34e^{-3}$	1.89	$2.21e^{-3}$	$2.26e^{-5}$
Latitude	2	$9.64e^{+3}$	$4.82e^{+3}$	$3.91e^{+6}$	$9.64e^{+3}$	98.99
Longitude	2	$4.32e^{-4}$	$2.16e^{-4}$	$1.75e^{-1}$	$2.03e{-}3$	$2.09e^{-5}$
Azimuth	2	6.1	3.06	$2.48e^{+3}$	6.12	0.06
Vitesse	2	2.33	1.16	$9.42e^{+2}$	2.32	0.02
FPA	2	$8.61e^{-1}$	$4.34e^{-1}$	$3.52e^{+4}$	$8.68e^{+1}$	0.89
Cp de AA7075	2	$1.46e^{-3}$	$7.31e^{-4}$	$5.92e^{-1}$	$1.0e^{-3}$	$1.03e^{-5}$
Emissivité de AA7075	2	$2.75e^{-3}$	$1.37e^{-3}$	1.11	$2.87e^{-4}$	$2.95e^{-6}$
Température de fusion de AA7075	2	$6.28e^{-4}$	$3.14e^{-4}$	$2.54e^{-1}$	$1.84e^{-3}$	$1.89e^{-5}$
Chaleur de fusion de AA7075	2	$4.25e^{-4}$	$2.12e^{-4}$	$1.72e^{-1}$	$2.04e^{-3}$	$2.1e^{-5}$
Erreur	4	$4.93e^{-3}$	$1.23e^{-3}$	1.0		

TABLE 5.15: Table d'Analyse de la Variance pour le paramètre de survivabilité

	f	SS	V	F	SS'	P
Masse	2	$1.18e^{-4}$	$5.88e^{-5}$	1.34	$3.01e^{-5}$	0.68
Altitude	2	$1.14e^{-4}$	$5.71e^{-5}$	1.30	$2.65e^{-5}$	0.6
Latitude	2	$6.09e^{-5}$	$3.05e^{-5}$	$6.95e^{-1}$	$2.67e^{-5}$	0.6
Longitude	2	$1.65e^{-6}$	$8.25e^{-7}$	$1.88e^{-2}$	$8.59e^{-5}$	1.94
Azimuth	2	$1.18e^{-4}$	$5.88e^{-5}$	1.34	$3.01e^{-5}$	0.68
Vitesse	2	$1.17e^{-4}$	$5.88e^{-5}$	1.34	$3.00e^{-5}$	0.67
FPA	2	$3.55e^{-3}$	$1.77e^{-3}$	$4.05e^{+1}$	$3.46e^{-3}$	78.1
Cp de AA7075	2	$6.11e^{-5}$	$3.05e^{-5}$	$6.97e^{-1}$	$2.65e^{-5}$	0.59
Emissivité de AA7075	2	$1.14e^{-4}$	$5.73e^{-5}$	1.31	$2.69e^{-5}$	0.61
Température de fusion de AA7075	2	$1.62e^{-6}$	$8.14e^{-7}$	$1.85e^{-2}$	$8.59e^{-5}$	1.94
Chaleur de fusion de AA7075	2	$1.70e^{-6}$	$8.52e^{-7}$	$1.94e^{-2}$	$8.59e^{-5}$	1.94
Erreur	4	$1.75e-4$	$4.3e-5$	1.0		

TABLE 5.16: Récapitulatif des propriétés statistiques

	Longitude	Latitude	Ps
Min	-125.082	-89.943	-1.00
Max	118.782	42.576	-0.273
Moyenne	-9.635	-9.893	-0.983
Sigma	15.814	15.167	0.085
Skewness	0.367	-0.098	4.232
Kurtosis	5.667	3.511	22.692

l'émissivité, la température de fusion et la chaleur de fusion de AA7075 ayant une contribution de l'ordre ou inférieure à $10^{-5}\%$. Pour ce qui est du paramètre de survivabilité, le paramètre qui a le plus d'influence sur sa dispersion est l'angle de trajectoire de vol (FPA) avec une contribution de 78.1%. La masse, l'altitude, la latitude, l'azimuth, la vitesse, la capacité thermique et l'émissivité de AA7075 ont une influence négligeable sur la dispersion du paramètre de survivabilité avec une contribution inférieure au pourcent. Suite à l'analyse de Taguchi on peut donc négliger les incertitudes sur la masse, l'altitude, l'azimuth, la vitesse, la capacité thermique et l'émissivité du AA7075, réduisant notre espace des paramètres de 11 à 5 paramètres. A noter que ces conclusions ne sont valables que pour ce cas, avec ces conditions de perturbations et ne seraient être utilisées pour tirer des conclusions générales. De la même façon il peut être hasardeux de chercher à expliquer ces résultats par rapport à la physique implémentée dans le logiciel tant ceux-ci dépendent des valeurs d'incertitudes employées (cf. [64]).

5.11.2.3 Méthode de Monte-Carlo

Pour notre analyse de Monte-Carlo on ne considère plus que les incertitudes sur la latitude, la longitude, l'angle de la trajectoire de vol, la température et la chaleur de fusion de AA7075, toujours avec les incertitudes définies dans la Table 5.11. Nous renvoyons à la démonstration effectuée à la sous-section 2.2.4.3 concernant la validation de cette réduction de paramètre par la méthode de Taguchi. Nous effectuons 300 000 simulations sur un accélérateur de calcul GPU K5000 en $1400s$ soit un peu moins de 23 minutes. A titre de comparaison si on avait dû effectuer le même nombre de simulations avec le logiciel Debrisk, en parallèle sur les 20 cœurs d'un Xeon E5-2620-v2 ce calcul aurait duré environ 120 $000s$ soit un peu plus de 33 heures.

La Table 5.16 présente les propriétés statistiques des trois résultats que nous avons choisis de considérer pour cette analyse : la position du point d'impact en latitude et longitude, et le paramètre de survivabilité. Dans un premier temps, considérons le paramètre de survivabilité. On note une faible dispersion du paramètre de survivabilité avec une valeur moyenne à -0.975 et un écart type de 0.085. Le kurtosis de 22.69 nous indique que la distribution des valeurs du paramètre de survivabilité est piquée tandis que le skewness positif nous indique que la queue de la distribution est étalée sur la

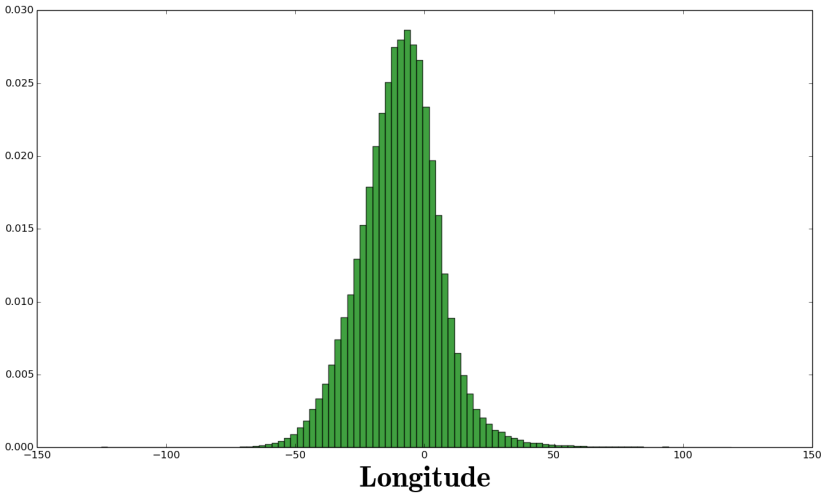

FIGURE 5.23: Fonction de densité de probabilité de la longitude du point d'impact

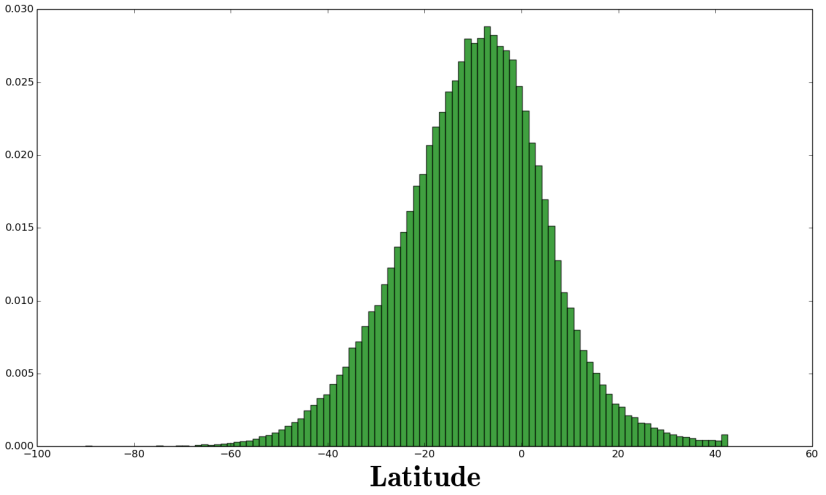

FIGURE 5.24: Fonction de densité de probabilité de la latitude du point d'impact

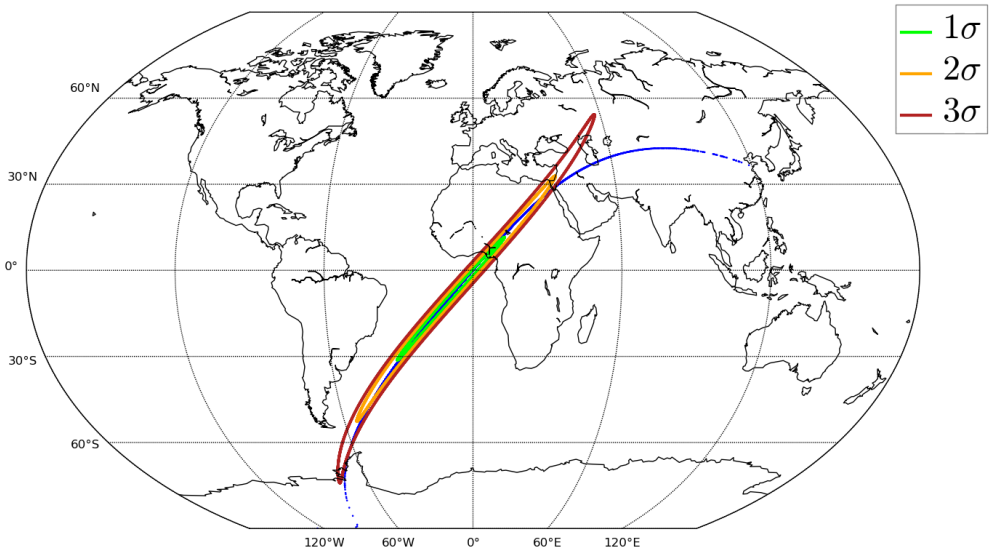

FIGURE 5.25: Distribution du point d'impact en longitude-latitude (points bleu) et zone probable d'impact à 1,2 et 3 σ

droite de la moyenne. En d'autre terme la majorité des résultats de ces simulations se trouvent entre -1.0 et -0.974. Si on considère ensuite la distribution du point d'impact en latitude-longitude, la première chose que l'on note, c'est l'importante distribution des résultats. En effet les valeurs possibles de longitude vont de $-125.082°$ à $118.782°$ avec un écart type pour la distribution de $15.815°$ tandis que les valeurs possibles de latitude vont de $-89.943°$ à $42.576°$ avec un écart type pour la distribution de $15.167°$. L'autre constat que l'on peut faire c'est que les valeurs moyennes ne correspondent pas du tout aux résultats obtenus lors de la simulation non perturbée (cf. sous-section 5.11.2.1). Le phénomène de rentrée atmosphérique étant un phénomène hautement non linéaire ce résultat n'est pas surprenant. Si l'on considère ensuite le skewness et le kurtosis on constate qu'avec un kurtosis de 5.66, la distribution de la position du point d'impact en longitude est piquée et avec un skewness de 0.367 la queue de la distribution est plus décalée sur la droite de la moyenne, tandis que pour la latitude le skewness de 3.511 nous indique une distribution plus piquée qu'une gaussienne avec une queue de distribution décalée à gauche avec un skewness de -0.09. Et en effet lorsqu'on considère les fonctions de densité de probabilité de ces deux grandeurs, Figures 5.23 et 5.24, on constate bien que pour la longitude on a une distribution assez piquée tout comme pour la latitude. Par contre pour ce qui est de la queue de distribution, si on note bien que la queue est plus étendue à gauche de la valeur moyenne pour la latitude, c'est moins évident pour la longitude. Notre outil d'analyse statistique nous permet ensuite de déterminer aussi les caractéristiques des ellipses de confiance à 1, 2 et 3 σ définissant des zones probables d'impact. Pour notre distribution on trouve ainsi une ellipse avec un demi-grand axe de 3 426.4 km et un demi-petit axe de 103.9 km, pour une surface totale de 1 119 058.4 km^2. La Figure 5.25 présente ainsi la dispersion des points d'impact en latitude-longitude ainsi que les zones probables d'impact à 1, 2 et 3 σ. Á noter que nous définissons ici des ellipses, l'allure distordu sur la Figure 5.25 vient de la projection sur le planisphère.

5.11.3 Bilan des résultats expérimentaux

Dans cette sous-section nous avons confronté les résulats de Calima avec ceux d'autres logiciels produits dans le cadre de la première édition du Spacecraft Demise Workshop. Nous avons ainsi étudié les cas aérothermodynamiques de ce workshop et un cas d'intégration que nous avons étendu afin de mettre en avant les capacités de Calima. Concernant les cas aérothermodynamiques, nous avons mis en évidence que dans le domaine de validité de la méthode de calcul des coefficients aérodynamiques, Calima donnait des résultats quasi-identiques à ceux d'un logiciel purement orienté vehicule spatial : Pampero. Nous avons ensuite mis en avant le fait qu'à cause des limitations de la méthode de calcul des coefficients de pression sur les faces d'un objet, Calima

sous-estime systématiquement d'environ 30% le flux thermique sur un objet par rapport à des codes CFD ou DSMC. En effet, Calima ne permet de calculer le flux thermique reçu, uniquement pour les parois exposées au flux, et néglige les contributions des parois parallèles à l'écoulement ou même masquées. Concernant la section intégration, nous avons étudié le cas 1 de cette section. Nous avons constaté dans un premier temps sur une simulation non perturbée, que Calima produisait des résultats qui, bien que différents, restaient cohérents avec ceux des autres logiciels. Nous avons ensuite justifié les différences entre les outils et avons montré que celles-ci avaient deux origines : un taux d'ablation un peu plus important dans Calima et une force de traînée différente. Nous avons ensuite dévellopé ce cas avec Calima en montrant comment on pouvait utiliser ses outils d'analyses statistiques pour étudier la dispersion du point d'impact de l'objet ainsi que celle de son paramètre de survivabilité qui nous éclaire sur l'ablation de l'objet. Nous avons ainsi mené dans un premier temps une analyse de Taguchi sur 11 paramètres afin de limiter le nombre de paramètres à considérer en vue d'une analyse de Monte-Carlo. Cette analyse nous a ainsi permis de réduire notre ensemble de paramètres à 5 ayant le plus d'influence sur la dispersion de ces 3 résultats (longitude et latitude du point d'impact et paramètre de survivabilité). Nous avons ensuite mené une analyse de Monte-Carlo, calculant 300 000 simulations en un peu plus de 23 minutes grâce à l'utilisation d'accélérateur de calcul GPU (contre 33h pour un outil classique sur une machine à 20 cœurs). À partir des outils mis en œuvre dans le script d'analyse actuel nous avons ensuite présenté quelles informations nous pouvons extraire de cette analyse de Monte-Carlo. A l'heure actuelle l'analyse des résultats de Calima ne permet que d'obtenir des propriétés statistiques globales sur les résultats. Dans le futur il sera intéressant de pouvoir coupler les données de simulations de Calima avec une base de donnée de type SQL afin de permettre une recherche et une analyse plus fine, par exemple pour analyser en détail l'influence de la valeur d'un paramètre initial sur la variation d'un résultat. Ce genre de base de donnée s'avèrerait particulièrement pertinente dans le cadre d'une analyse de sensibilité poussée, ou dans le cadre d'une recherche d'optimum (via un test de $\chi 2$ par exemple) où l'on souhaiterait trouver les combinaisons de paramètres initiaux permetant d'obtenir un résultat donnée. En effet une telle analyse n'est pour l'instant pas possible avec Calima, ce dernier ne conservant pas l'historique des conditions initiales amenant à un résutat donné : par exemple quel jeu de conditions initiale permet d'obtenir une ablation maximale de l'objet.

5.12 Conclusion

Dans ce chapitre nous avons présenté le logiciel de simulation de rentrée atmosphérique de débris parallèle sur GPU que nous avons conçu et développé : Calima. Nous avons dans un premier temps présenté l'ensemble des composants de ce logiciel orienté objets

ainsi que la façon dont ils ont été mis en œuvre afin d'obtenir un logiciel pouvant effectuer des simulations sur CPU et sur GPU. Nous avons notamment présenté comment nous avons mis en œuvre notre approche nouvelle consistant à précalculer les coefficients aérothermodynamiques pour les réutiliser dans le cœur de la simulation. Nous avons aussi présenté les différentes méthodes d'analyses statistiques qui sont mises en œuvre dans Calima. Nous avons ensuite réalisé une série de comparaisons en nous basant sur les résultats fournis par l'ensemble de contributeurs de la première édition du Spacecraft Demise Workshop. Une des conclusions majeures qui ressort de cette étude comparative est la sous-estimation systématique du flux de chaleur par Calima par rapport aux logiciels de référence CFD ou DSMC, venant d'une mauvaise détermination des coefficients de pression. Ces derniers sont nuls pour les faces parallèles ou masquées à l'écoulement dans Calima, alors que les calculs CFD et DSMC nous montrent le contraire. Compte tenu de la difficulté qu'implique la modification de la méthode de détermination des coefficients de pression, il serait intéressant de généraliser cette étude comparative sur un ensemble plus large de formes géométriques et de conditions d'écoulements. De cette façon on pourrait alors essayer de caractériser de façon plus générale l'écart entre le flux de chaleur calculé par Calima et celui calculé par les outils CFD, et ainsi définir l'incertitude que l'on a sur la détermination du coefficent thermique afin de l'utiliser dans le cadre d'une analyse statistique avec Calima. Dans un second temps nous avons comparé les résultats du premier cas de la section intégration du workshop. Nous avons montré que Calima obtenait des résultats cohérents avec les autres contributeurs du workshop, bien que légèrement différents. Nous avons alors justifié ces différences à partir des différences de modélisation, notamment en comparant les résultats avec ceux de Debrisk. Nous avons ensuite décrit les capacités d'analyses statistiques qu'offre Calima en effectuant successivement une analyse de Taguchi, nous permettant de réduire notre espace des paramètres de 11 à 5, puis une analyse de Monte-Carlo. Au vu de la taille de la tâche de dispersion des résultats du cas Test 1 du workshop lors de l'analyse de Monte-Carlo, on comprend aisément l'intérêt de l'approche statistique que nous avons mis en œuvre. Mais la question de la détermination de ces incertitudes demeure. En effet, pour les besoins de cette analyse nous nous sommes basés sur des incertitudes qui ont été définies en partenariat avec des experts de l'agence spaciale française. Néanmoins, ces incertitudes ont été définies de manière intuitive. À l'avenir, pour permettre une meilleure exploitation des résultats d'analyse statistique, il conviendrait de baser la détermination des incertitudes sur des comparaisons avec des résultats expérimentaux.

Conclusions

Les travaux de cette thèse ont porté sur la modélisation numérique de la rentrée atmosphérique d'engin spatiaux. En particulier, un nouvel outil permettant d'effectuer diverses analyses statistiques, telles que les analyses de type Monte-Carlo ou Taguchi, de rentrée atmosphérique de satellites a été développé et à fait l'objet d'étude de parallélisation au moyen d'accélérateurs de calculs tels que les Graphics Processing Units ou le Xeon Phi d'Intel.

Dans le Chapitre 2, nous avons commencé par présenter les différents outils d'analyse statistique que nous avons mis en œuvre dans le cadre de ces travaux. Nous avons ainsi fait une présentation des méthodes de Monte-Carlo, décrivant leurs modes de fonctionnement ainsi que les informations que permettait d'obtenir une analyse statistique de type Monte-Carlo. Nous avons ensuite discuté des limitations de ce type d'analyse et avons introduit la méthode de Taguchi, qui permet en peu de calculs de déterminer, dans un ensemble de paramètres dotés d'incertitudes, quels sont ceux qui ont le plus d'influence sur la distribution d'un résultat, et quels sont ceux qui ont une influence négligeable. Nous avons ensuite exposé comment la méthode de Taguchi pouvait être utilisée en préambule d'une analyse de Monte-Carlo, en permettant de ne considérer dans l'analyse de Monte-Carlo que les paramètres ayant le plus d'influence, sans altérer la qualité des résultats : on a les mêmes propriétés statistiques sur la distribution d'un résultat obtenu avec une analyse de Monte-Carlo avec ou sans analyse de Taguchi au préalable.

Dans le Chapitre 3, nous avons présenté les accélérateurs de calculs de type carte graphique et Xeon Phi, décrivant tour à tour les architectures matérielles de ces produits ainsi que les environnenements logiciels et de programmation qui ont été développés par les constructeurs respectifs de ces cartes accélératrices : NVIDIA et Intel. Nous avons ainsi mis en avant le fait que ces deux accélérateurs constituaient deux réponses différentes à une même problématique de calcul intensif : les cartes graphiques misant leur puissance de calcul sur une approche massivement parallèle avec un nombre important d'unité de calcul, tandis que les cartes Xeon Phi compensent leur plus faible nombre d'unité de calcul par une approche massivement vectorielle. Ces cartes accélératrices s'avèrent toutes désignées pour traiter des problèmes agréablement parallèles tels que

les problèmes d'analyse de Monte-Carlo que nous cherchons à traiter.
Dans le Chapitre 4, nous avons ensuite présenté une première étude que nous avons menée concernant la mise en œuvre d'une analyse de Monte-Carlo sur cartes accélératrices GPU et Xeon Phi. Nous avons ainsi traité un problème, issue du monde réel, le cas de la dérive descente de ballon stratosphérique pour ces similitudes avec la problématique de la rentrée atmosphérique de ballons stratosphériques et pour la simplicité des modéles physiques mis en œuvre permettant de se concentrer sur les problématiques liées à la mise en œuvre de ce processus sur accélérateur de calcul. Au cours de cette étude nous avons mis en évidence la faisabilité de notre projet d'outil de simulation et d'analyse statistique sur carte accélératrice, permettant d'effectuer des analyses de types Monte-Carlo dans des conditions opérationnelles. Nous avons montré qu'une mise en œuvre de type algorithme à parallélisme de tâches avec gestion de la mémoire constituait la meilleure stratégie de mise en œuvre sur les deux types de cartes accélératrices. Nous avons aussi montré qu'en considérant un certain nombre d'impératifs la solution GPU s'avérait la mieux adapté à notre projet. Nous avons ensuite poursuivi nos études avec l'outil de dérive descente de ballon stratosphérique, en nous concentrant sur l'exploitation des résultats. Nous avons réanalisé des vols de la campagne de vols de ballons du CNES à Timmins en 2014. Nous avons ainsi montré dans cette étude que notre nouvelle approche permettait de définir des zones probables de retombées à 1, 2 et 3 σ contenant le point de retombée réel. Plus précisément sur six vols réanalysés, le point de retombée réel se trouvait dans la zone à 1 σ pour quatre de ces vols et pour les deux autres il se trouvait dans la zone à 2 σ. Notre méthode de perturbation des profils météorologiques volontairement naïve a, lors de cette étude, montré ses limites. Elle a notamment défini des zones de retombées plus grandes que celles définis par la méthode actuellement utilisée par les équipes du CNES. Mais notre méthode définit une borne maximale sur la qualité des résultats que peut produire ce logiciel. Ainsi dans le futur il faudrait travailler sur une amélioration de la méthode de détermination des incertitudes sur les prévisions météorologiques et la façon dont on les utilise pour perturber ces prévisions afin d'optimiser la méthode actuellement mise en œuvre dans notre code.
Dans le Chapitre 5 pour finir, nous avons étudié la rentrée atmosphérique d'engins spatiaux, en particulier nous avons présenté le logiciel parallèle de simulation de rentrée atmosphérique de satellite accéléré sur carte graphique que nous avons conçu et développé : Calima. Aprés avoir présenté les différents composants de ce logiciel ainsi que la façon dont nous les avons mis en œuvre, nous avons ensuite effectué des comparaisons entre les résultats obtenus par Calima et des logiciels de référence (Debrisk, Pampero, SAM, Debris, ...) en nous basant sur les résultats fournis par les différents contributeurs du Spacecraft Demise Workshop. Durant cette étude comparative nous avons montré que Calima permettait d'obtenir des coefficients aérodynamiques cohérents avec ceux calculés par le logiciel orienté véhicule spatial Pampero, et que sur une simulation de

rentrée atmosphérique complète Calima optenait des résultats cohérents avec ceux des autres participants. Nous avons ensuite montré l'intérêt de l'approche statistique de Calima sur le cas de rentrée atmosphérique complet traité auparavant de façon déterministe, en montrant l'étendue de la zone probable de retombée en considérant des incertitudes caractéristiques sur les conditions initiales définies en partenariat avec le Centre National d'Étude Spatiales, après avoir sélectionné les paramètres ayant le plus d'influence sur la dispersion du point de retombée et sur la survivabilité de l'engin par la méthode de Taguchi. Au cours de ces études nous avons aussi eu l'occasion de pointer les limites actuelles de notre code de simulation. Tout d'abord nous avons montré que la méthode de calcul des coefficients aérodynamiques actuellement mise en œuvre donne de mauvais résultats dans le domaine raréfié, notamment à cause de la formule employée pour le calcul de la force de pression qui n'est plus valide dans ce régime et aussi à cause du fait que l'on néglige les forces de frottement. Nous avons ensuite indiqué que lors de la détermination du coefficient thermique, Calima sous-évalue le flux de chaleur sur certaines faces de l'objet par rapport à des codes de CFD ou de DSMC. Dans le futur il serait intéressant de changer la méthode mise en œuvre pour calculer le flux de chaleur ou alors essayer de caractériser de façon plus systématique l'incertitude que l'on a sur cette sous-estimation afin de la prendre en compte dans les simulations.

Aujourd'hui Calima permet de simuler la rentrée atmosphérique d'un objet quelque soit sa forme, grâce à notre stratégie hybride unique, aliant les méthodes de détermination des coefficients aéro et aérothermodynamique inspirées des logiciels orientés véhicules spaciaux et les méthodes de simulation des logiciels orientés objets nous permettant précision et maîtrise des temps de calcul. Calima permet aussi d'étudier statistiquement les résultats de cette rentrée en tirant profit de la puissance de calcul parallèle des cartes graphiques. Afin d'inscrire Calima définitivement dans le cadre des futurs projets "Design for Demise" il sera pertinant dans le futur d'étendre les capacités de Calima, en permettant la prise en considération de plusieurs objets dans une simulation, de considérer des modèles de fragmentations, de traiter les intéractions entre les objets au cours de la rentrée. D'ores et déjà Calima peut traiter toutes les nouvelles géométries ou de nouveaux matériaux conçus dans le cadre de projets "Design for Demise". Aussi afin de rendre toujours plus pertinente les analyses statistiques, il faudra continuer à investiguer la problématique de la détermination des incertitudes sur les différents paramètres de la rentrée ainsi que sur les modèles notamment via des comparaisons avec d'autres simulations et à l'avenir certainement avec des résultats expérimentaux fournis par des engins spatiaux qui seront instrumentés.

Annexe A

Exemple de Tables Orthogonales

A.1 Table orthogonale L9

Nombre de Facteurs = 4
Nombre de niveaux pour chaque Facteur = 3
Nombre d'Observations = 9

1 1 1 1
1 2 2 2
1 3 3 3
2 1 2 3
2 2 3 1
2 3 1 2
3 1 3 2
3 2 1 3
3 3 2 1

A.2 Table orthogonale L27

Nombre de Facteurs = 13

Nombre de niveaux pour chaque Facteur = 3

Nombre d'Observations = 27

1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 2 2 2 2 2 2 2 2 2

1 1 1 1 3 3 3 3 3 3 3 3 3

1 2 2 2 1 1 1 2 2 2 3 3 3

1 2 2 2 2 2 2 3 3 3 1 1 1

1 2 2 2 3 3 3 1 1 1 2 2 2

1 3 3 3 1 1 1 3 3 3 2 2 2

1 3 3 3 2 2 2 1 1 1 3 3 3

1 3 3 3 3 3 3 2 2 2 1 1 1

2 1 2 3 1 2 3 1 2 3 1 2 3

2 1 2 3 2 3 1 2 3 1 2 3 1

2 1 2 3 3 1 2 3 1 2 3 1 2

2 2 3 1 1 2 3 2 3 1 3 1 2

2 2 3 1 2 3 1 3 1 2 1 2 3

2 2 3 1 3 1 2 1 2 3 2 3 1

2 3 1 2 1 2 3 3 1 2 2 3 1

2 3 1 2 2 3 1 1 2 3 3 1 2

2 3 1 2 3 1 2 2 3 1 1 2 3

3 1 3 2 1 3 2 1 3 2 1 3 2

3 1 3 2 2 1 3 2 1 3 2 1 3

3 1 3 2 3 2 1 3 2 1 3 2 1

3 2 1 3 1 3 2 2 1 3 3 2 1

3 2 1 3 2 1 3 3 2 1 1 3 2

3 2 1 3 3 2 1 1 3 2 2 1 3

3 3 2 1 1 3 2 3 2 1 2 1 3

3 3 2 1 2 1 3 1 3 2 3 2 1

3 3 2 1 3 2 1 2 1 3 1 3 2

A.3 Table orthogonale L81

Nombre de Facteurs = 40

Nombre de niveaux pour chaque Facteur = 3

Nombre d'Observations = 81

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1
1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
1 1 1 1 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3
1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1
1 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
1 1 1 1 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2
1 1 1 1 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3
1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1
1 2 2 2 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3
1 2 2 2 1 1 1 2 2 2 3 3 3 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3 1 1 1
1 2 2 2 1 1 1 2 2 2 3 3 3 3 3 3 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2
1 2 2 2 2 2 2 3 3 3 1 1 1 1 1 1 2 2 2 3 3 3 2 2 2 3 3 3 1 1 1 3 3 3 1 1 1 2 2 2
1 2 2 2 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3 1 1 1 3 3 3 1 1 1 2 2 2 1 1 1 2 2 2 3 3 3
1 2 2 2 2 2 2 3 3 3 1 1 1 3 3 3 1 1 1 2 2 2 1 1 1 2 2 2 3 3 3 2 2 2 3 3 3 1 1 1
1 2 2 2 3 3 3 1 1 1 2 2 2 1 1 1 2 2 2 3 3 3 3 3 3 1 1 1 2 2 2 2 2 2 3 3 3 1 1 1
1 2 2 2 3 3 3 1 1 1 2 2 2 2 2 2 3 3 3 1 1 1 1 1 1 2 2 2 3 3 3 3 3 3 1 1 1 2 2 2
1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 2 2 2 3 3 3 1 1 1 1 1 1 2 2 2 3 3 3
1 3 3 3 1 1 1 3 3 3 2 2 2 1 1 1 3 3 3 2 2 2 1 1 1 3 3 3 2 2 2 1 1 1 3 3 3 2 2 2
1 3 3 3 1 1 1 3 3 3 2 2 2 2 2 2 1 1 1 3 3 3 2 2 2 1 1 1 3 3 3 2 2 2 1 1 1 3 3 3
1 3 3 3 1 1 1 3 3 3 2 2 2 3 3 3 2 2 2 1 1 1 3 3 3 2 2 2 1 1 1 3 3 3 2 2 2 1 1 1
1 3 3 3 2 2 2 1 1 1 3 3 3 1 1 1 3 3 3 2 2 2 2 2 2 1 1 1 3 3 3 3 3 3 2 2 2 1 1 1
1 3 3 3 2 2 2 1 1 1 3 3 3 2 2 2 1 1 1 3 3 3 3 3 3 2 2 2 1 1 1 1 1 1 3 3 3 2 2 2
1 3 3 3 2 2 2 1 1 1 3 3 3 3 3 3 2 2 2 1 1 1 1 1 1 3 3 3 2 2 2 2 2 2 1 1 1 3 3 3
1 3 3 3 3 3 3 2 2 2 1 1 1 1 1 1 3 3 3 2 2 2 3 3 3 2 2 2 1 1 1 2 2 2 1 1 1 3 3 3
1 3 3 3 3 3 3 2 2 2 1 1 1 2 2 2 1 1 1 3 3 3 1 1 1 3 3 3 2 2 2 3 3 3 2 2 2 1 1 1
1 3 3 3 3 3 3 2 2 2 1 1 1 3 3 3 2 2 2 1 1 1 2 2 2 1 1 1 3 3 3 1 1 1 3 3 3 2 2 2
2 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
2 1 2 3 1 2 3 1 2 3 1 2 3 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
2 1 2 3 1 2 3 1 2 3 1 2 3 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
2 1 2 3 2 3 1 2 3 1 2 3 1 1 2 3 1 2 3 1 2 3 2 3 1 2 3 1 2 3 1 3 1 2 3 1 2 3 1 2
2 1 2 3 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 3 1 2 3 1 2 3 1 2 1 2 3 1 2 3 1 2 3

2 1 2 3 2 3 1 2 3 1 2 3 1 3 1 2 3 1 2 3 1 2 1 2 3 1 2 3 1 2 3 2 3 1 2 3 1 2 3 1
2 1 2 3 3 1 2 3 1 2 3 1 2 1 2 3 1 2 3 1 2 3 3 1 2 3 1 2 3 1 2 2 3 1 2 3 1 2 3 1
2 1 2 3 3 1 2 3 1 2 3 1 2 2 3 1 2 3 1 2 3 1 1 2 3 1 2 3 1 2 3 3 1 2 3 1 2 3 1 2
2 1 2 3 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 2 3 1 2 3 1 2 3 1 1 2 3 1 2 3 1 2 3
2 2 3 1 1 2 3 2 3 1 3 1 2 1 2 3 2 3 1 3 1 2 1 2 3 2 3 1 3 1 2 1 2 3 2 3 1 3 1 2
2 2 3 1 1 2 3 2 3 1 3 1 2 2 3 1 3 1 2 1 2 3 2 3 1 3 1 2 1 2 3 2 3 1 3 1 2 1 2 3
2 2 3 1 1 2 3 2 3 1 3 1 2 3 1 2 1 2 3 2 3 1 3 1 2 1 2 3 2 3 1 3 1 2 1 2 3 2 3 1
2 2 3 1 2 3 1 3 1 2 1 2 3 1 2 3 2 3 1 3 1 2 2 3 1 3 1 2 1 2 3 3 1 2 1 2 3 2 3 1
2 2 3 1 2 3 1 3 1 2 1 2 3 2 3 1 3 1 2 1 2 3 3 1 2 1 2 3 2 3 1 1 2 3 2 3 1 3 1 2
2 2 3 1 2 3 1 3 1 2 1 2 3 3 1 2 1 2 3 2 3 1 1 2 3 2 3 1 3 1 2 2 3 1 3 1 2 1 2 3
2 2 3 1 3 1 2 1 2 3 2 3 1 1 2 3 2 3 1 3 1 2 3 1 2 1 2 3 2 3 1 2 3 1 3 1 2 1 2 3
2 2 3 1 3 1 2 1 2 3 2 3 1 2 3 1 3 1 2 1 2 3 1 2 3 2 3 1 3 1 2 3 1 2 1 2 3 2 3 1
2 2 3 1 3 1 2 1 2 3 2 3 1 3 1 2 1 2 3 2 3 1 2 3 1 3 1 2 1 2 3 1 2 3 2 3 1 3 1 2
2 3 1 2 1 2 3 3 1 2 2 3 1 1 2 3 3 1 2 2 3 1 1 2 3 3 1 2 2 3 1 1 2 3 3 1 2 2 3 1
2 3 1 2 1 2 3 3 1 2 2 3 1 2 3 1 1 2 3 3 1 2 2 3 1 1 2 3 3 1 2 2 3 1 1 2 3 3 1 2
2 3 1 2 1 2 3 3 1 2 2 3 1 3 1 2 2 3 1 1 2 3 3 1 2 2 3 1 1 2 3 3 1 2 2 3 1 1 2 3
2 3 1 2 2 3 1 1 2 3 3 1 2 1 2 3 3 1 2 2 3 1 2 3 1 1 2 3 3 1 2 3 1 2 2 3 1 1 2 3
2 3 1 2 2 3 1 1 2 3 3 1 2 2 3 1 1 2 3 3 1 2 3 1 2 2 3 1 1 2 3 1 2 3 3 1 2 2 3 1
2 3 1 2 2 3 1 1 2 3 3 1 2 3 1 2 2 3 1 1 2 3 1 2 3 3 1 2 2 3 1 2 3 1 1 2 3 3 1 2
2 3 1 2 3 1 2 2 3 1 1 2 3 1 2 3 3 1 2 2 3 1 3 1 2 2 3 1 1 2 3 2 3 1 1 2 3 3 1 2
2 3 1 2 3 1 2 2 3 1 1 2 3 2 3 1 1 2 3 3 1 2 1 2 3 3 1 2 2 3 1 3 1 2 2 3 1 1 2 3
2 3 1 2 3 1 2 2 3 1 1 2 3 3 1 2 2 3 1 1 2 3 2 3 1 1 2 3 3 1 2 1 2 3 3 1 2 2 3 1
3 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2
3 1 3 2 1 3 2 1 3 2 1 3 2 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3
3 1 3 2 1 3 2 1 3 2 1 3 2 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1
3 1 3 2 2 1 3 2 1 3 2 1 3 1 3 2 1 3 2 1 3 2 2 1 3 2 1 3 2 1 3 3 2 1 3 2 1 3 2 1
3 1 3 2 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 3 2 1 3 2 1 3 2 1 1 3 2 1 3 2 1 3 2
3 1 3 2 2 1 3 2 1 3 2 1 3 3 2 1 3 2 1 3 2 1 3 2 1 1 3 2 1 3 2 1 3 2 2 1 3 2 1 3
3 1 3 2 3 2 1 3 2 1 3 2 1 1 3 2 1 3 2 1 3 2 3 2 1 3 2 1 3 2 1 2 1 3 2 1 3 2 1 3
3 1 3 2 3 2 1 3 2 1 3 2 1 2 1 3 2 1 3 2 1 3 1 3 2 1 3 2 1 3 2 3 2 1 3 2 1 3 2 1
3 1 3 2 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 2 1 3 2 1 3 2 1 3 2
3 2 1 3 1 3 2 2 1 3 3 2 1 1 3 2 2 1 3 3 2 1 1 3 2 2 1 3 3 2 1 1 3 2 2 1 3 3 2 1
3 2 1 3 1 3 2 2 1 3 3 2 1 2 1 3 3 2 1 1 3 2 2 1 3 3 2 1 1 3 2 2 1 3 3 2 1 1 3 2
3 2 1 3 1 3 2 2 1 3 3 2 1 3 2 1 1 3 2 2 1 3 3 2 1 1 3 2 2 1 3 3 2 1 1 3 2 2 1 3
3 2 1 3 2 1 3 3 2 1 1 3 2 1 3 2 2 1 3 3 2 1 2 1 3 3 2 1 1 3 2 3 2 1 1 3 2 2 1 3
3 2 1 3 2 1 3 3 2 1 1 3 2 2 1 3 3 2 1 1 3 2 3 2 1 1 3 2 2 1 3 1 3 2 2 1 3 3 2 1
3 2 1 3 2 1 3 3 2 1 1 3 2 3 2 1 1 3 2 2 1 3 1 3 2 2 1 3 3 2 1 2 1 3 3 2 1 1 3 2
3 2 1 3 3 2 1 1 3 2 2 1 3 1 3 2 2 1 3 3 2 1 3 2 1 1 3 2 2 1 3 2 1 3 3 2 1 1 3 2
3 2 1 3 3 2 1 1 3 2 2 1 3 2 1 3 2 2 1 3 3 2 1 1 3 2 2 1 3 3 2 1 1 3 2 2 1 3 2 1 3
3 2 1 3 3 2 1 1 3 2 2 1 3 3 2 1 1 3 2 2 1 3 3 2 1 1 3 2 2 1 3 3 2 1 1 3 2 2 1 3

3 2 1 3 3 2 1 1 3 2 2 1 3 3 2 1 1 3 2 2 1 3 2 1 3 3 2 1 1 3 2 1 3 2 2 1 3 3 2 1

3 3 2 1 1 3 2 3 2 1 2 1 3 1 3 2 3 2 1 2 1 3 1 3 2 3 2 1 2 1 3 1 3 2 3 2 1 2 1 3

3 3 2 1 1 3 2 3 2 1 2 1 3 2 1 3 1 3 2 3 2 1 2 1 3 1 3 2 3 2 1 2 1 3 1 3 2 3 2 1

3 3 2 1 1 3 2 3 2 1 2 1 3 3 2 1 2 1 3 1 3 2 3 2 1 2 1 3 1 3 2 3 2 1 2 1 3 1 3 2

3 3 2 1 2 1 3 1 3 2 3 2 1 1 3 2 3 2 1 2 1 3 2 1 3 1 3 2 3 2 1 3 2 1 2 1 3 1 3 2

3 3 2 1 2 1 3 1 3 2 3 2 1 2 1 3 1 3 2 3 2 1 3 2 1 2 1 3 1 3 2 1 3 2 3 2 1 2 1 3

3 3 2 1 2 1 3 1 3 2 3 2 1 3 2 1 2 1 3 1 3 2 1 3 2 3 2 1 2 1 3 2 1 3 1 3 2 3 2 1

3 3 2 1 3 2 1 2 1 3 1 3 2 1 3 2 3 2 1 2 1 3 3 2 1 2 1 3 1 3 2 2 1 3 1 3 2 3 2 1

3 3 2 1 3 2 1 2 1 3 1 3 2 2 1 3 1 3 2 3 2 1 1 3 2 3 2 1 2 1 3 3 2 1 2 1 3 1 3 2

3 3 2 1 3 2 1 2 1 3 1 3 2 3 2 1 2 1 3 1 3 2 2 1 3 1 3 2 3 2 1 1 3 2 3 2 1 2 1 3

Annexe B

Évolution des accélérateurs de calcul

B.1 GPU

Fin 2016, la société NVIDIA a sorti la nouvelle architecture PASCAL pour ses GPUs. Cette nouvelle architecture présente une bande passante mémoire plus élevée (jusqu'à 1TB/s) et peut effectuer deux fois plus d'opération flottantes simple précision par seconde que les précédentes architectures Maxwell (pour un même nombre de coeur CUDA). De plus cette nouvelle architecture benificie aussi de plus de mémoire et de plus d'unité de calcul à précision mixte (simple et double).
Afin de réduire au maximum les latences introduites par le transfert des données entre le CPU et le GPU via le port PCI-Express, les nouvelles cartes avec l'architecture Pascal sont aussi dotées de la technologie NVLink. Cette nouvelle technologie NVLink permet de faire transiter les données entre le CPU et le GPU entre cinq et douze fois plus vite qu'avec les ports PCI-Express actuels (PCI-E 3.0) grâce à une bande passante plus importante [54].
Enfin ces architectures Pacal bénificient de la mémoire stack HBM Gen2, qui est plus dense au niveau du chip et permet d'augmenter la mémoire HBM sur la carte de 16 jusqu'à 32 GB, avec une bande passante jusqu'à 1TB/s.

B.2 Xeon Phi

La société Intel est en train de mettre en service la nouvelle génération de produits Intel Xeon Phi, les Knights Landing. Le coeur de ces nouvelles architectures est basé sur des coeurs Silvermont Atom lourdement modifié. Chaque coeur sera équipé de deux unités vectorielles AVX-512. Le processeur Knights Landing est ainsi trois fois plus performant (pour un seul thread) que les anciens coeurs Pentium 54C utilisés dans les Knights

Corner. Cette nouvelle architecture est annoncée pour 3 teraflops double précision. Les Knights Landing seront disponibles sous deux versions: sous forme de coprocesseurs classiques se branchant sur le CPU via le port PCI-E (comme les précédents Knights Corner), et sous la forme de CPU standard.
Enfin Intel a déjà annoncé préparer la troisième génération de Xeon Phi le Knight Hill, qui sera basé sur la seconde génération d'architecture Intel Omni-Path [33]

Annexe C

Conditions initiales des simulations de dérive descente enveloppe

Position initiale:

- Altitude = 4500.0 m
- Latitude = 4.24°
- Longitude = 45.5°
- Vitesse verticale = 0 $m.s^{-1}$

TABLE C.1: Profil météorologique

Pression hPa	Altitude m	Température K	Vent U $m.s^{-1}$	Vent V $m.s^{-1}$
1000	131	299.7	3	3
925	812	293	1	1
850	1538	289.3	0	0
700	3168	282	5	5
500	5873	267	4	4
400	7587	257.4	3	3
300	9680	239.9	2	2
250	10933	229.7	6	6
200	12393	217.5	8	8
150	14164	204.3	9	9
100	16524	195.7	1	1
70	18600	203.1	1	1
50	20645	212.2	1	1
30	23881	218.7	2	2
20	26490	222.1	0	0
10	31062	227.8	8	8
7	33437	227.3	7	7
5	35703	235.2	1	1
3	39315	246.5	2	2
2	42291	256.9	3	3
1	47691	269.3	0	0

TABLE C.2: Table d'évolution du coefficient balistique de l'enveloppe

Temps s	Coefficient balistique
-1	1000
100	100
200	1
300	0.3
400	0.3
500	0.3
600	0.5
700	0.5
800	0.5
900	0.5
1000	0.1
1200	0.1
1400	0.1
1600	0.3
1800	0.3
2000	0.3
2500	0.001
3000	0.001
4000	0.001

Annexe D

Résultats des réanalyses des vols de la campagne Timmins 2013

D.1 Vol 2

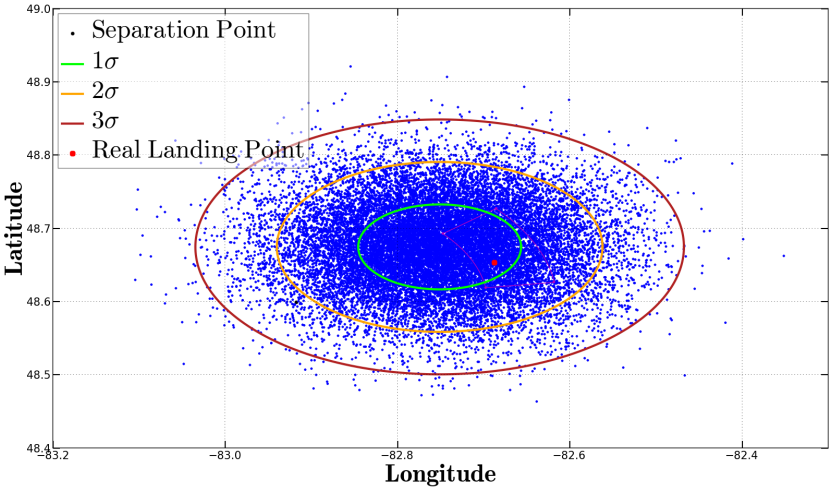

FIGURE D.1: Zones de retombée à 1,2 et 3 sigma pour le vol 2

D.2 Vol 3

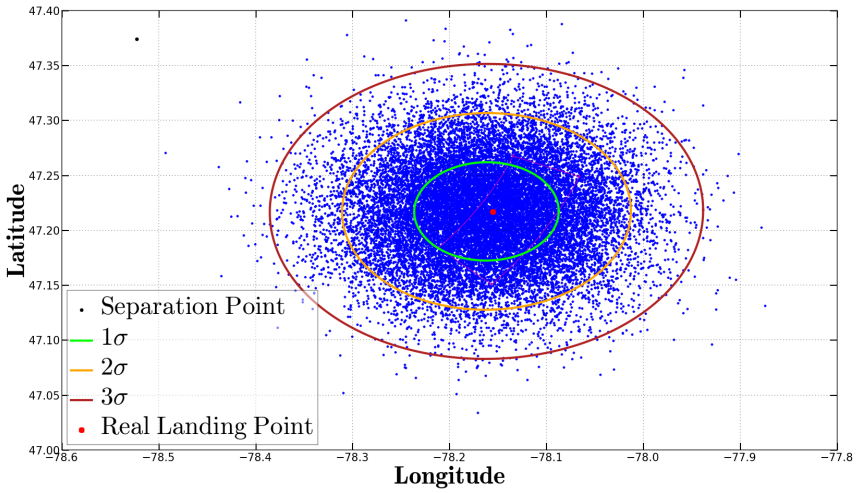

FIGURE D.2: Zones de retombée à 1,2 et 3 sigma pour le vol 3

D.3 Vol 5

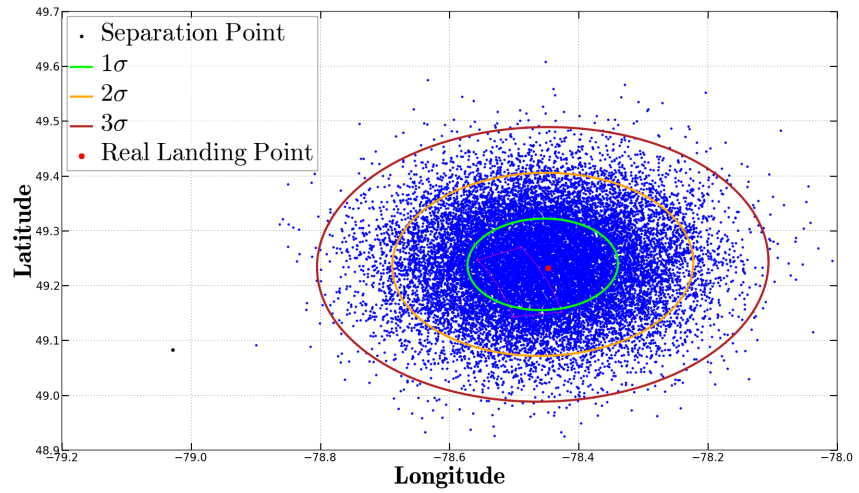

FIGURE D.3: Zones de retombée à 1,2 et 3 sigma pour le vol 5

D.4 Vol 6

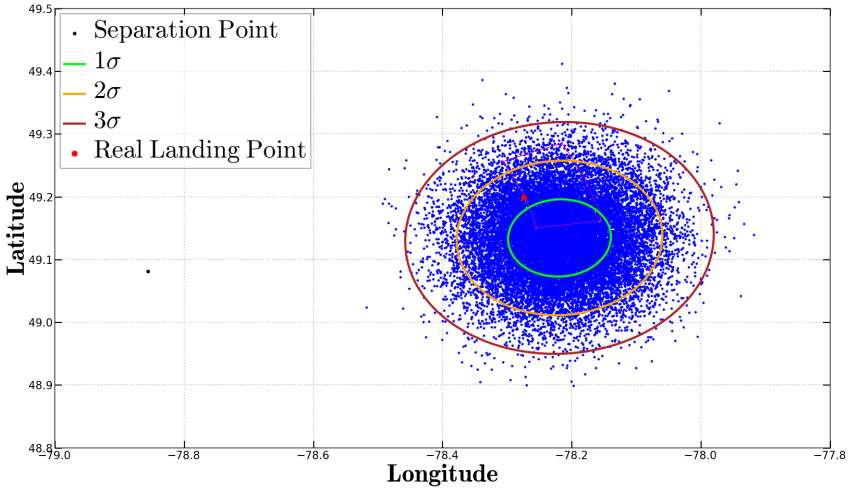

FIGURE D.4: Zones de retombée à 1,2 et 3 sigma pour le vol 6

D.5 Vol 7

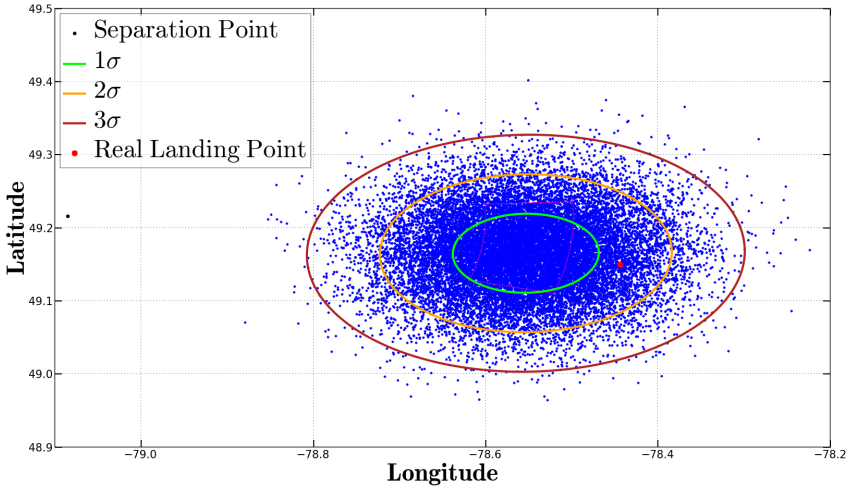

FIGURE D.5: Zones de retombée à 1,2 et 3 sigma pour le vol 7

Annexe E

Formulaire Calima

Le présent annexe regroupe les définitions de certaines grandeurs physiques principales utilisées dans le corps du manuscrit ainsi que certaines des équations implémentées dans Calima, mais non détaillées dans le corps du texte par soucis de lisibilité.

E.1 Référentiels

Afin de décrire mathématiquement le mouvement d'un objet durant une rentrée atmosphérique il est nécessaire d'utiliser plusieurs référentiels que nous allons passer en revue.

Référentiel géocentré Référenciel d'origine le centre de gravité de la Terre et dont les axes sont orientés en direction d'étoile lointaine et supposées fixes.

Référentiel Terrestre (ERF ou R0) Référentiel en rotation uniforme autour du référentiel géocentré, ses axes sont fixes sur la surface de la Terre et l'axe z est suivant la verticale locale.

Référentiel Géodésique (R2) Référentiel lié à l'objet. Il est orienté de la même façon que le référentiel Terrestre

Référentiel de la trajectoire de vol (R4) Référentiel lié à l'objet et dont l'axe y est aligné sur le vecteur vitesse de l'objet.

Référentiel de l'objet (SRF ou R7) Référentiel lié à l'objet, dans lequel la géométrie de l'objet est définie.

Référentiel Aéro (R5) Référentiel lié à l'objet, et définit par deux rotations (β,α) à partir de SRF. Le premier axe de ce repère est aligné avec le vecteur vitesse de l'objet.

E.2 Quaternions

Les quaternions sont des objets mathématiques à 4 éléments aux propriétés mathématiques nombreuses. Entre autre chose, ils peuvent être utilisés pour définir l'orientation d'un solide et peuvent être combinés pour effectuer plusieurs rotations tout en éliminiant les singularités qui peuvent apparaitre lorsque ce travail est effectué avec les angles d'Euler. Ainsi les quaternions sont utilisés dans Calima afin d'effectuer les différents changement de référentiels. Soit $Q = (q0, q1, q2, q3)$ un quaternion permetant de passer d'un référenciel d'origine $\mathcal{R}$ vers un autre $\mathcal{R}'$. On obtient alors la matrice de passage P :

$$P = \begin{pmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & 2(q_0^2 + q_3^2) - 1 \end{pmatrix} \tag{E.1}$$

E.3 Liste des grandeurs physiques principales

Dans cette section nous allons recencer les principales grandeurs physiques qui interviennent dans la simulation de la rentrée atmosphérique de satellite, en donnant pour chacune d'entre elle sa définition.

E.3.1 Longueur de référence

La longueur de référence L_{ref}, est une longueur considérée comme représentative pour un objet donné en fonction de son attitude.

E.3.2 Surface de référence

La surface de référence S_{ref}, est une surface définie pour adimensionner la force de traînée.

E.3.3 Surface totale

La surface totale correspond à la surface extérieure de l'objet étudié. Dans Calima cette valeur peut être imposée par l'utilisateur ou être calculée à partir du maillage de l'objet dans le module aeroCoef.

E.3.4 Constante des gaz réel pour l'air

La constante des gaz réel pour l'air est calculée à partir de la constante des gazs parfaits divisée par la masse molaire de l'air qui est considérée comme constante sur l'ensemble de l'atmosphère.

$$R_{air} = \frac{\mathcal{R}}{M_{air}} \tag{E.2}$$

E.3.5 Vitesse du son

La vitesse du son c_s dans l'air est proportionnel à la température de l'air :

$$c_s = \sqrt{\gamma R_{air} T_{air}} \tag{E.3}$$

avec $\gamma = \frac{C_p}{C_v} = 1.4$ dans le cas de l'air

E.3.6 Nombre de Mach

On définit le nombre de Mach comme étant le ratio entre la vitesse d'un objet et la vitesse du son :

$$M = \frac{v}{c_s} \tag{E.4}$$

E.3.7 Libre parcours moyen

Le libre parcours moyen λ défini la distance moyenne parcouru par une molécule dans un gaz entre deux collisions. On définit alors le libre parcours moyen dans l'air λ_{air}, tel que :

$$\lambda_{air} = \frac{\mathcal{R} T_{air}}{\sqrt{2}\pi P_{air} N_A d^2} \tag{E.5}$$

Avec T_{air} et P_{air} la température et la pression de l'atmosphère à une distance suffisament éloigné de l'objet étudié pour ne pas être perturbé le phénomène étudié, d le diamètre moyen de collision de l'air, ici pris à $3.65.10^{-10}m$, et constante sur l'ensemble de l'atmosphère. Cette valeur est issue de [56], et est correcte pour une atmosphère non humide du niveau de la mer et jusqu'à 85km. Au delà, la validité de la valeur décroit avec l'altitude, mais reste correcte.

E.3.8 Nombre de Knudsen

Le nombre de Knudsen est un nombre adimensionné permettant de déterminer le régime d'écoulement d'un fluide. Il est définit comme étant le ratio entre le libre parcours moyen dans l'air et la longueur de référence de l'objet étudié.

$$K_n = \frac{\lambda_{air}}{L_{ref}} \tag{E.6}$$

E.3.9 Énergie cinétique

On définit l'énergie cinétique E d'un objet à partir de sa masse m et de sa vitess v :

$$E = \frac{1}{2} m v^2 \tag{E.7}$$

E.4 Module aéroCoef

Dans cette section nous allons détailler certaines des fonctions employées dans le module aeroCoef.

E.4.1 Coefficient de pression maximum : C_pmax

Le coefficient de pression maximum C_pmax est calculé en fonction du nombre de Mach : Si $M \sup 1.0$:

$$C_pmax = \frac{2 * ([\frac{(\gamma+1)^2 M^2}{4\gamma M^2} - 2(\gamma-1)]^{\frac{\gamma}{\gamma-1}} * \frac{(1-\gamma)+2\gamma M^2}{\gamma+1}) - 1}{\gamma M^2} \tag{E.8}$$

Sinon $C_pmax = 1$

E.4.2 Coefficient de pression : C_p

Dans le module aeroCoef de Calima le calcul du coefficient de pression dépend du régime d'écoulement :

E.4.2.1 Régime continu

On considère que le régime est continu pour un nombre de Knudsen inférieur à 0.001.

Dans ce cas on calcule le coefficient de pression par la méthode de Newton modifiée [5] :

$$C_p = C_pmax * \cos^2(\theta) \tag{E.9}$$

Avec θ l'angle entre la normale à la surface d'une cellule du maillage orienté vers l'extérieur de la structure et la direction de l'écoulement.

E.4.2.2 Régime raréfié

On considère que le régime est raréfié pour un nombre de Knudsen supérieur à 100.

Dans ce cas on calcule le coefficient de pression via : Si $\cos(\theta) ! = 0$: Soit ϵ le ratio de la température spécifique (pris par défaut à 0.1). Soit : $T_r = (\epsilon * T_{air}) + [(1-\epsilon) * T_{paroi}]$ avec T_{paroi} la température de la paroi de l'objet. On définit aussi P_r le ratio entre la pression infini et la pression autour de l'objet :

$$P_r = (1+\epsilon)\frac{s\cos(\theta)}{\sqrt{\pi}} + [\frac{1}{2}(1-\epsilon)(\frac{T_r}{T_{air}})^{1/2}\exp(-s^2\cos^2(\theta))](1-\epsilon)(0.5+s^2\cos^2(\theta)) + [0.5(1-\epsilon)(\frac{T_r}{T_{air}})^{-1/2}\sqrt{\pi}s\cos\theta][1+erf(\frac{M}{\sqrt{\gamma/2}}\cos\theta] \tag{E.10}$$

On a calcule alors C_p :

$$C_p = \frac{P_r - 1}{\frac{M^2}{\gamma/2.0}} \tag{E.11}$$

E.4.2.3 Régime transitoire

On définit le régime transitoire le régime pour un nombre de Knudsen compris entre 0.1 et 100.

En régime transitoire on calcule la valeur du coefficient de pression à partir d'une interpolation entre sa valeur dans le régime continue $C_{p,continue}$ calculé à partir de E.9 et celle dans le millieu raréfié $C_{p,raréfié}$ calculé par E.11, selon la fonction d'interpolation suivante issue d'ORSAT [83] :

$$C_p = C_{p,continue} + (C_{p,continue} - C_{p,raréfié}) * \sin[\pi(0.5 + 0.25 \log Kn)]^3 \tag{E.12}$$

E.4.3 Calcul de C_H : Aérothermodynamique

E.4.3.1 Calcul des rayons de courbure locaux

Afin de pouvoir calculer les flux de chaleurs qui s'exercent sur chaque cellule du maillage de l'objet étudié, il faut au préalable calculer pour chaque cellule le rayon de courbure local. Pour ce faire nous nous basons sur la bibliothèque libigl [22] qui permet de calculer pour chaque point du maillage la courbure moyenne en ce point κ. On calcule ainsi le rayon de courbure R_C en ce point via :

$$R_C = \frac{1}{\kappa} \tag{E.13}$$

Une fois ce calcul effectué pour chaque point du maillage on calcule ensuite le rayon de courbure de la cellule R_T pour une cellule en moyennant les rayons de courbure de chacun de ses sommets :

$$R_T = \frac{R_{C\ P1} + R_{C\ P2} + R_{C\ P3}}{3} \tag{E.14}$$

Avec P1, P2 et P3 les trois sommets de la cellule.

On lisse ensuite ce résultat en identifiant toutes les cellules i se situant dans un rayon donné autour de la cellule que l'on considère. Soit le ryon de courbure lissé R_m :

$$R_m = \frac{\sum_i R_{T,i}}{\sum_i S_{celllule,i}} \tag{E.15}$$

Avec $R_{T,i}$ le rayon de courbure de chaque cellule contenue dans le rayon donné et $S_{celllule,i}$ la surface de chaque cellule.

E.4.3.2 Flux de chaleur

Le calcul du flux de chaleur s'exerçant sur chaque cellule est différent en fonction du régime d'écoulement.

Régime continue On définit le régime continue pour un nombre de Knudsen inférieur à 0.01. Dans ce régime on calcule le flux de chaleur sur une cellule à partir de la formule de Filippis-Serpico [30] : Soit $p1$:

$$p1 = P_{air}(1 + \frac{\gamma - 1}{2}M^2)^{\frac{\gamma}{\gamma-1}}) \tag{E.16}$$

Et soit α tel que :

$$\alpha = \frac{C_{p,face}}{C_p max}, \tag{E.17}$$

On définit ensuite $p2$ tel que : Si $p1 \sup 1$:

$$p2 = p1\frac{((\gamma+1)M^2}{(\gamma-1)M^2+2})^{\frac{\gamma}{\gamma-1}}(\frac{\gamma+1}{2\gamma M^2-(\gamma-1)}\frac{1}{\gamma-1}) * \alpha^{0.9} \tag{E.18}$$

Sinon $p2 = p1$. On définit alors le flux de chaleur q_{FS} :

$$q_{FS} = 2.75e^{-5}\sqrt{\frac{p2}{R_{eq}}}(h_0 - h_w)^{1.17} \tag{E.19}$$

Avec h_0 l'enthalpy libre au point d'arrêt et h_w l'enthalpy libre à la paroi.

Régime raréfié On définit le régime raréfié pour un nombre de Knudsen supérieur à 10. Il est donné par la loi de Bird [11] :

$$\begin{aligned} q_{raréfié} = \frac{\rho_{air}}{2*\beta^3} * [\frac{1-\epsilon}{2\pi^{1/2}} * [s^2 + \frac{\gamma}{\gamma-1} - (0.5 * \frac{\gamma+1}{\gamma-1})\frac{T_r}{T_{air}}] * [exp(-s^2 * sin^2\theta) + \\ \pi^{1/2} ssin\theta[1 + erf(ssin\theta)]] - 0.5exp(-s^2 sin^2\theta] \end{aligned} \tag{E.20}$$

Régime transitoire On définit le régime transitoire le régime pour un nombre de Knudsen compris entre 0.01 et 10. On utilise la formule de transition utilisée par SCARAB [83] :

$$q_{transitionnel} = \frac{q_{FS}}{\sqrt{1 + (\frac{q_{FS}}{q_{raéfié}})^2}} \tag{E.21}$$

E.5 Aérodynamique

E.5.1 Modèle d'atmosphère

Les données atmosphériques sont recalculées à chaque pas de temps à partir du modèle d'atmosphère standard [56].

Liste des publications

Plazolles Bastien, Spel Martin, Rivola Vincent, El Baz Didier. Monte-Carlo Analysis of Object Reentry in Earth's Atmosphere Based on Taguchi Method, *Conférence : 8th European Symposium on Aerothermodynamics for Space Vehicles, Mar 2015, Lisbonne, Portugal. 2015*
lien vers l'article.

Spel Martin, Rivola Vincent, Plazolles Bastien. First Spacecraft Demise Workshop - Test Case Description and Results, *Conférence : 8th European Symposium on Aerothermodynamics for Space Vehicles, March 2015, Lisbonne, Portugal. 2015*
lien vers l'article.

Plazolles Bastien, El Baz Didier, Spel Martin, Rivola Vincent, Gegout Pascal. Parallel Monte-Carlo Simulations on GPU and Xeon Phi for Stratospheric Balloon Envelope Drift Descent Analysis, *p 611-619, Conference: 16th IEEE International Conference on Scalable Computing and Communications (ScalCom 2016), At Toulouse France. 18-21 July 2016*
Cet article a reçu le prix du Best Paper de la conférence IEEE ScalCom 2016
lien vers l'article.

Plazolles Bastien, El Baz Didier, Spel Martin, Rivola Vincent, Gegout Pascal. SIMD Monte-Carlo Simulations accelerated on GPU and Xeon Phi, *Soumis à International Journal of Parallel Programming*

Bibliographie

[1] OpenMP Application Program Interface. [Online]. Available: http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf

[2] *A Debris Risk Assessment Tool Supporting Mitigation Guidelines, Proceedings of the 4th European Conference on Space Debris (ESA SP-587)*, April 2005.

[3] *Tecplot 360, Data Format Guide*, 2016.

[4] AMD. AMD Launches 'Boltzmann Initiative' to Dramatically Reduce Barriers to GPU Computing on AMD FirePro Graphics. [Online]. Available: http://www.amd.com/en-us/press-release/Pages/boltzmann-initiative-2015nov16.aspx

[5] J. Anderson, *Hypersonic and High Temperature Gas Dynamics*, ser. McGraw-Hill series in aeronautical and aerospace engineering. American Institute of Aeronautics and Astronautics, 2000. [Online]. Available: https://books.google.fr/books?id=G0u7e71QBakC

[6] J. Annaloro, P. Omaly, V. Rivola, and M. Spel, "Elaboration of a New Spacecraft-Oriented Tool: PAMPERO," in *8th European Symposium on Aerothermodynamics for Space Vehicles*, ser. ESA Special Publication, 2015.

[7] M. Balat-Pichelin and P. Omaly, "Study of the Atmospheric Entry of Metallic Space Debris - Oxidation and Emissivity Evaluation to Contribute to "Design for Demise"," in *Proceedings of the 8th European Symposium on Aerothermodynamics for Space Vehicle, Lisbon*, 2015.

[8] J. Beck, J. Merrifield, I. Holbrough, G. Markelov, and R. Molina, "Application of the SAM Destructive re-entry Code to the Spacecraft Demise Thermodynamics and Integration Test Cases," in *8th European Symposium on Aerothermodynamics for Space Vehicles*, ser. ESA Special Publication, 2015.

[9] T. D. Bess, "Mass Distribution of Orbiting Man-Made Space Debris," NASA, Tech. Rep., 1975.

[10] G. A. Bird, *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*. Oxford Engineering Science Series, 1994.

[11] ——, *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*, 2nd ed., ser. Oxford Engineering Science Series. Oxford University Press, USA, 1994.

[12] G. E. P. Box and D. W. Behnken, "Some New Three Level Designs for the Study of Quantitative Variables," *Technometrics*, vol. 2, no. 4, pp. 455–475, 1960.

[13] V. Boyer and D. El Baz, "Recent Advances on GPU Computing in Operations Research," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, May 2013, pp. 1778–1787.

[14] D. R. Brooks, T. D. Bess, and G. G. Gibson, "Predicting the Probability that Earth-Orbiting Spacecraft will Collide with Man-Made Objects in Space," *International Astronautical Federation, International Astronautical Congress, 25th, Amsterdam, Netherlands*, 1974.

[15] R. E. Caflish, "Monte Carlo and quasi-Monte Carlo Methods," *Acto Numerica*, pp. 1–49, 1998.

[16] G. C. Christian P. Robert, *Introducing Monte Carlo Methods with R*. Springer, 2004.

[17] ——, *Monte Carlo Statistical Methods*. Springer, 2004.

[18] S. Collange, J. Flórez, and D. Defour, "A GPU interval library based on Boost.Interval," in *8th Conference on Real Numbers and Computers*, Santiago de Compostela, Spain, Jul. 2008, pp. 61–71. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00263670

[19] A. Cosnuau, "2014 International Conference on Computational Science Computation on GPU of Eigenvalues and Eigenvectors of a Large Number of Small Hermitian Matrices," *Procedia Computer Science*, vol. 29, pp. 800 – 810, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S187705091400249X

[20] L. Cropp, *ANALYTICAL METHODS USED IN PREDICTING THE RE-ENTRY ABLATION OF SPHERICAL AND CYLINDRICAL BODIES*, Sep 1965.

[21] D. Danesy, Ed., *Scarab -a Multi-Disciplinary Code for Destruction Analysis of Space-Craft during Re-Entry*, ser. ESA Special Publication, vol. 563, Feb. 2005.

[22] P. Daniele and J. Alec. libigl. [Online]. Available: libigl.github.io/libigl/tutorial/tutorial.html

[23] R. W. Detra and H. Hidalgo, "Generalized Heat Transfer Formulas and Graphs for Nose Cone Re-Entry Into the Atmosphere," *ARS Journal*, 1961. [Online]. Available: http://arc.aiaa.org/doi/abs/10.2514/8.5471

[24] S. Dindar, E. B. Ford, M. Juric, Y. I. Yeo, J. Gao, A. C. Boley, B. Nelson, and J. Peters, "Swarm-NG: A CUDA library for Parallel n-body Integrations with focus on simulations of planetary systems," *New Astronomy*, vol. 23–24, pp. 6 – 18, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1384107613000092

[25] J. Dobarco-Otero, N. Smith, K. J. Bledsoe, R. M. Delaune, W. C. Rochelle, and N. L. Johnson, "The Object Reentry Survival Analysis Tool (ORSAT)-Version 6.0 and its Application to Spacecraft Entry," *AIAA Journal*, 1998. [Online]. Available: http://dx.doi.org/10.2514/6.IAC-05-B6.3.06

[26] R. Dolbeau, S. Bihan, F. Bodin, and C. Entreprise, "1 HMPP^{TM}: A Hybrid Multi-core Parallel Programming Environment."

[27] A. Falsone, F. Noce, and M. Prandini, "A Randomized Approach to Space Debris Footprint Characterization," *IFAC World Congress 2014*, 2014.

[28] R. Farber. Programming Intel's Xeon Phi: A Jumpstart Introduction. [Online]. Available: http://www.drdobbs.com/parallel/programming-intels-xeon-phi-a-jumpstart/240144160

[29] H. Faure, "Discrépances de suites associées à un système de numération (en dimension un)," *Bulletin de la Société Mathématique de France*, vol. 109, pp. 143–182, 1981. [Online]. Available: http://eudml.org/doc/87390

[30] F. D. Filippis and M. Serpico, "Air High-Enthalpy Stagnation Point Heat Flux Calculation," *Journal of Thermophysics and Heat Transfer*, vol. 12, no. 4, November 1998.

[31] F.Mistree, U. Lautenschlager, S. Erikstad, and J. Allen, "Simulation Reduction using Taguchi Method," NASA, Tech. Rep., 1993.

[32] G. C. Fox, R. D. Williams, and P. C. Messina, *Parallel Computing Works!*, B. M. Spatz and D. Sery, Eds. Morgan Kaufmann, 1994.

[33] E. Gardner. What public disclosure has Intel made about Knights Landing? [Online]. Available: https://software.intel.com/en-us/articles/what-disclosures-has-intel-made-about-knights-landing

[34] J. H. Halton, "Algorithm 247: Radical-inverse Quasi-random Point Sequence," *Commun. ACM*, vol. 7, no. 12, pp. 701–702, Dec. 1964. [Online]. Available: http://doi.acm.org/10.1145/355588.365104

[35] W. E. Hoover and U. States., *Algorithms for Confidence Circles and Ellipses [microform]*. U.S. Dept. of Commerce, National Oceanic and Atmospheric Administration, National Ocean Service Rockville, Md, 1984.

[36] K. Hwang, G. C. Fox, and J. Dongarra, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[37] J. Jeffers and J. Reinders, *Intel Xeon Phi Coprocessor High-Performance Programming*, ser. Morgan Kaufmann. Elsevier Science & Technology Books, 2013.

[38] A. Karsten and M. Mario. odeint. [Online]. Available: http://headmyshoulder.github.io/odeint-v2/

[39] A. Kato, Y. Hyodo, and K. Kishida, "Toward the world common re-entry safety assessment procedure," in *Twenty-Third International Symposium on Space Technology and Science*, 2003, pp. 2366–2371.

[40] F. R. Kemp and N. H. Riddell, "Heat Transfer to Satellite Vehicles Re-entering the Atmosphere," *Journal of Jet Propulsion*, 1957. [Online]. Available: http://arc.aiaa.org/doi/abs/10.2514/8.12603

[41] D. J. Kessler and B. G. Cour-Palais, "Collision Frequency of Artificial Satellites: The Creation of a Debris Belt," *Journal of Geophysical Research: Space Physics*, vol. 83, no. A6, pp. 2637–2646, 1978. [Online]. Available: http://dx.doi.org/10.1029/JA083iA06p02637

[42] R. Klett, *DRAG COEFFICIENTS AND HEATING RATIOS FOR RIGHT CIRCULAR CYLINDERS IN FREE- MOLECULAR AND CONTINUUM FLOW FROM MACH 10 TO 30*. Sandia Corporation, Dec 1964. [Online]. Available: http://www.osti.gov/scitech/servlets/purl/4630398

[43] M. Lalami and D. El-Baz, "GPU Implementation of the Branch and Bound method for knapsack problems," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops and PhD Forum*, 2012.

[44] M. R. Laurance and D. E. Brownlee, "The flux of meteoroids and orbital space debris striking satellites in low Earth orbit," *Nature*, vol. 323, pp. 136–138, 1986. [Online]. Available: http://dx.doi.org/10.1038/323136a0

[45] N. Metropolis, "The Monte Carlo Method," *Journal of the American Statistical Association*, pp. 3Trott O.35–341, 1949.

[46] ——, "The Beginning of the Monte Carlo Method," *Los Alamos Science*, pp. 125–130, 1987.

[47] R. Misawa, J.-P. Bernard, P. Ade, Y. André, P. de Bernardis, M. Bouzit, M. Charra, B. Crane, J. P. Dubois, C. Engel, M. Griffin, P. Hargrave, B. Leriche, Y. Longval, S. Maes, C. Marty, W. Marty, S. Masi, B. Mot, J. Narbonne, F. Pajot, G. Pisano, N. Ponthieu, I. Ristorcelli, L. Rodriguez, G. Roudil, M. Salatino, G. Savini, and C. Tucker, "Pilot: a balloon-borne experiment to measure the polarized fir emission of dust grains in the interstellar medium," in *Millimeter, Submillimeter, and Far-Infrared Detectors and Instrumentation for Astronomy VII*, vol. 9153, jul 2014, p. 91531H.

[48] NVIDIA. cuRAND. [Online]. Available: http://docs.nvidia.com/cuda/curand/

[49] ——. Nvidia. CUDA 7.0. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[50] ——. Nvidia. CUDA 7.0 Programming Guide. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[51] ——. Optix^{TM} technology. [Online]. Available: https://developer.nvidia.com/optix-technology

[52] ——. Parallel Thread Execution ISA Version 4.3. [Online]. Available: http://docs.nvidia.com/cuda/parallel-thread-execution/

[53] ——. Profiler user's guide. [Online]. Available: http://docs.nvidia.com/cuda/profiler-users-guide/#nvprof-overview

[54] ——. www.nvidia.com.

[55] P. Omaly and M. Spel, "DEBRISK, a Tool for Re-Entry Risk Analysis," in *A Safer Space for Safer World*, ser. ESA Special Publication, vol. 699, Jan. 2012, p. 70.

[56] U. S. C. on Extension to the Standard Atmosphere, *U.S. standard atmosphere, 1976*. National Oceanic and Amospheric [sic] Administration : for sale by the Supt. of Docs., U.S. Govt. Print. Off., 1976. [Online]. Available: https://books.google.fr/books?id=x488AAAAIAAJ

[57] OpenACC. OpenACC Directives for Accelerators. [Online]. Available: http://www.openacc.org/

[58] J. N. Opiela, E. Hillary, D. O. Whitlock, and M. Hennigan, *Debris Assessment Software User's Guide Version 2.0*, NASA, January 2012.

[59] C. Parigini, I. Pontijas Fuentes, R. Haya Ramos, D. Bonetti, and S. Cornara, "DEBRIS Tool and its Use in Mission Analysis Activities," in *8th European Symposium on Aerothermodynamics for Space Vehicles*, ser. ESA Special Publication, 2015.

[60] S. G. Parker, H. Friedrich, D. Luebke, K. Morley, J. Bigler, J. Hoberock, D. McAllister, A. Robison, A. Dietrich, G. Humphreys, M. McGuire, and M. Stich, "GPU Ray Tracing," *Communications of the ACM*, vol. 56, no. 5, March 2013. [Online]. Available: http://cacm.acm.org/magazines/2013/5/163758-gpu-ray-tracing/fulltext

[61] Phadke, *Quality Engineering Using Robust Design*. P T R Prentice Hall, 1989.

[62] R. L. Plackett and J. P. Burman, "The Design of Optimum Multifactorial Experiments," *Biometrika*, vol. 33, no. 4, pp. 305–325, 1946. [Online]. Available: http://www.jstor.org/stable/2332195

[63] B. Plazolles, D. El Baz, M. Spel, V. Rivola, and P. Gegout, "Parallel Monte-Carlo Simulations on GPU and Xeon Phi for Stratospheric Balloon Envelope Drift Descent Analysis," in *International IEEE Conference on Scalable Computing and Communications*, 2016, pp. 611–619.

[64] B. Plazolles, M. Spel, V. Rivola, and D. El Baz, "Monte-Carlo analysis of Object Reentry in Earth s Atmosphere Based on Taguchi Method," in *Proceedings of the 8th European Symposium on Aerothermodynamics for Space Vehicle, Lisbon*, 2015.

[65] B. Plazolles, M. Spel, and V. Rivola, "Spacecraft Demise: Test Case Definitions." [Online]. Available: http://scdw.rtech.fr/SCDW_WEB/UK/INT-DFD-BKLT-131220-1086-RTECH.pdf?AWPIDCB82F898=61D89367C8B1D2A2DF5C92A690E46E41A4A42856

[66] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press, 1992.

[67] F. Rabier, A. Bouchard, E. Brun, A. Doerenbecher, S. Guedj, V. Guidard, F. Karbou, V.-H. Peuch, L. E. Amraoui, D. Puech, C. Genthon, G. Picard, M. Town, A. Hertzog, F. Vial, P. Cocquerez, S. A. Cohn, T. Hock, J. Fox, H. Cole, D. Parsons, J. Powers, K. Romberg, J. VanAndel, T. Deshler, J. Mercer, J. S. Haase, L. Avallone, L. Kalnajs, C. R. Mechoso, A. Tangborn, A. Pellegrini, Y. Frenot, J.-N. Thépaut, A. McNally, G. Balsamo, and P. Steinle, "The concordiasi project in antarctica," *Bulletin of the American Meteorological Society*, vol. 91, no. 1, pp. 69–86, 2010. [Online]. Available: http://dx.doi.org/10.1175/2009BAMS2764.1

[68] R. Rahman, *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*, 1st ed. Berkely, CA, USA: Apress, 2013.

[69] J.-B. Renard, G. Berthet, V. Salazar, V. Catoire, M. Tagger, B. Gaubicher, and C. Robert, "In situ detection of aerosol layers in the middle stratosphere," *Geophysical Research Letters*, vol. 37, no. 20, pp. n/a–n/a, 2010, l20803. [Online]. Available: http://dx.doi.org/10.1029/2010GL044307

[70] A. T. C. Robert V. Hogg, *Introduction to Mathematical Statistics*, 4th ed. Macmillan USA, 1978. [Online]. Available: http://gen.lib.rus.ec/book/index.php?md5=9A1E3297F90A13A43A43F1BC12258AE3

[71] M. B. L. Rocchi, D. Sisti, M. Ditroilo, and R. P. A. Calavalle, "The Misuse of the Confidence Ellipse in Evaluating Statokinesigram," *ITALIAN JOURNAL OF SPORT SCIENCES*, vol. 12, no. 2, pp. 169–171, 2005. [Online]. Available: http://hdl.handle.net/11576/2504321

[72] J. Rogers and N. Slegers, "Robust Parafoil Terminal Guidance Using Massively Parallel Processing," *AIAA Atmospheric Flight Mechanics Conference*, 2013. [Online]. Available: http://dx.doi.org/10.2514/6.2012-4736

[73] P. Ross, *Taguchi techniques for quality engineering: loss function, orthogonal experiments, parameter and tolerance design*, ser. Library of Congress Cataloging-Publication Data. McGraw-Hill, 1988. [Online]. Available: https://books.google.fr/books?id=NOhTAAAAMAAJ

[74] R. Roy, *A Primer on the Taguchi Method*. Society of Manufacturing Engineers, 1990.

[75] E. Saule, K. Kaya, and Ü. V. Çatalyürek, "Performance Evaluation of Sparse Matrix Multiplication Kernels on Intel Xeon Phi," *CoRR*, vol. abs/1302.1078, 2013. [Online]. Available: http://arxiv.org/abs/1302.1078

[76] L. S. Schramm, D. S. McKay, H. A. Zook, and G. A. Robinson, "N.a," in *Proc lunar planet Sci Conf*, vol. 16, 1985, pp. 434–435.

[77] I. Sobol, "On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 86 – 112, 1967. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0041555367901449

[78] R. R. Sokal and F. J. Rohlf, *Biometry The Principles and Practices of Statistics in Biological Research*. W. H. Freeman, 1995.

[79] M. Spel, V. Rivola, and B. Plazolles, "First Spacecraft Demise Workshop - Test Case Description and Results," in *Proceedings of the 8th European Symposium on Aerothermodynamics for Space Vehicle, Lisbon*, 2015.

[80] G. Taguchi, *System of experimental design: engineering methods to optimize quality and minimize costs*, ser. System of Experimental Design: Engineering Methods to Optimize Quality and Minimize Costs. UNIPUB/Kraus International Publications, 1987, no. vol. 1. [Online]. Available: https://books.google.fr/books?id=-PVQAAAAMAAJ

[81] G. Taguchi, S. Chowdhury, and Y. Wu, *Taguchi's Quality Engineering Handbook*. Wiley, 2005. [Online]. Available: https://books.google.fr/books?id=zc4mAQAAMAAJ

[82] G. Teodoro, T. Kurc, J. Kong, L. Cooper, and J. Saltz, "Comparative Performance Analysis of Intel (R) Xeon Phi (TM), GPU, and CPU: A Case Study from Microscopy Image Analysis," in *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, ser. IPDPS '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 1063–1072. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2014.111

[83] B. F. Tobias Lips, "A Comparison of Commonly used Re-Entry Analysis Tools," *Acta Astronautica*, vol. 57 (2005), p. 312 – 323, 21 April 2005.

[84] O. Trott and A. J. Olson, "Software News and Update AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading," *Journal of Computational Chemistry*, vol. 31(2), pp. 455–461, January 2010.

[85] A. ul Hasan Khan, M. Al-Mouhamed, and L. Firdaus, "Evaluation of Global Synchronization for Iterative Algebra Algorithms on Many-Core," in *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015 16th IEEE/ACIS International Conference on*, June 2015, pp. 1–6.

[86] N. X. Vinh, A. Busemann, and R. D. Culp, "Hypersonic and planetary entry flight mechanics," *NASA STI/Recon Technical Report A*, vol. 81, 1980.

[87] C. Weiland, *Computational Space Flight Mechanics*. Springer Berlin Heidelberg, 2010. [Online]. Available: https://books.google.fr/books?id=ouIUPqnqXZAC

[88] Z. Wu, R. Hu, X. Qu, X. Wang, and Z. Wu, "Space debris reentry analysis methods and tools," *Chinese Journal of Aeronautics*, vol. 24, no. 4, pp. 387 – 395, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1000936111600460

[89] H. A. Zook, "A preliminary report on the study of the impact sites and particles of the solar maximum satellite thermal blanket," in *NASA. Goddard Space Flight Center Proceedings of the SMRM Degradation Study Workshop*, 1985, pp. 247–264.

Plate-forme de calcul parallèle "Design for Demise"

Résumé :

Les risques liés aux débris spatiaux sont à présent considérés comme critiques par les gouvernements et les agences spatiales internationales. Durant la dernière décénie les agences spatiales ont développé des logiciels pour simuler la rentrée atmosphérique des satellites et des stations orbitales afin de déterminer les risques et possibles dommages au sol. Néanmoins tous les outils actuels fournissent des résultats déterministes alors que les modèles employés utilisent des valeurs de paramètres qui sont mal connues. De plus les résultats obtenus dépendent fortement des hypothèses qui sont faites. Une solution pour obtenir des résultats pertinents et exploitables est de prendre en considération les incertitudes que l'on a sur les différents paramètres de la modélisation afin d'effectuer des analyses de type Monte-Carlo. Mais une telle étude est particulièrement gourmande en temps de calcul à cause du grand espace des paramètres à explorer (ce qui nécessite des centaines de milliers de simulations numériques). Dans le cadre de ces travaux de thèse nous proposons un nouveau logiciel de simulation numérique de rentrée atmosphérique de satellite, permettant de façon native de prendre en considération les incertitudes sur les différents paramètres de modélisations pour effectuer des analyses statistiques. Afin de maitriser les temps de calculs cet outil tire avantage de la méthode de Taguchi pour réduire le nombre de paramètres à étudier et aussi des accélérateurs de calculs de type Graphics Processing Units (GPU) et Intel Xeon Phi.

Mots clés : débris spatiaux, calcul parallèle, GPU, CUDA, Xeon Phi, CPU multi-coeur, OpenMP, Monte-Carlo, Taguchi, intégrateur numérique, ballon stratosphérique, dérive descente

Parallel computing platform "Design for Demise"

Abstract :

The risk of space debris is now perceived as primordial by governments and international space agencies. Since the last decade, international space agencies have developed tools to simulate the re-entry of satellites and orbital stations in order to assess casualty risk on the ground. Nevertheless, all current tools provide deterministic solutions, though models include various parameters that are not well known. Therefore, the provided results are strongly dependent on the assumptions made. One solution to obtain relevant and exploitable results is to include uncertainties around those parameters in order to perform Monte-Carlo analysis. But such a study is very time consuming due to the large parameter space to explore (that necessitate hundreds of thousands simulations). As part of this thesis work we propose a new satellite atmospheric reentry simulation tool that permits one to natively consider uncertainty on models parameters in order to perform statistical analysis. To master computing time this tool takes advantage of Taguchi method to restrain the amount of parameter to study and also takes advantage of computing accelerators like Graphics Processing Units (GPU) and Intel Xeon Phi.

Keywords : space debris, parallel computing, GPU, CUDA, Xeon Phi, multi-core CPU, OpenMP, Monte-Carlo, Taguchi, numerical integrator, stratospheric baloon, drift descent

Université Toulouse III Paul Sabatier

Laboratoire d'Analyse et d'Architecture des Systèmes

R.Tech