HAL
open science

Couplage de modèles population et individu-centrés pour la simulation parallélisée des systèmes biologiques : application à la coagulation du sang

Laurent Crépin

To cite this version:

Laurent Crépin. Couplage de modèles population et individu-centrés pour la simulation parallélisée des systèmes biologiques : application à la coagulation du sang. Bio-informatique [q-bio.QM]. Université de Bretagne occidentale - Brest, 2013. Français. NNT : 2013BRES0051 . tel-00880516v2

HAL Id: tel-00880516

https://theses.hal.science/tel-00880516v2

Submitted on 9 Sep 2014

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

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

université de bretagne occidentale

THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE
sous le sceau de l'Université Européenne de Bretagne

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE
Mention : Informatique
École Doctorale SICMA

présentée par
Laurent Crépin
préparée au Centre Européen de Réalité Virtuelle,
Laboratoire CNRS Lab-STICC

Couplage de modèles population et individu-centrés pour la simulation parallélisée des systèmes biologiques. Application à la coagulation du sang.

Thèse soutenue le 28 octobre 2013
devant le jury composé de :

Patrick Amar (Rapporteur)
Maître de Conférence, Université de Paris Sud

Éric Ramat (Rapporteur)
Professeur, Université du Littoral Côte d'Opale

Coen Hemker (Examinateur)
Professeur émérite, Université de Maastricht

Vincent Rodin (Examinateur)
Professeur, Université de Bretagne Occidentale

Jacques Tisseau (Directeur de thèse)
Professeur, ENI Brest

Fabrice Harrouet (Encadrant)
Maître de Conférence, ENI Brest

Pascal Redou (Encadrant)
Maître de Conférence, ENI Brest

Sébastien Kerdélo (Invité)
Responsable de service R&D, Diagnostica Stago

Université Européenne de Bretagne

— Thèse de doctorat —

Spécialité : Informatique

Couplage de modèles population et individu-centrés pour la simulation parallélisée des systèmes biologiques

Application à la coagulation du sang

Laurent Crépin

Soutenue le 28 octobre 2013 devant le jury :

Rapporteurs

Patrick	Amar	Maître de Conférence, Université de Paris Sud, Orsay
Éric	Ramat	Professeur, Université du Littoral Côte d'Opale, Calais

Examinateurs

Coen	Hemker	Professeur émérite, Université de Maastricht, Pays-Bas
Vincent	Rodin	Professeur, Université de Bretagne Occidentale, Brest

Directeur de thèse

Jacques	Tisseau	Professeur, ENI Brest

Encadrants

Fabrice	Harrouet	Maître de Conférence, ENI Brest
Pascal	Redou	Maître de Conférence, ENI Brest

Invité

Sébastien	Kerdélo	Responsable de service R&D, Diagnostica Stago, Gennevilliers

Couplage de modèles population et individu-centrés pour la simulation parallélisée des systèmes biologiques

Application à la coagulation du sang

Thèse de doctorat

Laurent Crépin

— 28 octobre 2013 —

Laboratoire en Sciences et Techniques de l'Information, de la Communication et de la Connaissance
UMR 6285 (CNRS, UEB, ENIB)

Laurent Crépin
Tel : +33 (0)2 98 05 89 56
Mail : crepin@enib.fr

Centre Européen de Réalité Virtuelle

25 rue Claude Chappe
BP 38
F-29280 Plouzané
Tel : +33 (0)2 98 05 89 89
Fax : +33 (0)2 98 05 89 79
Mail : contact@cerv.fr
Site internet : http://www.cerv.fr

Diagnostica Stago

9 rue des Frères Chausson
F-92600 Asnières sur Seine
Tel : +33 (0)1 46 88 20 20
Fax : +33 (0)1 47 91 08 91
Mail : stago@stago.fr
Site internet : http://www.stago.fr

Remerciements

Je tiens à remercier ici l'ensemble des personnes qui ont contribué, de près ou de loin, à la réalisation et à la réussite de ces travaux de thèse.

Tout d'abord, j'adresse mes remerciements à Patrick Amar et Eric Ramat pour avoir rapporté cette thèse, ainsi que pour les remarques pertinentes qu'ils ont formulées à l'égard de mes travaux. Je remercie également Vincent Rodin d'avoir présidé mon jury de thèse et Coen Hemker pour avoir accepté de participer à ce jury.

Je remercie également Jacques Tisseau pour avoir dirigé cette thèse et pour m'avoir promulgué de nombreux conseils lors de la préparation de ma soutenance. J'adresse des remerciements particuliers à mes encadrants, Fabrice Harrouet, Sébastien Kerdélo et Pascal Redou, pour leur bonne humeur, leur investissement, ainsi que leur souci du détail sans lequel le contenu de ce mémoire n'aurait pas été aussi complet.

Je souhaite ensuite remercier la société Diagnostica Stago ainsi que l'Association Nationale de la Recherche et de la Technologie (ANRT) pour avoir financé mes travaux de thèse durant trois ans, ainsi que Muriel Gonidec pour les excellentes conditions de travail qu'elle a pu m'offrir. Je remercie également les sociétés Cervval et Synapse sans qui une partie de ces travaux n'aurait pu voir le jour.

Je remercie également l'ensemble des membres du CERV pour la bonne humeur générale qui y règne. Merci à Pit, Julien, Yohann, Charlotte, Marie, Alex, Steevy, Fabien, Mihai, Mukesh, Seb, Gireg, Marc, Manu, Camille, Goby, Ronan, Bidou, Céline, Elisabetta, Nathalie, Alexis, Cyril, Éric, les Pierre et tous les autres. Merci à Nicole sans qui la vie de thésard serait beaucoup moins confortable.

Je n'oublie pas mes proches, qu'il s'agisse aussi bien de mes amis que de ma famille, et les remercie pour le soutien qu'ils ont su m'apporter avant et pendant la réalisation de ces travaux de thèse. Je remercie tout particulièrement mes parents pour m'avoir permis de réaliser cette thèse, mais également les talents culinaires du Minou et de ses commis. Un grand merci à François pour m'avoir « dégagé du temps » (bien que cela lui soit difficile...) et permis d'oublier le travail de temps à autre.

Enfin, j'adresse un remerciement tout particulier à ma femme, Aurélie, qui m'a supporté (au sens propre comme au figuré) et encouragé lors des différentes étapes de cette thèse. Merci d'avoir accepté mes absences répétées et le rythme de vie que je t'ai fait mener durant ces trois dernières années. Encore merci.

Avant-propos

Ce document synthétise les travaux de thèse que j'ai effectués durant trois ans dans le cadre d'une Convention Industrielle de Formation par la REcherche (CIFRE) dont les partenaires sont le Laboratoire en Sciences et Techniques de l'Information, de la Communication et de la Connaissance (Lab-STICC), la société Diagnostica Stago, ainsi que l'Association Nationale de la Recherche et de la Technologie (ANRT). J'ai réalisé ces travaux au sein du Centre Européen de Réalité Virtuelle (CERV), un centre de recherche académique situé près de Brest, dépendant de l'École Nationale d'Ingénieurs de Brest (ENIB), et regroupant des chercheurs de divers horizons tels que l'informatique, les mathématiques ou encore la psychologie.

Ces travaux, réalisés sous la direction du Professeur Jacques Tisseau, ont été encadrés par Fabrice Harrouet, Sébastien Kerdélo et Pascal Redou. Ils s'inscrivent dans le cadre des recherches effectuées au sein de l'équipe pluridisciplinaire Interaction Humain Système et Environnement Virtuel (IHSEV), dont l'objectif est d'étudier les interactions entre les humains et les systèmes artificiels, et plus particulièrement dans le cadre de la simulation des systèmes biologiques complexes.

La réalisation de ces travaux a abouti à diverses présentations et démonstrations au cours de conférences internationales dans le domaine de la bioinformatique et de la biologie. Nous les présenterons dans ce mémoire au moment opportun. En outre, par l'intermédiaire de ces travaux de thèse, j'ai pu participer à une collaboration scientifique avec l'équipe du Professeur Coen Hemker de la société Synapse, située à Maastricht aux Pays-Bas. Certaines des idées proposées dans ce mémoire tirent leur inspiration de cette collaboration très constructive.

Table des matières

Remerciements **v**

Avant-propos **vii**

Table des matières **ix**

Liste des figures **xi**

Liste des tableaux **xiii**

Liste des notations **xv**

Introduction **1**

1 Simulation parallélisée des systèmes biologiques **5**
1.1 Modélisation et simulation des systèmes biologiques 5
1.1.1 Approches population-centrées 6
1.1.2 Approches individu-centrées 10
1.1.3 Approches mixtes 12
1.2 Solutions de parallélisation et de distribution 15
1.2.1 Architectures multicœur et multiprocesseur 16
1.2.2 Architectures graphiques 24
1.2.3 Architectures distribuées 36
1.3 Bilan de notre étude 40
1.3.1 Intérêt d'une approche mixte 40
1.3.2 Intérêt de la parallélisation face à la distribution 41
1.3.3 Parallélisation d'approches mixtes 42

2 Modélisation et parallélisation mixtes **45**
2.1 Modèle individu-centré asynchrone 45
2.1.1 Description de nos entités autonomes 46
2.1.2 Schéma d'ordonnancement 46

2.1.3 Ordonnancement parallèle 50
2.1.4 Bilan de notre modèle individu-centré 59
2.2 Modèles population-centrés synchrones 60
2.2.1 Maillage de l'espace 60
2.2.2 Méthodes de parallélisation 62
2.2.3 Bilan de nos modèles population-centrés 68
2.3 Couplage de modèles population et individu-centrés 69
2.3.1 Déroulement d'une simulation 69
2.3.2 Interaction entre les modèles 70
2.4 Bilan de notre proposition 73

3 Exemples d'application à la coagulation du sang 75
3.1 Cinétique biochimique à l'échelle microscopique 75
3.1.1 La cinétique biochimique 76
3.1.2 Modélisation individu-centrée de la cinétique biochimique 77
3.1.3 Validation et résultats 82
3.1.4 Discussion et perspectives 84
3.2 Vaisseau sanguin virtuel 86
3.2.1 Notions élémentaires d'hématologie 86
3.2.2 Modélisation mixte d'un vaisseau sanguin 89
3.2.3 Évaluation des performances 102
3.2.4 Potentiel applicatif 108
3.2.5 Discussion et perspectives 118
3.3 Bilan de nos exemples d'application 119

Conclusion 121

A Posters 125

B Données physiologiques du vaisseau sanguin virtuel 129
B.1 Caractéristiques du vaisseau et de l'écoulement sanguin 129
B.1.1 Dimensions du problème 129
B.1.2 Pression intravasculaire 130
B.1.3 Vitesse de l'écoulement 130
B.1.4 Viscosité et densité plasmatique 131
B.1.5 Bilan 132
B.2 Caractéristiques des éléments figurés 132
B.3 Paramètres de simulation 133

Références bibliographiques 135

Résumé 149

Liste des figures

1 Méthodes d'expérimentation des systèmes biologiques 2

1.1 Illustration du « jeu de la vie » 9
1.2 Exemple d'architecture mémoire actuelle 18
1.3 Addition de deux vecteurs avec les *threads* POSIX 21
1.4 Addition de deux vecteurs avec OpenMP 22
1.5 Modèle matériel d'un GPU 26
1.6 Hiérarchie des *threads* sur GPU 27
1.7 Masquage de latences par changement de contexte 29
1.8 Branchement conditionnel sur CPU et GPU 29
1.9 Addition de deux vecteurs sur GPU 33
1.10 Exemple d'architecture distribuée 36

2.1 Exemple de modélisation synchrone 49
2.2 Exemple de modélisation asynchrone 49
2.3 Ordonnancement asynchrone et chaotique 50
2.4 Déroulement d'une simulation 52
2.5 Exemple de vol de tâches hiérarchisé 54
2.6 Exemple de situation de *deadlock* 56
2.7 Résolution d'une situation de *deadlock* 58
2.8 Exemple de situation de *livelock* 59
2.9 Déroulement d'une simulation mono-GPU 63
2.10 Distribution d'un maillage entre deux GPUs 64
2.11 Déroulement d'une simulation multi-GPUs 66
2.12 Déroulement d'une simulation population-centrée multi-CPUs et multi-GPUs 68
2.13 Déroulement d'une simulation individu et population-centrée 70
2.14 Fonctionnement des tableaux de variations 73
2.15 Déroulement complet d'une simulation individu et population-centrée 74

3.1 Diagramme de classe UML de notre modèle 78
3.2 Cycle de vie d'une entité 79
3.3 Validation des réactions réversible et enzymatique 83
3.4 Évaluation des performances 84

3.5 Illustration de notre simulateur de cinétique biochimique 85
3.6 La balance hémostatique 88
3.7 Représentation du thrombus 88
3.8 Cascade simplifiée de la coagulation 90
3.9 Géométrie du vaisseau sanguin virtuel 91
3.10 Représentation 3D du maillage du vaisseau sanguin virtuel 91
3.11 Écoulement de Couette 94
3.12 Phénomènes présents au sein du vaisseau sanguin virtuel 99
3.13 Impact du nombre de cellules sur les performances 107
3.14 Parallélisation du vaisseau sanguin virtuel 108
3.15 Fenêtre de sélection des patients 113
3.16 Fenêtre de contrôle (onglet génération de thrombine) 114
3.17 Fenêtre de contrôle (onglet dynamique du caillot) 115
3.18 Fenêtre de rendu des simulations 116

A.1 Poster ECCB 2012 126
A.2 Poster ISTH 2013 127

B.1 Écoulement de Poiseuille 131

Liste des tableaux

3.1 Modélisation et simulation de la cinétique biochimique 77
3.2 Paramètres des simulations de validation 83
3.3 Évaluation des performances 104
3.4 Impact des transferts de frontière sur les performances 105
3.5 Impact des barrières de synchronisation sur les performances 106

B.1 Caractéristiques du vaisseau et de l'écoulement sanguin 132
B.2 Caractéristiques des éléments figurés 132

Liste des notations

Unités

Å	Ångström
g	Gramme
Hz	Hertz
m	Mètre
M	Molaire
mol	Mole
o	Octet
Pa	Pascal
P	Poise
s	Seconde

Constante

N_A	Nombre d'Avogadro, soit $6.02214129 \times 10^{23}$ mol^{-1}

Acronymes

CERV	Centre Européen de Réalité Virtuelle
CNRS	Centre National de la Recherche Scientifique
CPU	*Central Processing Unit*
CUDA	*Compute Unified Device Architecture*

EDO	équation différentielle ordinaire
EDP	équation aux dérivées partielles
EDS	équation différentielle stochastique
ENI	École Nationale d'Ingénieurs
ENIB	École Nationale d'Ingénieurs de Brest
ETP	*endogenous thrombin potential*
GPGPU	*General-Purpose computation on Graphics Processing Units*
GPU	*Graphics Processing Unit*
IHM	interface homme-machine
IHSEV	Interaction Humain Système et Environnement Virtuel
Lab-STICC	Laboratoire en Sciences et Techniques de l'Information, de la Communication et de la Connaissance
MPI	*Message Passing Interface*
NUMA	*Non Uniform Memory Access*
OpenCL	*Open Computing Language*
OpenMP	*Open Multi Processing*
PVM	*Parallel Virtual Machine*
R&D	Recherche et Développement
SIMD	*Single Instruction, Multiple Data*
SIMPLE	*Semi-Implicit Method for Pressure-Linked Equation*
SIMT	*Single Instruction, Multiple Thread*
SMT	*Simultaneous MultiThreading*
TCA	temps de céphaline avec activateurs
TP	taux de prothrombine
UEB	Université Européenne de Bretagne
UML	*Unified Modeling Language*
UMR	Unité Mixte de Recherche

Introduction

La coagulation du sang, ou hémostase, est le phénomène physiologique responsable de la formation d'un caillot en cas de brèche vasculaire. L'intérêt de ce caillot est d'arrêter l'hémorragie, nocive pour l'organisme. Ce processus fait intervenir des protéines plasmatiques, facteurs ou inhibiteurs de la coagulation, ainsi que des cellules, endothéliales ou plaquettaires, qui mettent en œuvre des phénomènes d'activation et d'inhibition au niveau de la brèche pour contrôler la formation du caillot sanguin venant l'obstruer. Ces phénomènes mis en jeu par la coagulation du sang comprennent une grande variété et un nombre important de composants et d'interactions qui révèlent une grande complexité. Afin de mieux appréhender de tels phénomènes, il est nécessaire d'étudier cette complexité.

Contexte et problématique

L'étude des systèmes biologiques, complexes par nature, c'est-à-dire faisant intervenir à la fois un grand nombre et une grande diversité d'entités et d'interactions, passe inéluctablement par l'expérimentation. La démarche permettant de réaliser une telle expérimentation nécessite l'élaboration de modèles, c'est-à-dire de représentations simplifiées des phénomènes biologiques réels entrant en jeu. Dès lors, l'expérience consiste à interagir avec ce modèle en testant des hypothèses, notamment en faisant varier ses paramètres : il s'agit de l'activité de simulation. Les résultats produits lors de cette étape de simulation peuvent être extrapolés au phénomène réel afin de mieux le comprendre. Historiquement, les simulations étaient qualifiées d'*in vivo*, c'est-à-dire réalisées sur des organismes vivants. Plus tard, la simulation *in vitro* vit le jour. En proposant des expériences dans des tubes à essais, elle offre un meilleur contrôle des simulations. Plus récemment, l'avènement de l'informatique et la multiplication des ordinateurs ont abouti au développement d'une nouvelle approche : la simulation *in silico*[1]. Cette dernière consiste à simuler des modèles mathématiques ou informatiques numériquement sur ordinateur. Elle permet ainsi de répéter de nombreuses fois une expérience en

1. Néologisme construit par analogie avec les termes latins *in vivo* et *in vitro* ainsi que le terme anglais *silicon* désignant la matière constituant la puce des processeurs informatiques.

contrôlant totalement l'environnement et les paramètres des simulations, et ceci à moindre coût. La figure 1 illustre ces trois méthodes de simulation.

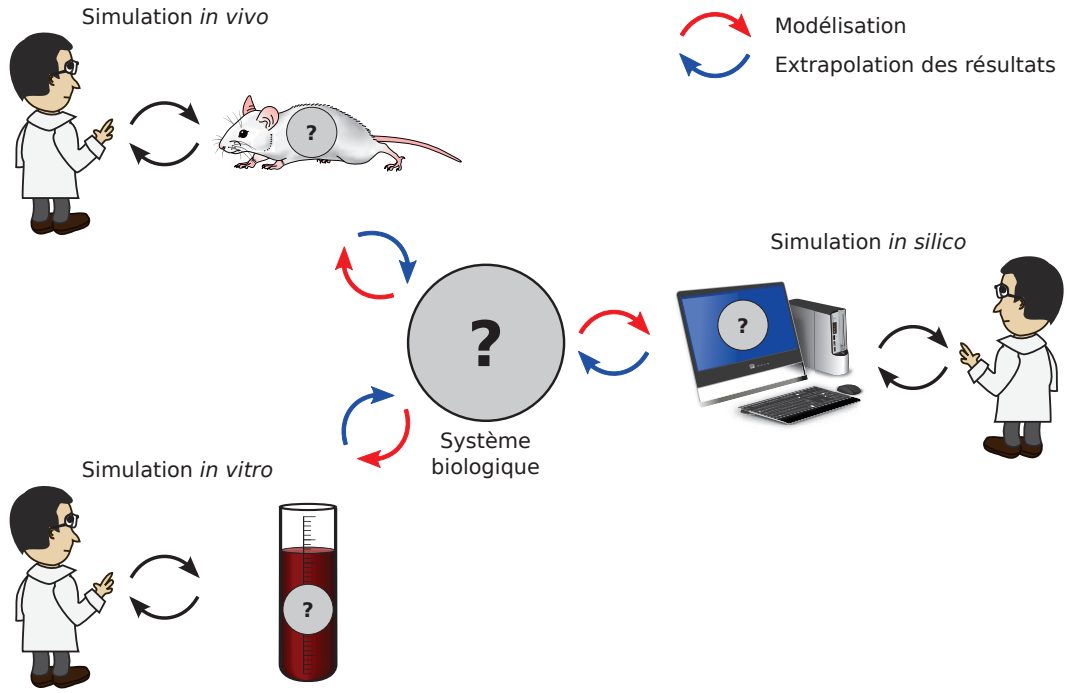


Figure 1 – **Méthodes d'expérimentation des systèmes biologiques** – La démarche expérimentale consiste à modéliser le système biologique étudié, puis à le simuler pour enfin extrapoler les résultats obtenus au système réel. Ce schéma illustre les trois principaux types de simulation : *in vivo*, c'est-à-dire avec un modèle vivant, *in vitro*, avec un tube à essais et *in silico*, avec un modèle numérique sur un ordinateur.

La modélisation des systèmes biologiques peut être obtenue par l'intermédiaire de deux paradigmes distincts : la modélisation population-centrée et la modélisation individu-centrée. La première approche propose d'étudier le système à l'échelle des populations, c'est-à-dire d'un ensemble d'individus. Ainsi, les propriétés pertinentes du système sont modélisées sous la forme de variables caractérisant l'état global du système (par exemple des concentrations chimiques). De telles approches permettent d'obtenir des simulations rapides mais ne sont capables de prendre en compte ni l'hétérogénéité des individus composant les populations, ni les contraintes géométriques pouvant survenir et évoluer dans le système. En outre, elles ne s'appliquent que lorsque les populations étudiées sont de taille suffisante. Pour pallier ces problèmes, la seconde approche propose d'étudier le comportement de chacun de ces individus, plutôt que celui de la population globale. Ainsi, la modélisation individu-centrée permet d'atteindre un niveau de détail très élevé qu'une approche population-centrée ne peut modéliser. Néanmoins, en raison du nombre important d'entités que font intervenir les systèmes biologiques, l'approche individu-centrée peut entraîner des simulations beaucoup plus lentes que celles obtenues à partir de son homologue population-centré. Face à la multitude et à la variété des phénomènes composant les systèmes biologiques, il peut être pertinent de coupler ces deux approches pour obtenir une modélisation mixte, afin que chaque phénomène puisse être modélisé de manière adéquate.

En raison de la quantité conséquente d'informations que représente l'ensemble des composants et des interactions à modéliser, la simulation numérique des systèmes biologiques est particulièrement coûteuse en temps de calcul informatique. Pour réduire ce coût, une solution est d'utiliser des machines de simulation de plus en plus puissantes, c'est-à-dire dont la fréquence de calcul est plus élevée ou dont le nombre d'unités de traitement est plus important. Or, bien que jusqu'en 2004 ou 2005 les fabricants informatiques se soient essentiellement focalisés sur la montée en fréquence des processeurs, cette tendance s'est arrêtée car le rapport entre le coût de fabrication et les performances des processeurs est devenu beaucoup trop élevé. Ainsi, les fabricants se concentrent dorénavant sur l'augmentation du nombre d'unités de calcul au sein d'un même processeur, plutôt que d'en augmenter la fréquence. L'exploitation de l'ensemble de ces unités de calcul se nomme parallélisation et permet de bénéficier pleinement des performances de calcul offertes par les architectures multicœur et multiprocesseur actuelles ainsi que les architectures graphiques récentes.

De nos jours, la plupart des stations de travail informatiques sont munies de ces deux types d'architecture. Elles mettent ainsi une puissance de calcul colossale à disposition des expérimentateurs qui, correctement utilisée, devrait permettre de réaliser des simulations de systèmes biologiques très performantes. Ce constat nous pousse à nous interroger sur la manière d'exploiter de telles architectures parallèles pour simuler le plus efficacement possible les systèmes biologiques modélisés par l'intermédiaire d'approches population et individu-centrées. Ainsi, dans ce mémoire, nous défendons la thèse selon laquelle la simulation de tels systèmes peut bénéficier avantageusement d'un couplage entre des modèles population et individu-centrés parallélisés sur des architectures graphiques ainsi que sur leurs homologues multicœur et multiprocesseur.

Organisation du mémoire

Ce mémoire est constitué de trois chapitres. En premier lieu, il présente un état de l'art des travaux existants dans le domaine de la modélisation et de la simulation parallélisée des systèmes biologiques. Puis il détaille notre proposition répondant à la problématique soulevée précédemment. Enfin des exemples concrets d'application de notre proposition sont présentés. Nous proposons maintenant un résumé détaillé de chacun de ces trois chapitres.

Le **chapitre 1**, intitulé « Simulation parallélisée des systèmes biologiques », propose une revue des différents paradigmes de modélisation et de simulation dont peuvent bénéficier les systèmes biologiques. Cette étude met en évidence deux approches principales, les approches population et individu-centrées, desquelles découle une troisième approche les couplant et que nous qualifions de mixte. Nous discutons de l'intérêt de chacune d'elles et insistons sur le besoin d'utiliser une approche mixte pour étudier les systèmes biologiques. Nous proposons ensuite un état de l'art étudiant les différentes solutions techniques de parallélisation et de distribution permettant de simuler efficacement les systèmes biologiques, qu'ils soient modélisés par l'intermédiaire d'une approche mixte, population ou individu-centrée. Nous concluons cet état de l'art en insistant sur le besoin d'utiliser des approches mixtes parallélisées pour la simulation des systèmes biologiques.

Le **chapitre 2**, intitulé « Modélisation et parallélisation mixtes », s'appuie sur les conclusions de notre état de l'art et a pour objectif de proposer des méthodes informatiques pour la simulation parallélisée de systèmes biologiques modélisés par l'intermédiaire d'approches mixtes. Nous débutons ce chapitre en présentant un modèle individu-centré utilisant un ordonnancement asynchrone et capable d'exploiter au mieux les architectures informatiques multicœur et multiprocesseur. Suite à cela, nous proposons des méthodes permettant la simulation parallélisée de modèles population-centrés synchrones par des architectures graphiques, pouvant néanmoins être épaulées par les architectures multicœur et multiprocesseur utilisées précédemment. Enfin, nous mettons en œuvre un couplage de ces deux approches pour réaliser des simulations mixtes.

Le **chapitre 3**, intitulé « Exemples d'application à la coagulation du sang », constitue la dernière partie de ce mémoire et propose deux applications de nos travaux aux phénomènes liés à la coagulation du sang. La première application est un simulateur de cinétique biochimique[2] à l'échelle microscopique, c'est-à-dire à l'échelle des molécules. Elle a pour objectif d'étudier le comportement des protéines plasmatiques en solution ou fixées sur des membranes physiologiques. La seconde représente l'application phare de nos travaux. Elle consiste à simuler la formation d'un caillot sanguin au sein d'un vaisseau virtuel, dans des conditions physiologiques les plus proches possibles de celles trouvées *in vivo*. Pour chacune de ces deux applications, nous présentons les performances obtenues par les solutions techniques que nous détaillons dans le chapitre précédent.

Ces trois chapitres aboutissent à une conclusion qui synthétise le contenu de ce mémoire. Elle établit également un bilan de notre proposition et énonce des perspectives potentielles concernant de futurs travaux.

L'**annexe A**, intitulée « Posters », présente les deux posters produits lors de ces travaux de thèse.

L'**annexe B**, intitulée « Données physiologiques du vaisseau sanguin virtuel », regroupe quant à elle les données permettant de reproduire les simulations que nous réalisons au cours du chapitre 3.

2. La cinétique biochimique est l'étude de l'évolution des concentrations chimiques au sein d'un système biologique au cours du temps.

Chapitre 1

Simulation parallélisée des systèmes biologiques

Ce travail de thèse propose des méthodes informatiques pour la modélisation et la simulation parallèle de systèmes biologiques. Aussi, dans ce chapitre, nous donnons un état de l'art détaillé des solutions informatiques existantes s'inscrivant dans cette problématique. Il est organisé selon le schéma suivant : tout d'abord, nous effectuons une revue des différentes approches permettant la modélisation et la simulation de phénomènes biologiques. Nous détaillons ensuite les différentes solutions de parallélisation et de distribution à notre disposition pour l'optimisation des performances de calcul des simulations. Nous dressons enfin un bilan de notre étude bibliographique qui introduit les méthodes informatiques que nous proposons dans le cadre de ces travaux de thèse.

1.1 Modélisation et simulation des systèmes biologiques

La simulation des systèmes biologiques que nous considérons peut-être mise en œuvre par l'intermédiaire de deux paradigmes principaux de modélisation : l'approche population-centrée ou l'approche individu-centrée. Le couplage de ces deux paradigmes donne naissance à une troisième approche subsidiaire que nous qualifions de mixte. La littérature abonde de publications qui comparent les approches population et individu-centrées de manière plus ou moins détaillée. Ces comparaisons sont illustrées au travers de différents exemples dans le domaine de la biologie : la biochimie [Amar, 2012; Beurton-Aimar *et al.*, 2011; Andrews *et al.*, 2009], la biologie cellulaire [Byrne et Drasdo, 2009], la biologie animale [Nugala *et al.*, 1998], la microbiologie [Hellweger et Bucci, 2009], l'immunologie [D'Souza *et al.*, 2009] ou encore l'écologie [1] [Abbott *et al.*, 1997].

1. Selon les définitions des termes écologie et biologie du dictionnaire Larousse 2014, l'écologie peut-être assimilée à une science biologique.

Cette section a pour objectif d'introduire les concepts fondamentaux liés aux approches population et individu-centrées, et notamment les avantages et inconvénients que chacune d'elles propose. Elle se poursuit par la présentation des approches mixtes qui couplent les deux approches précédentes pour bénéficier de leurs atouts respectifs, tout en s'affranchissant de leurs défauts, lors de la simulation des systèmes biologiques.

1.1.1 Approches population-centrées

Les approches population-centrées permettent de représenter un système dans sa globalité en modélisant le comportement d'une population d'individus. Dans cette section, nous distinguons deux approches population-centrées. Tout d'abord, nous introduisons la méthode basée sur les équations différentielles. Ensuite, nous présentons les automates cellulaires qui sont des systèmes dont l'évolution est gouvernée par un ensemble de règles prédéfinies.

1.1.1.1 Les équations différentielles

La modélisation par équations différentielles d'un système biologique consiste à exprimer les propriétés pertinentes du système par l'intermédiaire de variables qui caractérisent son état global. L'évolution de ces variables est gouvernée par des modèles mathématiques, déterministes ou stochastiques à base d'équations différentielles, c'est-à-dire liant une ou plusieurs fonctions à leurs dérivées ordinaires ou partielles, ou plus rarement par des systèmes dynamiques discrets. Ces dérivées décrivent les variations que subissent les fonctions lorsque l'un ou plusieurs de leurs paramètres varient. Selon le problème étudié, différentes équations sont utilisées :

Les équations différentielles ordinaires (EDO) permettent de suivre l'évolution d'une fonction par rapport à une variable continue indépendante qui, dans le cadre des systèmes biologiques, est souvent le temps. Si nous considérons une fonction u dont les variations dépendent d'une variable t, alors les EDOs s'écrivent sous la forme :

$$F\left(t, u, \frac{du}{dt}, \frac{d^2u}{dt^2}, \ldots, \frac{d^{n-1}u}{dt^{n-1}}, \frac{d^nu}{dt^n}\right) = 0 \tag{1.1}$$

où F désigne la fonction liant la dérivée ordinaire de u à l'ordre n, c'est-à-dire $\frac{d^nu}{dt^n}$, avec ses dérivées d'ordres inférieurs, la variable t et la fonction u elle-même.

Les équations aux dérivées partielles (EDP) tiennent compte de variables continues supplémentaires (très souvent les coordonnées spatiales) et utilisent donc des dérivées partielles plutôt qu'ordinaires. Si nous considérons une fonction u dépendante de m variables continues, alors les EDPs se formulent comme suit :

$$F\left(x_1, \ldots, x_m, u, \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_m}, \ldots, \frac{\partial^2 u}{\partial x_1^2}, \ldots, \frac{\partial^2 u}{\partial x_1 \partial x_m}, \ldots\right) = 0 \tag{1.2}$$

où F désigne de nouveau la fonction liant l'ensemble des dérivées partielles de u entre elles, ainsi que les m variables, de x_1 à x_m, et la fonction u elle-même.

Les équations différentielles stochastiques (EDS) sont des EDOs ou des EDPs auxquelles vient s'ajouter la notion de stochasticité, c'est-à-dire la capacité à tenir compte des phénomènes aléatoires. Ainsi une EDS possède au minimum un terme désignant un phénomène stochastique (typiquement du bruit blanc). Si nous considérons à nouveau une fonction u dépendante d'une variable t, les EDSs sont définies de la manière suivante :

$$\frac{du}{dt} = F_1(u,t) + F_2(u,t)\xi(t) \tag{1.3}$$

où F_1 et F_2 sont des fonctions continues et $\xi(t)$ représente une fonction aléatoire.

La simulation de tels modèles consiste à résoudre les équations différentielles les définissant. Cette résolution ne peut que très rarement être effectuée analytiquement. En effet, en raison de la complexité des systèmes biologiques, il est généralement impossible d'exprimer explicitement la solution de telles équations avec des fonctions et des opérateurs de référence. Pour pallier ce problème, des procédés de résolution numérique ont vu le jour. Ils consistent à déterminer des approximations numériques de la solution des équations en discrétisant le domaine d'étude en une succession de pas élémentaires. Ces discrétisations permettent de calculer les valeurs approximatives que prend la solution en faisant évoluer pas à pas les variables du système (généralement le temps et l'espace). Bien que plus élevé que celui de leurs homologues analytiques, le coût en temps de calcul de ces méthodes de résolution est relativement minime. Il dépend principalement du pas de discrétisation utilisé qui conditionne le nombre d'itérations à réaliser pour déterminer la solution souhaitée. Plus ce pas est grand, plus le temps de calcul sera court mais plus la méthode utilisée sera susceptible de diverger et de poser des problèmes de stabilité comme la perte du contrôle de l'erreur. À l'inverse, plus ce pas est faible, plus les résultats obtenus seront précis mais les simulations nécessiteront plus de temps. Le lecteur intéressé trouvera une explication détaillée du fonctionnement des méthodes de résolution numérique dans [Stoer *et al.*, 2002] ainsi que [Hairer *et al.*, 2009; Hairer et Wanner, 2010].

Comme l'illustre la littérature, les approches à base d'équations différentielles sont fréquemment utilisées pour modéliser toutes sortes de phénomènes biologiques. Les travaux présentés dans [Andrews *et al.*, 2009] proposent différentes méthodes à base d'EDOs et d'EDPs (déterministes ou probabilistes) visant à modéliser la cinétique biochimique. Un modèle à base d'EDPs est également mis en œuvre dans [Hogea *et al.*, 2006] pour décrire l'évolution de tumeurs, alors que des EDOs sont privilégiées dans [Luo et Rudy, 1994] pour caractériser l'activité électrique au sein du cœur. Par ailleurs, les travaux détaillés dans [Manninen *et al.*, 2006] modélisent la transduction de signal neuronal[2] par l'intermédiaire d'EDSs. Les avantages de ces méthodes sont nombreux et les exemples précédents témoignent de leur large utilisation dans de nombreux domaines liés à la biologie. Nous notons en outre leur forte présence dans d'autres domaines variés tels que la physique [Versteeg et Malalasekera, 2007] ou encore la finance [El Karoui *et al.*, 1997]. En sus de cette méthode de modélisation mathématique, une seconde approche population-centrée est largement utilisée pour modéliser les phénomènes biologiques : il s'agit des automates cellulaires.

2. La transduction de signal décrit le mécanisme par lequel des cellules répondent aux informations qu'elles reçoivent.

1.1.1.2 Les automates cellulaires

Contrairement aux équations différentielles, les automates cellulaires modélisent le domaine étudié de manière discrète. Ils sont ainsi définis par une grille régulière contenant des cellules, chacune possédant un état distinct synthétisant l'état de cette portion d'espace. Ce dernier appartient à un ensemble fini d'états prédéterminés lors de l'élaboration du modèle. Cependant, il en existe des variantes continues, équivalentes à des systèmes différentiels discrétisés dans le temps et dans l'espace [Lenuzza, 2009]. L'évolution de l'état de chacune des cellules au cours du temps est déterminée par un ensemble de règles prédéfinies le liant à l'état des cellules voisines. La simulation d'un automate cellulaire consiste à appliquer ces règles d'évolution de manière répétée sur l'ensemble de ses cellules. Un pas de simulation est terminé lorsque ces règles ont été appliquées successivement à chacune des cellules. Classiquement, les états de toutes les cases sont modifiés simultanément : la totalité de la grille est mise à jour de manière synchrone. Néanmoins, certains travaux proposent des automates cellulaires asynchrones où les cellules évoluent tour à tour plutôt que simultanément [Schönfisch et de Roos, 1999]. L'invention des automates cellulaires est accordée à John von Neumann (ainsi qu'à Arthur Walter Burks qui compléta son travail) qui étudiait les systèmes capables de s'auto-répliquer [Von Neumann et Burks, 1966]. Il mit ainsi au point un automate cellulaire possédant 29 états et une règle de transition prenant la forme d'une machine à états finis. Cette règle permettait à une cellule de changer d'état en fonction d'un voisinage de von Neumann, c'est-à-dire de son état actuel et de celui de ses quatre voisines (verticalement et horizontalement). Il montra finalement qu'une certaine configuration de son automate parvenait à se répliquer.

L'exemple le plus célèbre d'automate cellulaire est celui du « jeu de la vie » développé par John Conway dont une explication détaillée est proposée dans [Gardner, 1970]. Il consiste en une grille en deux dimensions composée de cellules à deux états (vivante et morte). L'évolution de l'état d'une cellule est déterminée en fonction de son voisinage de Moore, c'est-à-dire par son état propre ainsi que celui de ses huit autres voisines. Les règles pilotant cette évolution sont au nombre de deux et contrôlent la naissance (une cellule morte naît si elle possède trois voisines vivantes) ou la mort d'une cellule (une cellule vivante meurt si elle ne possède ni deux ni trois voisines vivantes). Malgré des règles extrêmement simples uniquement basées sur le nombre de cellules vivantes adjacentes, le « jeu de la vie » est capable de faire apparaître des comportements complexes. La complexité de ces comportements est responsable de son succès au travers de différentes communautés scientifiques. Cet exemple est principalement utilisé à des fins pédagogiques afin de montrer qu'une certaine auto-organisation peut « émerger » d'un système autonome muni de règles extrêmement basiques. Une illustration de cet automate cellulaire est présentée sur la figure 1.1. Le « jeu de la vie » est le parent de différentes simulations à base d'automates cellulaires en biologie. Par exemple, les travaux présentés dans [Kaufman *et al.*, 1985] proposent d'appliquer un automate booléen à l'immunologie, alors que dans [Alarcón *et al.*, 2003], un automate cellulaire est utilisé pour modéliser la croissance d'une tumeur. Le lecteur intéressé peut trouver de nombreux autres exemples d'applications biologiques dans [Ermentrout et Edelstein-Keshet, 1993].

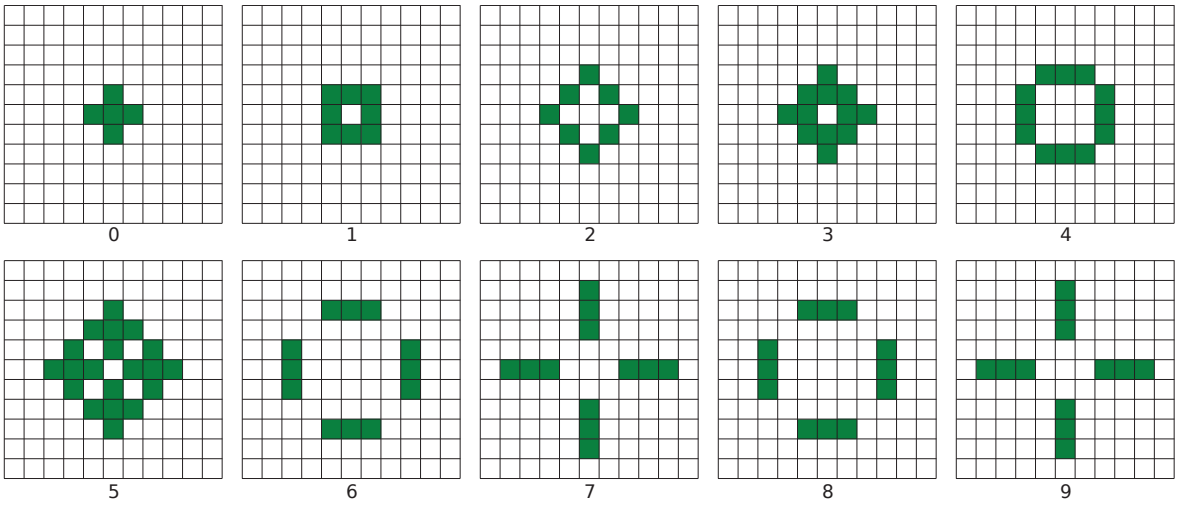


Figure 1.1 – **Illustration du « jeu de la vie »** – Cette figure représente dix itérations du « jeu de la vie ». Les cellules vertes et blanches représentent respectivement les cellules vivantes et mortes. Cette illustration met en évidence l'émergence d'un motif se répétant à partir de la sixième itération.

1.1.1.3 Bilan des approches population-centrées

Nous avons présenté deux approches population-centrées permettant de simuler les systèmes biologiques : les méthodes à base d'équations différentielles et les automates cellulaires. De telles approches permettent de simuler rapidement et efficacement les phénomènes biologiques en caractérisant le comportement de larges populations d'individus dans leur globalité. Néanmoins, en raison du faible niveau de détail que procure cette échelle globale, ces méthodes ne sont pas capables de tenir compte de la possible hétérogénéité des populations modélisées. Il peut s'agir de l'hétérogénéité au sein des comportements, c'est-à-dire lorsque plusieurs individus d'une même population se comportent différemment les uns des autres, ou encore d'une hétérogénéité spatiale comme par exemple, lors de l'apparition de contraintes locales (telle que la formation d'un obstacle) à l'intérieur du milieu étudié. Une telle situation induirait en permanence des changements sur les conditions aux limites du système d'équations différentielles ou de l'automate cellulaire utilisé.

En outre, de telles approches, et particulièrement les équations différentielles, ne sont généralement applicables que lorsque les populations considérées sont suffisamment importantes, c'est-à-dire au sein desquelles la loi des grands nombres est valable. Cependant, certaines approches à base d'équations différentielles probabilistes telles que les équations maîtresses stochastiques permettent de s'affranchir de cette contrainte en modélisant des populations de faible taille [van Kampen, 2007]. Nous ajoutons qu'en assimilant une cellule à un individu (comme notamment dans le « jeu de la vie »), les automates cellulaires semblent pouvoir modéliser un système à l'échelle de l'individu. Or, ce n'est pas le cas. En effet, ils ne permettent pas de suivre précisément l'histoire d'un individu spécifique au cours du temps, c'est-à-dire la succession d'événements l'amenant d'une situation à une autre. Nous ne pouvons, au contraire, suivre au cours du temps que l'état de portions d'espace qui ne peuvent

refléter la présence d'individus qu'à travers leur quantité. Pour obtenir cette propriété il est nécessaire de faire appel aux approches individu-centrées [MacCullough, 1996].

1.1.2 Approches individu-centrées

Contrairement à leurs homologues population-centrés, les approches individu-centrées, parfois qualifiées de systèmes à base de particules dans le domaine de la biochimie, se focalisent sur la modélisation des individus composant une population. En décrivant ainsi le comportement de chacun d'eux individuellement, ce type d'approche est capable de modéliser des systèmes particulièrement hétérogènes, aussi bien au niveau des comportements que de la spatialisation. En outre, il est souvent plus intuitif de décrire le comportement d'un individu unique plutôt que celui de l'ensemble d'une population par l'intermédiaire d'équations différentielles parfois difficiles à formuler. En effet, l'établissement de telles équations est le fruit d'un travail d'analyse et d'abstraction très spécifique au domaine étudié et n'a rien de trivial. Cette simplicité *a priori* et cette capacité à modéliser les systèmes hétérogènes à population réduite peuvent être très attrayantes pour le modélisateur mais les approches individu-centrées ne sont pas dépourvues de défauts. Nous présentons dans cette section l'approche principale permettant de modéliser des systèmes biologiques à l'échelle de l'individu plutôt que celle de la population : il s'agit des systèmes multi-agents.

1.1.2.1 Les systèmes multi-agents

Les systèmes multi-agents sont composés d'individus autonomes (les agents) évoluant au sein d'un environnement qu'ils perçoivent localement. Comme le mentionne Michael Wooldridge dans [Wooldridge, 2009], il n'existe pas de définition universelle des systèmes multi-agents. L'élaboration de cette définition est même particulièrement sujette à débat et controverse en raison de l'origine interdisciplinaire des systèmes multi-agents. En effet, ces derniers sont le résultat de recherches réalisées dans des domaines variés tels que l'intelligence artificielle (autonomie des individus), les systèmes distribués (concurrence entre les individus) ou encore les sciences humaines (sociétés d'individus). Or, ces communautés scientifiques ne définissent pas les systèmes multi-agents de manière identique. Malgré cela, les différents acteurs de ces communautés s'accordent sur l'importance de l'autonomie des agents au sein du système [Tisseau, 2001]. Michael Wooldridge propose ainsi de définir un agent de la manière suivante :

> *« Un agent est un programme informatique situé dans un environnement et capable d'effectuer des actions de manière autonome au sein de cet environnement afin d'atteindre les objectifs pour lesquels il a été conçu*[3]*. »*

Nous adhérons à cette définition et définissons les modélisations multi-agents par l'intermédiaire des caractéristiques suivantes :

- l'autonomie : chaque agent est indépendant et capable de prendre des décisions en fonction des éléments (agents voisins ou environnement) qu'il perçoit ;

3. Traduction française de la définition proposée en anglais dans [Wooldridge, 2009].

- la perception locale : un agent ne perçoit qu'une portion de l'environnement et des agents qui l'entourent ;
- la décentralisation : aucun mécanisme ne contrôle la simulation à l'échelle globale.

La simulation de tels systèmes est obtenue en faisant « vivre » ces agents, c'est-à-dire en exécutant le programme informatique définissant leur comportement, tour à tour à chaque cycle de simulation.

À l'instar des méthodes à base d'équations différentielles et des automates cellulaires, les systèmes multi-agents comptent de nombreuses applications liées à la biologie [Merelli *et al.*, 2007; An *et al.*, 2009]. Nous retrouvons des exemples récurrents tels que l'étude de la cinétique biochimique [Andrews *et al.*, 2009] ou encore celle de la croissance de tumeurs [Byrne et Drasdo, 2009]. À cela s'ajoutent des travaux appliqués au domaine de la biologie animale comme par exemple l'étude du phytoplancton dans les océans [Hellweger et Bucci, 2009] ou l'étude classique de la collecte de nourriture au sein d'une colonie de fourmis [Nugala *et al.*, 1998].

Habituellement, un agent représente un individu de la population modélisée. Cependant, certains travaux tels que [Querrec, 2005] et [Desmeulles, 2006] proposent un modèle de système multi-agents particulier et original en réifiant les interactions des systèmes biologiques. Cela signifie que les auteurs individualisent les interactions liant les individus de la population, plutôt que les individus eux-mêmes. Ainsi, contrairement à une système multi-agents classique dont la simulation est obtenue en faisant « vivre » tour à tour chacun des individus de la population, ces travaux proposent de faire « vivre » les interactions.

Nous notons que la littérature fait état de deux types d'agent : les agents cognitifs (ou intelligents), c'est-à-dire dotés de capacités de raisonnement, et les agents réactifs, c'est-à-dire obéissant à un schéma réflexe stimulus → réponse [Wooldridge, 2009]. Lors de la simulation de systèmes biologiques, la majorité des agents utilisés sont réactifs. En effet, hormis dans certaines situations (notamment l'étude d'animaux dotés de facultés cognitives) la plupart des entités biologiques (cellules, molécules, animaux primitifs) sont dépourvues de capacités cognitives évoluées et se contentent d'effectuer une action lorsqu'une situation particulière survient (interaction avec un autre agent, variation de l'état de l'environnement...). En outre, face aux différentes connotations qui peuvent être données aux termes « agent » et « système multi-agents » en fonction des domaines étudiés, nous privilégions l'utilisation des termes « individu » et « système individu-centré » dans le contexte de ce mémoire.

1.1.2.2 Bilan des approches individu-centrées

En proposant des modélisations à un niveau de détail nettement plus élevé que leurs homologues population-centrés, les approches individu-centrées permettent de tenir compte efficacement de l'hétérogénéité des populations. Cela est dû au fait que le comportement de chaque individu appartenant à la population étudiée est modélisé individuellement. Néanmoins, en raison de cette capacité intrinsèque à modéliser chaque élément du système biologique individuellement, les approches individu-centrées peuvent être extrêmement coûteuses en temps de calcul. En effet, pour obtenir des résultats significatifs, il est souvent nécessaire de simuler un nombre considérable d'individus (de quelques milliers à plusieurs millions)

et parfois, sur une très longue durée. En outre, malgré leur simplicité apparente, lorsque les comportements des individus sont évolués, les modèles individu-centrés peuvent très rapidement devenir compliqués et beaucoup plus difficiles à maintenir qu'un modèle à base d'équations différentielles.

Dans cet état de l'art des approches individu-centrées, nous ne considérons volontairement pas l'approche dite de microsimulation, dont la paternité est généralement accordée à [Orcutt, 1957], qui, bien qu'étant une approche individu-centrée très utilisée (notamment dans les domaines des sciences sociales et de l'économie), est inadéquate pour modéliser la biologie. En effet, la microsimulation consiste à prédire l'évolution d'individus en fonction de larges bases de données et de règles statistiques définissant les préférences et affinités des individus et ne permet clairement pas de modéliser les interactions liant ces individus au sein des systèmes biologiques.

L'étude des approches population et individu-centrées que nous avons proposée nous permet de mettre en évidence leurs principaux avantages et inconvénients lors de la modélisation et simulation de systèmes biologiques. Afin d'obtenir des simulations qui partagent les atouts et s'affranchissent des défauts de ces deux types d'approche, un troisième a vu le jour : il s'agit des approches que nous qualifions de mixtes.

1.1.3 Approches mixtes

Le principal objectif d'une modélisation mixte est de coupler les approches population et individu-centrées. En effet, en les faisant interagir entre elles, ce nouveau type d'approche permet de bénéficier du niveau de détail élevé de modélisation des modèles individu-centrés et des performances des modèles population-centrés lors de la simulation des systèmes biologiques. Par ailleurs, ces derniers font généralement entrer en jeu de multiples phénomènes. Or certains d'entre eux peuvent profiter avantageusement d'une modélisation individu-centrée alors qu'une modélisation population-centrée serait plus adéquate pour les autres. Par exemple, si nous étudions la formation d'un amas de cellules immergées au sein d'un fluide, il est intéressant de modéliser les cellules par l'intermédiaire d'un modèle individu-centré (pour prendre en compte les contraintes spatiales liées à la formation de l'amas mais aussi parce qu'eu égard à la faible quantité de cellules l'approche population-centrée n'est pas idoine) et le fluide par un modèle population-centré basé sur des équations différentielles éprouvées définissant la physique globale du fluide. La littérature propose quelques travaux mettant en œuvre de tels couplages pour la modélisation et la simulation de systèmes biologiques. Nous les organisons selon deux catégories : les modélisations multi-échelles et les modélisations multi-phénomènes.

1.1.3.1 Modélisations multi-échelles

Le premier type de couplage permet de réaliser des modélisations multi-échelles, c'est-à-dire capables de modéliser un même phénomène à plusieurs niveaux de détail différents. Ce couplage est basé sur le fait que, bien que plus rapides que leurs homologues individu-

centrés, les approches population-centrées ne sont pas nécessairement assez fines pour étudier précisément certains phénomènes biologiques. De ce fait, les méthodes multi-échelles visent à favoriser la collaboration entre ces deux approches pour obtenir des simulations à la fois rapides et précises.

Deux types de solutions multi-échelles sont présents dans la littérature. Bien que sensiblement similaires, nous jugeons nécessaire de les distinguer. La première solution consiste à utiliser globalement une approche population-centrée pour modéliser un système. Cependant, si certaines parties représentent un intérêt particulier, alors elles sont modélisées par l'intermédiaire d'approches individu-centrées, parfaitement adaptées pour étudier des contraintes locales. L'influence d'un modèle sur l'autre est légère et n'a lieu qu'à leurs interfaces, c'est-à-dire à la jonction entre les parties modélisées de manière population-centrée et individu-centrée. Ce type de couplage permet ainsi de bénéficier de la rapidité d'exécution des approches population-centrées, sans pour autant négliger les contraintes locales pouvant survenir au sein du système. Il est notamment utilisé dans [Klann *et al.*, 2012] pour la simulation de phénomènes intracellulaires. Dans ces travaux, les molécules du cytoplasme d'une cellule sont modélisées par une méthode utilisant des EDOs basées sur des probabilités alors que celles présentes près des régions d'intérêt, c'est-à-dire près du noyau ou de la membrane, sont modélisées individuellement.

Le second type de solutions multi-échelles met en place un couplage plus fort que le précédent. En effet, dans ce cas, les modèles individu-centrés (de bas niveau) influent directement sur le comportement des modèles population-centrés (de haut niveau). Les résultats émergeant à l'échelle des individus sont utilisés pour alimenter et adapter les modèles gérant les populations. Parfois, cette interaction peut avoir lieu dans le sens inverse, c'est-à-dire que les modèles population-centrés modifient les propriétés de chacun des individus dynamiquement. Cette approche est notamment mise en œuvre dans [Duboz *et al.*, 2003] pour la simulation de copépodes [4] en présence de phytoplancton, où un système d'EDOs de type proie-prédateur est paramétré par un système individu-centré simulant le comportement individuel des copépodes. Cette approche est également présentée dans [Bobashev *et al.*, 2007] dans le cadre de la simulation d'une épidémie. Cette dernière débute par l'utilisation d'une approche individu-centrée, profitant du faible nombre de personnes infectées lors de l'apparition de la maladie, puis se poursuit avec une approche population-centrée, initialisée avec les données produites par le premier modèle, lorsque le nombre de personnes infectées devient suffisamment important. Ces exemples fonctionnent sur le même principe : ils utilisent un modèle individu-centré lorsque la précision souhaitée ne peut pas être atteinte par un modèle population-centré. Ainsi, le couplage multi-échelles permet d'étudier un phénomène biologique en bénéficiant de la fine précision offerte par les approches individu-centrées lorsque celle-ci est nécessaire, tout en tendant vers les performances que proposeraient des approches population-centrées à l'échelle globale.

1.1.3.2 Modélisations multi-phénomènes

En sus de ce premier type de couplage qui utilise deux représentations d'un même phénomène, mais à des niveaux de détail différents, la littérature en propose un second.

4. Les copépodes sont de petits crustacés marins.

Aucunement orienté vers l'aspect multi-échelles, ce dernier consiste plus simplement à établir une collaboration entre les approches population et individu-centrées modélisant chacune des phénomènes distincts. Plusieurs exemples d'une telle collaboration sont présentés dans la littérature. Dans [Jeschke et Uhrmacher, 2008], les auteurs étudient l'influence des populations denses de macromolécules sur leur entourage. Ils utilisent une approche individu-centrée pour modéliser ces macromolécules et une approche population-centrée basée sur des EDOs probabilistes pour les molécules de plus faibles volumes. Un tel couplage est également présenté dans [Miller *et al.*, 1998] où des copépodes, modélisés individuellement, subissent l'advection d'un flux marin calculé grâce à une méthode utilisant des EDPs. Nous pouvons également citer comme dernier exemple la plateforme générique *Virtual Laboratory Environment* présentée dans [Quesnel *et al.*, 2009] qui permet la multi-modélisation de systèmes complexes, qu'ils soient biologiques ou non. Ces travaux proposent un exemple d'utilisation de cette plateforme qui s'éloigne sensiblement du domaine de la biologie puisqu'il s'agit d'une simulation au sein de laquelle des pompiers luttent contre des feux de forêts initiés par des pyromanes. Les pompiers et pyromanes sont modélisés par un système individu-centré tandis que le comportement du feu est géré par un automate cellulaire couplé à des EDOs. Bien que l'exemple proposé par les auteurs se situe hors du contexte biologique de ce mémoire, la plateforme proposée se veut générique et peut donc tout à fait être utilisée pour modéliser et simuler des systèmes biologiques. Notons que les auteurs ne semblent pas se focaliser particulièrement sur l'accélération des simulations réalisées par l'intermédiaire de leur plateforme.

1.1.3.3 Bilan des approches mixtes

L'intérêt majeur des approches mixtes réside dans leur capacité à offrir un compromis intéressant entre performances de calcul (provenant des modèles population-centrés) et niveau de détail (provenant des modèles individu-centrés). De plus, elles favorisent le développement de modélisations et de simulations multi-échelles et multi-phénomènes, un aspect très important des systèmes biologiques où de très nombreux phénomènes entrent en jeu. Néanmoins, le couplage de ces approches est relativement complexe à mettre en œuvre et nécessite de développer des mécanismes leur permettant d'interagir et de se synchroniser.

Quelle que soit l'approche utilisée, c'est-à-dire qu'il s'agisse d'une approche mixte, individu ou population-centrée, la simulation de systèmes biologiques est fréquemment coûteuse en temps de calcul. Cela est principalement dû à la complexité des phénomènes étudiés. Ainsi, il est primordial d'optimiser pleinement les simulations afin d'en maximiser les performances. En effet, une telle optimisation offre deux opportunités : soit augmenter la quantité de données modélisées (nombre d'individus, dimensions du domaine...), soit accélérer la simulation. Pour ce faire, il est nécessaire d'utiliser des solutions informatiques de parallélisation ou de distribution des calculs. La section suivante a pour objectif de présenter les différentes architectures disponibles ainsi que leurs applications à la simulation de systèmes biologiques.

1.2 Solutions de parallélisation et de distribution

Plusieurs solutions techniques s'offrent aux développeurs pour optimiser les performances des simulations informatiques. Nous distinguons deux notions principales que nous définissons de la manière suivante : la parallélisation consiste à répartir des traitements informatiques localement, c'est-à-dire entre les unités de calcul d'une même machine, alors que la distribution permet de les répartir au travers d'un réseau de machines plus ou moins distantes, c'est-à-dire appartenant à un même réseau local ou reliées par internet [Padua, 2011].

Qu'il s'agisse de parallélisation ou de distribution, l'objectif principal de l'opération est de répartir la résolution d'un problème sur l'ensemble des unités de calcul composant l'architecture utilisée. Cette répartition permet aux unités de calcul de résoudre chacune simultanément le sous-problème qui leur est confié, ce qui a pour conséquence d'accélérer le traitement global du problème. Deux méthodes de répartition se distinguent [Zomaya, 2006]. La première consiste à paralléliser ou distribuer plusieurs simulations indépendantes. En effet, pour appréhender et étudier certains phénomènes biologiques, il peut être nécessaire de répéter à de très nombreuses reprises une simulation identique en ne modifiant de l'une à l'autre que certains paramètres d'entrée. Ce type de répartition est relativement trivial car les simulations sont indépendantes. Aucune synchronisation n'est nécessaire au cours des simulations. Néanmoins, cette méthode ne convient que lorsque les simulations doivent être répétées de nombreuses fois comme dans le cadre d'un balayage de paramètres ou d'études statistiques. Cette méthode est notamment utilisée dans [Beberg *et al.*, 2009] pour étudier l'impact de la structure des protéines ou encore dans [Burrage *et al.*, 2003] pour étudier la transcription des gènes. Parfois, une unique simulation est nécessaire, mais elle doit être de taille suffisamment importante pour produire des résultats significatifs. Dans cette situation, une seconde méthode de répartition des calculs est utilisée. Elle consiste à découper le domaine de simulation pour le répartir sur l'ensemble des unités de calcul à disposition, ce qui a pour conséquence de réduire la charge de travail de chacune d'elle, et donc d'accélérer la simulation. Des exemples de ce type de répartition sont notamment présentés dans [Rodin *et al.*, 2011] dans le cadre de la modélisation de phénomènes biochimiques et dans [dos Santos *et al.*, 2004] pour l'étude de l'activité électrique au sein des tissus cardiaques.

Cette section vise à présenter les solutions techniques de parallélisation et de distribution de calculs informatiques à disposition des développeurs et des modélisateurs. Nous débutons par la présentation des principes de fonctionnement des architectures multicœur et multiprocesseur actuelles puis ceux des architectures graphiques, qui depuis quelques années permettent d'effectuer des calculs scientifiques. Nous terminons par une brève étude concernant les systèmes distribués. Notons que nous attachons une attention toute particulière à l'explication du fonctionnement des architectures graphiques. Ceci est dû à la fois à leur récent développement par rapport aux architectures multicœur et multiprocesseur et aux systèmes distribués, mais également à l'intérêt grandissant que leur portent les différentes communautés scientifiques. Nous ajoutons que cette section n'a pas pour objectif de remplacer une documentation technique mais vise uniquement à décrire les concepts, avantages et inconvénients liés à chacune de ces solutions techniques, et notamment dans le cadre de leur application à la simulation de systèmes biologiques.

1.2.1 Architectures multicœur et multiprocesseur

Les processeurs sont des composants électroniques constitués d'une multitude de transistors [5]. Ces transistors sont gravés dans un petit morceau de semi-conducteur en silicium nommé *die*. Ce dernier contient deux éléments principaux : le ou les cœurs, c'est-à-dire les unités de calcul responsables du traitement des instructions provenant des programmes informatiques exécutés, ainsi que les mémoires caches qui permettent de stocker les instructions ou les données fréquemment utilisées par les programmes. Au fil des années, la gravure des transistors s'est affinée. Ces derniers sont donc de plus en plus petits, ce qui permet à la fois d'augmenter leur fréquence et de limiter leur échauffement. En effet, plus un transistor est petit, plus sa vitesse de commutation sera élevée, mais également plus sa consommation électrique sera faible. L'amélioration de la finesse de gravure a aussi pour conséquence d'augmenter le nombre de transistors par *die*. Cela permet aux fabricants d'augmenter la quantité de mémoire des processeurs ou de les faire bénéficier de nouvelles fonctionnalités plus évoluées.

Bien que jusqu'en 2004 ou 2005 les fabricants se soient essentiellement focalisés sur la montée en fréquence des processeurs, la tendance a aujourd'hui changé. En effet, il devient extrêmement difficile et coûteux de continuer à augmenter la fréquence des transistors avec le processus de fabrication actuel. Pour cette raison, les fabricants de processeurs se sont orientés dans une nouvelle direction et ont décidé de multiplier le nombre de cœurs au sein d'un même *die*, plutôt que d'en augmenter la fréquence. Cela permet d'augmenter la puissance de calcul tout en limitant l'échauffement et la consommation électrique des composants. Ainsi, le terme d'architecture multicœur et multiprocesseur désigne une architecture constituée de plusieurs processeurs, chacun doté de plusieurs cœurs. Nous débutons cette section en introduisant les détails matériels concernant ce type d'architecture, puis nous poursuivons par une présentation des modèles de programmation qui lui sont associés.

1.2.1.1 Architecture matérielle

Sur les architectures multicœur et multiprocesseur, la parallélisation d'un traitement est assurée par la mise en œuvre de *threads* (ou fils d'exécution). Composé d'une série d'instructions, chacun d'eux constitue l'élément de base nécessaire à la parallélisation. Cette dernière est obtenue en exécutant plusieurs *threads* simultanément, c'est-à-dire sur des cœurs ou processeurs différents. Nous notons que pour qu'un cœur exécute plusieurs *threads*, il doit alternativement exécuter les instructions de chacun d'eux afin d'émuler la parallélisation du traitement. Néanmoins, certains processeurs sont équipés de la technologie *Simultaneous MultiThreading* (SMT), permettant de scinder un cœur physique en plusieurs cœurs logiques [Padua, 2011]. Ainsi, cette technologie exploite les temps de latence des instructions pour partager les ressources d'un cœur physique, notamment ses unités de calcul et ses mémoires caches, entre différents *threads*, chacun d'eux constituant ainsi un cœur logique pouvant effectuer ses traitements parallèlement aux autres. Sans utiliser cette technologie, un cœur physique ne peut exécuter qu'un seul *thread* simultanément, tandis que grâce à elle, plusieurs

5. Actuellement, ce nombre peut atteindre plusieurs milliards.

threads peuvent l'être. Le nombre de traitements simultanés est défini par le nombre de voies que propose la technologie SMT. Ainsi, l'utilisation de deux voies ou de quatre voies permet d'exécuter respectivement deux ou quatre *threads* parallèlement. Afin de favoriser la lecture de ce mémoire, nous définissons par *Central Processing Unit* (CPU) toute unité de calcul liée à l'architecture multicœur et multiprocesseur, qu'il s'agisse aussi bien d'un processeur, d'un cœur physique ou bien d'un cœur logique obtenu par l'intermédiaire de la technologie SMT.

Ces architectures possèdent une mémoire principale partagée entre les différents CPUs, c'est-à-dire qu'ils bénéficient d'un espace d'adressage mémoire unique où toutes les données sont stockées. Cet espace d'adressage est ainsi accessible par n'importe quel CPU partageant cette mémoire. Cette accessibilité est le principal avantage d'une telle mémoire car elle favorise grandement les interactions et les communications pouvant survenir entre les différentes unités de calcul. Cependant, en raison du fait que tous les CPUs peuvent manipuler cette mémoire partagée, il est nécessaire de le faire avec précaution afin de ne pas corrompre la cohérence des données qu'elle stocke. Lorsqu'une simulation (ou plus généralement, un programme informatique) est exécutée, elle accède régulièrement et fréquemment à cette mémoire principale partagée pour rapatrier les données et les instructions nécessaires à son fonctionnement. Or, ces accès à la mémoire ne sont pas uniformes. En effet, des observations ont montré qu'un programme a tendance à réutiliser les données et les instructions dont il a eu besoin récemment [Hennessy et Patterson, 2011]. Ce phénomène, nommé *principe de localité*, se divise en deux sous-principes :

Le principe de localité spatiale stipule que des éléments contigus en mémoire ont tendance à tous être utilisés dans un intervalle de temps très bref.

Le principe de localité temporelle indique que les zones mémoires récemment manipulées ont une forte probabilité de l'être de nouveau dans un futur proche.

Ainsi, afin de profiter pleinement de ces deux phénomènes, l'architecture mémoire des équipements informatiques actuels est hiérarchisée en plusieurs niveaux. En effet, bien que les données des programmes soient stockées dans la mémoire principale, le ou les CPUs du système n'y accèdent pas directement. Ils passent par l'intermédiaire d'une mémoire spécifique beaucoup plus rapide : la mémoire cache. Le rôle de cette dernière est de stocker temporairement les données qui transitent entre la mémoire principale et le CPU qui les manipule. D'après le principe de localité énoncé précédemment, un CPU a tendance à fréquemment manipuler des données contiguës en mémoire. Ainsi, conserver ces données provenant de la mémoire principale du système (plutôt lente, c'est-à-dire plusieurs centaines de cycles CPU [6]) dans les mémoires caches (très rapides, c'est-à-dire quelques cycles CPU) du CPU qui les utilise, produit un gain de performance conséquent lors de leur manipulation. Un exemple d'architecture mémoire actuelle est présentée sur la figure 1.2. Elle met en évidence l'aspect hiérarchique de la mémoire en représentant ses différents niveaux. Chacun des cœurs possède ici en effet deux niveaux de mémoire cache distincts (niveaux 1 et 2) extrêmement rapides mais de très faible capacité. Qui plus est, ils partagent un cache commun de niveau 3 de plus grande capacité, favorisant le partage de données entre CPUs.

6. Chaque processeur peut de plus intégrer son propre contrôleur mémoire, ce qui rend le temps d'accès aux différentes plages d'adresses physiques variable selon le processeur concerné (NUMA : *Non Uniform Memory Access*).

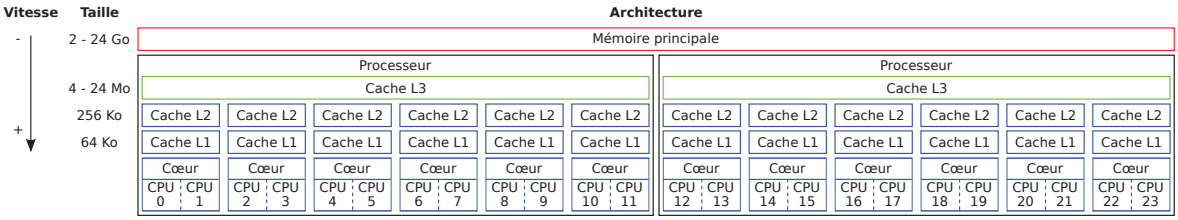


Figure 1.2 – **Exemple d'architecture mémoire actuelle** – Ce schéma représente l'architecture d'un système bi-processeur basé sur une architecture Intel Westmere. Chaque processeur est doté de six cœurs physiques, chacun d'entre eux possédant des caches de niveau 1 et 2 indépendants, et partageant un cache commun de niveau 3. Grâce à la technologie SMT à 2 voies, les 12 cœurs physiques peuvent être assimilés à 24 CPUs logiques s'exécutant simultanément. Notons que la vitesse des mémoires caches augmente selon la proximité de l'unité de calcul, alors que la taille diminue. L'accès non uniforme des processeurs à la mémoire principale (NUMA) n'est pas représenté ici.

Le principe de fonctionnement de la mémoire cache peut se résumer simplement. Lorsqu'un CPU a besoin d'une donnée située à une certaine adresse physique, il vérifie en premier lieu si une copie de cette dernière est déjà présente dans sa mémoire cache de plus bas niveau. Si tel est le cas, il peut l'utiliser immédiatement et rapidement. Dans le cas contraire, la ressource est recopiée depuis les niveaux supérieurs de la hiérarchie (qu'il s'agisse de la mémoire principale ou d'autres niveaux de cache) vers les niveaux inférieurs, par blocs de taille fixe appelés *lignes de cache*. Cette ressource sera ainsi disponible beaucoup plus rapidement lors des futurs accès. Le fonctionnement détaillé de la mémoire cache est présenté dans [Hennessy et Patterson, 2011].

Les mauvaises manipulations de la mémoire cache peuvent entraîner des pertes de performances dramatiques [Harrouet, 2012]. En effet, si plusieurs CPUs essayent d'accéder à une ressource identique, cette dernière va être recopiée dans chacune de leurs mémoires caches. Par conséquent, chaque CPU va manipuler sa propre copie de la ressource, ce qui aboutira fatalement à des inconsistances entre les contenus des différentes mémoires caches. Pour pallier ce problème, des mécanismes de contrôle de cohérence sont utilisés par les architectures actuelles. Ces derniers surveillent les accès à la mémoire et sont capables de répercuter les modifications d'une ligne de cache sur l'ensemble de ses copies en les invalidant et en provoquant des transferts pour leur remise à jour. Bien qu'elles garantissent la cohérence des données en mémoire, la durée de ces opérations de mises à jour a un impact négatif sur les performances du programme, si bien qu'il est primordial de les limiter lors des simulations, qu'elles soient individu ou population-centrées. Ces mécanismes de contrôle de cohérence peuvent potentiellement provoquer de multiples recopies de lignes de cache intempestives et inutiles. En effet, plusieurs CPUs peuvent manipuler des données distinctes indépendamment les uns des autres, alors que celles-ci appartiennent à une même ligne de cache. Cette situation, nommée faux partage, peut notamment survenir lorsque ces données ont été placées de manière contiguë lors du développement ou de l'exécution du programme. Lorsqu'un CPU modifie la ligne de cache concernée, les mécanismes de contrôle de cohérence provoquent l'invalidation immédiate de l'ensemble des copies de cette ligne de cache. De ce fait, quand les autres CPUs accéderont à ces copies invalides, plusieurs recopies de mise à jour vont avoir lieu afin d'actualiser le contenu des mémoires caches avec la modification précédente. Or, cette

mise à jour, néfaste en terme de performances, n'apporte rien car aucun autre CPU n'utilise la donnée qui vient d'être actualisée (ils travaillent tous sur des données différentes). Pour minimiser ces problèmes de faux partage, il est nécessaire de faire en sorte que les données indépendantes soient stockées dans des lignes de cache différentes. Cela peut être réalisé en précisant l'alignement mémoire de chaque donnée. Pour cette raison, il est particulièrement important que les simulations développées pour les architectures multicœur et multiprocesseur portent une attention toute particulière à l'utilisation des mémoires caches et au placement des données en mémoire.

Pour tirer pleinement partie de l'ensemble des CPUs qu'offrent les architectures multicœur et multiprocesseur, un développeur doit faire appel à des modèles de programmation spécifiques permettant d'implémenter des traitements de manière parallèle. La section suivante introduit les deux méthodes les plus populaires et expose les avantages et inconvénients de chacune d'elles.

1.2.1.2 Modèles de programmation

Plusieurs méthodes de programmation permettent d'utiliser des *threads* afin de paralléliser une application informatique sur une architecture multicœur et multiprocesseur. Les deux plus populaires consistent à utiliser, au choix, l'une des interfaces de programmation suivantes : les *threads* POSIX et *Open Multi Processing* (OpenMP). Nous notons que d'autres modèles de programmation alternatifs existent, à l'instar des *Threading Building Blocks* développés par Intel qui encapsulent les *threads* POSIX pour fournir des services similaires à OpenMP [Intel, 2012]. Cependant, nous ne nous intéressons pas à ces derniers car ils semblent très peu utilisés contrairement aux deux méthodes que nous présentons.

Les *threads* POSIX, couramment nommés *pthreads* sont définis dans [IEEE, 1995] et constituent une interface de programmation de bas niveau permettant de gérer les *threads*. Bien que compatibles nativement uniquement avec les systèmes d'exploitation basés sur la plateforme Unix tels que Linux ou Mac OS, plusieurs implémentations Windows sont également disponibles. L'interface est composée de nombreuses fonctions permettant notamment de créer et d'attendre des *threads*. Elle offre également des mécanismes, comme les verrous, visant à protéger les données partagées afin d'en garantir leur cohérence. Chaque *thread* est responsable de l'exécution d'une fonction informatique qui lui est attribuée lors de sa création. Cette fonction définit le traitement que le *thread* doit effectuer avant d'être détruit. Ainsi, l'ensemble des *threads* peut exécuter la même fonction ou alors, des fonctions distinctes. La première situation permet de diviser l'exécution d'un traitement unique entre tous les *threads* (parallélisme de données), alors que dans la seconde, il s'agit plutôt de paralléliser des traitements différents (parallélisme de tâches). Quelle que soit la situation, l'utilisation de plusieurs *threads* permet d'accélérer l'exécution du programme en exécutant un ou plusieurs traitements simultanément. La figure 1.3 propose un exemple d'utilisation des *threads* POSIX et illustre leur fonctionnement sur l'addition de deux vecteurs. La fonction `parallelAdd` représente le traitement parallélisé. Nous constatons que chaque *thread* l'exécutant n'additionne que les composantes du vecteur dont il a la responsabilité. L'exécution parallèle de cette fonction est réalisée par l'intermédiaire de la fonction `vectorAdd` qui s'occupe de la création des différents *threads*. Cet exemple très simple nous montre que

l'utilisation de l'interface de programmation des *threads* POSIX n'est pas triviale et nécessite la mise en œuvre de divers mécanismes alourdissant le code source. Plus récemment, une autre interface de programmation a vu le jour. Nommée OpenMP, elle propose une approche de plus haut niveau qui automatise la gestion des *threads*, facilitant ainsi le travail du développeur et favorisant la lisibilité du code source.

L'interface de programmation OpenMP, définie dans [OpenMP, 2013], est spécialement conçue pour faciliter le développement de traitements parallèles sur les architectures utilisant une mémoire partagée. De plus haut niveau que les *threads* POSIX, elle supporte de très nombreuses architectures de processeurs et une multitude de systèmes d'exploitation (notamment les plus célèbres : Windows, Linux et Mac OS) ce qui la fait bénéficier d'une très grande portabilité. Contrairement aux *threads* POSIX qui sont mis en œuvre en appelant des fonctions classiques, OpenMP est basée sur l'utilisation de directives préprocesseur qui indiquent au compilateur les sections de code que le développeur souhaite paralléliser. Au début de chacune de ces sections, OpenMP lance automatiquement l'exécution de *threads* permettant la parallélisation des traitements. Une fois une section terminée, les *threads* sont stoppés. OpenMP fournit de nombreuses directives permettant notamment de synchroniser les opérations ou de paramétrer le partage de données entre les différents *threads*. L'utilisation de ces directives n'implique pas de lourdes modifications du code source et ainsi, la parallélisation peut être développée de manière incrémentale, c'est-à-dire section par section. En outre, le code source est identique pour les versions séquentielles et parallèles de l'application. En effet, les directives OpenMP sont traitées comme étant des commentaires lors d'une compilation produisant du code purement séquentiel. L'inconvénient majeur d'OpenMP est également l'un de ces principaux avantages : sa mise en œuvre est relativement simple mais malheureusement, contrairement aux *threads* POSIX, elle ne permet pas de bénéficier d'un parallélisme très fin et il est notamment impossible d'attribuer explicitement un *thread* à un CPU. La figure 1.4 illustre l'utilisation d'OpenMP sur l'exemple précédent traitant de l'addition de deux vecteurs (cf. figure 1.3). Cet exemple se veut minimaliste afin de bénéficier d'une bonne lisibilité.

Les deux exemples présentés précédemment mettent en évidence le principal atout d'OpenMP face aux *threads* POSIX : sa facilité de mise en œuvre. En effet, grâce à OpenMP, il est possible de paralléliser des sections de code en ajoutant simplement quelques directives préprocesseur. Malheureusement, le revers de cette simplicité réside dans l'incapacité d'OpenMP à gérer un parallélisme plus fin et plus paramétrable, ce que permettent les *threads* POSIX. Ce paramétrage donne notamment la possibilité de contrôler le nombre exact de *threads* à lancer pour chaque traitement, de les affecter de manière permanente aux CPUs (ce qui a pour conséquence principale d'éviter les mises à jour intempestives des mémoires caches) et à envisager des mécanismes d'équilibrage de charge dynamique. La section suivante propose différents exemples de simulations biologiques utilisant une architecture multicœur et multiprocesseur et mettant en œuvre l'un des deux modèles de programmation présentés précédemment. Nous étudions des exemples basés à la fois sur des approches population-centrées et sur leurs homologues individu-centrées.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

// Définition de la structure contenant les données d'un thread
typedef struct {
  float *A, *B, *C; // Vecteurs
  int N;            // Longueur totale des vecteurs
  int nbThreads;    // Nombre total de threads
  int tid;          // Index du thread
  pthread_t thread; // Thread
} ThParams;

// Définition de la fonction ajoutant une portion de deux vecteurs A et B
void *parallelAdd(void *params)
{
  ThParams *p   = (ThParams*)params;
  int chunkSize = p->N/p->nbThreads;
  int start     = p->tid*chunkSize;
  int end       = p->tid==(p->nbThreads-1) ? p->N : start+chunkSize;
  for (int i = start ; i < end ; ++i)
  {
    p->C[i] = p->A[i] + p->B[i];
  }
  return NULL;
}

// Définition de la fonction ajoutant deux vecteurs A et B de taille N
void vectorAdd(float *A, float *B, float *C, int N)
{
  // Alloue les données liées aux threads POSIX
  int nbThreads = 24;
  ThParams *thParams = (ThParams *)malloc(nbThreads*sizeof(ThParams));

  // Addition parallélisée (avec 24 threads)
  for (int i = 0 ; i < nbThreads ; ++i)
  {
    thParams[i].A         = A;
    thParams[i].B         = B;
    thParams[i].C         = C;
    thParams[i].N         = N;
    thParams[i].nbThreads = nbThreads;
    thParams[i].tid       = i;
    pthread_create(&thParams[i].thread, NULL, parallelAdd, &thParams[i]);
  }

  // Attend la fin de l'exécution des threads (synchronisation explicite)
  for (int i = 0 ; i < nbThreads ; ++i)
  {
    pthread_join(thParams[i].thread, NULL);
  }

  // Libère la mémoire
  free(thParams);
}
```

Figure 1.3 – **Addition de deux vecteurs avec les *threads* POSIX** – Ce programme minimaliste effectue l'addition de deux vecteurs A et B et en sauvegarde le résultat dans un troisième vecteur, nommé C. Les instructions liées aux *threads* POSIX sont préfixées par `pthread_`.

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

// Définition de la fonction ajoutant deux vecteurs A et B de
    taille N
void vectorAdd(float *A, float *B, float *C, int N)
{
  // Addition parallélisée
  int i = 0;
#pragma omp parallel for
  for (i = 0 ; i < N ; ++i)
  {
    C[i] = A[i] + B[i];
  }
  // Synchronisation implicite
}

int main(void)
{
  // Indique la longueur N et l'empreinte mémoire des vecteurs
  int N = 50000;
  size_t size = N * sizeof(float);

  // Alloue les vecteurs A, B et C
  float *A = (float *)malloc(size);
  float *B = (float *)malloc(size);
  float *C = (float *)malloc(size);

  // Remplit les vecteurs A et B avec des valeurs aléatoires
  for (int i = 0 ; i < N ; ++i)
  {
    A[i] = rand()/(float)RAND_MAX;
    B[i] = rand()/(float)RAND_MAX;
  }

  // Ajoute deux vecteurs A et B (calcul parallélisé)
  vectorAdd(A, B, C, N);

  // Libère la mémoire et quitte le programme
  free(A); free(B); free(C);
  return 0;
}
```

Figure 1.4 – **Addition de deux vecteurs avec OpenMP** – Ce programme minimaliste effectue l'addition de deux vecteurs A et B et en sauvegarde le résultat dans un troisième vecteur, nommé C. Les directives OpenMP débutent par `#pragma omp`. Notons qu'OpenMP parallélise les boucles `for` automatiquement et beaucoup plus simplement que les *threads* POSIX. La fonction `main` est identique que ce soit avec OpenMP ou les *threads* POSIX (présentés sur la figure 1.3).

1.2.1.3 Exemples d'applications à la biologie

Exemples basés sur une approche population-centrée

Le premier exemple que nous étudions provient des travaux présentés dans [Islam et Alias, 2010] où les auteurs simulent la croissance de tumeurs cérébrales. Pour cela, ils mettent en œuvre un modèle basé sur des EDPs qu'ils discrétisent temporellement et spatialement. Cette discrétisation spatiale permet de résoudre numériquement les équations. Par ailleurs, cette résolution numérique est parallélisée sur une architecture multicœur et multiprocesseur où chaque CPU est responsable des calculs portant sur une partie du maillage du domaine global. Les simulations montrent de bons résultats mais les auteurs notent cependant l'impact négatif des communications entre processeurs.

Un second exemple d'application peut être trouvé dans [Aldinucci *et al.*, 2011] où les auteurs proposent d'étudier la parallélisation sur une architecture multicœur et multiprocesseur d'algorithmes stochastiques basés sur des EDOs probabilistes. Cette application porte sur l'étude de phénomènes biochimiques. Ces travaux considèrent les deux types de parallélisation que nous avons mentionnés dans l'introduction de la section 1.2, à savoir celle d'une seule simulation répartie sur plusieurs CPUs et celle de plusieurs simulations indépendantes, chacune exécutée par un CPU distinct. Ils proposent des méthodes de parallélisation pour chacun d'eux. Par ailleurs, ils mettent en évidence les points sensibles auxquels il faut faire attention lors de l'utilisation d'un système multicœur et notamment l'importance de la mémoire cache et le besoin d'équilibrage de charge dynamique. Les auteurs montrent qu'en portant un intérêt particulier à ces points sensibles, les architectures multicœur et multiprocesseur permettent d'améliorer sensiblement les performances de leurs simulations.

Exemples basés sur une approche individu-centrée

Dans [Cournède *et al.*, 2009], les auteurs proposent d'étudier la croissance d'une forêt en modélisant chaque arbre individuellement. Le comportement d'un arbre est relativement simple : sa croissance est déterminée par la résolution d'une équation et dépend très peu du comportement des arbres voisins. Cela facilite grandement la parallélisation du problème. Chaque arbre est ainsi attribué à un CPU et est simulé presque indépendamment des autres. Les auteurs observent un gain de performance notable par rapport à une approche monocœur. Nous notons que la parallélisation est obtenue par une plateforme de passage de messages, habituellement dédiée aux systèmes distribués (cf. section 1.2.3).

Un second exemple est présenté dans [Massaioli *et al.*, 2005] où les auteurs proposent une approche générique de simulation des systèmes individu-centrés qu'ils envisagent d'appliquer à la simulation de la réponse du système immunitaire face à un agent pathogène. Le cas d'étude utilisé dans l'article appartient au domaine de la finance mais des analogies sont faites avec le domaine de la biologie. Les simulations sont parallélisées par l'intermédiaire d'OpenMP, le standard présenté précédemment. Les auteurs observent un gain de performance lorsque le nombre de CPUs utilisés augmente. Ils notent cependant que les techniques de parallélisation utilisant OpenMP qu'ils proposent peuvent difficilement être généralisées à

l'ensemble des systèmes individu-centrés et dépendent fortement de l'application modélisée.

Bilan sur les exemples

Ces exemples tendent à indiquer que la simulation parallélisée de systèmes biologiques sur les architectures multicœur et multiprocesseur offre de meilleurs performances que la simulation classique, c'est-à-dire sur un processeur monocœur. Néanmoins, les divers auteurs de ces travaux mentionnent régulièrement l'impact négatif qu'engendrent les communications sur les performances. Selon ces exemples, les architectures multicœur et multiprocesseur semblent, de par la polyvalence de leurs unités de calcul, tout à fait adaptées pour simuler aussi bien des modèles individu-centrés que des modèles population-centrés. Ce résultat nous semble intéressant, d'autant plus que, dans les sections qui suivent, nous montrerons qu'il n'en est pas de même pour les autres architectures capables d'effectuer du calcul parallèle (telles que les architectures graphiques) ou distribué. Malgré cette polyvalence, à notre connaissance, il n'existe pas de travaux qui utilisent les architectures multicœur et multiprocesseur actuelles pour la simulation de systèmes biologiques par l'intermédiaire d'une approche mixte couplant modèles population et individu-centrés.

1.2.2 Architectures graphiques

Les cartes graphiques, ou *Graphics Processing Units* (GPUs), sont des cartes d'extension informatiques, initialement dédiées à la production et à l'affichage d'images sur des périphériques de sortie vidéo tels que les écrans. En outre, ces composants sont spécialisés dans le rendu de scènes 3D. Ces opérations de rendu mettent en œuvre une multitude de calculs similaires sur des ensembles de données représentant une scène 3D, telles que les coordonnées spatiales, les normales ou encore les textures. Afin d'y parvenir efficacement, les GPUs sont équipés d'un nombre important d'unités de calcul (nettement plus élevé que celui des processeurs classiques) conçues pour appliquer simultanément un traitement identique sur un ensemble de données. Bien qu'initialement spécialisés dans les opérations de rendu, les GPUs actuels sont aujourd'hui plus polyvalents et leurs unités de calcul peuvent être utilisées pour effectuer des calculs scientifiques n'ayant pas pour finalité immédiate la production de rendu graphique. Cette pratique porte le nom de *General-Purpose computation on Graphics Processing Units* (GPGPU) [Harris, 2003].

Les premières tentatives de programmation GPGPU ont été rendues possibles grâce à la naissance des langages de calcul de *shaders*, des programmes de rendu exécutés sur GPU. En effet, en détournant les langages de programmation de *shaders* tels que GLSL [OpenGL, 2012], Cg [Mark *et al.*, 2003] ou encore HLSL [Gray, 2003], les développeurs ont pu élaborer les premiers programmes non-graphiques à destination des GPUs. Malheureusement, ces langages ne sont absolument pas adaptés à la mise en œuvre de calculs scientifiques, plus complexes que de simples projections 3D. Pour pallier ce problème, les universités de Stanford et de Waterloo ont proposé des langages de haut niveau plus adéquats. Respectivement nommés BrookGPU [Buck *et al.*, 2004] et Sh [McCool *et al.*, 2004], ils ont pour objectif de simplifier la programmation GPGPU et ont participé à l'émergence des normes et plateformes actuelles.

Les recherches liées au mouvement GPGPU ont donné naissance en 2006 à une plateforme de développement spécialisée mise au point par NVIDIA. Nommée *Compute Unified Device Architecture* (CUDA), cette dernière propose un modèle de programmation dédié aux GPUs de la marque [NVIDIA, 2012a]. L'avènement du mouvement GPGPU a ensuite abouti, quelques années plus tard en 2008, à l'élaboration de la norme *Open Computing Language* (OpenCL) par le consortium industriel Khronos [Khronos, 2012]. Elle spécifie, indépendamment de la marque et du modèle des GPUs, une interface et un langage de programmation (similaire au langage C) à destination de ce type de matériel. Tandis qu'OpenCL propose une interface polyvalente et indépendante du matériel, la plateforme propriétaire CUDA bénéficie quant à elle de nombreuses optimisations à destination des GPUs de marque NVIDIA. Notons que des études récentes ont démontré que les programmes OpenCL optimisés pour les architectures NVIDIA proposent des performances similaires vis-à-vis de leurs homologues développés en CUDA [Karimi *et al.*, 2010; Fang *et al.*, 2011]. Bien que le fonctionnement de ces deux approches soit très similaire, certaines différences subsistent, notamment au niveau des terminologies utilisées. D'autres technologies alternatives moins répandues existent, telles que Microsoft *DirectCompute* [Boyd, 2008], Intel *Array Building Block* [Newburn *et al.*, 2011] ou encore AMD *Accelerated Parallel Processing* [AMD, 2012].

Contrairement à OpenCL, la plateforme CUDA offre l'avantage d'être nativement optimisée pour les GPUs NVIDIA. De ce fait, elle intègre immédiatement, c'est-à-dire dès leurs sorties, les différentes nouveautés proposées par les cartes graphiques de la marque. En outre, CUDA permet de bénéficier d'un meilleur contrôle sur les transferts de données entre la mémoire principale de la machine hôte et les différents GPUs qu'elle contient. Pour ces raisons, nous choisissons de considérer uniquement la technologie CUDA dans la suite de ce mémoire. Néanmoins, l'ensemble des concepts et principes détaillés dans cette section est aisément transposable à la norme OpenCL ainsi qu'à toute autre technologie de programmation GPGPU introduite précédemment.

1.2.2.1 Architecture matérielle

Un GPU est une carte d'extension dont les ressources (aussi bien les mémoires que les unités de calcul) s'ajoutent aux ressources intrinsèques de l'équipement informatique hôte. Ainsi, un GPU peut exécuter un programme de manière autonome, sans solliciter les ressources de la machine hôte, les laissant disponibles pour l'exécution potentielle d'autres applications.

La figure 1.5 représente l'architecture d'une carte graphique selon CUDA. Le GPU (*Device*) possède une mémoire globale de grande capacité (jusqu'à plusieurs gigaoctets) visant à stocker l'ensemble des données du programme exécuté. Il est constitué de plusieurs multiprocesseurs (*Streaming Multiprocessors*), eux-mêmes dotés de nombreux processeurs responsables de l'exécution des traitements confiés au GPU. Ces processeurs (*Stream Processors* ou *CUDA cores*) possèdent une mémoire partagée commune plus rapide que la mémoire globale, mais de faible capacité (typiquement quelques kilooctets). Elle permet aux processeurs de partager des données communes très efficacement et peut être explicitement utilisée en tant que mémoire cache pour la mémoire globale. Par ailleurs, sur les GPUs actuels, une partie (un quart, la moitié ou les trois quarts) de cette mémoire partagée est utilisée automatiquement

comme mémoire cache. Deux mémoires caches supplémentaires sont disponibles. Nommées *Constant Cache* et *Texture Cache*, elles permettent d'accélérer respectivement les accès à la zone mémoire stockant les constantes et à la zone stockant les textures (structures 2D). Notons également que chacun des processeurs possède un ensemble de registres qu'il utilise pour stocker ses variables locales.

Afin d'exploiter pleinement cette architecture matérielle, la plateforme CUDA introduit un modèle de programmation permettant d'exécuter le plus efficacement possible des traitements parallèles sur les GPUs NVIDIA.

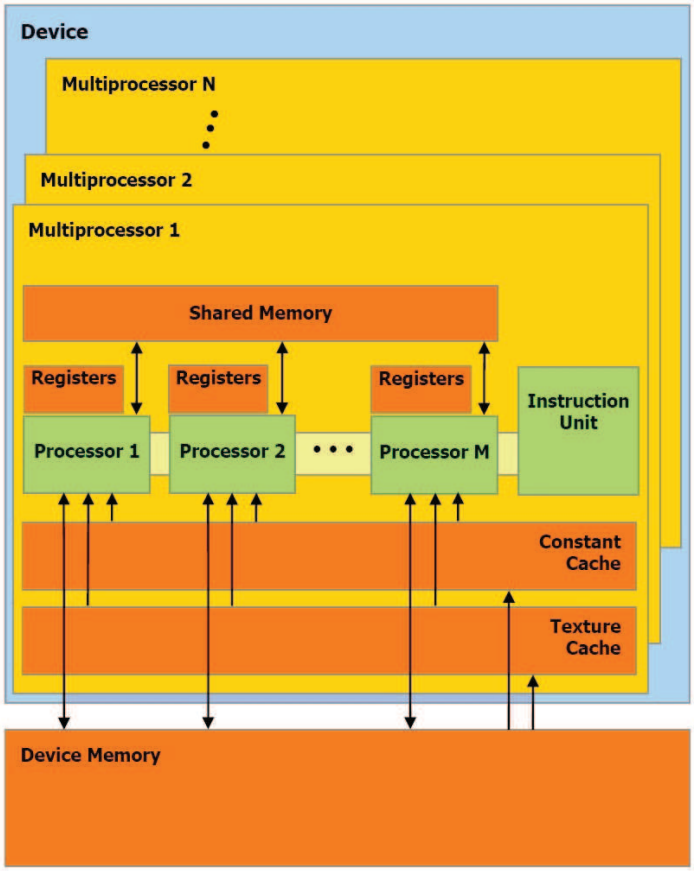


Figure 1.5 – **Modèle matériel d'un GPU** – Ce diagramme provient de la première version du guide de programmation CUDA de NVIDIA. Il illustre l'architecture d'un GPU et présente la terminologie utilisée pour désigner les différents composants.

1.2.2.2 Modèle de programmation

Chaque traitement parallèle est défini par l'intermédiaire d'une fonction spécifique, nommée *kernel*, agissant sur des tableaux de données et dont l'exécution est divisée entre une multitude de *threads*. Ces derniers sont regroupés par blocs au sein desquels ils s'exécutent parallèlement et peuvent communiquer par l'intermédiaire d'une mémoire partagée. Des primitives de synchronisation sont disponibles pour coordonner l'exécution de l'ensemble

des *threads* appartenant à un même bloc. Un ensemble de blocs exécutant un traitement identique, et donc le même *kernel*, forme une grille. La figure 1.6 illustre l'exécution de deux *kernels* et représente la hiérarchie de grilles, blocs et *threads* correspondante.

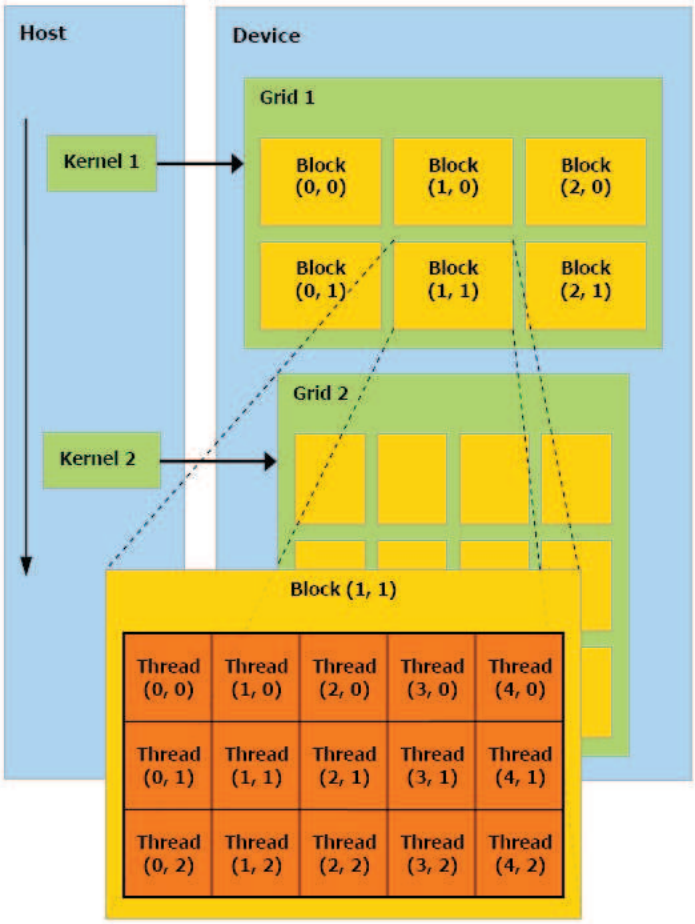


Figure 1.6 – **Hiérarchie des *threads* sur GPU** – Ce diagramme provient de la première version du guide de programmation CUDA de NVIDIA. Une grille (*grid*) est composée de blocs (*blocks*), eux-mêmes composés de *threads* exécutant simultanément un même *kernel*.

Chaque grille (et par conséquent chaque *kernel*) doit être exécutée par un unique GPU. L'ensemble des blocs de *threads* qu'elle contient est ensuite attribué aux multiprocesseurs composant le GPU. Si un multiprocesseur dispose de suffisamment de ressources, il peut exécuter plusieurs blocs parallèlement. Dans le cas contraire, il traite les blocs de manière séquentielle. Les blocs sont complètement indépendants les uns des autres, ce qui assure un passage à l'échelle très efficace. En effet, si nous considérons un nombre fixe de blocs, une augmentation du nombre de multiprocesseurs entraînera automatiquement une amélioration des performances de traitement. Enfin, chaque *thread* est exécuté individuellement par un processeur CUDA.

Les multiprocesseurs manipulent les *threads* par groupes de 32, nommés *warps*[7]. Les

7. Nom donné par analogie avec les activités de tissage où un *warp* désigne l'ensemble des fils (*threads*) longitudinaux parallèles composant un tissu.

threads présents au sein d'un même *warp* sont liés par la propriété suivante : ils exécutent nécessairement la même instruction simultanément sur leurs propres données. L'origine de cette propriété se situe au niveau du matériel. En effet, plusieurs processeurs partagent un même décodeur d'instructions (cf. figure 1.5). Cela permet d'économiser de nombreux transistors et de simplifier les circuits électroniques, et ainsi, d'augmenter les performances brutes des GPUs (au détriment de leur souplesse et de leur polyvalence). Cette propriété est caractéristique de l'architecture *Single Instruction, Multiple Thread* (SIMT), une architecture similaire à l'architecture *Single Instruction, Multiple Data* (SIMD). Elles se distinguent par leur manière de gérer des opérations sur un ensemble de données. En effet, l'architecture SIMD décrit les opérations sous forme vectorielle, alors que l'architecture SIMT spécifie les opérations pour un élément individuel (en l'occurrence, un *thread*). Cela rend l'architecture SIMT plus adaptée aux situations où certaines données doivent subir un traitement différent les unes des autres, comme lors des branchements conditionnels.

Lorsqu'un *warp* exécute une instruction manipulant des données stockées en mémoire globale, il doit préalablement les récupérer. Cela induit un transfert de données au cours duquel le *warp* ne peut plus exécuter d'instructions. Il doit attendre la fin du transfert pour pouvoir poursuivre son exécution, ce qui entraîne des latences. Pour masquer ces phases d'attente, les ordonnanceurs des *warps* peuvent effectuer des changements de contexte leur permettant d'interrompre temporairement l'exécution d'un *warp* en attente de données, au profit d'un autre, prêt à exécuter une instruction. Ce mécanisme est illustré sur la figure 1.7. Ces opérations de changement de contexte sont, contrairement à leurs homologues sur CPU, très peu coûteuses en termes de performances. En effet, les changements de contexte sur CPU entraînent généralement la mise à jour des contenus des mémoires caches, alors que sur GPU, cette opération ne modifie pas l'état des différentes mémoires. Cela est dû au fait que chaque multiprocesseur maintient à jour l'ensemble des contextes d'exécution (registres, mémoire partagée...) des *warps* de son bloc courant tant que leur exécution n'est pas achevée. Par conséquent, les échanges de contexte n'ont aucun impact sur les performances. Ce mécanisme permet ainsi de masquer idéalement les latences survenant lors des transferts de données. Notons cependant que, pour maximiser l'efficacité de ce mécanisme, les blocs doivent nécessairement contenir un nombre important de *warps*.

De plus, le principe de fonctionnement de cette architecture SIMT impose une synchronisation implicite de l'ensemble des *threads* d'un *warp*. En effet, à chaque instant, ces *threads* exécutent tous une instruction identique, ce qui leur assure de terminer leur exécution simultanément. Néanmoins, cette propriété des architectures SIMT implique de lourdes conséquences sur le flot d'exécution des *kernels* mettant en œuvre des branchements conditionnels. Lorsqu'une telle divergence survient au sein d'un *kernel*, les *threads* respectant la condition de branchement sont exécutés alors que les autres sont temporairement désactivés (cette séquence d'instruction ne les affecte pas). Si une condition complémentaire est proposée, les premiers *threads* sont à leur tour désactivés tandis que les seconds s'exécutent. De ce fait, le temps d'exécution total d'une alternative correspond à la somme des temps d'exécution de ses deux branches. Notons qu'en programmation classique sur CPU, le temps d'exécution total correspondrait à la durée de la branche la plus longue. La figure 1.8 illustre cet état de fait. Toutefois, si tous les *threads* d'un *warp* empruntent la même branche de l'alternative, alors la durée de l'alternative est réduite à la durée de cette branche. Ainsi, afin de minimiser les temps d'exécution des *kernels*, il est primordial de maximiser les chances

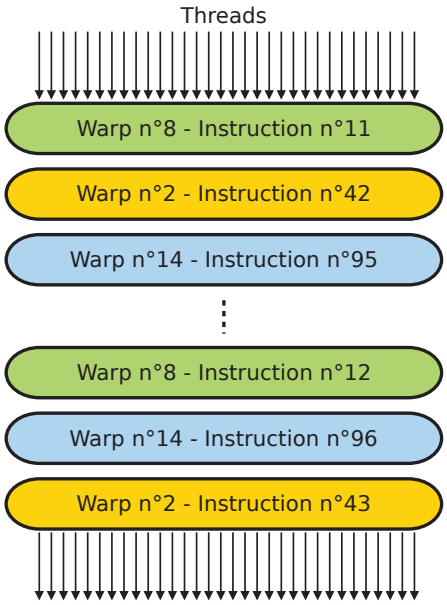


Figure 1.7 – **Masquage de latences par changement de contexte** – Ce schéma illustre le mécanisme de changement de contexte mis en œuvre par les ordonnanceurs de *warps* afin de masquer les latences. Ces dernières surviennent lorsque l'exécution d'une instruction nécessite un transfert de données en provenance de la mémoire globale. Pour masquer l'attente produite par de tels transferts, un ordonnanceur peut interrompre temporairement l'exécution d'un *warp* (par l'intermédiaire d'un changement de contexte) pour en exécuter d'autres pendant le transfert de données.

que l'ensemble des *threads* de tout un *warp* suivent un chemin identique au sein des branchements conditionnels. Un moyen simple d'y parvenir consiste à trier préalablement les données d'entrée du problème.

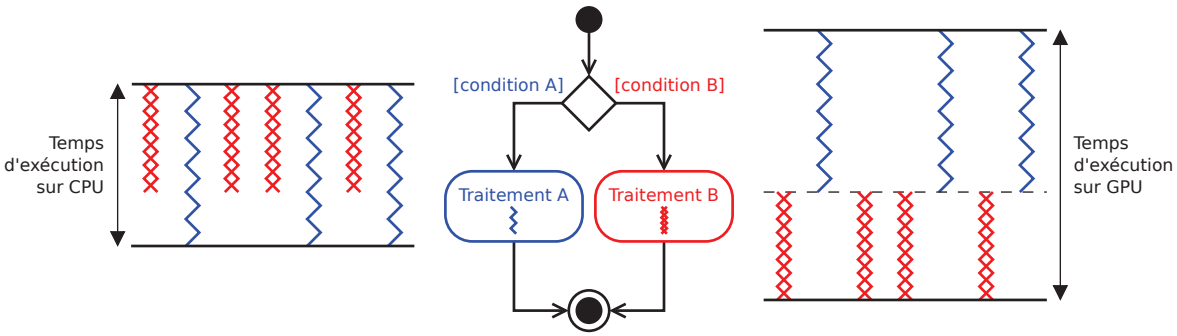


Figure 1.8 – **Branchement conditionnel sur CPU et GPU** – Cette figure représente un diagramme d'activités UML (au centre) représentant une divergence entre deux traitements alternatifs. Les schémas entourant ce diagramme présente la manière dont ces traitements sont exécutés sur CPU (à gauche) et sur GPU (à droite). Ils mettent notamment en évidence la différence de temps d'exécution entre les deux architectures.

De plus, puisque tous les *threads* d'un *warp* exécutent simultanément la même instruc-

tion, si celle-ci concerne un accès à la mémoire globale, alors il s'agit de faire transiter entre les unités de calcul et cette mémoire tout un paquet de données. Le matériel est en conséquence conçu pour optimiser les accès à la mémoire globale en transférant des blocs de 32 mots (un *warp*) contigus. L'utilisation de ces transferts est donc optimale lorsque tous les *threads* constitutifs d'un *warp* accèdent simultanément à des données contiguës en mémoire globale. Dans ce cas, toutes les données transférées sont utiles aux *threads* du *warp* ; on parle alors d'accès coalescent. Dans le cas contraire, plusieurs transferts successifs sont nécessaires pour contenter tous les *threads* d'un *warp* mais chacun d'eux est parsemé de données qui ne seront exploitées par aucun *thread* du *warp* lors de cette instruction, ce qui constitue un gaspillage du débit de données. Ceci implique que le placement des données dans la mémoire globale d'un GPU doit être choisi de manière très spécifique à l'algorithme exécuté par les *threads* afin d'exploiter pleinement les capacités de ce matériel.

En raison des contraintes engendrées par leur architecture matérielle, la programmation sur GPU est moins intuitive que celle sur CPU et demande ainsi d'importants efforts d'adaptation de la part des développeurs. Conscients de cet état de fait, les concepteurs de plateformes de développement telle que CUDA ne cessent de proposer de nouvelles améliorations permettant de simplifier la tâche des développeurs. Par exemple, la technologie *Dynamic parallelism* introduite dans la version 5.0 de CUDA permet à des *kernels* d'en exécuter d'autres [NVIDIA, 2012b]. Cette opération, auparavant impossible à réaliser, simplifie grandement le processus de développement d'applications sur GPUs. Bien que ces améliorations rendent la programmation sur GPU plus accessible, il est tout de même nécessaire de respecter les contraintes inhérentes à l'architecture SIMT lors de la mise en œuvre d'une application sur GPU.

1.2.2.3 Démarche générale de mise en œuvre

Lors du développement d'une application sur GPU, il convient de formuler le problème en respectant les spécificités et les contraintes énoncées précédemment. Ainsi, pour exploiter pleinement l'architecture SIMT, il est préalablement nécessaire de diviser le problème étudié en une multitude de sous-problèmes élémentaires identiques dont le comportement est défini par un *kernel*. Chacun de ces sous-problèmes est associé à un *thread*, exécutant ce *kernel* sur ses propres données. En effet, les données d'entrée et de sortie du problème global sont généralement exprimées sous la forme de tableaux informatiques et chaque *thread* agit uniquement sur les éléments présents à un indice lui correspondant. La démarche de mise en œuvre généralement adoptée est la suivante :

1. Lister l'ensemble des GPUs disponibles sur la machine hôte.
2. Compiler le *kernel* spécifiquement pour le GPU sélectionné. Il est possible de sauvegarder le code binaire généré afin d'éviter une nouvelle compilation à chaque exécution.
3. Allouer les zones mémoire nécessaires au stockage des données d'entrée et de sortie du problème, à la fois sur la machine hôte mais également sur le GPU sélectionné. En effet, comme mentionné précédemment, la machine hôte et le GPU ne partagent pas de mémoire.

4. Transférer les données d'entrée depuis la mémoire principale de la machine hôte vers la mémoire globale du GPU.
5. Demander l'exécution du *kernel* sur le GPU sélectionné.
6. Transférer les données de sortie générées par le *kernel* depuis la mémoire globale du GPU vers la mémoire principale de la machine hôte. Cette opération s'effectue lorsque tous les *threads* de tous les blocs ont terminé leur exécution, ce qui assure l'intégrité des données de sortie.

Afin de ne pas monopoliser les ressources de la machine hôte inutilement, les opérations telles que les transferts mémoire depuis la machine hôte vers le GPU (uniquement dans ce sens) ainsi que les lancements des *kernels*, sont exécutées de manière asynchrone. Cela permet à la machine hôte de poursuivre ses traitements dès l'instant où l'une de ces opérations a été lancée, sans nécessairement attendre qu'elle se termine.

Il est important de remarquer que les opérations de transferts mémoire, quel qu'en soit le sens (étapes 4 et 6), ne sont pas anodines. En effet, les communications entre la mémoire principale de la machine hôte et le GPU sont très lentes. Dans certaines circonstances, ces transferts peuvent même devenir prépondérants face au temps d'exécution d'un *kernel*. Dans de telles situations, il est généralement préférable de renoncer à effectuer ces traitements sur GPU et de privilégier une approche sur CPU.

Lors de la demande d'exécution du *kernel* (étape 5), il est nécessaire de spécifier les dimensions du problème étudié, c'est-à-dire le nombre de blocs et le nombre de *threads* par bloc nécessaires au traitement de l'intégralité des données. Ces dimensions sont accessibles à l'intérieur du *kernel* par l'intermédiaire de deux variables, respectivement nommées `gridDim` et `blockDim`, mises à disposition par CUDA. Ces deux informations peuvent chacune être fournie en une, deux ou trois dimensions, en fonction du problème étudié. Par exemple, s'il s'agit d'additionner des vecteurs, alors on indiquera la longueur de ces vecteurs (chaque *thread* calcule la somme à un indice donné) ; s'il s'agit d'appliquer un traitement sur une image, alors on indiquera la hauteur et la largeur de l'image (chaque *thread* manipule un pixel de l'image) ; enfin s'il s'agit d'étudier le comportement d'un fluide au sein d'un volume, alors on indiquera le nombre de subdivisions du volume sur chacun des trois axes caractérisant l'espace (chaque *thread* détermine l'état d'une portion infinitésimale du volume). Notons qu'il est tout à fait envisageable, moyennant quelques divisions euclidiennes, de n'utiliser qu'une seule dimension quel que soit le problème étudié.

Afin d'identifier aisément les données qu'un *thread* manipule, chacun d'eux est désigné par un indice local, nommé `threadIdx`, permettant de le repérer au sein du bloc auquel il appartient. Cet indice est accompagné d'un second indice, nommé `blockIdx`, visant à identifier la position de ce bloc au sein de la grille exécutant le *kernel* étudié (cf. figure 1.6 de la page 27). Ces deux indices peuvent, à l'instar des dimensions du problème, être exprimés par l'intermédiaire d'une, deux ou trois composantes (`x`, `y` et `z`). L'ensemble de ces variables définies au sein de CUDA permet de repérer chaque *thread* de manière unique. Par exemple, l'expression `threadIdx.x + blockIdx.x * blockDim.x` détermine l'indice global d'un *thread* au sein d'une grille, dans le cadre d'un problème unidimensionnel.

Le code source présenté sur la figure 1.9 page 33 permet d'illustrer nos propos sur un exemple simple à une dimension : l'addition de deux vecteurs. Il met en évidence les étapes 3 à

6 de la démarche énoncée précédemment. Afin d'en faciliter la lisibilité, ce code minimaliste ne gère volontairement aucune erreur et a été enrichi de nombreux commentaires. Notons que les étapes 1 et 2, c'est-à-dire le choix du GPU et la compilation du *kernel* sont automatiquement prises en charge par le préprocesseur du compilateur CUDA proposé par NVIDIA : `nvcc`. Cependant, ces deux étapes peuvent également être réalisées explicitement au sein du code exécuté sur la machine hôte.

Dans la section suivante, nous proposons d'étudier quelques exemples d'utilisation de cette architecture parallèle. L'ensemble de ces exemples appartient au domaine de la biologie, cependant, nous distinguons ceux utilisant une approche population-centrée de ceux privilégiant une approche individu-centrée. Cette étude nous permet d'identifier les multiples avantages et inconvénients des GPUs dans le cadre de la parallélisation de simulations de systèmes biologiques.

1.2.2.4 Exemples d'applications à la biologie

Exemples basés sur une approche population-centrée

Un premier exemple de simulation biologique population-centrée parallélisée sur GPU peut se trouver dans [Fidjeland *et al.*, 2009] où les auteurs étudient le déchargement de neurones grâce à un modèle de neurone basé sur des EDOs. Leur approche consiste à diviser l'ensemble des neurones du réseau étudié en partitions, qui sont ensuite confiées aux différents multiprocesseurs du GPU. Dans leur article, les auteurs n'exécutent leur simulation que sur un GPU mais prévoient d'essayer d'implémenter leur modèle sur une architecture multi-GPUs. En outre, ils mentionnent le fait qu'à l'instar des communications entre processeurs, le coût du dialogue entre plusieurs GPUs ne sera pas négligeable lors de ces futures simulations.

Un deuxième exemple est proposé dans [Rocha *et al.*, 2010] où les auteurs simulent l'activité électrique au sein du cœur. Pour cela, ils utilisent un modèle couplant des EDOs avec des EDPs qu'ils résolvent grâce à un GPU. La résolution numérique des EDPs nécessite une discrétisation temporelle et spatiale, sous la forme d'un maillage, et se prête donc de ce fait très bien à l'utilisation d'un GPU : chaque maille est traitée par un *thread* distinct. En effet, les traitements à effectuer au sein de chacune des mailles sont identiques et donc ce type de problème est un candidat idéal pour l'utilisation d'architectures SIMT comme les GPUs.

Certains travaux, notamment [Li et Petzold, 2007], préfèrent paralléliser de nombreuses simulations indépendantes plutôt que de répartir les traitements d'une seule simulation. Les auteurs utilisent cette méthode pour exécuter un grand nombre de simulations de systèmes biochimiques. Pour tenir compte de l'aspect stochastique inhérent à ce type de phénomène, ils utilisent un modèle à base d'EDOs probabilistes qu'ils simulent de très nombreuses fois pour en extraire des données statistiques telles que la moyenne ou la variance des simulations. Ce type de simulation se prête également très bien à l'utilisation d'un GPU qui propose un très grand nombre d'unités de calcul (capable chacune d'exécuter une simulation indépendante) à un coût financier très faible. En outre, les auteurs mentionnent l'importance de respecter le modèle de programmation prescrit par CUDA pour obtenir des performances optimales.

```
1  #include <stdio.h>
2  #include <cuda_runtime.h>
3
4  // Définition du kernel ajoutant deux vecteurs A et B de taille N
5  __global__
6  void vectorAdd(const float *A, const float *B, float *C, int N)
7  {
8    int id = threadIdx.x + blockIdx.x * blockDim.x; // Indice global
9    if (id < N) C[id] = A[id] + B[id];
10 }
11
12 int main(void)
13 {
14   // Indique la longueur N et l'empreinte mémoire des vecteurs
15   int N = 50000;
16   size_t size = N * sizeof(float);
17
18   // Alloue les vecteurs A, B et C dans la mémoire de l'hôte (étape 3)
19   float *h_A = (float *)malloc(size);
20   float *h_B = (float *)malloc(size);
21   float *h_C = (float *)malloc(size);
22
23   // Remplit les vecteurs A et B avec des valeurs aléatoires
24   for (int i = 0 ; i < N ; ++i)
25   {
26     h_A[i] = rand()/(float)RAND_MAX;
27     h_B[i] = rand()/(float)RAND_MAX;
28   }
29
30   // Alloue les vecteurs A, B et C dans la mémoire du GPU (étape 3)
31   float *d_A = NULL, *d_B = NULL, *d_C = NULL;
32   cudaMalloc((void **)&d_A, size);
33   cudaMalloc((void **)&d_B, size);
34   cudaMalloc((void **)&d_C, size);
35
36   // Copie les vecteurs A et B de l'hôte vers le GPU (étape 4)
37   cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
38   cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);
39
40   // L'hôte demande l'exécution du kernel au GPU avec des blocs de 256
        threads (étape 5)
41   int threadsPerBlock = 256;
42   int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;
43   vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);
44
45   // Copie le vecteur résultat C depuis le GPU vers l'hôte (étape 6)
46   cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
47
48   // Libère les mémoires du GPU et de l'hôte
49   cudaFree(d_A); cudaFree(d_B); cudaFree(d_C);
50   free(h_A); free(h_B); free(h_C);
51
52   // Réinitialise le GPU et quitte le programme
53   cudaDeviceReset();
54   return 0;
55 }
```

Figure 1.9 – **Addition de deux vecteurs sur GPU** – Ce programme minimaliste effectue l'addition de deux vecteurs A et B et en sauvegarde le résultat dans un troisième vecteur, nommé C. Les termes de couleur orange mettent en évidence les mots clés et fonctions directement liés à CUDA.

L'ensemble de ces exemples offre des performances nettement plus élevées que des simulations similaires exécutées sur CPU. Qui plus est, aucun inconvénient majeur ne semble être souligné par ces trois études. Nous apportons des explications tempérant ces conclusions dans la suite de ce document, immédiatement après une série d'exemples de simulations individu-centrées réalisées sur GPU.

Exemples basés sur une approche individu-centrée

Le premier exemple [Dematté, 2012] vise à simuler des systèmes de réaction-diffusion biochimiques. L'auteur modélise chaque molécule comme un point évoluant sur une grille en trois dimensions. Lorsque ces molécules sont suffisamment proches, elles sont susceptibles de réagir entre elles et d'en créer une nouvelle. Bien que l'ajout et la destruction d'entités soient aisés sur CPU, les opérations d'allocation dynamique leur étant liées sont difficiles à mettre en œuvre sur GPU et leur réalisation peut nécessiter des transferts de données supplémentaires pénalisants entre CPU et GPU. Malgré cela l'auteur obtient des performances nettement supérieures à une approche sur CPU.

Nous présentons un deuxième exemple, détaillé dans [D'Souza *et al.*, 2009], concernant la simulation de la réaction du système immunitaire se défendant contre la tuberculose. Les auteurs modélisent les cellules du système immunitaire par l'intermédiaire d'une approche individu-centrée implémentée sur GPU. Le comportement des individus est sensiblement le même que dans l'exemple précédent, c'est-à-dire déplacement et collision, bien que le domaine d'application soit différent. Les auteurs mettent en évidence les mêmes difficultés que dans [Dematté, 2012] et notamment au niveau de l'allocation de mémoire dynamique d'individus. Ils contournent le problème en créant les nouveaux individus dans les cases des tableaux de données laissées vides par la destruction d'autres individus. Cette méthode ne nous semble pas fonctionner si le nombre d'individus à ajouter est supérieur au nombre d'individus détruits. Par ailleurs, ils mentionnent également l'incapacité des GPUs à simuler des phénomènes asynchrones (à l'instar des automates cellulaires) et insistent sur le fait que l'exécution d'une entité ne doit pas dépendre de celle des autres au cours du même pas de simulation. Dans le cadre de leur application, les auteurs observent un gain de performance conséquent par rapport aux simulations réalisées sur CPUs.

Le troisième et dernier exemple que nous proposons provient de [Tripodi *et al.*, 2012]. Dans ces travaux, les auteurs appliquent leur modèle individu-centré à la simulation de la morphogénèse. Plusieurs entités évoluent dans un environnement discret sur lequel elles peuvent se déplacer, s'agrandir, se réduire, se diviser (et donc créer une nouvelle entité) et finalement mourir. L'aspect discret du modèle est idéal pour la simulation sur GPU, ce qui explique le gain de performance que les auteurs observent par rapport à une exécution sur CPU. À l'instar des travaux précédents, les auteurs mentionnent le besoin d'éviter des allocations dynamiques de mémoire. Ainsi, pour pouvoir créer des entités dynamiquement au cours des simulations, les tableaux de données sont dotés d'une taille maximale (comme dans [D'Souza *et al.*, 2009]), ce qui limite la quantité d'informations pouvant être modélisée.

Bilan sur les exemples

Les exemples présentés précédemment mettent en évidence l'efficacité de la simulation sur GPU dans le cadre des approches population-centrées et donc, de la résolution d'équations différentielles. En effet, généralement ce type d'approche peut se paralléliser très aisément, soit en utilisant un découpage du domaine étudié comme dans [Fidjeland *et al.*, 2009; Rocha *et al.*, 2010], soit en exécutant plusieurs simulations indépendantes en parallèle à l'instar de [Li et Petzold, 2007]. Malheureusement, la situation est différente en ce qui concerne la simulation de modèles individu-centrés sur GPU. En effet, dans ce type d'approche, la quantité de données à traiter (le nombre d'individus principalement) varie au cours du temps, ce qui, sur CPU, nécessite l'allocation dynamique de mémoire. Malheureusement, ce mécanisme (uniquement disponible sur les dernières générations de GPUs) est assez contraignant à mettre en œuvre sur GPU et il est donc nécessaire de surdimensionner les tableaux de données arbitrairement pour anticiper la création dynamique d'individus. Si besoin il peut de plus être nécessaire de ré-allouer de la mémoire pour agrandir ces tableaux, ce qui implique des transferts de données très pénalisants entre la machine hôte et le GPU. En outre, nous notons que les traitements effectués sur GPUs sont, en raison de l'architecture SIMT, nécessairement effectués de manière synchrone et donc les actions d'un individu ne peuvent pas dépendre de celles des autres au cours du même cycle de simulation [D'Souza *et al.*, 2009]. Par ailleurs, il n'existe pas de travaux qui semblent réaliser des simulations mixtes, c'est-à-dire mêlant approches population et individu-centrées, de systèmes biologiques sur GPU.

1.2.2.5 Remarques générales sur les performances

En dépit de l'engouement général porté envers le mouvement GPGPU [Owens *et al.*, 2007], plusieurs études insistent sur le fait que des applications développées sur CPU peuvent être tout aussi performantes que leurs homologues sur GPU [Vuduc *et al.*, 2010; Lee *et al.*, 2010; Bordawekar *et al.*, 2010]. En effet, comme indiqué précédemment, l'architecture SIMT inhérente aux GPUs impose de nombreuses contraintes aux développeurs, et la moindre inadéquation du problème à ces contraintes peut entraîner des conséquences dramatiques en termes de performances. De ce fait, les applications développées sur GPU ont tendance à demander un effort d'attention bien plus important et donc à être beaucoup plus optimisées que leur version CPU, biaisant ainsi les comparatifs de performances. En optimisant le code développé pour CPU, notamment au niveau de l'utilisation des mémoires caches, aussi finement que celui développé à destination des GPUs, les écarts de performances entre les deux approches s'avèrent fréquemment minimes. De plus, les transferts de données entre la mémoire principale de la machine hôte et la mémoire globale du GPU sont extrêmement coûteux en temps et ont un impact très significatif sur les performances des applications. Ainsi, le choix entre des développements sur CPU ou GPU doit être mûrement réfléchi.

En sus de ces deux architectures de parallélisation que nous venons de présenter, il est possible d'avoir recours à des architectures distribuées qui permettent de répartir les traitements entre plusieurs machines indépendantes plutôt que de les paralléliser au sein d'une même machine, sur CPUs ou GPUs. Nous présentons ce type d'architecture dans la section suivante.

1.2.3 Architectures distribuées

Un système distribué est une infrastructure informatique composée d'une multitude de machines reliées entre elles par l'intermédiaire d'un réseau, pouvant être aussi bien local que distant, et qui leur permet de communiquer et de coordonner leurs actions. Chacun des éléments de ce réseau constitue un nœud et est défini comme ayant sa propre mémoire locale. En effet, à la différence des architectures parallèles multicœur et multiprocesseur ou des architectures graphiques, les systèmes distribués ne possèdent pas de mémoire partagée mais une mémoire dite distribuée, c'est-à-dire répartie entre les différents nœuds composant le système. Les transferts de données entre ces nœuds sont donc nécessairement réalisés par l'intermédiaire du réseau qui les interconnecte. Cette communication est basée sur un mécanisme d'envoi de messages. La figure 1.10 présente une architecture distribuée, qui se différencie fortement de l'architecture parallèle multicœur et multiprocesseur présentée sur la figure 1.2.

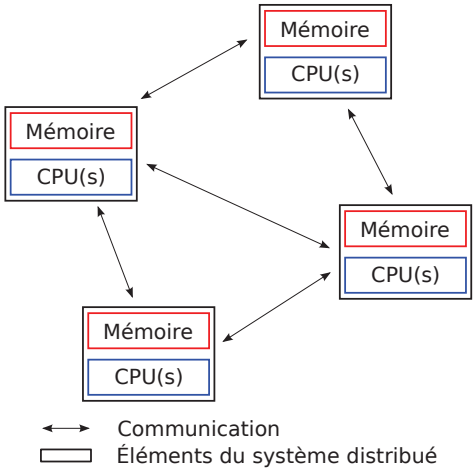


Figure 1.10 – **Exemple d'architecture distribuée** – Ce schéma représente une architecture distribuée composé de quatre nœuds individuels, chacun pouvant être un ordinateur personnel, un serveur, un téléphone portable ou tout autre matériel informatique. Il met en évidence l'absence de mémoire partagée ainsi que les communications réseau survenant entre les nœuds.

1.2.3.1 Architecture matérielle et modèle de programmation

Deux types d'infrastructure matérielle peuvent être utilisés afin d'effectuer du calcul distribué :

la grappe de serveurs, ou en anglais *cluster*, est un ensemble de machines homogènes, localisées au même endroit.

la grille de calcul est, contrairement aux *clusters*, constituée d'un ensemble de machines hétérogènes (ordinateurs personnels, téléphones portables, *clusters*) délocalisées, pouvant être réparties partout dans le monde.

Quel que soit le type d'infrastructure utilisé, les communications entre nœuds sont généralement assurées par un mécanisme de passage de messages [Padua, 2011]. L'un des principaux

protocoles définissant ce mécanisme se nomme *Message Passing Interface* (MPI) [Snir *et al.*, 1995]. Implémenté dans de très nombreux langages et supporté par une multitude d'architectures, il s'est imposé comme un standard de communication au sein des architectures distribuées. Notons que ce standard est également parfois utilisé pour gérer les communications au sein d'architectures parallèles à mémoire partagée. MPI est le successeur du logiciel *Parallel Virtual Machine* (PVM) qui permet d'assembler un ensemble d'ordinateurs hétérogènes en un supercalculateur virtuel [Geist, 1994]. Malgré son remplacement par MPI, PVM est cependant toujours utilisé lors du développement de systèmes distribués.

Le standard MPI, mais également le logiciel PVM, sont des moyens de communication de plus haut niveau que les classiques *sockets* réseaux et se veulent plus efficaces pour établir des communications au sein d'un système distribué. Les *sockets* constituent un procédé de communication plus général permettant la communication locale au sein d'une machine ou distante au travers d'un réseau de machines. Ce procédé se situe à un bas niveau d'abstraction. En effet, pour établir un dialogue avec une machine, il est nécessaire de connaître son adresse IP ainsi que le numéro de port à utiliser. En outre, il est à la charge du développeur de mettre en place un protocole de communication, c'est-à-dire de définir le langage commun reconnu par les machines constituant le réseau, d'établir des codes d'erreur... En proposant une abstraction de communication de plus haut niveau, MPI et PVM offrent des services plus simples à mettre en œuvre (abstraction des adresses IP par des identifiants, protocole d'écriture et de lecture de messages...). En outre, ils sont capables d'utiliser la mémoire partagée au sein d'une même machine afin d'optimiser les communications locales. Globalement, MPI est aux *sockets* réseaux ce qu'est OpenMP aux *threads* POSIX.

Nous présentons dans la section suivante une série d'exemples utilisant une architecture distribuée afin d'améliorer les performances de simulations de systèmes biologiques. Nous expliquons brièvement le principe de fonctionnement de chacun d'entre eux et relevons les observations pertinentes faites par les auteurs de ces travaux concernant les performances qu'ils obtiennent.

1.2.3.2 Exemples d'applications à la biologie

Exemples basés sur une approche population-centrée

Les travaux [dos Santos *et al.*, 2004] présentent un modèle déterministe mêlant EDPs et EDOs pour étudier l'activité électrique au sein des tissus cardiaques. En raison de la charge de calcul qu'entraîne la résolution des équations composant le modèle, les auteurs proposent de distribuer les différents traitements grâce à un *cluster* de 16 nœuds munis chacun d'un processeur mono-cœur afin d'optimiser les performances de leur simulation. La communication entre les nœuds composant le *cluster* est assurée grâce au standard MPI présenté brièvement précédemment. Leur implémentation consiste à découper l'ensemble du domaine étudié et à le répartir entre les processeurs. Les auteurs proposent des algorithmes de résolution numérique spécifiques afin de minimiser les communications superflues entre chacun des nœuds du *cluster*. Ils observent que plus le ratio entre temps de calcul et temps de communication est élevé, plus l'approche distribuée apporte un gain de performance

important. Pour réduire significativement les temps de communication, les auteurs envisagent d'utiliser un réseau plus rapide.

D'autres travaux utilisent des modèles stochastiques à base d'EDOs probabilistes afin de modéliser les réactions chimiques intervenant lors de la transcription de gènes [Burrage *et al.*, 2003]. Les simulations basées sur des modélisations stochastiques nécessitent, de par leur caractère aléatoire, d'être répétées un très grand nombre de fois afin de pouvoir calculer des données statistiques (moyenne, variance...) sur les solutions obtenues. Pour cela, les auteurs de ces travaux utilisent une architecture distribuée afin de répartir toutes ces répétitions indépendantes entre différentes machines appartenant à une grille de calcul. Les simulations sont attribuées aux machines clientes par l'intermédiaire d'un serveur centralisant les résultats. Grâce à l'indépendance des répétitions, les performances offertes par cette application distribuée sont très intéressantes. Le peu d'échanges entre les unités de calcul permet de maximiser le ratio entre temps de calcul et temps de communication.

Exemples basés sur une approche individu-centrée

L'un des plus célèbres exemples de simulation distribuée dans le domaine de la biologie est le projet *Folding@home* initié par l'université de Stanford [Beberg *et al.*, 2009]. Il consiste à simuler le repliement des protéines [8] afin de comprendre comment se déroule ce phénomène et de pouvoir prévenir les maladies qui lui sont liées telles que la maladie de Creutzfeldt-Jakob, la maladie d'Alzheimer ou encore l'encéphalopathie spongiforme bovine (couramment appelée maladie de la « vache folle »). Pour ce faire, *Folding@home* utilise un modèle de dynamique moléculaire pour étudier la trajectoire de chaque atome composant les protéines pour une position initiale, une vitesse et une température données. Face au nombre colossal de simulations nécessaires à l'obtention de résultats significatifs, le projet propose une architecture de type clients-serveurs où chaque client est responsable de l'exécution d'une ou plusieurs simulations indépendantes qui lui sont confiées par des serveurs à l'université de Stanford. Il utilise une grille de calcul afin de pouvoir bénéficier de la puissance de calcul offerte par les matériels informatiques hétérogènes du monde entier (CPUs, GPUs, console Playstation 3, *clusters*...) et ne pas être limité à une grappe de serveurs. Grâce à l'utilisation du standard MPI, chaque client de *Folding@home* est capable d'utiliser l'ensemble des CPUs mis à disposition par les architectures multicœur et multiprocesseur, mais sans profiter particulièrement de leurs mémoires partagées.

Un deuxième exemple de simulation biologique distribuée peut être trouvé dans [Nugala *et al.*, 1998] où les auteurs simulent le comportement d'une colonie de fourmis par l'intermédiaire d'une grille de calcul composée de stations de travail de l'université de l'Utah. L'ensemble de la colonie est répartie entre les différentes stations composant la grille. Les auteurs proposent deux méthodes, toutes deux basées sur une architecture de type clients-serveur. Leur principale différence réside dans la manière, statique ou dynamique, dont est effectué l'équilibrage de charge entre les différentes stations de travail du système distribué. Les auteurs mesurent différentes données pour évaluer les performances de leur système et notamment le gain de performance et les temps de communication en fonction du nombre de

8. Il s'agit d'une modification de la forme et de la structure des protéines leur permettant de devenir fonctionnelles.

stations de travail utilisées. Ils remarquent que les phases de communication entre les clients et le serveur sont coûteuses et absolument non négligeables : selon le paramétrage du système, plus de la moitié du temps de simulation peut être revendiquée par les communications réseaux.

Nous présentons un troisième et dernier exemple, détaillé dans [Rodin *et al.*, 2011], où les auteurs simulent des phénomènes de type réaction-diffusion, notamment au sein d'un vaisseau sanguin. Contrairement aux approches individus-centrées classiques, ces travaux adoptent le paradigme multi-interactions brièvement présenté dans la section 1.1.2.1. Cela signifie qu'ils individualisent les interactions liant les entités plutôt que les entités elles-mêmes. Bien que les interactions soient moins nombreuses que les entités, cette approche reste coûteuse en temps de calcul. C'est pourquoi il peut être nécessaire de distribuer les traitements régissant ce type de simulation. Pour ce faire, les auteurs de ces travaux optent pour une architecture réseau *peer-to-peer* où chaque machine est responsable de la simulation d'une partie du domaine étudié. Au cours de la simulation, il est régulièrement nécessaire de synchroniser les différentes machines pour garantir la cohérence globale des simulations. Cette synchronisation consiste à transférer, entre chaque machine, les données se situant aux frontières des sous-domaines voisins. Bien que grâce à leur approche distribuée les auteurs observent une amélioration des performances des simulations par rapport à une approche classique, ils mentionnent également le fait que chaque communication réseau liée aux synchronisations a un coût affectant les performances des simulations.

Bilan sur les exemples

Les architectures distribuées peuvent offrir une immense puissance de calcul, car elles relient un très grand nombre de machines entre elles. Malheureusement, les communications réseaux entre elles ont un coût non négligeable pouvant dans certaines situations être rédhibitoire. En effet, si régulièrement au cours de l'exécution d'une simulation distribuée les différentes portions de la simulation doivent dialoguer entre elles pour échanger des informations, les performances vont chuter dramatiquement et le temps de communication peut rapidement devenir prépondérant face au temps de calcul pur. Notons que si les temps de communication au sein d'une architecture parallèle peuvent parfois être élevés, ils le sont d'autant plus lorsqu'il est nécessaire de parcourir un réseau, qu'il soit local ou distant. Ce problème est commun aux approches population et individu-centrées. Les travaux [dos Santos *et al.*, 2004; Nugala *et al.*, 1998; Rodin *et al.*, 2011] présentés précédemment font état de cette situation et tentent de répondre au problème en proposant diverses solutions adaptées aux applications étudiées. Pour cela, ils effectuent des choix spécifiques dépendant de ces applications et donc difficilement généralisables. À l'inverse, nous notons que les simulations résultant des travaux [Beberg *et al.*, 2009; Burrage *et al.*, 2003], qui étudient la distribution de sous-problèmes indépendants les uns des autres, ne semblent aucunement affectées par une potentielle chute de performances liées aux communications réseaux. De plus, nous remarquons également qu'à l'instar des architectures multicœur et multiprocesseur et de leurs homologues graphiques, aucune simulation distribuée utilisant une approche mixte ne semble présente dans la littérature.

1.3 Bilan de notre étude

Dans ce chapitre, nous avons proposé un état de l'art traitant de la simulation parallélisée (au sens large du terme) des systèmes biologiques, divisé en deux parties distinctes. La première, c'est-à-dire la section 1.1, présente les deux paradigmes majeurs de modélisation permettant de réaliser de telles simulations : les approches population et individu-centrées. Nous identifions ainsi les avantages et inconvénients de chacune d'elles et convenons que la simulation de systèmes biologiques nécessite souvent de coupler ces deux approches. En effet, lorsque plusieurs phénomènes biologiques interviennent, il est souvent pertinent d'en modéliser certains par l'intermédiaire d'une approche population-centrée alors que d'autres le seront plus efficacement par une approche individu-centrée. Face au constat qu'il est nécessaire d'optimiser les performances des simulations afin d'en accélérer l'exécution ou d'augmenter la quantité de données modélisées (nombre d'individus, dimensions du domaine...), nous proposons une seconde partie. Ainsi, la section 1.2 vise à expliquer le fonctionnement des solutions de parallélisation, sur CPUs et GPUs, et de distribution capables d'accélérer l'exécution des simulations de systèmes biologiques. Cette section conclut notre état de l'art en présentant un bilan de l'ensemble de notre étude. Tout d'abord nous rappelons l'intérêt d'une approche mixte pour la modélisation et la simulation de systèmes biologiques. Ensuite, nous poursuivons en expliquant l'intérêt que proposent les architectures parallélisées face à leurs homologues distribuées. Nous concluons enfin en introduisant notre proposition pour la simulation des systèmes biologiques.

1.3.1 Intérêt d'une approche mixte

L'objectif principal des approches mixtes consiste à exploiter pleinement les atouts offerts par les approches population et individu-centrées, tout en s'affranchissant de leurs défauts respectifs. Elles permettent ainsi de réaliser deux types distincts de modélisation des systèmes biologiques : la modélisation multi-échelles et la modélisation multi-phénomènes.

La modélisation multi-échelles consiste à modéliser le comportement d'un système à plusieurs niveaux de détail différents, tout en conservant des temps de simulation acceptables. Pour parvenir à cela, de telles modélisations couplent les approches population et individu-centrées pour modéliser un même phénomène. Ces deux approches peuvent être utilisées simultanément, notamment lorsque certaines zones nécessitent un niveau de détail élevé alors que d'autres n'en ont pas besoin, ou successivement, par exemple lorsque les modèles s'exécutent tour à tour et se paramètrent dynamiquement mutuellement. Ainsi, tout en s'exécutant relativement rapidement (grâce à l'aspect global des modèles population-centrés), les simulations multi-échelles tiennent également compte des contraintes locales apparaissant dans les systèmes biologiques (grâce aux modèles individu-centrés).

La modélisation multi-phénomènes consiste quant à elle à modéliser l'ensemble des phénomènes plus ou moins complexes interagissant au sein des systèmes biologiques. En effet, comme nous l'avons mentionné dans cet état de l'art, chaque phénomène étudié peut bénéficier avantageusement d'une modélisation à base d'approches population ou individu-

centrées, et il n'est généralement pas judicieux d'utiliser le même type d'approche pour tous les phénomènes (cf. notre exemple dans l'introduction de la section 1.1.3). Ainsi, ceux où de nombreuses contraintes locales sont susceptibles de survenir peuvent profiter du niveau de détail élevé offert par les approches individu-centrées, alors que pour les phénomènes plus globaux, il n'est pas nécessaire de prendre en compte individuellement chacun des individus d'une population, sous peine d'obtenir des temps de calcul extrêmement importants. Pour permettre ce couplage, les méthodes multi-phénomènes consistent à proposer des mécanismes permettant aux modèles décrivant les différents phénomènes d'interagir entre eux.

1.3.2 Intérêt de la parallélisation face à la distribution

Cet état de l'art nous a permis d'expliquer les concepts principaux liés à chacune des solutions de parallélisation et de distribution étudiées mais également de faire ressortir les atouts et défauts de chacune d'elles, ainsi que leur bonne ou mauvaise adéquation avec les paradigmes de modélisation individu et population-centrée couramment utilisés lors de l'étude des systèmes biologiques.

En premier lieu, nous avons mis en évidence l'existence de deux types de problèmes nécessitant chacun une solution de parallélisation ou de distribution différente [Zomaya, 2006]. Le premier type de problème requiert un très grand nombre de répétitions d'une même simulation, soit afin de calculer des indicateurs statistiques sur l'ensemble des simulations (moyenne, variance...), soit afin d'étudier l'effet que produit la variation des paramètres d'entrée de la simulation sur le résultat final (dans le cadre d'un balayage de paramètres par exemple). Le second type de problème requiert quant à lui d'effectuer une unique simulation nécessitant un nombre important de calculs afin de produire des résultats significatifs et pertinents. Cette situation se produit par exemple dans le cadre des approches population-centrées lorsque le domaine étudié est discrétisé (notamment spatialement) de manière très fine, ou dans le cadre des approches individu-centrées lorsque le nombre d'individus à simuler est très important.

La parallélisation ou distribution du premier type de problème est relativement triviale puisque chaque répétition de simulation est indépendante des autres. Ainsi, il suffit d'attribuer (statiquement ou dynamiquement) une ou plusieurs répétitions à chaque unité de calcul de l'architecture utilisée afin d'accélérer la réalisation de l'ensemble de ces simulations. Cependant, cette méthode n'est pas applicable lorsque nous considérons l'accélération d'une seule simulation. Ainsi, si nous considérons le second type de problème, il est nécessaire de découper le problème global en un ensemble de sous-problèmes, chacun pouvant être attribué à une unité de calcul différente, qu'il s'agisse d'un CPU, d'un *thread* de GPU ou d'un nœud d'une infrastructure distribuée. Ces sous-problèmes ne sont pas indépendants et chacun d'eux requiert souvent des données provenant de ses homologues. L'échange des données entre ces sous-problèmes implique de nombreuses communications entre les unités de calcul. Or, les exemples étudiés précédemment montrent que les performances d'une simulation sont directement liées à la fréquence et à la quantité de données échangées lors de ces communications. Par ailleurs, le coût de ces communications varie selon l'infrastructure de parallélisation ou de distribution utilisée. En effet, les communications sont beaucoup plus

lentes à l'intérieur d'un système distribué composé de machines reliées par un réseau internet que lorsque celles-ci sont reliées grâce à un réseau local ou encore que dans une architecture multicœur et multiprocesseur à mémoire partagée.

Bien que cet état de l'art présente à la fois la parallélisation ou distribution de plusieurs simulations indépendantes mais également d'une simulation unique, nous ne considérons plus que le second cas dans la suite de ce mémoire. Celui-ci nous semble beaucoup plus pertinent pour traiter de l'accélération de simulations de systèmes biologiques de manière générale. En effet, le premier cas n'est applicable que dans un nombre très restreint de situations telles que le balayage de paramètres ou l'observation de résultats statistiques. De plus, une fois une simulation parallélisée ou distribuée, il est tout à fait envisageable de la répéter de nombreuses fois, de manière séquentielle. Ainsi, nous insistons sur le fait que, dans la suite de ce mémoire, nous nous focalisons uniquement sur la situation où une unique simulation est suffisante et ne considérons plus celle où plusieurs simulations indépendantes sont nécessaires. Comme nous l'avons indiqué précédemment, ce type de parallélisation est celui qui nécessite le plus de communications entre les unités de calcul. De ce fait, nous jugeons l'approche distribuée totalement inadaptée pour traiter cette sorte de problème. En effet, selon nous, l'approche distribuée est beaucoup plus adaptée à l'exécution de plusieurs simulations indépendantes, une situation minimisant grandement le besoin de communication. Pour cette raison, dans la suite de ce mémoire, nous nous focalisons sur l'utilisation d'architectures parallèles (aussi bien orientées CPUs que GPUs) où toutes les unités de calcul sont présentes au sein d'une même machine et peuvent communiquer entre elles beaucoup plus rapidement.

1.3.3 Parallélisation d'approches mixtes

L'étude des architectures parallélisées utilisées pour la simulation des systèmes biologiques met en évidence une certaine affinité entre le type de modèle (population ou individu-centré) mis en œuvre et l'architecture utilisée. En effet, en raison de leur architecture polyvalente, les CPUs nous semblent tout à fait adaptés pour paralléliser tout type de simulations, qu'elles utilisent des approches population ou individu-centrées. Leur principal inconvénient provient du faible nombre d'unités de calcul qu'ils mettent à disposition face aux GPUs. À l'inverse, cette étude a également tendance à montrer que les GPUs sont à privilégier pour la parallélisation de simulations population-centrées plutôt qu'individu-centrées. Cela est principalement dû à leur architecture SIMT ainsi qu'à la difficulté de mise en œuvre de mécanismes d'allocation dynamique de mémoire sur GPU et à l'obligation d'effectuer les traitements de manière synchrone. Les exemples d'applications biologiques présentés au cours de la section 1.2 justifient ces propos. Ainsi, si nous considérons conjointement le besoin de développer des approches mixtes parallélisées, c'est-à-dire couplant modèles individu-centrés et modèles population-centrés, et nos remarques concernant l'affinité des architectures parallèles vis-à-vis des différents modèles, alors il nous semble nécessaire de proposer des solutions techniques mixtes au niveau architectural, c'est-à-dire couplant CPUs et GPUs, permettant de simuler les systèmes biologiques.

À notre connaissance, très peu de travaux mettent en œuvre des architectures parallèles mixtes, c'est-à-dire faisant participer conjointement des CPUs avec des GPUs. En effet, notre

revue de la littérature fut relativement infructueuse et seulement quelques travaux très récents semblent pertinents, tels que ceux présentés dans [Shen *et al.*, 2010] et [Hampton *et al.*, 2010] où les auteurs simulent respectivement des électrocardiogrammes et des phénomènes biochimiques. Selon nous, ce faible nombre de publications (du moins en ce qui concerne la simulation de systèmes biologiques) s'explique par le fort engouement des développeurs envers la programmation sur GPU. En raison de cet engouement, les développeurs ont tendance à délaisser les architectures multicœur et multiprocesseur. Néanmoins, comme expliqué dans la section 1.2.2.5, cet engouement n'est pas nécessairement justifié. Par ailleurs, parmi le faible nombre de travaux couplant ces deux architectures, aucun d'entre eux ne semble proposer d'approche mixte mêlant approches population et individu-centrées comme le requièrent la majeure partie des systèmes biologiques. Afin de pallier ce problème, nous proposons de mettre en œuvre un couplage entre modèles individu-centrés et modèles population-centrés pour la simulation des systèmes biologiques, capable de tirer pleinement partie des spécificités des architectures parallèles que nous avons étudiées au cours de cet état de l'art, à savoir les architectures multicœur et multiprocesseur ainsi que leurs homologues graphiques.

Chapitre 2

Modélisation et parallélisation mixtes

L'état de l'art précédent a mis en évidence l'intérêt d'utiliser des approches mixtes, c'est-à-dire couplant modèles individu-centrés et modèles population-centrés, pour simuler les systèmes biologiques. Nous avons également insisté sur la nécessité de paralléliser de telles simulations afin d'en accélérer l'exécution en exploitant au mieux les performances offertes par les architectures parallèles actuelles, qu'il s'agisse aussi bien d'architectures multicœur et multiprocesseur que d'architectures graphiques. L'étude que nous avons menée au cours de cet état de l'art nous a permis d'établir la présence de certaines affinités entre les approches de modélisation et le type d'architecture de parallélisation utilisés. Pour tirer pleinement partie de ces affinités, nous proposons dans ce chapitre de développer un couplage entre un modèle individu-centré asynchrone parallélisé sur CPU et un modèle population-centré synchrone parallélisé principalement sur GPU mais également sur CPU. Ce chapitre est organisé de la manière suivante : nous présentons en premier lieu un modèle individu-centré capable de tenir compte de la hiérarchie mémoire des architectures multicœur et multiprocesseur. Ensuite, nous proposons une méthode de parallélisation de modèles population-centrés capable d'exploiter l'ensemble des GPUs et des CPUs présents sur la machine exécutant la simulation. Enfin, pour obtenir une approche multi-modèles permettant de simuler les systèmes biologiques, nous détaillons le couplage de ces deux approches en présentant les algorithmes régissant les interactions survenant entre eux.

2.1 Modèle individu-centré asynchrone

Nous débutons ce chapitre en proposant un modèle individu-centré asynchrone parallélisé permettant la simulation de systèmes biologiques et capable d'exploiter pleinement les performances offertes par les architectures multicœur et multiprocesseur. Dans un premier temps, nous décrivons les individus peuplant notre modèle. Ensuite, nous détaillons le schéma d'ordonnancement que nous utilisons afin de les faire évoluer temporellement. Enfin, nous présentons les algorithmes que nous proposons afin de paralléliser l'exécution de ce modèle individu-centré au travers de nombreux CPUs, et ceci, en prenant soin de respecter les contraintes liées à la hiérarchie mémoire utilisée par ces architectures.

2.1.1 Description de nos entités autonomes

Notre modèle individu-centré est peuplé par de nombreuses entités autonomes (plusieurs dizaines ou centaines de milliers), chacune représentant un individu du système biologique étudié. Chacune de ces entités est modélisée par une structure informatique regroupant les champs suivants :

- un ensemble de variables qui définissent l'état individuel de l'entité à un instant donné,
- une fonction d'activation qui décrit le comportement de l'entité.

La fonction d'activation est responsable de l'évolution des variables de l'entité (et éventuellement des autres entités et de l'environnement) au cours du temps. Elle est appelée systématiquement dès lors que l'entité est exécutée, c'est-à-dire lorsque le comportement de l'individu qu'elle modélise est simulé. Cette fonction est classiquement scindée en trois étapes [Wooldridge, 2009] :

- perception : l'entité perçoit son environnement,
- décision : grâce aux informations perçues, l'entité décide de sa future action,
- action : l'entité applique sa décision.

Pour illustrer ces propos, nous pouvons considérer un système modélisant le déplacement d'espèces biochimiques en solution dans une volume réactionnel, où chaque molécule est représentée individuellement par une entité. Dans cet exemple, les variables propres d'une entité décrivent sa position, sa taille ou encore la liste des molécules voisines. La fonction d'activation définit quant à elle le déplacement d'une molécule (phase d'action) et également les mécanismes de détection et de gestion de collisions (phases de perception et de décision). La manière dont sont exécutées les entités de notre modèle individu-centré est déterminée par un ordonnanceur dont le fonctionnement est détaillé dans la section suivante.

2.1.2 Schéma d'ordonnancement

Notre modèle individu-centré est piloté par des règles d'ordonnancement qui garantissent la cohérence temporelle des simulations. Ces règles déterminent et contrôlent, par l'intermédiaire d'un ordonnanceur, la manière dont le temps s'écoule au sein de nos simulations, et également la manière dont sont exécutées les entités peuplant notre modèle. Cette section a pour objectif de détailler les mécanismes principaux de cet ordonnanceur. Elle se divise en deux parties. La première présente brièvement le paradigme de modélisation du temps auquel nous avons eu recours, tandis que la seconde introduit le fonctionnement de notre ordonnanceur.

2.1.2.1 Modélisation du temps

La simulation est l'activité d'étudier le comportement d'un système, au cours du temps, par l'intermédiaire de son modèle, c'est-à-dire de son abstraction. Il est donc primordial de définir la manière dont nous allons représenter la notion de temps au sein de notre modèle. Plusieurs paradigmes de modélisation temporelle s'offrent à nous [Zeigler *et al.*, 2000] :

La modélisation à temps discret subdivise le temps en une succession d'intervalles, dont la longueur peut être variable ou constante. L'état du système est ainsi défini uniquement à certains instants, c'est-à-dire entre chaque intervalle.

La modélisation à temps continu considère le temps comme un phénomène continu pouvant prendre un nombre infini de valeurs. L'état du système peut être déterminé à tout instant t réel (contrairement à une représentation discrète).

La modélisation à événements discrets fait abstraction du temps. La notion de temps est remplacée par celle d'événement. Ainsi, l'évolution de la simulation est définie par un enchaînement d'événements prédéterminés.

Modéliser le temps de manière continue implique de pouvoir exprimer l'état global du système à tout instant t réel. Or, cela est uniquement envisageable lorsque le modèle utilisé est suffisamment simple, comme par exemple un système d'équations différentielles dont la solution analytique est connue. Malheureusement, en raison du grand nombre d'interactions pilotant les systèmes biologiques et de la complexité qu'elles entraînent, ce paradigme de modélisation temporelle est difficilement applicable à de tels systèmes.

Alternativement, l'approche basée sur des événements discrets nécessite de déterminer, à chaque occurrence d'un événement, la date du prochain. Pour cela il est nécessaire de calculer tous les événements pouvant survenir au sein du système, de ne retenir que le plus proche dans le temps et de réitérer ce processus tout le long de la simulation. Ces événements correspondent généralement aux interactions liant les composants du système étudié. Cette approche est particulièrement intéressante lorsque les événements surviennent peu fréquemment. En effet, elle évite de discrétiser le temps en une multitude d'intervalles où aucun événement n'a lieu, et donc s'affranchit de tout calculs inutiles. Or, il paraît évident que la fréquence d'occurrence des événements au sein des systèmes biologiques, complexes par nature, sera élevée. C'est pourquoi nous orientons notre choix vers une modélisation à temps discret.

Une telle discrétisation du temps nous semble proposer une approche plus polyvalente et pertinente pour notre modèle que les deux méthodes précédentes. En effet, subdiviser le temps en une multitude d'intervalles permet de simplifier la résolution du problème étudié, et ce quel que soit le problème. Suite à cette discrétisation, le déroulement d'une simulation consiste à faire évoluer le modèle pas à pas, d'un instant courant à un instant futur, autant de fois que nécessaire. Notons qu'afin d'assurer la convergence des algorithmes pilotant la simulation, il est primordial de choisir un intervalle de temps adapté au phénomène modélisé.

Ainsi, nous choisissons de modéliser l'évolution temporelle de nos simulations par l'intermédiaire d'une succession de cycles de durée fixe Δt, au cours desquels l'ensemble des

entités du modèle est exécuté. Chacune d'elles est exécutée exactement une fois durant chaque cycle, ce qui garantit la cohérence temporelle de l'ensemble de la simulation : à chaque cycle, toutes les entités ont vu le temps progresser de la même durée depuis le début de la simulation. La manière dont ces entités sont exécutées est déterminée par un schéma d'ordonnancement qui permet d'arbitrer et de contrôler le déroulement des simulations. Ce schéma est présenté dans la section suivante.

2.1.2.2 Ordonnancement des entités

Notre modèle individu-centré est constitué de plusieurs entités autonomes en interaction, évoluant pas à pas dans un environnement commun. Deux types d'ordonnancement sont à notre disposition afin de piloter leur exécution à l'intérieur d'un cycle de simulation :

L'ordonnancement synchrone où les actions de chacune des entités sont déclenchées au même instant logique, c'est-à-dire que les calculs de leurs comportements respectifs sont basés sur un état commun de l'environnement (l'état courant à l'instant t). L'ordre dans lequel ces calculs sont réalisés n'a aucune influence sur l'état futur de la simulation (à l'instant $t + \Delta t$). Le calcul de ce dernier correspond au cumul des actions que chacune des entités a effectuées au cours de l'intervalle de temps courant.

L'ordonnancement asynchrone où contrairement à l'ordonnancement synchrone, les actions de chacune des entités sont déclenchées successivement et de ce fait, se conditionnent entre elles. Plutôt que d'être cumulables, ces actions sont concurrentes et sont liées par une relation de causalité. En effet, l'impact des actions d'une entité est immédiatement pris en compte par l'entité suivante. En raison de cette relation de causalité, le calcul de l'état futur est dorénavant dépendant de l'ordre de calcul des différentes entités et ne correspond pas simplement au bilan des actions exécutées au cours du cycle actuel.

Afin d'illustrer le fonctionnement des approches synchrones et asynchrones, nous proposons deux exemples. Le premier, présenté sur la figure 2.1 consiste à étudier la diffusion de deux espèces chimiques au sein d'un environnement représenté par un maillage en deux dimensions. À l'instant courant t, chacune des deux espèces occupe une case distincte du maillage. À l'issue de l'intervalle Δt, c'est-à-dire à l'instant futur $t + \Delta t$, les deux espèces ont diffusé (horizontalement et verticalement) dans leurs voisinages respectifs, et notamment, dans la case centrale où elles se sont mélangées. Ainsi, l'évolution du contenu d'une case (et donc plus largement, du maillage global) au cours d'un intervalle de temps Δt, consiste à sommer les quantités ayant diffusé depuis et vers cette case. L'ordre des calculs n'influence donc pas le résultat final puisqu'il s'agit simplement d'un cumul de quantités. De ce fait, l'approche synchrone est particulièrement adéquate pour simuler ce phénomène.

Le second exemple, illustré sur la figure 2.2, porte sur l'étude d'une réaction bimoléculaire au sein d'un système de type réaction-diffusion. Ce type de réactions survient lorsque uniquement deux réactifs se rencontrent. En effet, les déplacements induits par le phénomène de diffusion aboutissent régulièrement à de multiples collisions entre molécules, pouvant donner lieu à des réactions bimoléculaires. À l'issue d'un tel déplacement, une molécule

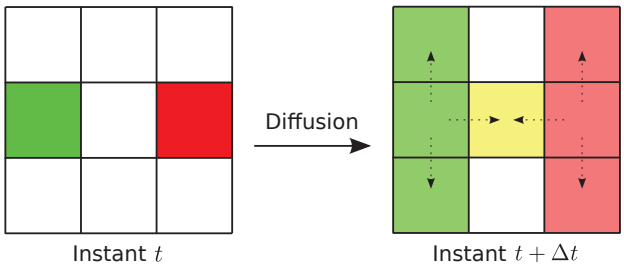


Figure 2.1 – **Exemple de modélisation synchrone** – Cette figure représente la diffusion en deux dimensions de deux espèces chimiques, respectivement symbolisées en vert et en rouge, au cours d'un intervalle de temps Δt. Elle met en évidence le fait que l'état d'une case correspond au cumul des influences provenant de son voisinage.

est susceptible de percuter simultanément plusieurs réactifs potentiels. Parmi ces collisions, seulement une peut engendrer une réaction bimoléculaire. En effet, cette dernière va modifier irréversiblement l'état du système en consommant les deux réactifs qui ne pourront par conséquent pas participer à de futurs réactions. Ainsi, il est primordial que les prochaines réactions tiennent compte de ce nouvel état global. Les actions d'une molécule sont donc conditionnées par celles des autres et ne peuvent pas se cumuler comme dans l'exemple de la figure 2.1. Dans cette situation, les méthodes asynchrones sont plus adaptées que leurs homologues synchrones car elles permettent de prendre immédiatement en compte les actions de chaque molécule.

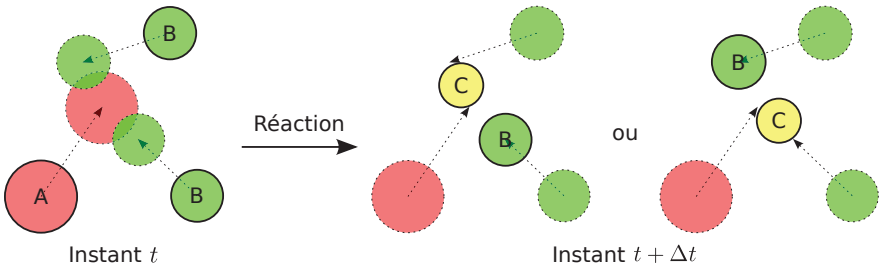


Figure 2.2 – **Exemple de modélisation asynchrone** – Cette figure représente un cycle de simulation d'un système de type réaction-diffusion. Deux espèces chimiques (A et B) diffusent jusqu'à se rencontrer et potentiellement réagir selon le schéma réactionnel A + B → C. Dans la situation présentée, une molécule A rencontre simultanément deux molécules B mais ne peut évidemment réagir qu'avec l'une d'elles. De ce fait, il est nécessaire d'utiliser une approche asynchrone pour arbitrer la situation car chaque réaction va irréversiblement modifier l'état du système.

Les deux exemples précédents mettent en évidence le fait qu'un ordonnancement synchrone peut uniquement être mis en place lorsque la modélisation du système étudié peut se résumer à un cumul d'influences prédéterminées comme sur la figure 2.1. Or, nous ne pouvons pas garantir cette condition pour l'ensemble des systèmes biologiques. Nous choisissons donc d'utiliser un ordonnancement asynchrone pour piloter l'exécution de nos entités. En outre, la littérature semble indiquer que les ordonnancements asynchrones sont plus appropriés que leurs homologues synchrones dans le cadre des approches individu-centrées [Caron-Lormier *et al.*, 2008; Cornforth *et al.*, 2005]. En effet, en prenant immédiatement en compte les actions des entités de notre modèle, un ordonnancement asynchrone permet de gérer des situations

conflictuelles qu'un ordonnancement synchrone ne saurait résoudre efficacement, hormis en affinant le pas de temps des simulations (cf. figure 2.2).

Cependant, l'utilisation d'un tel schéma d'ordonnancement nécessite de prendre quelques précautions. En effet, contrairement à l'approche synchrone, le choix de l'ordre d'exécution des entités au sein d'un cycle est lourd de conséquences. Ainsi, afin d'éviter l'apparition de biais dans la simulation, il est primordial de s'assurer que l'ordre d'exécution des entités n'est pas identique à chaque cycle de simulation. Pour remédier à cette problématique, notre laboratoire a développé la notion d'ordonnancement asynchrone et chaotique [Harrouet, 2000]. Ce mécanisme, illustré sur la figure 2.3, permet d'éliminer les biais qu'entraînerait un ordonnancement fixe préétabli [Desmeulles *et al.*, 2009; Michel *et al.*, 2001]. Pour cela, il réordonne l'ensemble des entités entre tous les cycles, par l'intermédiaire de tirages aléatoires sans remise. Le fait que notre modèle utilise une discrétisation temporelle implique que tous les instants séparés par une durée plus courte que celle du pas de temps sont considérés indiscernables. Les aléas provoqués par notre ordonnancement chaotique au sein d'un pas de temps représentent cette indiscernabilité entre les dates des événements. En choisissant la précision de la discrétisation, l'expérimentateur accepte ainsi l'indiscernabilité des événements survenant au cours d'un pas de temps. Si ces aléas ne sont pas acceptables, cela signifie que les dates précises des événements sont déterminantes pour le modèle, et donc que la discrétisation choisie est trop grossière. Notons que la convergence et la stabilité de ce schéma d'ordonnancement ont été formellement validées dans le cadre de la résolution de systèmes différentiels dans [Redou *et al.*, 2005, 2010].

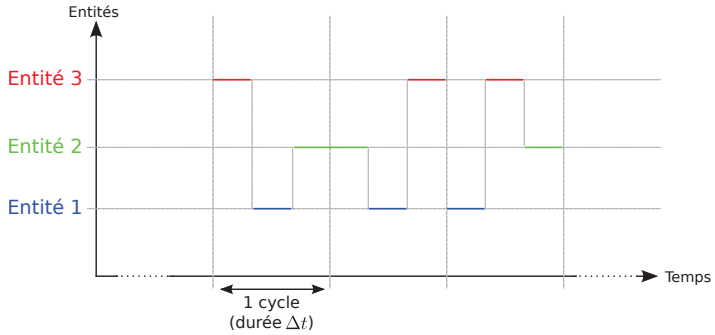


Figure 2.3 – **Ordonnancement asynchrone et chaotique** – Cette figure illustre l'ordre d'exécution de trois entités suivant le schéma des itérations asynchrones et chaotiques. Trois cycles de simulation (de durée fixe Δt) sont représentés. L'ordonnanceur exécute chaque entité exactement une fois à chaque cycle puis les réordonne de manière aléatoire au début du suivant.

2.1.3 Ordonnancement parallèle

Les systèmes biologiques sont constitués d'un très grand nombre d'entités liées par une multitude d'interactions. De ce fait, il n'est pas rare de devoir exécuter plusieurs milliers d'entités au sein d'un cycle de simulation. Qui plus est, en fonction de la durée du phénomène biologique modélisé et de la durée du pas de temps utilisé, certaines simulations peuvent nécessiter d'exécuter un très grand nombre d'itérations. Considérons par exemple

la modélisation de phénomènes liés à la coagulation du sang [Kerdélo, 2006]. En raison de l'échelle microscopique (aussi bien temporelle que spatiale) de ces phénomènes, leur modélisation exige l'utilisation de pas de temps extrêmement faibles (de l'ordre de la nanoseconde) afin de fournir des résultats suffisamment précis. Or, la durée de certains de ces phénomènes peut atteindre vingt à trente minutes, ce qui implique d'effectuer plusieurs billions d'itérations pour les simuler complètement. Ainsi, ces deux contraintes (grand nombre d'entités et d'itérations) entraînent des simulations extrêmement coûteuses, aussi bien en temps de calcul qu'en quantité de mémoire utilisée. Afin de minimiser ce problème, nous proposons un modèle individu-centré parallélisé adapté aux architectures multicœur et multiprocesseur, et basé sur l'ordonnanceur asynchrone décrit précédemment.

En raison du grand nombre d'entités qui les composent, les simulations à base de systèmes individu-centrés sont dans l'obligation de manipuler très fréquemment d'importantes quantités de données stockées en mémoire. Ces dernières représentent l'ensemble des variables décrivant l'état individuel de chacune des entités, comme indiqué dans la section 2.1.1. Ces manipulations sont responsables de nombreux accès à la mémoire, qui ne sont pas sans conséquences sur les performances des simulations. En effet, des études menées au sein de notre laboratoire [Harrouet, 2012], ont mis en évidence l'importance du placement des données en mémoire, et plus particulièrement, l'intérêt d'utiliser habilement la mémoire cache dans le cadre des simulations individu-centrées. Nous avons présenté une explication détaillée du fonctionnement de la mémoire cache dans l'état de l'art de ce mémoire (cf. section 1.2.1.1). Cela nous a permis de mettre en évidence l'impact négatif qu'engendre des mauvaises manipulations de cette mémoire sur les performances des simulations, notamment dans les situations de faux-partage. Ainsi, pour éviter cela, les différents algorithmes que nous proposons au sein de notre modèle ont été développés en portant une attention toute particulière à l'utilisation des mémoires caches. La première partie de cette section introduit le déroulement général de nos simulations, tandis que les suivantes détaillent successivement l'ensemble des algorithmes pilotant notre modèle. Pour rappel, nous désignons par CPU toute unité de calcul, qu'il s'agisse aussi bien d'un processeur, d'un cœur physique, que d'un cœur logique obtenu grâce à la technologie SMT.

2.1.3.1 Déroulement des simulations

Les architectures parallèles permettent d'effectuer plusieurs traitements de manière simultanée (un pour chaque CPU disponible). Dans le cadre de notre modèle individu-centré, ce type d'architecture nous donne l'opportunité d'exécuter plusieurs entités parallèlement plutôt que séquentiellement, et donc de fortement améliorer les performances de nos simulations. Pour parvenir à cela, l'ensemble des entités à simuler est fractionné en plusieurs sous-ensembles distincts de taille similaire. Chaque sous-ensemble est confié à un *thread* responsable de l'exécution de toutes les entités qu'il contient. Chacun de ces *threads* est attaché de manière permanente à un CPU distinct afin que l'ordonnanceur du système d'exploitation ne provoque pas de migrations intempestives de ces tâches. Ceci provoquerait en effet la recopie des données accédées par les *threads* entre les diverses mémoires caches, ce qui nuirait aux performances.

Chaque CPU maintient deux listes d'entités. La première rassemble les entités n'ayant

pas encore été exécutées pendant le cycle courant. La seconde, quant à elle, regroupe les entités traitées auparavant. Ainsi, au début de chaque cycle de simulation, la première liste est complètement remplie alors que la seconde est vide. Dès qu'une entité a été exécutée, elle est transférée de la première liste vers la seconde. Lorsqu'un cycle s'achève, et donc que le contenu des deux listes a été inversé, les listes sont permutées et le prochain cycle peut débuter.

La fin de chaque cycle de simulation est marquée par une barrière de synchronisation. Cette dernière permet de coordonner l'ensemble des CPUs entre eux afin de garantir la cohérence globale de la simulation. Elle assure que chaque CPU a terminé d'exécuter l'ensemble des entités dont il est responsable et par conséquent, que la simulation peut se poursuivre. En raison des attentes qu'elles engendrent, les barrières de synchronisation ont un impact négatif sur les performances globales mais, malheureusement, elles sont nécessaires à la cohérence des simulations. La figure 2.4 résume le déroulement d'une simulation et illustre la présence de la barrière de synchronisation. Dans cette figure, et dans les suivantes, nous qualifions de principal le CPU qui exécute le premier *thread* de l'application (existant implicitement au démarrage du processus). Il n'est en rien différent des autres et nous l'avons choisi arbitrairement pour orchestrer l'ensemble des traitements. Dans la section suivante, nous proposons un mécanisme d'équilibrage de charge dynamique minimisant l'impact de telles barrières.

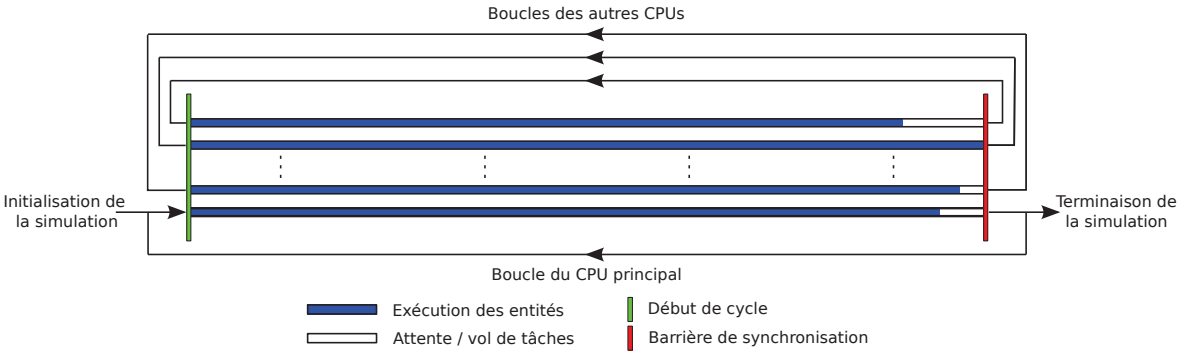


Figure 2.4 – **Déroulement d'une simulation** – Ce schéma illustre le déroulement d'une simulation sur plusieurs CPUs. Il met notamment en évidence les phases d'exécution des entités mais également les phases d'attente entraînées par la présence d'une barrière de synchronisation.

2.1.3.2 Équilibrage de charge dynamique

Principe du vol de tâches

Nous expliquions dans la section précédente que chaque cycle de simulation est ponctué par une barrière de synchronisation garantissant la cohérence temporelle de la simulation (cf. figure 2.4). Lorsque cette barrière est atteinte par un CPU, c'est-à-dire que ce CPU a terminé d'exécuter son sous-ensemble d'entités, il doit nécessairement attendre que les autres le rejoignent. Il est évident que cette attente nuit gravement aux performances du système.

Idéalement, tous les CPUs devraient achever leurs traitements respectifs simultanément, et donc aucune phase d'attente ne serait nécessaire. Malheureusement, cette situation idéale ne survient que dans les rares cas où la charge de travail de chaque CPU peut être parfaitement prédéterminée, c'est-à-dire lorsque toutes les entités effectuent exactement le même traitement. Cette condition ne peut pas être satisfaite dans le cadre de la modélisation des problèmes à forte dynamique tels que les systèmes biologiques, où le comportement de chaque entité dépend de très nombreux facteurs indéterminés.

Néanmoins, dans le but d'optimiser les performances de nos simulations, il est nécessaire de s'assurer que l'ensemble des CPUs soit pleinement utilisé et donc, d'éviter que les CPUs les plus rapides attendent passivement les plus lents. Pour cela, nous avons mis en place un mécanisme de vol de tâches qui permet à un CPU de subtiliser des entités non exécutées à l'un de ses voisins [Padua, 2011]. En allégeant ainsi leur charge de travail, les premiers CPUs aident les autres à terminer leurs tâches plus rapidement. De cette manière, nous sommes capables d'équilibrer dynamiquement la charge globale du système et ainsi, de rentabiliser au maximum les temps de calcul de l'ensemble des CPUs.

Algorithme de vol de tâches hiérarchisé

Lorsqu'un CPU effectue un vol de tâches, il accède nécessairement à des structures de données (les sous-ensembles d'entités) manipulées par d'autres CPUs. Cet accès se traduit généralement par plusieurs recopies de données dans les différents niveaux de mémoire cache du CPU effectuant le vol de tâches. Néanmoins, sous réserve que les deux CPUs partagent une partie de leurs mémoires caches, ces recopies peuvent être facultatives ou même s'avérer inutiles. Par conséquent, un vol de tâches intervenant entre deux CPUs voisins, c'est-à-dire partageant différents niveaux de mémoire cache, est beaucoup plus performant qu'un vol de tâches entre deux CPUs ne partageant aucun niveau de mémoire cache, car il ne nécessite que peu ou pas de recopies. Ce postulat nous amène à proposer une méthode de vol de tâches respectueuse de la hiérarchie des mémoires caches.

À l'initialisation de la simulation, l'ordonnanceur construit, pour chaque CPU, une liste classant l'ensemble des autres CPUs par ordre de proximité (en terme de mémoires caches). La construction de ces listes est basée sur la hiérarchie mémoire du système. Chacune de ces listes permet à un CPU de savoir avec quels autres CPUs il va interagir en priorité. Grâce à cette méthode, un CPU privilégie le vol de tâches avec ses proches voisins et donc, ceux avec lesquels il partage un maximum de niveaux de cache. L'utilisation de ces listes nous permet ainsi de limiter les recopies de lignes de cache les plus pénalisantes. La figure 2.5 illustre, avec un exemple, la construction de l'une de ces listes ainsi que cette notion de proximité des CPUs.

Une solution alternative est proposée dans [Harrouet, 2012]. Elle consiste à faire abstraction de l'architecture mémoire du système lors du vol de tâches et de simplement subtiliser des entités au CPU le plus en retard, c'est-à-dire celui en possédant encore un maximum. Cette méthode, plus intuitive de prime abord, nuit aux performances des simulations. En effet, déterminer le CPU possédant le plus d'entités non exécutées implique de comparer la taille de la liste d'entités de chaque CPU, et donc de recopier inutilement de nombreuses

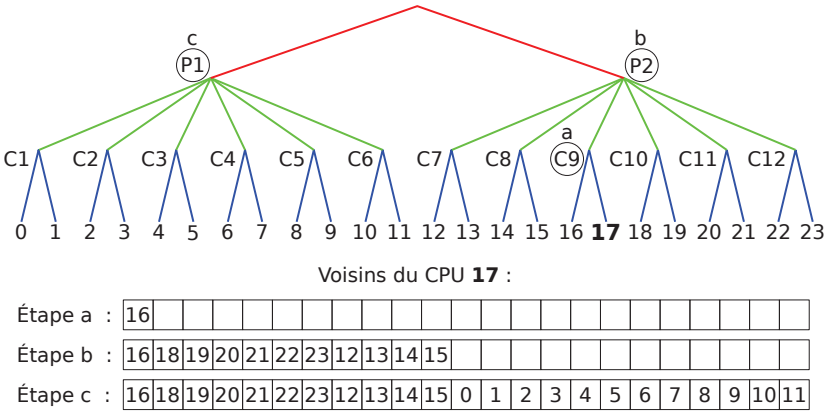


Figure 2.5 – **Exemple de vol de tâches hiérarchisé** – Ce diagramme représente sous forme d'arbre la hiérarchie mémoire introduite sur la figure 1.2. Il détaille les étapes de la construction de la liste contenant les voisins (classés par proximité décroissante) du CPU numéro 17. L'étape a ajoute le seul CPU qui partage l'ensemble de ses niveaux de cache avec le 17. Ensuite, l'étape b ajoute les CPUs partageant la même mémoire cache de niveau 3. Enfin, l'étape c complète avec les CPUs restants. Cette liste regroupe donc, de gauche à droite, les CPUs avec lesquels le vol de tâches est de plus en plus coûteux. Les noeuds nommés P_i et C_j correspondent respectivement aux processeurs et aux cœurs physiques du système. Les nœuds terminaux (numérotés de 0 à 23) représentent l'ensemble des CPUs.

lignes de cache avant de réellement débuter l'opération de vol de tâches.

2.1.3.3 Affectation des entités aux CPUs

Précédemment, nous indiquions que la totalité des entités à simuler était confiée à l'ensemble des CPUs lors de l'initialisation des simulations. Chaque CPU exécute donc les mêmes entités d'un cycle à l'autre, à l'exception des entités volées par le mécanisme d'équilibrage de charge dynamique. De ce fait, les structures de données des entités restent la plupart du temps stockées dans les mêmes mémoires caches, limitant ainsi les recopies intempestives.

Néanmoins, au cours d'une simulation, les entités du modèle sont susceptibles d'interagir les unes avec les autres. Bien que ces interactions soient uniquement définies au niveau applicatif, il est utile de les anticiper au niveau de notre modèle. En effet, le CPU responsable de l'entité à l'initiative d'une telle interaction doit recopier dans sa mémoire cache les structures de données des entités ciblées. Or, si ces dernières étaient auparavant exécutées par ce même CPU, leurs données respectives sont probablement déjà présentes et à jour dans ses mémoires caches. Ainsi aucune recopie n'est nécessaire. De plus, lorsque les entités cibles seront exécutées à leur tour, il est fort probable qu'elles soient toujours en relation avec les entités précédentes, déjà chargées dans la même mémoire cache. Ainsi, nous proposons un algorithme permettant de regrouper les entités de notre modèle par affinités.

Au cours de la simulation, chaque entité a connaissance du CPU sur lequel elle est exécutée. De surcroît, elle analyse également sur quels CPUs se trouvent les entités avec

lesquelles elle interagit. De cette manière, chaque entité est capable de déterminer quel CPU est le plus représenté au cours de ces interactions. Grâce à cette observation, les entités sont capables de changer dynamiquement de CPU pendant la simulation. Pour cela, après son exécution, une entité peut se déplacer dans la liste d'entités d'un CPU différent. Cette distribution par affinités permet de répartir dynamiquement et automatiquement les entités en fonction de leurs interactions au sein de l'application finale, bien qu'*a priori* ces interactions ne soient pas connues.

Ces interactions sont responsables d'un autre type de problèmes inhérents aux systèmes parallèles : les accès concurrents. La section suivante détaille donc ces phénomènes et présente les algorithmes que nous proposons pour les résoudre.

2.1.3.4 Gestion de la concurrence

Accès concurrents

L'une des difficultés majeures lors du développement d'un système individu-centré parallèle réside dans le fait de garantir un état cohérent pour toutes les entités et les ressources de la simulation, alors que de nombreux CPUs peuvent les manipuler simultanément. Pour assurer cette cohérence au sein du système, il est nécessaire de mettre en place un mécanisme d'exclusion mutuelle constitué de verrous (couramment appelés *mutex*) qui permettent d'arbitrer les accès concurrents à une ressource partagée entre plusieurs processus. Les principes fondamentaux d'un tel mécanisme ont été introduits dans [Dijkstra, 1965; Lamport, 1974]. L'inconvénient majeur de ces verrous est de limiter l'accès d'une ressource partagée à exclusivement un seul et unique processus, que ce soit pour une opération de lecture (pour consulter la ressource) ou une opération d'écriture (pour modifier la ressource). Or, il est tout à fait envisageable d'autoriser la consultation d'une ressource par de multiples processus conjointement, sous réserve qu'elle ne soit pas simultanément modifiée par un autre processus.

Pour pallier ce désagrément et ainsi augmenter le rendement des accès concurrents, des verrous de type *lecteurs-écrivain*, ou en anglais *readers-writer-locks*, ont été proposés par [Courtois *et al.*, 1971]. Ils garantissent un accès exclusif aux processus « écrivains », c'est-à-dire susceptibles de modifier la ressource partagée, comme dans le problème d'exclusion mutuelle original. Cependant, ils autorisent la consultation simultanée de la ressource partagée par plusieurs processus « lecteurs », sous réserve qu'aucun « écrivain » ne soit en train de la manipuler.

Ainsi, chaque entité composant notre système individu-centré est dotée de son propre verrou *lecteurs-écrivain*. Ce dernier est verrouillé en écriture dès l'instant où l'entité est exécutée par l'ordonnanceur. De cette manière, il protège l'entité d'éventuelles modifications, mais également de simples consultations, concurrentes alors qu'elle est en train d'être exécutée et donc potentiellement modifiée. Ce verrou est immédiatement libéré lorsque l'entité a terminé son action. Lorsque une entité en cours d'exécution souhaite interagir avec une seconde entité, elle demande un accès en lecture à son verrou. Cet accès sera autorisé uniquement si aucune modification de l'entité par un autre CPU n'est en cours. Néanmoins, si la première entité entreprend de modifier la seconde, elle peut demander à ce que son verrou

« lecteur » soit promu au rang de verrou « écrivain » (sous réserve qu'aucune entité ne la consulte à ce moment précis).

La mise en place de ces verrous *lecteurs-écrivain* nous permet de garantir la cohérence de la simulation vis à vis des accès concurrents. Cependant, cette solution peut être responsable de situations d'interblocage problématiques telles que les *deadlocks*.

Gestion des deadlocks

Les *deadlocks* surviennent lorsque plusieurs entités attendent que chacune d'elles libère l'accès à une ressource partagée, entraînant ainsi une situation d'attente sans fin [Padua, 2011]. La figure 2.6 présente un exemple simple de situation de *deadlock* entre deux entités.

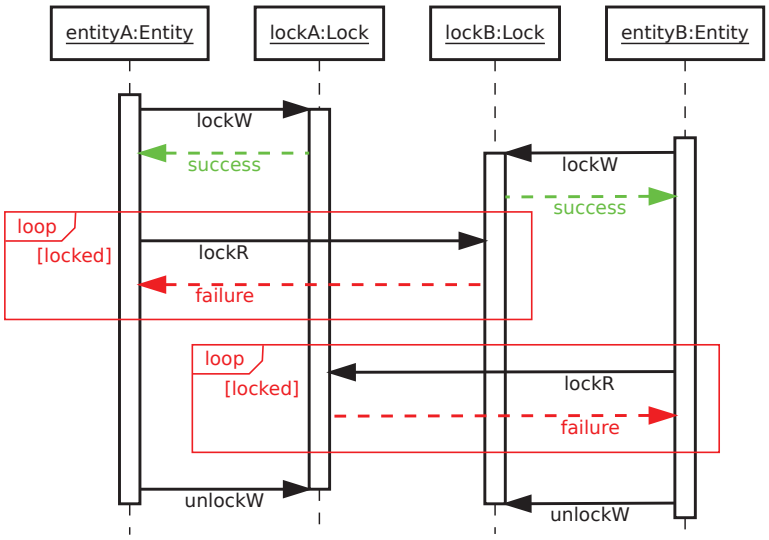


Figure 2.6 – **Exemple de situation de *deadlock*** – Ce diagramme de séquence UML illustre une situation de *deadlock* entre deux entités nommées A et B. Tout d'abord, les deux entités verrouillent leur verrou en écriture. Ensuite, l'entité A souhaite consulter l'état de B (et donc verrouiller le verrou B), ce qui entraîne une boucle d'attente bloquante (en rouge) car le verrou B est indisponible. De son côté, l'entité B fait simultanément de même avec le verrou A, ce qui crée la situation de *deadlock* entre les 2 entités. Notons que les opérations de déverrouillage ne sont donc jamais atteintes.

La gestion de ce type de problème est primordiale pour assurer le déroulement correct des simulations. Plusieurs solutions sont présentées dans la littérature et notamment dans [Isloor et Marsland, 1980] :

La prévention qui consiste simplement, dès la conception d'un programme, à rendre impossible les *deadlocks*.

L'évitement qui consiste à détecter des situations de *deadlock* potentielles, pour pouvoir les éviter avant qu'elles ne se présentent.

La détection qui consiste à déterminer, pendant l'exécution d'un programme et par l'intermédiaire d'un superviseur global, si des ressources partagées sont en situation de

deadlock, et à les débloquer dynamiquement.

Notre choix s'est porté sur une stratégie d'évitement. En effet, comme indiqué précédemment dans la section 2.1.2.2, les interactions au sein de notre modèle ne sont pas prédéterminées, et donc, nous ne pouvons pas utiliser une stratégie de prédiction. Par ailleurs, il nous paraît inadapté de mettre en place un mécanisme de détection qui nécessite une surveillance constante, et donc coûteuse, des *deadlocks*. Il nous semble donc préférable de simplement éviter l'apparition de telles situations avant qu'elles ne se produisent. Pour cela, nous transformons nos opérations de verrouillage en tentatives de verrouillage, qui échoueront immédiatement si le verrou cible est indisponible. Cette amélioration impose que toutes les décisions prises par une entité le soient de manière locale. Les modifications résultantes de ces décisions seront ainsi appliquées uniquement si toutes les tentatives de verrouillage ont abouti. Il suffit d'un seul échec pour que l'intégralité des modifications soit abandonnée, entraînant ainsi le retour de l'entité à son état d'origine. De cette manière, le déroulement de la simulation ne sera pas altéré et les *deadlocks* seront évités. Le fonctionnement de ce mécanisme de report est illustré sur la figure 2.7.

Afin de ne pas nuire à la cohérence globale de la simulation, l'exécution des entités dont les actions ont été abandonnées sera repoussée à plus tard dans le cycle de simulation courant. Néanmoins, à l'approche de la fin du cycle, cette stratégie peut entraîner un nouveau type de situations d'interblocage : les *livelocks*.

Gestion des livelocks

Contrairement aux *deadlocks*, les *livelocks* ne bloquent pas la simulation dans un état figé telle qu'une situation d'attente bloquante, mais elles empêchent tout de même la simulation de progresser [Padua, 2011]. Ces situations ont tendance à survenir lors de la résolution des *deadlocks*. L'exemple fréquemment utilisé pour illustrer ce problème est celui de deux personnes qui, par politesse, se laissent mutuellement passer pour accéder à un couloir, trop étroit pour leur permettre de le traverser ensemble. Dans cet exemple, les deux protagonistes ne sont pas réellement bloqués puisqu'ils ne cessent de se déplacer simultanément, cependant il est impossible que la situation progresse dans ces circonstances.

Dans notre cas, les entités dont l'exécution a été précédemment reportée sont probablement les seules à ne pas avoir effectué d'action durant le cycle de simulation courant. De surcroît, il est fort probable qu'elles soient réparties sur des CPUs différents. Ainsi, lors de nouvelles interactions mutuelles, leurs tentatives de verrouillage vont vraisemblablement échouer une nouvelle fois. Cet échec entraîne un nouveau report de leur exécution mais, étant donné que ces entités sont les seules à ne pas avoir été exécutées, cette situation va se reproduire indéfiniment. Il s'agit d'une situation de *livelock*. Un exemple est donné sur la figure 2.8.

Pour remédier à ce problème, notre ordonnanceur peut sous l'initiative d'une entité, reporter de nouveau son exécution, non pas au sein du cycle de simulation courant, mais au suivant. En effet, si l'exécution d'une entité échoue de façon répétée au cours d'un même cycle, il est probable que cette dernière se trouve dans une situation de *livelock*. Comme

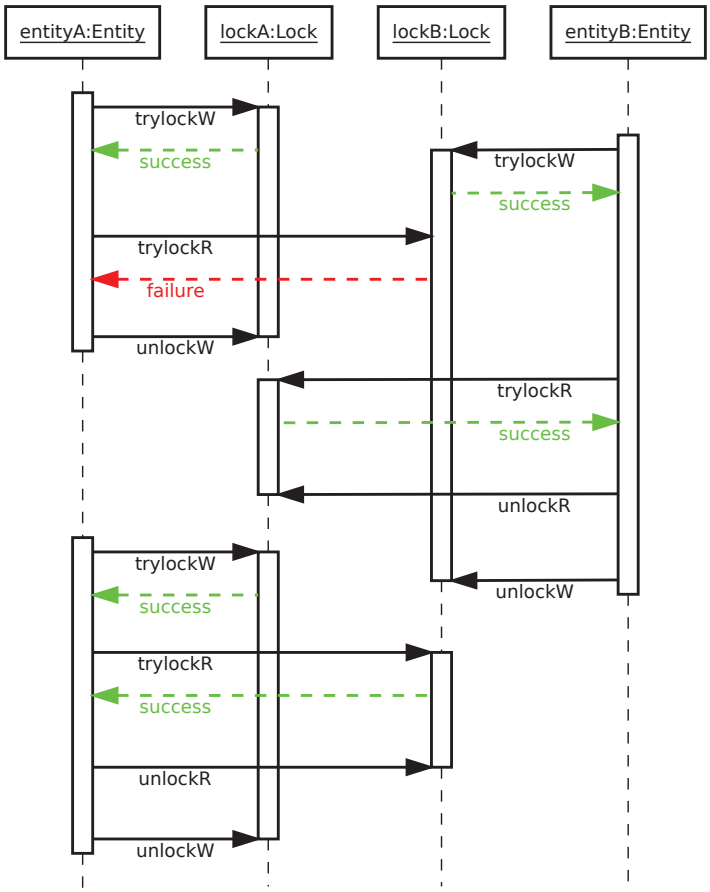


Figure 2.7 – **Résolution d'une situation de *deadlock*** – Ce diagramme de séquence UML montre la résolution de la situation de *deadlock* présentée sur la figure 2.6. Grâce à la transformation des opérations de verrouillage en tentatives de verrouillage et à notre mécanisme de report d'exécution, l'entité A (qui ne peut verrouiller le verrou B), ne va finalement pas être exécutée immédiatement. De cette manière, l'entité B va pouvoir s'exécuter normalement alors que l'entité A sera exécutée de nouveau en fin de cycle.

mentionné dans la section 2.1.2.2, nos entités sont exécutées de manière totalement aléatoire d'un cycle à l'autre. Ainsi, il est très peu probable que les entités concurrentes responsables de la situation de *livelock* (qui sont dorénavant mélangées aux autres entités) se retrouvent une nouvelle fois exécutées simultanément. Notons qu'évidemment les entités dont l'exécution a été repoussée seront exécutées deux fois durant ce cycle de simulation, ce qui évitera tout biais temporel dans la simulation.

Bilan

En dépit du fait que ces tentatives de verrouillage et ces réordonnancements semblent accroître drastiquement la charge de travail de chaque CPU, ce n'est qu'une fausse impression. En effet, étant donné que la distribution de nos entités sur les différents CPUs est réalisée en fonction de leurs affinités (cf. section 2.1.3.3), les entités en interaction seront la plupart

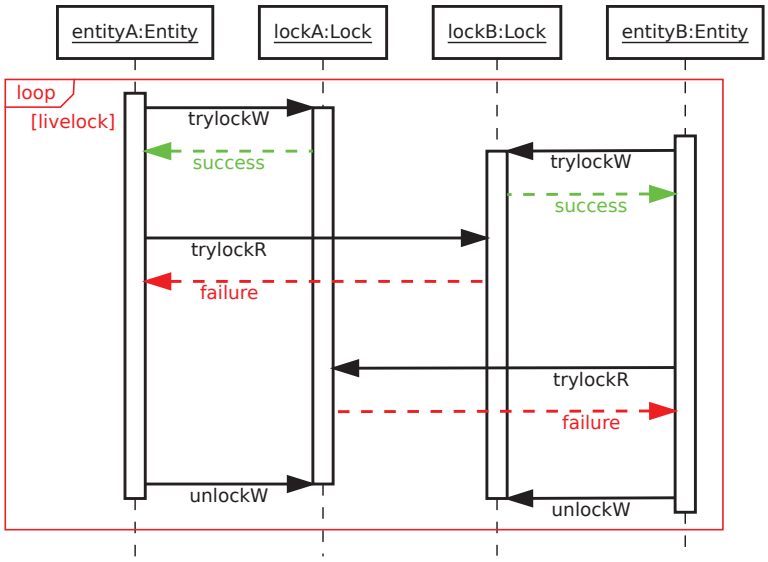


Figure 2.8 – **Exemple de situation de *livelock*** – Ce diagramme de séquence UML illustre une situation de *livelock* entre deux entités nommées A et B. Il suppose que ces deux entités sont les seules restantes avant la fin du cycle de simulation courant. Lorsque chacune des deux tentatives de verrouillage en lecture échoue, l'exécution des entités est reportée à la fin du cycle, pour éviter une situation de *deadlock*. Malheureusement, les deux entités étant les dernières à être exécutées, cette situation va se reproduire indéfiniment, entraînant ainsi une boucle (en rouge) infinie où la simulation ne progresse plus. La situation de *livelock* est atteinte.

du temps exécutées séquentiellement par le même CPU. Par conséquent, la probabilité que les tentatives de verrouillage échouent est extrêmement faible, aussi est-il très rare de devoir reporter l'exécution d'une ou plusieurs entités.

2.1.4 Bilan de notre modèle individu-centré

Au cours des sections précédentes, nous avons proposé des algorithmes permettant la parallélisation d'un modèle individu-centré asynchrone sur une architecture multicœur et multiprocesseur. Pour cela, nous avons tout d'abord expliqué l'intérêt d'ordonnancer les entités de nos simulations de manière asynchrone plutôt que, classiquement, de manière synchrone. Suite à cette présentation de notre schéma d'ordonnancement, nous avons détaillé une succession d'algorithmes visant à exploiter au mieux le parallélisme et la hiérarchie mémoire offerts par les architectures multicœur et multiprocesseur. Ces algorithmes consistent principalement à garantir la cohérence des simulations, notamment en gérant la concurrence qui peut survenir entre les entités autonomes, et également en optimisant la charge de travail attribuée à chacun des multiples CPUs utilisés.

Bien que notre proposition permette de modéliser différents phénomènes biologiques, et notamment ceux où de nombreuses contraintes locales peuvent apparaître, elle n'est pas suffisamment complète pour prétendre pouvoir simuler un système biologique dans son

ensemble. En effet, comme indiqué précédemment dans ce mémoire, la majorité des systèmes biologiques font intervenir de nombreux phénomènes variés, certains pouvant bénéficier avantageusement d'une approche individu-centrée, d'autres de leurs homologues population-centrées. Ainsi, face à cette hétérogénéité des phénomènes, nous proposons, en plus de notre modèle individu-centré asynchrone parallélisé, des méthodes permettant la simulation parallélisée de modèles population-centrés synchrones. La section suivante détaille la mise en œuvre de ces méthodes.

2.2 Modèles population-centrés synchrones

Nous avons indiqué dans l'état de l'art de ce mémoire (cf. section 1.1.1) que les approches population-centrées permettent de modéliser un phénomène biologique à un échelle globale. Ses propriétés pertinentes sont alors représentées sous la forme de variables continues caractérisant l'état global du système. L'évolution de ces variables est gouvernée par des modèles mathématiques, déterministes ou stochastiques, à base d'équations différentielles telles que les EDOs, les EDPs ou encore les EDSs. Contrairement aux modèles individu-centrés, ces dernières sont incapables de tenir compte des contraintes pouvant survenir localement au sein d'un système. Cependant, elles offrent des performances de calcul nettement supérieures.

En raison de leur dépendance au temps, mais également à l'espace, les phénomènes régissant le comportement des systèmes biologiques sont majoritairement modélisés par des EDPs. Ce postulat peut être illustré par de nombreux exemples tels que la cinétique biochimique [Purich et Allison, 2000], la dynamique des fluides [Wendt et Anderson, 2009], la morphogénèse [1] [Turing, 1990] ou encore la chimiotaxie [2] [Sleeman et Levine, 2001] où l'aspect spatial des phénomènes étudiés joue un rôle important. Nous présentons ainsi dans un premier temps le besoin d'utiliser un maillage de l'espace pour résoudre de telles équations. Ensuite, nous proposons des méthodes de parallélisation permettant d'accélérer leur résolution grâce à un ou plusieurs GPUs, potentiellement accompagnés de CPUs.

2.2.1 Maillage de l'espace

Les systèmes d'EDPs utilisés pour modéliser les phénomènes présents au sein des systèmes biologiques ne possèdent que rarement des solutions analytiques, c'est-à-dire pouvant être exprimées avec des fonctions et opérateurs de références. Ainsi, leur résolution exige nécessairement l'utilisation de méthodes numériques capables d'approximer la solution exacte, sans pour autant la formaliser :

La méthode des éléments finis est majoritairement utilisée pour résoudre des EDPs linéaires. Elle se base sur la formulation variationnelle des équations (ou forme faible) et sur l'approximation d'intégrales. Cette notion d'affaiblissement permet de résoudre

1. La morphogénèse est responsable du développement progressif des formes d'un organisme.
2. La chimiotaxie définit le mouvement d'un organisme sous l'action d'un stimulus chimique.

un système plus simplement qu'avec une forme dite « classique » en évitant certaines contraintes et en étant moins exigeant sur la régularité de la solution. Cependant, cette formulation n'existe pas pour toutes les équations aux dérivées partielles.

La méthode des volumes finis est, tout comme celle des éléments finis, basée sur l'approximation d'intégrales. Néanmoins, elle utilise la forme forte des équations (et non la forme faible décrite précédemment). Cette formulation correspond à la présentation habituelle d'un système différentiel, c'est à dire à une relation entre les différentes dérivées. Elle permet de résoudre des systèmes possédant des termes non-linéaires.

La méthode des différences finies est fondamentalement différente des deux précédentes. Considérée comme la plus abordable des trois, elle est basée sur l'approximation des opérateurs différentiels par discrétisation grâce aux développements de Taylor tronqués.

Chacune de ces méthodes nécessite une discrétisation temporelle et spatiale du problème étudié. Ceci implique la mise en place d'un maillage de l'espace permettant de fragmenter le domaine étudié en un ensemble de mailles, de taille et forme similaires ou variables. Suite à cette discrétisation spatiale, le système global est ainsi divisé en une multitude de sous-systèmes identiques. Notons que l'utilisation de telles méthodes nécessite un conditionnement précis du problème afin d'éviter les instabilités numériques dues à la propagation des erreurs d'arrondis, de troncatures ou encore de discrétisation.

La discrétisation temporelle implique que l'évolution du système consiste en une succession de cycles de simulation. Au cours de chacun de ces cycles, pour faire évoluer le système d'un instant courant t à l'instant suivant $t + \Delta t$, il est nécessaire de calculer les variations des quantités présentes à l'intérieur de chacune des mailles. Ces variations permettent de déterminer l'état futur de chacune des mailles à partir de son état courant et de celui de ces voisines. Les calculs sont effectués de manière synchrone, contrairement à ceux réalisés au sein de notre modèle individu-centré. En effet, chaque maille évolue en calculant le cumul des influences qu'elle subit entre l'instant courant et le suivant. Le nouvel état d'une maille n'est pris en compte qu'au cycle de simulation suivant et donc, l'ordre de recalcul importe peu. De surcroît, les calculs effectués au sein d'une maille ne modifient qu'elle et aucune autre : il n'y a pas de phénomènes de concurrence (contrairement à notre modèle individu-centré). Les influences que les mailles subissent décrivent les interactions les liant entre elles. Une simulation complète du système se résume donc à une succession de cycles au cours desquels chaque maille évolue d'un instant courant à un instant futur.

La discrétisation de l'espace engendrée par la mise en place d'un maillage nous pousse à structurer les données du problème sous la forme de tableaux informatiques, c'est-à-dire une succession de données similaires, dont la taille correspond au nombre total de mailles. Nous stockons chaque sorte de données dans un tableau distinct. Il existe par conséquent autant de tableaux que le problème compte de paramètres. Si nous considérons l'exemple d'un fluide décrit par ses champs de vitesse et de pression, deux tableaux seront mis en place (un pour chacun des paramètres). De plus, chacun de ces tableaux est dupliqué en deux exemplaires : un premier tableau nommé « *from* » qui contient l'état courant de chaque maille et un second nommé « *to* » qui stocke l'état futur de chaque maille, c'est-à-dire l'état résultant des calculs effectués durant le cycle de simulation en cours. Au début de chaque cycle, les tableaux « *from* » sont inversés avec les tableaux « *to* », ce qui transforme le précédent état futur

en état courant. Cette distinction entre deux états « *from* » et « *to* » permet d'assurer la cohérence des données contenues dans le maillage : les données « *from* » sont lues (pour la mise à jour de plusieurs mailles) alors que les données « *to* » sont écrites (par chaque maille elle-même).

La fragmentation des calculs qu'entraîne l'utilisation de maillages est tout à fait propice à la parallélisation du problème. En effet, le calcul de l'état futur de chaque maille est totalement indépendant du calcul de l'état futur des voisins, ce qui évite ainsi les accès concurrents à une ressource partagée et donc, les multiples synchronisations qui y sont liées. Seule une barrière de synchronisation est nécessaire à la fin de chaque cycle pour garantir la cohérence temporelle globale de la simulation.

2.2.2 Méthodes de parallélisation

Les problèmes dont la résolution nécessite la mise en place d'un maillage sont des candidats idéaux pour une parallélisation sur GPU. En effet, les calculs effectués au sein de chacune des mailles sont identiques et donc, se prêtent parfaitement à l'utilisation de l'architecture SIMT détaillée dans la section 1.2.2 de ce mémoire. Pour se conformer aux impératifs liés à cette architecture, les traitements à réaliser à l'intérieur de chaque maille sont décrits sous la forme de *kernels* (cf. figure 1.9) dont l'exécution est assurée par un ensemble de *threads* appartenant au GPU utilisé. Chacun de ces *threads* est par conséquent responsable des calculs se déroulant au sein d'une maille distincte.

Dans cette section, nous proposons trois solutions, classées par ordre croissant des performances qu'elles offrent, permettant de paralléliser la simulation de systèmes population-centrés utilisant un maillage de l'espace. La première décrit principalement le déroulement d'une telle simulation en utilisant un unique GPU. La seconde propose ensuite des techniques permettant d'assurer la collaboration entre plusieurs GPUs travaillant sur un même problème. Finalement, la troisième et dernière méthode propose de faire participer les CPUs de la machine hôte à la simulation, pour réduire la charge de travail initialement confiée aux GPUs.

2.2.2.1 Démarche mono-GPU

Cette démarche est basée sur la démarche générale présentée dans la section 1.2.2 et décrit les étapes importantes nécessaires à la simulation mono-GPU de modèles population-centrés basés sur un maillage de l'espace. Étant donné que les méthodes numériques utilisées lors de la simulation de ces modèles requièrent une discrétisation temporelle, l'évolution du système entre l'instant initial et l'instant final prend la forme d'une succession de cycles de simulation, chacun correspondant à une nouvelle exécution du *kernel* sur le maillage global. Sur une architecture mono-GPU, une simulation population-centrée peut être décrite par la démarche suivante :

1. Discrétiser le domaine étudié sous la forme d'un maillage adapté à la méthode de résolution utilisée (différences finies, volumes finis...).

2. Transférer l'intégralité du maillage initialisé (tableaux « *to* ») depuis la mémoire principale de la machine hôte vers la mémoire globale du GPU utilisé.
3. Pour chaque intervalle de temps Δt entre l'instant initial et l'instant final :
   (a) Inverser les tableaux « *from* » avec les tableaux « *to* ».
   (b) Demander au GPU d'exécuter le *kernel* décrivant le traitement à appliquer sur chacune des mailles. Les données d'entrée proviennent des tableaux « *from* » alors que les résultats sont inscrits dans les tableaux « *to* ».
4. Transférer l'intégralité du maillage contenant les résultats du traitement (tableaux « *to* »), depuis le GPU vers la mémoire principale de la machine hôte afin de pouvoir les exploiter avec le ou les CPUs.

La figure 2.9 illustre cette approche mono-GPU en mettant en évidence certaines des étapes mentionnées ci-dessus. Elle représente la boucle de simulation (effectuée par le CPU principal) permettant d'exécuter le *kernel* décrivant les traitements s'appliquant sur chacune des mailles au cours d'un cycle de simulation. Notons que seul le GPU est actif ; les CPUs de la machine hôte sont inactifs et restent disponibles pour d'autres traitements.

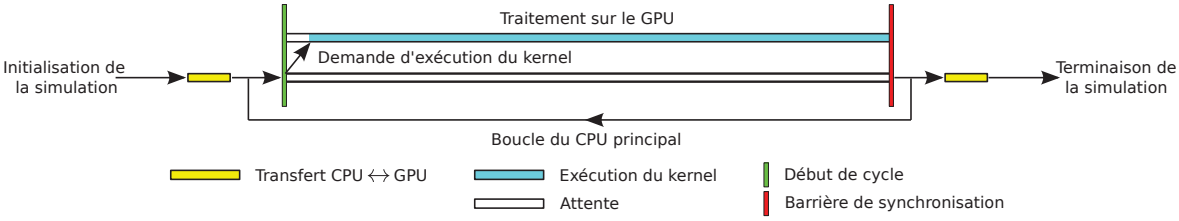


Figure 2.9 – **Déroulement d'une simulation mono-GPU** – Ce schéma illustre le déroulement d'une simulation selon la démarche mono-GPU que nous décrivons. Il met en évidence la boucle de simulation qu'effectue le CPU principal pour faire évoluer la simulation au cours du temps et également le fait que le CPU reste disponible durant l'exécution du *kernel* par le GPU.

2.2.2.2 Démarche multi-GPUs

À la manière des CPUs, il est possible d'utiliser parallèlement plusieurs GPUs. Matériellement, cette association consiste à utiliser plusieurs cartes d'extension distinctes, ou alors une carte unique contenant de multiples puces graphiques. Cela permet entre autre d'augmenter le nombre d'unités de calcul à disposition, et donc d'effectuer un traitement plus efficacement en répartissant la charge de travail à réaliser entre plusieurs GPUs. Pour exploiter une telle configuration, il est nécessaire de décomposer le maillage du système en autant de sous-maillages que la machine hôte compte de GPUs. De cette manière, le traitement de chaque sous-maillage peut être confié à un GPU distinct, chacun d'eux opérant parallèlement sur son propre sous-maillage, sans influer sur celui des autres. Ce découpage peut être réalisé de manière statique, c'est-à-dire une fois pour toute à l'initialisation de la simulation. Ceci provient du fait que les calculs effectués au sein de chacune des mailles sont similaires et que très peu de divergences sont susceptibles de survenir (hormis aux frontières du maillage). Cette homogénéité des calculs garantit l'absence de fluctuations significatives dans les temps de traitement, rendant complètement inutile une répartition dynamique du

maillage entre les différents GPUs qui évolue au cours du temps. Ce découpage est effectué selon les performances qu'offrent les divers GPUs utilisés pour la simulation : un puissant GPU se verra attribuer un sous-maillage plus conséquent qu'un GPU d'entrée de gamme. Ces performances sont constatées en instrumentant les simulations lors de leur élaboration.

Chaque GPU possède sa propre mémoire globale et, hormis la mémoire principale de la machine hôte, les GPUs ne partagent aucune mémoire commune. Pour cette raison, il est nécessaire de transférer les données du problème étudié sur chacun d'eux. Bien que chaque GPU agisse uniquement sur une portion du maillage global, il peut toutefois avoir besoin de consulter les données présentes à l'extérieur de son sous-maillage, c'est-à-dire dans un sous-maillage traité par un autre GPU. Cette situation peut par exemple se produire lorsque le calcul effectué au sein d'une maille nécessite l'accès au contenu des mailles voisines. En effet, le calcul de l'état futur d'une maille frontalière, c'est-à-dire à la jonction de deux sous-maillages, peut potentiellement nécessiter l'exploration de son voisinage, et donc la consultation du contenu de mailles appartenant à un autre sous-maillage. Ainsi, malgré le fait que chaque GPU modifie uniquement son propre sous-maillage (et non pas l'intégralité du maillage global), il lui est toutefois nécessaire de posséder une copie de l'ensemble des données du problème étudié pour assurer correctement ses traitements. La figure 2.10 illustre la division d'un maillage en deux dimensions entre deux GPUs.

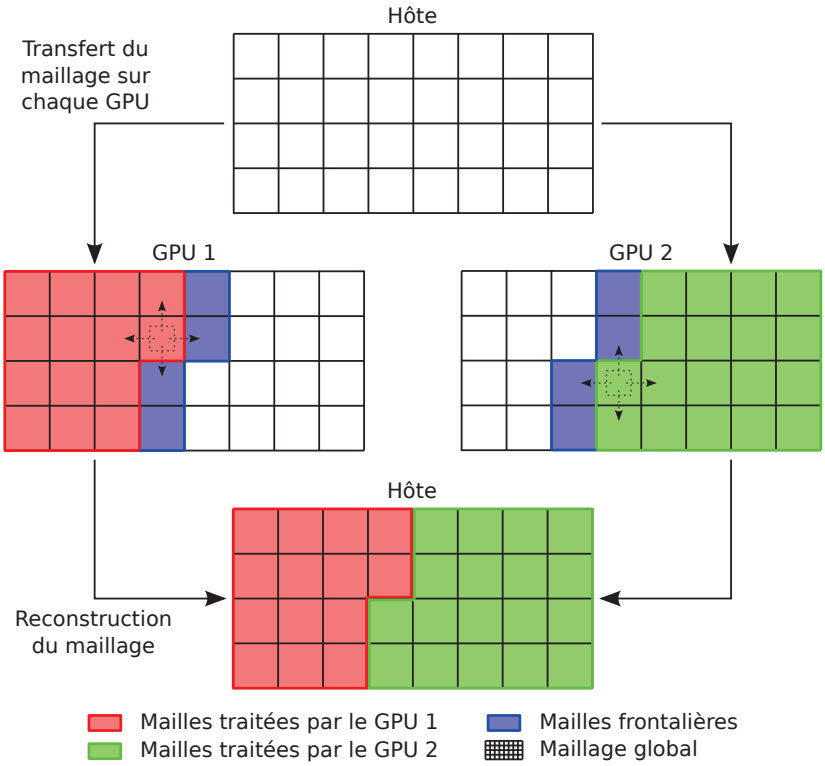


Figure 2.10 – **Distribution d'un maillage entre deux GPUs** – Ce schéma illustre la manière dont le traitement d'un maillage peut être partagé entre deux GPUs. Il met notamment en évidence le fait qu'une maille frontalière puisse consulter les données de mailles appartenant à un sous-maillage différent, et donc la nécessité pour chaque GPU de posséder une copie du sous-maillage sur lequel il opère et également une copie des frontières adjacentes.

Afin de spécifier à chaque GPU la zone du maillage qui lui est attribuée, il est nécessaire de modifier légèrement le *kernel* par rapport à sa version mono-GPU. Cette modification intervient lors du calcul de l'indice global permettant au *kernel* d'identifier la maille sur laquelle il est exécuté. Elle consiste à introduire un décalage passé en paramètre d'entrée du *kernel* (que nous nommons selon le terme anglais *offset*). Ainsi, le premier GPU exécutera le *kernel* sans décalage, c'est-à-dire qu'il opérera sur les premières mailles du maillage, tandis que les suivants l'exécuteront avec un décalage correspondant à l'indice de la première maille de leur sous-maillages respectifs. Si nous considérons le *kernel* mono-GPU donné en exemple sur la figure 1.9 de la section 1.2.2.3, celui-ci s'écrit dorénavant de la manière suivante :

```
1  // Définition du kernel CUDA pour une architecture multi-GPUs
2  __global__
3  void vectorAdd(const float *A,
4                 const float *B,
5                 float *C,
6                 int N,
7                 int offset // Décalage indiquant l'indice de départ des
                      calculs
8                 )
9  {
10   int id = threadIdx.x + blockIdx.x * blockDim.x + offset; // Indice global
11   if (id < N) C[id] = A[id] + B[id];
12 }
```

À cette modification du *kernel* s'ajoute une adaptation de la démarche de simulation. En effet, la démarche permettant de simuler des modèles population-centrés avec une architecture multi-GPUs diffère légèrement de celle proposée précédemment pour son homologue mono-GPU :

1. Discrétiser le domaine étudié sous la forme d'un maillage adapté à la méthode de résolution utilisée (différences finies, volumes finis...).
2. Découper (de manière statique) le maillage global en autant de sous-maillages que la machine hôte compte de GPUs.
3. Transférer chaque sous-maillage initialisé (tableaux « *to* »), ainsi que les frontières adjacentes, depuis la mémoire principale de la machine hôte vers la mémoire globale du GPU l'utilisant.
4. Pour chaque intervalle de temps Δt entre l'instant initial et l'instant final :
   - **(a)** Inverser les tableaux « *from* » avec les tableaux « *to* ».
   - **(b)** Recopier le contenu des mailles frontalières entre chaque GPU.
   - **(c)** Demander à chacun des GPUs d'exécuter le *kernel* sur son propre sous-maillage.
5. Transférer les différents sous-maillages contenant les résultats du traitement, depuis chaque GPU vers la mémoire principale de la machine hôte et reconstruire le maillage global.

Hormis la mise en place des sous-maillages dont le fonctionnement est détaillé précédemment, le changement majeur par rapport à la démarche mono-GPU réside dans l'ajout de l'étape 4b qui consiste à recopier le contenu des mailles frontalières entre les différents GPUs.

En effet, entre les cycles de simulation, chaque GPU doit être informé des modifications que le maillage global a subies. Pour cela, il est nécessaire que les GPUs échangent entre eux les données contenues dans les mailles susceptibles d'être consultées par les autres (généralement les mailles frontalières). Le transfert de données entre les GPUs est une opération coûteuse en terme de performances. En effet, en raison du fait que les GPUs ne partagent pas de mémoires communes, les données doivent transiter par la mémoire principale de la machine hôte. Ainsi, un transfert entre deux GPUs implique un premier transfert depuis le GPU « source » vers la machine hôte puis un second depuis la machine hôte vers le GPU « destination ». Pour minimiser le nombre de transferts, NVIDIA a mis au point la technologie *GPUDirect* qui permet une communication pair à pair directe entre plusieurs GPUs connectés à un même bus PCI Express [NVIDIA, 2012a]. Ces derniers peuvent ainsi échanger des données sans effectuer de recopie dans la mémoire principale de la machine hôte. Cependant, cette technologie est uniquement disponible sur les GPUs récents.

Cette mise à jour du contenu des mailles frontalières implique une synchronisation supplémentaire. En effet, avant de demander l'exécution du traitement dont les GPUs sont responsables, il est primordial de s'assurer que les transferts de données relatifs aux frontières des sous-maillages soient tous achevés. Cela permet de garantir la validité des données contenues dans les mailles lorsque l'exécution du *kernel* débutera. La figure 2.11 illustre cette démarche de simulation sur une architecture multi-GPUs.

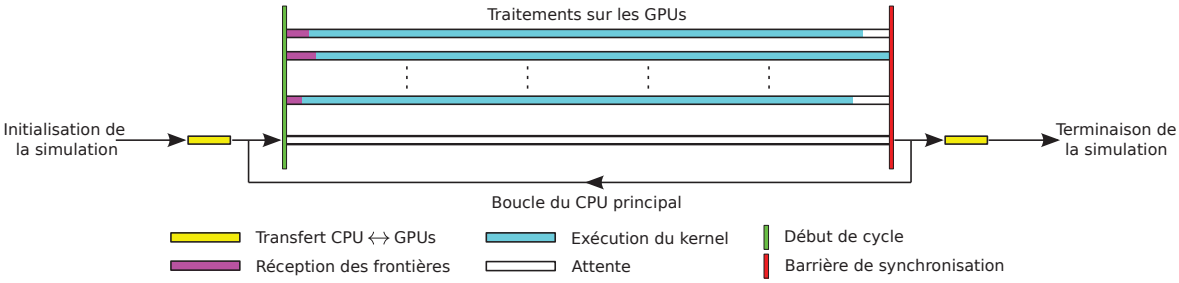


Figure 2.11 – **Déroulement d'une simulation multi-GPUs** – Ce schéma illustre le déroulement d'une simulation selon la démarche multi-GPUs que nous décrivons. Basé sur la figure 2.9, il met en évidence les phases de recopies de données aux frontières qui surviennent au début de chaque cycle.

Comme indiqué dans la section 1.2.2.3, les opérations de transferts de données ne sont pas anodines. En effet, qu'il s'agisse de transferts entre la machine hôte et un GPU ou bien entre deux GPUs distincts, ces opérations ont un impact négatif sur les performances. Néanmoins, nous remarquons que la recopie du contenu des mailles frontalières entre les GPUs participant à la simulation n'est pas aussi pénalisante qu'il n'y paraît. Tel serait le cas si le volume des zones frontalières devenait comparable à celui des sous-maillages recalculés. Mais un problème impliquant un si faible nombre de mailles serait peu adapté au calcul sur GPU et à plus forte raison sur multi-GPUs. Notons également que lors de la réception des données, c'est-à-dire lors de leur transfert depuis les GPUs vers la machine hôte (étape 5), il est primordial que chaque GPU transfère uniquement le contenu de son propre sous-maillage. Il ne doit en aucun cas transférer l'intégralité du maillage, ni même la ou les zones frontalières adjacentes, car cela entraînerait la corruption des données calculées par les autres GPUs.

Durant les cycles de simulation, le ou les CPUs de la machine hôte ne sont que très peu sollicités. En effet, hormis lors des étapes de recopie des frontières et de lancement de *kernels* auxquelles le CPU principal participe, l'ensemble des CPUs de la machine hôte sont inactifs. Pour améliorer les performances des simulations, il nous semble pertinent de permettre aux CPUs disponibles de collaborer avec les GPUs pour alléger leur charge de travail.

2.2.2.3 Collaboration avec les CPUs

Au même titre que les GPUs, l'ensemble des CPUs de la machine hôte peut réaliser les calculs prenant place au sein des mailles. En effet, cet ensemble dispose de nombreuses unités de traitement parfaitement capables de réduire la charge de travail des GPUs et d'améliorer significativement les performances des simulations. Pour permettre cette collaboration, nous proposons de créer un sous-maillage supplémentaire lors de la division du maillage global et d'en attribuer la gestion aux CPUs. Ainsi, d'un point de vue global, l'ensemble des CPUs se comporte comme un GPU [3], c'est-à-dire qu'il est responsable de l'évolution d'un sous-maillage au cours du temps.

À l'instar du découpage statique du maillage utilisé pour répartir les traitements entre les différents GPUs (cf. section 2.2.2.2), nous scindons une nouvelle fois le sous-maillage attribué aux CPUs en autant de parties que la machine hôte compte de CPUs. Ainsi, chaque CPU traite son propre sous-ensemble de mailles parallèlement à ses homologues. Nous rappelons que cet équilibrage statique est rendu possible uniquement par le fait que les traitements à réaliser au sein de chacune des mailles sont identiques.

Comme nous l'indiquions précédemment, le fonctionnement des CPUs est très différent de celui des GPUs. En effet, contrairement à un GPU qui de par son architecture SIMT est capable de traiter parallèlement l'ensemble des mailles qui lui sont confiées, un CPU ne peut les traiter qu'une à une, de manière séquentielle. De ce fait, chaque CPU nécessite la mise en place d'une boucle appliquant le traitement successivement sur chacune des mailles dont il a la responsabilité suite au découpage statique décrit précédemment. Cette succession de traitements est réalisée parallèlement à celles effectuées par les autres CPUs et également parallèlement à l'exécution des calculs sur les sous-maillages confiés aux GPUs. En outre, nous notons qu'il est nécessaire de décrire le traitement appliqué à chaque itération de cette boucle sous une forme adaptée à la programmation sur CPU, c'est-à-dire une fonction classique et non pas un *kernel*.

Étant donné que l'ensemble des CPUs joue le même rôle qu'un GPU, ils doivent, au même titre que ce dernier, participer à l'étape de recopie des frontières prenant place à chaque début de cycle de simulation. Le sous-maillage confié aux CPUs est stocké dans la mémoire principale de la machine hôte, contrairement à ceux manipulés par les GPUs. Par conséquent, dans cette situation spécifique, les échanges de données ont lieu entre les GPUs traitant les sous-maillages adjacents et la machine hôte. Par ailleurs, le fait que le contenu des mailles manipulées par les CPUs soit stocké dans la mémoire de la machine hôte permet d'éviter une recopie lors de l'étape de réception des données qui, pour rappel, consiste à rapatrier les

3. Notons que la participation des CPUs est l'un des aspects mis en avant par la technologie OpenCL.

données depuis les mémoires des GPUs vers celle de la machine hôte. La figure 2.12 propose un schéma illustrant le déroulement d'une simulation population-centrée utilisant à la fois les CPUs et les GPUs. Elle fait notamment apparaître la participation du CPU principal à l'étape de transfert des frontières.

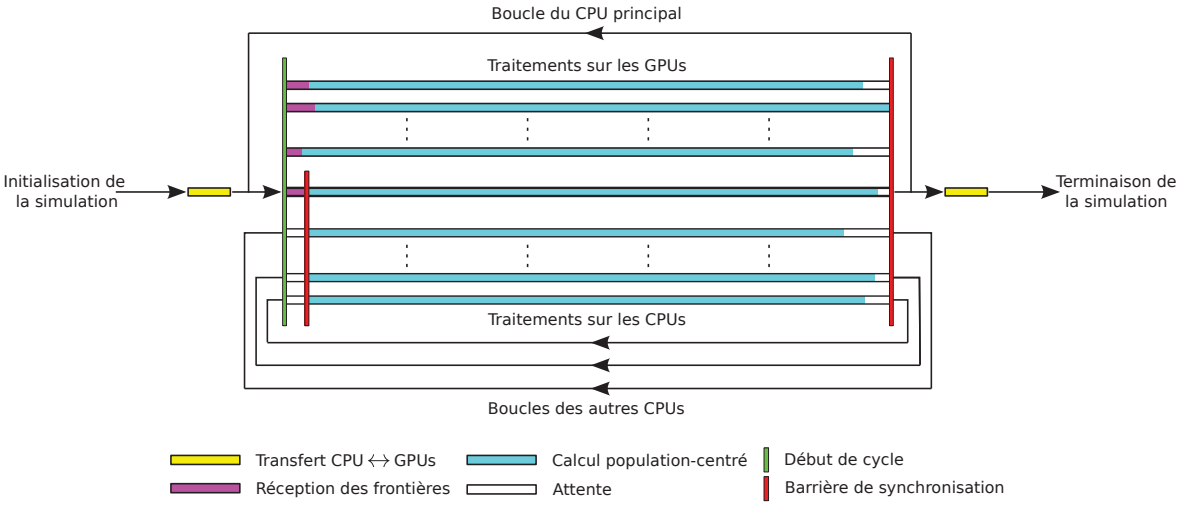


Figure 2.12 – **Déroulement d'une simulation population-centrée multi-CPUs et multi-GPUs** – Ce schéma illustre le déroulement d'une simulation population-centrée sur une architecture multi-CPUs et multi-GPUs. Il met en évidence le fait que les CPUs de la machine hôte peuvent participer aux traitements confiés initialement aux GPUs. Notons que seul le CPU principal participe au transfert de frontières.

2.2.3 Bilan de nos modèles population-centrés

Dans cette section, nous avons proposé des méthodes permettant d'utiliser toutes les unités de calcul présentes dans la machine hôte, qu'elles proviennent des GPUs ou des CPUs, et ainsi de bénéficier au mieux des performances offertes par cette dernière lors de la simulation de modèles population-centrés basés sur un maillage de l'espace. Nous évaluons les performances obtenues par notre approche dans le prochain chapitre de ce mémoire où nous l'appliquons à la simulation des phénomènes liés à la coagulation du sang.

Les approches population-centrées permettent de simuler de nombreux phénomènes régissant le comportement des systèmes biologiques. Malheureusement, elles ne sont pas capables de modéliser les contraintes géométriques et spatiales susceptibles d'intervenir localement au sein de tels systèmes. Elles se révèlent donc inadaptées pour la simulation de phénomènes où les interactions locales ont une grande importance. Pour remédier à ce problème, il est nécessaire de recourir à des modèles individu-centrés. Généralement plus précis que leurs homologues population-centrés, mais cependant moins performants, ils sont particulièrement adaptés pour modéliser des phénomènes où les contraintes locales jouent un rôle important. Ainsi, bien que leur fonctionnement diffère, nous proposons de coupler ces deux approches afin de bénéficier des avantages que chacune d'elles nous offre, tout

en s'abstrayant de leurs inconvénients respectifs. Ce couplage, qui fait l'objet de la section suivante, nous permet ainsi de mettre en œuvre des modélisations multi-échelles et multi-phénomènes essentielles à la simulation des systèmes biologiques.

2.3 Couplage de modèles population et individu-centrés

La modélisation de nombreux systèmes biologiques nécessite la mise en œuvre de méthodes multi-échelles ou multi-phénomènes exploitant à la fois les approches population et individu-centrées pour être pertinente. En effet, si nous cherchons par exemple à étudier le comportement de cellules immergées dans un fluide, alors nous privilégierons des méthodes population-centrées pour modéliser globalement l'écoulement, alors qu'en ce qui concerne les cellules, nous privilégierons une méthode individu-centrée capable de tenir compte des interactions locales survenant entre elles.

Les modèles individu et population-centrés que nous avons précédemment introduits dans ce mémoire se prêtent bien à un tel couplage. En effet, il est tout à fait envisageable de réduire la participation des CPUs aux calculs population-centrés assurés par les GPUs, leur permettant ainsi d'exécuter des modèles individu-centrés basés sur notre précédent moteur de simulation. Un tel couplage nous permet de maximiser les performances de nos simulations et de tirer pleinement partie de l'ensemble des unités de calcul présentes à l'intérieur des architectures actuelles. Dans cette section, nous détaillons les mécanismes liés à ce couplage.

2.3.1 Déroulement d'une simulation

L'évolution de nos simulations mêlant des modèles population-centrés simulés sur GPUs et CPUs et des modèles individu-centrés simulés sur CPUs est obtenue en couplant les démarches présentées dans les sections 2.2.2.2 et 2.1. Elle se résume de la manière suivante :

1. Discrétiser le domaine étudié sous la forme d'un maillage pour simuler les modèles population-centrés.
2. Découper (de manière statique) le maillage global en autant de sous-maillages que la machine hôte compte de GPUs, plus un supplémentaire confié aux CPUs.
3. Transférer chaque sous-maillage initialisé (tableaux « *to* »), ainsi que les frontières adjacentes, depuis la mémoire principale de la machine hôte vers la mémoire globale du GPU l'utilisant.
4. Pour chaque intervalle de temps Δt entre l'instant initial et l'instant final :
   - **(a)** Inverser les tableaux « *from* » avec les tableaux « *to* ».
   - **(b)** Recopier le contenu des mailles frontalières entre chaque GPU ainsi que la machine hôte.
   - **(c)** Traiter les différents sous-maillages par l'intermédiaire des CPUs et des GPUs.

(d) Exécuter les entités du modèle individu-centré sur les différents CPUs une fois leur sous-maillage traité (étape parallèle à l'exécution du *kernel*).

5. Transférer les différents sous-maillages contenant les résultats du traitement, depuis chaque GPU vers la mémoire principale de la machine hôte et reconstruire le maillage global.

La figure 2.13 illustre le déroulement d'une simulation utilisant un tel couplage sur une architecture multi-GPUs et multi-CPUs. Nous remarquons que lorsqu'un CPU a terminé de traiter les mailles qui lui sont attribuées, il peut immédiatement débuter la simulation des entités du modèle individu-centré : aucune synchronisation n'est nécessaire. Cela provient du fait que les résultats des calculs population-centrés sont inscrits dans l'état « *to* » du maillage alors que le calcul des entités est basé sur son état « *from* ». Il n'est donc pas nécessaire de s'assurer que l'intégralité du maillage soit traitée pour débuter la simulation des entités. Les mécanismes d'interaction entre les démarches population et individu-centrées sont détaillés dans la section suivante.

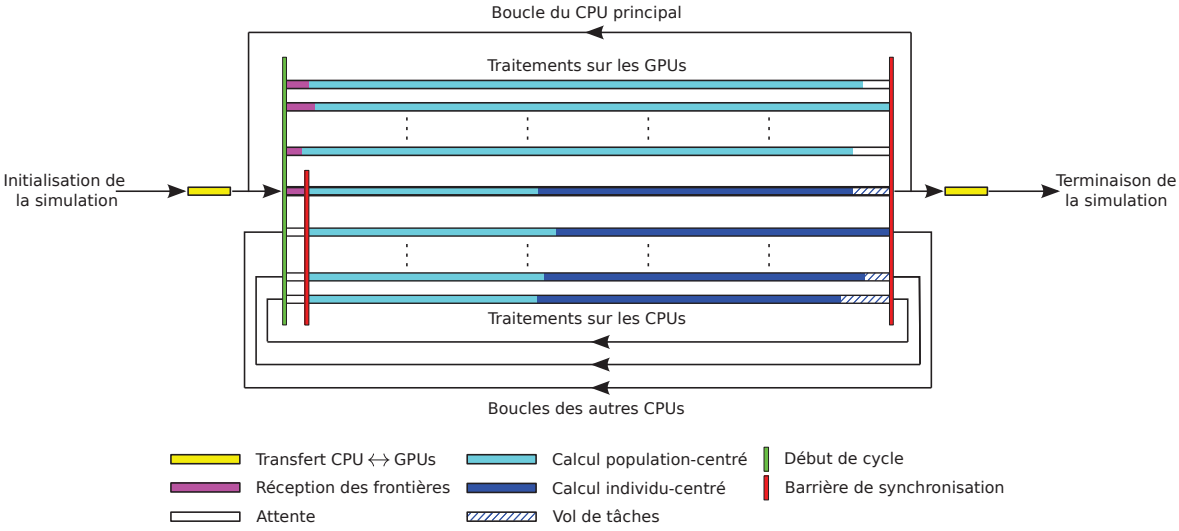


Figure 2.13 – **Déroulement d'une simulation individu et population-centrée** – Ce schéma illustre le déroulement d'une simulation couplant une démarche individu-centrée et une démarche population-centrée. Il résume les différents événements prenant place au sein d'un cycle de simulation et notamment la capacité des CPUs à participer aux deux démarches. Il met également en évidence l'absence de synchronisation entre les calculs population-centré et individu-centré.

2.3.2 Interaction entre les modèles

Le couplage entre notre modèle population-centré synchrone exécuté majoritairement sur GPUs et notre modèle individu-centré asynchrone exécuté sur CPUs est mis en œuvre par l'intermédiaire de deux mécanismes différents. Nous proposons deux méthodes de communication distinctes, une pour chaque sens d'interaction entre les deux modèles. La plus aisée à mettre en œuvre est celle modélisant l'influence du modèle population-centré sur le modèle individu-centré.

2.3.2.1 Influence du modèle population-centré sur le modèle individu-centré

Nous indiquions dans la section 2.2.1 que l'évolution de notre modèle population-centré utilise des tableaux de données à deux états, « *from* » et « *to* », décrivant respectivement l'état courant et futur du maillage. L'évolution d'un instant au suivant de notre modèle population-centré est contrôlée par des calculs s'effectuant à partir de l'état « *from* » et déterminant l'état « *to* ». De ce fait, lorsque notre modèle individu-centré a besoin de consulter des données produites par le modèle population-centré, il lui suffit de consulter l'état « *from* » des tableaux de données. En aucun cas, il ne doit consulter leur état « *to* » qui est mis à jour parallèlement à l'évolution du modèle individu-centré. Cela entraînerait la consultation de données corrompues.

Néanmoins, afin qu'une entité de notre modèle individu-centré puisse consulter des données contenues dans un tableau manipulé par le modèle population-centré (et donc les GPUs), il est primordial de le recopier depuis les mémoires globales des GPUs vers celle de la machine hôte à chaque cycle. Sans cela, notre modèle individu-centré, exécuté sur CPU, ne peut pas accéder à ces données. Notons que les portions de tableau traitées par les CPUs n'ont pas besoin d'être transférées, étant donné qu'elles sont déjà présentes dans la mémoire de la machine hôte.

2.3.2.2 Influence du modèle individu-centré sur le modèle population-centré

Les échanges de données depuis notre modèle individu-centré vers notre modèle population-centré sont plus compliqués à mettre en œuvre que dans le sens inverse, où il suffit que les entités consultent les données dont elles ont besoin dans les tableaux de l'état « *from* ». L'influence des entités de notre modèle individu-centré sur les tableaux de notre modèle population-centré ne peut pas être modélisée si aisément. En effet, au cours d'un cycle de simulation, notre modèle population-centré consulte les données de l'état « *from* » pour calculer celles de l'état « *to* ». Pour éviter toute corruption des données et assurer leur cohérence, les entités de notre modèle individu-centré ne peuvent donc pas modifier directement les données de ces deux états.

La solution que nous proposons pour remédier à ce problème consiste à mettre en place un nouveau tableau (pour chaque donnée) contenant les variations que chaque entité souhaite appliquer sur le maillage. Comme l'ensemble des tableaux de données que nous utilisons, ce tableau est dupliqué en deux exemplaires « *from* » et « *to* ». Lorsque les entités du modèle individu-centré sont exécutées, elles inscrivent, par l'intermédiaire d'ajouts ou de retraits, les modifications qu'elles souhaitent apporter au maillage du modèle population-centré dans le tableau de variations « *to* ». Simultanément, les *kernels* pilotant l'évolution du maillage sont exécutés avec un paramètre supplémentaire : le tableau de variations « *from* » (qui correspond au tableau « *to* » rempli par les entités au cycle précédent). De cette manière, le modèle population-centré est capable de prendre en compte l'influence des entités de notre modèle individu-centré au cours de ses calculs.

Les modifications qu'effectuent les entités dans le tableau de variations « *to* » sont réalisées par l'intermédiaire d'opérations atomiques, c'est-à-dire ne pouvant être interrompues par un autre CPU avant la fin de leur exécution. Ce type d'opération répond idéalement à nos besoins car les valeurs que nous manipulons dans les mailles ne sont que de simples quantités scalaires (une pression, une concentration...) provenant des équations pilotant notre modèle population-centré. Il ne s'agit aucunement de structures de données complexes au sein desquelles la modification d'une valeur se répercuterait potentiellement sur une autre. Cela surviendrait par exemple si plusieurs valeurs étaient liées entre elles, telles que les coordonnées d'un point évoluant sur un cercle où l'abscisse est directement liée à l'ordonnée. De plus, les opérations effectuées sur ces quantités sont par essence très simples : il s'agit uniquement de sommes d'influences. De ce fait, elles se prêtent particulièrement bien à l'utilisation d'opérations atomiques, ce qui évite ainsi de mettre en œuvre des mécanismes de synchronisation plus lourds tels que des verrous dans chacune des mailles.

Entre tous les cycles de simulation, il est primordial de réinitialiser le contenu des tableaux de variations en inscrivant une valeur nulle dans chacune de leurs cases. Sans cela, les variations proposées par le modèle individu-centré sur le modèle population-centré s'accumuleraient de cycle en cycle, ce qui entraînerait des biais dans les résultats. La méthode la plus intuitive pour mettre en œuvre cette réinitialisation consisterait à effectuer, à la fin de chaque cycle, une boucle parcourant l'ensemble des cases des tableaux de variations pour y inscrire une valeur nulle. L'inconvénient de cette solution réside dans la perte de performances qu'entraîne le parcours séquentiel de l'ensemble des cases des tableaux. Pour s'abstraire de cet inconvénient, nous proposons une méthode plus astucieuse permettant d'éviter l'utilisation d'une telle boucle. Elle consiste à faire en sorte que chacune des entités de notre modèle individu-centré retienne les informations liées aux modifications qu'elle applique sur le maillage, c'est-à-dire l'indice de la maille ainsi que les valeurs des variations, pour être en mesure de les soustraire plus tard. En raison de l'alternance des états « *from* » et « *to* » d'un cycle de simulation à l'autre, cette soustraction doit nécessairement survenir deux cycles après l'inscription de la valeur dans le tableau de variations. Elle a pour effet d'annuler les modifications survenues lors de l'avant-dernier cycle, et donc imite une réinitialisation du contenu des tableaux de variations. Comme plusieurs entités sont susceptibles de modifier une même maille, plusieurs ajouts et retraits peuvent s'accumuler au sein de chaque case des tableaux de variations. Nous notons que les retraits et ajouts survenant au sein d'une même maille peuvent être effectués en une seule opération. Cette méthode affecte très peu les performances de nos simulations car elle se résume uniquement à quelques soustractions effectuées ponctuellement par chacune des entités. La figure 2.14 propose un exemple illustrant le fonctionnement de ces tableaux de variations.

Cette proposition, de la même manière que celle concernant l'influence du modèle population-centré sur le modèle individu-centré, entraîne un transfert de données supplémentaire depuis la mémoire de la machine hôte vers les mémoires globales des GPUs. En effet, les tableaux de variations remplis par les entités se trouvent dans la mémoire de la machine hôte. Il est donc nécessaire, à chaque cycle de simulation, de transférer leur contenu vers la mémoire des GPUs pour que ces derniers puissent les consulter et les intégrer à leurs calculs.

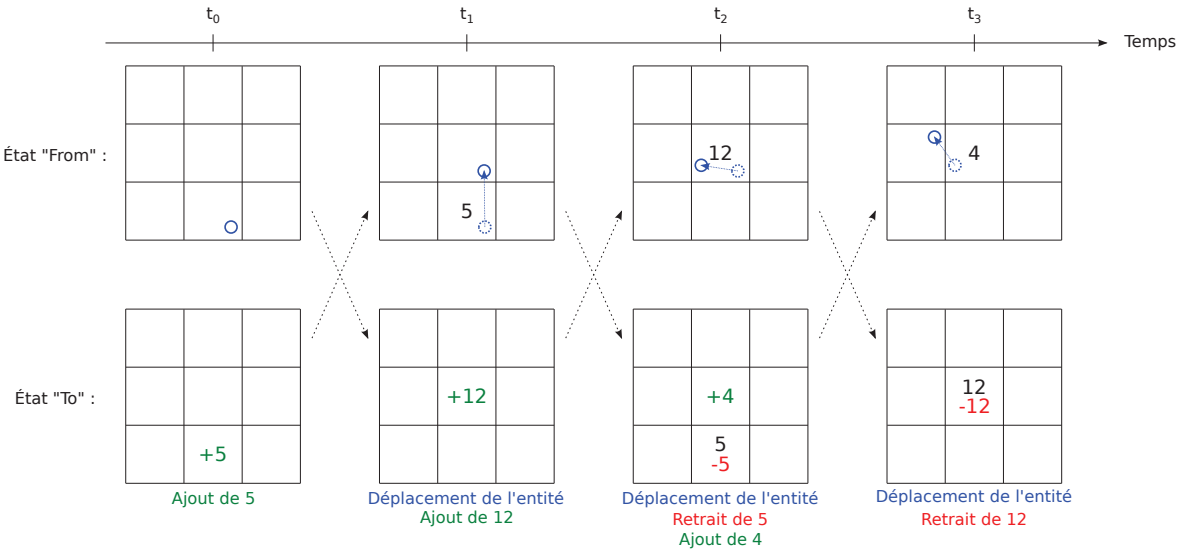


Figure 2.14 – **Fonctionnement des tableaux de variations** – Ce schéma présente un exemple d'utilisation des tableaux de variations permettant aux entités de notre modèle individu-centré d'influer sur les calculs population-centrés. Il représente un maillage en deux dimensions sur lequel se déplace une entité (cercle bleu) capable de modifier les valeurs contenues dans chacune des mailles. Les modifications à l'initiative de l'entité sont inscrites en vert tandis que les soustractions indiquées en rouge correspondent à la réinitialisation des variations.

2.3.2.3 Bilan des interactions entre les modèles

Nos deux propositions permettent d'obtenir un système global où le couplage entre modèle individu-centré et modèle population-centré est complet. En effet, des entités de notre modèle individu-centré sont capables d'agir sur le contenu du maillage de notre modèle population-centré et ce dernier peut à son tour influer sur le comportement des entités. Le déroulement d'une simulation basée sur un tel couplage est représenté sur la figure 2.15. Pour illustrer cela, nous pouvons prendre l'exemple de cellules immergées dans un fluide. Le fluide agit sur les cellules en leur communiquant son mouvement. De leur coté, les cellules sont capables d'interagir entre elles pour s'agréger. L'amas formé est responsable d'une modification de l'écoulement du fluide, ce qui influera donc par la suite sur le comportement des cellules et ainsi de suite.

2.4 Bilan de notre proposition

Dans ce chapitre, nous avons présenté une solution mixte à deux niveaux, c'est-à-dire celui de la modélisation et celui de la parallélisation. En effet, nous avons proposé un couplage entre un modèle individu-centré, parallélisé par l'intermédiaire d'une architecture multicœur et multiprocesseur, et un modèle population-centré, principalement parallélisé par une architecture graphique pouvant être épaulée par des CPUs. Ce couplage nous permet

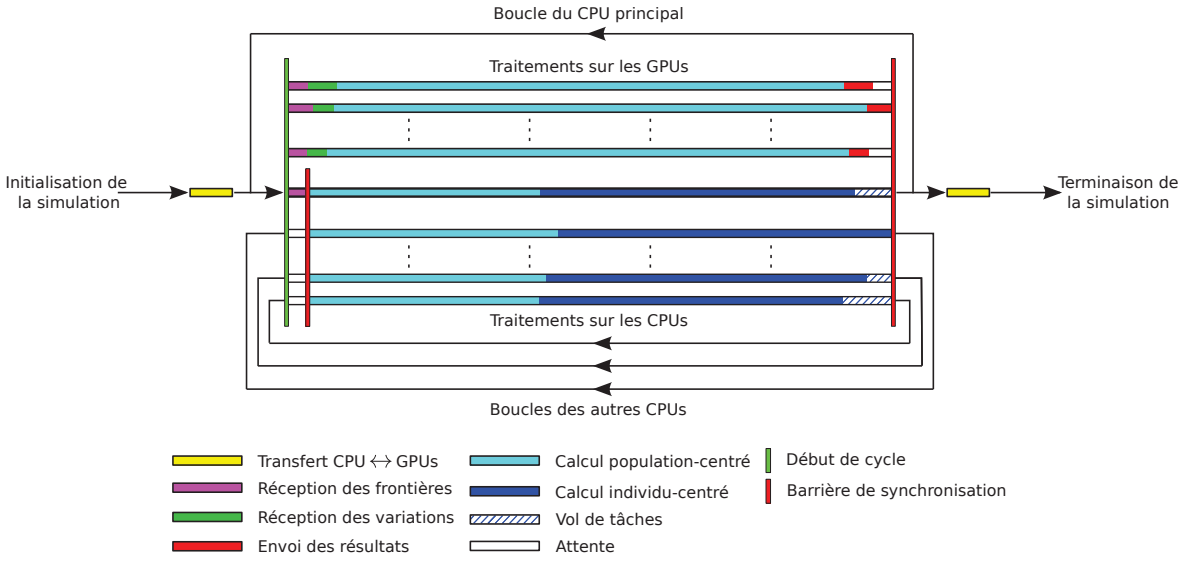


Figure 2.15 – **Déroulement complet d'une simulation individu et population-centrée** – Ce schéma enrichit celui de la figure 2.13 en mettant en évidence les interactions entre les deux modèles, c'est-à-dire le transfert des tableaux de variations et de résultats. Notons que le CPU principal a la responsabilité de déclencher, de manière asynchrone, les transferts de données et les calculs sur les GPUs.

de modéliser la grande variété des phénomènes pilotant les systèmes biologiques tout en bénéficiant des performances offertes par les architectures parallèles.

La mise en œuvre d'une parallélisation mixte permet d'exploiter au mieux les affinités des différentes architectures parallèles envers les modèles individu-centrés et les modèles population-centrés. Ainsi, les CPUs, de par leur polyvalence, participent à la simulation de tous les modèles, aussi bien individu que population-centrés, alors que les GPUs simulent uniquement les phénomènes modélisés à l'échelle des populations. Le développement de nos algorithmes a été guidé par les spécificités propres à chacune de ces deux architectures parallèles. En effet, afin de maximiser les performances de notre modèle individu-centré, nous avons porté une attention toute particulière à la manière d'utiliser les mémoires caches des CPUs, notamment lors des étapes de vol de tâches permettant d'équilibrer dynamiquement la charge de chaque CPU. En ce qui concerne la parallélisation de modèles population-centrés, nous avons insisté sur le besoin de transférer les frontières entre les sous-maillages lors d'une démarche multi-GPUs, ainsi que le coût que de tels échanges de données peuvent engendrer. Nous avons conclu ce chapitre en présentant les mécanismes d'interaction entre modèles que nous avons mis en œuvre pour réaliser notre couplage multi-modèles.

Le chapitre suivant de ce mémoire propose deux applications de notre méthode de simulation à des systèmes biologiques appartenant au domaine de la coagulation du sang. La première d'entre elles vise à étudier la cinétique biochimique à l'échelle microscopique, uniquement par l'intermédiaire de notre modèle individu-centré. La seconde, nettement plus complexe, fait interagir de nombreux modèles entre eux afin de simuler le phénomène de coagulation du sang au sein d'un vaisseau sanguin virtuel en trois dimensions.

Chapitre 3

Exemples d'application à la coagulation du sang

Dans le cadre de ce mémoire, nous avons choisi d'appliquer les méthodes de simulation des systèmes biologiques que nous proposons dans le chapitre précédent au domaine de la coagulation du sang, ou hémostase. Nous nous intéressons tout particulièrement à deux applications bien distinctes : la simulation de la cinétique biochimique à l'échelle microscopique puis celle d'un vaisseau sanguin virtuel. Les organisations des deux sections détaillant ces applications sont très similaires. Nous présentons dans un premier temps les phénomènes biologiques en jeu, puis nous détaillons les modèles que nous avons développés pour enfin les simuler. Chacune de ces deux sections se conclut par une analyse des résultats obtenus lors de nos simulations, ainsi que par une discussion de potentielles perspectives liées à l'application concernée. Notons que le lecteur curieux trouvera une explication détaillée de la coagulation du sang dans la section 3.2.1.

3.1 Cinétique biochimique à l'échelle microscopique

Dans cette section nous proposons un exemple d'application à la coagulation du sang exploitant uniquement le modèle individu-centré développé dans la section 2.1 de ce mémoire. Ce premier exemple consiste à simuler les phénomènes liés à la cinétique biochimique à l'échelle microscopique. Ainsi, après une brève explication des phénomènes entrant en jeu, nous proposons un état de l'art des simulateurs existants et expliquons notre motivation pour cette approche. La description de notre modèle biologique se poursuit ensuite par une phase de validation des résultats obtenus. Enfin, nous discutons de l'intérêt de notre approche et proposons quelques perspectives pour nos futures recherches.

3.1.1 La cinétique biochimique

Les systèmes biologiques, notamment la coagulation du sang, font nécessairement apparaître des événements de nature biochimique tels que les réactions chimiques. Ainsi, la simulation numérique de ces systèmes implique la modélisation de leur cinétique biochimique, c'est-à-dire de la manière dont évoluent les concentrations des différentes espèces chimiques (protéines, ions, enzymes...) au cours du temps. Dans le cadre de la coagulation du sang, de telles réactions chimiques interviennent lors de la formation du caillot sanguin. La cinétique biochimique peut être simulée par l'intermédiaire de différentes méthodes, mais classiquement, cela est accompli en implémentant des lois empiriques, telles que la loi d'action de masse ou celle de Henri-Michaelis-Menten [Purich et Allison, 2000], grâce à des équations différentielles résolues par des méthodes numériques [Alves *et al.*, 2006]. La présence d'éléments biologiques, comme les membranes, peut introduire une hétérogénéité dans le milieu où les événements biochimiques surviennent. Des EDPs sont ainsi utilisées pour tenir compte de cette hétérogénéité spatiale. L'approche population-centrée que constitue l'utilisation des équations différentielles s'appuie sur l'hypothèse que le milieu étudié est suffisamment grand et peuplé pour fournir des résultats significatifs. Lorsqu'il s'agit de modéliser de faibles volumes, comme l'intérieur d'une cellule, cette hypothèse n'est pas valable. Par conséquent, cette approche basée sur des EDPs est uniquement applicable aux simulations à l'échelle macroscopique. Ainsi, Gillespie introduit des algorithmes de simulation stochastique permettant d'imiter précisément le comportement de la solution de l'équation maîtresse stochastique qui décrit la cinétique biochimique à l'échelle mésoscopique [Gillespie, 1976, 1977]. Cette méthode tient compte des aspects discret et stochastique des réactions biochimiques et permet la simulation numérique de la cinétique biochimique quel que soit le volume du domaine étudié [van Kampen, 2007]. Notons que le cas des domaines hétérogènes a également été traité dans [Stundzia et Lumsden, 1996].

De nos jours, il est largement reconnu que, en plus des aspects discret et stochastique, les contraintes spatiales doivent être prises en compte pour simuler précisément la cinétique biochimique [Resat *et al.*, 2011]. Par exemple, la cascade de la coagulation se déroulant lors de l'hémostase fait intervenir des réactions enzymatiques prenant place à la fois en solution et à la surface de membranes. Qui plus est, au fur et à mesure que le caillot sanguin se construit, un réseau de fibrine se forme et le milieu devient insoluble : la cinétique enzymatique classique ne s'applique plus. La simulation de tels phénomènes complexes est typiquement obtenue par l'intermédiaire de systèmes individu-centrés qui proposent de décrire la cinétique biochimique à l'échelle microscopique [Tolle et Le Novere, 2006]. Comme expliqué dans notre état de l'art (cf. section 1.1.2), cette approche considère les molécules individuellement, souvent nommées particules ou entités, pendant qu'elles diffusent en trois dimensions, collisionnent et réagissent. L'inconvénient majeur d'une telle méthode provient du faible pas de temps nécessaire et du très grand nombre d'entités requises pour obtenir des résultats significatifs, ce qui rend les simulations extrêmement coûteuses en temps de calcul. Le tableau 3.1 synthétise les différentes méthodes de modélisation et de simulation de la cinétique biochimique. Plusieurs modèles individu-centrés à l'échelle microscopique existent : Smoldyn [Andrews *et al.*, 2010], MCell [Stiles et Bartol, 2001], ChemCell [Plimpton et Slepoy, 2003], HSIM [Amar *et al.*, 2004] ou encore GFRD [van Zon et ten Wolde, 2005]. Cependant, à notre connaissance, hormis Smoldyn qui propose depuis très récemment une implémentation parallélisée sur GPU [Dematté, 2012],

aucun de ces travaux ne se focalise sur une notable amélioration des performances.

	Modélisation temporelle	Modélisation spatio-temporelle	
Échelle	**Milieu homogène**	**Milieu hétérogène**	
macroscopique	EDOs	EDPs	Modèle continu et déterministe
mésoscopique	Équation maîtresse stochastique		Modèle discret et stochastique
microscopique	Systèmes individu-centrés		Modèle discret et/ou continu et stochastique

Tableau 3.1 – **Modélisation et simulation de la cinétique biochimique** – Ce tableau présente les différentes méthodes de modélisation et de simulation de la cinétique biochimique en fonction de l'échelle et du type de milieu considérés.

Nous adressons ce problème en appliquant notre modèle individu-centré capable d'exploiter au mieux les architectures multicœur et multiprocesseur, modèle présenté dans le chapitre 2, à la simulation de la cinétique biochimique à l'échelle microscopique. La section suivante décrit donc le modèle biologique que nous proposons.

3.1.2 Modélisation individu-centrée de la cinétique biochimique

Cette application s'appuie sur des travaux précédemment effectués au sein de notre laboratoire et présentés dans [Kerdélo, 2006]. Contrairement à ces derniers qui décrivent une démarche de simulation séquentielle, nous proposons d'utiliser notre modèle individu-centré parallélisé pour optimiser les performances des simulations de cinétique biochimique. Ainsi, les algorithmes que nous présentons dans cette section sont sensiblement identiques à ceux émanant de ces précédents travaux, et notre contribution réside principalement dans la parallélisation des simulations.

Notre modélisation permet de simuler la cinétique des systèmes biochimiques en tenant compte des aspects discret, stochastique et spatial du problème. Notre modèle, illustré par la figure 3.1, est capable de représenter deux types d'espèce moléculaire : les entités `Species3D` qui diffusent en solution (dans un `Volume`) et leurs homologues `Species2D` qui diffusent sur une membrane physiologique (sur une `Surface`). Chaque espèce moléculaire est représentée par sa forme géométrique : respectivement un ellipsoïde pour les espèces en trois dimensions en solution et un disque pour les espèces en deux dimensions fixées sur une membrane.

Chaque entité diffuse dans le volume réactionnel selon le mouvement Brownien. Ce déplacement est paramétré par un coefficient de diffusion déterminé à partir des rayons de

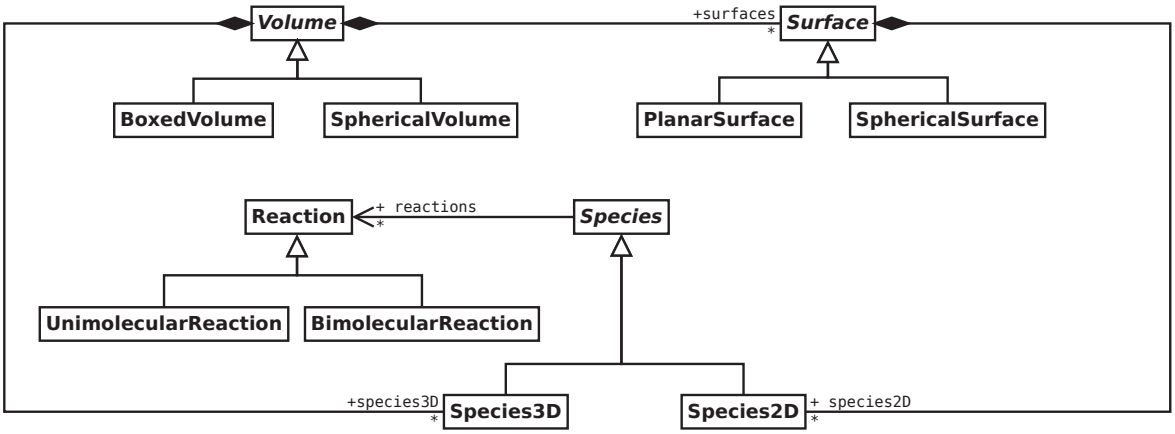


Figure 3.1 – **Diagramme de classe UML de notre modèle** – Ce diagramme simplifié met en évidence la classe `Species` représentant une entité autonome diffusant en trois dimensions dans un `Volume` ou en deux dimensions sur une `Surface`, et pouvant subir une ou plusieurs réactions définies par la classe `Reaction`.

l'entité ainsi que de la viscosité et de la température du volume réactionnel. Ces entités peuvent subir deux types de réaction biochimique : les réactions unimoléculaires et les réactions bimoléculaires. Elles sont responsables de la création ou de la destruction d'autres entités et, par conséquent, elles régissent les variations des concentrations chimiques dans le système. Ce cycle de vie définit la fonction d'activation de notre modèle individu-centré (cf. section 2.1.1) et est détaillé sur l'algorithme de la figure 3.2. Les sections suivantes décrivent chacune des étapes de cet algorithme.

3.1.2.1 Réactions unimoléculaires

Les réactions unimoléculaires sont des phénomènes capables de transformer une espèce biochimique (le réactif) en un ou plusieurs produits. Une réaction R convertissant une espèce moléculaire C en un couple de produits A et B est représentée par le schéma suivant :

$$R: \quad \mathrm{C} \quad \overset{k}{\rightarrow} \quad \mathrm{A} + \mathrm{B} \tag{3.1}$$

où k est la constante de réaction qui caractérise la vitesse du phénomène.

Une espèce peut participer à une ou plusieurs réactions unimoléculaires. La simulation d'un système de n réactions nécessite de calculer la probabilité de chacune d'elles, c'est-à-dire la probabilité $P(R_i)$ de l'événement « la réaction R_i survient lors d'un pas de temps Δt donné » et la probabilité $P(\overline{R})$ de l'événement « aucune réaction n'a lieu ». Ces probabilités,

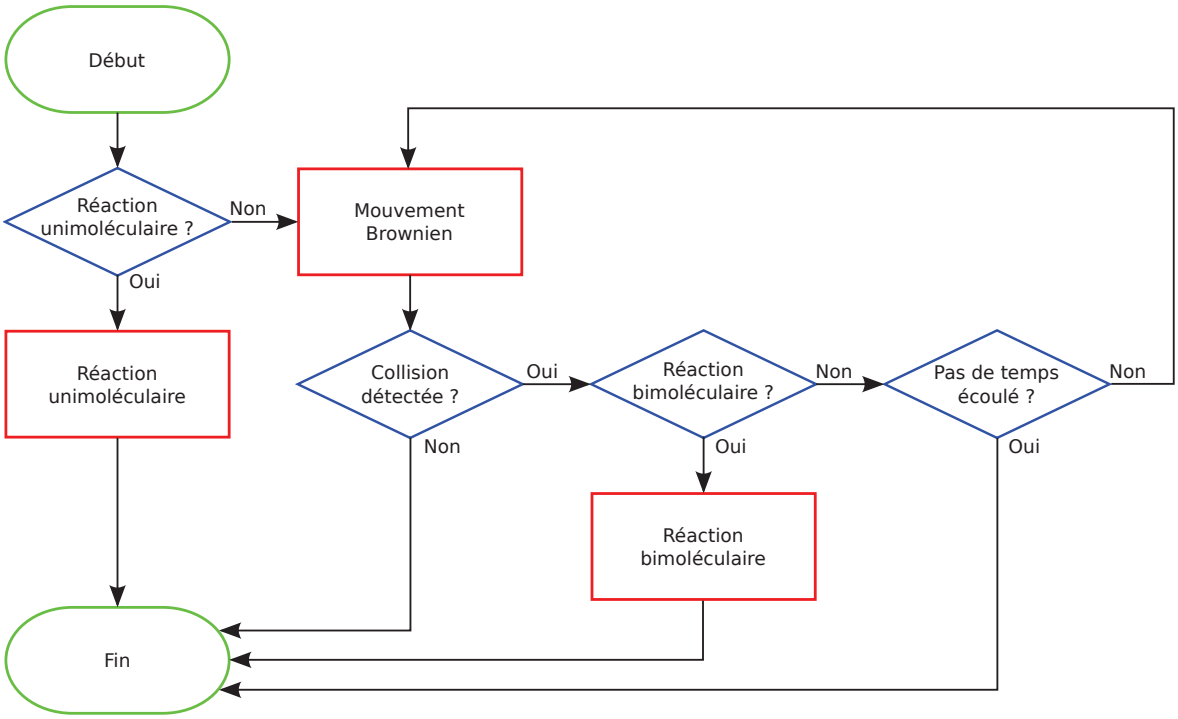


Figure 3.2 – **Cycle de vie d'une entité** – Ce schéma résume le comportement global d'une entité au cours d'un cycle de simulation. Tout d'abord, l'entité teste si elle doit subir une réaction unimoléculaire. Si ce n'est pas le cas, l'entité diffuse dans son environnement selon un mouvement Brownien. Ensuite, si une collision survient, l'entité teste si elle doit réagir selon un schéma bimoléculaire avec celle rencontrée. Ce processus est répété jusqu'à ce que le pas de temps alloué à l'entité soit écoulé.

détaillées notamment dans [Andrews et Bray, 2004], sont données par les équations suivantes :

$$\begin{cases} P(R_i) &= \dfrac{k_i}{\sum_{j=1}^{n} k_j} \times \left(1 - \exp(-\sum_{j=1}^{n} k_j \times \Delta t)\right) \\ P(\overline{R}) &= \exp(-\sum_{j=1}^{n} k_j \times \Delta t) \ . \end{cases} \tag{3.2}$$

Ce système détermine quelle réaction surviendra au cours du cycle de simulation de durée Δt courant grâce à un tirage aléatoire suivant une loi uniforme.

3.1.2.2 Mouvement Brownien et détection de collision

Le mouvement Brownien caractérise le mouvement erratique d'une entité au sein d'un fluide. Il résulte des collisions entre les particules de solvant et l'entité elle-même, qui semble ainsi se déplacer de manière aléatoire sous l'action de ces impacts [Berg, 1993]. La théorie associée au mouvement Brownien de molécules ellipsoïdales est étudiée dans

[Perrin, 1934, 1936]. À l'échelle microscopique, ce phénomène est habituellement modélisé par l'intermédiaire de marches aléatoires en trois dimensions, c'est-à-dire un processus de Markov défini par une succession de déplacements aléatoires élémentaires. Ainsi, les futures position et orientation d'une entité dépendent uniquement de son état actuel. Cette succession de déplacements élémentaires peut être approximée par des tirages aléatoires suivant une loi Gaussienne. En l'occurrence, au cours d'un pas de temps Δt, chaque translation et chaque rotation est calculée à partir de six (en trois dimensions) ou quatre (en deux dimensions) variables aléatoires suivant une loi Gaussienne de moyenne nulle et d'écart-type $\sqrt{2D\Delta t}$, où D est le coefficient de diffusion de l'entité concernée.

Au cours de cette étape de diffusion, une entité peut rencontrer une de ses voisines proches. Pour déterminer si une collision a lieu entre deux espèces moléculaires, nous commençons par tester si les deux sphères englobant les réactifs sont en contact. Si tel n'est pas le cas, les deux espèces moléculaires ne sont pas en collision et l'étape de détection de collision prend fin. L'échec de ce test préliminaire simple nous permet d'éviter un test de détection de collision plus précis et donc plus coûteux. Nous remarquons que nous aurions également pu utiliser un mécanisme de boîtes englobantes comme celui décrit dans [Kerdélo, 2006]. Cependant, notre approche à base de sphères est plus simple à mettre en œuvre (il suffit de comparer la distance entre les centres des entités avec la somme de leurs plus grands rayons respectifs). Néanmoins, si ce test préliminaire réussit, nous poursuivons en procédant à une détection de collision entre ellipsoïdes utilisant l'algorithme présenté dans [Wang *et al.*, 2001]. Ce dernier permet d'établir si un plan de séparation existe entre deux ellipsoïdes en fonction des racines de leur polynôme caractéristique. Ce polynôme s'écrit $f(\lambda) = det(\lambda A + B)$ où A et B sont les matrices caractérisant chacun des deux ellipsoïdes. Ces matrices se présentent sous la forme suivante :

$$A = \begin{pmatrix} \frac{1}{a^2} & 0 & 0 & 0 \\ 0 & \frac{1}{b^2} & 0 & 0 \\ 0 & 0 & \frac{1}{c^2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

où a, b et c représentent les trois rayons de l'ellipsoïde concerné. La résolution de l'équation $f(\lambda) = 0$ permet de déterminer les racines du polynôme. Selon l'article, deux ellipsoïdes sont séparés par un plan si les deux racines sont distinctes et positives. Ils se touchent s'il existe une racine double positive (deux racines identiques). Dans ces deux situations, nous considérons que les deux espèces moléculaires sont entrées en collision.

Habituellement, une telle étape de détection de collision est très coûteuse en temps de calcul. En effet, elle implique de devoir tester si une collision a lieu entre l'entité en cours d'exécution et l'ensemble des autres entités du système. Afin de restreindre cet ensemble à une poignée d'entités, nous mettons en place un maillage en trois dimensions au sein duquel les entités indiquent leur présence. Ainsi, lorsqu'une entité cherche à détecter ses voisines proches, elle n'a plus besoin de parcourir l'ensemble du volume, elle peut se contenter d'explorer son voisinage local composé de quelques mailles et donc de peu d'entités. Suite à cette étape de détection de collision, nous utilisons un algorithme de dichotomie pour gérer la collision et éviter le chevauchement des deux entités. Cet algorithme itératif permet de décomposer le déplacement effectué par l'entité au cours du pas de temps actuel puis de le parcourir jusqu'à

trouver une position et une orientation précédant immédiatement l'impact. De telles collisions peuvent aboutir à des réactions bimoléculaires.

3.1.2.3 Réactions bimoléculaires

Contrairement à leurs homologues unimoléculaires, les réactions bimoléculaires peuvent uniquement se produire lorsque des collisions entre molécules surviennent dans le système. De ce fait, deux réactifs sont nécessaires pour créer un produit. Une réaction R entre deux espèces A et B, produisant une espèce C, se représente comme suit :

$$R: \quad \mathrm{A} + \mathrm{B} \quad \overset{k}{\rightarrow} \quad \mathrm{C} \tag{3.4}$$

où k est à nouveau la constante de réaction caractérisant la vitesse du phénomène.

À l'instar de la simulation des réactions unimoléculaires, nous cherchons à déterminer la probabilité $P(R)$ qu'une réaction bimoléculaire R survienne lorsqu'une collision entre deux entités A et B a lieu dans le volume réactionnel. Les travaux présentés dans [Kerdélo, 2006] sur lesquels nous nous sommes appuyés jusqu'ici proposent une démarche permettant de calculer cette probabilité de réaction à partir de différents paramètres caractérisant les molécules, et notamment leurs coefficients de diffusion et leurs rayons. Cette démarche utilise la théorie de von Smoluchowski qui indique que la constante maximale d'une réaction est donnée par $k_{\max} = 4\pi(D_{\mathrm{A}} + D_{\mathrm{B}})(r_{\mathrm{A}} + r_{\mathrm{B}})\mathrm{N_A}$, où D_{A} et r_{A} sont respectivement le coefficient de diffusion et le rayon moyen de l'espèce A (de même pour B), et $\mathrm{N_A}$ est le nombre d'Avogadro [Purich et Allison, 2000]. Cette constante maximale de réaction correspond au fait que chaque collision entre une entité A et une entité B entraîne une réaction, ce qui se traduit par l'égalité $P(R) = 1$. L'auteur utilise ensuite cette constante maximale pour calculer le ratio $\frac{k}{k_{\max}}$, où k est la constante de réaction que nous souhaitons simuler. Ce ratio, compris entre 0 et 1, correspond à la probabilité $P(R)$ qu'une réaction survienne suite à une collision entre les espèces A et B. Ainsi, une réaction R a lieu seulement si la valeur obtenue suite à un tirage aléatoire suivant une loi uniforme est inférieure à $P(R)$.

Malheureusement, en implémentant cet algorithme, nous avons observé des incohérences entre les résultats provenant de nos simulations et ceux obtenus par la résolution de la loi de référence, c'est-à-dire la loi d'action de masse. En effet, bien que dans certaines situations cet algorithme fournisse des résultats corrects, ce n'est pas toujours le cas. Selon nous, cela provient du fait que si deux entités ne réagissent pas lors de leur première collision, alors la probabilité qu'elles le fassent lors des prochains cycles de simulation est très élevée. Cela est dû au fait que les entités sont toujours très proches l'une de l'autre après cette première rencontre. Nous pensons que les résultats incorrects obtenus avec cet algorithme proviennent de ce biais.

Par conséquent, nous proposons notre propre algorithme afin de pallier ce problème. Ce dernier consiste à déterminer la probabilité de réaction empiriquement par un mécanisme d'essais/erreurs. Pour cela, nous exécutons une succession de simulations, chacune d'elles avec une probabilité de réaction $P(R)$ différente fixée arbitrairement. Nous comparons ensuite chacun des résultats avec ceux de la loi d'action de masse. Ainsi, nous sommes capables de

sélectionner la probabilité $P(R)$ permettant d'approcher au mieux la constante de réaction k souhaitée. Cette méthode est plus contraignante que celle proposée dans [Kerdélo, 2006] car elle nécessite la répétition de plusieurs simulations afin de déterminer la probabilité cherchée. Cependant, elle fournit une probabilité correcte dans toutes les situations.

3.1.3 Validation et résultats

Cette section présente quelques résultats validant nos algorithmes sur des simulations simples. Nous réalisons cette validation en comparant les résultats obtenus par notre approche avec ceux déterminés par la résolution des équations différentielles de la loi d'action de masse. Ensuite, nous étudions la manière dont évoluent les performances de nos simulations en fonction du nombre de CPUs que nous utilisons.

3.1.3.1 Validation

Nous choisissons d'illustrer notre approche sur une réaction réversible et une réaction enzymatique car elles sont les plus fréquentes dans les systèmes biochimiques. Ces réactions sont modélisées par une réaction bimoléculaire et respectivement une et deux réactions unimoléculaires. Comme exemple pour la simulation des réactions réversibles, nous illustrons notre approche sur l'interaction du facteur de la coagulation Xa avec son inhibiteur, le TFPI. La description biochimique complète de cette interaction peut être trouvée dans [Baugh *et al.*, 1998]. Ensuite, nous illustrons notre méthode sur l'activation de la prothrombine (II) en thrombine (IIa) par un complexe enzymatique ($\mathrm{PT}_{\mathrm{Ptex}}$) qui peut se trouver dans le venin du serpent Australien *Pseudonaja textilis*. Cette interaction est complètement détaillée dans [Bos *et al.*, 2009]. Le tableau 3.2 rassemble toutes les données que nous utilisons pour paramétrer nos simulations de validation.

Les résultats obtenus lors de nos simulations sont illustrés sur la figure 3.3 et sont cohérents avec ceux provenant des lois de référence : l'erreur moyenne relative lors de l'état stationnaire est d'environ 1 %, aussi bien pour la réaction réversible que pour son homologue enzymatique. Ces deux validations sont essentielles avant de pouvoir simuler des situations plus complexes. Le lecteur peut remarquer que nous ne nous occupons pas des événements de fixation sur des membranes dans ces exemples. Ce point sera développé dans les perspectives de cette application.

3.1.3.2 Performances

Nous avons exécuté de nombreuses simulations, similaires aux précédentes, sur un seul ordinateur équipé de deux processeurs Intel Xeon X5680 doté chacun d'une vitesse d'horloge de 3.33 GHz et de 12 Mo de mémoire cache de niveau 3. Grâce à la technologie SMT à deux voies, les douze cœurs physiques peuvent être assimilés à vingt-quatre cœurs logiques pouvant fonctionner simultanément. La figure 3.4 représente la fréquence de calcul, c'est-à-

Réaction réversible	Réaction enzymatique
$\mathrm{Xa} + \mathrm{TFPI} \underset{k_{\mathrm{off}}}{\overset{k_{\mathrm{on}}}{\rightleftarrows}} \mathrm{Xa} \cdot \mathrm{TFPI}$	$\mathrm{PT_{Ptex}} + \mathrm{II} \underset{}{\overset{K_{\mathrm{M}}}{\rightleftarrows}} \mathrm{PT_{Ptex}} \cdot \mathrm{II} \overset{k_{\mathrm{cat}}}{\rightarrow} \mathrm{PT_{Ptex}} + \mathrm{IIa}$
$k_{\mathrm{on}} = 0.9 \times 10^{6}\,\mathrm{M}^{-1} \cdot \mathrm{s}^{-1}$ $k_{\mathrm{off}} = 3.6 \times 10^{-4}\,\mathrm{s}^{-1}$	$K_{\mathrm{M}} = 1.83 \times 10^{-6}\,\mathrm{M}$ $k_{\mathrm{cat}} = 5.87\,\mathrm{s}^{-1}$

Concentrations initiales	
$[\mathrm{Xa}]_0 = 170\,\mathrm{nM}$ $[\mathrm{TFPI}]_0 = 2.5\,\mathrm{nM}$	$[\mathrm{PT_{Ptex}}]_0 = 100\,\mathrm{nM}$ $[\mathrm{II}]_0 = 1.4\,\mu\mathrm{M}$

Dimensions						
Xa	TFPI	Xa·TFPI	$\mathrm{PT_{Ptex}}$	II	$\mathrm{PT_{Ptex}} \cdot \mathrm{II}$	IIa
$r_{\mathrm{x}} = 26.0\,\text{Å}$ $r_{\mathrm{y}} = 26.0\,\text{Å}$ $r_{\mathrm{z}} = 51.5\,\text{Å}$	$r = 22.5\,\text{Å}$	$r = 35.9\,\text{Å}$	$r = 39.3\,\text{Å}$	$r_{\mathrm{x}} = 22.5\,\text{Å}$ $r_{\mathrm{y}} = 22.5\,\text{Å}$ $r_{\mathrm{z}} = 60.0\,\text{Å}$	$r = 45.0\,\text{Å}$	$r = 25.0\,\text{Å}$

Tableau 3.2 – **Paramètres des simulations de validation** – Ce tableau présente les conditions de validation de notre modèle. Il rassemble l'ensemble des données nécessaires à la reproduction de nos résultats. Nous notons que la forme des facteurs Xa et II est considérée comme ellipsoïdale alors que les autres espèces ont une forme sphérique.

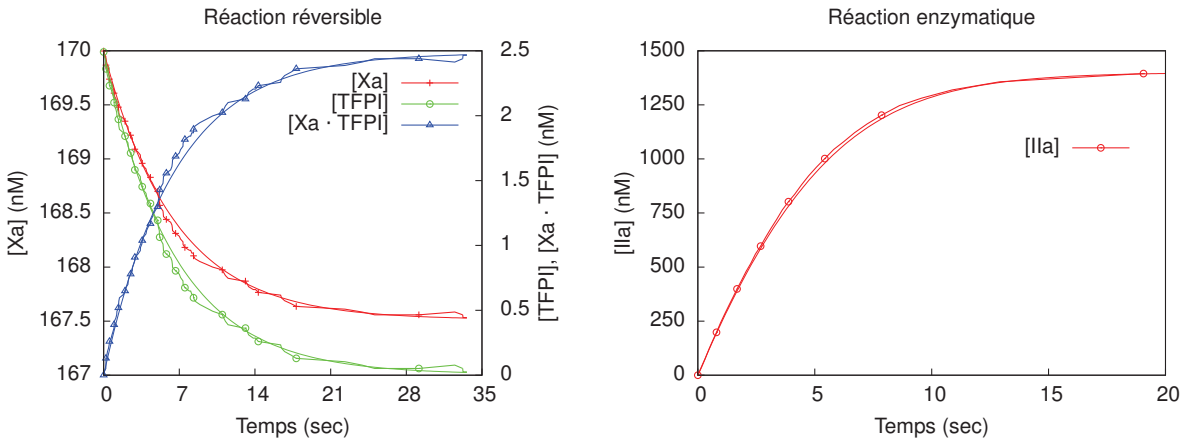


Figure 3.3 – **Validation des réactions réversible et enzymatique** – Ces courbes illustrent la validation de notre modèle sur deux types de réaction. Les courbes lisses proviennent de la résolution des équations de la loi d'action de masse alors que les courbes bruitées sont le résultat de nos simulations.

dire le nombre de cycles de simulation par seconde, ainsi que le nombre d'entités simulées pour une fréquence donnée, en fonction du nombre de CPUs utilisés. Les deux courbes semblent évoluer linéairement avec cependant un léger changement de pente aux alentours des douze CPUs. En effet, lorsque la simulation s'exécute sur la première douzaine de CPUs, les cœurs physiques exploitent pleinement leurs ressources matérielles et mémoires caches respectives, alors qu'en ce qui concerne la douzaine suivante, la technologie SMT à deux voies est mise à contribution, ce qui implique le partage des mêmes ressources matérielles pour une charge de travail deux fois plus élevée. Cela explique le fait que l'efficacité soit légèrement amoindrie.

Ces résultats montrent que, tant que les CPUs s'accompagnent de mémoires caches

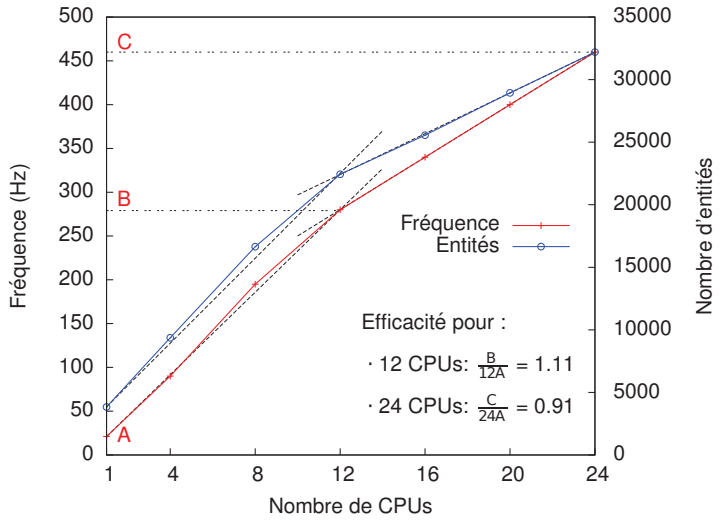


Figure 3.4 – **Évaluation des performances** – La courbe rouge illustre l'évolution de la fréquence de calcul en fonction du nombre de CPUs utilisés pour une simulation avec 22438 entités. La courbe bleue montre le nombre d'entités que notre simulateur peut exécuter à une fréquence donnée (280 Hz).

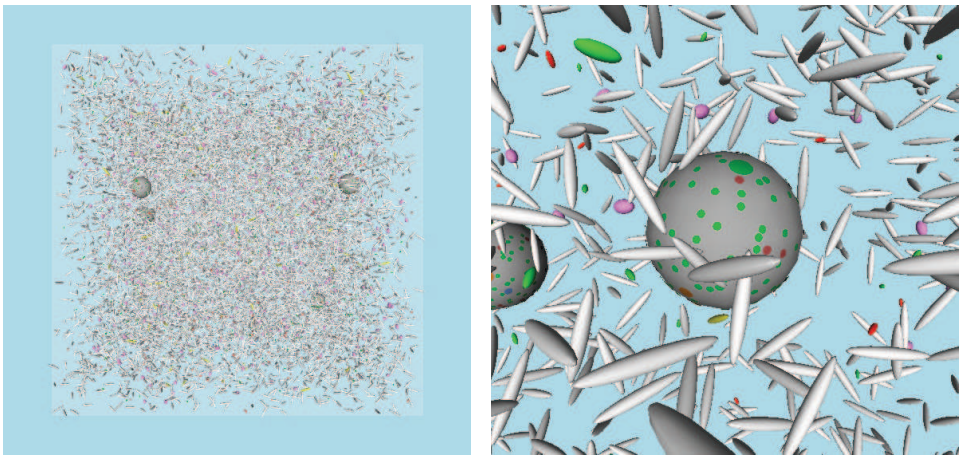
(cf. section 1.2.1.1), les performances des simulations augmentent linéairement en fonction du nombre de CPUs utilisés. Cela nous laisse présager que notre modèle individu-centré permettrait des simulations plus conséquentes, c'est-à-dire avec plus d'entités, ou plus rapides si la machine utilisée disposait de plus de vingt-quatre CPUs.

3.1.4 Discussion et perspectives

En raison du fait que nos simulations se déroulent à l'échelle microscopique, nos algorithmes de diffusion requièrent un pas de temps très court afin de ne manquer aucune collision (approximativement dix nanosecondes). Cette contrainte limite quelque peu les performances de nos simulations. Dans les faits, deux paramètres principaux régissent les performances : le nombre d'entités exécutées au cours de chaque cycle de simulation et la durée de ce cycle. La mise en œuvre du calcul parallèle nous permet d'augmenter le premier paramètre en utilisant plus d'unités de calcul. Cependant, cela n'a absolument aucun impact sur le second paramètre. Nous travaillons actuellement sur une méthode statistique permettant de remédier à ce problème. À l'heure actuelle, nous recommandons d'utiliser notre simulateur de cinétique biochimique pour des applications biologiques de courte durée, c'est-à-dire pour reproduire des phénomènes qui dans la réalité ne durent pas plus de dix secondes.

En sus de l'optimisation du pas de temps, une partie de notre modèle reste à valider. En effet, comme indiqué dans la section 3.1.2, les espèces moléculaires en solution peuvent se fixer sur des membranes physiologiques. Bien que de telles fixations soient complètement implémentées, nous devons nous assurer de la cohérence de nos résultats avec les résultats expérimentaux. Dans la situation actuelle, nous avons conscience que notre simulateur de

cinétique biochimique ne fournit rien de plus que les approches classiques à base d'équations différentielles puisque nous ne l'avons éprouvé que dans les conditions d'homogénéité spatiale dans lesquelles les équations différentielles sont utilisables. Néanmoins, cette méthode est la seule capable de prendre en compte les contraintes géométriques locales pouvant survenir au sein des systèmes biochimiques. Par conséquent, nous focalisons nos futurs travaux concernant cette application à la modélisation et à la validation de ces événements de fixation sur les membranes physiologiques. La figure 3.5 propose une visualisation en trois dimensions d'un exemple de simulation faisant intervenir des molécules en solution et des molécules fixées sur des membranes sphériques.



(a) Vue globale du système (b) Vue centrée sur une membrane

Figure 3.5 – **Illustration de notre simulateur de cinétique biochimique** – Ces deux captures d'écran illustrent une simulation faisant intervenir des molécules en solution (ellipsoïdes), des membranes physiologiques (sphères) et des molécules fixées sur ces membranes (disques). La capture 3.5a représente le volume réactionnel global tandis que la capture 3.5b est centrée sur l'une des membranes physiologiques présentes dans le système.

En faisant abstraction du côté applicatif du simulateur de cinétique biochimique que nous présentons ici, nous mettons en évidence le fait que notre approche individu-centrée, développée dans le chapitre 2 de ce mémoire, propose un gain de performance important lorsque le nombre de CPUs utilisés augmente. Ce gain permet soit d'augmenter le nombre d'entités simulées, soit d'accélérer les simulations. Bien que l'application biologique proposée ici n'apporte pas, en l'état, de contribution majeure à la simulation de la cinétique biochimique, elle constitue un moyen de validation de nos algorithmes. En effet, la simulation des systèmes complexes nécessite préalablement une étape de validation sur des exemples simples dont les résultats peuvent être aisément vérifiés grâce à des méthodes classiques (dans notre cas, les équations différentielles de la loi d'action de masse). Lorsque ces résultats sont validés, nous considérons que nos algorithmes peuvent être utilisés dans des situations plus complexes où les méthodes classiques ne peuvent pas s'appliquer, et c'est là que réside tout l'intérêt de la démarche de modélisation et de simulation. D'un point de vue purement informatique et malgré l'absence de validation des membranes physiologiques, les résultats présentés dans cette section nous semblent cependant très positifs.

Ces travaux sur la simulation parallélisée de la cinétique biochimique à l'échelle micro-

scopique ont abouti à la publication d'un article présenté lors de la conférence internationale ISBRA 2013 [Crépin *et al.*, 2013], ainsi qu'à la rédaction d'un poster pour ECCB 2012, une seconde conférence internationale (cf. figure A.1 de l'annexe A). La seconde partie de ce chapitre dédié à l'application de nos modèles aux phénomènes liés à la coagulation du sang propose de modéliser et de simuler un vaisseau sanguin virtuel.

3.2 Vaisseau sanguin virtuel

Dans cette section, nous nous intéressons à la modélisation et à la simulation des phénomènes de coagulation du sang au sein d'un vaisseau sanguin virtuel. Notre étude a ainsi pour objectif de modéliser la formation d'un caillot sanguin lorsqu'apparaît une brèche vasculaire, c'est-à-dire une lésion de la paroi du vaisseau, dans des conditions de flux les plus proches de celles existant *in vivo*. Pour simuler les multiples phénomènes survenant au sein de ce vaisseau sanguin, nous proposons d'utiliser notre approche mixte (individu et population-centrés) parallélisée à la fois sur des architectures multicœur et multiprocesseur ainsi que sur des architectures graphiques.

Cette section s'organise de la manière suivante. Tout d'abord, nous présentons quelques notions élémentaires d'hématologie. Ensuite, nous proposons une modélisation des phénomènes physiologiques prenant place au sein d'un vaisseau sanguin, et plus particulièrement l'écoulement plasmatique, la coagulation du sang et le comportement des cellules sanguines. Suite à cela, nous proposons une étude des performances obtenues lors de nos simulations ainsi qu'une présentation des applications potentielles de nos travaux. Enfin, nous concluons par une discussion afin de dégager quelques perspectives.

3.2.1 Notions élémentaires d'hématologie

Nous proposons dans cette section une brève explication du fonctionnement du système cardio-vasculaire responsable du transport du sang dans notre organisme. Cela nous permet d'introduire les phénomènes biologiques qu'il nous est nécessaire de modéliser pour simuler la coagulation du sang au sein d'un vaisseau sanguin.

3.2.1.1 Brefs rappels sur la circulation sanguine

Le sang est un fluide vital dont le rôle principal est d'assurer les apports en dioxygène et en nutriments nécessaires aux cellules composant les différents organes du corps humain. Il assure également le transport des déchets métaboliques produits par certains organes vers les organes responsables de leur élimination : c'est par exemple le cas du dioxyde de carbone qui est éliminé par les poumons. Le sang est composé de cellules, nommées éléments figurés, immergées dans un liquide à base d'eau, le plasma. Le transport du sang au travers du corps humain est assuré par un système circulatoire : la circulation sanguine. Son but est

d'acheminer le sang depuis le cœur vers les différents organes afin de les irriguer. Ce processus de transport s'effectue par l'intermédiaire du système cardio-vasculaire.

Le système cardio-vasculaire est constitué du cœur auquel est relié un réseau de vaisseaux sanguins de différents diamètres : les artères, les veines, les capillaires... Le fonctionnement du cœur peut être assimilé à celui d'une pompe permettant la circulation du sang au sein des vaisseaux. Dans des conditions physiologiques normales, le sang circule en circuit fermé : à tout instant, le système cardio-vasculaire contient exactement la même quantité de sang. Suite à une blessure, une brèche peut apparaître sur l'un des vaisseaux. Une telle rupture du circuit entraîne un saignement qu'il faut enrayer. Pour cela, le corps humain propose un processus de réponse : la coagulation du sang.

3.2.1.2 La coagulation du sang

La coagulation du sang ou hémostase est le processus physiologique responsable de la formation d'un caillot, ou thrombus, en réponse à l'apparition d'une brèche vasculaire. Le but de ce caillot est d'arrêter le saignement afin de limiter les pertes sanguines, nocives pour l'organisme. La pression intravasculaire étant supérieure à la pression extravasculaire, la rupture de la paroi d'un vaisseau va induire un saignement qui sera arrêté par l'apparition d'un thrombus obstruant la brèche. Afin d'assurer et restreindre la construction du caillot où et quand ce dernier est nécessaire, un judicieux équilibre entre mécanismes d'activations et d'inhibitions va en réguler la formation et la dissolution. Aussi, la rupture de cet équilibre induit deux familles de pathologies : les maladies hémorragiques (maladie de von Willebrand, hémophilies... : le patient saigne) et les maladies thrombotiques (infarctus du myocarde, phlébites... : le patient développe des caillots lorsque ce n'est pas nécessaire). C'est pourquoi l'hémostase est souvent comparée à une balance, comme illustré sur la figure 3.6.

La coagulation du sang s'articule autour de quatre grandes étapes :

La vasoconstriction correspond à la contraction du vaisseau qui se produit immédiatement suite à la lésion de la paroi. En diminuant le diamètre du vaisseau, elle ralentit l'écoulement sanguin et facilite les étapes qui suivent.

L'hémostase primaire est responsable de la formation du thrombus plaquettaire, un amas composé essentiellement de plaquettes se formant au niveau de la brèche vasculaire, notamment lorsque le collagène est présent au niveau de la lésion. Cet amas se forme par l'intermédiaire des phénomènes d'adhésion, d'activation et d'agrégation plaquettaires et permet d'interrompre partiellement le saignement en attendant la formation complète du caillot sanguin.

La coagulation plasmatique a pour but de consolider le thrombus formé précédemment en créant au sein de l'amas de plaquettes un réseau de fibrine, une protéine plasmatique. La formation du réseau de fibrine est l'aboutissement d'une succession de réactions biochimiques présentées sous la forme d'une cascade faisant intervenir de nombreuses protéines, les facteurs et inhibiteurs de la coagulation.

La fibrinolyse met en œuvre la dissolution du caillot lorsque la brèche a été résorbée. Elle permet, au travers d'un ensemble de réactions biochimiques, d'éliminer le caillot une

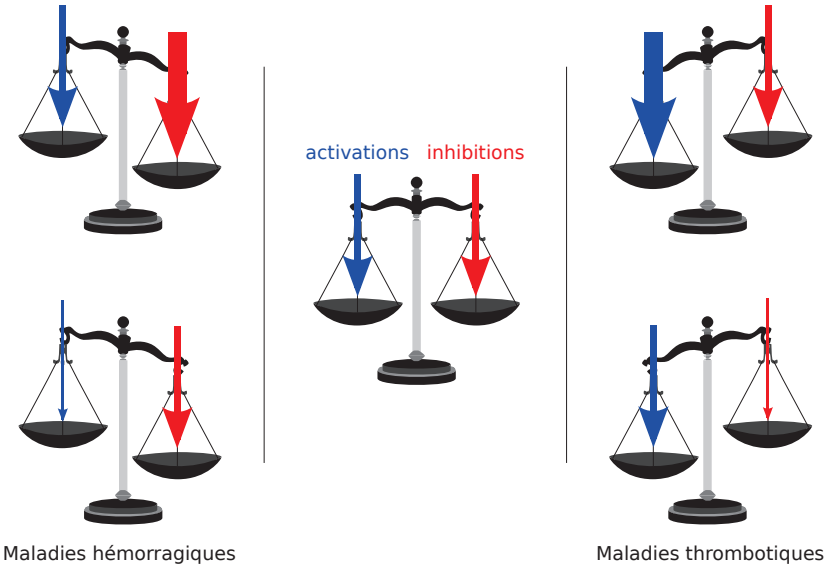


Figure 3.6 – **La balance hémostatique** – Ce schéma représente l'hémostase sous la forme d'une balance. Un patient est sain lorsque l'équilibre entre les mécanismes d'activation et d'inhibition est atteint. Si, l'un de ces deux mécanismes devient prépondérant face au second, le patient développe une maladie hémorragique (les inhibitions sont supérieures aux activations) ou thrombotique (les activations sont supérieures aux inhibitions).

fois que ce dernier n'est plus utile en scindant principalement le réseau de fibrine.

Ces quatre étapes sont indispensables au bon déroulement de l'hémostase et donc à la formation et à la dissolution du caillot. Toutefois dans un souci de simplification du problème, nous ne modélisons dans cette thèse que la coagulation plasmatique. La modélisation et la simulation des trois autres étapes feront l'objet de futurs travaux.

Le thrombus issu de la coagulation du sang est souvent comparé à un mur arrêtant l'hémorragie où les briques seraient les élément figurés (principalement les plaquettes mais aussi les globules rouges et les globules blancs) collées entre elles par un ciment (la fibrine) comme illustré sur la figure 3.7. Aussi, dans cette thèse, nous ne modélisons et simulons que l'apparition de ce « ciment » en condition de flux et en présence des éléments figurés. La section suivante donne plus de détails sur le fonctionnement de la coagulation plasmatique.

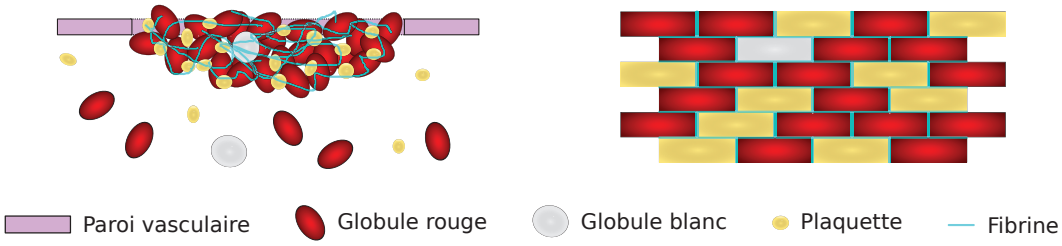


Figure 3.7 – **Représentation du thrombus** – Cette figure présente une analogie entre le thrombus issu de la coagulation du sang et un mur où les briques sont les éléments figurés collées entre elle par la fibrine jouant le rôle de ciment [2].

3.2.1.3 La coagulation plasmatique

La coagulation plasmatique est le processus initié par la mise en contact du facteur tissulaire (FT, protéine présente au niveau de la brèche) avec le plasma, et qui va aboutir à la transformation du fibrinogène soluble en un réseau de fibrine insoluble (le « ciment » du caillot). Tout comme l'hémostase, la coagulation plasmatique est régie par une balance entre activations et inhibitions :

Deux voies d'activation, nommées respectivement extrinsèque et intrinsèque, se déclinant en une succession de réactions biochimiques (essentiellement enzymatiques) font intervenir des protéines, les facteurs de la coagulation, numérotés de 1 à 13 en chiffres romains, avec le suffixe « a » lorsqu'ils sont sous leur forme active ;

Plusieurs inhibiteurs régulent les réactions enzymatiques précédentes : les trois systèmes d'inhibition de la coagulation actuellement les plus connus sont le TFPI (*Tissue Factor Pathway Inhibitor*), l'antithrombine (AT) ou encore le système protéine C (PC).

La coagulation plasmatique est par ailleurs classiquement comparée à une cascade, concept introduit par [Davie et Ratnoff, 1964] et [Macfarlane, 1964] et illustré dans une version simplifiée sur la figure 3.8. La modélisation de vaisseau sanguin que nous proposons dans la suite de ce mémoire s'appuie sur les notions élémentaires d'hématologie que nous venons d'introduire, et notamment celles concernant la coagulation plasmatique.

3.2.2 Modélisation mixte d'un vaisseau sanguin

La modélisation d'un vaisseau sanguin implique de manière évidente l'étude et la modélisation des phénomènes physiques et biologiques intervenant en son sein. Comme indiqué précédemment, le rôle des vaisseaux sanguins est de transporter le sang entre les différents organes du corps humain sous l'action du flux imposé par le cœur. Plutôt que de modéliser le sang dans son ensemble, nous choisissons de modéliser séparément l'écoulement plasmatique et les éléments figurés. Ainsi, à terme, nous pourrons étudier des phénomènes cellulaires tels que les mécanismes plaquettaires lors de l'hémostase primaire, ce qui ne serait pas envisageable si nous simulions le sang comme un unique composant (regroupant à la fois le plasma et les éléments figurés).

L'objectif principal de notre étude consiste à étudier la formation d'un caillot lorsqu'une brèche vasculaire apparaît au sein de la paroi d'un vaisseau sanguin. Ainsi, en sus de l'écoulement plasmatique et du comportement des cellules, il nous est nécessaire de modéliser les phénomènes liés à la coagulation du sang. Face à la complexité que représente la modélisation de l'ensemble des quatre étapes de l'hémostase (présentées dans la section 3.2.1.2), nous rappelons que nous choisissons de ne modéliser que la coagulation plasmatique. Ainsi, nous négligeons volontairement la vasoconstriction, l'hémostase primaire et la fibrinolyse.

2. Notons que la fibrinolyse, responsable de la dissolution du caillot, pourrait être assimilée à l'action d'un marteau-piqueur cassant le ciment du mur de briques.

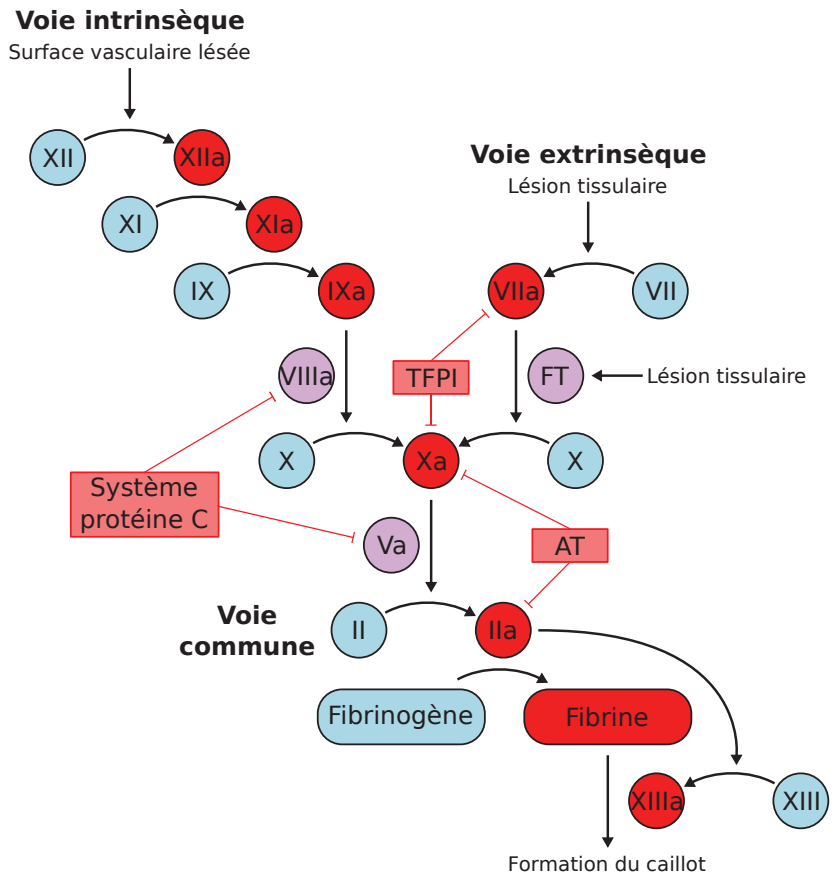


Figure 3.8 – **Cascade simplifiée de la coagulation** – Ce diagramme propose une version simplifiée de la cascade de coagulation, où de nombreux facteurs et inhibiteurs (respectivement représentés avec des formes arrondies et rectangulaires) interviennent. Bien qu'*in vitro* la cascade puisse être initiée de deux manières différentes, la voie extrinsèque joue un rôle prépondérant *in vivo*. Pour favoriser la lisibilité de ce schéma, les boucles de rétroactivation des facteurs V, VIII et XI par la thrombine (IIa) ont été omises.

3.2.2.1 Géométrie du problème

Un vaisseau sanguin est constitué d'une paroi vasculaire classiquement cylindrique séparant la zone intravasculaire (à l'intérieur du vaisseau) de la zone extravasculaire (à l'extérieur du vaisseau). Cette paroi est constituée d'un pavage de cellules endothéliales qui forment l'endothélium. Par souci de simplification, le domaine de modélisation sera restreint à un parallélépipède rectangle constitué des zones intravasculaire et extravasculaire ainsi que de la paroi vasculaire au sein de laquelle une brèche peut survenir. Cette simplification peut être considérée comme un grossissement effectué sur la zone d'intérêt de la paroi vasculaire, c'est-à-dire le lieu de création de la brèche. Nous supposons que cet agrandissement est suffisamment important pour que la courbure de la paroi soit négligeable, ce qui nous permet de l'approximer par une surface plane, plutôt que cylindrique. La brèche est quant à elle modélisée sous la forme d'un rectangle au centre de ce plan. La figure 3.9 illustre la géométrie de notre problème, et notamment le parallélépipède rectangle et la brèche vasculaire modélisés.

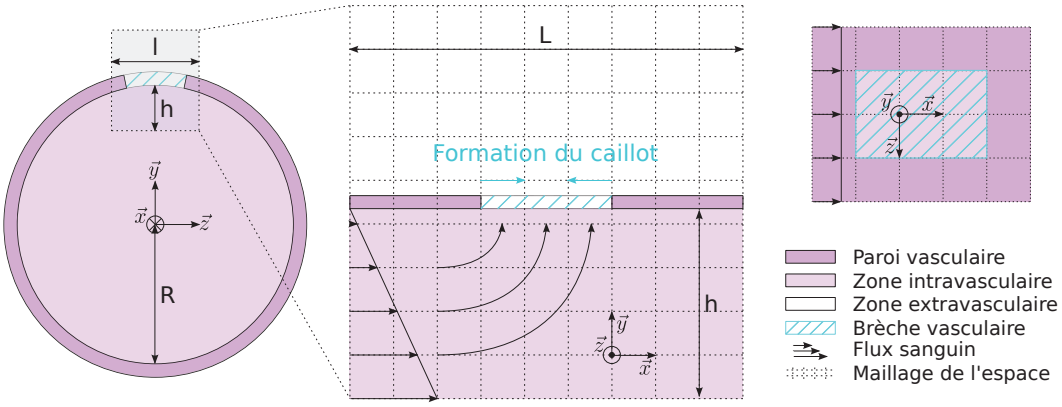


Figure 3.9 – **Géométrie du vaisseau sanguin virtuel** – Le schéma de gauche met en évidence l'emplacement du domaine parallélépipédique que nous étudions par rapport à la section du vaisseau. Le schéma central le représente en vue de profil en illustrant le maillage de l'espace, le système d'axes utilisé ainsi que le flux sanguin et la brèche. Le schéma de droite représente la brèche vasculaire en vue de dessus.

Afin d'effectuer les calculs modélisant les phénomènes prenant place au sein de notre vaisseau sanguin virtuel, nous proposons de discrétiser le domaine étudié par l'intermédiaire d'un maillage de l'espace en trois dimensions. Le domaine que nous modélisons étant parallélépipédique, nous proposons d'utiliser un maillage cartésien, particulièrement adapté à une telle géométrie. Ce maillage est nécessaire pour résoudre numériquement les EDPs composant notre modèle que nous introduisons dans la suite de ce mémoire. La figure 3.10 propose une représentation en trois dimensions de ce maillage.

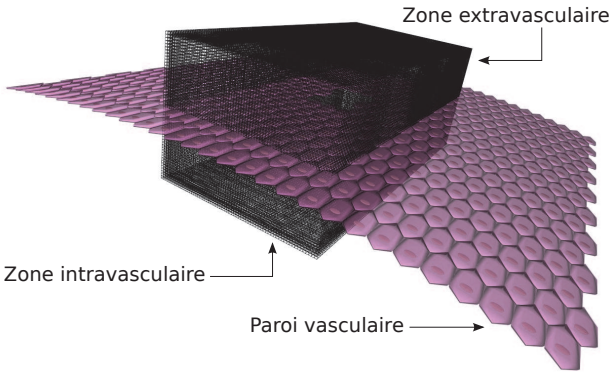


Figure 3.10 – **Représentation 3D du maillage du vaisseau sanguin virtuel** – Cette figure illustre le parallélépipède rectangle représentant le domaine modélisé ainsi que le maillage de l'espace utilisé lors des calculs. Elle propose également une représentation de la paroi vasculaire, nommée endothélium et formée par un pavage de cellules endothéliales.

Considérer un domaine parallélépipédique focalisé sur la brèche vasculaire plutôt qu'un domaine cylindrique représentant l'intégralité de la section du vaisseau offre deux avantages. Le premier provient du fait qu'en limitant le volume étudié, nous minimisons le nombre d'éléments à modéliser, notamment au niveau des éléments figurés. De surcroît, cela nous permet également de limiter le nombre de mailles de notre maillage, ce qui maximise les performances de nos simulations. Le second avantage provient de l'utilisation d'un

maillage cartésien plutôt que cylindrique, ce qui simplifie fortement la mise en équations des phénomènes présentés dans la suite de ce document.

Le sang circule habituellement à l'intérieur du vaisseau, dans la zone intravasculaire. Suite à une lésion vasculaire, le sang s'échappe par la brèche dans la zone extravasculaire. Afin d'arrêter les saignements, la formation du caillot sanguin a lieu au niveau de la brèche vasculaire. La figure 3.9, introduite précédemment, propose des schémas représentant la géométrie du problème lorsque cette situation se produit. Ils présentent notamment le système d'axes utilisé dans notre modélisation : le sang s'écoule selon l'axe $\vec{x}$ et l'hémorragie a lieu selon l'axe $\vec{y}$. Notons que pour une meilleure lisibilité des schémas, le système d'axes n'est pas positionné à l'origine du repère. Il illustre uniquement le sens et l'orientation des trois axes de l'espace. La figure présente également quelques notations utilisées plus tard dans ce mémoire, notamment le rayon R du vaisseau et la hauteur h, la longueur L et la largeur l de la zone intravasculaire au sein de laquelle surviennent les phénomènes physiologiques dont les modélisations sont détaillées dans les sections suivantes.

3.2.2.2 Modélisation du flux sanguin

***Les équations de* Navier-Stokes**

Le flux sanguin correspond à l'écoulement du plasma. Contrairement au sang total, le plasma est un fluide dit Newtonien [Marder *et al.*, 2012]. Mécaniquement, cela signifie que les contraintes de cisaillement sont proportionnelles aux vitesses de déformation. Les fluides Newtoniens sont généralement modélisés par l'intermédiaire des EDPs de *Navier-Stokes* où les inconnues sont la vitesse et la pression du fluide [Versteeg et Malalasekera, 2007]. Dans notre cas, nous considérons le fluide étudié, c'est-à-dire le plasma sanguin, comme étant un liquide incompressible. Notons qu'en réalité les liquides sont compressibles, mais très difficilement. Cette approximation est donc raisonnable et permet de fortement simplifier les équations de *Navier-Stokes*. Dans le cadre de l'étude d'une fluide Newtonien incompressible, ces équations s'écrivent sous la forme d'une équation dite de continuité et d'une équation dite de quantité de mouvement :

$$\left\{ \begin{array}{rcl} \vec{\nabla} \cdot \vec{v} & = & 0 \\ \dfrac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \vec{\nabla})\vec{v} & = & -\dfrac{1}{\rho}\vec{\nabla} p + \nu \nabla^2 \vec{v} \end{array} \right. \tag{3.5}$$

où

- t désigne le temps ;
- $\vec{v}$ désigne la vitesse du fluide ;
- p désigne la pression du fluide ;
- ρ désigne la masse volumique du fluide ;
- $\nu = \frac{\mu}{\rho}$ désigne la viscosité cinématique du fluide ;
- μ désigne la viscosité dynamique du fluide.

Bien que le vaisseau sanguin étudié soit cylindrique, nous n'en modélisons qu'une petite portion parallélépipédique au niveau de la brèche vasculaire. Cette géométrie, introduite dans la section 3.2.2.1, nous incite à utiliser des coordonnées cartésiennes pour exprimer les équations. L'équation de continuité devient donc :

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} = 0 \ , \tag{3.6}$$

tandis que l'équation de bilan de la quantité de mouvement devient :

$$\begin{cases} \dfrac{\partial v_x}{\partial t} + v_x \dfrac{\partial v_x}{\partial x} + v_y \dfrac{\partial v_x}{\partial y} + v_z \dfrac{\partial v_x}{\partial z} = -\dfrac{1}{\rho}\dfrac{\partial p}{\partial x} + \nu \left(\dfrac{\partial^2 v_x}{\partial x^2} + \dfrac{\partial^2 v_x}{\partial y^2} + \dfrac{\partial^2 v_x}{\partial z^2} \right) \\ \dfrac{\partial v_y}{\partial t} + v_x \dfrac{\partial v_y}{\partial x} + v_y \dfrac{\partial v_y}{\partial y} + v_z \dfrac{\partial v_y}{\partial z} = -\dfrac{1}{\rho}\dfrac{\partial p}{\partial y} + \nu \left(\dfrac{\partial^2 v_y}{\partial x^2} + \dfrac{\partial^2 v_y}{\partial y^2} + \dfrac{\partial^2 v_y}{\partial z^2} \right) \\ \dfrac{\partial v_z}{\partial t} + v_x \dfrac{\partial v_z}{\partial x} + v_y \dfrac{\partial v_z}{\partial y} + v_z \dfrac{\partial v_z}{\partial z} = -\dfrac{1}{\rho}\dfrac{\partial p}{\partial z} + \nu \left(\dfrac{\partial^2 v_z}{\partial x^2} + \dfrac{\partial^2 v_z}{\partial y^2} + \dfrac{\partial^2 v_z}{\partial z^2} \right) \end{cases} \tag{3.7}$$

où (v_x, v_y, v_z) désigne le triplet de composantes du vecteur vitesse $\vec{v}$ selon les axes $\vec{x}$, $\vec{y}$ et $\vec{z}$.

Résolution numérique

En raison des termes non linéaires qui les composent, les équations de *Navier-Stokes* n'acceptent de solutions analytiques que dans de très rares situations. Leur résolution requiert donc l'utilisation de méthodes numériques telles que les éléments finis, les volumes finis ou encore les différences finies. Ces méthodes sont détaillées dans la section 2.2.1 de ce mémoire. Néanmoins, la mise en œuvre de telles méthodes n'est pas suffisante pour résoudre les équations de *Navier-Stokes*. En effet, ces équations sont faiblement couplées. L'équation de quantité de mouvement permet de déterminer un champ de vitesse uniquement si le gradient de pression est *a priori* connu. Or, ce gradient fait partie des inconnues du système, il ne peut donc pas être fixé arbitrairement. De ce fait, il est nécessaire d'utiliser un algorithme de couplage pression-vitesse pour résoudre ces équations. L'un des plus aisés à appréhender se nomme *Semi-Implicit Method for Pressure-Linked Equation* (SIMPLE). Notons que de nombreuses alternatives existent telles que SIMPLER [3], SIMPLEC [4] ou encore PISO [5].

Afin de ne pas trop alourdir la lecture de cette section, nous ne détaillons pas le fonctionnement du SIMPLE dans ce mémoire. Cependant, le lecteur curieux pourra se référer aux publications suivantes pour obtenir de plus amples informations à son sujet : [Patankar, 1980] et [Versteeg et Malalasekera, 2007]. Nous résumons l'algorithme du SIMPLE de la manière suivante :

1. Proposer un champ de pression $p^\star$.
2. Discrétiser le domaine selon la méthode des volumes finis et résoudre l'équation de quantité de mouvement pour obtenir un champ de vitesse $\vec{v}^\star$ à partir de $p^\star$.

3. SIMPLE Revised
4. SIMPLE Corrected/Consistent
5. Pressure Implicit with Splitting Operators

3. Transformer l'équation de continuité en une équation permettant de déterminer la correction p' à apporter au champ de pression $p^\star$.
4. Résoudre l'équation précédemment obtenue et déterminer p'.
5. Calculer le nouveau champ de pression p en sommant $p^\star$ et p'.
6. Utiliser la formule de correction des vitesses (obtenue à partir de l'équation de quantité de mouvement) pour déterminer le nouveau champ de vitesse $\vec{v}$ corrigé.
7. Retourner à l'étape 2 et répéter le processus (en utilisant la nouvelle valeur de pression corrigée p comme pression proposée $p^\star$) jusqu'à ce que l'algorithme ait convergé.

Conditions initiales

La résolution numérique nécessite de connaître les conditions initiales du problème. Elles définissent l'état du système à l'initialisation de la simulation. Nous supposons qu'à cet instant, la paroi du vaisseau est encore intacte, c'est-à-dire sans lésion vasculaire. Dans cette situation, l'écoulement sanguin à l'intérieur du vaisseau (dans la zone intravasculaire) est modélisé par un écoulement de Couette comme illustré sur la figure 3.11.

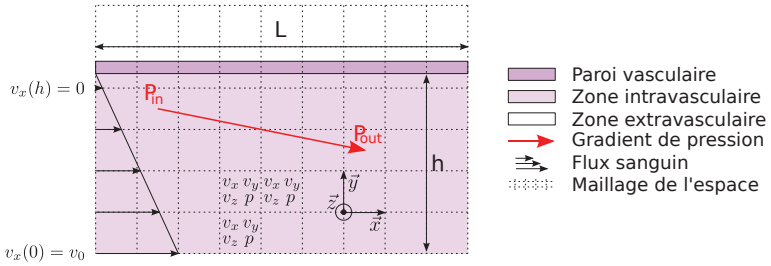


Figure 3.11 – **Écoulement de Couette** – Ce schéma représente un écoulement de Couette selon l'axe $\vec{x}$ ainsi que le gradient décroissant de pression, entre P_{in} et P_{out} lui étant associé. Les valeurs v_x, v_y, v_z et p présentes à l'intérieur de chacune des mailles correspondent respectivement aux composantes de vitesse et à la valeur de pression du plasma. La valeur v_0 désigne la vitesse maximale présente dans la tranche inférieure du maillage, c'est-à-dire pour $y = 0$.

Selon l'axe $\vec{x}$ et en respectant les notations introduites précédemment, l'équation d'un tel écoulement s'écrit :

$$\frac{\partial^2 v_x(y)}{\partial y^2} = \frac{1}{\mu}\frac{\partial p}{\partial x}. \tag{3.8}$$

Les conditions aux limites sont données par $v_x(0) = v_0$ et $v_x(h) = 0$, où v_0 désigne la vitesse maximale du fluide dans la tranche inférieure du maillage, c'est-à-dire pour $y = 0$, et h désigne la hauteur de la zone intravasculaire modélisée. Notons que la vitesse est nulle selon les axes perpendiculaires à l'écoulement, c'est-à-dire $\vec{y}$ et $\vec{z}$.

À l'extérieur du vaisseau, dans la zone extravasculaire, la vitesse est nulle et la pression est supposée constante. En effet, la paroi vasculaire n'est pas encore lésée et donc ni le plasma ni les cellules sanguines ne peuvent s'échapper de la zone intravasculaire.

3.2.2.3 Modélisation de la coagulation plasmatique

L'équation d'advection-diffusion-réaction

Notre modélisation compte une trentaine d'espèces chimiques interagissant par l'intermédiaire d'une quarantaine de réactions biochimiques (de premier ordre, de second ordre et enzymatiques). La majorité de ces espèces sont présentes sur le schéma de la cascade de la coagulation que nous proposons sur la figure 3.8. À cela s'ajoutent de nombreux complexes chimiques constitués de plusieurs protéines liées entre elles (par exemple le complexe FT.VIIa liant le facteur tissulaire au facteur VII activé) [6]. Ces espèces chimiques sont sujettes à l'action de trois phénomènes biologiques :

La diffusion est un processus de transport ayant tendance à homogénéiser les concentrations chimiques des espèces à l'intérieur d'un système biologique. Cela se traduit par une migration des molécules depuis les zones de forte concentration vers les zones de faible concentration. Un exemple classique est la dispersion d'une goutte d'encre colorée dans un liquide tel que l'eau. Mathématiquement, la diffusion est définie par les deux lois de Fick :

$$J = -D\nabla C \text{ et } \frac{\partial C}{\partial t} = D\Delta C \tag{3.9}$$

où C est la concentration de l'espèce chimique étudiée, D est son coefficient de diffusion et J est le flux de diffusion.

L'advection est un second processus de transport caractérisant le mouvement d'espèces chimiques en solution sous l'action d'un écoulement. Le déplacement de substances solubles sous l'action d'un courant marin est un exemple de phénomène d'advection. Mathématiquement, ce processus est caractérisé par l'équation :

$$\frac{\partial C}{\partial t} + \vec{v} \cdot \nabla C = 0 \tag{3.10}$$

où C est la concentration de l'espèce chimique étudiée et $\vec{v}$ est la vitesse du fluide au sein duquel l'espèce est immergée ou dissoute.

Les réactions sont des processus entraînant la transformation d'un ensemble d'espèces chimiques (les réactifs) en un autre (les produits). Si nous considérons une réaction de premier ordre décrite par le schéma suivant :

$$\mathrm{A} \overset{k}{\rightarrow} \mathrm{B} \tag{3.11}$$

où A est un réactif, B est un produit et la constante de réaction k spécifie la vitesse de la réaction, alors sa cinétique est décrite par le système d'EDOs suivant :

$$\frac{d[\mathrm{A}]}{dt} = -\frac{d[\mathrm{B}]}{dt} = -k \times [\mathrm{A}]. \tag{3.12}$$

Notons que [A] et [B] désignent respectivement la concentration de A et de B. Si nous considérons une réaction de second ordre décrite par le schéma suivant :

$$\mathrm{A} + \mathrm{B} \overset{k}{\rightarrow} \mathrm{C} \tag{3.13}$$

6. Pour des raisons de confidentialité vis-à-vis de Diagnostica Stago, nous ne détaillons dans ce mémoire ni les espèces chimiques peuplant notre modèle, ni les réactions biochimiques qui les lient.

où A et B sont des réactifs, C est un produit et la constante de réaction k spécifie la vitesse de la réaction, alors sa cinétique est décrite par le système d'EDOs suivant :

$$\frac{d[\mathrm{A}]}{dt} = \frac{d[\mathrm{B}]}{dt} = -\frac{d[\mathrm{C}]}{dt} = -k \times [\mathrm{A}] \times [\mathrm{B}]. \tag{3.14}$$

Si nous considérons une réaction enzymatique décrite par le schéma suivant :

$$\mathrm{E} + \mathrm{S} \quad \overset{k_{\mathrm{M}}}{\rightleftarrows} \quad \mathrm{E} \cdot \mathrm{S} \quad \overset{k_{\mathrm{cat}}}{\rightarrow} \quad \mathrm{E} + \mathrm{P} \tag{3.15}$$

où E est un enzyme, S est un substrat, P est un produit et où les différentes constantes de réaction k_{cat} et k_{M} spécifient la vitesse des réactions, alors sa cinétique est décrite par le système d'EDOs suivant :

$$\frac{d[\mathrm{S}]}{dt} = -\frac{d[\mathrm{P}]}{dt} = -\frac{k_{\mathrm{cat}} \times [\mathrm{E}] \times [\mathrm{S}]}{k_{\mathrm{M}} + [\mathrm{S}]} \tag{3.16}$$

connu sous le nom d'équation de Henri-Michaelis-Menten [Henri, 1903; Michaelis et Menten, 1913].

L'ensemble de ces trois phénomènes forme un système d'advection-diffusion-réaction qui peut être modélisé par l'EDP suivante :

$$\frac{\partial C}{\partial t} + \vec{v} \cdot \nabla C = D \Delta C + R \tag{3.17}$$

où R est composé de termes provenant des différents systèmes d'EDOs décrivant les réactions modélisées. Les autres termes proviennent des équations définissant l'advection et la diffusion présentées précédemment. En coordonnées cartésiennes, cette équation s'écrit de la manière suivante :

$$\frac{\partial C}{\partial t} + v_x \frac{\partial C}{\partial x} + v_y \frac{\partial C}{\partial y} + v_z \frac{\partial C}{\partial z} = D \left(\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2} + \frac{\partial^2 C}{\partial z^2} \right) + R. \tag{3.18}$$

Résolution numérique

À l'instar des équations de *Navier-Stokes*, cette EDP ne dispose pas de solution analytique et il est donc nécessaire d'utiliser des méthodes numériques pour pouvoir déterminer l'évolution d'une concentration au cours du temps et de l'espace. Ainsi, le phénomène de coagulation plasmatique est modélisé par un gigantesque système composé de nombreuses EDPs de la forme de l'équation 3.18 (une équation par concentration C d'espèce biochimique simulée). Pour résoudre ce système, nous utilisons le maillage établi lors de la résolution des équations de *Navier-Stokes*. En discrétisant l'espace de la sorte et en appliquant la méthode des différences finies sur les trois variables spatiales, nous transformons le système d'EDPs en une multitude de sous-systèmes composés uniquement d'EDOs. Chacun de ces sous-systèmes pilote l'évolution des concentrations en facteurs et inhibiteurs de la coagulation présents au sein d'une maille. Ils dépendent uniquement du temps et des concentrations des mailles

adjacentes. Ils s'écrivent comme un ensemble d'équations de la forme suivante :

$$\begin{aligned}
\frac{\partial C(t,x,y,z)}{\partial t} &= -v_x \frac{C(t,x+\Delta x,y,z) - C(t,x-\Delta x,y,z)}{2\Delta x} \\
&\quad -v_y \frac{C(t,x,y+\Delta y,z) - C(t,x,y-\Delta y,z)}{2\Delta y} \\
&\quad -v_z \frac{C(t,x,y,z+\Delta z) - C(t,x,y,z-\Delta z)}{2\Delta z} \\
&\quad +D\left(\frac{C(t,x+\Delta x,y,z) - 2C(t,x,y,z) + C(t,x-\Delta x,y,z)}{\Delta x^2}\right. \\
&\quad +\frac{C(t,x,y+\Delta y,z) - 2C(t,x,y,z) + C(t,x,y-\Delta y,z)}{\Delta y^2} \\
&\quad \left.+\frac{C(t,x,y,z+\Delta z) - 2C(t,x,y,z) + C(t,x,y,z-\Delta z)}{\Delta z^2}\right) + R
\end{aligned} \tag{3.19}$$

où les triplets (x, y, z) et $(\Delta x, \Delta y, \Delta z)$ spécifient respectivement les coordonnées cartésiennes de la maille étudiée et les pas de discrétisation spatiale, c'est-à-dire la distance entre le centre de deux mailles adjacentes. Nous notons que la discrétisation spatiale mise en œuvre dans cette équation utilise un schéma de différences finies centrées (d'ordres un et deux) où chaque maille consulte les concentrations de sa voisine de droite et de celle de gauche (selon l'axe concerné). Cependant, au sein des mailles présentes aux extrémités du maillage, ce schéma ne peut pas s'appliquer et donc il est nécessaire d'adapter cette équation en mettant en place des schémas de différences finies à droite ou à gauche.

Suite à cette discrétisation spatiale, une telle EDO peut se résoudre aisément en appliquant une méthode numérique telle que celle d'Euler. En effet, les concentrations présentes au sein de notre maillage varient relativement peu d'un pas de temps à l'autre et donc nous pouvons nous permettre d'utiliser la méthode d'Euler bien qu'elle ne soit que d'ordre (de convergence) un. Cependant, si ces variations de concentration avaient été plus conséquentes, il aurait été nécessaire d'utiliser une méthode d'ordre plus élevé telle que la méthode de Runge-Kutta quatre implicite, ce qui entraînerait des temps de calcul plus importants, et donc une chute des performances.

3.2.2.4 Modélisation des cellules

Un système individu-centré composé de cellules

Le sang comporte trois types d'éléments figurés différents, chacun ayant un rôle spécifique :

Les globules rouges, ou érythrocytes, sont des cellules ayant la forme d'un disque biconcave [7]. Ils contiennent une protéine, l'hémoglobine, qui leur permet d'assurer le transport du dioxygène entre les poumons et les différents organes du corps humain.

7. Disque dont la zone centrale est moins épaisse que la périphérie.

Les globules blancs, ou leucocytes, sont des cellules participant au système immunitaire responsable de la reconnaissance et de la défense contre les agents infectieux. Leur forme s'approche de celle d'une sphère.

Les plaquettes, ou thrombocytes, sont des cellules dont la forme est similaire à celle d'un ellipsoïde. Elles sont indispensables au bon déroulement de l'hémostase primaire et contribuent à la formation du thrombus plaquettaire se formant au niveau d'une brèche vasculaire pour interrompre les saignements.

À l'instar des molécules présentes dans l'application de cinétique biochimique détaillée dans la première partie de ce chapitre, chacun des éléments figurés modélisés est représenté individuellement. Ainsi, l'ensemble des cellules immergées dans le plasma constitue un système individu-centré que nous simulons grâce au modèle que nous avons développé dans le chapitre 2. Tour à tour, chaque cellule est exécutée de manière asynchrone au cours d'un cycle de simulation. Son cycle de vie est défini par le comportement suivant.

Comportement d'une cellule

Les éléments figurés sont immergés dans le plasma. De ce fait, leur mouvement est dicté par l'action qu'exerce l'écoulement sanguin sur eux. Cette action se nomme force de traînée et s'exprime de la manière suivante :

$$\vec{F}_{\mathrm{T}} = \frac{1}{2}\,\rho\, C_{\mathrm{T}}\, S\,(\vec{v}_{\mathrm{f}} - \vec{v})^2 \tag{3.20}$$

où

- ρ désigne la masse volumique du fluide ;
- C_{T} désigne le coefficient de traînée de l'objet immergé ;
- S désigne la surface de référence ;
- $\vec{v}_{\mathrm{f}}$ désigne la vitesse du fluide ;
- $\vec{v}$ désigne la vitesse de l'objet immergé.

Notons que le coefficient de traînée ainsi que la surface de référence dépendent directement de la forme de l'objet étudié.

Le principe fondamental de la dynamique indique que :

$$\sum \vec{F}_{\mathrm{ext}} = m\,\vec{a} \tag{3.21}$$

où $\vec{F}_{\mathrm{ext}}$ désigne une force extérieure appliquée au solide S (ici la cellule étudiée), m désigne sa masse et $\vec{a}$ son accélération. Cette équation décrit le mouvement en translation de nos cellules, et plus généralement d'un solide quelconque. En ce qui concerne la rotation, ce principe s'écrit :

$$\sum \vec{M}_{\mathrm{G}_{\mathrm{ext}}} = J_{\mathrm{G}}\,\vec{\alpha} \tag{3.22}$$

où $\vec{M}_{\mathrm{G}_{\mathrm{ext}}}$ désigne le moment du solide S en son centre de gravité G induit par une force extérieure, J_{G} son moment d'inertie et $\vec{\alpha}$ son accélération angulaire.

Ces deux équations permettent de déterminer l'accélération linéaire et angulaire d'une cellule sous l'action de la force de traînée exercée par l'écoulement plasmatique. Pour cela,

il est nécessaire de les intégrer puisque l'accélération est définie comme étant la dérivée de la vitesse. Ces intégrations permettent ainsi d'obtenir la vitesse linéaire et angulaire d'une cellule et par conséquent son déplacement, aussi bien en translation qu'en rotation, au cours d'un pas de temps de simulation.

Ainsi, à chaque cycle de simulation, chacune des cellules de notre modèle individu-centré perçoit la vitesse au sein des mailles dans lesquelles elle se situe. Cette vitesse permet de calculer une ou plusieurs forces de traînée qui à leur tour permettent de déterminer le déplacement en translation et en rotation de la cellule grâce au principe fondamental de la dynamique. Le déplacement ainsi calculé permet de faire évoluer la position de la cellule au sein du vaisseau. Dans l'état actuel de nos travaux, seul ce comportement est implémenté ; les cellules n'ont aucun impact sur la coagulation du sang.

3.2.2.5 Implémentation des modèles

Les sections précédentes présentent les différents modèles que nous proposons de mettre en œuvre pour simuler notre vaisseau sanguin virtuel. La figure 3.12 illustre l'ensemble des phénomènes que nous modélisons et spécifie le type de modélisation que nous choisissons d'utiliser pour chacun d'eux. Elle synthétise ce que nous avons détaillé précédemment et regroupe les modèles que nous implémentons dans notre application. À l'instar de notre état de l'art, nous distinguons nos modélisations en deux catégories : les systèmes d'équations différentielles et les systèmes individu-centrés.

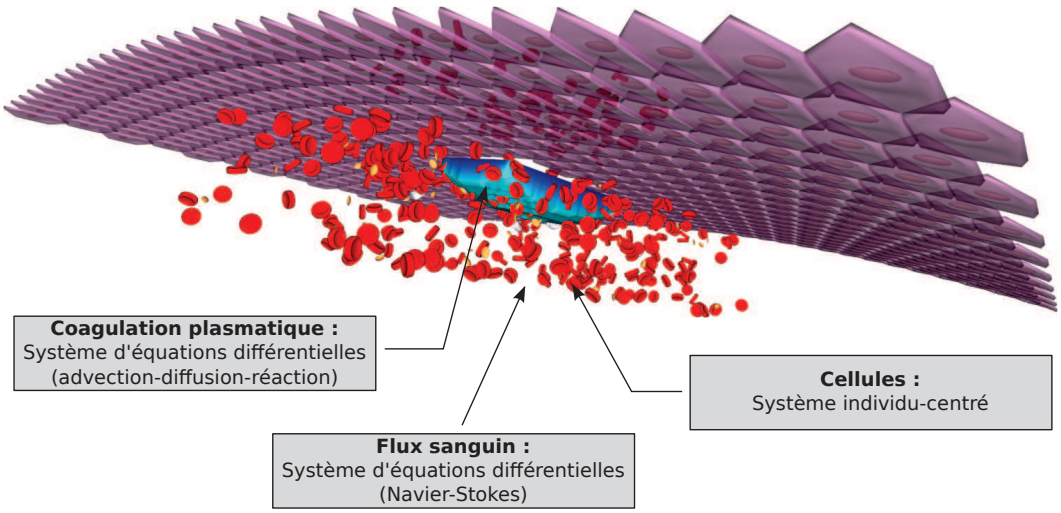


Figure 3.12 – **Phénomènes présents au sein du vaisseau sanguin virtuel** – Cette figure propose un visuel de notre vaisseau sanguin virtuel et met en évidence les phénomènes présents en son sein. En outre, le type de modélisation envisagé pour chacun d'eux est précisé.

Les systèmes d'équations différentielles

La résolution numérique des équations de *Navier-Stokes*, par l'algorithme du SIMPLE (basé sur la méthode des volumes finis), et celle des équations d'advection-diffusion-réaction, par la méthode d'Euler et des différences finies, nécessitent la mise en place d'un maillage de l'espace. Cette discrétisation spatiale provoque une fragmentation des problèmes étudiés (écoulement et coagulation plasmatique) entre l'ensemble des mailles. Ainsi, chacune d'elles est responsable de l'évolution d'un vecteur vitesse $\vec{v}$ et d'un scalaire pression, tous deux caractérisant le flux sanguin, ainsi que d'un ensemble de concentrations chimiques décrivant la quantité de protéines plasmatiques en son sein à un instant t donné. Comme indiqué précédemment dans la section 2.2, de telles approches population-centrées utilisant un maillage de l'espace se prêtent particulièrement bien à une implémentation sur GPU. Cela provient du fait que les traitements à réaliser au sein de chacune des mailles sont sensiblement identiques, ce qui permet d'exploiter au mieux l'architecture SIMT des GPUs. De ce fait, nous proposons une implémentation parallélisée sur GPU de l'algorithme du SIMPLE (pour la résolution des équations de *Navier-Stokes*) et de la méthode d'Euler (pour la résolution des équations d'advection-diffusion-réaction). Cette implémentation permet ainsi à de nombreuses mailles d'être traitées simultanément par des *threads* distincts. Elle consiste à développer sept *kernels*. Les six premiers implémentent chacun une partie de l'algorithme du SIMPLE, comme par exemple la résolution de l'équation de quantité de mouvement ou la correction des vitesses et pressions, alors que le septième implémente la méthode d'Euler. L'exécution de chacun de ces *kernels* est ponctuée par une barrière de synchronisation. En effet, pour garantir la cohérence des résultats de nos simulations, nous devons nous assurer que chaque *kernel* a terminé son traitement avant d'exécuter le suivant. Nous évaluons l'impact de ces barrières de synchronisation sur les performances de nos simulations dans la suite de ce document. Notons qu'en raison de la taille conséquente des différents codes sources (plusieurs centaines de lignes chacun) définissant l'ensemble des sept *kernels* que nous proposons, nous ne les présentons pas dans ce mémoire.

Afin de potentiellement améliorer les performances de nos simulations, nous pouvons, plutôt que d'utiliser un seul et unique GPU, en utiliser plusieurs, qui plus est accompagnés des CPUs de la machine hôte. En effet, dans le chapitre 2 de ce mémoire nous avons décrit une méthode permettant de simuler les modèles population-centrés sur plusieurs GPUs et également sur les CPUs. Nous pouvons tout à fait appliquer cette méthode à nos implémentations du SIMPLE et de la méthode d'Euler. L'utilisation de plusieurs GPUs requiert un découpage du maillage global en autant de sous-maillages que la machine hôte compte de GPUs (cf. figure 2.10 page 64 pour un exemple avec deux GPUs). La participation des CPUs nécessite la création d'un sous-maillage supplémentaire qui leur est attribué. Ce découpage du maillage global requiert des transferts de données survenant au niveau des frontières des sous-maillages. En effet, en raison des dérivées partielles spatiales présentes dans les équations de *Navier-Stokes* et d'advection-diffusion-réaction, chaque maille a besoin de consulter le contenu de ses voisines directement adjacentes. Ainsi, les mailles frontalières, c'est-à-dire à l'interface entre deux sous-maillages, doivent nécessairement consulter le contenu de mailles appartenant à un autre sous-maillage traité par un autre GPU ou par les CPUs. Pour que chaque sous-maillage prenne connaissance des modifications apportées par ses voisins, il doit donc recopier les données contenues dans les mailles frontalières leur appartenant. Ces transferts ont lieu immédiatement avant l'exécution des *kernels* nécessitant

l'accès au contenu de mailles voisines. Cela permet de garantir la cohérence des données manipulées par chacun d'eux. Nous notons que la participation des CPUs nécessite la mise en place de barrières de synchronisation supplémentaires afin de garantir qu'ils aient tous terminé leur traitement avant de passer au suivant. Nous mesurons l'impact de ces transferts de données sur les performances lors de la présentation des résultats de nos simulations. Ainsi, l'exécution de chaque *kernel* suit le protocole suivant :

1. Transfert des données contenues dans les mailles frontalières (si besoin) ;
2. Demande d'exécution du *kernel* concerné sur un ou plusieurs GPUs ainsi que les CPUs ;
3. Synchronisation de l'ensemble des *threads* participant à l'exécution du *kernel* (à la fois sur GPU et CPU).

Ce protocole est répété à chaque cycle de simulation pour l'exécution de chacun des sept *kernels* développés.

Le système individu-centré

L'implémentation de notre modèle individu-centré décrivant le comportement des cellules sanguines suit les principes établis dans la première partie du chapitre 2. Qui plus est, cette implémentation est extrêmement similaire à celle proposée dans notre exemple d'application pour la simulation de la cinétique biochimique à l'échelle microscopique dans la section 3.1. En effet, le principe de fonctionnement des cellules sanguines au sein de notre vaisseau est très proche de celui des protéines modélisées dans notre précédente application. Chaque cellule, qu'il s'agisse d'un globule ou d'une plaquette, constitue un individu du système. Elle est décrite par une structure de données composée de nombreuses variables décrivant son état (position, taille...) et évolue au cours du temps en faisant appel à une fonction d'activation définissant son comportement au cours d'un cycle de simulation. L'ordonnancement de ces cellules suit le schéma des itérations asynchrones et chaotiques défini dans la section 2.1.2.2.

À l'heure actuelle, le comportement défini par la fonction d'activation de nos cellules est extrêmement basique. Dans un premier temps, la cellule détermine la maille du maillage de résolution des équations de *Navier-Stokes* dans laquelle se situe son centre. Suite à cela, elle perçoit la vitesse du fluide au sein de cette maille. Cette étape de perception correspond à une lecture dans les trois tableaux de données « *from* », manipulées par les *kernels* décrivant l'algorithme du SIMPLE, contenant les vitesses du fluide selon les axes $\vec{x}$, $\vec{y}$ et $\vec{z}$. Cette interaction entre le modèle individu-centré décrivant le comportement des cellules et le modèle population-centré décrivant l'écoulement plasmatique s'appuie sur les mécanismes que nous avons proposés précédemment dans la section 2.3.2. Notons que, pendant ce temps, les tableaux de données « *to* » qui seront consultés au prochaine cycle de simulation, sont mis à jour par les *kernels* du SIMPLE. Lorsque la vitesse du fluide au sein de la maille est déterminée, la cellule l'utilise pour calculer la force de traînée qu'elle subit de la part de l'écoulement. Pour cela, elle intègre les équations obtenues en appliquant le principe fondamental de la dynamique et présentées précédemment. Nous choisissons d'utiliser la méthode d'Euler pour résoudre l'équation différentielle obtenue et déterminer le déplacement qu'effectue la cellule au cours d'un cycle de simulation. Ce déplacement nous permet de faire évoluer la position de la cellule au sein du vaisseau.

Dans l'état actuel, ce comportement basique nous permet uniquement de visualiser l'écoulement plasmatique dans notre vaisseau sanguin. La prochaine évolution majeure de ce modèle de cellule consiste à implémenter des mécanismes de détection de collision afin de proposer par la suite une modélisation des mécanismes plaquettaires. Pour cela, nous envisageons d'utiliser un maillage de l'espace similaire à celui implémenté dans l'application de simulation de la cinétique biochimique à l'échelle microscopique, ce qui nous permettra de maximiser les performances de l'étape de détection de collision en ne considérant qu'un voisinage local et non pas l'ensemble des cellules modélisées. D'un point de vue informatique, cet algorithme sera en tout point identique à celui présent dans notre application de cinétique biochimique. Il permettra par la suite l'élaboration d'un algorithme de gestion des collisions qui modélisera les mécanismes plaquettaires. Lorsqu'un amas de plaquettes se formera au niveau de la brèche, chaque cellule « marquera » les mailles dans lesquelles elle se trouve, ce qui permettra de modifier le comportement de l'écoulement et de la coagulation plasmatique. Cette modification utilisera les mécanismes d'interaction présentés dans la section 2.3.2 de ce mémoire. Elle consistera à remplir un tableau de variations qui sera ensuite passé en paramètre des *kernels* définissant les comportements à altérer.

Bilan

L'ensemble des implémentations détaillées dans cette section est fondé sur les méthodes de couplage multi-modèles que nous proposons dans la troisième partie du chapitre 2. Grâce à elles, la simulation de notre vaisseau sanguin virtuel peut bénéficier à la fois de la puissance de calcul offerte par les multiples GPUs de la machine de simulation, mais également de celle offerte par l'ensemble des CPUs composant cette dernière. Dans la section suivante, nous présentons une étude des performances que nous obtenons lors de l'exécution de nos simulations. Afin de ne pas trop alourdir la lecture de ce chapitre, nous ne détaillons pas ici les paramètres et données physiologiques gouvernant notre modèle. Cependant, ces derniers peuvent être trouvés dans l'annexe B.

3.2.3 Évaluation des performances

En raison des plus de 450 000 mailles (cf. annexe B) et du nombre de variables numériques qui composent nos modèles, nous devons résoudre environ 160 millions d'équations pour réaliser un cycle de simulation (de 1 ms) au cours duquel 300 éléments figurés sont simulés. Ce cycle est répété plus d'un million de fois pour simuler une vingtaine de minutes de coagulation du sang. Pour exécuter de telles simulations nous utilisons un unique ordinateur massivement parallèle constitué de nombreuses unités de calcul provenant de :

2 processeurs Intel Xeon X5680 dotés chacun d'une vitesse d'horloge de 3.33 GHz et de 12 Mo de mémoire cache de niveau 3. Grâce à la technologie SMT à deux voies, les douze cœurs physiques peuvent être assimilés à vingt-quatre cœurs logiques pouvant fonctionner simultanément.

3 cartes graphiques NVIDIA GeForce GTX TITAN basées sur l'architecture Kepler de NVIDIA et utilisant une puce GK110. Chacune est munie de 2688 cœurs CUDA

cadencés à 837 MHz, ainsi que de 6 Go de mémoire GDDR5.

1 carte graphique NVIDIA GeForce GTX 690 également basée sur l'architecture Kepler mais composée de deux puces GK104, possédant chacune 1536 cœurs CUDA cadencés à 915 MHz, ainsi que 2 Go de mémoire GDDR5. Elle peut-être assimilée à deux GeForce GTX 680. Nous nommons ces deux puces GTX 690a et GTX 690b.

Ainsi, nous bénéficions d'un total de 24 CPUs et de 5 GPUs pour exécuter nos simulations. Afin d'évaluer les performances de ces dernières, et donc des solutions techniques proposées dans le chapitre 2 de ce mémoire, nous proposons de les exécuter avec des configurations matérielles variées. Ces simulations utilisent un équilibrage de charge statique réalisé lors de leur initialisation. Cela signifie que les sous-maillages attribués à chacun des composants de la configuration matérielle (aussi bien GPU que CPU) participant à la simulation sont de taille fixe et ne peuvent pas varier au cours de la simulation. Les tailles de ces différents sous-maillages sont choisies de manière à ce que leurs temps de traitement soient sensiblement identiques. Cela nous permet de minimiser les attentes inutiles lors des synchronisations entre les composants survenant à la fin de chaque *kernel*. Par ailleurs, nous notons que ces simulations sont dans un premier temps réalisées sans éléments figurés, c'est-à-dire uniquement avec les phénomènes d'écoulement et de coagulation plasmatique. L'impact des éléments figurés sur les performances de nos simulations sera évalué à la fin de cette section.

Le fonctionnement de ces simulations suit la démarche présentée sur la figure 2.15 page 74, mais avec de multiples *kernels* successifs et sans modèle individu-centré. Malgré l'absence des éléments figurés lors de ces simulations, les tableaux de données représentant la vitesse de l'écoulement sont tout de même transférés à chaque cycle de simulation depuis la mémoire globale des GPUs vers la mémoire principale de la machine hôte. Cette étape de réception des données permettra ultérieurement aux éléments figurés de consulter la vitesse de l'écoulement définissant la force de traînée qu'ils subissent. Le tableau 3.3 regroupe l'ensemble des mesures effectuées. Il met en évidence la fréquence de calcul de nos simulations, c'est-à-dire le nombre de cycles par seconde, ainsi que le temps passé à transférer les frontières entre sous-maillages, à réceptionner des données depuis les GPUs vers la machine hôte et enfin à calculer. Le temps restant correspond aux opérations annexes, autres que le calcul et le transfert de données, telles que le lancement des *kernels* ou encore les attentes liées aux synchronisations survenant entre eux. Notons que ces temps sont exprimés sous la forme de pourcentage de la durée d'un cycle de simulation.

Nous observons que les meilleures performances sont obtenues avec une GeForce GTX TITAN. En outre, nous notons que plus un GPU possède de cœurs, plus il semble performant. En effet, les unités de calcul supplémentaires qu'offre une GeForce GTX TITAN (2688 cœurs) face à l'une des deux puces de la GeForce GTX 690 (1536 cœurs) lui permettent d'être nettement plus performante. Le gain de performance que nous observons entre ces deux cartes est d'environ 1.5, ce qui correspond approximativement au ratio entre la puissance théorique approximative[8] des deux GPUs, c'est-à-dire $\frac{2688\times827}{1536\times915} \simeq 1.6$. Par ailleurs, nous remarquons que les transferts de frontières sont prépondérants sur le temps de calcul lorsque plusieurs GPUs sont utilisés. De plus, les transferts de données multidirectionnels et concurrents sur le

8. Nous définissons la puissance théorique approximative comme le produit entre le nombre d'unités de calcul et leur fréquence d'horloge. Cette valeur nous donne une estimation de la puissance théorique d'un GPU ou d'un CPU.

Configuration	Fréquence	Frontières	Réceptions	Calculs	Reste
1 CPU	0.3 Hz	-	-	-	-
24 CPUs	1.3 Hz	-	-	-	-
GTX 690a	32 Hz	-	0.6 %	95 %	4.4 %
GTX 690a + 690b	34 Hz	27 %	0.7 %	52 %	20.3 %
1 GTX TITAN	46 Hz	-	0.7 %	94 %	5.3 %
1 GTX TITAN + 24 CPUs	25 Hz	18 %	0.3 %	50 %	31.7 %
2 GTX TITAN	42 Hz	32 %	0.7 %	45 %	22.3 %
3 GTX TITAN	25 Hz	30-60 %	0.5 %	20 %	19.5-49.5 %
5 GPUs	15 Hz	35-60 %	0.3 %	9 %	30.7-55.7 %
5 GPUs + 24 CPUs	12 Hz	40-80 %	0.2 %	7 %	12.8-52.8 %

Tableau 3.3 – **Évaluation des performances** – Ce tableau indique la fréquence de calcul (nombre de cycles de simulation par seconde) en fonction de la configuration matérielle utilisée. Il met également en évidence le pourcentage de temps (par rapport à la durée d'un cycle) que passent les GPUs à transférer leurs frontières, réceptionner des données et calculer. Notons que ces résultats proviennent de simulations sans éléments figurés.

bus PCI-express reliant les GPUs semblent soumis à de fortes variations de performance. Cela confirme ce que nous avons déjà répété à de nombreuses reprises : les opérations de transferts de données sont particulièrement néfastes dans ce type de simulation. De plus, nous notons qu'un fort pourcentage de la durée des cycles de simulation ne correspond ni aux transferts de données, ni aux calculs. Nous émettons l'hypothèse que ce pourcentage restant provient des synchronisations entre les différents composants de la configuration matérielle choisie. Nous vérifions cette hypothèse dans la suite de cette section. En raison du temps négligeable qu'occupe l'étape de réception des données entre les GPUs et la machine hôte, nous l'incluons dorénavant dans ce pourcentage restant afin de faciliter la lecture de nos tableaux.

Contrairement aux idées reçues, l'utilisation de plusieurs GPUs ne permet pas d'augmenter les performances de ce type de simulation. En effet, les unités de calcul supplémentaires que proposent un deuxième ou troisième GPU ne compensent pas la chute de performance entraînée par les transferts de données nécessaires entre eux. Par ailleurs, nous notons que, quelle que soit la configuration, les CPUs ne permettent pas d'augmenter les performances, bien que le sous-maillage qui leur est attribué soit nettement plus petit que celui des GPUs : seulement 1 % de la charge de travail est confié aux CPUs. Nous justifions cela en indiquant que la puissance théorique approximative d'une GeForce GTX TITAN est environ trente fois supérieure à celle des CPUs utilisés ($\frac{2688\times827}{3330\times24} \simeq 30$), ce qui la rend beaucoup plus efficace pour ce genre de traitement. En outre, cette observation tend à mettre en évidence la supériorité de l'architecture SIMT des GPUs pour simuler des modèles population-centrés utilisant un maillage de l'espace.

Afin de mieux mettre en évidence l'impact des transferts de données entre les GPUs, nous proposons de nouvelles mesures, sensiblement identiques aux précédentes, mais où

ces transferts n'ont plus lieu. Cela a pour conséquence de biaiser les résultats de nos simulations, mais nous souhaitons ici uniquement étudier les performances obtenues dans de telles situations. Le tableau 3.4 recense les fréquences de calcul obtenues lorsqu'aucun transfert de données n'a lieu, le pourcentage de temps passé à réaliser les calculs et le temps restant.

Configuration	**Fréquence**	**Calculs**	**Reste**
1 CPU	0.3 Hz	-	-
24 CPUs	1.3 Hz	-	-
GTX 690a	32 Hz	95 %	5 %
GTX 690a + 690b	55 Hz	85 %	15 %
1 GTX TITAN	46 Hz	94 %	6 %
1 GTX TITAN + 24 CPUs	46 Hz	92 %	8 %
2 GTX TITAN	76 Hz	78 %	22 %
3 GTX TITAN	87 Hz	69 %	31 %
5 GPUs	82 Hz	50 %	50 %
5 GPUs + 24 CPUs	76 Hz	45 %	55 %

Tableau 3.4 – **Impact des transferts de frontière sur les performances** – Ce tableau indique la fréquence de calcul (nombre de cycles de simulation par seconde) en fonction de la configuration matérielle utilisée et sans transferts de frontières. Il met en évidence le pourcentage de temps que les GPUs passent à calculer au cours d'un cycle de simulation.

Les résultats obtenus confirment l'impact des transferts de données sur les performances de nos simulations. En effet, les fréquences de calcul obtenues dans cette situation sont largement supérieures à celles obtenues lorsque les transferts de frontières sont présents. Malheureusement, pour que nos simulations fournissent des résultats pertinents et significatifs, nous devons impérativement transférer les frontières entre les sous-maillages à chaque cycle de simulation. Toutefois, nous constatons que cette mise à jour des frontières n'est pas le seul facteur limitant la performance puisque, même sans elle, l'augmentation du nombre de GPUs impliqués est contre-productif dans certains cas. Par ailleurs, nous remarquons qu'un pourcentage non négligeable du temps total n'est toujours pas utilisé pour calculer.

Ce second mécanisme limitant les performances de nos simulations est dû aux barrières de synchronisation. En effet, nous avons indiqué, lors de l'implémentation de nos modèles dans la section 3.2.2.5, que l'exécution de chacun des *kernels* développés est ponctuée par une barrière de synchronisation où les *threads* des GPUs et des CPUs s'attendent mutuellement avant de poursuivre avec l'exécution des autres *kernels*. Cette étape de synchronisation permet de s'assurer que les traitements précédents sont terminés, et donc que les futurs calculs utiliseront des données cohérentes. Néanmoins, ce mécanisme a un coût en performance non négligeable comme le montre le tableau 3.5 qui regroupe nos mesures de performance lorsque ce mécanisme est désactivé[9]. Bien que la désactivation de ce mécanisme empêche la production de résultats cohérents, elle permet de mettre en évidence son impact sur les performances de nos simulations. En comparant les mesures de ce tableau avec celles contenues dans le tableau 3.4, nous observons une nette augmentation des performances lorsque plusieurs GPUs sont utilisés. Nous attribuons le faible pourcentage de temps restant

9. Nous conservons cependant une unique barrière de synchronisation à la fin de chaque cycle, ce qui permet de les délimiter.

aux différentes opérations annexes que les CPUs et GPUs peuvent réaliser mais que nous ne pouvons guère maîtriser (appels asynchrones de fonctions CUDA ...).

Configuration	Fréquence	Calculs	Reste
1 CPU	0.3 Hz	-	-
24 CPUs	1.5 Hz	-	-
GTX 690a	34 Hz	99.5 %	0.5 %
GTX 690a + 690b	66 Hz	99.5 %	0.5 %
1 GTX TITAN	51 Hz	99.5 %	0.5 %
1 GTX TITAN + 24 CPUs	51 Hz	99.5 %	0.5 %
2 GTX TITAN	98 Hz	99.5 %	0.5 %
3 GTX TITAN	131 Hz	97 %	3 %
5 GPUs	170 Hz	98 %	2 %
5 GPUs + 24 CPUs	143 Hz	97 %	3 %

Tableau 3.5 – **Impact des barrières de synchronisation sur les performances** – À l'instar du tableau 3.4, ce tableau indique la fréquence de calcul (nombre de cycles de simulation par seconde) en fonction de la configuration matérielle utilisée. Cependant, celui-ci indique les performances sans transferts de données, mais également sans barrières de synchronisation.

En dépit du fait que la charge de travail attribuée aux CPUs soit de l'ordre de 1 %, nous notons que, quelles que soient les conditions dans lesquelles les mesures sont réalisées, lorsque les CPUs sont couplés à des GPUs, ils ne parviennent pas à accélérer les simulations. Ils font parfois même chuter leur performance. En effet, lorsque, comme ici, le problème traité est parfaitement conditionné et adapté au mode de fonctionnement d'un GPU, les quelques CPUs d'une machine ne peuvent rivaliser avec les milliers de cœurs d'un GPU. Cette observation nous incite à ne pas les utiliser pour simuler les modèles population-centrés décrivant l'écoulement et la coagulation plasmatique, mais à les mettre uniquement au service de la simulation des éléments figurés. De plus, nous choisissons de n'utiliser qu'une seule carte graphique GeForce GTX TITAN pour simuler l'écoulement et la coagulation au sein de notre vaisseau, puisque lorsque nous utilisons plusieurs GPUs, la nécessité de maintenir la cohérence des calculs nous impose des transferts de données et des synchronisations qui induisent une chute dramatique de performance. La figure 3.13 indique les fréquences de calcul mesurées en fonction du nombre d'éléments figurés modélisés. Ces mesures sont réalisées dans des conditions normales de simulation.

Cette figure met en évidence la capacité des CPUs à participer à nos simulations en exécutant le comportement des éléments figurés parallèlement à l'écoulement et la coagulation plasmatique. Bien que lorsque le nombre de cellules simulées est très important la fréquence de calcul diminue, nous remarquons la présence d'un palier entre 0 et 500 000 cellules. Il traduit la capacité des CPUs à simuler les éléments figurés parallèlement aux traitements effectués sur GPUs sans pour autant affecter les performances des simulations. Ainsi, tant que nous simulons moins de 500 000 cellules, les temps de calcul des CPUs sont masqués par ceux des GPUs et n'ont donc aucun impact sur les performances. Une fois ce seuil dépassé, les CPUs commencent à ralentir les simulations : leurs traitements deviennent prépondérants face à ceux des GPUs. De ce fait, nous pouvons très largement simuler les 300 cellules que requièrent nos simulations (cf. section B.2).

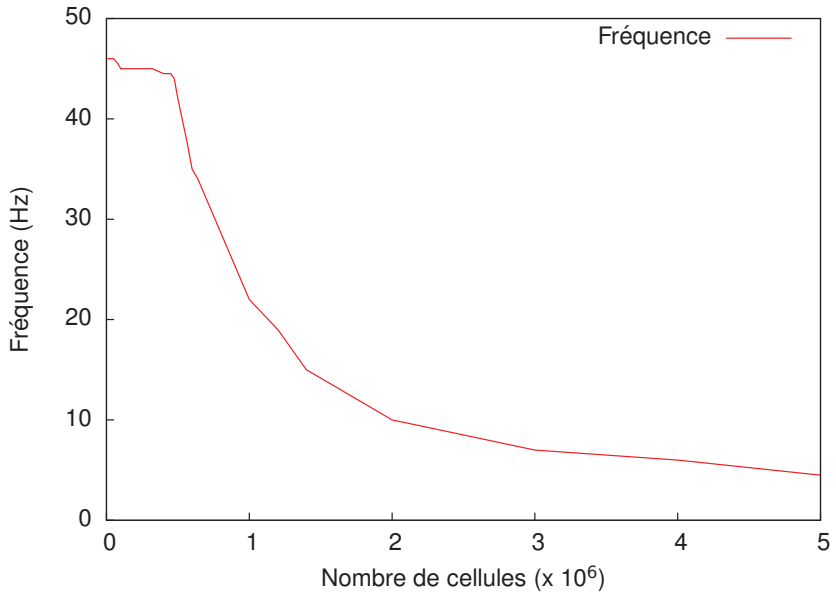


Figure 3.13 – **Impact du nombre de cellules sur les performances** – Cette courbe représente l'évolution de la fréquence de calcul (nombre de cycles de simulation par seconde) en fonction de la quantité d'éléments figurés simulés. Pour rappel, lors de ces mesures, les modèles population-centrés sont simulés sur une GeForce GTX TITAN et le modèle individu-centré sur les 24 CPUs.

Bilan

Dans cette section, nous avons réalisé différentes mesures pour évaluer les performances de nos simulations. Ainsi, nous avons observé que l'architecture SIMT des GPUs est très adaptée à la simulation de modèles population-centrés et que plus un GPU est muni de cœurs, plus il semble performant. Néanmoins, lorsque ces unités de calcul appartiennent à différents GPUs, cette observation n'est plus valable. En effet, notre étude nous a permis de mettre en évidence l'impact négatif des opérations de transfert de données et de synchronisation entre les unités de calcul sur des configurations multi-GPUs. Nos mesures montrent également que, quelle que soit la configuration utilisée, les CPUs ne sont pas capables d'augmenter les performances s'ils participent à la simulation de nos modèles population-centrés. À l'inverse, ils se prêtent très bien à la simulation de notre modèle individu-centré. Ces constats nous ont poussés à sélectionner une configuration matérielle avec une unique GeForce GTX TITAN pour la simulation des modèles population-centrés et les vingt-quatre CPUs pour la simulation de notre modèle individu-centré. Cette configuration est illustrée sur la figure 3.14. Dans de telles conditions, nos simulations nécessitent environ huit heures pour s'exécuter, contrairement à plusieurs jours sur une architecture mono-CPU. La section suivante propose une solution permettant de visualiser et de traiter les résultats de telles simulations. Elle présente également des idées d'application.

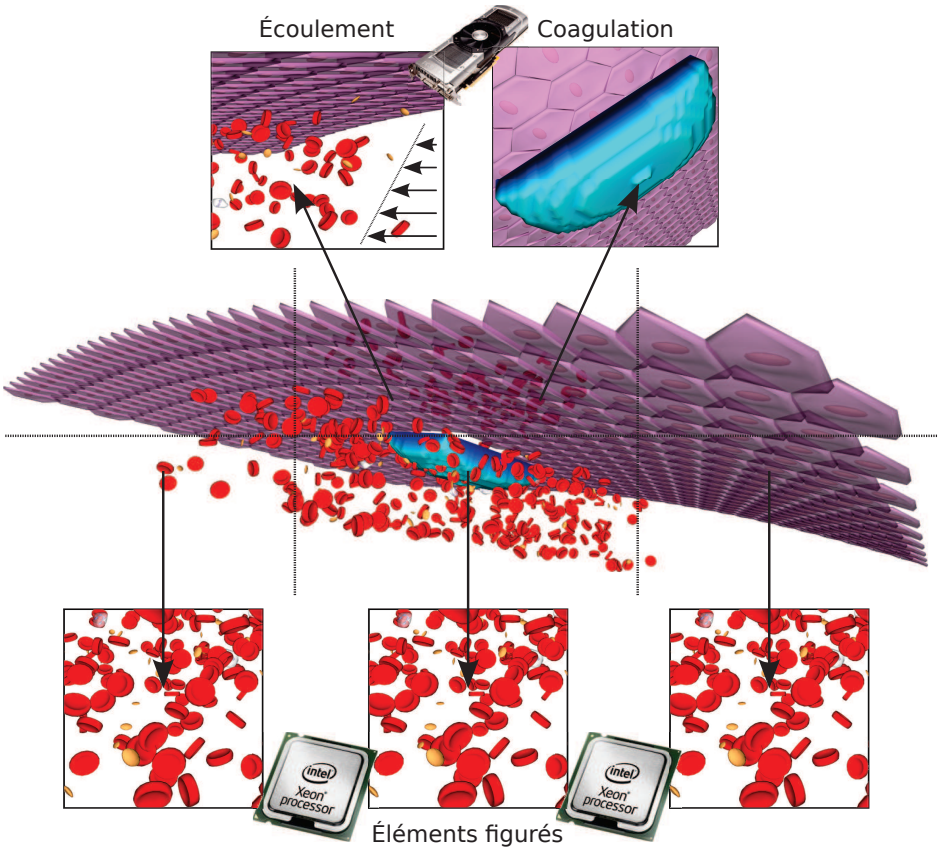


Figure 3.14 – **Parallélisation du vaisseau sanguin virtuel** – Cette figure présente un bilan des différentes architectures utilisées pour paralléliser nos modèles. L'écoulement et la coagulation plasmatique sont calculés sur une GeForce GTX TITAN alors que les éléments figurés sont traités par les CPUs..

3.2.4 Potentiel applicatif

Comme nous l'avons indiqué à de nombreuses reprises, notre modélisation est incomplète. En effet, en raison de la complexité des phénomènes de coagulation du sang, nous avons choisi de ne modéliser pour l'instant que la coagulation plasmatique, et donc de ne pas considérer la vasoconstriction, l'hémostase primaire et la fibrinolyse. De ce fait, il nous est tout à fait impossible de valider biologiquement notre modèle. En outre, en supposant que notre modélisation soit complète, nous ne pourrions simplement comparer nos résultats qu'avec ceux provenant de mesures *in vitro*, et donc sans la contribution de l'écoulement sanguin et de la paroi endothéliale, étant donné qu'il n'existe, à notre connaissance, aucun moyen de mesure de caillot *in vivo*. Néanmoins, malgré cette absence de validation biologique, nous proposons d'étudier le potentiel applicatif de nos simulations dans les domaines distincts de la sensibilisation de patients et du développement pharmaceutique.

Bien que nos simulations soient massivement parallélisées, le temps de calcul d'une simulation complète, c'est-à-dire une vingtaine de minutes de coagulation plasmatique, est élevé (de l'ordre de huit heures). De ce fait, il n'est pas envisageable de visualiser de

telles simulations durant leur exécution. Afin de pallier ce problème, nous présentons une application de visualisation permettant de rejouer nos simulations tout en proposant des moyens d'interaction.

3.2.4.1 Description de l'application

Mécanismes de visualisation

L'application que nous décrivons dans cette section nous permet de visualiser nos simulations une fois ces dernières calculées. L'un des aspects importants de cette application est de conserver le caractère interactif des simulations. En effet, il ne s'agit aucunement d'un lecteur vidéo classique avec lequel l'utilisateur ne peut interagir que temporellement, en se déplaçant dans la vidéo. Au contraire, dans notre application de visualisation, l'utilisateur est libre de se déplacer dans la simulation, aussi bien dans le temps que dans l'espace, et également d'interagir avec les éléments qui la composent, principalement en choisissant de les masquer ou de les afficher pour plus de lisibilité.

Le fonctionnement de ce visualisateur est relativement simple. Il consiste à lire le contenu de fichiers de résultats produits lors de l'exécution de nos simulations. Ces fichiers contiennent l'ensemble des valeurs nécessaires au rejeu de nos simulations, et notamment les concentrations en fibrine (la protéine « ciment » responsable de la formation du caillot) et en thrombine, la vitesse et la pression du plasma dans chacune des mailles du maillage, ainsi que les positions et orientations de toutes les cellules modélisées. Afin de ne pas faire chuter dramatiquement les performances de nos simulations, ces fichiers ne sont pas actualisés à chaque cycle de simulation. En effet, l'écriture de tant de données dans un fichier a un coût non négligeable. Ainsi, les fichiers nécessaires au rendu visuel en trois dimensions sont écrits à une fréquence de 25 Hz soit toutes les 40 ms, alors que les fichiers nécessaires à l'étude des résultats (notamment pour la création de courbes) ne sont actualisés que toutes les deux secondes. Malgré ces fréquences d'écriture très peu élevées par rapport à la fréquence de calcul de nos simulations, la taille des fichiers de résultats ainsi obtenus peut atteindre plusieurs centaines de gigaoctets. Cela est notamment dû au fait que notre maillage est composé d'environ 450 000 mailles dont l'état doit être sauvegardé régulièrement dans ces fichiers. Pour remédier à ce problème, nous avons choisi de compresser les résultats produits par nos simulations. Pour cela, nous faisons appel à l'algorithme *DEFLATE* [Salomon et Motta, 2010], implémenté notamment par la bibliothèque informatique spécialisée *zlib*, permettant de produire des fichiers compressés au format *gzip*. Ce mécanisme de compression nous permet d'obtenir des fichiers de résultats d'environ deux ou trois gigaoctets pour une simulation d'une vingtaine de minutes. Nous avons mis en place un mécanisme d'index qui permet une navigation temporelle efficace dans le flux de données compressées lors de l'utilisation de notre visualisateur interactif.

Outil d'étude de la coagulation

Nos travaux de simulation d'un vaisseau sanguin virtuel ont abouti au développement d'un prototype de logiciel informatique permettant d'étudier la formation d'un caillot chez une personne saine ou malade, mais également d'observer l'impact d'un médicament sur le processus de coagulation du sang. Cet outil est composé d'une interface homme-machine (IHM) permettant de piloter la simulation et d'observer certains paramètres caractérisant la coagulation, ainsi que d'une fenêtre de rendu stéréoscopique représentant notamment la formation du caillot en trois dimensions. Comme nous l'avons mentionné précédemment, nos simulations demandent environ huit heures de calcul. Ainsi, notre outil ne peut pas se permettre d'interagir directement avec elles et se contente de les rejouer à partir des fichiers de résultats qu'elles produisent.

Le développement de cet outil trouve son origine dans la balance hémostatique illustrée sur la figure 3.6. De ce fait, il propose de simuler la coagulation de trois profils de patients : un patient sain (son bilan sanguin est normal), un patient hémophile B (il possède cent fois moins de facteur IX qu'un patient sain) et un patient thrombotique (il possède moitié moins d'antithrombine que le patient sain). L'idée générale de ce logiciel est de pouvoir comparer le processus de coagulation du sang chez ces trois profils et en plus d'étudier l'impact d'un traitement sur les patients malades. Lorsque notre logiciel démarre, il fait donc apparaître une fenêtre permettant de sélectionner le ou les patients dont l'utilisateur souhaite étudier la formation du caillot (illustrée sur la figure 3.15 page 113). Cette fenêtre présente le bilan sanguin des trois profils de patients que nous considérons, c'est-à-dire l'ensemble des concentrations en facteurs et inhibiteurs de la coagulation requis par nos simulations pour fonctionner. Le bilan sanguin des patients malades est complété par une liste de médicaments permettant de les traiter et d'étudier l'impact d'un tel traitement. À l'heure actuelle, seuls le *NovoSeven*® et le *Pradaxa*® sont modélisés. Par ailleurs, si les simulations ont été préalablement calculées, il est également possible de modifier le bilan sanguin d'un patient afin d'étudier l'impact de cette modification sur le processus de coagulation du sang.

Lorsque l'utilisateur a sélectionné les profils de patient qu'il souhaite comparer, une nouvelle IHM apparaît, accompagnée d'une fenêtre de rendu en trois dimensions permettant de visualiser le caillot sanguin. La nouvelle fenêtre d'interface est composée de deux parties. La première permet de piloter le rejeu de la simulation, c'est-à-dire de lire ou de mettre en pause la simulation ainsi que de se déplacer dans le temps. Elle reprend les contrôles d'un lecteur vidéo classique. La seconde partie de la fenêtre, divisée en deux onglets, propose de comparer divers paramètres caractérisant la coagulation du sang. Les captures d'écrans présentées sur les figures 3.16 et 3.17 pages 114 et 115 illustrent les résultats des simulations d'un patient sain, d'un patient hémophile B et de ce même patient traité avec une dose de *NovoSeven*®. Elles mettent en évidence les différents paramètres de la coagulation du sang que notre outil permet de comparer. Nous avons choisi d'afficher les paramètres suivants pour étudier la coagulation du sang chez nos patients : la génération de thrombine, la cinétique du volume du caillot et la quantité de pertes sanguines durant le saignement.

La thrombine est l'enzyme-clé de la coagulation : en effet, elle est la seule responsable de la transformation du fibrinogène soluble en réseau de fibrine insoluble ; par ailleurs, elle potentialise son action procoagulante en rétroactivant les facteurs V, VIII et XI ainsi que les

plaquettes ; d'autre part, elle participe à son inhibition en activant le système protéine C. Son rôle majeur dans la coagulation a conduit au concept de mesure et suivi de son activité *in vitro* afin d'en faire un test « global » de coagulation. Le suivi de cette activité est motivé par le postulat que « plus il y a de thrombine, plus le potentiel à coaguler du patient est fort et inversement ». Ainsi, la mesure de cette quantité de thrombine doit permettre de dresser un bilan hémostatique de l'échantillon sanguin et donner des informations pour l'établissement d'un diagnostic, c'est-à-dire un profil hémorragique, sain ou thrombotique. La courbe obtenue est appelée thrombinogramme et donne l'évolution de la quantité de thrombine générée au cours du temps. Plusieurs paramètres peuvent être extraits de ce cette courbe, classiquement :

l'*endogenous thrombin potential* (ETP), ou potentiel de thrombine endogène, qui correspond à l'aire sous la courbe de génération de thrombine ;

le *lag time*, ou temps de latence, qui correspond au temps que nécessite la génération de thrombine pour débuter : il s'agit classiquement du temps qu'il faut pour obtenir 10 nM de thrombine ;

le *peak*, ou pic, qui correspond à la concentration maximale de thrombine atteinte ;

le *time to peak*, ou temps au pic ou de montée, qui correspond au temps mis par la thrombine pour atteindre son pic.

Les premiers travaux sur la génération de thrombine ont débuté notamment avec [Macfarlane et Biggs, 1953], mais les principes, techniques de mesure ainsi que le concept ont été largement améliorés et popularisés par les recherches des professeurs H.C. Hemker et S. Béguin (voir par exemple [Hemker et Béguin, 1995; Hemker *et al.*, 2003]). Aussi, nous avons inclus le thrombinogramme et les paramètres qui lui sont associés (ETP, *lag time*, *peak*, *time to peak*) dans notre IHM afin d'étudier l'évolution de la coagulation au cours de nos simulations comme illustré sur la figure 3.16 page 114 [10].

Les tests standards de routine utilisés de nos jours pour le diagnostic *in vitro* de la coagulation plasmatique consistent à mesurer le temps nécessaire à un échantillon plasmatique pour produire un caillot de fibrine. Le premier, le temps de Quick (TQ) ou taux de prothrombine (TP) introduit par [Quick *et al.*, 1935], explore la voie extrinsèque tandis que le second, le temps de céphaline avec activateurs (TCA), explore la voie intrinsèque de la coagulation. Une éventuelle anomalie de la coagulation peut se traduire par un allongement du temps d'apparition du caillot dans un ou dans les deux tests. Ajoutons brièvement qu'ils permettent aussi le suivi de certains anticoagulants (anti-vitamine K pour le TP, héparines non fractionnées pour le TCA). L'utilisation quotidienne de ces tests en laboratoire soulève l'intérêt de la mesure du caillot pour l'étude de la coagulation, même s'ils présentent leurs limites (ils sont principalement sensibles aux maladies hémorragiques et très peu aux maladies thrombotiques). Mentionnons par ailleurs la thromboélastographie et la thromboélastométrie [Johansson *et al.*, 2009] qui étudient les propriétés mécaniques du caillot en sang total ou encore la « *clot waveform analysis* » [Shima *et al.*, 2013] qui mesure les propriétés optiques du caillot au cours du temps afin d'en déduire un diagnostic. Au final tous ces exemples indiquent que la mesure du caillot est un paramètre important, c'est pourquoi nous affichons

10. La courbe de génération de thrombine que nous affichons est issue de la thrombine observée au niveau de la paroi vasculaire.

la cinétique du volume du caillot au cours du temps dans notre IHM ; celle-ci est exprimée en pourcentage par rapport à celle du patient sain et est illustrée sur la figure 3.17 page 115.

Nous rappelons que le but du processus de coagulation du sang est de former un caillot qui va arrêter l'hémorragie afin de limiter les pertes sanguines qui sont nocives pour l'organisme : il paraît ainsi évident que l'affichage de la quantité de sang perdu au cours du phénomène est une donnée extrêmement intéressante, tout particulièrement dans l'étude des pathologies hémorragiques. Elle est incluse dans notre IHM comme l'illustre la figure 3.17 page 115.

La fenêtre de rendu (illustrée par la figure 3.18 page 116) permet de visualiser l'écoulement sanguin (par l'intermédiaire des éléments figurés) et la formation du caillot de fibrine au niveau de la brèche vasculaire. Nous insistons une nouvelle fois sur le fait que cette fenêtre est interactive, il ne s'agit pas d'un lecteur vidéo et l'utilisateur peut se déplacer, aussi bien temporellement que spatialement dans la simulation. Il peut de plus interagir avec les simulations en choisissant de masquer ou d'afficher certains éléments afin d'améliorer la lisibilité globale de la simulation. Cette fenêtre bénéficie en outre d'un rendu stéréoscopique permettant de mieux appréhender le relief, et ainsi d'augmenter l'immersion de l'utilisateur.

3.2.4.2 Application à la sensibilisation de patients à leur traitement

Selon les médecins Jeffrey Kline et Daren Beam de l'université d'Indiana [11], notre application de vaisseau virtuel aurait un fort intérêt pour sensibiliser des patients (américains) présentant des troubles de l'hémostase au bon respect de la posologie de leur traitement en cours. En effet, il semblerait que de nombreux patients arrêtent brutalement leur traitement les jours où ils se sentent mieux et à l'inverse, augmentent la posologie les jours où ils se sentent moins bien. Cela aurait pour conséquence de provoquer des accidents hémorragiques ou thrombotiques qui ne seraient certainement pas arrivés si la prescription avait été suivie scrupuleusement. Aussi, afin d'expliquer au patient comment fonctionne l'hémostase mais surtout quels sont les risques encourus en cas de non respect du traitement, l'utilisation de notre simulateur et particulièrement du rendu stéréoscopique serait une très bonne approche pour sensibiliser le patient au bon usage de son traitement [12]. En outre, nous pourrions envisager d'utiliser notre application de simulation de vaisseau virtuel comme outil pédagogique de formation à l'hémostase.

3.2.4.3 Application au développement de molécules dans l'industrie pharmaceutique

Le développement d'un nouveau médicament, de sa conception à sa mise sur le marché, se déroule classiquement en cinq phases :

La phase préclinique vise à tester différentes molécules en utilisant essentiellement des modèles *in vitro* et animaux.

11. Communication personnelle, ISTH 2013, Amsterdam
12. Nous illustrerions alors une valeur ajoutée de la Réalité Virtuelle, concept fédérateur au sein de notre laboratoire

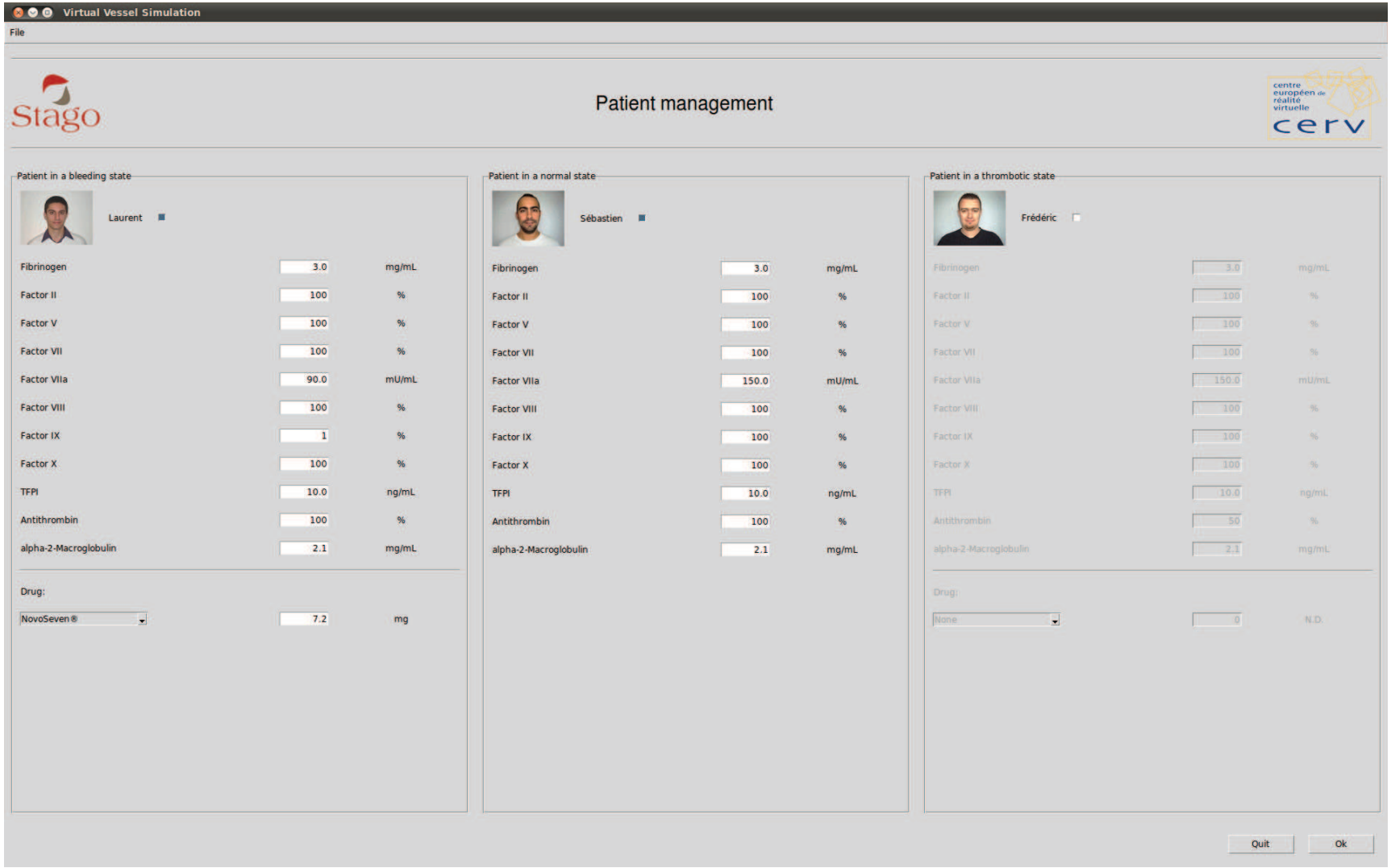


Figure 3.15 – **Fenêtre de sélection des patients** – Cette capture d'écran représente la fenêtre de sélection des patients que propose notre IHM. À titre démonstratif, elle présente les trois profils classiques de patient, à savoir un patient sain (au centre), un patient hémophile (à gauche) et un patient thrombotique (à droite). Notons que ces deux derniers peuvent être traités, c'est-à-dire recevoir une médicament.

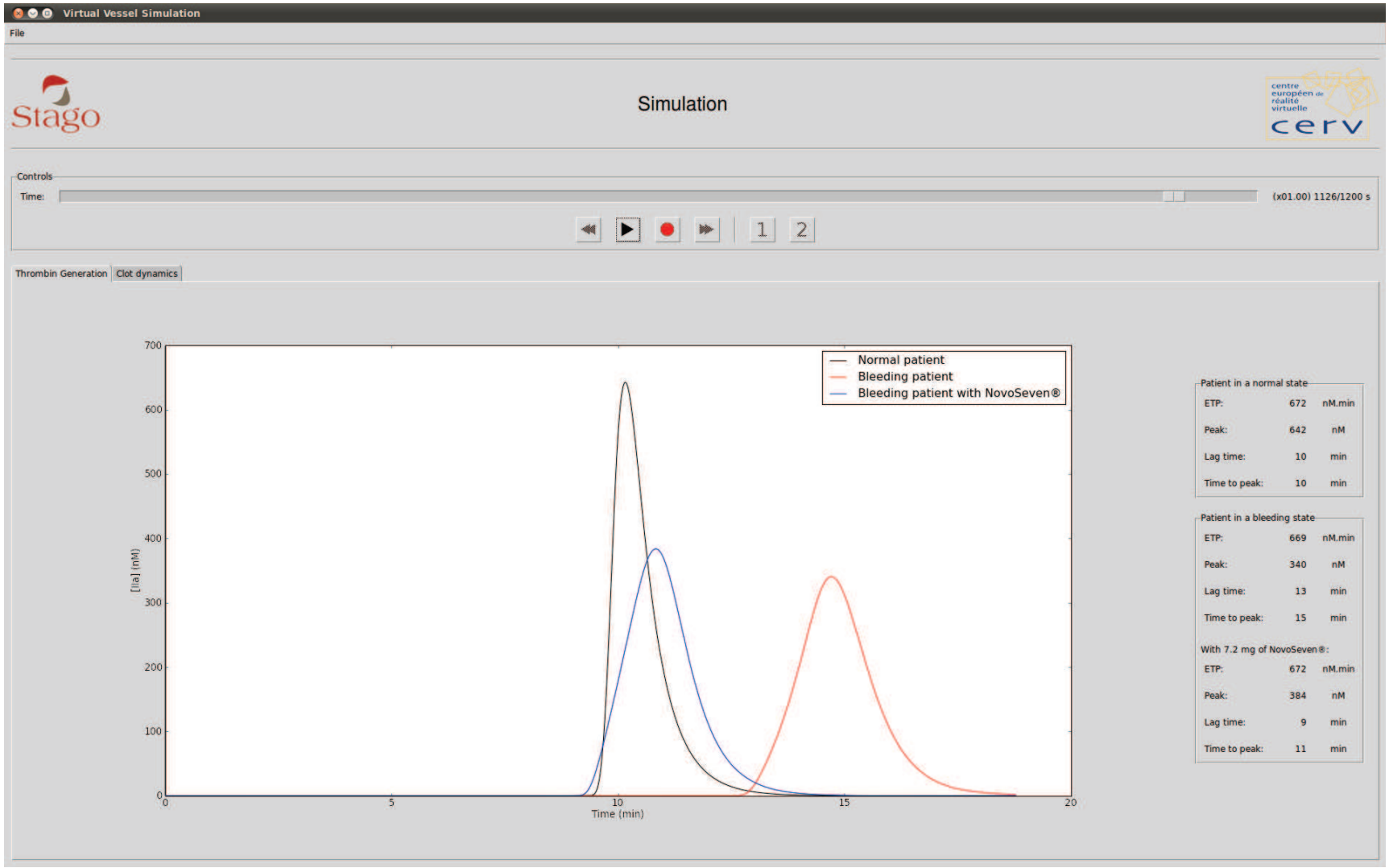


Figure 3.16 – **Fenêtre de contrôle (onglet génération de thrombine)** – Cette capture d'écran représente la fenêtre de contrôle de nos simulations. La partie supérieure présente les contrôles temporels, c'est-à-dire lecture, pause, avance et retour rapides. La partie inférieure rassemble les informations liées au test de génération de thrombine pour trois patients (sain, malade et malade traité).

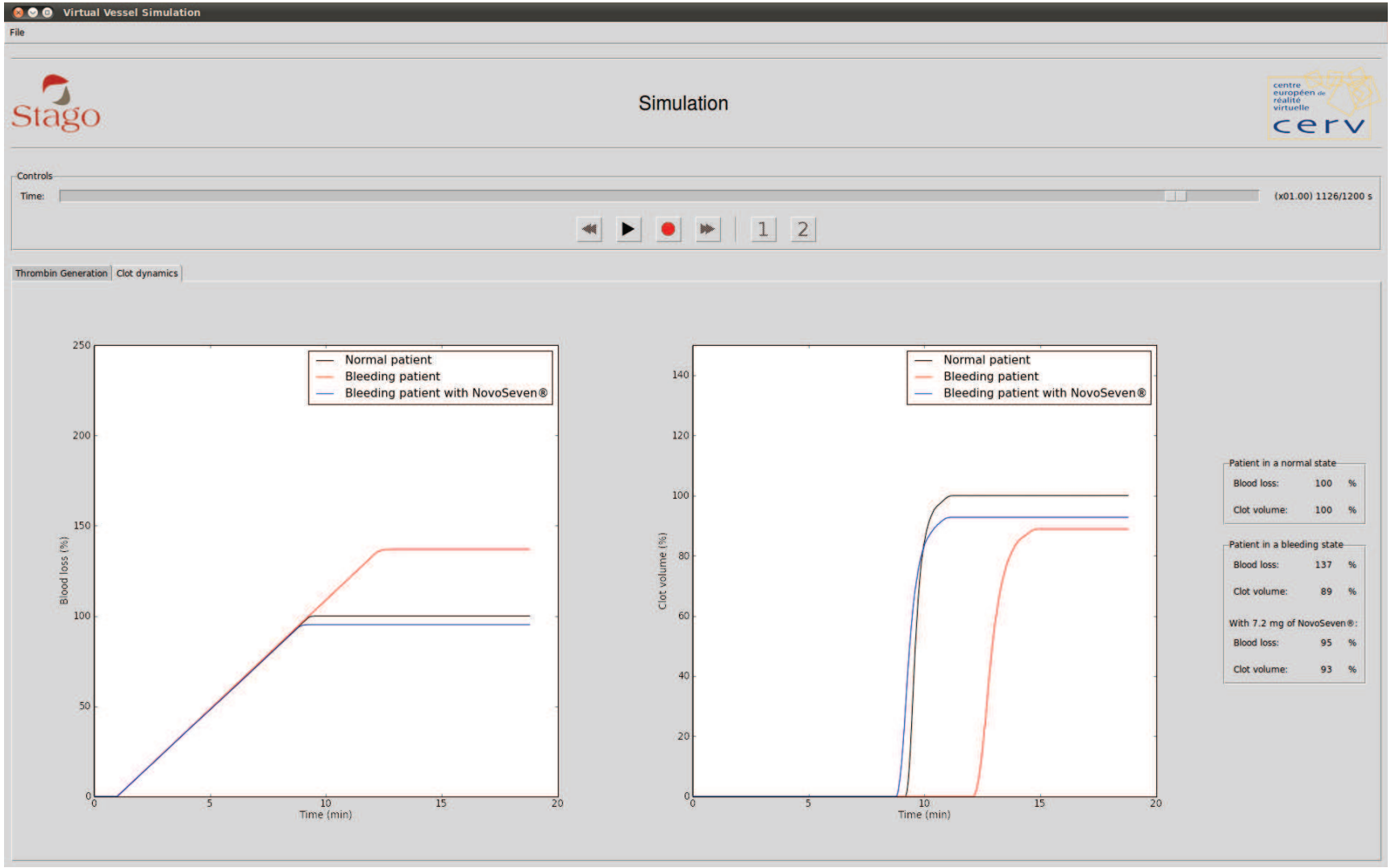


Figure 3.17 – **Fenêtre de contrôle (onglet dynamique du caillot)** – À l'instar de la figure 3.16, cette capture d'écran représente la fenêtre de contrôle de nos simulations. Cependant elle met en évidence la partie de l'application liée à la dynamique de formation du caillot qui propose une estimation des pertes sanguines et du volume du caillot.

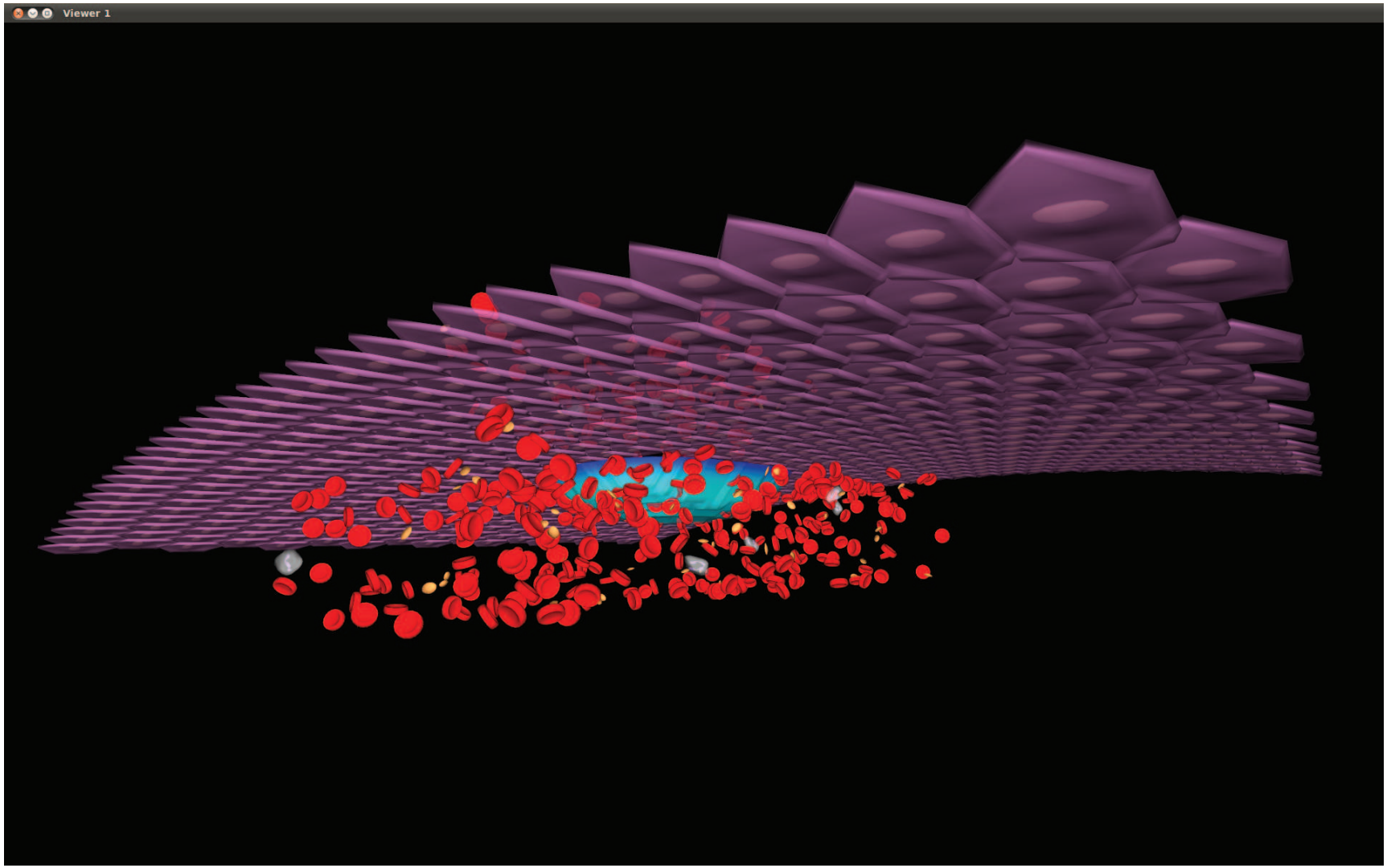


Figure 3.18 – **Fenêtre de rendu des simulations** – Cette capture d'écran représente le rendu en trois dimensions de nos simulations. Elle met en évidence la paroi endothéliale, les éléments figurés suivant l'action du plasma et la formation du caillot sanguin (de couleur bleue). Soulignons qu'un rendu stéréoscopique est possible dans l'application.

La phase I consiste à étudier principalement la sécurité du composé mais aussi des paramètres comme la tolérance, la pharmacocinétique et la pharmacodynamique, la relation dose-effet. Les essais se font sur des volontaires sains.

La phase II cherche à déterminer la dose optimale du médicament ; elle se fait sur une population de malades.

La phase III compare l'efficacité du composé à celle du traitement de référence ou éventuellement à un placebo.

La phase IV est le suivi à long terme d'un traitement alors que celui-ci est autorisé sur le marché. Elle doit permettre de dépister des effets secondaires rares ou des complications tardives.

Nous rappelons que le prototype logiciel présenté dans la section 3.2.4 permet de tester des molécules sur une population (ici restreinte à trois profils) afin d'étudier la coagulation des individus la composant au travers de certains paramètres (génération de thrombine, cinétique du volume du caillot, pertes sanguines). En supposant qu'un jour notre modèle numérique de la coagulation du sang soit suffisamment complet et que le nombre de profils de patients soit beaucoup plus conséquent, nous pensons qu'un tel outil pourrait contribuer au développement clinique d'une molécule et tout particulièrement à la préparation de la phase II. En effet, le but de cette phase est de déterminer la dose optimale du médicament, c'est-à-dire la posologie à administrer à une population de malades maximisant les bénéfices et minimisant les risques associés au traitement ; une mauvaise estimation de cette dose implique la relance de l'étude de phase II et donc un surcoût loin d'être négligeable... Un outil de simulation devrait permettre d'améliorer la prédiction de la dose optimale de la molécule en donnant la possibilité à l'utilisateur de tester virtuellement (et à moindre coût !) par une méthode essai/erreur l'impact de la dose administrée sur les paramètres de coagulation de tous les profils disponibles dans le logiciel. Soulignons qu'une approche similaire a été utilisée avec succès par le laboratoire Bayer dans le développement du rivaroxaban (Xarelto®) [Burghaus *et al.*, 2011].

3.2.4.4 Bilan

Comme indiqué dans l'introduction de cette section, il nous est impossible de valider biologiquement notre modélisation, mais c'est justement lorsqu'il est question d'explorer les domaines dans lesquels l'expérimentation est difficile que la simulation numérique permet de pousser un peu plus loin les investigations. Néanmoins et bien que notre modélisation soit incomplète, nous proposons un prototype d'application permettant d'étudier les troubles de la coagulation *in vivo*. Ce prototype pourrait à terme permettre de sensibiliser des patients à leur traitement, c'est-à-dire leur faire comprendre l'impact de leurs médicaments sur l'organisme, ou de participer au développement des molécules d'un nouveau médicament en proposant des estimations des doses optimales.

3.2.5 Discussion et perspectives

À l'heure actuelle, notre modélisation est loin d'être complète. En effet, un phénomène majeur est absent : il s'agit de l'hémostase primaire et des mécanismes plaquettaires qu'elle met en œuvre. Ils permettent aux plaquettes de constituer, au niveau de la brèche, un amas qui sera consolidé par le caillot de fibrine lors de la coagulation plasmatique, tel le ciment dans un mur de briques. Ce phénomène nous paraît être l'étape la plus importante pour enrichir notre modèle et produire des simulations les plus proches possibles des conditions *in vivo*. Nous avons introduit la manière dont nous envisageons d'implémenter de tel mécanismes plaquettaires (notamment l'étape d'agrégation) à la fin de la section 3.2.2.5 qui présente l'implémentation de nos modèles.

Bien qu'incomplète, la modélisation de vaisseau sanguin virtuel que nous proposons dans cette section nous permet d'étudier les performances offertes par l'approche mixte parallélisée sur CPUs et GPUs que nous détaillons dans le chapitre 2 de ce mémoire. Par l'intermédiaire de cette étude, nous mettons en évidence l'intérêt d'utiliser un GPU possédant de nombreux cœurs, ainsi que les limitations liées à l'utilisation de configurations multi-GPUs. En effet, les communications nécessaires entre les différents GPUs lors de la simulation de modèles population-centrés entraînent des transferts de données entre GPUs, ainsi que des barrières de synchronisation qui impactent négativement les performances. Ainsi, bien que l'ordinateur que nous utilisons soit composé d'une multitude de cartes graphiques, nous nous résignons à n'en utiliser qu'une seule pour éviter ces transferts et synchronisations. En outre, face au constat que la participation des CPUs fait grandement chuter les performances de nos simulations, et ce quelle que soit la configuration matérielle utilisée, nous choisissons de leur confier uniquement la gestion de notre modèle individu-centré décrivant le comportement des éléments figurés. Ces derniers peuvent ainsi être exécutés parallèlement aux traitements se déroulant sur les GPUs et n'impactent pas les performances.

Les limitations liées à l'utilisation d'une multitude de GPUs nous laissent à penser que ces configurations n'ont pas une forte capacité de passage à l'échelle. En effet, étant donné que seule l'utilisation d'un GPU plus puissant, c'est-à-dire doté de plus de cœurs ou d'une fréquence d'horloge plus élevée, nous permet d'augmenter les performances de nos simulations, nous ne pouvons pas produire des simulations de taille plus conséquente ou plus rapides en multipliant simplement les GPUs. Nous retombons sur les limitations de même nature que dans dans le cas du calcul distribué qui ne révèle toute sa puissance que dans le cas de multiples calculs indépendants les uns des autres (ce qui n'est pas le cas ici). Nous pouvons directement opposer ce résultat à celui obtenu lors de l'évaluation des performances de notre modèle individu-centré où la multiplication du nombre de CPUs, tant qu'ils s'accompagnent de mémoire cache, nous permettait d'augmenter linéairement les performances.

Ces travaux de modélisation et de simulation numérique d'un vaisseau sanguin ont fait l'objet de démonstrations dans un salon privé lors de l'ISTH 2013, un congrès international de référence spécialisé dans le domaine de la coagulation du sang. Ces démonstrations ont permis de recevoir les impressions et opinions d'experts mondiaux de l'hémostase au sujet de nos activités de simulation. Ces échanges ont en outre permis de dégager quelques perspectives concernant l'application de nos travaux à la sensibilisation de patients et au domaine pharmaceutique.

3.3 Bilan de nos exemples d'application

Dans ce chapitre, nous avons proposé deux exemples d'application de nos travaux à la coagulation du sang. Le premier propose d'utiliser le modèle individu-centré développé dans la section 2.1 pour simuler le phénomène de cinétique biochimique à l'échelle microscopique, alors que le second propose de simuler un vaisseau sanguin virtuel dans des conditions de flux les plus proches possibles de celles présentes *in vivo*, et ceci en utilisant l'approche mixte développée dans la section 2.3. Ces deux exemples constituent chacun un support pour l'évaluation des performances des solutions techniques de parallélisation que nous proposons.

Par l'intermédiaire de notre première application nous présentons et validons une série d'algorithmes pour la simulation de la cinétique biochimique à l'échelle microscopique, et notamment pour la simulation de réactions bimoléculaires. De surcroît, nous proposons surtout une étude des performances de notre modèle individu-centré dont l'exécution parallélisée exploite avantageusement la hiérarchie mémoire proposée par les architectures multicœur et multiprocesseur. Cette étude montre que, tant que les CPUs s'accompagnent de mémoires caches, les performances de nos simulations augmentent linéairement en fonction du nombre de CPUs utilisés. Cette observation nous laisse penser que notre modèle individu-centré est capable de réaliser des simulations de taille plus conséquente, ou plus rapidement, si la machine utilisée était équipée d'un nombre de CPUs plus élevé.

La seconde application, plus complexe, se concentre sur l'utilisation de notre solution mixte, couplant approches population et individu-centrées parallélisées à la fois sur GPUs et CPUs, pour la simulation de l'écoulement sanguin et de la coagulation plasmatique au sein d'un vaisseau sanguin. Bien qu'incomplète en raison de l'absence de plusieurs phénomènes importants biologiquement, notre application permet de mettre en œuvre l'ensemble des algorithmes et concepts développés au cours de notre proposition dans le chapitre 2 de ce mémoire. Elle propose ainsi une simulation multi-phénomènes massivement parallélisée et exploitant au mieux les atouts des architectures multicœur et multiprocesseur, ainsi que ceux des architectures graphiques, mis en avant dans l'état de l'art de ce mémoire. Les résultats obtenus mettent clairement en évidence l'impact néfaste sur les performances des mécanismes de transfert de données et de synchronisation inhérents à ce type de simulation. À défaut de permettre des simulations à la hauteur de ce qui pourrait être naïvement attendu en multipliant les GPUs, nos méthodes proposent tout de même de simuler simultanément un modèle individu-centré parallélisé sur CPUs avec des modèles population-centrés traités sur un unique GPU, et cela avec des performances nettement supérieures face à celles des approches séquentielles classiques s'exécutant sur un unique CPU.

Ces deux exemples d'application nous permettent d'insister sur le fait que les modèles individu-centrés peuvent bénéficier avantageusement des architectures multicœur et multiprocesseur, alors que les modèles population-centrés sont plus adaptés pour une parallélisation sur architecture graphique. En outre, ces applications montrent que, tant qu'ils s'accompagnent de mémoires caches, l'augmentation du nombre de CPUs, y compris par la multiplication de processeurs distincts, est bénéfique pour les performances des simulations individu-centrées. Néanmoins, il n'en va pas de même pour les GPUs. En effet, en raison des communications que demande une configuration multi-GPUs, la multiplication des GPUs ne permet pas d'ac-

célérer les simulations population-centrées efficacement. De la même manière qu'avant 2004 l'augmentation de puissance des CPUs reposait essentiellement sur leur montée en fréquence, il semble que les GPUs soient condamnés à être constitués de puces toujours plus rapidement cadencées et dotées de toujours plus de cœurs pour aborder plus efficacement des simulations dans lesquelles la synchronisation des multiples traitements reste cruciale.

Conclusion

La problématique soulevée dans ce travail de thèse est d'étudier la manière optimale d'exploiter les architectures parallèles actuelles pour simuler des systèmes biologiques selon des modèles mixtes, c'est-à-dire couplant des approches population et individu-centrées. Cette étude nous a amenés à proposer des solutions techniques permettant de simuler conjointement des modèles individu-centrés sur les architectures multicœur et multiprocesseur, ainsi que des modèles population-centrés sur des architectures graphiques pouvant néanmoins être épaulées par les processeurs à disposition si nécessaire. Pour développer ces solutions techniques, nous nous sommes attachés à respecter les contraintes imposées par chacune de ces architectures, et notamment la hiérarchie mémoire des CPUs et l'architecture SIMT des GPUs, ainsi que celles imposées par les modélisations étudiées. En outre, nous avons illustré notre proposition sur l'exemple de la coagulation du sang. Nous dressons ici un bilan de nos travaux et proposons quelques perspectives.

Bilan

Le point de départ de nos recherches fut d'effectuer une revue des différents travaux de modélisation et de simulation des systèmes biologiques existants. Cette étude a mis en évidence deux manières de modéliser les nombreuses et diverses entités que font intervenir de tels systèmes. En effet, ces entités peuvent être considérées soit comme une population d'individus, répartie dans un espace discrétisé, dont les variations d'effectif traduisent le comportement de manière synthétique, soit individuellement, ce qui permet de décrire le comportement de chacune d'elles à une échelle plus fine. La variété des phénomènes survenant au sein des systèmes biologiques implique généralement le besoin de coupler ces deux approches pour que chaque phénomène puisse être modélisé de manière adéquate. Un tel couplage permet en outre de bénéficier des avantages offerts par chacune de ces deux approches, tout en s'affranchissant de leurs défauts respectifs.

Face à la quantité importante d'informations manipulées par les modélisations de

systèmes biologiques, leur simulation peut être extrêmement coûteuse en temps de calcul. Afin de remédier à ce problème, le recours à des mécanismes de parallélisation s'avère nécessaire. Ces derniers permettent d'augmenter considérablement le nombre d'unités de calcul dédiées à l'exécution des simulations, notamment en exploitant les architectures multicœur et multiprocesseur ou les architectures graphiques. Cependant, bien que de telles architectures parallèles offrent des performances élevées, elles imposent également des contraintes qu'il est nécessaire de respecter, sous peine de fortement sous-exploiter les capacités de parallélisation mises à disposition.

Ces deux constats révèlent l'intérêt de développer des simulations de systèmes biologiques à la fois parallélisées et basées sur des modélisations mixtes. Cependant, notre revue des travaux présents dans la littérature nous amène à constater qu'aucune approche de ce type n'est actuellement proposée. De ce fait, nous avons introduit nos propres solutions techniques pour répondre à ce problème. Ainsi, nous proposons une modélisation mixte au sein de laquelle nous simulons les modèles individu-centrés sur CPU, alors que les modèles population-centrés sont majoritairement simulés par l'intermédiaire de GPUs, pouvant également être accompagnés des CPUs. Le fil conducteur guidant nos travaux fut de respecter au maximum les contraintes imposées par chacune des architectures parallèles que nous utilisons, notamment la hiérarchie mémoire des architectures multicœur et multiprocesseur, l'architecture SIMT de leurs homologues graphiques, ainsi que leur adéquation aux différentes modélisations.

Nous avons appliqué notre proposition à la simulation des phénomènes liés à la coagulation du sang. En premier lieu, nous avons proposé un modèle individu-centré pour la simulation de la cinétique biochimique à l'échelle microscopique. Ensuite, nous avons présenté l'application phare de nos travaux : la modélisation et simulation d'un vaisseau sanguin virtuel au sein duquel une brèche vasculaire survient. Cette application, plus complète et complexe que la précédente, met en œuvre l'intégralité des solutions techniques que nous proposons, c'est-à-dire la simulation de modèles individu-centrés sur CPUs et la simulation de modèles population-centrés sur GPUs et CPUs.

L'application de nos travaux à la simulation de la cinétique biochimique puis à celle d'un vaisseau sanguin virtuel nous a permis d'évaluer les performances de nos algorithmes. Notre première application, uniquement basée sur notre modèle individu-centré, a montré que, tant que les CPUs s'accompagnent de mémoires caches, les performances de nos simulations augmentent linéairement en fonction du nombre de CPUs utilisés. Ainsi, ce résultat nous laisse penser que nous pourrions obtenir des résultats encore meilleurs sur des ordinateurs disposant de plus de vingt-quatre CPUs. Contrairement à cette application, la seconde met en œuvre l'intégralité des solutions techniques que nous proposons, c'est-à-dire la simulation de modèles individu-centrés sur CPUs et la simulation de modèles population-centrés sur GPUs et CPUs. Elle nous a permis d'évaluer les performances de nos simulations en fonction de différentes configurations matérielles allant de l'utilisation d'un unique GPU à celle de cinq GPUs, accompagnés de vingt-quatre CPUs. Cette étude a mis en évidence l'impact néfaste sur les performances des transferts de données entre GPUs lorsque plusieurs d'entre eux sont utilisés, ainsi que celui des barrières de synchronisation. Ces deux précautions sont indispensables pour garantir la cohérence des simulations sur de multiples GPUs. C'est lorsque ces précautions sont rendues inutiles, en dédiant un unique GPU à l'ensemble des simulations des modèles population-centrés (représentant l'écoulement et la coagulation plasmatiques) et

en consacrant les CPUs à la simulation du modèle individu-centré (représentant les éléments figurés), que notre application de vaisseau sanguin virtuel atteint ses performances maximales.

Les solutions techniques parallèles que nous proposons offrent des performances nettement supérieures à celles obtenues par une exécution séquentielle classique. Ainsi, nos travaux témoignent de l'intérêt de coupler les architectures multicœur et multiprocesseur avec les architectures graphiques pour la simulation de systèmes biologiques. Ils mettent notamment en évidence les adéquations respectives des CPUs et des GPUs pour la simulation des modèles individu-centrés et des modèles population-centrés et expliquent la limite inhérente à l'utilisation de multiples GPUs pour ce type de problème. Nous proposons maintenant quelques perspectives et réflexions sur de futurs travaux.

Perspectives

Deux types de perspectives se dégagent de nos travaux. Les premières sont d'ordre général et concernent les solutions techniques de couplage et de parallélisation que nous proposons. Les secondes émanent des travaux que nous avons menés dans le cadre de la simulation de la coagulation du sang, et particulièrement celui de la simulation d'un vaisseau sanguin virtuel.

Perspectives générales

Puisque nos travaux mettent en évidence le contraste qui existe entre les bonnes performances des simulations biologiques population-centrées exécutées sur un unique GPU avec celles, très en deçà, des mêmes simulations exécutées sur de multiples GPUs, ils nous encouragent à attaquer ce facteur limitant afin de le dépasser. Une première solution envisageable pourrait consister à dédier chaque GPU à la simulation population-centrée d'un phénomène distinct dans l'application. Par exemple, dans le cas de notre application autour du vaisseau sanguin, il parait tentant de confier le calcul de la coagulation à un GPU alors qu'un autre se chargerait simultanément de celui du flux sanguin. Seulement ces deux calculs sont liés : le premier subit l'advection des vitesses produites par le second alors que le second subit l'influence des obstacles produits par le premier. Il serait alors indispensable de transférer l'intégralité des informations concernant la densité du milieu et les vitesses du fluide de l'un vers l'autre et réciproquement, ce qui serait probablement encore plus pénalisant que les transferts de frontières qui limitent les performances. De plus, rien ne garantit que ces calculs distincts aient la même durée ; certains des GPUs impliqués ne seraient alors pas utilisés à leur plein potentiel. Et enfin, ceci ne passerait pas à l'échelle puisqu'il est peu probable de simuler dans une même application suffisamment de phénomènes population-centrés distincts pour occuper tout le matériel (dans notre application, seulement deux GPUs sur les cinq disponibles seraient utilisés). Ceci est une limitation inhérente au parallélisme de tâche.

Pour viser un meilleur passage à l'échelle des performances il nous faut donc persister dans le parallélisme de donnée en prenant à bras-le-corps le problème de la synchronisation des GPUs et de la mise à jour des frontières des sous-maillages qui leur sont confiés. Nous

envisageons de relâcher quelque peu ces contraintes en n'effectuant ces opérations nécessaires qu'une fois tous les quelques cycles de simulation. Ceci est d'autant plus envisageable que ces deux contraintes sont liées : il est inutile de synchroniser un GPU avec ses voisins s'ils n'échangent pas leurs zones frontalières. Bien entendu, ceci aura un impact sur la validité des résultats obtenus. Il est donc nécessaire de mener une étude sur l'ampleur des erreurs induites, voire d'envisager l'introduction de termes correctifs pour compenser une éventuelle dérive des résultats. La difficulté majeure vient du fait que cette étude ne peut être menée une fois pour toutes de manière générique. Elle doit au contraire être adaptée à chaque sujet applicatif puisque ces erreurs et termes correctifs sont très dépendants des équations régissant chaque phénomène spécifique simulé.

Perspectives liées à la simulation de l'hémostase

Nous avons appliqué nos travaux à la simulation de phénomènes liés à la coagulation du sang. Dans le cadre de notre première application, la cinétique biochimique, nous envisageons d'utiliser notre implémentation pour l'étude des réactions enzymatiques de la coagulation plasmatique, nécessitant la présence de membranes physiologiques pour être efficaces et transformant un milieu soluble homogène en un milieu insoluble hétérogène dû à la formation du réseau de fibrine. De plus, nous avons déjà entrepris un travail permettant d'augmenter la durée du pas de temps de nos simulations, sans pour autant en diminuer la précision. Pour cela, nous travaillons sur des méthodes mathématiques permettant de prédire les temps de première rencontre entre molécules. Ces méthodes visent à extrapoler une loi décrivant la fréquence des collisions des molécules en fonction de leur voisinage. Elles sont élaborées par l'intermédiaire d'une multitude de simulations distribuées en réseau dont les résultats sont utilisés pour ajuster les paramètres de lois empiriques décrivant la cinétique biochimique du système biologique étudié.

Dans le cadre de notre seconde application, à savoir le vaisseau sanguin virtuel, nous avons insisté à plusieurs reprises sur l'importance de modéliser l'hémostase primaire et notamment les mécanismes plaquettaires. Cet ajout consistera principalement à modifier le comportement des cellules en ajoutant des mécanismes de détection et de gestion de collisions au sein du modèle individu-centré les représentant. Ces mécanismes seront très similaires à ceux développés dans le cadre de notre première application et impacteront les modèles population-centrés régissant les comportements de l'écoulement et de la coagulation plasmatique. L'ajout d'un tel phénomène rendra notre modélisation de vaisseau sanguin plus complète et permettra le développement de potentielles applications dédiées par exemple à la sensibilisation de patients à leur traitement ou encore au développement de molécules dans l'industrie pharmaceutique.

Annexe A

Posters

Cette annexe rassemble les deux posters auxquels j'ai contribué durant mes trois années de thèse. Ils ont chacun fait l'objet d'une présentation lors d'une conférence internationale, respectivement en bioinformatique et en hémostase.

La figure A.1 illustre le poster présenté lors de l'*European Conference on Computational Biology* (ECCB 2012). Il propose un résumé de nos premiers travaux concernant la simulation de la cinétique biochimique à l'échelle microscopique (cf. section 3.1). Il se focalise principalement sur notre modélisation de la cinétique biochimique et met en évidence le besoin de paralléliser les calculs lors de telles simulations.

La figure A.2 illustre un poster présenté lors du 24e congrès de l'*International Society of Thrombosis and Haemostasis* (ISTH 2013) et qui résume les travaux issus de notre collaboration avec la société Synapse et l'équipe du Professeur Coen Hemker. Ces derniers avaient pour objectif de simuler, à la fois numériquement et biologiquement, la décroissance des courbes de génération de thrombine (concept introduit dans la section 3.2.4.1). Ces travaux, bien que réalisés durant cette thèse, n'utilisent aucun des mécanismes de simulation des systèmes biologiques que nous proposons dans ce mémoire.

3D microscopic scale simulation of biochemical kinetics

Laurent Crépin (1), Fabrice Harrouet (1), Sébastien Kerdélo (2) and Pascal Redou (1)

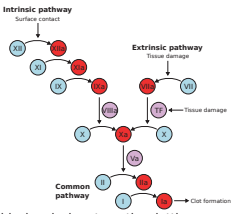
(1) European Center for Virtual Reality (CERV, EA3883), European University of Brittany, Plouzané, France (**crepin@enib.fr**)
(2) Diagnostica Stago, Gennevilliers, France

1. Biochemical kinetics simulation

The study of biological systems implies the study of biochemical kinetics. By governing reaction rates, this phenomenon is responsible for the evolution of chemical concentrations in such systems. The table below (Kerdélo 2006) shows different methods to model and simulate this biological process at various scales. For the sake of accuracy, we propose an individual-based system which enables the simulation of biochemical kinetics at the microscopic scale.

Scale \ Medium	Homogeneous medium	Heterogeneous medium
Macroscopic	ODE	PDE
Mesoscopic	Stochastic ODE	Stochastic PDE
Microscopic	Individual-based system	

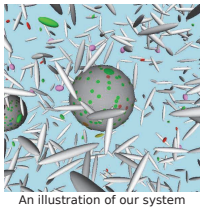
ODE: Ordinary Differential Equations
PDE: Partial Differential Equations



A biochemical system: the clotting cascade

2. Individual-based system

Our individual-based system is populated by many 3D entities which are:
- **autonomous** (Tisseau 2001), they follow a three time life-cycle:
  - perception, they analyse their environment
  - decision-making, they choose their next action
  - action, they apply the previous decision
- **reactive** (Drogoul 1993), they follow the stimulus→response



An illustration of our system

In opposition to classical modeling methods such as differential equations (Tolle and Le Novère 2006), our system is capable of handling easily geometrical constraints of biological processes such as **membrane binding** events.

3. Model and algorithms

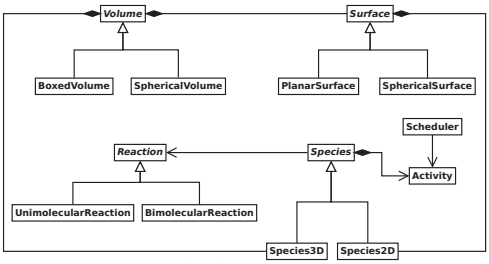
Model

Our model is composed of a chemical environment (*Volume*) in which three types of entity evolve:
- species in solution (*Species3D*)
- species bound to membranes (*Species2D*)
- membranes with which species in solution can bind (*Surfaces*)

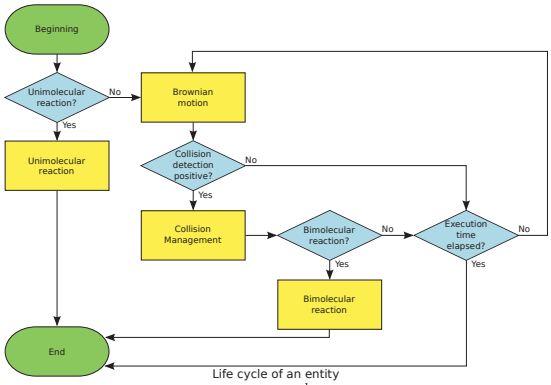
These entities are capable of reacting alone (*UnimolecularReaction*) or with one another (*BimolecularReaction*). The UML class diagram below illustrates the model that we propose, whereas the diagram opposite defines the behavior of an entity during each simulation timestep.

Diffusion

Each entity moves according to Brownian motion, i.e. an erratic motion of a particule caused by the collisions with solvent particules. The new position of an entity is computed by 3D random Gaussian sampling (on $\vec{x}, \vec{y}, \vec{z}$).



UML class diagram of our model



Life cycle of an entity

Unimolecular reactions (e.g. $R_i : A \xrightarrow{k} B + C$)
To determine if an entity should undergo an unimolecular reaction R_i, an uniform random sampling is performed among reaction probabilities $P(R_i)$:

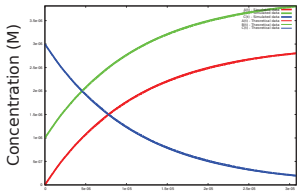
$$P(R_i) = \frac{k_i}{\sum_{j=1}^{n} k_j} \times \left(1 - \exp(-\sum_{j=1}^{n} k_j \times \Delta t)\right)$$

Bimolecular reactions (e.g. $R_i : A + B \xrightarrow{k} C$)
When two entities collide with each other, a bimolecular reaction R_i may happen depending on the reaction rate. According to Smoluchowski theory (Smoluchowski 1917), the probability $P(R_i)$ of such a reaction is:

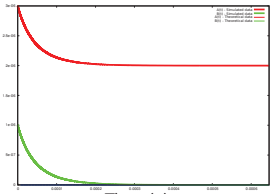
$$P(R_i) = \frac{k}{4\pi(D_A + D_B)(r_A + r_B)N_A}$$

4. Results and perspectives

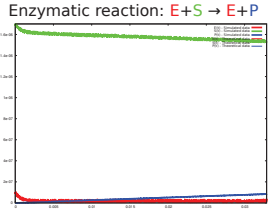
Unimolecular reaction: C → A+B



Bimolecular reaction: A+B → ∅



Enzymatic reaction: E+S → E+P



Results:
- accurate simulator
- validation on simple reactions
- comparison with ODE/PDE

Constraints of the microscopic scale:
- short simulation timestep (10^{-9}s)
- many entities (10^{6})
- computationally expensive

⇓

Perspective: **Parallel computing**

Figure A.1 – **Poster ECCB 2012** – Cette figure représente le poster que nous avons présenté lors de l'*European Conference on Computational Biology* (ECCB 2012). Il synthétise les premiers travaux réalisés sur notre simulateur de cinétique biochimique à l'échelle microscopique (cf. section 3.1).

Biochemical and numerical simulation of thrombin decay

R.M.W. Kremers[1], L. Crépin[2], S. Kerdélo[3], R.J. Wagenvoord[1], H.C. Hemker[1]

[1] Synapse bv, Maastricht University, the Netherlands; [2] CERV, European University of Brittany, France; [3] Specialized Systems, Diagnostica Stago, France

RATIONALE

Numerical models of the coagulation system are not successful in imitating real life thrombin generation curves (TG) as measured in plasma. One of the reasons is the discrepancy between the number of parameters required for the input (>50) and the parameters of the output (5). We tackle this problem by simulating the decay system first, which requires only a small number of input parameters. The decay part was studied either in non-recalcified plasma to which thrombin was added or in recalcified plasma to which a high concentration of tissue factor (TF) was added. In the latter case, all prothrombin is converted within a few minutes, and thrombin decay can be isolated from the TG curve. As a check we use a biochemical model, consisting of purified proteins, in which the decay system of the plasma is imitated.

AIM To obtain a numerical model that accurately describes thrombin decay in plasma.

BACKGROUND

- Thrombin decay during TG is predominantly caused by antithrombin (AT; ~65%) and α2Macroglobulin (α2M; ~25%).
- Fibrinogen is expected to influence thrombin decay because it contains thrombin binding sites which compete for thrombin binding with natural antithrombins.

MATERIALS & METHODS

AT and α2M levels of plasma were measured in functional assays and fibrinogen levels were measured by the Von Clauss method. Mock plasma was made by adding these concentrations of inhibitors and fibrinogen to Ringer solution containing bovine serum albumin and citrate. The course of thrombin concentration in time was assessed from the cleavage of thrombin substrate ZGGR-AMC in plasma. Thrombin decay was initiated in plasma either after the addition of 500 nM thrombin or by the addition of 50 pM TF which causes rapid conversion of prothrombin and a subsequent phase of measurable thrombin decay. Numerical simulation was performed by the model described below. The rate constants $k_{dec.AT}$ and $k_{dec.\alpha2M}$ were found by fitting the simulated curve to the experimental data.

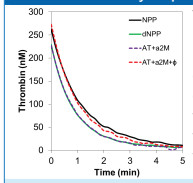
NUMERICAL MODEL

$$- dT/dt = k_{dec.AT} \cdot AT_t \cdot T_t + k_{dec.\alpha2M} \cdot \alpha2M_t \cdot T_t$$

$$d(\alpha2MT)/dt = k_{dec.\alpha2M} \cdot T_t$$

RESULTS

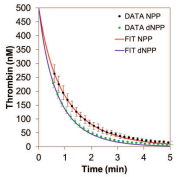
Thrombin decay in plasma and in a biochemical system



Thrombin decay in normal pool plasma (NPP; black) and defibrinated NPP (dNPP; green) are not comparable. A biochemical system consisting of AT and α2M at their plasma concentrations (purple) gives the same thrombin decay as defibrinated NPP, whereas the same biochemical system containing fibrinogen imitates thrombin decay in NPP.

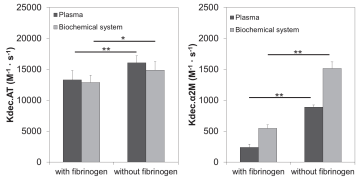
Taking into accoung fibrinogen is essential to the accurate simulation of thrombin decay.

Thrombin decay in plasma and in a numerical model



Thrombin decay curves measured in NPP (black) and defibrinated NPP (dNPP; green) can be fitted with the numerical model by varying the rate constants for thrombin decay by AT ($k_{dec.AT}$) and α2M ($k_{dec.\alpha2M}$). The obtained values for these rate constants in plasma and the biochemical model are shown below.

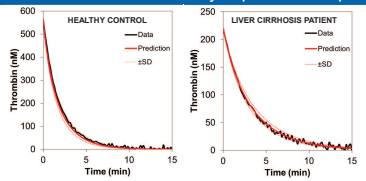
Effect of fibrinogen on the rate of thrombin decay



Fibrinogen significantly reduces the rate of thrombin decay by both AT (left) and α2M (right) in plasma and in a biochemical system.

The reversible binding of thrombin to fibrin causes free thrombin levels to decrease, and consequently less thrombin to interact to with AT and α2M. Thrombin binding to fibrinogen was introduced to the numerical model as a fibrinogen level-dependent decrease in free thrombin concentration.

Prediction of thrombin decay in plasma samples

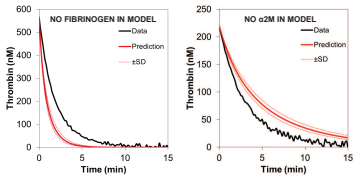


Thrombin decay was predicted with the numerical model based on plasma AT, α2M and fibrinogen levels (red; ±1SD dashed red lines). Thrombin decay was measured in a plasma sample (black). The left graph shows thrombin decay in a healthy control and the right one thrombin decay in a liver cirrhosis patient.

The table on the left summarizes the factor concentrations of both samples.

Factor	Healthy donor	Liver cirrhosis patient
[AT] (μM)	2.21	0.65
[α2M] (μM)	3.22	4.20
[Fibrinogen] (μM)	7.99	4.00

Importance of fibrinogen and α2M to the numerical model



Thrombin decay was predicted with a numerical model that did not reckon with the plasma fibrinogen level for the same healthy control sample (left). Thrombin decay in the same liver cirrhosis sample was predicted with a numerical model that does not take into account the plasma α2M level.

CONCLUSION

- Realistic simulation of thrombin decay in plasma requires that AT and α2M and fibrinogen are taken into account. This is especially important in conditions in which the AT/α2M balance is altered, such as liver cirrhosis and nephrotic syndrome.
- Realistic simulation of decay is a prerequisite to realistic simulation of TG.

ISTH 2013 NETHERLANDS

Contact information: r.kremers@thrombin.com

Synapse b.v.

Figure A.2 – **Poster ISTH 2013** – Cette figure représente le poster présenté lors du 24e congrès de l'*International Society of Thrombosis and Haemostasis* (ISTH 2013). Il synthétise les travaux réalisés en collaboration avec la société Synapse et l'équipe du Professeur Coen Hemker au sujet de la simulation de la phase de décroissance de la courbe de génération de thrombine.

Annexe B

Données physiologiques du vaisseau sanguin virtuel

Cette annexe vise à recenser l'ensemble des données physiologiques nécessaires à la modélisation de notre vaisseau sanguin virtuel. Elle regroupe des informations concernant le vaisseau sanguin en lui-même, mais également le plasma et les éléments figurés. Ces données ont été obtenues en recoupant les informations recueillies dans divers ouvrages de la littérature traitant de la physiologie humaine. Nous notons que la valeur de certaines données fluctue énormément en fonction de l'ouvrage consulté. De ce fait, nous nous efforçons de justifier nos choix de paramètres.

B.1 Caractéristiques du vaisseau et de l'écoulement sanguin

B.1.1 Dimensions du problème

Nous choisissons d'étudier une veine de diamètre moyen. Selon la littérature, et notamment [Brown *et al.*, 2006] et [Khurana, 2005], le diamètre d'un tel vaisseau est d'approximativement 5 mm. Néanmoins, comme le montre la figure 3.9, nous ne modélisons pas le vaisseau dans son intégralité : nous nous focalisons sur une zone parallélépipédique de dimensions 260 $\times$ 100 $\times$ 128 μm centrée sur la brèche vasculaire. Les dimensions de cette dernière correspondent à un tiers de la paroi modélisée soit approximativement 87 $\times$ 43 μm. Notons que les cellules endothéliales, de forme hexagonale, composant la paroi mesurent chacune 26 $\times$ 13 μm [Garipcan *et al.*, 2011].

B.1.2 Pression intravasculaire

La pression intravasculaire au sein d'une vaisseau sanguin de diamètre 5 mm peut varier entre 0 et 10 mmHg [1] [Khurana, 2005; Brown *et al.*, 2006; Sherwood, 2010; Rhoades et Bell, 2012]. De ce fait, la pression intravasculaire au sein de notre vaisseau virtuel doit valoir au minimum 0 Pa et au maximum 1333 Pa. Nous choisissons une valeur intermédiaire de 5 mmHg soit environ 667 Pa.

B.1.3 Vitesse de l'écoulement

Afin de pouvoir appliquer les conditions initiales présentées dans la section 3.2.2.2, nous souhaitons déterminer la vitesse maximale v_0 de l'écoulement sanguin dans la tranche inférieure du parallélépipède que nous modélisons, c'est-à-dire la tranche la plus proche du centre du vaisseau. En effet, cette vitesse conditionne la mise en œuvre de l'écoulement de Couette que nous fournissons en entrée de notre vaisseau sanguin.

Nous avons choisi d'utiliser un écoulement de Couette car la géométrie du problème nous l'impose. En effet, le volume au sein duquel nous modélisons l'écoulement plasmatique est constitué d'un parallélépipède rectangle et la paroi vasculaire est assimilée à une surface plane. Or, dans des conditions physiologiques normales, un vaisseau sanguin est cylindrique et l'écoulement le parcourant prend la forme d'une parabole : la vitesse au centre du vaisseau est maximale alors qu'elle est nulle contre la paroi vasculaire. Un tel écoulement suit une loi de Poiseuille-Hagen [Sherwood, 2010; Rhoades et Bell, 2012] et si nous considérons une section cylindrique du vaisseau, elle s'exprime de la manière suivante :

$$v(r) = v_{max} \times \left(1 - \frac{r^2}{R^2}\right) \tag{B.1}$$

où

- $v(r)$ désigne la vitesse de l'écoulement selon l'axe $\vec{x}$ en fonction du rayon r (unité SI : m.s^{-1}) ;
- v_{max} désigne la vitesse maximale de l'écoulement au centre du vaisseau, c'est-à-dire à $r = 0$ (unité SI : m.s^{-1}) ;
- R désigne le rayon total du vaisseau (unité SI : m).

Un tel écoulement est illustré sur la figure B.1. Celle-ci met en évidence les valeurs de vitesse qui nous intéressent, c'est-à-dire celle au centre du vaisseau, celle près de sa paroi et celle dans la tranche inférieure de notre parallélépipède. Cette dernière, que nous souhaitons déterminer, se situe à un rayon $r = R - h$ où h correspond à la hauteur de la zone intravasculaire que nous modélisons, c'est-à-dire 50 μm.

La vitesse maximale de l'écoulement sanguin v_{max} au centre d'un vaisseau de diamètre 5 mm (et donc de rayon R = 2.5 mm) varie fortement selon les ouvrages. Nous choisissons

1. 1 millimètre de Mercure (mmHg) $\simeq$ 133.322 Pascal (Pa).

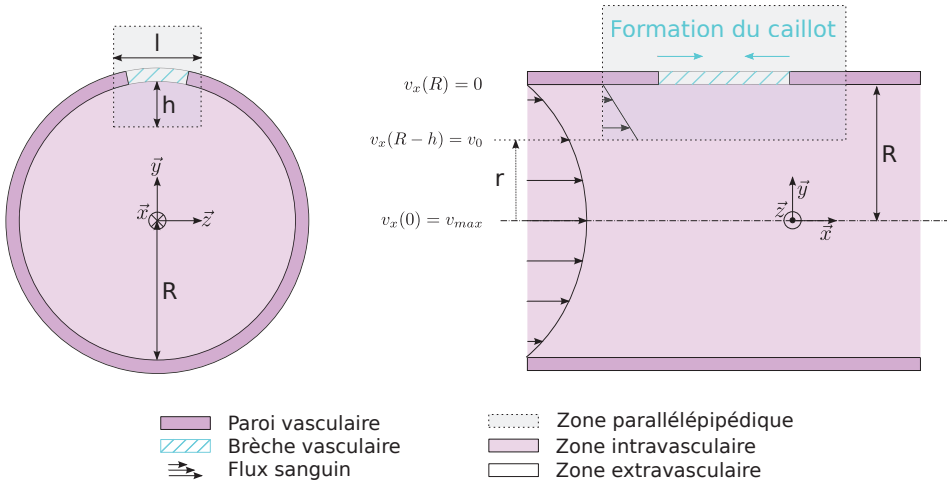


Figure B.1 – **Écoulement de Poiseuille** – Le schéma de gauche représente une section de vaisseau et situe le domaine parallélépipédique que nous étudions. Le schéma de droite constitue une vue de profil de cette section et représente un écoulement de Poiseuille selon l'axe $\vec{x}$. Il propose également une correspondance avec l'écoulement de Couette au sein du parallélépipède.

de prendre $v_{max} = 0.1$ m.s^{-1}, qui respecte la majorité des intervalles de valeurs que la littérature propose [Khurana, 2005; Sherwood, 2010; Rhoades et Bell, 2012]. Ces données nous permettent de déterminer la vitesse maximale v_0 dans la tranche inférieure du maillage :

$$\begin{aligned} v_0 = v(R-h) &= v_{max} \times \left(1 - \frac{(R-h)^2}{R^2}\right) \\ &= 0.1 \times \left(1 - \frac{(2.5 \times 10^{-3} - 50 \times 10^{-6})^2}{(2.5 \times 10^{-3})^2}\right) \simeq 0.004\ m.s^{-1}. \end{aligned} \tag{B.2}$$

La vitesse de l'écoulement sanguin dans les mailles de la tranche inférieure de notre maillage est donc de 0.004 m.s^{-1}.

B.1.4 Viscosité et densité plasmatique

Selon [Khurana, 2005] et [Rhoades et Bell, 2012], la viscosité du plasma est de 1.7 cP, c'est-à-dire 1.7×10^{-3} Pa.s. Elle est plus importante que celle de l'eau mais nettement plus faible que celle du sang (qui regroupe pour rappel, le plasma et les éléments figurés). La densité ou masse volumique du plasma vaut quant à elle 1025 kg.m^{-3}.

B.1.5 Bilan

Le tableau B.1 regroupe les différentes caractéristiques du vaisseau sanguin et de l'écoulement détaillées précédemment afin d'en faciliter leur lecture.

Diamètre du vaisseau		5	mm
Pression intravasculaire		5	mmHg
		667	Pa
Vitesse du sang	(au centre)	0.1	m.s $^{-1}$
	(tranche inférieure)	0.004	m.s $^{-1}$
Volume simulé		260 × 100 × 128	μm
Densité du plasma		1025	kg.m $^{-3}$
Viscosité du plasma		1.7×10^{-3}	Pa.s

Tableau B.1 – **Caractéristiques du vaisseau et de l'écoulement sanguin** – Ce tableau rassemble l'ensemble des données essentielles à la mise en œuvre de notre modèle de vaisseau sanguin virtuel.

B.2 Caractéristiques des éléments figurés

Le tableau B.2 regroupe les données physiologiques concernant les éléments figurés immergés dans le plasma. Les données inscrites dans ce tableau ne sont que des estimations. En effet, elles dépendent des individus sur lesquels elles ont été mesurées et également des systèmes de mesures utilisés.

	Nombre (par mm^3 de sang)	**Dimensions** (en μm)	**Masse** (en pg)
Plaquettes	entre 150 000 et 400 000	diamètre : entre 2 et 4	10
Globules rouges	5 000 000	diamètre : entre 7 et 9 épaisseur : entre 1 et 2	45
Globules blancs	entre 4 000 et 11 000	diamètre : entre 10 et 20	40

Tableau B.2 – **Caractéristiques des éléments figurés** – Ce tableau regroupe le nombre, la taille ainsi que la masse des éléments figurés du sang. Ces données sont approximatives mais procurent une estimation correcte. Notons que les valeurs des masses des plaquettes, globules rouges et blancs proviennent respectivement des articles [Kiem *et al.*, 1979], [Efrati *et al.*, 1977] et [Mysliwski et Korczak, 1986]. Les autres données sont tirées de [Khurana, 2005; Sherwood, 2010; Rhoades et Bell, 2012].

La quantité d'éléments figurés présents dans le sang est tellement importante qu'elle ne nous permet pas de visualiser la formation du caillot sanguin. De ce fait, nous choisissons arbitrairement d'en réduire le nombre. Cela n'a aucun impact sur les résultats de nos simulations puisqu'à l'heure actuelle, les cellules sanguines, et notamment les plaquettes, n'ont aucune influence sur notre modèle de coagulation du sang. Ainsi, nous choisissons de ne modéliser que 250 globules rouges, 5 globules blancs et 50 plaquettes dans le parallélépipède rectangle

que nous étudions. Ces valeurs, bien que largement inférieures aux valeurs physiologiques normales, respectent les proportions qui existent entre elles. Notons que ces trois quantités ne sont que des moyennes. En effet, ces cellules suivent l'écoulement plasmatique, et donc elles ont tendance à s'échapper du vaisseau (soit par la brèche, soit par sa sortie). Pour alimenter nos simulations en éléments figurés, nous introduisons donc régulièrement des cellules sanguines à l'entrée de notre vaisseau. Cette alimentation continue nous permet d'obtenir, en moyenne, la quantité de cellules que nous désirons.

B.3 Paramètres de simulation

Lors de la présentation de notre modèle individu-centré dans le chapitre 2, nous avons indiqué modéliser le temps de manière discrète. En outre, les méthodes numériques que nous utilisons pour résoudre les équations régissant le comportement de l'écoulement et de la coagulation plasmatique utilisent le même type de modélisation temporelle. Ce dernier nécessite de définir un pas de temps caractérisant la durée des cycles de nos simulations. Nous choisissons d'utiliser un pas de temps de 1 ms pour simuler les comportements de l'écoulement et des éléments figurés et un pas de temps de 0.1 ms pour simuler la coagulation plasmatique. L'utilisation de pas de temps plus élevés provoque des instabilités au sein de nos simulations.

Par ailleurs, les modèles population-centrés que nous utilisons nécessitent un maillage cartésien de l'espace. Nous choisissons d'utiliser des mailles cubiques de 2 μm de côté. Les dimensions du parallélépipède rectangle que nous considérons étant de $260 \times 100 \times 128$ μm, nous obtenons un maillage global constitué de $132 \times 52 \times 66$ mailles [2], soit 453 024 mailles au total.

2. Dans le cadre de la résolution des équations de *Navier-Stokes*, une rangée de mailles supplémentaires est ajoutée sur chaque face du parallélépipède rectangle que nous modélisons.

Références bibliographiques

Abbott, C., Berry, M., Comiskey, E. et Gross, L. (1997). Parallel individual-based modeling of everglades deer ecology. *IEEE Computational Science and Engineering*, 4(4):60–78.
Cité page 5

Alarcón, T., Byrne, H. et Maini, P. (2003). A cellular automaton model for tumour growth in inhomogeneous environment. *Journal of Theoretical Biology*, 225(2):257–274.
Cité page 8

Aldinucci, M., Coppo, M., Damiani, F., Drocco, M., Torquati, M. et Troina, A. (2011). On designing multicore-aware simulators for biological systems. *Dans 19th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, page 19.
Cité page 23

Alves, R., Antunes, F. et Salvador, A. (2006). Tools for kinetic modeling of biochemical networks. *Nature Biotechnology*, 24(6):667–72.
Cité page 76

Amar, P. (2012). Comparative study of some methods for simulation of biochemical reactions. *Dans Ecole de Printemps 2012 de la Société Francophone de Biologie Théorique*, Saint Flour, France.
Cité page 5

Amar, P., Bernot, G. et Norris, V. (2004). HSIM: a simulation programme to study large assemblies of proteins. *Journal of Biological Physics and Chemistry*, 4(2):50–63.
Cité page 76

AMD (2012). AMD accelerated parallel processing programming guide. Rapport technique, Advanced Micro Devices, Inc.
Cité page 25

An, G., Mi, Q., Dutta-Moscato, J. et Vodovotz, Y. (2009). Agent-based models in translational systems biology. *Wiley interdisciplinary reviews. Systems biology and*

medicine, 1(2):159–71.
Cité page 11

Andrews, S. S., Addy, N. J., Brent, R. et Arkin, A. P. (2010). Detailed simulations of cell biology with Smoldyn 2.1. *PLoS Computational Biology*, 6(3):e1000705.
Cité page 76

Andrews, S. S. et Bray, D. (2004). Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. *Physical Biology*, 1(3-4):137–51.
Cité page 79

Andrews, S. S., Dinh, T. et Arkin, A. (2009). Stochastic models of biological processes. *Dans Encyclopedia of Complexity and Systems Science*, pages 8730–8749. Springer.
Cité pages 5, 7 et 11

Baugh, R. J., Broze, G. J. et Krishnaswamy, S. (1998). Regulation of extrinsic pathway factor Xa formation by tissue factor pathway inhibitor. *Journal of Biological Chemistry*, 273(8):4378–4386.
Cité page 82

Beberg, A. L., Ensign, D. L., Jayachandran, G., Khaliq, S. et Pande, V. S. (2009). Folding@home: Lessons from eight years of volunteer distributed computing. *Dans 2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8. IEEE.
Cité pages 15, 38 et 39

Berg, H. (1993). *Random walks in biology*. Princeton University Press.
Cité page 79

Beurton-Aimar, M., Parisey, N. et Vallée, F. (2011). Multi-agent model for simulation at the subcellular level. *Dans Advances in Artificial Life. Darwin Meets von Neumann*, volume 5777 de *Lecture Notes in Computer Science*, pages 361–368. Springer.
Cité page 5

Bobashev, G. V., Goedecke, D. M., Feng, Y. et Epstein, J. M. (2007). A hybrid epidemic model: Combining the advantages of agent-based and equation-based approaches. *Dans 2007 Winter Simulation Conference*, pages 1532–1537. IEEE.
Cité page 13

Bordawekar, R., Bondhugula, U. et Rao, R. (2010). Believe it or not!: multi-core CPUs can match GPU performance for a FLOP-intensive application! *Dans Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, pages 537–538, Vienna, Austria. ACM Press.
Cité page 35

Bos, M. H. A., Boltz, M., St Pierre, L., Masci, P. P., de Jersey, J., Lavin, M. F. et Camire, R. M. (2009). Venom factor V from the common brown snake escapes hemostatic regulation through procoagulant adaptations. *Blood*, 114(3):686–92.
Cité page 82

Boyd, C. (2008). The DirectX 11 compute shader. *Dans ACM SIGGRAPH*.
Cité page 25

Brown, S. P., Miller, W. C. et Eason, J. M. (2006). *Exercise physiology: basis of human movement in health and disease*. Lippincott Williams & Wilkins.
Cité pages 129 et 130

Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M. et Hanrahan, P. (2004). Brook for GPUs: stream computing on graphics hardware. *Dans ACM Transactions on Graphics*, volume 23, page 777.
Cité page 24

Burghaus, R., Coboeken, K., Gaub, T., Kuepfer, L., Sensse, A., Siegmund, H.-U., Weiss, W., Mueck, W. et Lippert, J. (2011). Evaluation of the efficacy and safety of rivaroxaban using a computer model for blood coagulation. *PloS One*, 6(4):e17626.
Cité page 117

Burrage, K., Burrage, P. M., Jeffrey, S. J., Pickett, T., Sidje, R. B. et Tian, T. (2003). A grid implementation of chemical kinetic simulation methods in genetic regulation. *Dans Proceedings of the APAC Conference and Exhibition on Advanced Computing, Grid Applications and eResearch*, pages 1–8, Royal Pines Resort, Gold Coast.
Cité pages 15, 38 et 39

Byrne, H. et Drasdo, D. (2009). Individual-based and continuum models of growing cell populations: a comparison. *Journal of Mathematical Biology*, 58(4-5):657–87.
Cité pages 5 et 11

Caron-Lormier, G., Humphry, R. W., Bohan, D. A., Hawes, C. et Thorbek, P. (2008). Asynchronous and synchronous updating in individual-based models. *Ecological Modelling*, 212(3-4):522–527.
Cité page 49

Cornforth, D., Green, D. G. et Newth, D. (2005). Ordered asynchronous processes in multi-agent systems. *Physica D: Nonlinear Phenomena*, 204(1-2):70–82.
Cité page 49

Cournède, P.-H., Guyard, T., Bayol, B., Griffon, S., de Coligny, F., Borianne, P., Jaeger, M. et de Reffye, P. (2009). A forest growth simulator based on functional-structural modelling of individual trees. *Dans Third International Symposium on Plant Growth Modeling, Simulation, Visualization and Applications*, pages 34–41. IEEE.
Cité page 23

Courtois, P. J., Heymans, F. et Parnas, D. L. (1971). Concurrent control with “readers” and “writers”. *Communications of the ACM*, 14(10):667–668.
Cité page 55

Crépin, L., Harrouet, F., Kerdélo, S., Tisseau, J. et Redou, P. (2013). Computational methods for the parallel 3D simulation of biochemical kinetics at the microscopic scale. *Dans Bioinformatics Research and Applications*, volume 7875 de *Lecture Notes in Computer Science*, pages 28–39. Springer.
Cité page 86

Davie, E. W. et Ratnoff, O. D. (1964). Waterfall sequence for intrinsic blood clotting. *Science*, 145(3638):1310–1312.
Cité page 89

Dematté, L. (2012). Smoldyn on graphics processing units: massively parallel brownian dynamics simulations. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(3):655–67.
Cité pages 34 et 76

Desmeulles, G. (2006). *Réification des interactions pour l'expérience in virtuo de systèmes biologiques multi-modèles.* Thèse de doctorat, Université de Bretagne Occidentale.
Cité page 11

Desmeulles, G., Bonneaud, S., Redou, P., Rodin, V. et Tisseau, J. (2009). In virtuo experiments based on the multi-interaction system framework: the RéISCOP meta-model. *Computer Modeling in Engineering and Sciences*, 47(3):299–330.
Cité page 50

Dijkstra, E. W. (1965). Solution of a problem in concurrent programming control. *Communications of the ACM*, 8(9):569.
Cité page 55

dos Santos, R. W., Plank, G., Bauer, S. et Vigmond, E. J. (2004). Parallel multigrid preconditioner for the cardiac bidomain model. *IEEE Transactions on Bio-medical Engineering*, 51(11):1960–8.
Cité pages 15, 37 et 39

D'Souza, R. M., Lysenko, M., Marino, S. et Kirschner, D. (2009). Data-parallel algorithms for agent-based model simulation of tuberculosis on graphics processing units. *Dans Proceedings of the 2009 Spring Simulation Multiconference*, page 21, San Diego, California. Society for Computer Simulation International.
Cité pages 5, 34 et 35

Duboz, R., Ramat, É. et Preux, P. (2003). Scale transfer modeling: Using emergent computation for coupling an ordinary differential equation system with a reactive agent model. *Systems Analysis Modelling Simulation*, 43(6):793–814.
Cité page 13

Efrati, P., Sakal, E. et Nir, N. (1977). Microinterferometric measurement of dry weight of blood lymphocytes in hematologically normal patients and in patients with chronic lymphocytic leukemia. *Acta Cytologica*, 21(4):539–42.
Cité page 132

El Karoui, N., Peng, S. et Quenez, M. C. (1997). Backward stochastic differential equations in finance. *Mathematical Finance*, 7(1):1–71.
Cité page 7

Ermentrout, G. B. et Edelstein-Keshet, L. (1993). Cellular automata approaches to biological modeling. *Journal of Theoretical Biology*, 160(1):97–133.
Cité page 8

Fang, J., Varbanescu, A. L. et Sips, H. (2011). A comprehensive performance comparison of CUDA and OpenCL. *Dans 2011 International Conference on Parallel Processing*, pages 216–225, Taipei City. IEEE.
Cité page 25

Fidjeland, A. K., Roesch, E. B., Shanahan, M. P. et Luk, W. (2009). NeMo: a platform for neural modelling of spiking neurons using GPUs. *Dans 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 137–144, Boston, MA. IEEE.
Cité pages 32 et 35

Gardner, M. (1970). Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223(4):120–123.
Cité page 8

Garipcan, B., Maenz, S., Pham, T., Settmacher, U., Jandt, K. D., Zanow, J. et Bossert, J. (2011). Image analysis of endothelial microstructure and endothelial cell dimensions of human arteries - a preliminary study. *Advanced Engineering Materials*, 13(1-2):B54–B57.
Cité page 129

Geist, A. (1994). *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing*. MIT press.
Cité page 37

Gillespie, D. T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434.
Cité page 76

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361.
Cité page 76

Gray, K. (2003). *Microsoft DirectX 9 programmable graphics pipeline*. Microsoft Press.
Cité page 24

Hairer, E., Nørsett, S. et Wanner, G. (2009). *Solving ordinary differential equations I. Nonstiff problems*. Springer.
Cité page 7

Hairer, E. et Wanner, G. (2010). *Solving ordinary differential equations II. Stiff and differential-algebraic problems*. Springer.
Cité page 7

Hampton, S., Agarwal, P. K., Alam, S. R. et Crozier, P. S. (2010). Towards microsecond biological molecular dynamics simulations on hybrid processors. *Dans 2010 International Conference on High Performance Computing & Simulation*, pages 98–107, Caen, France. IEEE.
Cité page 43

Harris, M. (2003). *Real-time cloud simulation and rendering*. Thèse de doctorat, University of North Carolina.
Cité page 24

Harrouet, F. (2000). *oRis : s'immerger par le langage pour le prototypage d'univers virtuels à base d'entités autonomes*. Thèse de doctorat, Université de Bretagne Occidentale.
Cité page 50

Harrouet, F. (2012). Designing a multicore and multiprocessor individual-based simulation engine. *IEEE Micro*, 32(1):54–65.
Cité pages 18, 51 et 53

Hellweger, F. L. et Bucci, V. (2009). A bunch of tiny individuals - individual-based modeling for microbes. *Ecological Modelling*, 220(1):8–22.
Cité pages 5 et 11

Hemker, H. C. et Béguin, S. (1995). Thrombin generation in plasma: its assessment via the endogenous thrombin potential. *Thrombosis and Haemostasis*, 74(1):134–8.
Cité page 111

Hemker, H. C., Giesen, P., Al Dieri, R., Regnault, V., de Smedt, E., Wagenvoord, R., Lecompte, T. et Béguin, S. (2003). Calibrated automated thrombin generation measurement in clotting plasma. *Pathophysiology of Haemostasis and Thrombosis*, 33(1):4–15.
Cité page 111

Hennessy, J. L. et Patterson, D. A. (2011). *Computer architecture: a quantitative approach*. Morgan Kaufmann.
Cité pages 17 et 18

Henri, V. (1903). *Lois générales de l'action des diastases*. Librairie Scientifique A. Hermann.
Cité page 96

Hogea, C. S., Murray, B. T. et Sethian, J. A. (2006). Simulating complex tumor dynamics from avascular to vascular growth using a general level-set method. *Journal of Mathematical Biology*, 53(1):86–134.
Cité page 7

IEEE (1995). IEEE std 1003.1 c-1995, threads extensions. Rapport technique, IEEE Portable Applications Standards Committee and others.
Cité page 19

Intel (2012). Intel threading building blocks - reference manual. Rapport technique, Intel.
Cité page 19

Islam, M. et Alias, N. (2010). A case study: 2D vs 3D parallel differential equation toward tumor cell detection on multi-core parallel computing atmosphere. *Journal for the Advancement of Science and Arts*, 1(1):25–34.
Cité page 23

Isloor, S. et Marsland, T. (1980). The deadlock problem: An overview. *Computer*, 13(9):58–78.
Cité page 56

Jeschke, M. et Uhrmacher, A. M. (2008). Multi-resolution spatial simulation for molecular crowding. *Dans 2008 Winter Simulation Conference*, pages 1384–1392, Austin, Texas, USA. IEEE.
Cité page 14

Johansson, P. I., Stissing, T., Bochsen, L. et Ostrowski, S. R. (2009). Thrombelastography and tromboelastometry in assessing coagulopathy in trauma. *Scandinavian Journal of Trauma, Resuscitation and Emergency Medicine*, 17:45.
Cité page 111

Karimi, K., Dickson, N. G. et Hamze, F. (2010). A performance comparison of CUDA and OpenCL. *CoRR*, abs/1005.2.
Cité page 25

Kaufman, M., Urbain, J. et Thomas, R. (1985). Towards a logical analysis of the immune response. *Journal of Theoretical Biology*, 114(4):527–561.
Cité page 8

Kerdélo, S. (2006). *Méthodes informatiques pour l'expérimentation in virtuo de la cinétique biochimique. Application à la coagulation du sang.* Thèse de doctorat, Université de Rennes 1.
Cité pages 51, 77, 80, 81 et 82

Khronos (2012). The OpenCL specification. Rapport technique, Khronos OpenCL Working Group.
Cité page 25

Khurana, I. (2005). *Textbook of medical physiology.* Elsevier India Pvt. Limited.
Cité pages 129, 130, 131 et 132

Kiem, J., Borberg, H., Iyengar, G. V., Kasperek, K., Siegers, M., Feinendegen, L. E. et Gross, R. (1979). Elemental composition of platelets. Part II. Water content of normal human platelets and measurements of their concentrations of Cu, Fe, K, and Zn by neutron activation analysis. *Clinical Chemistry*, 25(5):705–10.
Cité page 132

Klann, M., Ganguly, A. et Koeppl, H. (2012). Hybrid spatial Gillespie and particle tracking simulation. *Bioinformatics*, 28(18):i549–i555.
Cité page 13

Lamport, L. (1974). A new solution of Dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8):453–455.
Cité page 55

Lee, V. W., Hammarlund, P., Singhal, R., Dubey, P., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A. D., Satish, N., Smelyanskiy, M. et Chennupaty, S. (2010). Debunking the 100X GPU vs. CPU myth. *Dans Proceedings of the 37th annual International Symposium on Computer Architecture*, volume 38, pages 451–460, Saint-Malo, France. ACM Press.
Cité page 35

Lenuzza, N. (2009). *Modélisation de la réplication des Prions : implication de la dépendance en taille des agrégats de PrP et de l'hétérogénéité des populations cellulaires.* Thèse de doctorat, École Centrale Paris.
Cité page 8

Li, H. et Petzold, L. (2007). Stochastic simulation of biochemical systems on the graphics processing unit. *Bioinformatics*.
Cité pages 32 et 35

Luo, C.-h. et Rudy, Y. (1994). A dynamic model of the cardiac ventricular action potential. I. Simulations of ionic currents and concentration changes. *Circulation Research*, 74(6):1071–1096.
Cité page 7

MacCullough, D. (1996). *Metapopulations and wildlife conservation*. Island Press.
Cité page 10

Macfarlane, R. et Biggs, R. (1953). A thrombin generation test: the application in haemophilia and thrombocytopenia. *Journal of Clinical Pathology*, 6(1):3–8.
Cité page 111

Macfarlane, R. G. (1964). An enzyme cascade in the blood clotting mechanism, and its function as a biochemical amplifier. *Nature*, 202(4931):498–499.
Cité page 89

Manninen, T., Linne, M.-L. et Ruohonen, K. (2006). Developing Itô stochastic differential equation models for neuronal signal transduction pathways. *Computational Biology and Chemistry*, 30(4):280–91.
Cité page 7

Marder, V. J., Aird, W. C., Bennett, J. S., Schulman, S. et White, G. C. (2012). *Hemostasis and thrombosis: basic principles and clinical practice*. Lippincott Williams & Wilkins.
Cité page 92

Mark, W. R., Glanville, R. S., Akeley, K. et Kilgard, M. J. (2003). Cg: a system for programming graphics hardware in a C-like language. *Dans Proceedings of ACM SIGGRAPH 2003*, volume 22, page 896, San Diego, California, USA. ACM Press.
Cité page 24

Massaioli, F., Castiglione, F. et Bernaschi, M. (2005). OpenMP parallelization of agent-based models. *Parallel Computing*, 31(10-12):1066–1081.
Cité page 23

McCool, M., Du Toit, S., Popa, T., Chan, B. et Moule, K. (2004). Shader algebra. *ACM Transactions on Graphics*, 23(3):787.
Cité page 24

Merelli, E., Armano, G., Cannata, N., Corradini, F., D'Inverno, M., Doms, A., Lord, P., Martin, A., Milanesi, L., Möller, S., Schroeder, M. et Luck, M. (2007). Agents in bioinformatics, computational and systems biology. *Briefings in Bioinformatics*, 8(1):45–59.
Cité page 11

Michaelis, L. et Menten, M. L. (1913). Die Kinetik der Invertinwirkung. *Biochemische Zeitschrift*, 49:333–369.
Cité page 96

Michel, F., Ferber, J. et Gutknecht, O. (2001). Generic simulation tools based on MAS organization. *Dans Proceedings of the 10th European Workshop on Modelling Autonomous Agents in a Multi Agent World*, volume 1.
Cité page 50

Miller, C. B., Lynch, D. R., Carlotti, F., Gentleman, W. et Lewis, C. V. W. (1998). Coupling of an individual-based population dynamic model of calanus finmarchicus to a circulation model for the georges bank region. *Fisheries Oceanography*, 7(3-4):219–234.
Cité page 14

Mysliwski, A. et Korczak, A. (1986). Increase of size and dry mass of human erythrocytes depending on age of donors. *Mechanisms of Ageing and Development*, 34(2):111–115.
Cité page 132

Newburn, C. J., So, B., Liu, Z., McCool, M., Ghuloum, A., Toit, S. D., Wang, Z. G., Du, Z. H., Chen, Y., Wu, G., Guo, P., Liu, Z. et Zhang, D. (2011). Intel's array building blocks: A retargetable, dynamic compiler and embedded language. *Dans International Symposium on Code Generation and Optimization*, pages 224–235, Chamonix, France. IEEE.
Cité page 25

Nugala, V., Allan, S. J. et Haefner, J. W. (1998). Parallel implementations of individual-based models in biology: bulletin- and non-bulletin-board approaches. *Biosystems*, 45(2):87–97.
Cité pages 5, 11, 38 et 39

NVIDIA (2012a). NVIDIA CUDA C programming guide. Rapport technique, NVIDIA.
Cité pages 25 et 66

NVIDIA (2012b). NVIDIA CUDA dynamic parallelism programming guide. Rapport technique, NVIDIA.
Cité page 30

OpenGL (2012). The OpenGL shading language. Rapport technique, Open Graphics Library.
Cité page 24

OpenMP (2013). OpenMP application program interface. Rapport technique, OpenMP Architecture Review Board.
Cité page 20

Orcutt, G. (1957). A new type of socio-economic system. *The Review of Economics and Statistics*, 39(2):116–123.
Cité page 12

Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E. et Purcell, T. J. (2007). A survey of general-purpose computation on graphics hardware. *Dans Computer Graphics Forum*, volume 26, pages 80–113.
Cité page 35

Padua, D. (2011). *Encyclopedia of parallel computing*. Springer.
Cité pages 15, 16, 36, 53, 56 et 57

Patankar, S. (1980). *Numerical heat transfer and fluid flow*. Taylor & Francis Group.
Cité page 93

Perrin, F. (1934). Mouvement Brownien d'un ellipsoïde (I). Dispersion diélectrique pour des molécules ellipsoidales. *Journal de Physique et le Radium*, 5(10):497–511.
Cité page 80

Perrin, F. (1936). Mouvement Brownien d'un ellipsoïde (II). Rotation libre et dépolarisation des fluorescences. Translation et diffusion de molécules ellipsoïdales. *Journal de Physique et le Radium*, 7(1):1–11.
Cité page 80

Plimpton, S. et Slepoy, A. (2003). ChemCell: a particle-based model of protein chemistry and diffusion in microbial cells. Rapport technique, Sandia National Laboratories.
Cité page 76

Purich, D. L. et Allison, R. D. (2000). *Handbook of biochemical kinetics: a guide to dynamic processes in the molecular life sciences*. Academic Press.
Cité pages 60, 76 et 81

Querrec, G. (2005). *Simulation des phénomènes complexes en oncologie : application aux myélomes multiples*. Thèse de doctorat, Université de Bretagne Occidentale.
Cité page 11

Quesnel, G., Duboz, R. et Ramat, É. (2009). The virtual laboratory environment - an operational framework for multi-modelling, simulation and analysis of complex dynamical systems. *Simulation Modelling Practice and Theory*, 17(4):641–653.
Cité page 14

Quick, A. J., Stanley-Brown, M. et Bancroft, F. (1935). A study of the coagulation defect in hemophilia and in jaundice.*. *The American Journal of the Medical Sciences*, 190(4):501–510.
Cité page 111

Redou, P., Gaubert, L., Desmeulles, G., Béal, P.-A., Le Gal, C. et Rodin, V. (2010). Absolute stability of chaotic asynchronous multi-interactions schemes for solving ODE. *Computer Modeling in Engineering & Sciences*, 70(1):11–40.
Cité page 50

Redou, P., Kerdélo, S., Le Gal, C., Querrec, G., Rodin, V., Abgrall, J.-F. et Tisseau, J. (2005). Reaction-agents: first mathematical validation of a multi-agent system for dynamical biochemical kinetics. *Dans Progress in Artificial Intelligence*, volume 3808, pages 156–166. Springer.
Cité page 50

Resat, H., Costa, M. N. et Shankaran, H. (2011). Spatial aspects in biological system simulations. *Dans Methods in Enzymology*, volume 487, pages 485–511. Academic Press.
Cité page 76

Rhoades, R. A. et Bell, D. R. (2012). *Medical physiology: principles for clinical medicine - 4e*. Lippincott Williams & Wilkins.
Cité pages 130, 131 et 132

Rocha, B. M., Campos, F. O., Plank, G., dos Santos, R. W., Liebmann, M. et Haase, G. (2010). Simulations of the electrical activity in the heart with graphic processing units. *Dans Parallel Processing and Applied Mathematics*, volume 6067, pages 439–448. Springer.
Cité pages 32 et 35

Rodin, V., Desmeulles, G., Ballet, P., Redou, P. et Le Gal, C. (2011). An efficient algorithm based on weak synchronization for distributed in virtuo biological experiments. *Multiagent and Grid Systems*, 7(4-5):159–182.
Cité pages 15 et 39

Salomon, D. et Motta, G. (2010). *Handbook of data compression*. Springer.
Cité page 109

Schönfisch, B. et de Roos, A. (1999). Synchronous and asynchronous updating in cellular automata. *Biosystems*, 51(3):123–143.
Cité page 8

Shen, W., Wei, D., Xu, W., Zhu, X. et Yuan, S. (2010). Parallelized computation for computer simulation of electrocardiograms using personal computers with multi-core CPU and general-purpose GPU. *Computer Methods and Programs in Biomedicine*, 100(1):87–96.
Cité page 43

Sherwood, L. (2010). *Human physiology: from cells to systems - 7e*. Brooks.
Cité pages 130, 131 et 132

Shima, M., Thachil, J., Nair, S. C. et Srivastava, A. (2013). Towards standardization of the clot waveform analysis and recommendations for its clinical applications. *Journal of Thrombosis and Haemostasis*, 11(7):1417–1420.
Cité page 111

Sleeman, B. D. et Levine, H. A. (2001). Partial differential equations of chemotaxis and angiogenesis. *Mathematical Methods in the Applied Sciences*, 24(6):405–426.
Cité page 60

Snir, M., Otto, S. W., Walker, D. W., Dongarra, J. et Huss-Lederman, S. (1995). *MPI: the complete reference*. MIT Press.
Cité page 37

Stiles, J. R. et Bartol, T. M. (2001). Monte carlo methods for simulating realistic synaptic microphysiology using MCell. *Dans Computational neuroscience: realistic modeling for experimentalists*, pages 87–127. CRC Press.
Cité page 76

Stoer, J., Bulirsch, R., Bartels, R., Gautschi, W. et Witzgall, C. (2002). *Introduction to numerical analysis*. Springer.
Cité page 7

Stundzia, A. B. et Lumsden, C. J. (1996). Stochastic simulation of coupled Reaction-Diffusion processes. *Journal of Computational Physics*, 127(1):196–207.
Cité page 76

Tisseau, J. (2001). *Réalité virtuelle : autonomie in virtuo.* Habilitation à diriger des recherches, Université de Rennes 1.
Cité page 10

Tolle, D. P. et Le Novere, N. (2006). Particle-based stochastic simulation in systems biology. *Current Bioinformatics*, 1(3):315–320.
Cité page 76

Tripodi, S., Ballet, P. et Rodin, V. (2012). GPU implementation and performance analysis of reactive agents having division and mobility capacities. *Multiagent and Grid Systems*, 8(4):289–309.
Cité page 34

Turing, A. (1990). The chemical basis of morphogenesis. *Bulletin of Mathematical Biology*, 52(1-2):153–197.
Cité page 60

van Kampen, N. (2007). *Stochastic processes in physics and chemistry.* North Holland.
Cité pages 9 et 76

van Zon, J. S. et ten Wolde, P. R. (2005). Green's-function reaction dynamics: a particle-based approach for simulating biochemical networks in time and space. *The Journal of Chemical Physics*, 123(23):234910.
Cité page 76

Versteeg, H. K. et Malalasekera, W. (2007). *An introduction to computational fluid dynamics. The finite volume method.* Pearson Education Limited.
Cité pages 7, 92 et 93

Von Neumann, J. et Burks, A. W. (1966). *Theory of self-reproducing automata.* University of Illinois Press - Urbana and London.
Cité page 8

Vuduc, R., Chandramowlishwaran, A., Choi, J., Guney, M. et Shringarpure, A. (2010). On the limits of GPU acceleration. *Dans Proceedings of the 2nd USENIX Conference on Hot Topics in Parallelism*, page 13.
Cité page 35

Wang, W., Wang, J. et Kim, M.-S. (2001). An algebraic condition for the separation of two ellipsoids. *Computer Aided Geometric Design*, 18(6):531–539.
Cité page 80

Wendt, J. F. et Anderson, J. D. (2009). *Computational fluid dynamics.* Springer.
Cité page 60

Wooldridge, M. (2009). *An introduction to multiagent systems - 2e.* Wiley.
Cité pages 10, 11 et 46

Zeigler, B. P., Praehofer, H. et Kim, T. G. (2000). *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems.* Academic press.
Cité page 47

Zomaya, A. Y. (2006). *Parallel computing for bioinformatics and computational biology: models, enabling technologies, and case studies*, volume 55. John Wiley & Sons.
Cité pages 15 et 41

Résumé

De nombreux types d'expérimentation existent pour étudier et comprendre les systèmes biologiques. Dans ces travaux, nous nous intéressons à la simulation *in silico*, c'est-à-dire à la simulation numérique de modèles informatiques ou mathématiques sur un ordinateur. Les systèmes biologiques sont composés d'entités, à la fois nombreuses et variées, en interaction les unes avec les autres. Ainsi, ils peuvent être modélisés par l'intermédiaire de deux approches complémentaires : l'approche population-centrée qui propose d'étudier le système à l'échelle des populations, c'est-à-dire d'un ensemble d'individus, et l'approche individu-centrée qui propose d'étudier le comportement de chacun de ces individus, plutôt que celui de la population globale. Face à la multitude et à la variété des phénomènes composant les systèmes biologiques, il nous semble pertinent de coupler ces deux approches pour obtenir une modélisation mixte, afin que chaque phénomène puisse être modélisé de manière adéquate.

En outre, en raison de la quantité conséquente d'informations que représente l'ensemble des entités et des interactions à modéliser, la simulation numérique des systèmes biologiques est particulièrement coûteuse en temps de calcul informatique. Ainsi, dans ce mémoire, nous proposons des solutions techniques de parallélisation permettant d'exploiter au mieux les performances offertes par les architectures multicœur et multiprocesseur et les architectures graphiques pour la simulation de systèmes biologiques à base de modélisations mixtes. Pour cela, nous présentons des algorithmes informatiques qui permettent de simuler des modèles individu-centrés et des modèles population-centrés en respectant au maximum les contraintes imposées par chacune des deux architectures utilisées.

Nous appliquons nos travaux à la simulation de phénomènes liés à la coagulation du sang. La première application consiste à simuler la cinétique biochimique à l'échelle microscopique et est basée sur notre modèle individu-centré parallélisé sur des architectures multicœur et multiprocesseur. La seconde consiste à étudier la formation d'un caillot sanguin au sein d'un vaisseau sanguin virtuel. Plus complète et plus complexe que notre première application, elle fait intervenir un modèle individu-centré et deux modèles population-centrés parallélisés sur architectures multicœur et multiprocesseur et sur architectures graphiques. Ces deux applications nous permettent d'évaluer les performances offertes par les solutions techniques de parallélisation que nous proposons, ainsi que leur pertinence dans le cadre de la simulation des systèmes biologiques.

Mots clefs : simulation des systèmes biologiques, parallélisation, couplage multi-modèles, coagulation du sang.

— Résumé —

Couplage de modèles population et individu-centrés pour la simulation parallélisée des systèmes biologiques. Application à la coagulation du sang.

Plusieurs types d'expérimentation existent pour étudier et comprendre les systèmes biologiques. Dans ces travaux, nous nous intéressons à la simulation *in silico*, c'est-à-dire à la simulation numérique de modèles sur un ordinateur. Les systèmes biologiques sont composés d'entités, à la fois nombreuses et variées, en interaction les unes avec les autres. Ainsi, ils peuvent être modélisés par l'intermédiaire de deux approches complémentaires : l'approche population-centrée et l'approche individu-centrée. Face à la multitude et à la variété des phénomènes composant les systèmes biologiques, il nous semble pertinent de coupler ces deux approches pour obtenir une modélisation mixte. En outre, en raison de la quantité conséquente d'informations que représente l'ensemble des entités et des interactions à modéliser, la simulation numérique des systèmes biologiques est particulièrement coûteuse en temps de calcul informatique. Ainsi, dans ce mémoire, nous proposons des solutions techniques de parallélisation permettant d'exploiter au mieux les performances offertes par les architectures multicœur et multiprocesseur et les architectures graphiques pour la simulation de systèmes biologiques à base de modélisations mixtes. Nous appliquons nos travaux au domaine de la coagulation du sang et plus particulièrement à l'étude de la cinétique biochimique à l'échelle microscopique ainsi qu'à la simulation d'un vaisseau sanguin virtuel. Ces deux applications nous permettent d'évaluer les performances offertes par les solutions techniques de parallélisation que nous proposons, ainsi que leur pertinence dans le cadre de la simulation des systèmes biologiques.

Mots clefs : simulation des systèmes biologiques, parallélisation, couplage multi-modèles, coagulation du sang.

— Abstract —

Population and individual-based model coupling for the parallel simulation of biological systems. Application to blood coagulation.

Several types of experimentation exist to study and understand biological systems. In this document, we take an interest in *in silico* simulation, i.e. numerical simulation of models on a computer. Biological systems are made of many various entities, interacting with each other. Therefore, they can be modeled by two complementary approaches: the population-based approach and the individual-based one. Because of the multitude and diversity of the phenomena constituting biological systems, we find the coupling of these two approaches relevant to provide a hybrid modelisation. Moreover, because of the huge quantity of data that the entities and interactions represent, numerical simulation of biological systems is especially computationaly intensive. This is why, in this document, we propose parallel computing methods to take advantage of the performances offered by multicore and multiprocessor architectures and by graphical ones for the simulation of biological systems using hybrid modelisations. We apply our work to blood coagulation and especially to the study of biochemical kinetics at the microscopic scale and the simulation of a virtual blood vessel. These two applications enable us to assess both the performances obtained by the parallel computing methods we proposed and their relevance for biological systems simulation.

Keywords: biological systems simulation, parallel computing, multi-model coupling, blood coagulation.