HAL
open science

Méthodes In-Situ et In-Transit : vers un continuum entre les applications interactives et offline à grande échelle.

Matthieu Dreher

▶ To cite this version:

Matthieu Dreher. Méthodes In-Situ et In-Transit : vers un continuum entre les applications interactives et offline à grande échelle.. Calcul parallèle, distribué et partagé [cs.DC]. Collège des Ecoles Doctorales de l'Université Grenoble Alpes, 2015. Français. NNT : . tel-01358477

HAL Id: tel-01358477
https://inria.hal.science/tel-01358477v1

Submitted on 31 Aug 2016

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 GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 7 aout 2006

Présentée par

Matthieu Dreher

Thèse dirigée par **Bruno Raffin**

préparée au sein **INRIA - Laboratoire d'Informatique de Grenoble (LIG)** et de **Mathématiques, Sciences et Technologies de l'Information, Informatique**

Méthodes In-Situ et In-Transit : vers un continuum entre les applications interactives et offline à grande échelle.

Thèse soutenue publiquement le **25 février 2015**,
devant le jury composé de :

M. Gabriel Antoniu
DR INRIA, Rapporteur
M. Tom Peterka
Assistant Computer Scientist Argonne, Rapporteur
M. Marc Baaden
DR CNRS, Examinateur
M. Stephane Redon
CR1 INRIA, Examinateur
M. Christian Perez
DR INRIA, Examinateur
M. Bruno Raffin
CR1 INRIA, Directeur de thèse

Remerciements

Je voudrais commencer par remercier mes rapporteurs : Gabriel Antoniu et Tom Peterka, ainsi que les autres membres du jury : Stephane Redon, Marc Baaden et Christian Perez d'avoir accepté de faire parti de ce jury et de prendre le temps d'évaluer ma thèse. Un merci tout particulier à Tom qui a eu la tâche difficile de lire ma thèse en français.

Cette thèse n'aurait jamais été possible sans le soutien et l'encadrement apportés par mon directeur de thèse Bruno Raffin. Ces conseils ont été extrêmement précieux tout au long de ces trois ans et il a toujours su me remettre sur les rails lorsqu'il le fallait. Je me sens très chanceux d'avoir eu un directeur de thèse aussi compétent et humain.

Un grand merci à l'ensemble des membres du projet ANR ExaViz/FvNano : MiCkael Trellet, Jessica Jonquet, Abderrahim Ait Wakrime, Nicolas Ferey, Sébastien Limet et Sophie Robert. Nous avons eu des discussions très intéressantes tout au long du projet. C'était un vrai plaisir de travailler avec vous sur des sujets très divers et variés.

Je voudrais également remercier mes deux collègues et compères de bureau pendant un long moment : Xavier Martin et Jeremy Jaussaud. On a eu un an et demi formidable à travailler ensemble sur FlowVR dans une ambiance mémorable.

Merci aux ingénieurs systèmes : Pierre Neyron, Elodie Bertoncello, Maxime Boutserin et Bruno Bzeznik avec qui j'ai eu l'occasion de travailler et qui m'ont permis de mener à bien mes projets dans les meilleures conditions possibles.

J'ai eu la chance d'être entouré au cours de cette thèse non pas par une mais deux équipes : MOAIS et MESCAL, très vivantes et conviviales !

J'ai une pensée également pour ma famille qui m'a soutenu pendant ces trois ans loin de ma région et plus particulièrement à ma mère et ma sœur qui ont fait le déplacement pour venir assister à ma soutenance. Une pensée également pour mes amis Alsaciens qui ont fait le voyage. Cela signifie beaucoup pour moi.

Au cours de cette thèse, j'ai également eu l'occasion de rencontrer des collègues et amis que cela soit à l'INRIA ou ailleurs : Generoso, Monica, Youenn, Alexis, Alexandre, Erick, Cristian, Gabriel, toute la troupe du babyfoot et bien d'autres encore. Merci d'avoir rendu ces années inoubliables.

Enfin, je souhaiterais remercier les personnes qui m'ont poussé à faire une thèse et donné l'opportunité de le faire : Solène, Denis, Fabrice, Guillaume, Christophe, Francis et Olivier. C'est un choix qui a changé ma vie et je vous en suis infiniment reconnaissant.

Table des matières

1 Introduction **1**

2 État de l'art **13**
2.1 Présentation des systèmes in-situ 13
2.1.1 Traitements in-situ synchrones 13
2.1.2 Traitements in-situ asynchrones 15
2.1.3 Traitements asynchrones sur noeuds dédiés (in-transit) 20
2.1.4 Infrastructures hybrides in-situ et in-transit 22
2.1.5 Services orientés données pour les applications in-situ 23
2.2 Exemples de calculs interactifs 24

3 Analyses in-situ avec FlowVR et Gromacs : présentation des outils **27**
3.1 FlowVR 27
3.1.1 Historique de FlowVR 27
3.1.2 Description d'un module 28
3.1.3 Le modèle de dataflow 29
3.1.4 Fonctionnement du démon 34
3.1.5 Gestion des messages 37
3.1.6 Placement contraint de la mémoire et des processus 38
3.1.7 Forces de FlowVR pour des traitements in-situ 39
3.2 Gromacs 40
3.2.1 Simulation de dynamique moléculaire 40
3.2.2 Description interne de Gromacs 42
3.3 Méthode de couplage à Gromacs 44
3.3.1 Enjeux 44
3.3.2 Utilisation des méthodes maître/escalve de Gromacs 45
3.3.3 Instrumentation de chaque processus MPI 46

4 Application interactive in-situ : simulation de dynamique moléculaire intéractive **49**
4.1 Contexte 49
4.2 Application biologique : le canal FepA 50
4.3 États des systèmes IMD actuels 52
4.4 Modification du code de Gromacs 53
4.5 Gestion des communications en entrée et en sortie de la simulation . . 54
4.6 Modulation de notre impact sur les performances de la simulation grâce au filtrage in-situ 56

4.7 Expérimentations et résultats 57
4.8 Résultats d'expérience sur le FepA 60
4.9 Discussion 61
4.9.1 Utilité et validité des systèmes IMD à grande échelle 61
4.9.2 Intérêts et limitations de l'interactif 62

5 Applications In-Situ à grande échelle 65
5.1 Introduction 65
5.2 Contribution 67
5.2.1 Adaptation de la fréquence d'extraction de données 67
5.2.2 Plateforme d'expérimentation 67
5.2.3 Exemple 1 : écriture de trajectoires en in-situ 68
5.2.4 Exemple 2 : génération d'un maillage en in-situ et in-transit . 71
5.3 Discussion 76
5.3.1 Besoin de flexibilité dans le placement des traitements in-situ . 76
5.3.2 Difficultés du partage des ressources 78
5.3.3 Passage de l'application IMD au contexte In-Situ 79

6 Usages dans le contexte de la dynamique moléculaire 81
6.1 Applications biologiques : le canal FepA et le canal du GLIC 81
6.2 Contribution 82
6.2.1 Intégration de modules d'analyses 82
6.2.2 Application à l'écriture de trajectoires multiples 84
6.2.3 Des simulations interactives vers les simulations de production 87
6.2.4 Des simulations longues aux analyses post-mortem et inversement 90
6.3 Discussion 93
6.3.1 Extension vers des pipelines d'analyse plus complexes 93
6.3.2 Besoin de stockage des données pour les traitements in-situ . . 94
6.3.3 Avantages et limites d'un environnement unifié 95

7 Conclusion 97
7.1 Bilan des travaux 97
7.2 La place de l'in-situ à l'ère des machines petascales 99
7.3 ... et à l'ère des machines exascales 100

Chapitre 1

Introduction

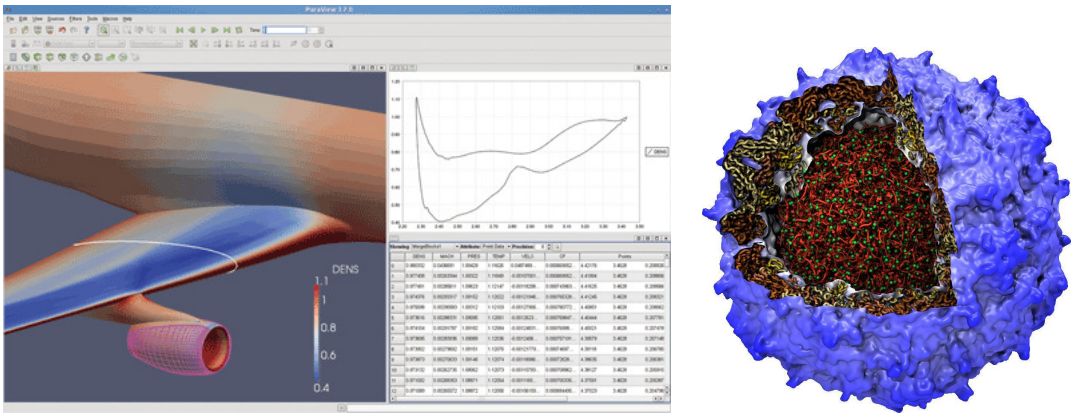
Les simulations informatiques sont devenues des outils couramment utilisés dans divers domaines scientifiques : dynamique des fluides, climatologie, biologie ou encore astrophysique. Une simulation vise à reproduire des phénomènes complexes à l'aide de modèles abstraits. À partir de l'état initial d'un système numérique, une simulation résout des séries d'équations définies par un modèle pour faire évoluer l'état du système. Si les modèles utilisés sont suffisamment précis, le système devrait évoluer de la même façon qu'un système réel. Les motivations pour utiliser de telles simulations sont multiples. Une des premières motivations est la difficulté voire l'impossibilité de contrôler ou reproduire le phénomène réel. Ceci est notamment le cas en climatologie où l'on ne peut qu'observer les différents phénomènes. Une autre raison est économique : faire une expérimentation réelle peut s'avérer coûteux. Enfin, les simulations peuvent également servir à des fins exploratoires en permettant de simuler diverses configurations du modèle initial ou de paramètres.

Au fur et à mesure des années, les systèmes numériques utilisés par les scientifiques sont devenus de plus en plus imposants. Ceci s'explique par plusieurs facteurs : les modèles numériques deviennent de plus en plus réalistes, de plus en plus de données expérimentales sont produites ou encore simplement les besoins évoluent. Pour pouvoir simuler ces systèmes complexes de grandes tailles, une importante puissance de calcul est nécessaire. Ces simulations sont donc généralement lancées dans des grands centres de calcul sur de larges machines parallèles.

Le Top500 répertorie deux fois par an les 500 supercalculateurs les plus puissants au monde. Ce classement est établi sur la base du benchmark LINPACK[31] testant principalement les capacités de calcul flottant de ces machines. En juin 2014, 37 supercalculateurs peuvent atteindre la barre symbolique du petaflops [1]. La machine actuellement la plus puissante au monde, le Tianhe-2 (MilkyWay-2), culmine à une vitesse de 33.8 petaflops répartie sur 3 120 000 cœurs. On estime l'arrivée des premières machines exascales [2] à l'horizon 2020.

Malgré la puissance offerte par de telles machines, les temps d'exécution des simulations de production peuvent durer plusieurs jours voire plusieurs semaines. Pendant cette période, des données sont sauvegardées de manière périodique par les simulations. Ces données représentent un état partiel du système simulé. Une fois la simulation terminée, ces données permettent aux chercheurs de pouvoir "relire"

1. 10^{15} opérations flottantes à la seconde.
2. Une machine exascale peut effectuer 10^{18} opérations flottantes à la seconde.



(a) (b)

Figure 1.1 – (a) Exemple de visualisation possible avec Paraview. Profil de densité autour d'une aile d'avion avec le tableau des données associées.(b) Exemple de rendu possible avec VMD. Ici la forme extérieure de la molécule est représentée par une surface appelée Quicksurf. Cette représentation permet d'avoir une meilleur appréhension de la forme d'un complexe moléculaire par rapport à une représentation en tout atome. [4]

le déroulement de la simulation et de l'analyser. Pour des raisons de commodité, la phase d'analyse se déroule bien souvent dans le laboratoire des chercheurs. Ceci permet aux chercheurs de disposer de leurs données et d'échanger avec des collègues sans avoir à se soucier des contraintes d'accès inhérentes aux centres de calcul. Ceci nécessite toutefois de transférer les données du centre de calcul vers le laboratoire du scientifique.

L'analyse va d'une lecture directe des données de la simulation à des traitements avancés allant de la production de graphes 2D jusqu'à la production de visualisations 3D interactives. La visualisation scientifique est par exemple couramment utilisée dans de nombreux domaines scientifiques. Des outils comme Visit[8] et Paraview[10] ont été développés pour permettre la visualisation de nombreux types de données comme des maillages ou des grilles (Figure 1.1(a)). Ces outils sont particulièrement utilisés pour analyser des données issues de simulations de dynamique des fluides, de climatologie ou d'études de structures. D'autres outils peuvent être plus proches de certains domaines scientifiques comme VMD[42] qui se spécialise dans la visualisation de systèmes moléculaires. Des algorithmes de rendu spécifiques à ce domaine (Figure-1.1(b)) sont nécessaires pour permettre une visualisation à la fois performante et plus compréhensible pour l'utilisateur. En plus de la visualisation, ces outils permettent souvent d'effectuer des requêtes complexes associant à la fois la génération de graphes et d'autres analyses comme des statistiques.

4. Source Figure 1.1(a) : http ://www.kitware.com/media/html/ParaViewInAerodynamics.html Source Figure 1.1(b) : http ://www.ks.uiuc.edu/images/ofmonth/2012-05/qsurf.jpg

De manière générale, les travaux autour des simulations se déroulent en trois phases :

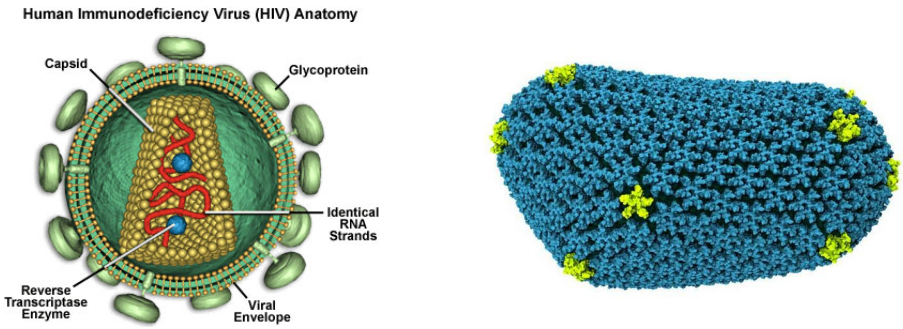
— Préparation. L'utilisateur prépare un modèle à simuler avec une série de paramètres. Des fichiers de configuration sont alors générés en vue de la simulation.
— Simulation. La simulation est lancée sur une machine parallèle pendant plusieurs heures, jours voire semaines et produit des données sauvegardées sur disque. Lorsque suffisamment de ressources sont disponibles sur la machine parallèle, l'ordonnanceur de taches de la machine lance la simulation de l'utilisateur. Dans cette phase, l'utilisateur n'a pas de moyens d'interaction avec la simulation.
— Post-traitement. L'utilisateur analyse les données générées par la simulation. Cette phase peut se faire soit sur la même machine parallèle hébergeant la simulation, soit sur un petit cluster local ou une station de travail dans le laboratoire de l'utilisateur si le transport des données est possible et s'il dispose des moyens de calcul nécessaires.

Ce cycle est un procédé bien souvent itératif. Suite à la phase d'analyse, de nouvelles hypothèses peuvent être formulées, des paramètres modifiés et de nouvelles simulations peuvent être nécessaires.

C'est par exemple le cas dans le domaine de la dynamique moléculaire. Les simulations de dynamique moléculaire modélisent le comportement de grands ensembles d'atomes. Mais avant de pouvoir modéliser ces grands ensembles, il faut déterminer leurs structures atomistiques. Il s'agit d'un travail long et complexe associant à la fois des manipulations expérimentales (cristallographie par rayon X) et simulations. La résolution de ces structures représente un travail de recherche à part entière. Il s'agit cependant d'une première étape indispensable pour l'étude de mécanismes biologiques. Les travaux d'Alex Tek autour du complexe SNARE ont par exemple permis d'étudier le mécanisme de fusion entre des membranes comme par exemple la fusion synaptique[78]. Cette étude a nécessité une thèse afin de construire les modèles atomistiques nécessaires et de les valider par simulation.

Les structures atomistiques sont également très utilisées pour étudier le comportement de virus et aider à la conception de traitements. Récemment, la structure atomistique complète du capside a été déterminée[92]. Le capside du virus VIH protège l'ADN du virus jusqu'à la phase de parasitage des cellules saines de l'hôte. Ce capside joue un rôle central dans la transmission du virus et pourrait être une des clés permettant l'élaboration d'un anti-viral contre le VIH. Le modèle complet du capside représente 64 millions d'atomes ce qui constitue l'un des plus gros modèles moléculaires à ce jour. Pour arriver à la construction de ce modèle, plusieurs dizaines de simulations ont été nécessaires sur la machine *BlueWaters* pour valider la

6. Source Figure 1.2(a) : http ://www.hivandhepatitis.com/2010_conference/croi/docs/0301_2010_a.html Source Figure 1.2(b) : http ://media.marketwire.com/attachments/201305/MOD-158571_All-atommodeloftheVIH-1capsid.jpg



(a) (b)

Figure 1.2 – (a) Schéma du virus VIH. Le capside protège l'ADN du virus jusqu'au moment du parasitage d'une cellule saine. (b) Représentation de la structure atomistique du capside. [6]

stabilité du système moléculaire.

La taille des systèmes simulés a beaucoup augmenté ces dernières années. Ceci est rendu possible notamment grâce à une augmentation considérable de la puissance de calcul disponible pour les scientifiques. Une conséquence de l'augmentation de la taille de ces simulations est l'augmentation de la quantité de données produites. En effet, plus un système simulé est imposant, plus il y a de données à sauvegarder. En 2010 déjà, une simulation de turbulence avec le code GTC[45] répartie sur 16384 cœurs produisait 260Go de données toutes les 120 secondes[95]. En 2012, une simulation de plasma (VPIC) répartie sur 120000 coeurs a produit environ 30 To de données en une seule itération de sauvegarde[22]. Pour chaque simulation nécessaire à la construction du capside du VIH, environ 50To de données ont été produites pour un total de 1Po. Entre chaque simulation, des analyses ont été effectuées sur les trajectoires produites pour formuler de nouvelles hypothèses sur la construction de ce modèle.

Cette augmentation de la production de données s'est révélée être un sérieux goulot d'étranglement de plus en plus important à la fois lors de la simulation et lors du post-traitement.

Lors de la simulation, de grands systèmes de fichiers parallèles sont connectés aux supercalculateurs pour pouvoir absorber cette quantité de données. Il y a toutefois une différence de plus en plus importante entre les capacités de calcul des supercalculateurs et les débits de lecture/écriture. Il y a deux générations ACSI Purple (2006), une machine de 100 teraflops disposait d'une bande passante de 140 Go/s vers son système de stockage. Une génération plus tard, JaguarPF (2009), une machine petaflop, disposait, elle, d'une bande passante de 200 Go/s. À l'ère de l'exascale, on estime que le débit des systèmes de fichiers parallèles sera de 60To/s[9]. Ce débit permettra d'absorber 0.00075% des données produites par une simulation exascale (8 Eo/s en double précision)[9, 60]. Evidemment, tous les calculs ne nécessitent pas

une sauvegarde. Toutefois, avec ces ordres de grandeurs, il semble raisonnable de considérer que moins de 1% des données produites pourront être sauvegardées.

Le stockage même de ces données devient de plus en plus problématique. Par exemple, les données stockées sur l'espace de production de la machine *BlueWaters*[1] ont une durée de vie autorisée de seulement 30 jours au maximum. Ceci est nécessaire pour éviter une surcharge trop importante du système Lustre de production. Si les utilisateurs souhaitent conserver plus longtemps leurs données, ils doivent transférer leurs fichiers sur l'espace de stockage longue durée avec des performances en lecture moindres.

La phase de post-traitement devient également de plus en plus complexe. Le volume de données produit rend très difficile voire impossible le transfert des données vers le laboratoire des chercheurs. Ceci implique que les analyses doivent de plus en plus se faire sur site. Par ailleurs, l'analyse de cette masse de données nécessite de plus en plus de bande-passante sur les entrées/sorties ainsi que de puissance de calcul. Des solutions comme DIY[65], PISTON[53], DAX[61] ou encore EAVL[58] par exemple donnent des briques de base pour faciliter la construction d'outils d'analyse passant à l'échelle. Des efforts restent toutefois encore à faire pour faire adopter de tels modèles à l'ensemble de la communauté scientifique.

Une méthode possible pour contrôler dans une certaine mesure ce phénomène peut être de limiter la quantité de données produites en jouant à la fois sur la fréquence d'écriture lors de la simulation et sur la quantité de données sauvegardées à chaque sauvegarde. Cette méthode est par exemple utilisée en dynamique moléculaire où seule une itération sur 5000 est sauvegardée. Cette méthode n'est toutefois pas pérenne pour deux raisons. La première est technique : la différence entre la puissance de calcul et les débits de stockage ne va pas cesser de croître. Par conséquent il faudra réduire de plus en plus les fréquences d'écriture. La deuxième raison est beaucoup plus importante : réduire la fréquence de sauvegarde est une approche trop naïve pour diminuer la quantité de données sans impacter de manière trop significative la qualité des données.

Il y a un véritable besoin d'une part de réduire la quantité de données produites de manière intelligente et d'autre part d'adresser le problème des analyses effectuées en post-traitement.

Face à ces problèmes, les traitements *in-situ* apparaissent comme une solution logicielle de plus en plus intéressante. Le principe fondamental de cette méthode est de traiter les données pendant l'exécution de la simulation alors qu'elles sont encore en mémoire. En pratique, des services supplémentaires sont couplés à la simulation pour effectuer des traitements sur les données qui seraient normalement effectués dans la phase de post-traitement. Ces traitements sont effectués à chaque itération de sauvegarde. Ces services sont hébergés sur les mêmes nœuds que la simulation et en partagent donc les ressources. L'avantage majeur de cette approche est double. Premièrement, les données sont directement accessibles en mémoire par les services ce qui permet d'éviter le goulot d'étranglement sur les entrées/sorties.

Deuxièmement, il est possible de profiter à la fois du niveau de parallélisme et de la localité des données. Des traitements peuvent donc être effectués avec un haut niveau de parallélisme. Ces traitements interviennent avant la phase d'écriture de la simulation et peuvent effectuer toute une variété d'opérations.

Une des premières motivations des méthodes in-situ est de permettre une réduction de données avant la phase d'écriture. Cette réduction peut prendre plusieurs formes comme du filtrage de données ou bien une transformation complète des données. Par exemple, pour certaines simulations il serait souhaitable d'adapter la fréquence d'écriture par rapport au déroulement de la simulation. En effet, il est peu intéressant d'écrire fréquemment des données si la simulation est relativement stable. En revanche, il devient pertinent d'écrire fréquemment lorsque des événements intéressants se produisent. Un tel service permettrait donc de considérablement réduire la quantité de données sauvegardées. Une autre application de plus en plus utilisée est la génération d'images au cours de la simulation. Ceci permet d'une part d'économiser plusieurs ordres de grandeur en taille de fichier et de proposer une forme très intuitive d'accès pour l'utilisateur aux données.

L'intégration de traitements in-situ permet toutefois d'aller bien au delà de la simple réduction de données. Tout d'abord, en extrayant des données d'une simulation en cours d'exécution, il est possible d'avoir une vue de l'état courant de la simulation. Traditionnellement, la simulation a très souvent été considérée comme une boîte noire lors de son exécution. Les méthodes in-situ permettent de casser cette barrière et ouvrent des portes pour bien d'autres applications.

Le monitoring est une application directe. En attachant des descripteurs à la simulation, il est possible de vérifier l'état de la simulation et générer des rapports. Les descripteurs peuvent prendre des formes très variées : visualisation, statistiques, analyses topologiques, etc. Ils peuvent par exemple servir à déterminer si la simulation converge vers un état désiré ou au contraire si elle diverge. Dans ce cas, le chercheur peut éventuellement décider de stopper prématurément la simulation et économiser du temps de calcul qui aurait été gaspillé.

Une autre application est héritée du *computationnal steering* : le chercheur a la possibilité de modifier des paramètres de la simulation en cours d'exécution. Ceci peut permettre par exemple d'essayer de faire converger une simulation qui serait en train de diverger ou d'accélérer la convergence d'une simulation.

En plus de gagner des informations sur le déroulement de la simulation, les traitements in-situ permettent également d'anticiper la phase de post-traitement. Ceci peut se faire sur plusieurs formes. Les données peuvent par exemple être préparées à la phase de post-traitement en les annotant ou en les indexant. Cela permettra dans les futurs post-traitements d'accélérer les temps d'accès aux données. Des sous-trajectoires se focalisant uniquement sur certains points d'une simulation peuvent également être extraites. Le fichier résultant serait alors beaucoup plus facilement manipulable.

Des traitements habituellement réalisés en post-traitement peuvent également se

prêter à une exécution in-situ. Comme décrit précédemment, les outils d'analyse ont de plus en plus de besoins en bande passante et en puissance de calcul. L'utilisation de machines parallèles dans un contexte in-situ adresse ces deux besoins.

Les traitements in-situ, quel que soit leurs applications, représentent un changement profond dans les méthodes de travail de la communauté scientifique. En effet, le contexte in-situ remet en cause la boucle préparation - simulation - analyses. Nous ne savons pas encore quels usages peuvent être portés vers un contexte in-situ et au delà de savoir si de nouveaux usages peuvent émerger. Des opérations comme la préparation de données sont des candidats évidents et directs à mettre en place comme traitements in-situ. Mais effectuer des traitements et analyses complexes comme ceux habituellement effectués en post-traitement représente des difficultés techniques et méthodologiques. Il semble clair que l'on ne pourra pas effectuer tous les traitements faits en post-traitements dans un contexte in-situ. La raison principale est qu'il est impossible de prévoir à l'avance l'ensemble des traitements que l'on souhaite faire sur une simulation. Une phase de sauvegarde sera toujours nécessaire. Il nous faut donc explorer ce qu'il est possible et raisonnable de faire dans un contexte in-situ et ce qui reste plus adapté au post-traitement.

Compte tenu des changements importants de méthodes de travail qu'impliquent les traitements in-situ, un certain nombre de principes doivent être considérés afin d'en faciliter leur adoption :

— Faible intrusion dans le code de simulation. Pour pouvoir utiliser des données de la simulation, il est nécessaire de modifier le code de la simulation. Ces codes sont complexes et une expertise du domaine est souvent nécessaire pour les comprendre. Un système in-situ devra nécessiter le moins possible de modifications afin de faciliter la connexion du système avec le plus de codes de simulation possibles.

— Faible impact sur les performances de la simulation. Les traitements in-situ permettent d'ajouter des services à la simulation. Ce faisant, il devient inévitable que ces services impactent négativement les performances de la simulation mais qu'il faut s'efforcer de les minimiser. Une convention souvent évoquée[90] est de ne pas dépasser 10% de pertes de performances pour une simulation. Dans certains cas, il est également intéressant de considérer le temps de découverte scientifique en cumulant le temps de simulation et le temps d'analyse pour arriver à une découverte. Ainsi, il est possible que la simulation soit plus lente mais que les traitements in-situ améliorent considérablement le temps d'analyse accélérant ainsi le temps global vers une découverte.

— Intégration des outils usuels d'analyse et de visualisation. Pour pouvoir faire adopter un tel système, il peut être utile de conserver au maximum les habitudes des utilisateurs. Un bon moyen pour y parvenir est d'attacher les outils d'analyse et de visualisation à la simulation. Il faut toutefois faire attention au fait que de nombreux outils ne sont pas adaptés pour un haut niveau de

parallélisme. Ces outils et algorithmes devraient être modifiés voire repensés pour ce nouveau contexte d'utilisation.
— Générique. L'objectif d'un tel système est de coupler différents codes hétérogènes (simulations, outils d'analyse et de visualisation) qui ne sont nativement pas compatibles entre eux. Une interface suffisamment générique doit être proposée pour permettre à ces codes de communiquer entre eux de manière simple.
— Robuste. La priorité étant le bon déroulement de la simulation, il ne faut pas qu'une erreur d'un des services in-situ cause l'arrêt de la simulation.

Ces principes forment une base souhaitable à atteindre pour un système in-situ mais ils posent un certain nombre d'enjeux techniques et pratiques.

L'impact que peut avoir un système in-situ sur les performances d'une simulation peut être particulièrement délicat à gérer. En effet, les codes de simulations sont bien souvent finement optimisés et la moindre perturbation dans leurs déroulements peut causer d'importantes pertes de performances. Les services in-situ vont créer à la fois de la contention sur les processeurs (caches) et les bancs mémoires hébergeant la simulation mais également sur le réseaux si ces services ont besoin de communiquer. Ces contentions peuvent mener à une dégradation significative des performances de la simulation. Il faut toutefois noter que les codes de simulation ne parviennent jamais à utiliser totalement l'ensemble des ressources d'une machine parallèle[96]. Il y a donc des ressources disponibles pour effectuer d'autres tâches et en particulier des traitements in-situ et améliorer ainsi l'usage global des ressources de calcul.

La synchronisation ou non de la simulation avec les services peut également avoir des répercutions importantes sur les performances de la simulation. Si ces services sont exécutés de manière synchrone, c'est-à-dire bloquante pour la simulation, alors le temps d'exécution de ces services est directement ajouté au temps total d'exécution de la simulation. Il s'agit d'une solution généralement simple à mettre en œuvre. LibSim (Visit)[87] et Catalyst (Paraview)[37] l'ont par exemple adopté. Dans un mode asynchrone, les services et la simulation s'exécutent de manière concurrente. Sa mise en œuvre est techniquement plus difficile et requiert généralement une copie des données. En effet, dans le cas contraire, la simulation peut modifier des données utilisées au même moment par les traitements in-situ. De plus, les services doivent avoir terminé avant que la simulation n'envoie à nouveau des données pour éviter tout débordement de mémoire.

L'intégration d'outils traditionnels peut également poser de nombreux challenges. Ces outils ne sont généralement pas conçus pour fonctionner sur des supercalculateurs. Les plate-formes cibles sont plutôt des machines de bureau ou des stations de travail[59, 85]. Ceci s'explique par le fait que les chercheurs n'ont généralement pas accès à de gros moyens de calcul dans leurs laboratoires lorsqu'ils effectuent leurs post-traitements. D'autre part, certains algorithmes sont difficilement parallélisables. C'est par exemple le cas des algorithmes qui ont besoin de l'intégralité des données d'une itération pour produire un résultat [50]. Or, pour pouvoir effectuer

des traitements in-situ de manière efficace, il est préférable d'utiliser des algorithmes distribués et qui passent à l'échelle au même niveau que la simulation. Des analyses inadaptées à ce niveau de parallélisme peuvent causer d'importantes contentions sur les ressources de la machine et dégrader significativement les performances de la simulation. Idéalement, ces algorithmes d'analyses seront réadaptés à ce nouveau contexte d'exécution. Toutefois le coût de développement peut être très important. Enfin, certains algorithmes sont développés en supposant l'accès à l'ensemble des données d'une simulation[88]. Or, pour des traitements in-situ, seules les données de l'itération courante voire quelques itérations précédentes sont accessibles. La question de la sauvegarde des données se pose alors pour ces algorithmes.

Au delà des aspects performances, il est également important de considérer les usages du contexte in-situ et ses implications sur la phase de post-traitement telle que nous la connaissons actuellement. Les post-traitements sont des suites d'opérations complexes sur les données qu'on appelle souvent un workflow d'analyse. Instancier ce workflow dans un contexte in-situ peut s'avérer difficile si l'infrastructure in-situ en place n'offre pas la flexibilité nécessaire. L'intégration d'outils usuels du domaine métier est un pas dans cette direction mais n'est pas suffisante. Des explorations sont nécessaires pour déterminer dans quelle mesure les infrastructures in-situ peuvent se rapprocher du workflow habituel des scientifiques et inversement comment les workflows de post-traitement peuvent s'adapter pour tirer bénéfice des infrastructures in-situ.

Pour réaliser ces rapprochements d'environnements, l'aide des utilisateurs scientifiques est indispensable. En effet, les traitements in-situ doivent refléter les méthodes de traitement des données des scientifiques. Il est donc nécessaire de les impliquer dans la conception des infrastructures in-situ afin de proposer des applications réalistes qui répondent à des problèmes et méthodes réels. Par ailleurs, ils sont également les plus à même de déterminer comment modifier leurs workflows habituels et éventuellement pointer les faiblesses actuelles des post-traitements dans leurs domaines respectifs.

Contributions. Depuis plusieurs années, différentes approches in-situ ont été proposées [32, 95, 87, 97, 30, 96]. Il s'agit d'un domaine en plein essor qui tend à se développer. Beaucoup d'efforts de recherche ont été effectués pour tenter de minimiser l'impact des traitements in-situ sur les performances de la simulation. Actuellement, il ne semble pas y avoir un paradigme particulier qui émerge comme dominant. Parallèlement à l'émergence de ces systèmes, assez peu de codes de simulation et d'outils d'analyse ont passé le cap des traitements in-situ. Une des premières explications possibles est le temps d'adaptation nécessaire pour développer et adapter les outils et codes de simulation usuels. Le choix du paradigme à adopter, notamment vis-à-vis du placement des traitements in-situ, peut également être un frein. En effet, il a été montré qu'aucune stratégie n'était optimale pour la totalité des applications in-situ [93]. Par conséquent, les utilisateurs peuvent attendre qu'un sys-

tème se dégage avant de se lancer dans des développements d'adaptation lourds. Une autre explication possible peut venir des possibilités d'expressivité des infrastructures in-situ, c'est-à-dire quelles analyses peuvent être intégrées facilement dans ces infrastructures. En effet, beaucoup d'efforts ont été déployés pour effectuer des traitements in-situ avec un faible coût mais la question des usages possibles avec ces infrastructures est restée plus en retrait.

Dans cette thèse, nous adressons ces trois points. Nous proposons une infrastructure suffisamment flexible pour proposer différentes stratégies de placement des traitements in-situ. Nous espérons ainsi pouvoir supporter un maximum de codes de simulation en adaptant les stratégies de placements et de traitements suivant les cas. Pour répondre au problème des usages, nous avons choisi d'adopter le paradigme du *dataflow*. Grâce à ce paradigme, des pipelines d'analyses complexes peuvent être exprimés de manière intuitive et proche des méthodes de traitement usuelles des scientifiques. Pour implémenter ce paradigme, nous avons réorienté l'intergiciel FlowVR originalement proposé pour des applications de réalité virtuelle intéractives. Celui ci nous permet de construire des applications in-situ asynchrones flexibles grâce à son modèle de programmation orienté composant. FlowVR nous permet également d'instrumenter des codes de simulation par un effort de développement modeste grâce à son API concise.

Notre application cible est le code de dynamique moléculaire Gromacs, un code de simulation couramment utilisé dans les grands centres de calcul et capable de passer à l'échelle sur plusieurs milliers de cœurs. Pour proposer des applications réalistes et répondant à des besoins concrets, nous avons collaboré avec des experts en biologie de l'Institut de Biologie Physico-Chimique de Paris disposant d'une forte expérience en simulation avec Gromacs et les analyses associées. Grâce à leur collaboration, nous pouvons proposer non seulement une infrastructure suffisamment flexible pour nous adapter aux contraintes liées à la dynamique moléculaire, mais également explorer les usages que peuvent avoir les biologistes des traitements in-situ. Notre objectif est à la fois de chercher quels sont les usages habituellement fait en post-traitement qui peuvent se faire en mode in-situ mais également d'explorer si de nouveaux usages peuvent émerger à l'aide des traitements in-situ. Pour motiver ces usages, nous avons étudié plusieurs problèmes biologiques pour le moment non résolus.

Notre première contribution s'articule autour du *computationnal steering*. L'objectif de notre application est de faire converger une simulation de dynamique moléculaire en permettant à un utilisateur de guider la simulation. Ce type d'application est généralement appelé *Interactive Molecular Dynamic*(IMD). Pour cela, une visualisation interactive ainsi qu'un système de bras haptique sont raccordés à la simulation. L'intégration de l'utilisateur dans une application pose des contraintes de performances sur la simulation et la visualisation. Il est important d'obtenir des fréquences interactives pour permettre une interaction fluide de l'utilisateur. Contrairement aux autres applications IMD du domaine, nous proposons un mo-

dèle d'extraction de données entièrement asynchrone au niveau de chaque processus de la simulation. Ceci nous permet d'une part de limiter considérablement notre impact sur les performances de la simulation mais également d'ajouter d'autres services comme du filtrage. Ce faible impact nous permet de supporter des simulations interactives de plus d'un million d'atomes sur plusieurs centaines de cœurs. Nous utilisons également notre application pour tenter de résoudre un problème réel de biologie moléculaire. Ces travaux ont donné lieu à une publication dans la conférence *International Conference on Computationnal Science 2013*(ICCS).

Notre deuxième contribution vise à étendre notre approche aux simulations longues de production sur plusieurs milliers de cœurs dans un mode in-situ. En utilisant toujours une méthode de couplage de codes parallèles entièrement asynchrone, nous proposons plusieurs scénarios de traitements in-situ allant jusqu'à 2048 cœurs. Nous démontrons en particulier que, grâce à différentes stratégies de placement des traitements in-situ, nous pouvons nous adapter à différents workflows d'analyes et réduire autant que possible notre impact sur la simulation. Notre premier scénario propose de remplacer le système d'écriture de Gromacs par deux méthodes in-situ. Notre approche nous permet de supporter des fréquences d'écriture nettement plus élevées que la méthode native de Gromacs. En particulier, nous avons pu réduire l'impact sur les performances de la simulation de l'écriture d'une trajectoire à une fréquence de 1100 sur 2048 cœurs de plus de 75% avec la méthode native de Gromacs à 9% avec notre méthode. Le deuxième scénario génère un maillage représentant la surface d'une molécule. Ce type de rendu est très souvent utilisé pour visualiser de grands ensembles moléculaires. Notre approche nous permet de tester différents placements des traitements sans avoir à modifier le code de génération du maillage pour déterminer la meilleure stratégie de placement. Ces travaux ont été publiés à la conférence *International Symposium on Cluster, Cloud and Grid Computing*(CCGRID).

Dans un troisième temps, nous proposons un *framework* autour de Gromacs permettant l'analyse de données issues soit de la simulation en in-situ, soit de trajectoires. Ce *framework* vise à créer un continuum entre la simulation et la phase de post-traitement. En plus de la visualisation, des outils usuels du domaine sont intégrés au *framework*. Ces outils ont été intégrés de telle sorte que les chercheurs puissent réutiliser leurs scripts avec seulement quelques modifications mineures. Ces outils sont utilisables que cela soit dans un contexte in-situ ou de post-traitement. Afin de préparer la phase d'analyse, nous proposons également diverses méthodes d'écriture de trajectoires. En particulier nous montrons l'intérêt d'écrire de multiples trajectoires à des fréquences et niveaux de filtrages différents. L'objectif est à la fois de diminuer le coût des phases d'écriture mais aussi de préparer des trajectoires plus légères et directement prêtes à être post-traitées.

Organisation du manuscrit. La suite du manuscrit est organisée comme suit. Le Chapitre 2 présente l'état de l'art des infrastructures in-situ. Nous présentons les

différents paradigmes jusque là proposés dans la littérature, notamment vis-à-vis du placement des traitements in-situ. Le Chapitre 3 présente en détail les deux principaux logiciels que nous utilisons dans cette thèse à savoir FlowVR et Gromacs. Nous décrivons en particulier pourquoi FlowVR est bien adapté aux applications in-situ, les challenges pour intégrer Gromacs efficacement dans une infrastructure in-situ et notre solution de couplage. Le Chapitre 4 présente nos travaux sur le guidage interactif d'un système moléculaire par un utilisateur au cours d'une simulation. Le Chapitre 5 étend les concepts présentés dans le chapitre précédent aux simulations longues de production sur les machines parallèles. Nous démontrons dans ce chapitre à la fois le faible coût de notre infrastructure sur les performances de la simulation ainsi que sa flexibilité pour construire des workflows d'analyses complexes. Le Chapitre 6 se focalise sur les usages des traitements in-situ dans le domaine de la biologie. Nous présentons différents scénarios pour résoudre des problèmes biologiques concrets. Nous démontrons également les capacités de notre infrastructure à s'adapter à de multiples contextes d'exécution. Nous terminons ce manuscrit par un bilan des travaux effectués ainsi que certaines considérations sur la place de l'in-situ actuellement et pour les années à venir.

Chapitre 2

État de l'art

2.1 Présentation des systèmes in-situ

De nombreuses infrastructures permettent d'effectuer des traitements in-situ sous des formes diverses. Nous proposons dans cette section un récapitulatif des différentes méthodes existantes ainsi que leurs implémentations. Nous avons séparé les différentes méthodes suivant le mode d'exécution (synchrone ou asynchrone) et les ressources utilisées pour effectuer les traitements (nœuds de simulation ou nœuds dédiés).

2.1.1 Traitements in-situ synchrones

Les traitements in-situ synchrones sont des traitements effectués sur les mêmes nœuds que la simulation et qui interrompent la simulation le temps d'effectuer les traitements. Cette méthode présente deux avantages majeurs. Tout d'abord, elle est relativement facile à mettre en place. Le code des traitements peut par exemple être directement intégré dans le code de la simulation. Le deuxième avantage est l'empreinte mémoire. La simulation étant interrompue, les traitements peuvent utiliser les structures de données de la simulation évitant ainsi des copies coûteuses en mémoire. Le principal défaut de cette méthode est son coût. En effet, cette méthode étant bloquante, le temps passé dans les traitements in-situ est directement additionné au temps de simulation. Par ailleurs, cette méthode ne peut pas profiter des temps où la simulation n'utilise pas complètement les ressources.

K. Liu Ma et al[90] proposent une des premières applications in-situ motivée par le goulot d'étranglement au niveau des I/O. L'objectif est de permettre la visualisation d'une simulation de combustion (S3D) en générant des images au cours de la simulation. Pour cela, un moteur de rendu volumique et de particules est intégré dans le code de la simulation. Pour chaque processus, une image partielle est générée à partir des particules locales en conservant notamment les informations de profondeur pour chaque pixel. Pour obtenir une image globale, une phase de recomposition d'image est effectuée avec l'algorithme 2-3 swap[91]. L'algorithme produit un résultat similaire à un MPI_REDUCE_SCATTER. Cette phase permet de reconstruire une image globale à partir des images locales. La reconstruction alterne les phases d'échanges d'images locales avec un schéma de communication en forme d'arbre et d'accumulation où des images sont fusionnées en tenant compte de la profondeur et de l'opacité des pixels des images locales. À la fin des étapes, chaque

processus dispose d'une partie de l'image globale. La phase de recomposition est de loin l'étape la plus coûteuse dans le processus de rendu (un ordre de grandeur plus long que les autres étapes). Le temps de rendu pour une image de 1024x1024 sur 15360 coeurs représente 8.75% du temps de la simulation et 36.74% pour une résolution de 2048x2048. Par rapport aux méthodes de visualisations traditionnelles pour la combustion, la méthode in-situ permet d'une part d'avoir une fréquence de visualisation beaucoup plus élevée et évite de nombreuses étapes de traitements des données normalement effectuées pour les visualiser.

Tu et al.[84] proposent une solution pour une simulation de tremblement de terre avec le framework Hercule. L'ensemble des composants liés à la simulation (maillage, partitionnement, solver, visualisation) est hébergé sur la même machine parallèle et se partage les mêmes structures sans fichiers intermédiaires. Ce système entièrement intégré tend à éviter toutes les I/Os et ne produit en résultat que des séries d'images au format JPEG. L'ensemble des opérations étant synchrone, le temps total de simulation inclut le temps d'exécution de tous les traitements et en particulier du rendu.

Visit[8] et Paraview[10] sont deux environnements de visualisations très repandus dans la communauté scientifique. Ils sont construits au-dessus de la bibliothèque de visualisation scientifique VTK[71]. Chacun de ces deux outils permet de se connecter à une simulation pour effectuer de la visualisation et des analyses in-situ.

Visit est constitué de deux entités : un client léger permettant de construite un pipeline de visualisation et d'interagir avec les données et enfin des serveurs chargés d'exécuter les filtres demandés sur de grandes machines parallèles. Une connexion est créée entre le client et les serveurs pour transmettre les commandes de l'utilisateur comme la définition des filtres, les changements de points de vue, etc. Les serveurs lisent les données en parallèle à partir de fichiers, les répartissent entre eux, et génèrent des images qui sont transmises au client. Pour passer en mode in-situ, les serveurs Visit sont intégrés dans les processus de la simulation (Figure 2.1). La bibliothèque Libsim[87] a été proposée pour créer l'interface entre la simulation et les serveurs Visit. Libsim remplit deux rôles : créer l'interface pour convertir les données de la simulation au format Visit et gérer les événements Visit (connexion, modification de paramètres, boucle d'événements). Si les données de la simulation ne sont pas compatibles avec le format VTK, elles sont alors copiées et transmises à Visit afin d'être converties. Les expériences sur la simulation de cosmologie GADGET-2[75] avec 100 millions de particules sur 512 cœurs montrent que le temps de génération d'un graphique Pseudocolor est de deux ordres de grandeur plus rapide que l'écriture collective d'un fichier (2.8Go) et un ordre de grandeur plus rapide que l'écriture d'un fichier par processus.

Paraview utilise une architecture semblable à Visit avec un client léger pour visualiser des données et piloter des serveurs parallèles distants. Pour passer en mode in-situ, les serveurs sont également intégrés dans la simulation et la bibliothèque

2. Source Figure 2.1 : [87]

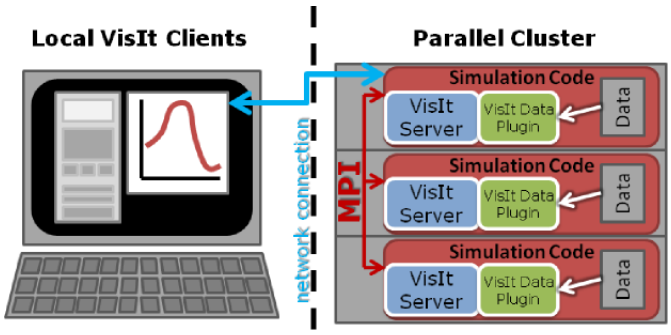


FIGURE 2.1 – Organisation de Visit en in-situ. Les serveurs Visit sont intégrés dans le code de simulation. Un client léger permet de se connecter à la simulation. [2]

Catalyst [37] est proposée pour faire l'interface entre la simulation et Paraview. Un mécanisme d'adaptateur est défini pour convertir les données de la simulation en un format compatible VTK. Le pipeline de traitement est ensuite effectué à l'intérieur de la simulation. Lorendeau et al[56] proposent une évaluation du coût de ce système pour le Code_Saturne[15], une simulation de dynamique des fluides. Pour une taille de modèle de 51 millions d'hexaèdres et des traitements in-situ lourds (rendu volumique, glyphes), le temps de simulation est augmenté de 20 à 30% en utilisant de 720 à 3200 cœurs et l'empreinte mémoire est augmentée d'un facteur 2 à 3. Cette consommation mémoire s'explique à la fois par la nécessité de copier les données de la simulation avant de les transmettre à l'adaptateur ainsi que par les besoins en mémoire propres à chaque filtre VTK pour effectuer les traitements.

2.1.2 Traitements in-situ asynchrones

Les traitements in-situ asynchrones sont exécutés sur les mêmes nœuds que la simulation en parallèle de celle ci. Les traitements in-situ et la simulation sont alors hébergés dans des processus séparés. En principe cette méthode permet de réduire l'impact sur les performances de la simulation en permettant des traitements in-situ non bloquants. En pratique, effectuer des traitements de manière concurrente à la simulation peut être difficile à mettre en place. Tout d'abord, il est nécessaire que le traitement d'une itération soit terminé avant la suivante pour éviter des problèmes de débordement. Ensuite, il est nécessaire de copier les données de la simulation. En effet, la simulation est susceptible de modifier les données transmises aux traitements in-situ alors que ceux-ci sont encore en cours de traitement. Un mécanisme de transfert des données des processus de la simulation vers les processus in-situ est également nécessaire. Cela peut se faire soit par socket, soit par mémoire partagée qui est généralement la solution choisie. Enfin, comme la simulation et les traitements in-situ sont hébergés sur les mêmes ressources de calcul, il peut être difficile de contrôler les interférences systèmes provoquées par les traitements in-situ. Des contentions sur l'ordonnanceur, le cache et la mémoire peuvent avoir lieu

impactant parfois de manière significative les performances de la simulation.

Approches multiples avec ADIOS

ADIOS[55] est une plateforme de recherche destinée à explorer différentes techniques d'I/O. Bien qu'ADIOS ne soit pas lié à une méthode d'I/O spécifique, nous le présentons ici car il est la base de plusieurs systèmes in-situ qui seront présentés plus loin. Cette infrastructure est née de deux constats. D'abord, les utilisateurs n'ont pas besoin de savoir comment sont organisées les données sur disque tant qu'il est possible d'écrire et de relire les données. Ensuite, les utilisateurs scientifiques ne devraient pas avoir besoin de s'occuper des optimisations nécessaires pour les I/O pour chaque plateforme. Partant de ce constat, ADIOS a été développé selon trois principes :

— La méthode d'I/O est abstraite. Une API générique de haut niveau est présentée à l'utilisateur mais elle n'est pas liée à une méthode d'I/O particulière. La méthode d'écriture la plus appropriée est choisie par l'utilisateur dans un fichier xml séparé de l'application pour une plateforme spécifique.

— Architecture orientée service. L'API utilisateur est stable et ne nécessite aucune instruction particulière pour une méthode donnée. En revanche, il est possible de créer de nouvelles méthodes d'écriture pour des besoins particuliers. Ceci permet de tester et comparer facilement les méthodes d'écriture entre elles.

— Support pour le traitement des données. Il est souvent nécessaire de traiter les données avant de les écrire. Pour cela, ADIOS utilise des nœuds dédiés. Les données sont transférées de manière asynchrone vers ces nœuds pour limiter les temps d'attente nécessaires aux traitements des données.

Nous allons ici présenter les aspects clés de ADIOS. Plus de détails concernant notamment les techniques d'optimisations d'I/O utilisées dans ADIOS peuvent être trouvés dans [51].

API de lecture/écriture. L'API d'écriture de ADIOS se veut aussi proche que possible de l'API POSIX. On retrouve des primitives pour ouvrir et fermer un fichier, écrire des variables, etc (voir Listing 2.1). Le seul ajout est la notion de groupe utilisée pour optimiser les mécanismes de *buffering*. Plusieurs variables peuvent être attachées à un même groupe et chaque groupe peut être associé à une ou plusieurs méthodes d'I/O différentes. L'API rend très facile l'écriture de tableaux, structure que l'on retrouve dans l'ensemble des codes de simulation tout en maintenant le support des autres types de variables (scalaire, entier, string, etc...). Chaque variable, attribut ou groupe est associé à un nom défini dans un xml externe (décrit plus loin).

Listing 2.1 – Exemple de code pour écrire des variables avec l'API d'ADIOS

```
adios_open(&fd, "analysis",filename, "w", &comm);
```

```
adios_group_size(fd, groupsize, total);
adios_write(fd, "NX", &NX);
adios_write(fd, "NY", &NY);
adios_write(fd, "temperature", t);
adios_close(fd);
```

Configuration par XML extérieur. L'ensemble des variables ainsi que les groupes sont définis dans un fichier xml. Des informations comme le type et la taille des données ainsi que les méthodes d'I/O sont données. Il est donc nécessaire que les noms de variables utilisés dans le code de la simulation soient cohérents avec le contenu du XML. Cette séparation permet entre autre à l'utilisateur de sélectionner des méthodes d'I/O à l'extérieur de l'application pour chaque groupe sans avoir besoin de recompiler l'application. Le XML est analysé au lancement de l'application pour déterminer les méthodes d'I/O à utiliser.

Méthodes d'I/O disponibles. ADIOS propose trois schémas d'écriture de fichier : un fichier global écrit collectivement par N processus, un fichier par processus et enfin un fichier pour M processus (agrégation). Pour chaque stratégie d'écriture, plusieurs techniques d'écriture peuvent être utilisées. L'écriture d'un fichier global par exemple peut utiliser MPI-IO[79], Parallel NetCDF[5, 3] ou Parallel HDF5[80, 81]. Le schéma 1 fichier par processus est disponible via la méthode POSIX qui génère des fichiers individuels ainsi qu'un fichier contenant les métadonnées. La méthode qui passe le mieux à l'échelle agrège les données sur un sous ensemble de processus puis chaque processus écrit un fichier séparé. D'autres infrastructures construites au-dessus de ADIOS et spécialisées pour effectuer des traitements in-situ ont été proposées. Elles seront présentées dans les sections suivantes.

Approche par cœur dédié

L'approche par cœur dédié consiste à réserver un ou des cœurs par nœud pour effectuer des traitements in-situ. Cette approche peut sembler coûteuse car le retrait d'un cœur normalement alloué à la simulation représente une perte en ressource de calcul. Cette stratégie peut néanmoins se justifier car les simulations passent rarement à l'échelle linéairement avec le nombre de cœurs. La perte de performance liée au retrait d'un cœur est donc bien souvent inférieure à la perte de ressources de calcul alloué. Par exemple, pour le cas de GTS, il a été montré que retirer un cœur par processeur sur des nœuds équipés de 4 processeurs quadricoeur (soit 25% des ressources) ne diminuait les performances que de 2.7% à 4096 cœurs[97].

Damaris[32] permet de s'insérer à l'intérieur d'un code parallèle MPI et de réserver des processus pour effectuer des traitements in-situ (un ou plusieurs cœurs par nœud). Pour cela, le communicateur global de la simulation est séparé en deux communicateurs : un pour Damaris qui ne regroupe que les cœurs dédiés et le deuxième

qui remplace le communicateur global de la simulation. Le code de la simulation est instrumenté avec une API de haut niveau similaire à celle d' ADIOS. Les données de la simulation sont écrites dans un segment de mémoire partagée géré par Damaris. Enfin, les utilisateurs peuvent signaler un événement nommé. Généralement, ces signaux sont placés juste après l'écriture de données. Un fichier xml externe est utilisé pour configurer Damaris. Ce fichier xml permet entre autre de décrire les données transmises par la simulation et d'associer un événement à une politique de traitement et une fonction de traitement. La politique de traitement peut être d'exécuter la fonction de traitement soit quand : 1) l'événement est reçu, 2) tous les événements de même nom d'un nœud pour une itération ont été reçus, 3) l'ensemble des processus de la simulation ont déclenché l'événement. La fonction de traitement est définie via une bibliothèque contenant les fonctions apportées par l'utilisateur. Les traitements peuvent avoir accès à l'ensemble des données écrites par l'utilisateur lors de l'itération courante.

Damaris a également été étendu pour supporter Visit comme méthode de visualisation scientifique[33]. Le XML de description des données est étendu pour permettre de décrire la structure des données graphiques (type de maillage, dimensions, etc). Une fois les données transmises à Damaris, la bibliothèque in-situ de Visit(Libsim) est appelée sur les cœurs dédiés dans le contexte Damaris. Ainsi les calculs de rendu habituellement effectués de manière synchrone dans la simulation sont exécutés de manière asynchrone.

Functionnal Partitioning[49] implémente également le concept de cœur dédié et l'applique aux opérations d'I/O et de visualisation en utilisant une interface FUSE. GePSea[72] implémente également ce concept et l'applique aux I/Os collectives. Cette interface nécessite néanmoins de multiples copies des données augmentant ainsi l'empreinte mémoire.

Approche avec concurrence sur les ressources

Bien que plus rares, certains systèmes permettent d'exécuter des traitements in-situ asynchrones mais sans utiliser de ressources dédiés. Les traitements sont exécutés sur les mêmes ressources que la simulation.

HDF5/DMS[73] utilise le mécanisme de driver de HDF5 pour intercepter les données écrites par une simulation. À la place d'être écrites sur fichiers, les données sont stockées en mémoire dans le driver HDF5 et peuvent être lues par une autre application. En particulier, ce mécanisme permet de coupler un code de simulation écrivant les données en HDF5 avec des outils d'analyse. Les données peuvent être lues soit sur les mêmes nœuds que la simulation, soit sur des nœuds distants, le driver s'occupant de transférer les données entre les processus. Cette interface a été utilisée notamment pour le plugin ICARUS[17] de Paraview. Ce plugin permet d'exécuter les calculs de rendu de manière asynchrone par rapport à la simulation.

Utiliser les mêmes ressources que la simulation peut provoquer des interférences avec la simulation difficiles à contrôler. En effet, les contentions sur l'ordonnanceur

ainsi que les trois niveaux de cache du processeur peuvent impacter négativement les performances de la simulation significativement.

Traditionnellement, les simulations sont déployées de la manière suivante : un processus MPI est placé sur chaque processeur puis OpenMP est utilisé pour utiliser l'ensemble des cœurs des processeurs. Cependant, des sections séquentielles sont présentes entre les parties OpenMP comme lors des phases de communication par exemple. Pendant ces phases, seul le cœur hébergeant le processus MPI est utilisé. Partant de ce constat, Goldrush[96] propose d'ordonnancer des traitements in-situ pendant les périodes où OpenMP n'est pas utilisé par la simulation. Goldrush est l'un des systèmes intégrés dans l'infrastructure d'ADIOS.

Pour ordonnancer les traitements in-situ au bon moment, Goldrush utilise les signaux systèmes. SIGCONT est envoyé au début d'une période creuse (après une section OpenMP) pour déclencher un processus de traitement. Inversement, SIGSTOP est envoyé juste à la fin d'une période creuse (avant une section OpenMP). L'envoi des signaux ainsi que le réveil des processus in-situ a un certain coût. Si la période entre un SIGCONT et SIGSTOP est trop courte, il n'est pas forcément rentable de réveiller les traitements in-situ. Pour détecter les périodes propices, Goldrush maintient un historique des périodes creuses d'une simulation. En supposant que les périodes creuses se reproduisent régulièrement à chaque itération, il est possible de prévoir la durée d'une période creuse et ainsi décider si une période creuse sera suffisamment longue. D'après les expérimentations, une période creuse utile doit durer au minimum 1 ms. Goldrush monitore également les compteurs processeurs liés aux défauts de cache et au nombre d'instructions par cycle. Si des pertes de performances trop importantes sont détectées, les analyses sont ralenties par des appels à usleep().

Ce système est appliqué sur différentes simulations et benchmarks dont le code de simulation de dynamique moléculaire Gromacs. Différents benchmarks visant à stresser différentes parties d'une machine parallèle (calcul, cache, mémoire, réseau) sont proposés comme traitements in-situ. Une comparaison est faite entre une approche naïve (ordonnancement fait par le système avec une priorité faible sur les processus in-situ) et l'approche Goldrush. Les expérimentations montrent que l'approche Goldrush permet de considérablement réduire l'impact des traitements comparé à l'approche système. En moyenne, la sélection des périodes creuses permet d'améliorer les performances de 42% par rapport à la solution système. La détection des interférences (compteurs CPU) permet de réduire encore le coût des traitements in-situ. En moyenne, les traitements in-situ avec détection d'interférences ne rallongent la durée de simulation que de 1.7% sur l'ensemble des tests comparé à 14% pour la solution système. Enfin notons que, pour le cas de Gromacs, 98% des périodes creuses ne sont pas utilisables car trop courtes.

2.1.3 Traitements asynchrones sur noeuds dédiés (in-transit)

Comme décrit précédemment, l'utilisation des mêmes ressources que la simulation est difficile à gérer. En plus des problèmes liés à la contention sur les ressources, il est également difficile de produire des outils d'analyse passant à la même échelle que la simulation.

Une alternative est d'utiliser des nœuds dédiés pour exécuter les traitements in-situ. Cette approche a deux motivations principales : réduire de manière drastique les contentions sur les ressources, notamment les processeurs et la mémoire, et réduire également le niveau de parallélisme nécessaire pour les traitements in-situ. En contrepartie, les données produites par la simulation doivent être envoyées vers les nœuds dédiés ce qui peut représenter un flux de données important. Cette approche est par exemple utilisée par ADIOS pour écrire les données sur disque de manière asynchrone.

Méthodes de transfert vers les nœuds dédiés

Le transfert des données vers les nœuds dédiés peut causer des contentions significatives sur la carte réseau des nœuds de calcul. En effet, celle-ci étant utilisée de manière intensive par la simulation, les transferts vers les nœuds dédiés peuvent perturber les schémas de communication de la simulation.

DataTap est une bibliothèque légère permettant le transport asynchrone de données utilisée sur les nœuds de simulation. Il est disponible sous forme de méthode de transport dans l'infrastructure ADIOS. DataTap utilise les protocoles RDMA [3] des interfaces réseaux haute performance (Infiniband, Cray SeaStar, Cray Gemini, etc) et propose un modèle demande-lecture pour le transport des données. DataStager[7] est la partie située sur les nœuds dédiés. DataStager est responsable de lire les données à partir de DataTap et d'effectuer l'agrégation des données pour les traitements suivants à effectuer sur les nœuds dédiés. Lorsque des données sont disponibles via DataTap, seul un petit message est envoyé vers la partie DataStager qui va le placer dans une file d'attente. Une fois le message traité, DataStager fait une demande de lecture distance à DataTap et déclenche le transfert effectif des données. Avant d'effectuer ce transfert, deux vérifications sont faites. La première est de vérifier si suffisamment de mémoire est disponible sur les nœuds dédiés. Deuxièmement, DataStager vérifie si la simulation est en phase de calcul ou non. Lorsque la simulation est en phase de calcul, elle n'utilise pas ou peu le réseau. Effectuer les transferts vers les nœuds dédiés pendant les phases de calcul permet de réduire considérablement les perturbations envers la simulation.

Nessie (NEtwork Scalable Service Interface)[63, 54] est une autre infrastructure pour la création de services orientés données sous la forme de client/serveurs pa-

3. Remote Direct Memory Access

rallèles à destination des grandes systèmes HPC. En particulier Nessie propose 1) des méthodes de transfert asynchrone pour la plupart des interfaces réseaux haute performance ; 2) Gestion des buffers par un serveur afin de gérer au mieux la bande-passante entre les clients ; 3) Des canaux de communication séparés entre les messages de contrôle et les messages de données ; 4) Un encodage XDR pour les messages de contrôle afin de supporter des architectures hétérogènes entre les nœuds de calcul et les nœuds de service.

Mercury[74] implémente une interface RPC spécifiquement dédiée aux applications HPC nécessitant des traitements sur de larges quantités de données. Il permet d'exécuter des fonctions distantes en transférant efficacement les données à l'aide de protocoles RDMA. Une interface de communication abstraite permet d'étendre l'infrastructure à de multiples types de réseaux haute performance.

DART (Decoupled and Asynchronous Remote Transfers)[29] permet de transférer de grandes quantités de données via une approche client/serveur. DART se focalise sur la minimisation des surcoûts induits sur l'application par le transfert des données, obtenir un haut débit de transfert, une faible latence et prévenir les pertes de données. Pour atteindre cet objectif, DART est conçu pour que le client situé sur un nœud dédié extraie les données de la mémoire du nœud de calcul de manière asynchrone grâce à un protocole RDMA.

Infrastructures in-transit

PreData est une infrastructure permettant de traiter des données produites par une simulation de manière asynchrone sur des nœuds dédiés. L'infrastructure propose un mécanisme de plugins pour permettre la déclaration des traitements utilisateurs. Ces traitements peuvent être la réorganisation de données, du filtrage, de la réduction ou encore des analyses ou pré-analyses. PreData s'interface avec ADIOS pour récupérer les données produites par la simulation. Cette infrastructure bénéficie également des travaux autour de DataStager pour le transport des données.

Le pipeline de PreData est organisé en quatre étapes :

- Les données sont extraites de la simulation et sont éventuellement une première fois traitées sur les nœuds de calcul de manière synchrone.
- Une agrégation peut être demandée au niveau des nœuds dédiés
- Transfert asynchrone des données des nœuds de calcul vers les nœuds dédiés.
- Traitement des données sur les nœuds dédiés.

Le traitement des données (phase 4) est organisé selon un modèle dans l'esprit Map/Reduce en ajoutant notamment des étapes au début et à la fin du pipeline de traitement.

JITStager[6] est une extension de PreData. Il permet le traitement dynamique des données tout au long du pipeline d'I/O. En particulier, des transformations de données peuvent maintenant avoir lieu directement lors de l'extraction de données et au niveau des nœuds dédiés.

2.1.4 Infrastructures hybrides in-situ et in-transit

Les stratégies in-situ sur les nœuds de la simulation ainsi que les stratégies in-transit sur nœuds dédiés présentent toutes deux des avantages et inconvénients. Il est cependant clair qu'aucune de ces stratégies n'est la plus optimale pour l'ensemble des scénarios possibles. Le choix d'une stratégie est fonction de l'architecture de la machine parallèle hôte, de la simulation et également des traitements à appliquer. Pour proposer une infrastructure complète, il est donc nécessaire de supporter plusieurs stratégies de placement des analyses afin de s'adapter à un maximum de scénarios possibles. L'intérêt de cette flexibilité de placement a été étudié par Zheng et al. [94] qui proposent un modèle analytique pour évaluer le coût que peut avoir une stratégie de placement des traitements in-situ sur les performances d'une simulation.

Bennett et al. [16] proposent un environnement permettant d'effectuer dans un premier temps des opérations légères locales en in-situ puis de transférer les résultats partiels vers des nœuds dédiés. Les traitements in-situ sont effectués de manière synchrone par rapport à la simulation tandis que les traitements in-transit sont asynchrones. Les mouvements de données sont assurés par un serveur DART. Un effort particulier a été déployé pour décomposer plusieurs algorithmes en deux phases distinctes pour s'adapter au mieux au contexte in-situ (opérations courtes, utilisation des données locales uniquement, réduction de données) et au contexte in-transit (contraintes mémoires, agrégation des résultats partiels). Des exemples de modifications d'algorithmes sont proposés dans le domaine de la visualisation, de l'analyse topologique et des statistiques pour s'adapter à ce modèle.

FlexIO est[97] est un système construit au-dessus de ADIOS. Il profite donc notamment de l'API haut niveau de ADIOS pour extraire les données de la simulation. Les traitements peuvent être placés soit sur des cœurs dédiés et/ou des nœuds dédiés. La transmission des données vers les cœurs dédiés est assurée via un segment de mémoire partagée comme c'est le cas pour Damaris. La transmission des données vers les nœuds dédiés se fait quant à elle via les protocoles RDMÀ implémentés dans EVPATH[34]. Pour transmettre les données vers les nœuds dédiés, les données sont réparties équitablement entre les nœuds. La flexibilité offerte sur le placement des traitements in-situ se fait néanmoins au détriment de l'utilisateur qui doit choisir la stratégie à adopter. Pour aider dans ce choix, FlexIO propose des outils de monitoring en ligne et à base de traces pré existantes pour proposer une solution de placement adéquate pour une application donnée. Les placements proposés tiennent également compte de la topologie des nœuds et notamment des domaines NUMA. FlexIO profite également des codelets chargés dynamiquement pour appliquer des traitements légers sur les données tout au long du pipeline.

GLEAN[86] est une infrastructure permettant d'exécuter des traitements in-situ synchrones sur les nœuds de la simulation et/ou des traitements in-transit asynchrones sur des nœuds dédiés. Les données peuvent être extraites de la simulation soit de manière transparente (sans modification de code) via les API de PNetCDF et HDF5, soit de manière explicite via l'API de GLEAN. Les traitements in-situ

ne nécessitent pas de copies car GLEAN permet de s'adapter à la sémantique des données. Un exemple d'adaptation de GLEAN est donné avec la simulation de cosmologie FLASH[38] et sa structure de données AMR. Capturer la sémantique des données permet à GLEAN de transformer les données à la volée et/ou de transmettre cette sémantique aux analyses suivantes. Cette sémantique permet par exemple de passer facilement d'un format NetCDF à un format HDF5 ou encore de transmettre la structure d'un maillage vers Paraview.

2.1.5 Services orientés données pour les applications in-situ

La gestion et l'organisation des données en provenance d'une simulation peut être une tâche complexe et lourde. Lorsque les données sont extraites d'une simulation, les données sont organisées et équilibrées pour les besoins de la simulation. Cette répartition des données peut cependant ne pas être optimale pour de futures analyses. Des infrastructures comme FlexIO proposent des méthodes pour rééquilibrer la répartition des données lors du transfert vers des nœuds dédiés. Toutefois la sémantique des données n'est pas prise en compte, les tableaux de données sont simplement subdivisés équitablement entre les nœuds dédiés.

DataSpaces[30]/ActiveSpaces[28] proposent une abstraction d'un espace de mémoire distribué. DataSpaces adopte un modèle publie/souscrit pour permettre à des applications parallèles de déposer des données (simulation) dans l'espace unifié et à d'autres d'accéder aux données via des requêtes(analyses). Les données sont généralement hébergées sur des nœuds dédiés et sont transférées via des serveurs DART. Une fonctionnalité clé de DataSpaces est sa table de hachage distribuée. Lorsque des données sont insérées dans DataSpaces, elles sont indexées selon différentes méthodes en fonction de la sémantique des données. Par exemple, pour des données en provenance d'un domaine de décomposition, une courbe de Hilbert est utilisée pour générer les index des données et les répartir à travers les serveurs. Cette indexation permet par la suite à d'autres applications parallèles d'effectuer des requêtes selon des critères dans l'espace. Pour cela, un moteur de requête est hébergé sur les serveurs DataSpaces pour faire la conversion entre les requêtes et les données correspondantes. Chaque serveur dispose des meta-données globales. Lorsqu'une requête adressée à un serveur nécessite des données situées sur un autre nœud, la requête est transmise au nœud concerné. Un autre aspect important de DataSpaces est sa dynamicité. Des clients peuvent se connecter à n'importe quel moment à DataSpaces pour déposer ou extraire des données. Dans un contexte in-situ, cela signifie qu'un utilisateur peut, au cours de la simulation, lancer dynamiquement une analyse supplémentaire suite à la découverte de résultats préliminaires. En revanche, dans la mesure où DataSpaces héberge les données exclusivement en mémoire, seules les données actuelles sont accessibles.

FlexPath[26] utilise un modèle publie/souscrit pour connecter des codes parallèles entre eux. À la différence de DataSpaces, la simulation et les analyses associées

sont connectées directement entre eux sans l'intermédiaire de serveurs. Un flux de données peut être partagé soit par plusieurs producteurs, soit par plusieurs consommateurs. Si un flux de données n'a pas de consommateurs connectés, le flux est alors fermé. Ainsi, un consommateur peut exposer autant de données que nécessaire, il n'y aura pas de surcoût si les données ne sont pas demandées par la suite. Cette propriété permet également une résistance aux pannes car l'ensemble des composants sont entièrement découplés. Pour permettre ces interactions entre composants parallèles, FlexPath propose en interne des schémas de communications parallèles comme le *scatter-gather* ou encore le MxN. FlexPath hérite également de DataSpaces la possibilité pour un consommateur de spécifier un sous ensemble de données à extraire.

2.2 Exemples de calculs interactifs

Les traitements in-situ sont conceptuellement proches des travaux effectués dans le domaine du calcul interactif ou *computationnal steering*. Tout comme pour les traitements in-situ, une simulation est équipée de services permettant d'interagir avec elle généralement à l'aide d'une visualisation en ligne et d'une interface. L'utilisateur a alors la possibilité de modifier des paramètres de la simulation au cours de son exécution. Les principales différences entre de tels systèmes et des infrastructures in-situ sont la taille de la simulation ainsi que les objectifs. Les simulations interactives visent avant tout à offrir des services aux utilisateurs comme la visualisation en ligne au prix éventuellement d'un impact significatif sur les performances de la simulation. Les applications in-situ quant à elles visent à adresser le problème des I/O tout en réduisant autant que possible leur impact sur les performances de la simulation.

SCIRun[44] est un environnement intégré permettant d'interagir avec une simulation. Il adopte un modèle de dataflow similaire à celui de FlowVR et permet entre autre la réutilisation de composants. Les composants peuvent communiquer entre eux via des canaux définis à travers une interface utilisateur. Ainsi des composants comme une simulation, un visualiseur et une interface peuvent être intégrés au sein d'une même application pour permettre à la fois de visualiser l'état courant de la simulation et de pouvoir modifier des paramètres de celle-ci selon les besoins de l'utilisateur. SCIRun est cependant limité au cas des machines à mémoire partagée ce qui limite fortement la taille des simulations possible. SCIRun nécessite également que la simulation soit développée pour l'infrastructure de SCIRun avec une API spécifique. Par conséquent il est beaucoup plus difficile d'intégrer des codes de simulation préexistants.

EPSN[35] est un environnement capable de coupler des codes de simulation à un visualiseur parallèle. Pour cela, l'infrastructure adresse notamment le problème du schéma de communication en NxM nécessaire pour transférer les données de la simulation vers le moteur de rendu parallèle. Trois types d'interactions sont possibles

avec la simulation : 1) contrôler le passage des pas de temps de la simulation ; 2) l'accès et l'envoi de données vers la simulation ; 3) Actions déclenchant des fonctions définies par l'utilisateur dans la simulation. Pour permettre ces interactions, le code de simulation est instrumenté. Une API est également disponible pour instrumenter des codes de visualisation et faire la connexion avec la simulation. Pour chaque composant parallèle (simulation, visualisation), un proxy est placé pour gérer les connexions et requêtes utilisateurs avec un thread par nœud jouant le rôle de serveur. Les requêtes utilisateurs sont échangées entre les proxys qui transmettent les requêtes vers les serveurs respectifs. Une fois les requêtes traitées, la redistribution des données se fait directement entre les nœuds impliqués grâce à la bibliothèque RedGRID de manière partiellement asynchrone à la simulation.

Le projet RealityGrid[68] propose une bibliothèque destinée à faciliter la création de simulations interactives. Cette bibliothèque permet entre autre d'exposer des données à d'autres composants comme la visualisation et également d'éditer certaines de ces données. Des commandes usuelles sont également intégrées comme la possibilité de mettre en pause la simulation. Un client générique est également disponible afin de piloter une simulation et de visualiser les grandeurs extraites de la simulation au fil de l'eau. Cette bibliothèque ignore toutefois l'aspect parallèle de la simulation. Il est à la charge du développeur de fournir à la bibliothèque une vue globale des données et de transmettre les modifications de variables faites par l'utilisateur à tous les processus. Cette charge peut s'avérer très lourde en terme de développement et sans doute coûteuse en terme de performance.

Chapitre 3

Analyses in-situ avec FlowVR et Gromacs : présentation des outils

La première partie de ce chapitre présente FlowVR. Nous décrivons en particulier ses différentes fonctionnalités générales et insistons sur les aspects qui en font un *middleware* de choix pour l'in-situ. Les modifications effectuées pour passer à l'échelle et s'adapter au contexte du HPC seront détaillées. La deuxième partie de ce chapitre présente Gromacs, notre application. Dans un premier temps nous présentons les principes de base de la dynamique moléculaire ainsi que les algorithmes en jeu. Nous décrivons ensuite l'implémentation de Gromacs, ses principes ainsi que son organisation interne. Enfin nous décrivons comment Gromacs a été instrumenté avec FlowVR. Cette instrumentation restera la même dans toute la suite de ce document.

3.1 FlowVR

FlowVR est un intergiciel implémentant le paradigme de programmation par composants. FlowVR permet de faire du couplage de codes parallèles. Une application FlowVR est décrite comme un graphe où les nœuds représentent des opérations sur des données et les arêtes représentent des canaux de communication. Les nœuds de ce graphe sont appelés modules ou composants. Chaque nœud peut recevoir des données d'autres modules, les traiter, puis envoyer des données vers d'autres modules. Une application FlowVR peut être exécutée sur un nœud de calcul ou en distribué. La coordination des modules et la gestion des canaux de communication est à la charge d'un processus spécifique appelé *démon FlowVR*. Les canaux de communication peuvent passer soit par la mémoire partagée pour les communications intra nœuds, soit par le réseau pour des communications inter nœuds. Dans la suite de cette section nous décrivons brièvement l'historique de FlowVR puis nous décrivons plus précisément les différents éléments de l'intergiciel.

3.1.1 Historique de FlowVR

FlowVR a été initialement proposé pour construire des applications de réalité virtuelle distribuées[11]. Il peut être utilisé pour piloter des caméras[12], effectuer du rendu parallèle[13] ou encore coupler des codes de simulation hétérogènes[14].

FlowVR a été intensivement utilisé dans le cadre de la plateforme Grimage[66]. Celle-ci vise à reconstruire en temps réel le modèle d'une personne capturée par un ensemble de caméras. FlowVR permet de gérer le flux de données en provenance des caméras ainsi que le workflow de traitement nécessaire pour convertir les images des caméras en un maillage 3D. D'autres applications ont également été construites comme le projet Dalia qui vise à visualiser, interagir et faire collaborer des personnes distantes au sein d'un même environnement virtuel. Typiquement, ces applications utilisaient une centaine de cœurs.

3.1.2 Description d'un module

Un module est un processus ou un thread exécutant une boucle infinie. Un module dispose de ports d'entrée pour recevoir des messages d'autres modules et de ports de sortie pour envoyer des messages aux autres modules. À chaque itération, un module peut recevoir des données, les traiter, et envoyer à son tour des données.

Pour créer un module, une API minimaliste permet d'implémenter cette boucle. Elle est composée de trois fonctions principales :

- Wait() : Fonction bloquante qui rend la main lorsque tous les ports d'entrée du module ont reçu un message.
- Get() : Retourne un pointeur sur un message reçu sur un port donné. Si à une itération un Get() n'est pas appelé sur un port, alors le message en attente sera perdu et remplacé par un nouveau message à l'itération suivante.
- Put() : Permet d'envoyer un message sur un port de sortie. À chaque itération le développeur peut envoyer un message par port de sortie.

Pour plus de flexibilité, certains ports peuvent être déclarés comme non bloquants. Dans ce cas, la fonction Wait() n'attendra pas ce port. Cette fonctionnalité est généralement utilisée sur des ports recevant des messages événementiels. Par exemple, lorsqu'un utilisateur modifie un paramètre dans une interface graphique, il faut envoyer un message au module pour l'instruire du changement mais il est inutile de renvoyer cette information à chaque itération. Lorsqu'un utilisateur fait un Get() sur un port non bloquant, il n'y a aucune garantie qu'un message soit disponible sur ce port. Si aucun message n'est présent, alors Get() renvoie un message vide.

Cette API est complétée par des fonctions d'initialisation. Pour initialiser un module, le développeur doit dans un premier temps déclarer un module puis lui attacher des ports d'entrée et/ou de sortie. Cette initialisation permet au module et au démon de se synchroniser. Une fois le module initialisé, le développeur peut commencer la boucle infinie du module. Lorsque l'application FlowVR se termine, la fonction Wait() renvoie faux et permet de sortir de la boucle. En plus des ports utilisateurs, deux ports systèmes sont créés systématiquement à l'initialisation du module : *beginIt* comme port d'entrée et *endIt* comme port de sortie. Lorsque le module appelle Wait(), un message est émis du port *endIt* contenant le numéro d'itération du module.

Cette API est disponible en C, C++ et Python. Celle ci est volontairement simple pour faciliter au maximum l'intégration de codes existants. Dans la suite de ce document, on appellera moduleAPI ces instructions. La Figure-3.1 présente un squelette de module. La lecture et l'écriture de message est décrite dans la section 3.1.5.

Au niveau d'un module, aucune information n'est disponible à la fois sur la provenance des messages en entrée et sur la destination des messages de sortie. Ceci est fait pour faciliter au maximum la réutilisation des modules. Il est possible par exemple d'avoir de multiples instances d'un même module dans une application.

Habituellement, un module correspond à un processus ou un *thread*. Il n'y a toutefois aucune obligation de suivre cette logique. Pour un code MPI par exemple, le développeur peut choisir d'instrumenter soit l'ensemble des processus MPI de l'application, soit uniquement le rang 0. Le développeur a également la possibilité de créer plusieurs modules dans un seul processus. Dans ces cas là, il y aura plusieurs boucles infinies qui tourneront simultanément. Cette méthode peut être utile par exemple lorsque l'on souhaite découpler la fréquence des messages d'entrées de la fréquence d'envoi des messages.

Il est important de noter également qu'un module n'est pas attaché à une ressource de calcul particulière. Un module FlowVR peut être un exécutable classique utilisant un accélérateur GPU ou Xeon Phi. De même un module peut être *multi-threadé*.

Pour lancer un module, FlowVR a uniquement besoin de connaître sa ligne de commande ("mpirun xx" par exemple). Dans le cas des codes parallèles, une ligne de commande peut lancer plusieurs modules. Pour des raisons d'identification, chaque module doit avoir un nom unique dans une application FlowVR. Ce nom est généralement basé sur le nom de son exécutable mais il reste à la charge de l'utilisateur. Pour pouvoir identifier chacun des modules dans un code parallèle, le nom du module est souvent complété par un numéro correspondant au rang du module. Ce rang peut être obtenu soit par FlowVR, soit fourni par le code utilisateur (le rang MPI par exemple).

3.1.3 Le modèle de dataflow

Une fois les modules implémentés, il reste à former le graphe de l'application. La déclaration du graphe se fait en trois étapes : décrire tous les modules que l'on va utiliser, instancier tous les modules de l'application et enfin créer les liens des ports d'entrée vers les ports de sortie. L'ensemble de ces trois étapes se fait dans un script Python. Dans la suite de ce document on appellera *flowvr-appy*[1] l'API pour décrire le réseau FlowVR.

L'entité de base d'un graphe FlowVR est le composant. Celui ci peut englober un ou plusieurs modules. Différentes informations sont attachées à un composant :

1. Pour toute réclamation concernant ce nom, merci d'adresser vos plaintes à Jeremy Jaussaud

```
#include <flowvr/module.h>
#include <iostream>
#include <unistd.h>

int main(int argc, const char** argv)
{
    flowvr::OutputPort *pOut = new flowvr::OutputPort("out");
                //Output messages
    flowvr::InputPort *pIn = new flowvr::InputPort("in");
                //Input messages
    flowvr::InputPort *pInEvent = new flowvr::InputPort("inEvent");
            //Optionnal input messages
    pInEvent->setNonBlockingFlag(true);

    std::vector<flowvr::Port*> ports;
    ports.push_back(pIn);
    ports.push_back(pInEven);
    ports.push_back(pOut);

    flowvr::ModuleAPI *flowvrModule = flowvr::initModule(ports);
    if(flowvrModule == NULL)
        return 1;

    while (flowvrModule->wait()){
        //Getting a message on a non blocking port
        flowvr::Message msgEvent;
        flowvrModule->get(pInEvent,msgEvent);

        //Checking if the message is empty
        if(msgEvent.data.getSize() > 0){
            //Processing the event...
        }

        //Getting new input message
        flowvr::Message msgData;
        flowvrModule->get(pIn,msgData);
        //Proces the data...

        //Creating output message
        flowvr::MessageWrite msgOut;
        //Filling the output message...

        flowvrModule->put(pOut, msgOut);
    }

    flowvrModule->close();

    return 0;
}
```

Figure 3.1 – Squelette d'un module FlowVR en C++. Le module dispose d'un port d'entrée bloquant et non bloquant ainsi que d'un port de sortie.

```
class Viewer(Component):

  def __init__(self, prefix, hosts):
    Component.__init__(self)

    viewer = Module(name = prefix , cmdline = "viewer", host =
       hosts)
    viewer.addPort("in", direction = "in")
    viewer.addPort("inEvent", direction = "in")
    viewer.addPort("out", direction = "in")
    ### expose input and output ports
    self.ports = viewer.ports
```

Figure 3.2 – Description en Python du module de la Figure 3.1. Un nom, une ligne de commande et un hôte lui sont attribués. L'interface de ses ports est également décrite. Finalement, l'ensemble des ports du module sont exposés au reste de l'application.

son nom, un lanceur (ssh, *mpirun*), une série d'hôtes (machines sur lesquelles vont être exécutée les modules), ainsi que la déclaration des modules de ce composant avec leurs ports. Pour exécuter une application dans un contexte distribué, FlowVR a besoin de pouvoir lancer un module à distance. Pour les codes MPI, le lanceur *mpirun* est suffisant. Pour les modules simples sans parallélisme, ssh est très souvent utilisé. Il faut impérativement que les ports décrits lors de cette phase soient cohérents avec les ports déclarés dans le code du module. Enfin le développeur doit choisir les ports qu'il souhaite exposer au niveau de l'application. Il peut décider soit d'exposer tous les ports de tous les modules inclus dans le composant, ou seulement une partie. Ceci peut être utile si des ports de certains modules ne sont utilisés qu'à l'intérieur de ce composant. La Figure-3.2 présente un exemple de description d'un composant contenant un module.

La seconde étape consiste à instancier les modules, c'est-à-dire de fixer les paramètres correspondant aux noms et aux hôtes. Finalement, il reste au développeur à lier les ports des modules entre eux. Créer un canal de communication ne peut se faire que d'un port de sortie vers un ou plusieurs (broadcast) ports d'entrée. La Figure-3.3 présente un réseau simple avec un producteur, un visualiseur et une interface. Certains ports ne sont pas connectés dans ce réseau. Dans le cas d'un port de sortie, le message émis est simplement détruit. Un port d'entrée non connecté est exclu des ports sur lesquels Wait() attend un message.

Aucune restriction n'est imposée quant à la configuration du graphe. Le développeur peut en particulier créer des boucles, ce qui peut s'avérer très utile pour des applications intégrant des retours d'information. Toutefois, étant donné que les ports d'entrée sont bloquants, créer une boucle dans le graphe de l'application provoquerait un *deadlock*. L'utilisateur a alors la possibilité d'ajouter un composant spécial appelé *PreSignal* qui insère un jeton dans la boucle permettant de l'initialiser.

```
#Assuming we have describe 3 components : Viewer, Interface and
    Engine
#Instanciate the modules
myViewer = Viewer("myViewer", "machine1")
myInterface = Interface("myInterface", "machine1")
myEngine = Engine("myEngine", "machine2")

#Creating te network of the application
myInterface.getPort("outEvent").link(myViewer.getPort("inEvent"))
myEngine.getPort("out").link(myViewer.getPort("in"))
```

Figure 3.3 – Description d'un réseau simple avec un producteur, un visualiseur et une interface. L'interface envoie de manière peu fréquente des données vers le visualiseur.

Par défaut, les canaux de communication sont FIFO (First In First Out). À chaque port d'entrée est associé une file d'attente. Lorsqu'un message arrive, il est inséré dans la file du port correspondant. Lorsque le module appelle Wait(), le message en tête de la file est transmis au module et retiré de la file d'attente. Une conséquence des canaux de communication FIFO est le risque de débordement des files d'attente de message. Il s'agit de la situation classique entre un producteur et son consommateur lorsque le producteur produit plus que ce que le consommateur ne peut traiter.

Pour affiner la politique de gestion des messages, le programmeur dispose de plusieurs composants appelés filtres et synchroniseurs pour éviter ce type de situation. Il s'agit de modules allégés servant à manipuler les messages dans les files d'attente. Une des principales différences des filtres avec les modules est qu'un filtre a accès à ses files d'attente de messages. Ceci permet entre autre de pouvoir soit échantillonner les messages d'entrée soit dupliquer des messages. Un composant classique utilisé dans les applications FlowVR est le *Greedy*. Ce composant est utilisé pour réenchantilloner un flux de données entre un producteur et un consommateur. La Figure 3.4 présente le schéma d'un Greedy entre un producteur et un consommateur. Lorsqu'un consommateur a fini de traiter une donnée, le *Greedy* transmet au consommateur la donnée la plus récente fournie par le producteur et détruit les plus anciennes. Si le producteur n'a pas produit de nouvelles données, alors le Greedy renvoie soit le dernier message reçu soit un message vide suivant les paramètres du composant. Ce composant est très souvent utilisé en amont d'un visualiseur par exemple.

Pour des applications complexes avec un grand nombre de modules, il serait fastidieux de créer l'ensemble des canaux de communication à la main. L'utilisation de Python permet de créer des schémas de communication complexes et réutilisables pour toutes sortes d'applications. Les applications parallèles utilisent notamment des schémas de communication classiques que l'on peut reproduire. On peut citer

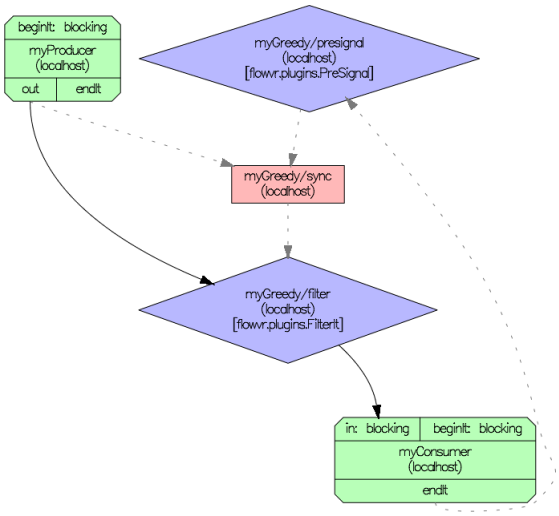


Figure 3.4 – Le *Greedy* échantillonne les données entre un producteur et un consommateur. Lorsqu'un producteur envoie un message le *FilterIt* reçoit le message et le stock. Parallèlement, un message léger est envoyé au synchroniseur avec le numéro de l'itération correspondant à l'envoie du message. Lorsque le consommateur a fini de traiter un message, un message est envoyé de son port *endIt* vers le synchroniseur. Une fois reçu, le synchroniseur choisit quel message le consommateur doit recevoir en fonction des numéros de message actuellement bufferisés. Le numéro d'itération choisi est envoyé au *FilterIt* qui va envoyer le message demandé vers le consommateur et supprimer les plus anciens. Notons que la politique d'échantillonnage peut être modifiée simplement en changeant le synchroniseur. Si l'on souhaite n'avoir que le dernier message reçu, il est alors possible de simplifier ce schéma en supprimant le synchroniseur et en envoyant directement le message de *endIt* au *FilterIt*.

par exemple le N-to-1, le 1-to-N ou le N-to-N. Les N-to-1 et 1-to-N génèrent chacun un graphe en forme d'arbre. Le développeur peut choisir quelle opération il souhaite appliquer sur les données pour chaque nœud ainsi que l'arité de l'arbre. En général, le N-to-1 est associé au *FilterMerge*. À chaque itération ce filtre concatène les données en entrée pour en faire un seul message en sortie. Le 1-to-N peut avoir plusieurs fonctionnalités. La première peut être simplement un *broadcast*. Dans ce cas la référence du message est dupliquée N fois en parallèle (pas de copie, voir la section 3.1.5). La seconde possibilité est de diviser le message d'entrée en N messages. À chaque nœud de l'arbre, le message d'entrée est divisé en P sous parties où P est l'arité de l'arbre. Pour chaque nœud de l'arbre, le développeur peut utiliser un filtre ou module qu'il aura développé lui même. Le N-to-N crée des communications point à point entre deux composants parallèles ayant le même nombre de modules.

À partir de ces schémas de bases, des schémas plus complexes peuvent être générés par combinaison. Par exemple, un schéma dans l'esprit map/reduce peut être obtenu en combinant un N-to-N avec un N-to-1. La Figure 3.5 présente une partie du code pour générer un tel schéma ainsi que la représentation du graphe générée.

Dans l'ensemble des exemples présentés jusqu'ici, localhost a toujours été le nom d'hôte. Toutefois le développeur peut utiliser n'importe quel nom de machine reconnu par le réseau où l'application sera déployée. Sans autre modification, l'application sera alors distribuée sur plusieurs nœuds de calcul.

Une fois l'application décrite, une série de fichiers xml sont générés. Ceux ci contiennent toutes les informations dont a besoin FlowVR pour pouvoir déployer l'application sur la plateforme cible. Le premier fichier (*.cmd.xml) contient des séries de commandes que doit exécuter FlowVR pour initialiser les modules avec leurs ports, placer les filtres ainsi que les canaux de communication. Ces commandes sont destinées aux démons FlowVR uniquement. Le deuxième fichier (*.run.xml) contient l'ensemble des lignes de commandes construites à partir des scripts Python pour lancer les différents modules. Ces deux fichiers sont lus lors du lancement de l'application par la commande *flowvr* qui permet de déployer l'application sur l'architecture cible.

Une modification sur le réseau de l'application ne nécessite pas de recompiler l'ensemble des modules. Ceci permet de pouvoir modifier et tester facilement différentes configurations d'application en un minimum de temps.

3.1.4 Fonctionnement du démon

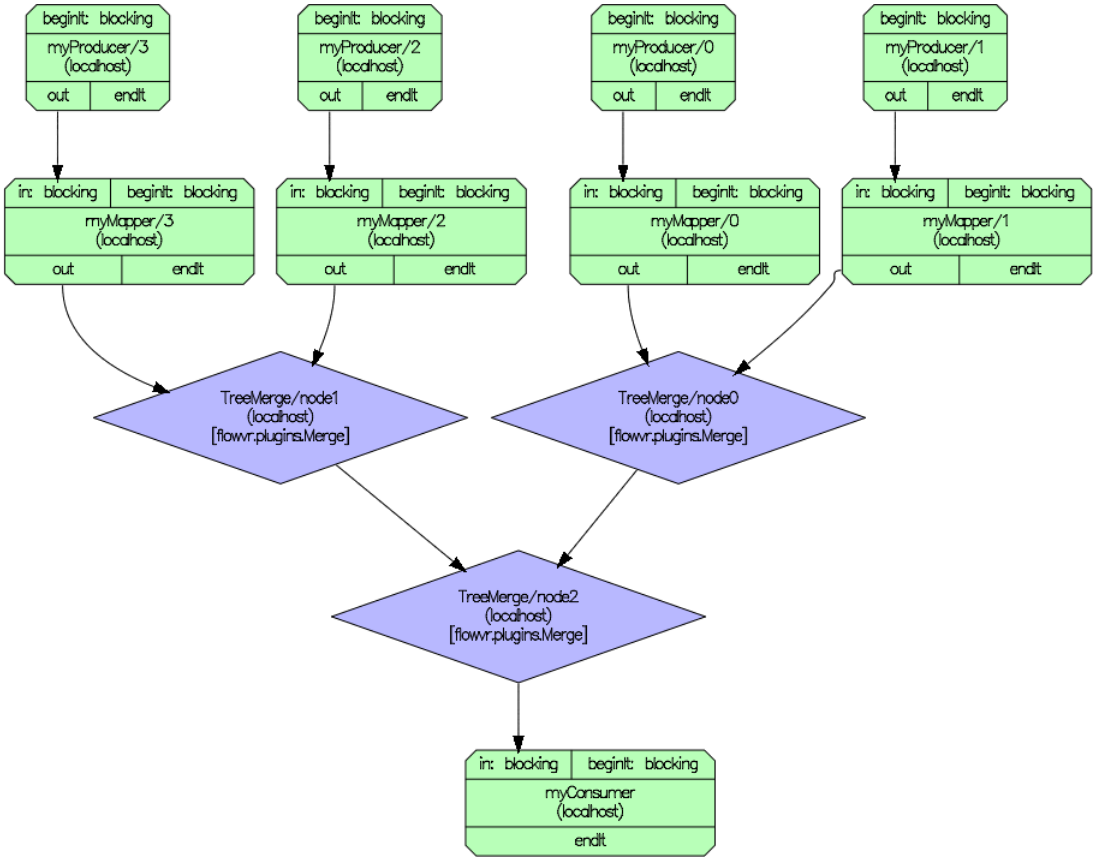
Pour assurer la coordination de l'application, un démon FlowVR est présent sur chacun des hôtes impliqués dans l'application. Un démon FlowVR est un processus multithreadé. Il assure principalement 3 taches : faire le lien avec les modules, gérer les communications entre modules et héberger un segment de mémoire partagée.

Les modules ne communiquent jamais entre eux directement. Un module com-

```
myProducer = ProducerParallel("myProducer", "localhost localhost
    localhost localhost")
myMapper = Mapper("myMapper", "localhost localhost localhost
    localhost")
myConsumer = Consumer("myConsumer", "localhost")

link_point_to_point(myProducer.getPort("out"), myMapper.getPort("in
    "))
make_filter_tree("TreeMerge", myMapper.getPort("out"), myConsumer.
    getPort("in"), arity = 2, node_class = FilterMerge)
```

(a)



(b)

Figure 3.5 – Présentation d'un réseau imitant un schéma Map/Reduce. *link_point_to_point* génère une connexion point à point entre le producteur et les mappers. *make_filter_tree* génère un schéma N-to-1 avec une arité de 2. À chaque nœud, le filtre *FilterMerge* est appliqué. Celui ci est interchangeable avec n'importe quel module utilisateur.

munique uniquement avec le démon local qui lui sait comment et à qui transmettre les messages. Pour chaque module, un thread appelé *Regulator* est créé sur le démon de la machine hôte. C'est lui qui gère entre autre les files d'attente de messages du module qui lui est associé. Lorsqu'un module appelle la fonction Wait(), une communication bloquante est effectuée avec son *Regulator*. Lorsque chaque file de message a au moins un message, les messages en tête de file d'attente sont retirés et transmis au module.

Chaque démon dispose d'une table d'action construite à partir du fichier *.cmd.xml. Cette table associe une source de message (c'est-à-dire un port de sortie) à une ou plusieurs actions. Lorsqu'un module effectue un *Put()*, le *Regulator* consulte la table d'action et peut effectuer trois types actions :

— Transmettre le message à une file d'attente. Cette action est effectuée lorsque deux modules situés sur la même machine s'échangent un message. En pratique cette manipulation consiste simplement à s'échanger des pointeurs.
— Transmettre le message au plugin de communication chargé par le démon. Cette action est effectuée lorsqu'un message doit être envoyé à un module situé sur une autre machine. Le message est alors transmis à un plugin spécifique du démon pour effectuer la transmission sur le réseau vers le démon cible.
— Exécuter un filtre. Si le message est envoyé à un filtre, celui-ci peut être exécuté directement par le *Regulator*. La conséquence est qu'un module ne peut sortir d'un appel à *Wait()* tant que son Regulator est en train d'exécuter un filtre.

Pour gérer les communications entre des modules distants, chaque démon dispose d'un plugin qui implémente un protocole de communication. Deux versions de ce plugin existent : la première utilisant TCP/IP et la deuxième utilisant MPI. La première est l'implémentation historique de FlowVR. L'utilisation de TCP/IP sur un réseau haute performance comme on en trouve sur les supercalculateurs est en général peu performante. Dans cette optique, la version MPI a été développée par Jeremy Jaussaud pour répondre aux exigences de performance. MPI permet à FlowVR de supporter un maximum de types de réseau haute performance avec un même plugin (Infiniband, Myrinet, Gemini, etc...). Pour cela, les implémentations MPI utilisent des protocoles bas-niveaux propres à chaque infrastructure réseau comme verbs pour l'Infiniband par exemple.

L'échange de messages entre modules et donc entre processus est réalisé par l'intermédiaire d'une mémoire partagée. Cette mémoire est hébergée par le démon. Les modules ont alors la possibilité de demander des *buffers* pour écrire leurs données. Il obtiennent alors un pointeur sur une zone mémoire qu'ils peuvent utiliser comme un pointeur classique. Un pointeur intelligent est associé à chaque buffer alloué. Un buffer est libéré de la mémoire partagée lorsque le compteur de référence de son pointeur intelligent passe à zéro. Deux modules d'un même nœud s'échangent donc des messages par le biais de la mémoire partagée, sans copie. La Figure 3.6 résume

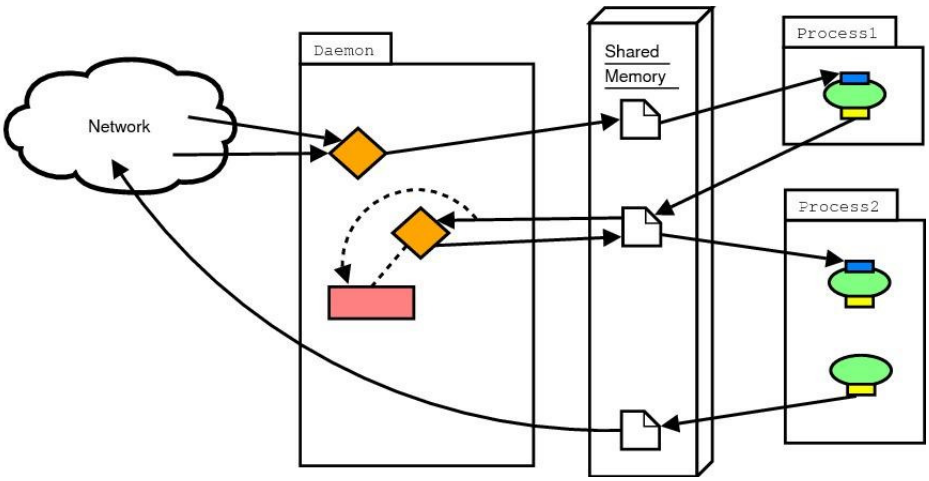


Figure 3.6 – Récapitulatif des actions possibles lors de l'échange de messages. Lors d'un échange entre deux modules d'un même hôte, une simple transmission de pointeur est faite. Pour deux modules distants, les démons s'occupent de transférer les données sur le réseau via un plugin Net. Enfin le *Regulator* hébergé sur le démon peut exécuter un filtre(losange orange et rectangle rouge).

les différents cas d'échanges de messages entre modules.

3.1.5 Gestion des messages

Contrairement à d'autres frameworks comme FlexIO ou Damaris, FlowVR donne un accès direct à la mémoire. Lorsqu'un module veut écrire ou lire un message, il reçoit un buffer alloué dans la zone de mémoire partagée gérée par le démon. Lorsqu'un module effectue un *Put()*, le *Regulator* récupère simplement le pointeur en mémoire partagée évitant toute copie. Une fois le *Put()* effectué, le développeur n'est plus autorisé à écrire dans le buffer. Les buffers associés à des messages obtenus par un *Get()* sont en lecture seule. Ce choix permet d'envoyer le même message à plusieurs modules locaux avec une seule instance du message en mémoire partagée. De plus, si l'utilisateur souhaite modifier les données qu'il a reçues, il est alors forcé de faire lui même une copie. Les allocations et écritures mémoires sont souvent coûteuses et peuvent avoir des impacts importants sur un code. En exposant ces allocations dans le code utilisateur, le développeur a la maîtrise de ces opérations coûteuses

FlowVR ne propose pas d'API haut niveau pour écrire ou lire des données comme c'est le cas dans ADIOS ou Damaris. Pour écrire ou lire en mémoire, le développeur doit manipuler des pointeurs comme il le ferait dans un code C/C++ classique. Ceci permet au développeur d'avoir toute la latitude nécessaire pour représenter ses structures de données.

Pour réduire le coût des allocations mémoires, FlowVR propose un système de

BufferPool. Lorsque le développeur demande un buffer pour écrire un message, le *BufferPool* regarde dans sa liste de buffers s'il dispose d'un buffer suffisamment grand et libre déjà alloué. Un autre mécanisme important pour éviter des allocations est la possibilité de chaîner des buffers. Ceci évite notamment toute copie lorsque l'on souhaite concaténer des messages. Cette fonctionnalité est très pratique par exemple pour regrouper l'ensemble des données produites par une simulation au niveau d'un nœud. On dit alors que le message produit est segmenté. Si un tel message doit être envoyé sur le réseau, celui ci peut être sérialisé en copiant tous les segments du message segmenté dans un même buffer avant de l'envoyer sur le réseau si cela permet un gain de performance.

En plus d'un buffer de données, une liste d'estampilles est attachée à chaque message. Les estampilles sont des métadonnées associées à un message. Les cinq premières estampilles sont gérées automatiquement par FlowVR. Elles contiennent des informations comme la source du message et le numéro d'itération de la source. Cette liste d'estampille peut être complétée par des estampilles utilisateurs, comme le numéro d'itération d'une simulation ou bien le temps simulé. Les estampilles peuvent être envoyées sans le buffer contenant les données. On appelle alors ce message un *messageStamp*. Le port recevant un tel message doit toutefois avoir été déclaré comme un port estampille. Le synchroniseur utilise beaucoup cette possibilité. En effet, dans le schéma du *Greedy* (Figure 3.4), le synchroniseur reçoit uniquement les estampilles des messages du producteur et du consommateur. Le message complet du producteur est envoyé au *FilterIt*. Comme le synchroniseur a uniquement besoin des numéros d'itération du producteur et du consommateur, il n'est pas utile de lui envoyer le buffer de données.

3.1.6 Placement contraint de la mémoire et des processus

Dans les architectures NUMA [2], le placement des processus et de la mémoire peut avoir un impact très significatif sur les performances de la simulation. En effet, à chaque processeur est associé un banc mémoire privilégié. Si des données doivent être chargées depuis un autre banc mémoire, les temps d'accès deviennent plus longs. Il est donc souhaitable qu'un processus n'accède qu'aux données de son banc privilégié. Dans un système d'exploitation classique, l'ordonnanceur à la possibilité de migrer un processus et le faire changer de processeur (et donc de banc mémoire privilégié). Ceci peut mener à des pertes de performances importantes à cause d'une part du coût de la migration de processus et d'autre part des accès en mémoire qui vont accéder à l'ancien banc mémoire privilégié. Pour renforcer ce placement, les implémentations MPI peuvent nativement contraindre le placement des processus. Ce mécanisme assure qu'un processus est contraint à une enveloppe de cœurs logiques et ne sera pas migré sur un autre cœur par le système d'exploitation.

Dans le contexte in-situ, la simulation n'est plus seule à utiliser les ressources.

2. Non Uniform Memory Access

Les analyses sont placées en concurrence avec les processus de la simulation. La *flowvr-appy* a été adaptée pour permettre au développeur d'un module de spécifier explicitement sur quels cœurs un module doit être exécuté. L'implémentation pour les modules non MPI repose sur l'outil système *taskset*, un outil système de Linux permettant de spécifier une enveloppe CPU à un processus. Pour les modules MPI, nous nous basons sur les mécanismes déjà présents dans MPI pour placer les processus. La syntaxe étant toutefois différente entre les différentes implémentations de *mpirun*, FlowVR dispose d'un lanceur pour chaque implémentation MPI (OpenMPI, MPICH, Intel, MVAPICH).

Lorsqu'un module est fixé, il peut demander à avoir un segment de mémoire partagée sur son domaine NUMA. Ceci se fait dans la partie *flowvr-appy* où le développeur peut transmettre certaines variables d'environnement. À l'initialisation d'un module, le *Regulator* se place sur la même enveloppe CPU que le module. Il crée alors son propre segment de mémoire partagée et le place dans son domaine NUMA. Ces mécanismes permettent d'assurer des temps d'allocation aussi rapides que possible pour un module ainsi que des temps de communication minimaux entre un module et son *Regulator*.

3.1.7 Forces de FlowVR pour des traitements in-situ

FlowVR apporte un certain nombre de fonctionnalités et concepts pouvant grandement aider à la conception d'applications in-situ.

Support de l'in-situ et l'in-transit. FlowVR supporte les traitements in-situ (traitements sur les nœuds de simulation) grâce notamment à un segment de mémoire partagée ainsi que les traitements in-transit via ses plugins de communication réseau. L'échange des données se fait de manière totalement transparente pour le développeur grâce au démon FlowVR. Pour passer d'un mode in-situ à in-transit, le développeur doit simplement changer les noms d'hôte des modules dans le script Python. Le *runtime* de FlowVR se charge alors de l'envoi des messages. Ceci permet de créer des applications hybrides in-situ/in-transit et de pouvoir tester différentes configurations de placement très facilement.

Paradigme de dataflow. Les analyses effectuées par les chercheurs sur les jeux de données sont très souvent des successions d'analyse où le résultat de la première est utilisé pour la seconde et ainsi de suite. La paradigme de dataflow proposé par FlowVR permet de reproduire de manière très intuitive cette démarche. Il permet de créer des pipelines d'analyse complexes en enchaînant des modules. Plusieurs pipelines d'analyse peuvent également être exécutés en parallèle à partir d'une même source de données.

Niveau de contrôle de l'application. En explicitant lui même les schémas de communications ainsi que les copies, le développeur a une vision aussi claire que possible de ce qui se passe dans son application. Ceci lui permet de mieux comprendre le comportement de son application et peut grandement aider lors des phases de debug. De plus, le placement explicite des processus permet à l'utilisateur de choisir lui même la meilleure stratégie à adopter pour réduire l'impact des traitements in-situ sur les performances de la simulation.

3.2 Gromacs

Gromacs[40, 69], est un code de dynamique moléculaire couramment utilisé dans le monde de la biologie. Gromacs et NAMD[67] sont les deux simulations de dynamique moléculaire les plus performantes. Toutes deux peuvent simuler des modèles de plusieurs millions d'atomes répartis sur plusieurs milliers de cœurs. D'autres solutions moins répandues existent également comme CHARMM[19] ou Amber[24].

3.2.1 Simulation de dynamique moléculaire

Une simulation de dynamique moléculaire vise à faire évoluer un système de particules. Généralement ce système de particules représente une macro-molécule (protéine, membrane, canal, etc...) plongée dans un solvant (de l'eau avec des ions). Pour simplifier, on supposera par la suite qu'une particule correspond à un atome. Il est toutefois possible de simuler des particules qui représentent plusieurs atomes qu'on appelle alors des gros grains.

En début de simulation, chaque particule est initialisée avec une position et une vitesse. Une simulation de dynamique moléculaire est un procédé itératif. Chaque itération d'une simulation de dynamique moléculaire est composée de deux étapes : 1) calculer les forces exercées sur chaque atome en appliquant les lois de la mécanique classique ; 2) utiliser ces forces pour mettre à jour la vitesse et la position des atomes. Entre deux itérations, la simulation avance dans le temps d'un intervalle δ_t qu'on appelle pas d'intégration. La valeur de δ_t est choisie en fonction des formules utilisées lors de la simulation.

Le calcul des forces utilise un modèle appelé champ de forces qui défini l'énergie potentielle du système comme une fonction par rapport aux coordonnées des atomes. La force appliquée à un atome est la dérivée de cette énergie potentielle par rapport à la position de l'atome. Des décennies de travaux ont été nécessaires pour développer et affiner les modèles des champs de force afin de faire correspondre ces modèles à la fois aux données expérimentales et quantiques. Ceci permet de pouvoir utiliser un pas d'intégration de quelques femtosecondes [3] tout en conservant un niveau d'approximation raisonnable.

3. 10^{-15} secondes

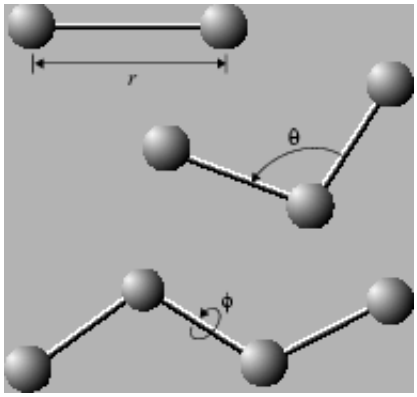


Figure 3.7 – Récapitulatif des interactions liées. La longueur du lien entre atomes, l'angle entre trois atomes et l'angle diédrale entre quatre atomes sont pris en compte pour la calcul de l'énergie potentielle. [5]

L'énergie potentielle totale d'un système chimique (E) est exprimée comme une somme de la forme :

$$E = E_{bonded} + E_{es} + E_{vdW}$$

E_{bonded} est la somme de plusieurs termes dit liés qui dépendent des liens de covalence de la structure moléculaire. Ces termes incluent :

— terme calculé en fonction de la longueur du lien entre deux particules,
— terme calculé en fonction de l'angle du lien entre trois atomes chaînes,
— terme calculé en fonction de l'angle diédrale (torsion) impliquant quatre atomes liés par trois liens.

La Figure-3.7 présente un récapitulatif des interactions liées.

E_{es} et E_{vdW}, respectivement les termes électrostatiques et *van der Waals*, sont appelés des termes non liés car ils incluent des interactions entre toutes les paires d'atomes possibles. Ils représentent une charge de calcul bien plus importante que le calcul des termes liés. Les forces Van der Waals diminuent d'intensité assez rapidement avec la distance entre deux particules. Elles peuvent être assez vite négligées pour des particules espacées d'une certaine valeur seuil généralement choisie entre neuf et douze Å [6]. Pour les forces électrostatiques, des études ont montré que négliger les interactions électrostatiques au delà d'un certain seuil n'était pas réaliste avec un solvant explicite. Les forces électrostatiques sont calculées par des méthodes efficaces mais approximatives qui prennent en compte les interactions longues distances sans toutefois calculer explicitement les interactions entre toutes les paires d'atomes. Une des méthodes est la *particule mesh Ewald* (PME). Elle utilise des transformées de Fourier (FFT) pour calculer le potentiel électrostatique sur un maillage en ayant la distribution des charges sur ce maillage.

5. source Figure 3.7 : http ://www.ks.uiuc.edu/Training/Tutorials/namd/namd-tutorial-win-html/node26.html
6. 1Å = $10^{-10}m$

La simulation de dynamique moléculaire classique est par nature une approximation. Une solution exacte serait de résoudre l'ensemble des équations de mécanique quantique. Il faudrait toutefois réduire l'ordre de grandeur du pas d'intégration de la simulation à l'attoseconde [7]. Avec un tel pas d'intégration, le temps nécessaire pour simuler un événement biologique devient bien trop long pour être acceptable. Le calcul de l'énergie potentielle est donc préféré à la résolution des équations quantiques notamment parce qu'il offre un bon compromis entre précision et temps de calcul. Pour de très grands ensembles moléculaires comme des virus, les simulations à gros grains sont utilisées. Agréger des atomes entre eux permet d'une part de réduire le nombre de particules à simuler mais aussi de simplifier les équations à résoudre. Ceci permet d'utiliser un pas d'intégration de l'ordre de la picoseconde au prix d'une simulation moins réaliste.

3.2.2 Description interne de Gromacs

Gromacs est un code de simulation de dynamique moléculaire avec une parallélisation hybride MPI/OpenMP/GPU. Dans Gromacs, comme dans beaucoup d'autres codes de dynamique moléculaire, chaque processus MPI prend en charge l'ensemble des atomes situés dans une région de l'espace. En général, l'ensemble de l'espace occupé par la simulation est un parallélépipède divisé en une grille régulière formée de boîtes. Pour simplifier, on supposera toujours dans la suite de ce document que la boîte globale de la simulation est un parallélépipède. D'autres formes sont toutefois possibles selon les implémentations des simulations de dynamique moléculaire. Chaque processus met à jour la position des atomes se trouvant dans sa boîte. On appelle ces atomes les *home atoms*.

Gromacs utilise un modèle maître/esclave. Le maître (rank 0) est responsable de la distribution initiale des atomes, de calculer les valeurs globales du système comme le niveau d'énergie ou la température ainsi que de toutes les opérations d'écriture. Pour pouvoir gérer et échanger des données avec les esclaves, des fonctions utilitaires sont disponibles dans Gromacs pour distribuer des données du maître vers les esclaves et inversement rassembler les données vers le maître. À l'initialisation, le maître propage les atomes aux esclaves. Lors d'une itération avec écriture, l'ensemble des positions, vitesses et forces sont rassemblées de manière synchrone sur le nœud maître puis sauvegardées sur disque. Ces opérations de *broadcast/gather* ont un coût certain sur les performances de la simulation. En effet, ces opérations sont bloquantes et passent difficilement à l'échelle avec le nombre de processus. Gromacs utilise donc ces opérations à une fréquence assez faible. Cela signifie également que la fréquence d'écriture de fichier, qui utilise ces méthodes, devrait être limitée. Typiquement, pour des simulations longues, seule une itération sur 1000 à 5000 itérations est sauvegardée.

Au fur et à mesure que la simulation avance, les atomes bougent et peuvent chan-

7. $1as = 10^{-18}s$

ger de boîte. Des communications inter-processus sont donc nécessaires pour échanger les positions des atomes en bordure des boîtes de chaque processus. Toutes les n itérations, une phase de calcul de voisinage est faite. Lors de cette phase, un atome peut changer de processus hôte. Des zones fantômes sont également construites. Lors des phases de calcul de voisinage, des communications globales sont effectuées. Lors des autres itérations, des communications sont effectuées uniquement entre des processus dont leurs boîtes sont voisines. Un compromis doit donc être trouvé pour la valeur de n. Généralement, un calcul de voisinage est effectué toutes les 10 itérations.

La composition d'un système moléculaire est très hétérogène. La densité d'atomes ainsi que la complexité des interactions entre atomes n'est pas la même entre les différentes boîtes. De plus, le système moléculaire peut subir des changements de conformations importants au cours de la simulation impliquant de grands mouvements d'atomes. Un découpage statique et régulier de l'espace serait donc inefficace. Gromacs utilise un système d'équilibrage de charge dynamique. Au cours de la simulation, Gromacs ajuste les dimensions de la grille du domaine de décomposition. Ainsi chaque processus peut avoir un nombre d'atomes ainsi qu'une dimension de boîte différente. L'objectif est d'équilibrer autant que possible le temps passé pour chaque processus dans le calcul des forces qui représente la plus grosse charge de calcul.

Le calcul des interactions électrostatiques requiert des calculs de FFT entre des boîtes distantes. Ceci implique des communications globales N-to-N très coûteuses qui sont généralement le facteur limitant du passage à l'échelle d'une simulation. Pour limiter l'impact que peuvent avoir ces communications globales, Gromacs utilise la stratégie du Multiple Programme Multiple Data (MPMD). À l'initialisation, un sous ensemble des processus MPI sont sélectionnés pour effectuer les calculs liés aux interactions électrostatiques. Nous les appelons processus PME (Particule Mesh Ewald). Les autres processus s'occupent du reste des calculs. L'intérêt de cette méthode est que les FFT impliquant des communications N-to-N vont être faites sur un nombre bien plus réduit de processus. Typiquement, entre 1/3 et 1/4 des processus sont des processus PME. En revanche, une synchronisation doit être faite entre les nœuds classiques et les nœuds PME. À chaque itération, les nœuds PME doivent recevoir les positions des atomes pour pouvoir commencer leurs calculs. De plus, à la fin du calcul des PME, ceux-ci doivent transférer les forces qu'ils ont calculées aux nœuds de calcul classiques pour qu'ils puissent calculer les forces totales pour chaque atome. Le ratio entre les nœuds classiques et PME devient ici important. Les nœuds de calcul ne peuvent pas finir le calcul des forces tant que les nœuds PME n'ont pas terminé. Il faut donc suffisamment de nœuds PME pour finir les calculs d'électrostatique pendant que les nœuds de calcul classiques calculent le reste des forces. Un équilibrage doit donc être trouvé à ce niveau également. Les développeurs suggèrent fortement, pour des simulations longues, de tester plusieurs ratios avant de lancer la simulation définitive.

Depuis la version 4.6, Gromacs peut utiliser plusieurs GPUs par nœud pour

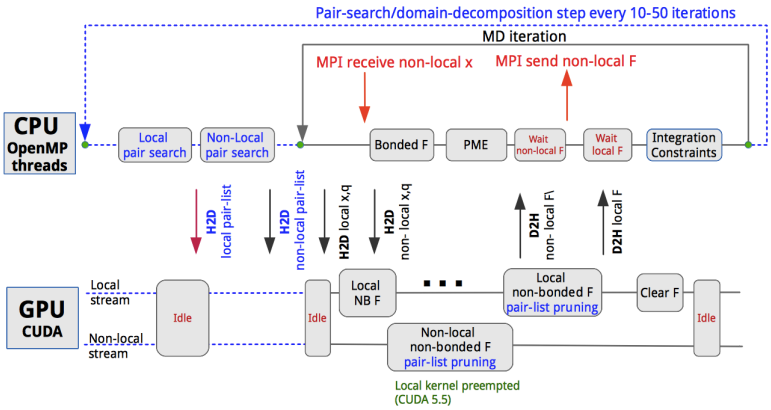


Figure 3.8 – Pipeline d'une itération avec support GPU. Les données sont d'abord transférées sur le GPU avant le calcul des forces non liées sur le GPU. [9]

l'ensemble du pipeline de calcul. Une des particularités des simulations de dynamique moléculaire est le temps d'une itération qui est de l'ordre de la milliseconde. Cela rend très difficile l'utilisation d'un accélérateur car les transferts de données sont coûteux. La solution adoptée est de déporter le calcul des forces non liées sur le GPU. Les données sont transférées en amont pendant que le CPU effectue d'autres calculs. La Figure 3.8 présente le pipeline d'une itération avec GPU.

Lorsqu'un ou des GPUs sont utilisés, un GPU est associé à un processus MPI. Un processus MPI correspond toujours à une boîte du domaine de décomposition. Lorsqu'il y a plus de cœurs que de GPU sur un nœud, OpenMP est utilisé pour paralléliser les calculs au niveau du nœud. Par exemple, pour un nœud composé de deux processeurs quadri-cœur ainsi que deux GPUs, la configuration classiquement utilisée est deux processus MPI, un GPU par processus MPI et quatre threads OpenMP par processus MPI.

3.3 Méthode de couplage à Gromacs

Au cours de toutes nos expérimentations, nous utilisons la même méthode de couplage avec Gromacs. Nous résumons ici les modifications apportées à Gromacs pour nous permettre d'extraire des données de la simulation. Le traitement de ces données en dehors de Gromacs sera décrit dans les chapitres suivants.

3.3.1 Enjeux

Gromacs est un code particulier à instrumenter pour plusieurs raisons. Ses particularités liées à la fois au domaine de la dynamique moléculaire mais aussi à son

9. Source Figure 3.8 : http ://www.gromacs.org/GPU_acceleration

implémentation posent un certain nombre de difficultés techniques :

— Temps d'une itération. Un temps d'itération en dynamique moléculaire est extrêmement court. Là où l'ordre de grandeur d'une itération dans d'autres types de simulation est la seconde, l'ordre de grandeur d'une itération de Gromacs est la milliseconde. Même pour les plus gros modèles disponibles il est possible d'atteindre 1000 itérations à la seconde. En conséquence notre instrumentation du code de Gromacs doit être extrêmement peu coûteuse si l'on souhaite conserver un impact minime sur les performances de la simulation.

— Hétérogénéité du code. L'ensemble du code de Gromacs n'exécute pas la même boucle. Au delà de dix processus MPI, une portion des processus MPI est réservée pour les nœuds PME. Or on ne souhaite extraire des données que des nœuds de calcul classiques. De plus, il n'y a pas de possibilité de prévoir quels processus MPI vont être PME ou non. Cette information est seulement disponible à l'exécution. Il faut donc que notre instrumentation soit capable de s'adapter dynamiquement à la configuration de Gromacs.

— Système dynamique d'équilibrage de charge. Pour équilibrer sa charge, Gromacs mesure les temps d'exécution de certaines fonctions comme le calcul des forces. Il est impératif que notre instrumentation n'influence pas le comportement de Gromacs lors de ces prises de mesure. Dans le cas contraire, Gromacs pourrait en déduire un déséquilibre et tenter de le résoudre en modifiant les dimensions du domaine de décomposition. Or, comme le déséquilibre ne viendrait pas de la simulation, un rééquilibrage de charge par Gromacs empirerait sans doute les performances de la simulation.

3.3.2 Utilisation des méthodes maître/escalve de Gromacs

Une des difficultés pour récupérer l'ensemble des données de la simulation est qu'elles sont distribuées. Pour pouvoir analyser ou traiter les données comme les positions des atomes, il faut pouvoir rassembler les données. Le processus maître de Gromacs dispose de méthodes pour rassembler les données. Notre première approche consiste à instrumenter uniquement le nœud maître de Gromacs et de nous servir des fonctions natives de Gromacs pour rassembler les données sur le nœud maître avant de les extraire. Cette méthode a deux avantages majeurs : 1) Elle permet de simplifier grandement l'instrumentation du code en utilisant des fonctions natives de Gromacs ; 2) La gestion des nœuds est gérée de manière transparente par les fonctions de Gromacs. Cette méthode est notamment utilisée par les systèmes d'IMD [10] classiques que nous détaillerons dans le Chapitre 4. Bien que simple à mettre en œuvre, cette méthode présente deux défauts majeurs : 1) La méthode passe difficilement à l'échelle avec le nombre de nœuds car le *gather* utilisé pour rassembler les données est bloquant ; 2)Le parallélisme des données est perdu. Il n'y a donc plus de possibilités pour effectuer des traitements in-situ autre que sur le nœud maître à

10. Interactive Molecular Dynamic

moins de redistribuer les données. Une autre approche est donc nécessaire.

3.3.3 Instrumentation de chaque processus MPI

Notre seconde approche consiste à instrumenter chaque processus MPI de la simulation. À l'initialisation, chaque processus MPI (PME inclus) initialise un module avec des ports. Pour simplifier, nous supposerons que chaque module déclare un port de sortie qui va correspondre aux positions des atomes ainsi que de leurs index. Ces index sont nécessaires pour pouvoir faire correspondre une position à son atome ainsi que ses propriétés.

Une fois les modules initialisés, il faut déterminer quels seront les processus émettant des données, c'est-à-dire les nœuds qui ne font pas les calculs PME. Une fois que la simulation a attribué les nœuds PME, chaque module effectue un premier *Wait* et construit un message vide. Si le processus du module est un nœud PME, alors une estampille est rajoutée au message vide. Ceci est fait pour permettre au reste de l'application de savoir quels modules vont produire des données et quels modules peuvent être ignorés.

Gromacs lance ensuite la boucle principale. À partir de ce moment, seuls les nœuds classiques vont envoyer des données au reste de l'application. Une fois la simulation initialisée, Gromacs effectue une boucle d'itération avec un nombre d'itérations prédéfini par l'utilisateur. Toutes les n étapes, des données sont extraites de la simulation. Le paramètre n est donné en ligne de commande par l'utilisateur et sera adapté suivant le contexte d'utilisation de Gromacs. Toutes les n étapes, un *Wait* est effectué. Après le calcul des forces, les données relatives aux positions et à leurs index sont copiées dans un buffer FlowVR. Seuls les atomes locaux sont copiés dans ce buffer. Chaque processus MPI, en plus de ces atomes locaux, dispose d'atomes supplémentaires nécessaires aux calculs des interactions moléculaires. Toutefois les forces appliquées sur ces autres atomes ne sont pas calculées. En ne copiant que les atomes locaux, nous garantissons que les atomes ne sont pas dupliqués. Sont également ajoutées des estampilles pour représenter le numéro d'itération de la simulation, le temps simulé ainsi que les dimensions de la boîte du processus. L'Algorithme 1 résume la boucle principale de Gromacs modifiée avec FlowVR. L'écriture de trajectoire par Gromacs est désactivée (utilisation d'un *gather* bloquant) et sera prise en charge par FlowVR comme nous le verrons dans les Chapitre 5 et 6.

Cette méthode de couplage est totalement transparente vis-à-vis de l'utilisation d'OpenMP ou de GPU. Aucune modification de code n'est nécessaire suivant la configuration choisie. Ceci est dû au fait que la boucle principale de Gromacs est fixe par rapport au contexte d'exécution. Il est important de noter que, comparées à la méthode précédente, les données ont préservé leur niveau de parallélisme. Par conséquent, si une analyse a besoin de recevoir toutes les données d'un pas de temps de la simulation, c'est au reste de l'application au niveau du pipeline de FlowVR qu'il incombera de regrouper les atomes et de les réordonner.

```
while step < nstep do
    if is_flowvr_step(step) then
        wait()
    end
    dd_partition_system() ;
    do_force() ;
    if traj_enable then
        write_traj()
    end
    if is_flowvr_step(step) then
        position_message ← home_atoms ;
        put(position_message) ;
    end
    update_atoms_positions() ;
    step++ ;
end
```

Algorithm 1: Version modifiée de la boucle principale de Gromacs permettant l'extraction de données

Chapitre 4

Application interactive in-situ : simulation de dynamique moléculaire intéractive

Les simulations sont souvent considérées comme des boîtes noires lors de leurs exécutions. Le *computationnal steering* a permis de réintégrer les utilisateurs dans leurs simulations. L'expertise des utilisateurs peut permettre à la fois de vérifier le bon comportement d'une simulation ou alors de guider une simulation dans une direction désirée pour une étude. Dans ce chapitre, nous allons nous nous intéressons à ce dernier point. Nous proposons une application interactive construite autour du code de dynamique moléculaire Gromacs pour permettre à un utilisateur de favoriser des événements biologiques. Nous montrons en particulier comment les méthodes d'extractions de données et de traitements in-situ peuvent permettre de manipuler des simulations à grande échelle avec un coût raisonnable sur les performances de la simulation. L'ensemble des travaux présentés dans ce chapitre ont été publiés et présentés lors de la conférence *International Conference on Computationnal Science 2013*(ICCS).Dans ce chapitre, nous décrivons uniquement la méthode de couplage entre la simulation et le visualiseur et en particulier la gestion du flux de données entre ces deux composants. Pour plus d'informations sur le reste de l'application, le lecteur est invité à se référer à l'article.

4.1 Contexte

Les simulations de dynamique moléculaire sont bien souvent difficiles à prévoir. Partant d'un état initial identique, deux simulations de dynamique moléculaire peuvent obtenir des état finaux bien différents. Si un chercheur souhaite étudier un phénomène biologique, plusieurs simulations peuvent être nécessaires avant que le phénomène ne se produise. Ces essais infructueux coûtent non seulement un temps de calcul important mais posent également la question de la validité et de la reproductibilité des résultats obtenus.

Plusieurs méthodes ont été proposées pour améliorer la probabilité d'apparition de certains événements. Une méthode appelée *steered molecular dynamic* (SMD)[43] permet de modifier le comportement normal de la simulation en intégrant des forces définies par l'utilisateur dans la simulation. Un sous ensemble d'atomes de la si-

mulation est alors contraint par ces forces extérieures. Ceci permet à un utilisateur de “guider” le système moléculaire vers une direction souhaitée. Cette méthode est toutefois difficile d'utilisation. Les forces utilisateurs sont souvent définies par des scripts établis en amont de la simulation. Il faut donc que l'utilisateur prévoit le déroulement de la simulation au cours du temps. En pratique, il est difficile de faire suivre une trajectoire non linéaire à un groupe d'atomes. Pour des trajectoires plus complexes, il est plus commode de subdiviser la simulation pour obtenir des sous trajectoires linéaires. Un avantage non négligeable de cette méthode reste qu'elle ne modifie en rien le contexte d'exécution de la simulation, seul un script supplémentaire est requis.

Pour rendre plus facile d'accès cette méthode, l'*interactive molecular dynamic* (IMD)[76, 77] a été proposée. Le principe est toujours d'appliquer des forces extérieures sur un groupe d'atomes mais en intégrant l'utilisateur directement dans le processus de guidage. La simulation est connectée à un visualiseur ainsi qu'un système de bras haptique. Le visualiseur permet d'avoir une image de l'état de la simulation et en particulier sur la position du groupe d'atomes guidés. Le bras haptique permet à l'utilisateur d'appliquer des forces de manière intuitive avec un retour d'effort et six degrés de liberté (voir Figure 4.1). L'intérêt du bras haptique pour étudier les interactions moléculaires a d'ailleurs été souligné par le projet Grope[21]. Pouvoir visualiser la simulation alors qu'elle est en cours d'exécution permet à l'utilisateur d'adapter la direction et l'intensité des forces à appliquer. Toutes les difficultés liées à la prévision des forces à appliquer dans les systèmes SMD sont reportées à des décisions que l'expert prend en temps réel en ayant connaissance de la dynamique du système. Le contexte matériel n'est pas non plus le même. Un bras haptique ainsi qu'un accès interactif à la machine de calcul sont nécessaires pour mettre en place un tel système.

4.2 Application biologique : le canal FepA

Les membranes protéiques sont essentielles dans le transport de molécules à travers la surface lipidique d'une cellule. Ces membranes forment des canaux permettant le passage de molécules à l'intérieur d'une cellule. Les mécanismes de ces membranes sont très différents les uns des autres, certains n'étant pas encore découverts. Le canal du FepA sert de passage à un complexe de fer associé à une structure bactérienne appelée Enterobactin à travers une couche lipidique (voir Figure 4.2). À l'heure actuelle, le mécanisme précis permettant le passage du complexe de fer à travers le canal n'est pas encore connu. En effet, des structures secondaires situées à l'intérieur du canal devraient empêcher le complexe de fer de traverser le canal.

Une hypothèse formulée pour expliquer le passage du complexe de fer serait que le complexe provoquerait des changements de conformation à l'intérieur du canal pour passer. Des travaux récents[36] ont identifié deux passages où des molécules d'eau pouvaient s'infiltrer grâce à de légers réarrangements structuraux. Le complexe de fer

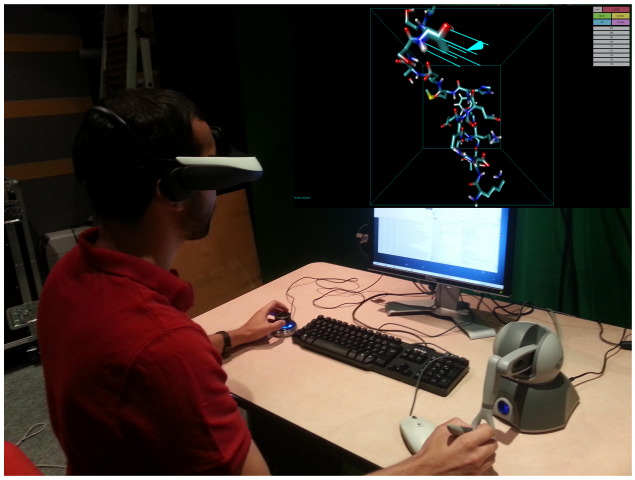


Figure 4.1 – Exemple d'utilisateur manipulant une simulation. Un visualiseur donne une image 3D de l'état courant de la simulation. En se basant sur cette information, l'utilisateur peut adapter le groupe d'atomes à guider ainsi que la direction et l'intensité des forces à appliquer.

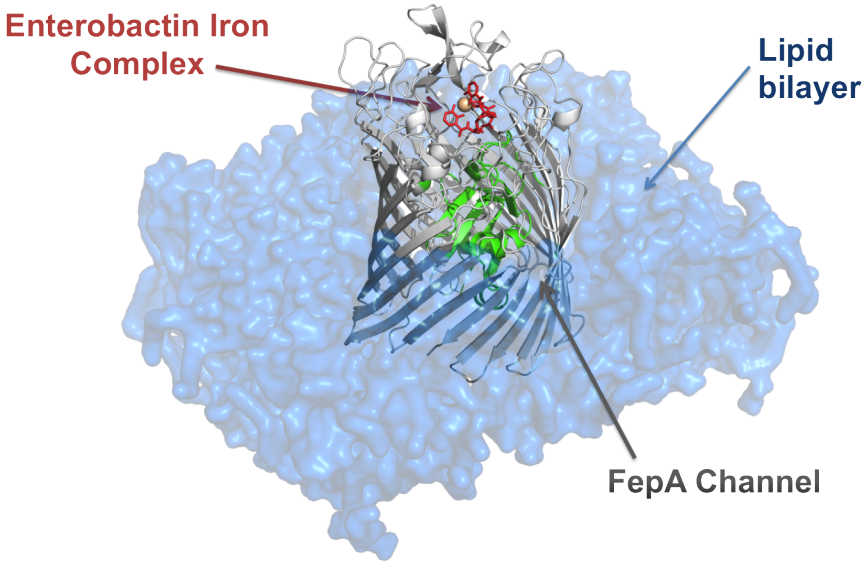


Figure 4.2 – Structure du canal du FepA. Ce canal (structure grise) permet le passage d'un complexe de fer (structure rouge) à travers la couche lipidique (structure bleue). Toutefois, des structures secondaires (vertes) semblent obstruer le passage dans le canal.

étant bien plus gros, il n'est pas clair que celui-ci puisse suivre les mêmes passages. Il s'agit d'un problème qui peut être traité par les techniques IMDs en guidant le complexe de fer vers les passages identifiés.

4.3 États des systèmes IMD actuels

Malgré les possibilités apportées par les systèmes IMD, leurs utilisations restent cantonnées à des démonstrations ou des applications éducatives. Une des raisons possibles est la difficulté du passage à l'échelle vers les grands systèmes moléculaires. Pour permettre une utilisation plus large des techniques IMDs, il est nécessaire de pouvoir manipuler les systèmes moléculaires étudiés actuels dont les ordres de grandeur sont de plusieurs centaines de milliers à plusieurs millions d'atomes. Mais l'utilisation de simulations à grande échelle, nécessaire pour traiter ces problèmes actuels, pose des difficultés particulières pour mettre un oeuvre un système IMD. À notre connaissance, les plus grands systèmes moléculaires manipulés intéractivement sont composés de quelques dizaines de millier d'atomes.

Le système d'IMD le plus connu est sans doute le plugin VMD[76]. Ce plugin définit un protocole de communication entre entre une simulation et un outil de visualisation. Usuellement l'outil de visualisation VMD et le logiciel de simulation NAMD sont utilisés ensembles. Toutefois le protocole de communication peut être utilisé par d'autres simulations ou visualiseurs. Gromacs dispose par exemple d'un plugin implémentant cette interface de communication[2]. MDDriver[27] propose une bibliothèque légère implémentant un protocole d'IMD pour coupler une simulation de dynamique moléculaire à un visualiseur. Cependant ces protocoles de communication sont définis entre deux processus. La composante parallèle de la simulation est donc totalement ignorée. Il est à la charge de la simulation de regrouper les atomes avant de pouvoir utiliser ce protocole de communication. Comme décrit dans la section 3.3.2, ce type d'approche ne passe pas à l'échelle. MolDRIVE[46] propose un environnement virtuel pour interagir et guider une simulation. La simulation est exécutée sur une machine Cray. À chaque itération, chaque nœud envoie la position des atomes à un serveur distant via TCP/IP. Ces données sont ensuite transmises à un client chargé d'effectuer la visualisation et de gérer les événements utilisateur. Le passage à l'échelle de ce système semble toutefois limité à huit nœuds à cause des communications réseaux supplémentaires d'extractions des données pour la visualisation. Ceci limite fortement la taille des systèmes moléculaires possibles.

Notre contribution s'articule autour du lien entre la visualisation et la simulation, principale cause limitant le passage à l'échelle des solutions actuelles. Pour passer à l'échelle, il est nécessaire de tenir compte du parallélisme de la simulation. Notre approche se distingue des systèmes classiques d'IMD par notre méthode d'extraction et de transmission des données. Nous proposons d'extraire les données au plus proche de leurs sources, au niveau de chaque processus, et de façon asynchrone afin de limiter au maximum notre impact sur la simulation.

```
while step < nstep && wait() do
    dd_partition_system() ;
    do_force() ;
    user_forces ← get(forces_port) ;
    if user_forces ∈ home_atoms then
        add_user_forces(user_forces) ;
    end
    if traj_enable then
        write_traj() ;
    end
    position_message ← home_atoms ;
    put(position_message) ;
    update() ;
    step++ ;
end
```

Algorithm 2: Boucle principale de Gromacs modifiée avec instrumentation de FlowVR pour une application IMD.

4.4 Modification du code de Gromacs

Dans la section 3.3.3, nous avons présenté notre méthode d'instrumentation du code de Gromacs. Grâce à une méthode d'extraction de données asynchrone, nous pouvons espérer un faible impact sur les performances de la simulation.

L'Algorithme 2 résume l'ensemble des modifications faites sur le code de Gromacs pour l'IMD. Pour tenir compte des contraintes d'interactivité, nous extrayons les données de la simulation à chaque itération. De plus, des ports d'entrée sont ajoutés sur chaque module afin de recevoir les forces utilisateurs. Enfin la fonction de Gromacs *do_force*, qui effectue le calcul des forces pour l'ensemble des atomes, est modifiée pour ajouter les forces utilisateurs dans le calcul des forces de la simulation. Les forces reçues correspondent à l'ensemble des forces utilisateurs. Dans un contexte distribué, un processus n'a généralement pas l'ensemble des atomes correspondant aux forces reçues dans son domaine. Par conséquent, nous vérifions pour chaque force reçue si l'atome associé à cette force appartient au domaine local. L'ensemble des modifications sur le code de Gromacs représente environ 50 lignes. Enfin, il est important de noter que la seule fonction bloquante dans cet algorithme modifié est Wait(). Dans notre cas d'utilisation, la fonction Wait() se débloque lorsque le module reçoit les forces utilisateurs.

Contrairement aux autres systèmes IMD, les positions des atomes sont extraites de la simulation au niveau de chaque processus de la simulation. Cela signifie que les positions des atomes sont distribuées sur l'ensemble des nœuds. Pour pouvoir associer une position à un atome, un index est associé à chaque position d'atome

correspondant à son index global dans la molécule. Cet index permet de faire le lien entre une position d'atome et ses caractéristiques (nom de l'atome, numéro de résidu associé, poids, etc...).

4.5 Gestion des communications en entrée et en sortie de la simulation

Les positions des atomes doivent être transmises au visualiseur. Le visualiseur attend en entrée un buffer contenant uniquement les positions des atomes ordonnées selon l'index global. Présentées de cette manière, les positions peuvent immédiatement être transférées sur le GPU pour être rendues. Il serait possible de conserver les données dans leur état initial, c'est-à-dire désordonnées et avec les index associés mais au prix d'un impact négatif sur les performances de rendu. Il nous faut donc regrouper l'ensemble des positions d'atomes en un seul message, réordonner les positions des atomes et enfin supprimer les index des atomes des messages. La Figure 4.3 représente le pipeline de modules pour transmettre les données de la simulation vers le visualiseur.

La première étape s'effectue au niveau de chaque nœud. Les données locales sont regroupées pour ne former qu'un seul message. À ce niveau, aucune communication réseau n'est nécessaire, tous les échanges se font par mémoire partagée. Nous utilisons le mécanisme des segments de FlowVR pour chaîner les messages produits par les processus locaux de la simulation évitant ainsi toute copie.

Dans un deuxième temps nous regroupons l'ensemble des messages à l'aide du schéma de communication N-To-1 où N correspond au nombre de nœuds hébergeant la simulation. L'opérateur utilisé au niveau de chaque nœud de l'arbre de communication est un filtre *merge* segmenté. Les *buffers* des messages reçus par ce filtre sont simplement chaînés sans copie pour ne former qu'un message. Une fois cette étape terminée, l'ensemble des positions des atomes sont regroupées avec leurs index respectifs.

La troisième étape consiste à réordonner l'ensemble des atomes et à supprimer l'ensemble des index. Ces opérations sont effectuées en même temps avec une complexité linéaire. Le message résultant est constitué d'un seul *buffer* non segmenté avec les positions des atomes triées dans l'ordre donné par l'index global. Ces opérations sont effectuées par un seul module.

L'envoi des forces est effectué par un schéma de communication 1-To-N. À chaque nœud du schéma de communication, la référence du message est dupliquée pour émettre au total N messages avec N le nombre de processus MPI de la simulation. En pratique, le message est dupliqué P fois avec P le nombre de nœuds hébergeant la simulation. Au niveau d'un nœud, la référence est dupliquée N/P fois mais toutes les références pointent sur le même *buffer* de données. Pour éviter ces duplications, on peut imaginer un système d'aiguillage pour amener chaque force directement au

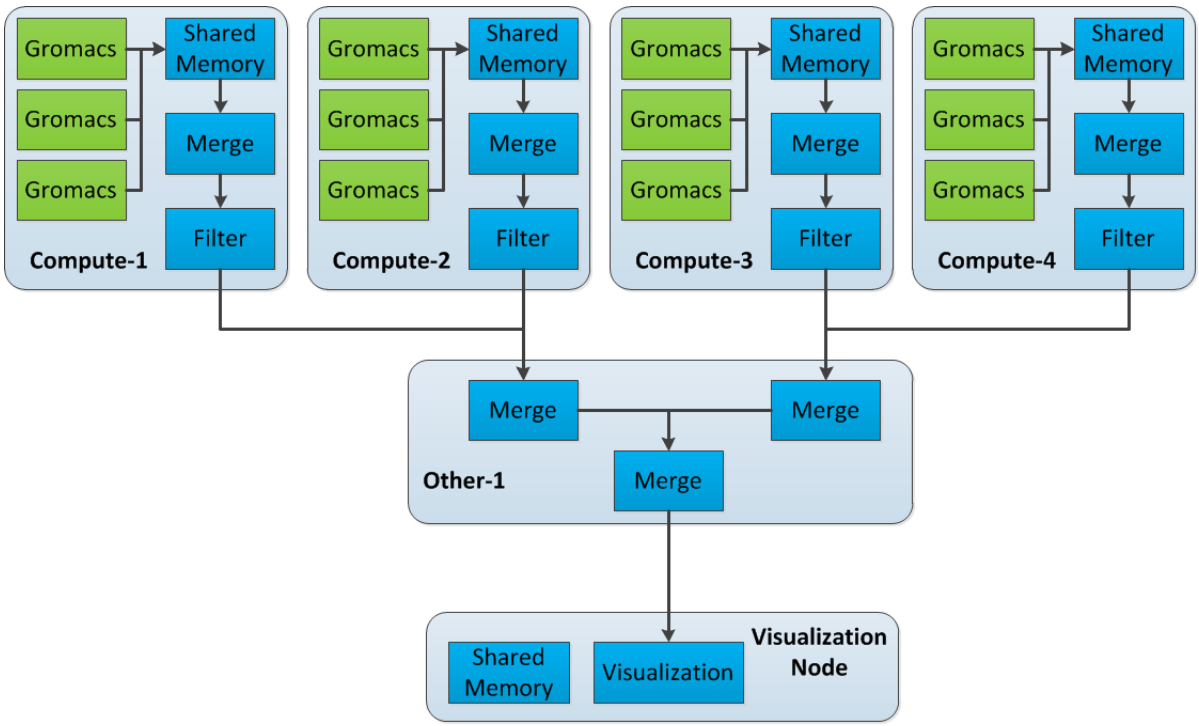


Figure 4.3 – Pipeline de traitement des positions des atomes de la simulation vers le visualiseur. Les positions des atomes sont d'abord rassemblées en un seul message au niveau de chaque nœud (sans copie). Puis un arbre de communication N-to-1 rassemble (soit sur les même noeud que la simulation, soit sur d'autres) les positions des atomes en un seul message et les envoie vers le nœud de visualisation. Enfin, les positions des atomes sont réordonnées et les index retirés avant d'être envoyés au visualiseur.

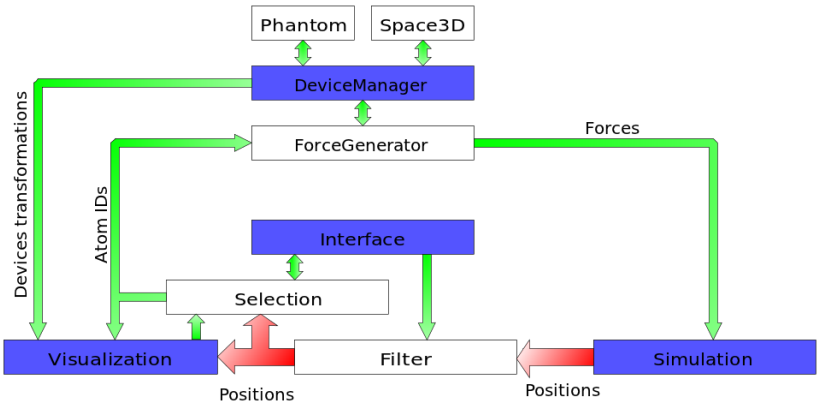


Figure 4.4 – Représentation synthétique de notre application d'IMD. Elle est composée de quatre composants principaux (boîtes bleues) avec quelques modules secondaires (boîtes blanches). Les flèches rouges représentent les principaux flux de données produits par la simulation. Les flèches vertes représentent des communications légères entre modules.

bon processus de la simulation. Un tel système serait toutefois compliqué à mettre en place et introduirait de la latence car les aiguillages devraient être mis à jour en même temps que la simulation. En effet la simulation peut modifier la répartition des atomes à n'importe quel moment. Dans la mesure où les messages contenant les forces sont petits (moins d'1Ko), le coût de la duplication est négligeable et permet de simplifier grandement l'acheminent des forces vers la simulation.

Le reste de notre système IMD est relativement classique. Nous utilisons un algorithme de visualisation spécifique appelé Hyperball[25] permettant d'obtenir des performances interactives de visualisation sur des systèmes moléculaires de plusieurs centaines de millier d'atomes. Afin de garantir une désolidarisation complète entre la visualisation et la simulation, nous effectuons un échantillonnage asynchrone des données (composant *Greedy*) après que les positions d'atomes soient triées. Celui-ci élimine le surplus de messages si la simulation a une fréquence supérieure à la visualisation ou au contraire renvoie le dernier message reçu si la simulation avance plus lentement. La Figure 4.4 présente une vision synthétique du réseau global de l'application.

4.6 Modulation de notre impact sur les performances de la simulation grâce au filtrage in-situ

Un besoin qui apparaît très vite pour les biologistes est la possibilité de filtrer certains groupes d'atomes. Une requête courante est le filtrage de l'eau qui peut

représenter jusqu'à 80% du nombre total d'atomes et occulter totalement la vision du système moléculaire. Bien entendu, il est possible de filtrer directement les atomes au niveau de la visualisation. Cela signifie toutefois qu'une partie des atomes a été transmise à la visualisation inutilement. Les communications réseaux générées pour rassembler les atomes entrent en concurrence directe avec les communications de la simulation. Pour minimiser l'impact sur la simulation, il est important de minimiser autant que possible ces communications supplémentaires.

Les traitements in-situ permettent de filtrer les atomes au niveau de chaque nœud avant qu'ils ne soient envoyés sur le réseau. Nous pouvons ainsi envoyer sur le réseau uniquement les données qui sont intéressantes pour la visualisation. Ce filtrage est effectué juste après avoir rassemblé les positions des atomes au niveau de chaque nœud.

Toujours dans le même objectif d'économie de bande-passante, il n'est pas forcément nécessaire d'envoyer sur le réseau toutes les itérations de la simulation. En effet, avec une puissance de calcul suffisante, il est possible et même courant que la simulation produise des données bien plus rapidement que le rendu n'est capable de les afficher. Un composant *Greedy* est placé avant le visualiseur pour éviter toute accumulation de données. Celui-ci peut supprimer des messages provenant de la simulation. Là encore, des données sont donc envoyées inutilement sur le réseau. Nous avons donc ajouté un module supplémentaire appelé *FilterIteration* avant le filtrage pour éliminer régulièrement certaines itérations. Ainsi seule une itération toutes les x itérations, avec x une valeur définie par l'utilisateur, est transmise au module de filtrage puis envoyée sur le réseau. Avec ce système, certaines itérations peuvent toujours être supprimées par le *Greedy*. Toutefois, cela permet déjà de limiter grandement le nombre d'envois inutiles sur le réseau avec une valeur de x appropriée. La Figure 4.5 résume l'ensemble du pipeline de traitement.

4.7 Expérimentations et résultats

Le nœud de visualisation est composé de deux CPU Xeon E5530 quadri-cœurs cadencés à 2.40GHz et une carte graphique GeForce 680 GTX. Le cluster de calcul est composé de 72 nœuds, chacun équipé de deux CPU Xeon E5520 quadri-cœurs cadencés à 2.27GHz, 24GB RAM. Les nœuds du cluster de calcul sont interconnectés par un réseau InfiniBand QDR (40Gbits/s). La liaison entre le cluster de calcul et le nœud de visualisation est assurée par un réseau InfiniBand DDR (20Gbits/s).

Pour placer les processus, nous avons testés différentes stratégies. La stratégie donnant les meilleurs performances pour cette application est la suivante :

— sept processus MPI de Gromacs sont instanciés par nœud sur le cluster de calcul (huit cœurs par nœud)

— Le premier *merge* rassemblant les atomes au niveau de chaque nœud ainsi que les modules de filtrage sont placés sur le 8ème cœur (*helper core*) de chaque nœud du cluster de calcul

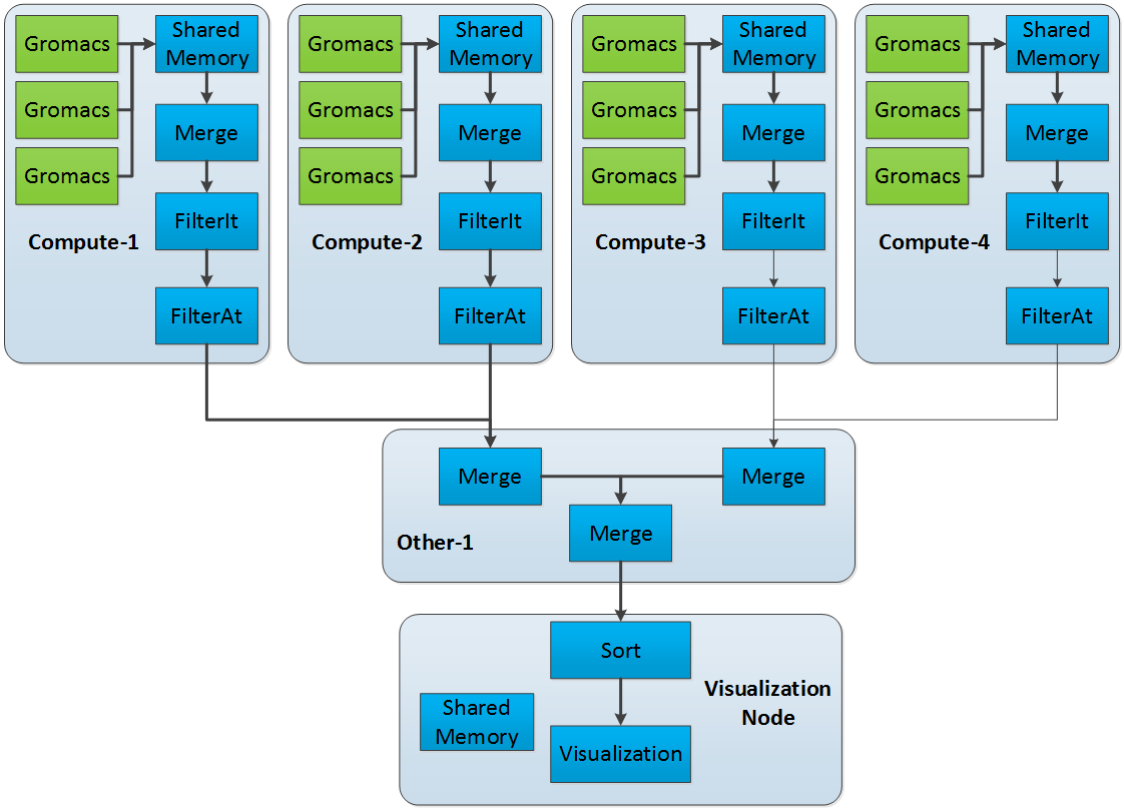


Figure 4.5 – Pipeline complet de traitement. Les données sont d'abord rassemblées localement. Les positions des atomes sont ensuite filtrées par itérations et puis par numéros d'atomes. Enfin, les données sont rassemblées en un seul message puis réorganisées pour la visualisation.

— L'arbre rassemblant les positions des atomes est hébergé sur des nœuds dédiés à raison d'un nœud dédié pour un noeud de calcul.
— Le module de tri des atomes ainsi que le reste de l'application (visualiseur, gestion des périphériques,...) sont hébergés sur le nœud de visualisation.

Nous utilisons donc la stratégie du cœur dédié sur le cluster de calcul ainsi que des nœuds dédiés pour y décharger les communications. Cette approche nous permet de minimiser l'impact à la fois sur le CPU et sur le réseau. En choisissant la stratégie du cœur dédié, nous limitons au maximum les contentions sur les cœurs de la simulation. Les simulations de dynamique moléculaire ayant une fréquence d'itération très élevée (plusieurs centaines voir milliers d'itérations par seconde), il est important d'éviter que nos traitements in-situ causent des interruptions sur les processus de la simulation. L'utilisation des nœuds dédiés nous permet de considérablement réduire le trafic réseau sur les nœuds hébergeant la simulation au prix de l'allocation de nœuds supplémentaires. En effet, par rapport à une utilisation normale de Gromacs sans IMD, nous ne générons qu'un message supplémentaire par nœud et par itération de sortie. Sans les *staging nodes*, l'arbre rassemblant les données serait hébergé sur les mêmes nœuds que la simulation et générerait des accès concurrents sur les cartes réseaux bien plus significatifs.

L'application a été testée grâce à des modèles de benchmarks générés de façon régulière. Le modèle du Glic initialement constitué de 150000 a été dupliqué un certain nombre de fois pour former des modèles moléculaires de taille arbitraire. Bien que les modèles obtenus ne soient pas réalistes biologiquement parlant, les calculs effectués notamment vis à vis du calcul des interactions longues distances (PME) sont représentatifs d'une simulation classique. Les systèmes moléculaires obtenus sont composés de 500000 (4 membranes Glic) à 1700000 atomes (12 membranes Glic).

La Figure 4.6 présente les résultats pour les quatre modèles de benchmarks cités précédemment. Pour l'ensemble des modèles, des vitesses interactives sont maintenues à la fois pour la visualisation et la simulation. Dans le pire des cas, nous avons relevé un impact de 17% sur les performances de la simulation en utilisant un modèle composé de 1.1 million d'atomes et 64 nœuds pour la simulation en extrayant l'ensemble des données à chaque itération. Sur 17%, 4% sont dues aux traitements locaux et 13% à cause des contentions réseaux. Les perturbations locales regroupent les temps de copie des positions des atomes, l'appel à Wait() ainsi que l'impact sur le cache du processeur. La stratégie du cœur dédié permet néanmoins d'obtenir un coût relativement faible sur la simulation. En revanche, le transfert des données semble bien plus coûteux malgré l'utilisation de nœuds dédiés. Compte tenu de la vitesse importante de la simulation, il est très probable que les communications vers les nœuds dédiés perturbent les communications internes de la simulation. Une approche similaire à DataStager notamment pour effectuer les transferts de données aux moments opportuns serait sans doute bénéfique. Les mécanismes de filtrage mis en place donnent néanmoins des leviers pour moduler ce coût. Finalement, 17% reste

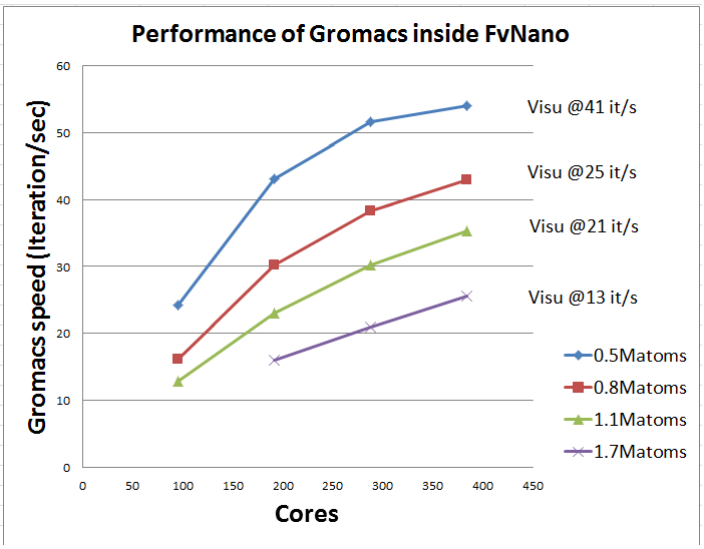


Figure 4.6 – Vitesse de la simulation et de la visualisation pour quatre modèles de test et différents nombres de cœurs. Pour l'ensemble des modèles, des vitesses interactives sont maintenues à la fois pour la visualisation et la simulation.

un coût global raisonnable pour une application interactive.

4.8 Résultats d'expérience sur le FepA

Une hypothèse pouvant expliquer le passage du complexe de fer dans le canal FepÀ est que celui-ci provoque le réarrangement de structures secondaires pour se créer un passage. En particulier, deux passages suffisamment grands ont pu être identifiés pour que de l'eau puisse s'y infiltrer. Ces passages forment des candidats intéressants pour le complexe de fer. Les systèmes de guidage (SMD et IMD) offrent une plateforme idéale pour tester ces passages. Le principe de l'expérience consiste à appliquer des forces sur le complexe de fer pour le tirer vers les passages identifiés pour l'eau. Les passages sont toutefois très sinueux. Il serait donc très difficile de décrire les forces à appliquer dans un contexte SMD. De plus, nous nous attendons à ce que le canal subisse des changements structuraux rendant encore plus difficile la description de ces forces. Un système interactif est donc préférable pour ajuster en temps réel la direction des forces à appliquer.

Le modèle utilisé pour simuler le FepÀ est composé d'environ 72000 atomes. Il contient la protéine du FepA, une portion de la couche lipidique, le complexe de fer ainsi que le solvant. La simulation a été exécutée sur le cluster de calcul présenté dans la section précédente avec 128 cœurs ainsi que le nœud de visualisation. Un biologiste est venu sur site pour effectuer la manipulation. L'œil expert d'un biologiste sur le système moléculaire s'est avéré vital pour identifier les deux passages possibles dans

la structure du canal. Ces expérimentations nous ont permis de confirmer la présence des deux passages trouvés pour le cas de l'eau. De légère déformations structurales sur des structures secondaires flexibles ont toutefois été provoquées par le passage du complexe de fer sous l'impulsion des forces utilisateurs.

Avec ce nombre de cœurs, la simulation s'exécutait à une vitesse de 230 Hz. Il est possible de monter cette vitesse à 250Hz environ en filtrant les atomes non nécessaires à la manipulation et en n'envoyant qu'une itération sur dix sur le réseau. Cette configuration est suffisante pour ne pas gêner l'utilisateur dans sa manipulation. La visualisation après filtrage de l'eau et de la membrane lipidique avait une vitesse de rendu de 80 fps [1] environ. Le temps total de la manipulation était de cinq minutes environ.

Le principe de l'IMD est d'appliquer des forces extérieures à une simulation. L'utilisateur ajoute donc de l'énergie dans le système pour favoriser l'apparition d'un événement souhaité. Cette méthode fait débat dans la communauté biologique car elle biaise le réalisme de la simulation en introduisant des forces qui n'ont pas une signification biologique. Il est donc important de minimiser ce biais autant que possible. Pour notre expérience du FepA, nous avons appliqué une force globale de 2000pN par itération répartie sur l'ensemble des atomes. Un niveau de force moins élevé ne nous a pas permis de finir la manipulation interactivement. En comparaison, les études menées en biologie avec des systèmes SMD, qui sont des simulations bien plus longues, utilisent des niveaux de force de 100 à 500 pN. Le niveau de force que nous avons utilisé est donc pour le moment bien trop important pour que notre expérimentation puisse être acceptée et validée par la communauté. Des expérimentations supplémentaires sont nécessaires avec un niveau de force moins élevé. Il s'agit néanmoins d'un premier pas important vers la validation des deux passages identifiés.

4.9 Discussion

4.9.1 Utilité et validité des systèmes IMD à grande échelle

Les systèmes IMD présentent deux avantages majeurs pour les biologistes : ils permettent une intégration de l'expert dans le processus de guidage et le coût supplémentaire en calcul lié à la génération des forces extérieures est faible. Ces deux avantages sont également des faiblesses. En intégrant l'utilisateur au processus de guidage, nous limitons le temps que peut durer la simulation. On imagine facilement qu'au delà de 15 minutes l'utilisateur s'ennuie. L'événement simulé doit être suffisamment court pour tenir sur cette session. Le guidage par l'utilisateur du système moléculaire a pour but d'accélérer cet événement mais souvent de manière insuffisante. Pour accélérer encore davantage le processus biologique il faut soit utiliser

1. frame per second

plus de puissance de calcul soit augmenter le niveau de force appliqué. L'augmentation de la puissance de calcul pose le problème à la fois du coût en ressource de la simulation ainsi que du passage à l'échelle du système. De l'autre coté, augmenter le niveau de force appliqué réduirait la crédibilité de la simulation.

Un événement biologique peut durer plusieurs microsecondes. En supposant une simulation avec un pas d'intégration de deux femtosecondes et une vitesse de simulation de 1000Hz, nous pouvons simuler deux nanosecondes en 15 minutes. Cela représente seulement un millième de ce que l'on voudrait pouvoir simuler interactivement. L'apport de puissance de calcul supplémentaire ne permet pas de combler ce fossé. En effet, compte tenu de la latence des réseaux, il est très difficile pour une simulation d'aller au delà de 5000Hz. De même, augmenter le niveau de force introduit dans la simulation impacterait directement la validité biologique de l'expérience.

D'autre part, l'ajout de forces extérieures n'est pas nécessairement une méthode physiquement robuste. D'autres méthodes comme l'*umbrella sampling*[83] utilisé notamment dans des systèmes SMDs sont physiquement plus réalistes. Au cours d'une simulation, certaines configurations peuvent nécessiter plus d'énergie que d'autres pour être atteintes et sont donc moins échantillonnées par la simulation. L'*umbrella sampling* permet d'équilibrer l'échantillonnage des configurations possibles. Les simulations peuvent ainsi plus facilement passer au dessus de certaines barrières énergétiques et potentiellement favoriser l'apparition d'autres configurations. Le temps de calcul nécessaire pour cette méthode est toutefois nettement plus élevé qu'avec les systèmes IMD qui n'impliquent pas de calculs supplémentaires pour la simulation.

Pour autant, l'IMD reste une méthode très utile à des fins exploratoires notamment. En effet, l'IMD permet de tester rapidement différents scénarios et configurations. Une fois certains scénarios validés, il est alors possible d'utiliser des méthodes plus robustes (et plus longues) pour valider ou non ces scénarios. Il y a donc toujours un intérêt pour les utilisateurs de pouvoir manipuler des modèles moléculaires de plus en plus grands. L'échelle de temps reste tout de même un problème important. Nous proposerons dans le Chapitre 6 une solution permettant de tirer profit des expérimentations faites en IMD pour les intégrer dans une longue simulation.

4.9.2 Intérêts et limitations de l'interactif

Les simulations de dynamique moléculaire sont des codes particulièrement difficiles à intégrer dans une application interactive à cause de leur vitesse. Il n'est pas rare que des simulations atteignent 1000 itérations par seconde même pour de très grands modèles moléculaires. À cette vitesse, la moindre interruption impacte les performances de la simulation. Avec notre méthode d'instrumentation, nous sommes obligés à chaque itération d'interrompre la simulation (Wait()) pour attendre de nouvelles forces. Ce temps d'arrêt est relativement court ($0,06$ms en moyenne) mais suffisant pour être perceptible dans les performances de la simulation. De plus, les positions des atomes sont copiées également à chaque itération pour être transmises

au visualiseur. Ceci est nécessaire pour garantir le maximum de réactivité possible au système.

Nous avons placé les différents modules de façon à minimiser au maximum l'impact sur les performances de la simulation. Toutefois, le coût d'instrumentation de la simulation à lui seul est déjà élevé à cause des contraintes d'interactivité. De plus, l'envoi très fréquent des positions des atomes perturbe encore davantage les performances de la simulation. Dans la pire des configurations testées (modèle de 1.1 millions d'atomes sur 448 cœurs sans filtrage d'itérations ou d'atomes), le coût sur les performances de la simulation est de 17%. Nous avons toutefois vu que ce coût pouvait être réduit en jouant notamment sur la quantité et la fréquence des données envoyées.

Ce coût est acceptable pour des sessions interactives limitées dans le temps. En effet, il ne s'agit pas d'un facteur limitant pour l'utilisateur à partir du moment où des fréquences interactives sont maintenues. Pour une session de 15 minutes, ce coût signifie simplement que la manipulation de l'utilisateur sera rallongée de deux minutes ce qui reste acceptable.

Chapitre 5

Applications In-Situ à grande échelle

5.1 Introduction

Dans le chapitre précédent, nous avons présenté une application interactive autour du code de simulation Gromacs. L'utilisateur a la possibilité de visualiser l'état courant de la simulation et de la guider en appliquant des forces qui sont intégrées dans le calcul de la simulation. Grâce à une méthode de couplage asynchrone au niveau de chaque processus de la simulation, nous pouvons coupler ces fonctions interactives à Gromacs au prix d'un impact modéré sur les performances de la simulation (17% mesuré dans le pire des cas testés).

Toutefois, nous avons également vu que ce mode interactif ne peut pas s'appliquer à toutes les simulations massives même à l'ère de l'exascale. Trois facteurs expliquent cette impossibilité. Tout d'abord, la différence d'échelle de temps entre le temps de simulation et le temps simulé est une barrière infranchissable même à long terme. Pour simuler interactivement des événements biologiques intéressants, il faut augmenter la vitesse de simulation d'un facteur 1000. Ce facteur est malheureusement inatteignable notamment à cause de la faible amélioration des temps de latence sur les réseaux. Le deuxième facteur est lié au mode d'exécution des simulations de production. Sur les machines parallèles, un ordonnanceur est responsable de lancer la simulation lorsque suffisamment de ressources sont disponibles. De fait, il n'est pas possible de savoir à l'avance quand la simulation sera exécutée. De plus la simulation est exécutée en tâche de fond par le système de la machine parallèle rendant ainsi toutes les interactions avec l'utilisateur assez contraignantes. Ces contraintes d'environnement rendent difficile toute expérience interactive. Enfin le troisième point est lié à la consommation des ressources. Les grandes machines parallèles permettent d'exécuter des simulations sur plusieurs milliers voire dizaines de milliers de cœurs pendant plusieurs jours. À ces échelles de temps de simulation et de quantités de ressources, 17% d'impact sur les performances de la simulation peut représenter plusieurs dizaines de milliers d'heures supplémentaires nécessaires pour compléter une simulation. Il est alors légitime de se demander si un utilisateur est prêt à dépenser autant de ressources pour ces fonctionnalités supplémentaires.

Les traitements in-situ permettent de traiter les données lorsqu'elles sont produites en mémoire par la simulation évitant ainsi le goulot d'étranglement situé au

niveau des I/O vers le système de fichier. Initialement proposée pour réduire les données avant d'être écrites sur disque, la palette des opérations possibles est maintenant bien plus vaste. Ces traitements incluent de la réduction et de l'annotation de données (compression, filtrage, indexation), de la visualisation, des statistiques ou encore des analyses plus spécifiques à chaque domaine. L'objectif est à la fois de réduire le goulot d'étranglement au niveau des I/Os, de profiter des ressources disponibles sur les grandes machines parallèles pour effectuer des traitements gourmands en calcul habituellement faits en phase de post-traitement, de préparer les données en vue d'autres post-traitements et enfin d'améliorer l'usage global des machines parallèles en effectuant des traitements supplémentaires lorsque la simulation n'exploite pas complètement les ressources à disposition.

Les besoins entre les applications scientifiques interactives et les simulations de production classiques équipées de méthodes in-situ sont bien différents. Dans une application interactive, la priorité est donnée à la qualité des services attachés à l'application afin d'améliorer autant que possible l'expérience de l'utilisateur. Pour une simulation de production, la priorité est donnée à la minimisation de l'impact des services in-situ sur les performances de la simulation. Cependant, certaines fonctionnalités peuvent être partagées entre les deux contextes comme la visualisation en ligne. L'implémentation de ces fonctionnalités doit toutefois être adaptée suivant les contraintes (batch mode contre session interactive) et priorités de chaque contexte.

Dans ce chapitre, nous nous intéressons plus particulièrement au mode in-situ et ses applications en nous basant sur notre précédente expérience avec les applications interactives. Notre objectif premier est de montrer que notre approche permet d'avoir un faible impact sur les performances de la simulation tout en conservant un haut niveau de flexibilité dans la création d'applications in-situ. Pour tester notre approche, nous proposons deux scénarios d'applications in-situ centrés sur le code de simulation Gromacs.

Le premier scénario consiste à déporter l'écriture d'une trajectoire vers les services in-situ. La phase d'écriture d'une simulation constitue un temps pendant lequel la simulation ne calcule pas. Pour certaines simulations, ce délai d'écriture peut représenter un temps non négligeable du temps total d'exécution de la simulation. En déportant la phase d'écriture sur les services in-situ asynchrones, la simulation devient libre de poursuivre ses calculs permettant ainsi de finir la simulation plus rapidement dans certains cas.

Le deuxième scénario implémente un algorithme de visualisation. Pour visualiser de grandes molécules, une représentation entièrement atomistique n'est pas souhaitable. En effet, l'utilisateur se retrouve très vite noyé par le nombre de particules représentées à l'écran. Pour de grands ensembles moléculaires, une des informations importantes à visualiser est la surface des molécules. Celle-ci permet de mieux percevoir la forme 3D d'une molécule et ses changements de conformation comme la formation de poches par exemple. Le Quicksurf est un algorithme de calcul d'isosurface. Il permet la construction d'un maillage de triangles représentant la surface

d'un ensemble moléculaire. L'implémentation initiale du Quicksurf a été proposée pour le GPU[48]. Nous avons développé une version distribuée que nous avons divisée en plusieurs composants. Nous proposons différentes stratégies de placement des processus et de communication afin de tester quelle configuration impacte le moins les performances de la simulation.

L'ensemble des travaux présentés dans ce chapitre ont été publiés au *14th IEEEACM International Symposium on Cluster, Cloud and Grid Computing* (CC-Grid 2014).

5.2 Contribution

Dans le chapitre précédent, nous avons vu que l'envoi moins fréquent de données sur le réseau permettait déjà de diminuer notre impact sur les performances de la simulation. Le contexte in-situ nous permet encore d'aller plus loin dans notre approche.

5.2.1 Adaptation de la fréquence d'extraction de données

Dans le cadre de la dynamique moléculaire, il n'est pas rare d'effectuer une sauvegarde seulement toutes les 5000 itérations, voire plus. Ce choix est fait pour des raisons à la fois de performances (la simulation est bloquée pendant la phase d'écriture) et de taille de trajectoire.

Dans le chapitre précédent, nous avons utilisé un module au niveau de chaque nœud pour supprimer certaines itérations. Dans un contexte in-situ, il est également souhaitable de ne pas traiter toutes les itérations. On pourrait donc ici réutiliser le même composant que précédemment. Toutefois, cette approche implique qu'à chaque itération des données soient copiées dans la mémoire partagée de FlowVR puis envoyées vers le reste de l'application. Bien que relativement légères, ces opérations peuvent avoir un impact de plus en plus important sur les performances de la simulation au fur et à mesure que celle-ci monte en fréquence. Dans un contexte in-situ, nous avons besoin d'extraire les données uniquement lorsque des traitements sont nécessaires. Nous avons rajouté une option dans le code de Gromacs pour indiquer la fréquence d'extraction des données souhaitée.

5.2.2 Plateforme d'expérimentation

L'ensemble des expérimentations présentées dans cette section sont effectuées sur la machine parallèle Froggy. Froggy fait partie de l'infrastructure du meso-centre Grenoblois Ciment. Elle est composée de 138 nœuds. Chaque nœud est équipé de deux processeurs octocœurs Sandy Bridge-EP E5-2670 cadencés à 2.6GHz, 64 GB de mémoire vive. Les nœuds sont interconnectés par un réseau Infiniband FDR (60GB/s théorique). FlowVR 2.1 et Gromacs 4.6 sont compilés avec Intel MPI 4.1.0.

La simulation de test utilisée est une simulation Martini (simulation à gros grain) composée d'un amas de 54000 lipides avec du solvant pour un total d'environ 2100000 particules. OpenMP n'est pas utilisé pour des raisons de performance.

5.2.3 Exemple 1 : écriture de trajectoires en in-situ

La phase d'écriture est un passage obligatoire de tout code de simulation. Il existe une variété extrêmement vaste de méthodes d'écriture et de format de fichiers. Certaines sont plus adaptées que d'autres aux exécutions en mode distribué. Gromacs a fait le choix d'écrire les données à partir d'un processus unique : le maître. À chaque itération d'écriture, les données sont rassemblées sur le processus maître de façon synchrone puis le maître effectue l'écriture. Pendant ce temps, la simulation est bloquée jusqu'à la fin de l'opération d'écriture.

Cette phase bloquante peut représenter une part importante du temps total d'exécution. À haute fréquence d'écriture, il peut s'agir d'un goulot d'étranglement majeur empêchant le passage à l'échelle de la simulation. La Figure 5.1 présente la vitesse de Gromacs en fréquence avec différentes configurations de méthodes et de fréquences d'écriture en fonction du nombre de nœuds. Pour rappel, un nœud est composé de 16 cœurs. La colonne 1 (bleu foncé) représente les performances de Gromacs utilisant tous les cœurs disponibles et sans effectuer d'écriture. Cette colonne nous donne la vitesse maximale que Gromacs peut atteindre. La colonne 3 (jaune) représente les performances de Gromacs utilisant tous les cœurs disponibles et écrivant des données toutes les 100 itérations. Entre ces deux colonnes nous observons une baisse de 77% des performances de Gromacs à 2048 cœurs imputable à la phase d'écriture. Pour les deux cas, il s'agit du code de Gromacs, sans aucune modification, qui est utilisé. Cette baisse très importante des performances est un phénomène connu et qui est généralement masqué par une fréquence d'écriture habituelle beaucoup plus faible de sauvegarde des données.

Notre objectif est de déporter cette opération bloquante vers l'infrastructure in-situ. Pour cela, nous utilisons la stratégie du cœur dédié. Un cœur par nœud est réservé pour les traitements in-situ. Gromacs n'utilisera donc que 15 cœurs par nœud. Cela représente dans notre cas 6.25% de ressources de calcul en moins pour la simulation. Les colonnes 2 (rouge) et 4 (verte) présentent les même résultats que précédemment mais en utilisant seulement 15 cœurs par nœud. À 2048, les performance de Gromacs ne baissent que de 2.6% en retirant un cœur par nœud. Deux facteurs peuvent expliquer ce résultat. Tout d'abord les simulations de dynamique moléculaire ne passent pas à l'échelle linéairement pour un problème donné. Enfin le changement de nombre de cœurs influe sur le système de décomposition de domaine de Gromacs et en particulier de son système d'équilibrage de charge. Par conséquent, la décomposition du domaine en utilisant 15 cœurs peut être plus favorable. En effet, dans certains cas l'indice de déséquilibrage de la simulation (valeur donnée par Gromacs) est plus faible dans la configuration utilisant 15 cœurs. Ce résultat est tou-

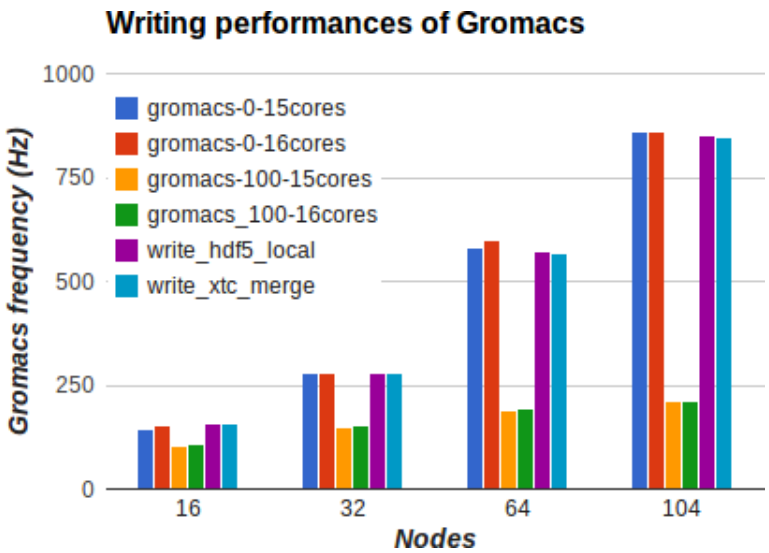


Figure 5.1 – Évolution de la fréquence de Gromacs avec différentes méthodes et fréquences d'écriture.

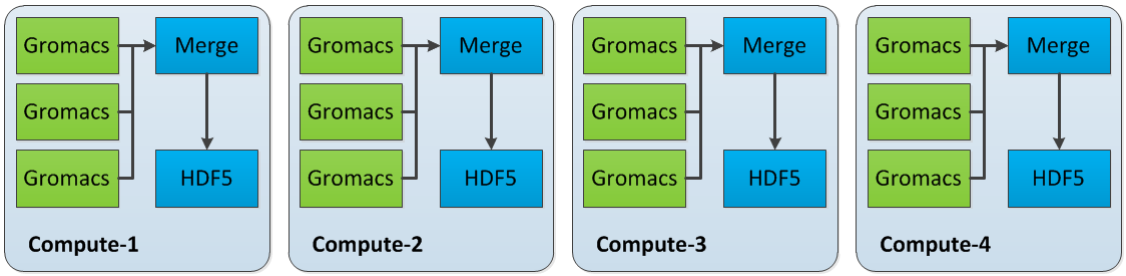


Figure 5.2 – Graphe du premier exemple d'écriture in-situ. Une fois extraites, les données sont d'abord rassemblées par nœud avant d'être envoyées vers un module d'écriture. Les données sont écrites au format HDF5.

tefois spécifique pour un nombre de cœurs et un système moléculaire donné. Cette faible perte de performance de Gromacs permet de libérer une puissance de calcul considérable sur chaque nœud disponible pour effectuer des traitements in-situ.

Notre première méthode d'écriture consiste à écrire un fichier par nœud au format HDF5. La Figure-5.2 présente le graphe de l'application. Une fois extraites de la simulation, les données sont d'abord rassemblées par nœud avant d'être envoyées à un module d'écriture hébergé sur le cœur dédié. Cette configuration devrait perturber le moins la simulation car l'application ne génère pas de communications réseaux supplémentaires à part celles vers le système de fichier. Comme pour Gromacs avec l'écriture activée, nous écrivons 1 fois toutes les 100 itérations de la simulation.

La colonne 5 (violet) présente les résultats obtenus avec cette configuration. À 2048 cœurs, l'écriture de fichier ne coûte que 3,6% par rapport à Gromacs ne faisant pas d'écriture et utilisant 15 cœurs. Par rapport à Gromacs utilisant 16 cœurs, les

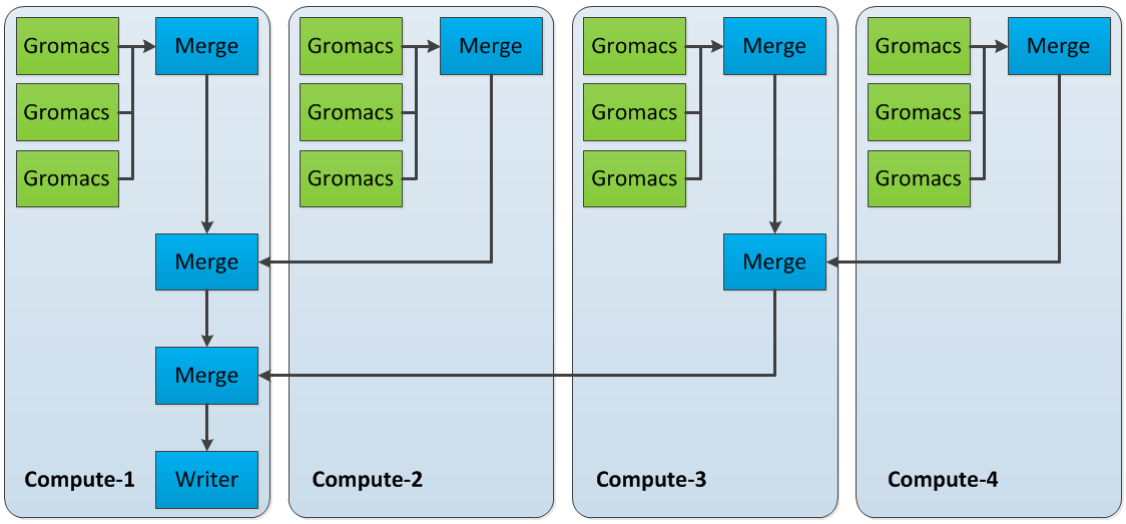


Figure 5.3 – Graphe du deuxième exemple d'écriture in-situ. Une fois extraites, les données sont d'abord rassemblées par nœud puis un arbre d'agrégation va rassembler les donner vers le nœud de rang 0. Une fois rassemblées, les données sont transmises à un module d'écriture.

performances ne sont impactées que de 6,25%. Ces coûts s'expliquent d'une part par le coût d'extraction des données et d'autre part les perturbations réseaux (écriture sur un système de fichier parallèle Lustre) et les contentions sur le cache occasionnées par les traitements in-situ. Ces coûts sont largement inférieurs aux 77% de coût avec la méthode native d'écriture de Gromacs.

Notre deuxième méthode d'écriture en in-situ vise à produire le même fichier que ce que Gromacs produit. Nous utilisons donc le format de compression xtc. Il n'existe pas à notre connaissance d'implémentation d'une méthode d'écriture parallèle pour ce format de fichier. Les données doivent donc être rassemblées avant d'être écrites, comme le fait Gromacs nativement. À la différence de Gromacs, nous rassemblons les données de manière asynchrone en dehors de la simulation. La Figure 5.3 présente le graphe de notre application. Les données sont d'abord rassemblées une première fois par nœud puis un arbre d'agrégation va rassembler l'ensemble des données vers le nœud de rang 0. Enfin les données sont transmises à un module d'écriture.

La colone 6 (bleu clair) présente les résultats obtenus par cette configuration. À 2048 cœurs, l'écriture des fichiers coûte 6,5% par rapport à Gromacs utilisant 15 cœurs sans écriture et 9% par rapport à Gromacs utilisant les 16 cœurs disponibles sans écriture. Cette méthode est plus coûteuse que la précédente. Cela s'explique par les communications réseaux supplémentaires nécessaires au rassemblement des données qui entrent en concurrence directe avec les communications de la simulation. Quand bien même ce coût est un peu plus important que celui de la première méthode, il reste largement inférieur à celui de la méthode native de Gromacs tout en produisant des données exploitables. Nous verrons dans le chapitre suivant comment

tirer partie de cette approche d'écriture pour des scénarios biologiques réalistes.

Notons que cette amélioration très significative des performances d'écriture est possible car Gromacs utilise une méthode synchrone d'écriture passant assez mal à l'échelle. D'autres méthodes (MPI-IO, PNetCDF, PHDF5) plus appropriées permettraient de gagner en rapidité lors des phases d'écriture. Nous avons par exemple vu que la méthode d'écriture par fichiers HDF5 donnait de meilleurs performances en évitant de coûteuses communications réseaux. Malheureusement ce format de fichier n'est pas un standard dans la communauté de la dynamique moléculaire.

5.2.4 Exemple 2 : génération d'un maillage en in-situ et in-transit

Une application courante faite en visualisation in-situ consiste à générer des images de l'état courant de la simulation[90]. En dynamique moléculaire toutefois, une image présente un intérêt assez limité. En effet, une perception 3D du système moléculaire étudié est indispensable pour bien en appréhender son comportement. Plutôt que de générer une image, nous proposons de générer in-situ un maillage représentant la surface d'une molécule. Ce maillage permet ainsi aux biologistes de pouvoir manipuler librement un modèle 3D de la molécule.

Différentes méthodes existent pour définir et générer une telle surface. Une des méthodes consiste à définir la surface accessible par le solvant (SAS)[70]. Cette surface est généralement calculée en utilisant l'algorithme de la *rolling ball*. La Figure 5.4 donne une vue 2D de la génération de ces surfaces. Le principe consiste à faire rouler une sphère sur la molécule. Le point de contact entre la sphère et l'atome définit la surface exclue par le solvant. La position du centre de la sphère au moment du contact entre la sphère et les atomes de la molécule constitue la surface accessible par le solvant.

Définie de cette manière, la surface moléculaire est très précise dans le sens où elle épouse au plus proche la molécule. Toutefois son coût en calcul est assez élevé. Krone et al.[47] proposent une implémentation pour GPU en modifiant légèrement l'algorithme initial. Même avec des GPUs actuels, il n'est malheureusement pas possible de visualiser des systèmes moléculaires de plus de quelques dizaines de millier d'atomes.

Pour résoudre ce problème de performance, Krone et al[48] proposent une autre méthode de génération de surface moléculaire appelée Quicksurf. Cet algorithme se décompose en trois étapes :

— La scène est plongée dans une grille 3D régulière
— Pour chaque cellule de la grille, une densité est calculée par une gaussienne en fonction du nombre d'atomes dans et autour de la cellule et de leurs diamètres.

2. Source Figure 5.4 : [47]

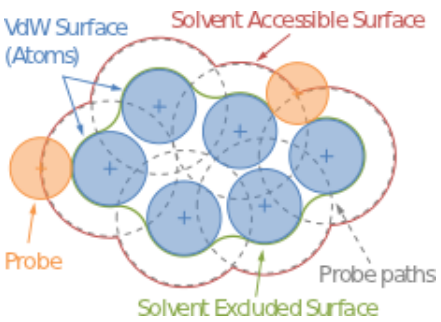


Figure 5.4 – Vue 2D de la génération d'une surface moléculaire. Une sphère (cercle orange) d'un diamètre correspondant au diamètre d'une molécule d'eau passe sur l'ensemble des atomes (cercles bleus) situés en bordure de la molécule. Le point de contact entre la sphère et l'atome définit la surface exclue par le solvant. La position du centre de la sphère au moment du contact entre la sphère et les atomes de la molécule constitue la surface accessible par le solvant. [2]

— Une isosurface est extraite à partir de cette grille en utilisant l'algorithme du *Marching Cube*[57].

La surface générée par cette méthode est moins précise qu'une SAS. Toutefois, le coût en calcul est bien moins élevé. La précision de la surface peut être contrôlée en ajustant la taille de la grille. Cette méthode présente également l'avantage d'être aisément parallélisable à chaque étape de l'algorithme. En effet, les calculs ,pour chaque atome dans la première étape et pour chaque cellule dans les deux étapes suivantes, sont indépendants. Le calcul distribué de cet algorithme est également assez simple à obtenir. Ces propriétés en font un candidat idéal comme traitement in-situ.

Notre implémentation de l'algorithme du Quicksurf est composée de quatre modules parallèles. La Figure 5.5 présente une vue globale de l'application. Comme pour l'application d'écriture, les positions des atomes sont d'abord agrégées par nœud (chaînage, pas de copie). Le premier module parallèle (Morton) calcule un code de Morton pour chaque position d'atome. Un code de Morton correspond à l'index d'une cellule dans une grille. Pour le calculer, le module dispose des dimensions de la grille régulière ainsi que de sa position dans l'espace. Ainsi à chaque atome est associé une cellule. Enfin les atomes sont triés par leurs codes de Morton. Les positions des atomes ainsi que leurs index de Morton sont ensuite envoyés au module suivant sans copie.

Le deuxième module est en charge de construire la structure de grille 3D. Pour chaque cellule de la grille, nous associons deux entiers. Le premier représente l'index du premier atome appartement à cette cellule, le deuxième représente le nombre d'atomes appartenant à cette cellule. Si une cellule ne contient pas d'atomes, le deuxième entier vaut zéro. Comme les dimensions de la grille sont relativement petites (20Mo pour notre benchmark), nous pouvons nous permettre de représen-

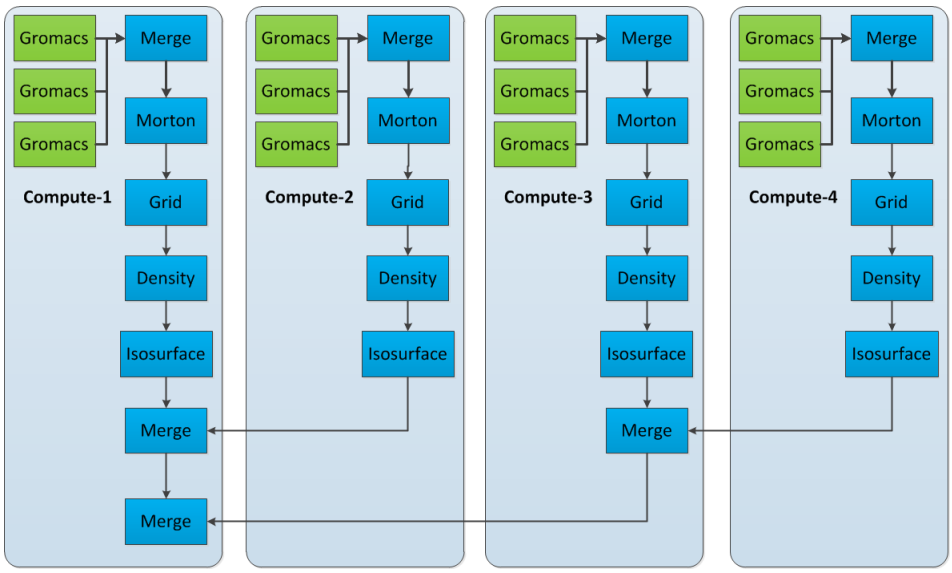


Figure 5.5 – Graphe de l'application du Quicksurf.

ter explicitement des cellules vides. Pour des grilles plus grandes, ces cellules vides doivent être supprimées. L'accès à une cellule particulière se fait alors par recherche binaire. Comme les atomes reçus sont déjà triés par leur code de Morton, la construction de la grille est assez rapide. Les positions des atomes, leurs index de Morton et la structure de grille sont ensuite envoyés au module suivant sans copie.

Le troisième module calcule les valeurs de densité pour chaque cellule. La densité d'une cellule est basée sur le nombre d'atomes, leurs positions et leurs diamètres dans la cellule courante ainsi que de ses 27 voisins. Finalement, la structure de grille ainsi que les densités sont envoyées sans copie vers le dernier module qui va effectuer le *Marching cube*. L'étape finale consiste à rassembler l'ensemble des triangles produits pour obtenir l'isosurface complète.

L'application présentée suppose néanmoins que chaque nœud dispose d'un sous-domaine strict ainsi que d'une zone fantôme aux frontières du domaine (pour le calcul de densité). Dans le cas contraire, les densités calculées sont partielles. Cette hypothèse n'est malheureusement pas valide avec Gromacs. Il serait possible d'extraire les informations nécessaires de la simulation à partir des informations du système de décomposition de domaine ainsi que les listes de voisinage des atomes maintenues par Gromacs. En pratique, les listes de voisinage de Gromacs seraient insuffisantes car elles sont maintenues pour un seuil de distance correspondant aux besoins de la simulation. En revanche, pour le Quicksurf, la taille de la zone fantôme est directement liée à la précision de l'isosurface souhaitée par l'utilisateur et peut donc différer de celle de Gromacs.

Ce problème est très similaire au cas de la recomposition d'image. Dans ce problème, chaque processus possède une image partielle obtenue à partir de données

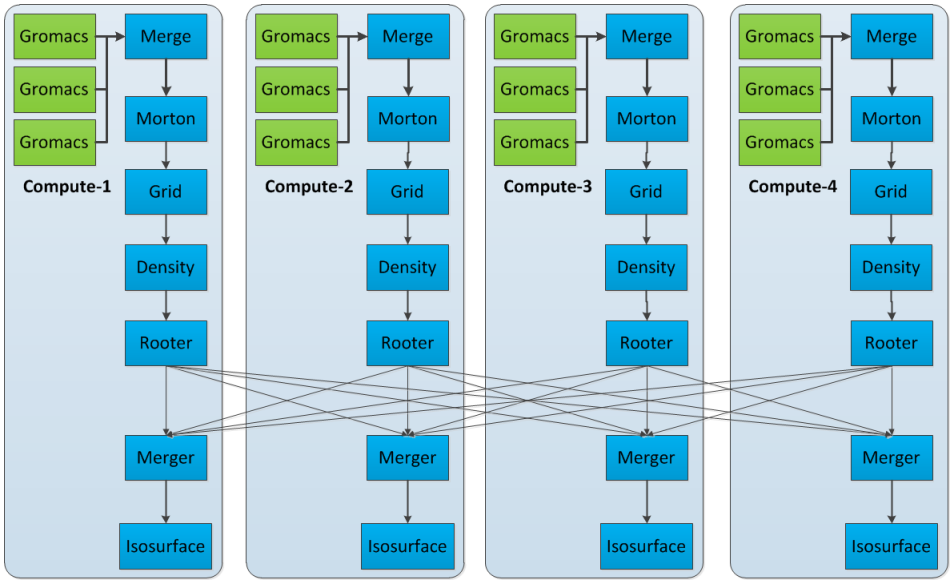


Figure 5.6 – Graphe de l'application du Quicksurf en redistribuant les valeurs de densité.

locales. Les images sont ensuite recomposées pour obtenir l'image finale. L'image finale est répartie entre les différents processus où chaque processus dispose d'une fraction de l'image finale. On peut voir cette opération comme une forme de MPI_REDUCE_SCATTER. Différents algorithmes ont été proposés pour effectuer la recomposition. Les plus utilisés sont le *Direct-Send*[41, 62], le *Binary-Swap*[52], le *2-3 Swap*[91] et le *Radix-k*[64]. Pour plus de détail sur ces algorithmes, nous invitons le lecteur à consulter les articles de référence.

Pour obtenir les valeurs finales de densité, nous avons choisi d'utiliser une approche de type *Direct-Send*. Chaque processus reçoit des autres processus les informations nécessaires à son sous-domaine respectif. Des communications en NxM ont donc lieu (M étant généralement égal à N). Nous pouvons effectuer cette redistribution à deux moments dans notre algorithme.

La première possibilité est après le calcul des densités partielles. Chaque processus devient responsable d'un sous-domaine de la grille globale et va recevoir les contributions des autres processus. Pour obtenir la densité globale d'une cellule, les densités partielles sont simplement sommées. Nous obtenons alors le graphe d'application de la Figure 5.6.

Pour redistribuer les densités, nous utilisons deux composants parallèles *Rooter* et *Merger*. Le composant *Rooter* est chargé de répartir les données, ici les densités, vers les différents hôtes. Le composant *Merger* reçoit l'ensemble des données affectées à son sous-domaine et les accumule, ici en sommant les densités. Afin d'éviter toutes communications inutiles, seules les densités non nulles sont envoyées sur le réseau.

Il est également possible de redistribuer les atomes directement. Ainsi chaque

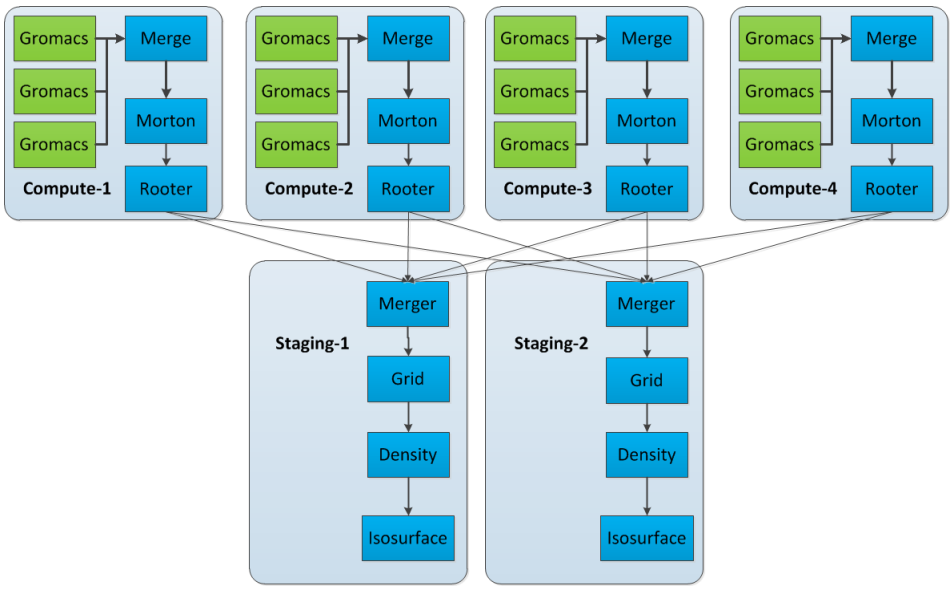


Figure 5.7 – Graphe de l'application du Quicksurf en redistribuant les positions des atomes.

processus dispose de l'ensemble des atomes correspondant à son sous-domaine et peut donc calculer une valeur de densité pour chaque cellule de son domaine. Le graphe de l'application obtenu est présenté Figure 5.7. Nous effectuons ici la redistribution après avoir calculé les codes de Morton pour chaque atome. Ce code de Morton est ensuite utilisé par le composant *Rooter* pour diriger les positions des atomes vers les destination appropriées.

Les composants *Rooter* et *Merger* ont été développés pour supporter en entrée plusieurs types de données, en particulier des positions d'atomes et des densités. À chaque type de donnée est associé un port d'entrée et de sortie. Le comportement de ces modules s'adapte selon le type de données en entrée, c'est à dire selon les ports d'entrée qui sont connectés au moment de l'exécution de l'application.

Avoir subdivisé l'algorithme du Quicksurf en différents modules nous permet d'obtenir une certaine flexibilité dans le placement des traitements. Nous proposons trois configurations différentes de placement du pipeline de traitement in-situ. La première configuration utilise uniquement un cœur dédié sur chaque nœud. L'ensemble des modules (hors simulation) ainsi que l'arbre d'agrégation final sont hébergés sur les cœurs dédiés. La redistribution des données a lieu après le calcul des densités. Cette configuration correspond à la Figure 5.6. La deuxième configuration utilise les cœurs dédiés ainsi que des nœuds dédiés. Pour transférer les données vers les nœuds dédiés, nous utilisons les composants *Rooter* et *Merger*. Le composant parallèle Rooter est réparti sur les nœuds de la simulation tandis que le composant parallèle *Merger* est réparti sur les nœuds dédiés. Le composant *Rooter* adapte la répartition des atomes selon le nombre de modules du composant *Merger*. Comme

précédemment, nous redistribuons les données après le calcul de densité. La troisième configuration utilise également les cœurs dédiés ainsi que des nœuds dédiés. Cette fois nous effectuons la redistribution des données après le calcul des codes de Morton. Les composants situés avant le composant *Rooter* sont exécutés sur les cœurs dédiés tandis que les autres sont exécutés sur les nœuds dédiés. Cette configuration correspond à la Figure 5.7.

La Figure 5.8 présente les résultats de performance obtenus avec ces trois configurations. Les performances de Gromacs sans écriture sont également données pour indiquer les valeurs de performance maximale possibles. Pour les configurations utilisant des nœuds dédiés, nous avons alloué un nœud dédié pour 64 nœuds de calcul. Nous constatons que pour les trois configurations testées, les performances sont assez similaires. La deuxième configuration (colonne C_intransit) est la moins coûteuse avec une baisse de 7% des performances par rapport à Gromacs utilisant 15 cœurs par nœud sur 128 nœuds. La première configuration (colonne C_insitu) arrive deuxième avec un impact de 8% sur les performances de la simulation mais en utilisant 1,5% de nœuds en moins. Cette différence de performance peut s'expliquer par les communications faites entre les nœuds de la simulation pour redistribuer les atomes. Dans le cas précédent, les communications se faisaient en direction des nœuds dédiés provocant ainsi moins de perturbations réseaux. La troisième configuration (colonne A_intransit) cause une baisse de performance de 8,6% de Gromacs malgré l'utilisation de nœuds dédiés. Dans ce scénario, nous redistribuons l'ensemble des atomes et non plus les densités. Les densités représentent environ 700000 flottants à échanger. Les positions des atomes nécessitent trois flottants soit environ 6000000 flottants. Redistribuer les atomes représente donc une charge réseau quasiment dix fois plus importante.

5.3 Discussion

5.3.1 Besoin de flexibilité dans le placement des traitements in-situ

L'exemple de l'algorithme du Quicksurf illustre la nécessité de pouvoir facilement composer avec les différents placements de traitements possibles. Ce besoin est motivé par plusieurs facteurs.

Le premier facteur est le coût en ressources. Quand bien même la différence maximale entre les trois configurations testées est de seulement 1,5%, cela peut tout de même représenter plusieurs milliers d'heures de calcul pour des simulations de production. Mais pour trouver la configuration la moins coûteuse, il est nécessaire de tester facilement et rapidement différentes configurations. La description de l'application par un script Python ainsi que la gestion transparente des communications par le démon nous permettent d'atteindre cet objectif.

Le second facteur vient du besoin d'adaptation aux différents traitements in-situ.

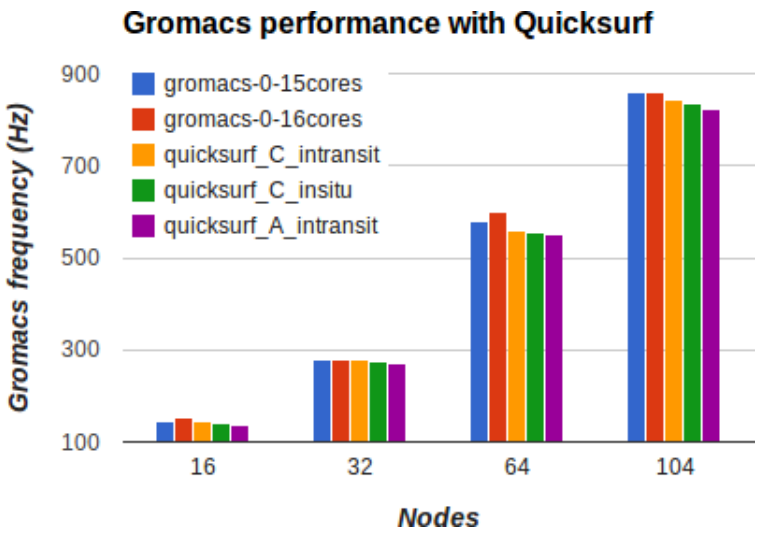


Figure 5.8 – Évolution de la fréquence de Gromacs avec différentes configurations de placement pour le calcul de l'algorithme Quicksurf.

Dans l'exemple du Quicksurf, nous n'avons utilisé qu'un seul traitement in-situ. Le choix de la configuration à adopter dépend donc uniquement du coût de chaque configuration. Si d'autres traitements in-situ doivent être exécutés en parallèle du Quicksurf, la configuration la moins coûteuse ne sera pas forcément la moins souhaitable. En particulier, la troisième configuration redistribuant les positions des atomes sur des nœuds dédiés s'avérera probablement plus rentable. En effet, les positions des atomes sont la base de la plupart des analyses effectuées par les biologistes. Il est donc possible de rentabiliser le transfert des atomes en ajoutant des traitements supplémentaires sur les nœuds dédiés. En comparaison, les densités présentent un intérêt beaucoup plus limité pour d'autres traitements. Notre infrastructure permet de changer très facilement de configuration suivant les différents traitements effectués.

Un troisième facteur peut venir d'un besoin d'adaptation aux architectures des supercalculateurs. De plus en plus de supercalculateurs ont des architectures hétérogènes. La machine *Blue Waters* dispose par exemple d'un sous-ensemble de nœuds équipés de cartes graphiques. Dans d'autres cas, un cluster de visualisation peut être disponible sur le même réseau qu'un cluster de calcul. Dans les années à venir, il est probable que les utilisateurs aient accès à des nœuds d'I/O. Ces différentes ressources offrent des opportunités pour effectuer des traitements in-situ spécifiques comme de la visualisation ou de la réduction de données avant écriture. Il faut toutefois être capable de gérer l'ensemble de ces ressources hétérogènes de façon cohérente au sein d'une même application. En permettant de placer chaque module individuellement de manière explicite, l'utilisateur peut gérer toutes ces différentes ressources et placer ses traitements sur les ressources adéquates. Le *runtime* de FlowVR se chargera ensuite de lancer l'application sur les différentes ressources.

Un quatrième facteur que nous n'avons pas exploré, mais qui va devenir de plus en plus important pour l'exascale, est la consommation énergétique[39, 89]. Pour les machines exascales, les machines parallèles ne pourront pas consommer plus de 20MW pour des raisons budgétaires[82]. Un objectif à atteindre pour une application avec des traitements in-situ peut être de réduire autant que possible sa consommation énergétique. Les accès mémoires et transferts réseaux sont des opérations coûteuses en énergie. Le placement des traitements in-situ peut avoir une importance significative sur la consommation de l'application globale car ils peuvent impliquer des transferts de données importants ainsi qu'une consommation mémoire supplémentaire. Le placement manuel des processus et en particulier la possibilité de gérer à la main les communications de FlowVR permet de tester rapidement différentes configurations et évaluer la moins coûteuse énergiquement (pas nécessairement la plus rapide en temps de calcul).

5.3.2 Difficultés du partage des ressources

Les traitements in-situ rentrent nécessairement en concurrence avec la simulation pour l'accès aux ressources processeurs, mémoires et réseaux. Des stratégies comme le cœur ou le nœud dédié visent à limiter cette concurrence mais un impact négatif sur les performances de la simulation reste inévitable. Les codes de dynamique moléculaire sont des codes extrêmement rapides qui utilisent intensivement le processeur et le réseau mais assez peu la mémoire. Le scénario d'écriture a montré en particulier l'impact que peuvent avoir les communications réseaux faites pour les traitements in-situ. Ce constat se retrouve également pour le cas du Quicksurf lorsque les données sont redistribuées sur les nœuds de la simulation. De manière générale, il semble assez difficile de réaliser des traitements in-situ nécessitant des communications réseaux sans payer un prix significatif sur les performances de la simulation. Il devient alors légitime de se demander s'il s'agit là d'une des limites de l'in-situ. Ces résultats obtenus sont cependant spécifiques au cas de Gromacs. D'autres types d'applications sont sans doute plus sensibles à d'autres types de stress. Les expérimentations faites pour le système Goldrush[96] en stressant différentes parties d'une machine montrent bien ce point. Il est donc nécessaire de pouvoir s'adapter à la fois à la simulation et aux traitements in-situ. Nous avons exposé dans ces exemples différents schémas de placement, bien d'autres sont également possibles. Il est cependant clair qu'il n'existe pas de réponse directe à n'importe quelle application. FlowVR permet à l'utilisateur de placer manuellement ses composants par machine et par cœur. Nous espérons que cette flexibilité permettra de répondre à un maximum de scénarios et d'applications possibles. Cette flexibilité se fait toutefois au détriment du développeur qui doit porter une attention particulière à l'architecture de la machine cible ainsi qu'à son application. Nous pensons cependant qu'il s'agit d'un passage obligatoire pour apporter une réponse globale au problème du placement des traitements in-situ.

5.3.3 Passage de l'application IMD au contexte In-Situ

Le code de Gromacs est strictement identique entre les versions interactives et simulations longues. Le seul ajout qui a été fait est la possibilité de régler la fréquence d'extraction de données au niveau de Gromacs. En réglant cette fréquence à un, nous obtenons le même comportement que pour l'application IMD. De même dans le contexte in-situ, les ports nécessaires pour la réception de forces sont toujours présents bien qu'ils ne soient pas utilisés. Comme les ports d'entrée correspondant aux forces ne sont pas connectés, la fonction Wait() de FlowVR n'attend pas sur ces ports. Nous pouvons donc utiliser la même simulation dans différents contextes d'utilisation. Ceci est notamment un apport du modèle de programmation par composant que propose FlowVR qui permet de réutiliser au maximum un code (c'est-à-dire un composant) dans différentes applications.

Chapitre 6

Usages dans le contexte de la dynamique moléculaire

Dans les chapitres précédents, nous avons proposé une infrastructure capable de connecter Gromacs à des traitements in-situ de façon flexible et avec un coût limité. Nous avons également montré que notre infrastructure s'adapte aussi bien à des contextes interactifs que des simulations longues sur de grandes machines parallèles. Dans ce chapitre, nous mettons l'accent sur l'usage par les biologistes d'une telle plateforme. En collaboration avec des experts du domaine, nous avons construit différents scénarios d'utilisation pouvant aider au travail des biologistes. Nous montrons en particulier que notre infrastructure est capable de répondre à des besoins aussi bien lors de la phase de simulation avec des traitements in-situ mais aussi lors de la phase de post-traitement.

Les travaux présentés dans ce chapitre ont été publiés et présentés dans le journal international *Faraday Discussions*. Nous détaillerons dans ce chapitre uniquement les travaux relatifs aux analyses faites en in-situ et post-traitement pour la dynamique moléculaire. D'autres volets comme la visualisation dans un CAVE sont présentés dans cet article mais ne seront pas détaillés dans ce chapitre.

6.1 Applications biologiques : le canal FepA et le canal du GLIC

Dans ce chapitre, nous proposons différents scénarios d'application centrés sur deux problèmes biologiques. Le premier cas est le canal du FepA que nous avons déjà présenté dans la section 4.2. Nous verrons en particulier comment résoudre le problème d'échelle de temps rencontré lors de la manipulation interactive de ce système.

Notre deuxième cas d'utilisation s'articule autour du canal GLIC (Figure 6.1). La transmission de signaux entre les neurones est un processus biologique essentiel qui n'est pas encore tout à fait compris. La famille des récepteurs pentamériques joue un rôle central. GLIC, de la bactérie *Gloeobacter violaceus*, est le système le mieux caractérisé à ce jour. Il s'agit d'un canal ionique avec un pore central formé par cinq sous-unités identiques. Lorsque le canal est ouvert, des ions peuvent s'écouler à l'intérieur. Au cours du temps, le canal peut se fermer et empêcher le passage des ions. Ce mécanisme de fermeture reste pour le moment assez incompris. Récemment,

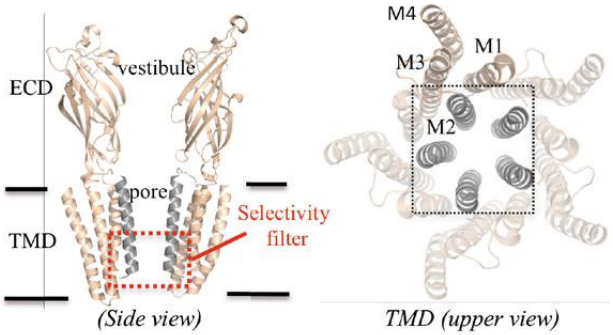


Figure 6.1 – Représentation du canal GLIC. Celui ci est composé en particulier de plusieurs hélices. L'orientation des hélices M2 semble être liée à l'état ouvert ou fermé du canal. [2]

un lien a été observé entre le passage de l'eau dans le canal et l'orientation des hélices M2 (les cinq sous structures). Nous souhaitons utiliser notre infrastructure pour étudier le passage de l'eau dans ce canal en corrélation avec l'orientation des hélices. Le système moléculaire complet est composé de 143000 atomes.

6.2 Contribution

6.2.1 Intégration de modules d'analyses

De nombreux outils et bibliothèques existent pour traiter et analyser des trajectoires issues de simulations de dynamique moléculaire. Parmi ceux-ci, nous pouvons citer VMD[42], MDAnalysis[59], AmberTools[23] ou encore les outils intégrés de Gromacs[69]. Au cours de nos travaux, nous nous sommes intéressés plus particulièrement à MDAnalysis ainsi qu'aux outils de Gromacs. MDAnalysis est une bibliothèque écrite en Python disposant de nombreux algorithmes classiques d'analyse ainsi qu'un langage de sélection puissant inspiré de celui de CHARMM[20]. N'étant pas rattaché à un paquet de simulation de dynamique moléculaire spécifique, il supporte également un grand nombre de formats de trajectoire couramment utilisés. Les outils de Gromacs sont de petits exécutables destinés à effectuer des analyses plus spécifiques. Ils sont généralement construits pour appliquer des post-traitements sur l'ensemble d'une trajectoire. Il est par exemple possible de filtrer une trajectoire, la recentrer sur un groupe d'atomes ou encore supprimer des sauts de positions d'atomes dus aux conditions périodiques de la simulation.

Le passage à un mode de traitement in-situ pour effectuer des traitements habituellement faits en post-traitement peut représenter un grand changement pour les

2. Source Figure 6.1 : http ://www.synchrotron-soleil.fr/Soleil/ToutesActualites/2013/PX1-EMBO

utilisateurs, notamment en terme d'environnement. Aussi il peut être très intéressant d'intégrer des outils déjà bien établis dans le domaine métier. D'une part, cela peut permettre aux utilisateurs d'effectuer une transition moins brutale en proposant des outils familiers. D'autre part, cela permet aux utilisateur de conserver les scripts et travaux qu'ils ont pu construire au fil des années.

Nous avons intégré MDAnalysis ainsi que deux des outils de Gromacs dans notre infrastructure, chacun répondant à des besoins précis. La méthode de couplage que nous proposons est générique et devrait pouvoir s'appliquer à de nombreux outils.

Pour traiter des trajectoires, la plupart des outils s'appuient sur une abstraction de fichier pour lire et écrire des données. Cette abstraction permet entre autre de supporter différents formats de fichier avec une seule et même interface de programmation. Nous avons étendu ces outils pour lire les données en provenance d'une application FlowVR en s'appuyant sur leurs interfaces d'entrée/sortie déjà en place. Aux détails d'implémentation près, chacun de ces outils propose une interface composée de quatre fonctions :

— Open : Ouvre le fichier et lit éventuellement les métadonnées de la trajectoire (nombre de pas de temps, nombre d'atomes, dimensions de la boîte englobante, etc)
— Read : Lis l'ensemble des positions des atomes correspondant à un pas de temps
— Write : Écris l'ensemble des positions des atomes correspondant à un pas de temps généralement après traitement
— Close : Ferme le fichier en lecture ou écriture.

Une fois cette interface adaptée pour notre infrastructure, les outils peuvent s'exécuter comme si les données venaient d'une trajectoire classique à condition toutefois de respecter le formatage des données des outils. Pour intégrer ces outils dans un contexte FlowVR, il faut déclarer un module, un port et exécuter la boucle itérative. Ces opérations peuvent s'intégrer de manière assez naturelle à l'interface de fichier :

— Open : Déclaration du module, création d'un port d'entrée (resp. de sortie) si le fichier doit être ouvert en lecture (resp. écriture) et enfin initialisation du module.
— Read : Wait() et Get() pour recevoir les nouvelles positions des atomes
— Write : Wait() et Put() pour envoyer le résultat du traitement vers le reste de l'application FlowVR
— Close : Fermeture du module.

Nous avons appliqué cette méthodologie avec succès à la fois dans MDAnalysis et pour les outils *trj_conv* et *trj_order* de Gromacs. Compte tenu de la généricité de cette interface, nous espérons que cette méthode puisse s'appliquer à de nombreux autres outils. Nous présentons des exemples d'application de ces outils dans les sections suivantes.

Des restrictions existent tout de même pour ces outils. La plus importante est

qu'il n'est pas possible de "rembobiner" une trajectoire dans un contexte in-situ de manière simple. En effet, bien souvent seule l'itération courante est disponible en mémoire. Certaines analyses peuvent avoir besoin de plusieurs pas de temps pour s'exécuter et ne peuvent donc pas être supportées sans modifications supplémentaires. Il est alors à la charge de l'outil d'analyse de conserver en mémoire éventuellement plus d'itérations. Une autre limitation vient des analyses nécessitant une trajectoire complète pour produire un résultat. Des résultats partiels sont généralement accumulés au cours de la lecture des pas de temps jusqu'à produire le résultat final. Dans un contexte in-situ, il est souhaitable de consulter ces résultats partiels au fur et à mesure qu'ils s'accumulent. Ce besoin implique toutefois des modifications de codes bien plus importantes dans les outils usuels, ceux-ci étant construits en supposant une trajectoire complète.

6.2.2 Application à l'écriture de trajectoires multiples

Usage des trajectoires par les biologistes

Nous avons démontré que notre infrastructure est capable d'écrire une trajectoire à haute fréquence avec un impact bien moins important sur les performances de la simulation que la méthode native de Gromacs. Il n'est toutefois pas raisonnable pour les utilisateurs de sauvegarder une trajectoire complète à de si hautes fréquences (1/100). En effet, la taille de la trajectoire produite serait bien trop lourde à manipuler et traiter par des outils usuels.

Dans beaucoup de cas, les biologistes ne travaillent pas directement sur une trajectoire complète. Une première passe de post-traitement est appliquée sur la trajectoire afin de la réduire et faciliter les futures analyses. Les outils de Gromacs *trj_conv* et *trj_order* sont utilisés dans ce but. L'outil *trj_conv* est utilisé pour recentrer une trajectoire et traiter de manière générale les artefacts provoqués par les conditions périodiques de la simulation. L'outil *trj_order* permet de filtrer une trajectoire selon des critères de distance par rapport à un ensemble d'atomes.

Partant de ce constat, il peut être intéressant de générer au moment de la simulation des trajectoires déjà prêtes pour de futures analyses. Cette seconde trajectoire est généralement de taille réduite par rapport à une trajectoire classique. Il devient alors plus raisonnable d'augmenter la fréquence d'écriture pour ces trajectoires en tirant bénéfice des performances d'écriture permises par notre infrastructure.

Bien que très utile, une trajectoire filtrée seule n'est par suffisante. En effet, le filtrage supprime des informations inutiles pour certaines analyses mais probablement utiles pour d'autres. La trajectoire complète reste donc nécessaire.

Description de l'application

Nous proposons un exemple d'application écrivant deux trajectoires distinctes. Notre simulation cible est le canal GLIC présenté dans la section 6.1. Ce système

moléculaire comprend deux points d'intérêt majeur pour notre étude : les hélices M2 ainsi que les molécules d'eau aux abords du pore et à l'intérieur du canal. La première trajectoire correspond à une trajectoire complète écrite une fois toutes les 1000 itérations. La deuxième trajectoire correspond à une trajectoire filtrée ne contenant que les hélices ainsi que les 1000 molécules d'eau les plus proches du centre du canal. La trajectoire est également recentrée sur un des résidus présents à l'intérieur du canal pour supprimer tout décalage provoqué par les conditions périodiques de la simulation. La trajectoire filtrée ne représente plus que 3000 atomes environ sur les 143000 initiaux.

Les outils de Gromacs ne supportent pas MPI et seuls quelques algorithmes sont accélérés pour le multi-cœur. Il est donc nécessaire de rassembler l'ensemble des données en un seul message pour traiter et filtrer les positions des atomes avant de les écrire. D'autres part, la sélection des 1000 molécules d'eau les plus proches du canal est une opération coûteuse qui nécessite un certain temps de calcul. En pratique, cette opération est trop lente pour absorber l'ensemble des données produites par la simulation. Il serait possible de bloquer la simulation lorsque trop de messages sont en file d'attente comme le fait FlexIO par exemple. Bloquer la simulation impacterait néanmoins directement ses performances. Une solution préférable serait de paralléliser les outils dont nous avons besoin à savoir *trj_order* mais le coût en développement serait sans doute important.

L'indépendance de chaque pas de temps nous permet de créer plus facilement un parallélisme horizontal. Plutôt que de paralléliser directement l'outil, nous créons plusieurs instances du même outil. Un composant spécial appelé *Rotate* placé en amont distribue les itérations de manière cyclique sur de multiples instances des composants *trj_order*. De même, un composant spécial appelé *Unrotate* réordonne les itérations en aval.

La Figure 6.2 présente le graphe complet de l'application. Les données sont extraites une fois toutes les dix itérations de Gromacs puis rassemblées une première fois par nœud. Un arbre de rassemblement vient agréger les positions des atomes. Une fois les données rassemblées, elle sont envoyées vers deux *pipelines* de traitement (sans copie). Le premier *pipeline* est composé d'un module *IterationFilter* (Section 4.6) réglé à 100 (soit une itération toutes 1000 itérations sachant que la simulation n'émet des données que toutes les 10 itérations) puis d'un module d'écriture. Le deuxième *pipeline* est composé des multiples instances du composant *trj_order*, du composant *trj_conv* pour recentrer la trajectoire ainsi que d'un module d'écriture.

Evalutation des performances de l'approche

Nous avons testé notre application sur la machine Froggy (voir Section 5.2.2). Le modèle du GLIC composé de 143000 atomes est utilisé. Nous avons alloué 16 nœuds à la simulation ainsi qu'un nœud dédié. La simulation utilise 15 cœurs par nœud (stratégie du cœur dédié). Un filtre *Merge* ainsi que le démon FlowVR sont hébergés sur le cœur dédié de chaque nœud. Enfin, les modules de filtrage et d'écriture sont

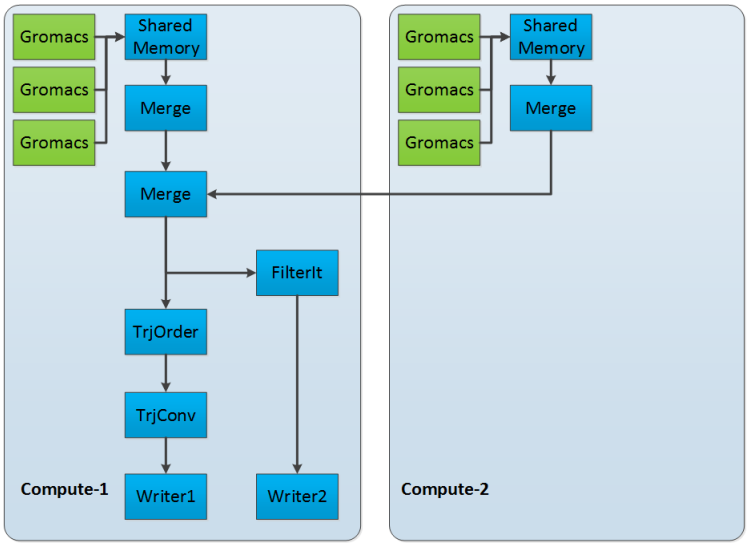


Figure 6.2 – Graphe de l'application générant deux trajectoires. La première trajectoire est sauvegardée à haute fréquence (1/10). Les données sont filtrées (*trj_order*) puis recentrées (*trj_conv*) avant d'être écrites. La seconde trajectoire sauvegarde l'ensemble des positions des atomes à faible fréquence(1/1000).

Setup	Performance (Hz)	Relative Cost
gromacs-0-16cores	272	-
gromacs-0-15cores	280	-
gromacs-1000-16cores	271	0.36%
gromacs-100-16cores	261	4.04%
gromacs-10-16cores	187	31.35%
exaviz-full-10-15cores	273	2.5%
exaviz-full-1000-filtre-10-15cores	273	2.5%

Table 6.1 – Comparaison des performances de Gromacs avec sa méthode d'écriture native par rapport à notre infrastructure. Le modèle GLIC (143000 atomes) est utilisé avec une configuration fixe de 16 nœuds soit 256 cœurs au maximum. *gromacs-X-Y cores* écrit toutes les X itérations avec la méthode native de Gromacs en utilisant Y cœurs par nœud. *exaviz-full-X-filtre-Y-Z cores* écrit une trajectoire complète toutes les X itérations et une trajectoire filtrée toutes les Y itérations avec notre infrastructure en utilisant Z cœurs par nœud. Les coût relatifs sont donnés par rapport aux configurations avec le même nombre de cœurs.

hébergés sur le nœud dédié. Les résultats des expérimentations sont présentés dans la Table 6.1.

Globalement, Gromacs ne passe pas bien à l'échelle lorsque l'on augmente la fréquence d'écriture (31% de baisse de performance en écrivant toutes les 10 itérations par rapport à Gromacs sans écriture). *exaviz-full-10-15cores* écrit une trajectoire

complète toutes les 10 itérations. Notre impact n'est que de 2,5% par rapport à Gromacs sans écriture utilisant 15 cœurs. *exaviz-full-1000-filtre-10-15cores* correspond à l'application décrite dans la Figure 6.2. Nous retrouvons ici les même performances que *exaviz-full-10-15cores*. Les deux configurations génèrent le même trafic réseau sur les nœuds de la simulation, seule la charge CPU sur le nœud dédié est modifiée. Comme l'on pouvait l'espérer, les traitements effectués sur des nœuds dédiés qui n'utilisent pas le réseau n'ont pas d'impact sur les performances de la simulation.

Vers une écriture des trajectoires adaptatives

Filtrer les trajectoires avant de les écrire est un bon moyen de réduire la taille de ces dernières. Une autre solution peut consister à adapter la fréquence d'écriture de la trajectoire au déroulement de la simulation. Les simulations de dynamique moléculaires sont souvent constituées de phases de repos et de phases d'activité. Lors des phases de repos, la simulation évolue assez peu. Il est alors censé de réduire la fréquence d'écriture car il n'y a pas de risque de manquer des événements biologiques à cause de l'échantillonnage. En revanche, lors des phases actives, il peut devenir plus intéressant d'augmenter la fréquence d'écriture.

Des analyses plus spécifiques peuvent également motiver la modulation de la fréquence d'écriture. Dans le cas du canal GLIC par exemple, les phases d'intérêt sont l'ouverture et la fermeture du pore. Un scénario possible consiste à moduler la fréquence d'écriture en fonction de la variation du nombre de molécule d'eau dans le pore.

Notre infrastructure nous permet très facilement d'obtenir ce type d'application. Il est toutefois nécessaire de définir des métriques pour calculer la fréquence d'écriture appropriée en fonction de l'état de la simulation. Au moment de l'écriture de ce mémoire, nous n'avons pas encore pu mettre en œuvre un tel scénario faute d'avoir trouvé une métrique appropriée.

6.2.3 Des simulations interactives vers les simulations de production

Dans le Chapitre 4, nous avons étudié l'intérêt des simulations interactives pour guider de manière intuitive un système moléculaire vers une configuration désirée en appliquant des forces extérieures sur un sous-ensemble d'atomes. En revanche, les échelles de temps en jeu sont incompatibles avec des manipulations interactives réalistes du point de vue biologique. Pour effectuer une expérimentation de guidage plus proche de la réalité, c'est à dire avec un niveau de force appliqué bien plus faible, des simulations longues sur de grandes machines parallèles sont nécessaires.

Comme indiqué dans la section 5.3.3, nous partageons la même base de code entre notre application interactive de guidage et notre application in-situ. Ceci nous permet d'utiliser la même application aussi bien sur un petit cluster de calcul

comme nous le faisons pour nos expérimentations interactives, mais aussi aussi sur de grandes machines parallèles. Certains modules doivent toutefois être adaptés ou remplacés pour tenir compte des contraintes liées au contexte d'exécution. Dans le cas de notre application in-situ, nous devons retirer le module de visualisation et de gestion des périphériques. La problématique est alors de savoir comment donner à l'utilisateur un moyen simple de guider le système moléculaire en s'appuyant sur l'infrastructure préexistante.

Système de guidage automatique

Nous proposons de nous servir des expériences effectuées en mode interactif par l'utilisateur. Au cours de la manipulation par un utilisateur, le sous-ensemble d'atomes guidé suit un chemin à travers le système moléculaire. Ce chemin peut être représenté par un ensemble de points clés que nous appelons cibles et qui forment une trajectoire. Nous remplaçons le gestion des périphériques par un module gérant un automate. Le rôle de cet automate est d'émettre les directions de forces adéquates pour imposer au sous-système d'atomes de suivre le chemin défini lors d'une session interactive.

À l'initialisation, l'automate lit une requête de sélection d'atomes dans la syntaxe MDAnalysis correspondant au sous-ensemble d'atomes à guider. Il lit également une liste de cibles à atteindre dans un ordre précis définissant le chemin à suivre. Les cibles peuvent être décrites soit de manière absolue dans l'espace, soit de manière relative avec une position initiale. L'utilisateur peut également spécifier des points de référence (sous-ensemble d'atomes, généralement un résidu) dans le système moléculaire complet. Ces références, supposées fixes dans le système moléculaire, sont utilisées pour calculer le déplacement global du système moléculaire. En effet, au cours du temps, un complexe moléculaire (comme le canal FepA par exemple) effectue des translations et rotations en plus de subir des changements de conformation. Pour des simulations courtes, des cibles fixes dans l'espace sont suffisantes car le complexe moléculaire n'a pas le temps de se déplacer significativement. En revanche, pour des simulations plus longues, il est nécessaire que les cibles suivent le déplacement du système moléculaire. Les points de référence sont utilisés pour calculer une transformation moyenne du système moléculaire (translation et rotation). Cette transformation est ensuite appliquée aux cibles. Il est absolument nécessaire que les points de référence représentent des points fixes dans le système moléculaire. Dans le cas contraire, le calcul de la transformation moyenne va être biaisé par des transformations plus locales que peut subir le système moléculaire.

Au cours de la simulation, l'automate suit la position à la fois des atomes de référence ainsi que du groupe d'atomes guidés. Si ces groupes sont composés de plus d'un atome, nous assimilons la position du groupe d'atomes à son centre de masse. La sélection d'atomes pour chaque groupe d'atomes ainsi que le calcul du centre de masse de chaque groupe est à la charge de modules séparés utilisant MDAnalysis.

L'Algorithme 3 présente la boucle principale de l'automate. Une fois toutes les

```
currentTarget ← 0;
while wait() do
    ReferencesPositions ← get();
    SelectionPosition ← get();
    updateRelativeTargets();
    if distance(SelectionPosition, targetsPositions[currentTarget]) < threshold
    then
        currentTarget++;
        if currentTarget == nbTargets then
            break;
        end
    end
    direction ← SelectionPosition − targetsPositions[currentTarget];
    normalize(direction);
    put(direction);
end
```

Algorithm 3: Boucle principale de l'automate de guidage.

informations à jour, l'automate vérifie si le centre de masse du groupe d'atomes guidés a atteint la cible courante. Si oui, il passe à la suivante ou s'arrête s'il s'agissait de la dernière cible à atteindre. Enfin, l'automate émet la direction de la force nécessaire pour atteindre la cible courante.

Application au cas du canal FepA

Nous avons appliqué notre système sur le canal FepA à partir des expérimentations interactives décrites dans le Chapitre 4. Le chemin à suivre est composé de huit cibles réparties à travers le canal. Huit résidus sont également utilisés comme points de référence. Ceux-ci sont répartis sur la structure principale du canal que l'on suppose suffisamment rigide.

Les premières expériences menées visent à reproduire les manipulations effectuées par l'utilisateur en interactif avec le même niveau de force (2000pN). L'automate est parvenu à chaque fois à guider convenablement le complexe de fer à travers le canal. De plus, le temps d'exécution nécessaire a également été réduit. Lorsque l'utilisateur effectue la manipulation, la trajectoire est souvent hésitante et ne va pas nécessairement au plus court. L'automate lui va toujours émettre des forces dans la direction précise du chemin permettant ainsi de gagner du temps de manipulation. Ces expérimentations ont été effectuées avec des cibles fixes et relatives. Le temps de l'expérience est d'en moyenne trois minutes pour l'utilisateur et deux minutes pour l'automate.

Nous avons effectué une deuxième série d'expériences en appliquant cette fois un

niveau de force de 1500pN. Interactivement, notre utilisateur n'a pas pu introduire le complexe de fer dans le canal. Notre automate, quant à lui, a pu compléter la manipulation en dix minutes environ. Le complexe de fer a semblé bloquer plusieurs fois au cours de l'expérience. Au bout d'un moment, des chaînes secondaires se sont réorientées permettant ainsi au complexe de fer de poursuivre son chemin.

Problématique des simulations de long terme

Nous avons tenté de diminuer encore le niveau de force appliqué. Malheureusement nous n'avons pas été en mesure de compléter ces expérimentations. Après un certains temps (30-40min), une partie du canal de FepA arrive à la frontière du domaine de décomposition. Deux cas de figure peuvent alors se produire. Le premier est que le canal soit subitement "coupé en deux" où une partie du canal est déplacée à l'opposé de la boîte englobante de la simulation. Dans le deuxième cas, tout le complexe moléculaire est subitement translaté en dehors de la boîte de la simulation (passage en espace image). Dans les deux cas, les cibles définies par l'utilisateur deviennent erronées. Dans le premier cas, calculer la transformation moyenne des points de référence n'a plus de sens. En effet, une partie du canal subit une transformation habituelle de faible amplitude tandis que l'autre partie est translatée à l'autre bout du domaine de décomposition. Dans le deuxième cas, l'automate pourrait poursuivre son fonctionnement normal si le guidage était synchrone par rapport à la simulation. Pour des raisons de performance, le guidage est fait de manière asynchrone. En pratique, lorsque le système de guidage reçoit de nouvelles positions, la simulation a déjà effectué une ou plusieurs itérations et a donc potentiellement translaté de manière très importante. La direction des forces qui en résulte devient donc incohérente par rapport à l'état courant de la simulation. En général, cet asynchronisme n'est pas un problème car nous supposons que le système moléculaire bouge très peu d'une itération à l'autre. Dans le cas du passage en espace image, cette hypothèse n'est plus valide.

Notre approche actuelle pour résoudre ce problème est d'utiliser l'outil *trj_conv* de Gromacs. Il s'agit d'un outil typiquement utilisé pour supprimer les artefacts liés aux conditions périodiques d'une simulation dans une trajectoire. Nous plaçons ce module entre la simulation et l'automate. De cette façon, nous espérons que l'automate n'ait pas à gérer les problèmes de périodicité. Au moment de la rédaction de ce mémoire, nous avons pu résoudre le premier cas possible. Le cas du passage en espace image reste encore à résoudre.

6.2.4 Des simulations longues aux analyses post-mortem et inversement

Grâce à l'intégration d'outils usuels, les biologistes ont la possibilité d'utiliser leurs scripts dans un contexte in-situ moyennant quelques modifications mineures.

Nous espérons ainsi faciliter l'adoption de notre infrastructure par la communauté. Il n'est toutefois pas possible d'effectuer toutes les analyses en in-situ. La raison principale est qu'il n'est pas possible de prévoir toutes les analyses qui seront utilisées lors de l'étude d'une simulation. En effet, une fois les premières analyses effectuées, d'autres analyses sont bien souvent nécessaires suite à la découverte de certains résultats et ainsi de suite. La phase de post-traitement sera donc toujours nécessaire.

Intégration des trajectoires dans notre infrastructure

Pour répondre à cette problématique, nous avons intégré dans notre infrastructure un lecteur de trajectoire développé par le LIFO [3]. Ce module lit une trajectoire produite par Gromacs et génère un flux de données contenant les positions des atomes ainsi que leurs index. Le format de données produit est exactement le même que celui produit par notre module de Gromacs à ceci près que les données ne sont pas distribuées. En conservant le même formatage de données, nous permettons aux biologistes d'utiliser les même modules d'analyse que ceux développés pour un contexte in-situ. Inversement, des modules développés pour fonctionner avec une trajectoire peuvent être utilisés dans un contexte in-situ.

Génération de graphes en ligne

Nous avons introduit un module permettant d'afficher des graphiques en ligne alors que les données sont en cours d'analyse. Les données sont envoyées au fur et à mesure au module qui va les intégrer au graphique désiré. Ce module est configurable via un fichier xml qui décrit les flux de données en entrée (nom, type), les fenêtres graphiques à afficher et enfin l'association entre un flux de données et une ou plusieurs fenêtres. En particulier, l'utilisateur a la possibilité de visualiser plusieurs flux de données dans une même fenêtre afin de les comparer.

Ce module est utilisé à la fois dans un contexte in-situ pour afficher le niveau d'énergie et la température de la simulation par exemple, mais également dans un contexte post-mortem. Nous l'avons par exemple utilisé pour l'étude d'une trajectoire produite par une simulation utilisant le modèle du GLIC (Figure 6.3). Deux modules d'analyse sont couplés au lecteur de trajectoire. Le premier module calcule les angles radiaux et latéraux des cinq hélices M2 du canal GLIC. Le deuxième module compte le nombre de molécules d'eau présentes à l'intérieur du canal. Ces deux modules d'analyse utilisent la bibliothèque MDAnalysis.

Extension au cas des trajectoires et simulations multiples

Une simulation est bien souvent exécutée plusieurs fois afin de pouvoir soit vérifier sa reproductibilité, soit de tester d'autres paramètres. Pour ces différentes simula-

3. Laboratoire d'Informatique Fondamentale d'Orléans

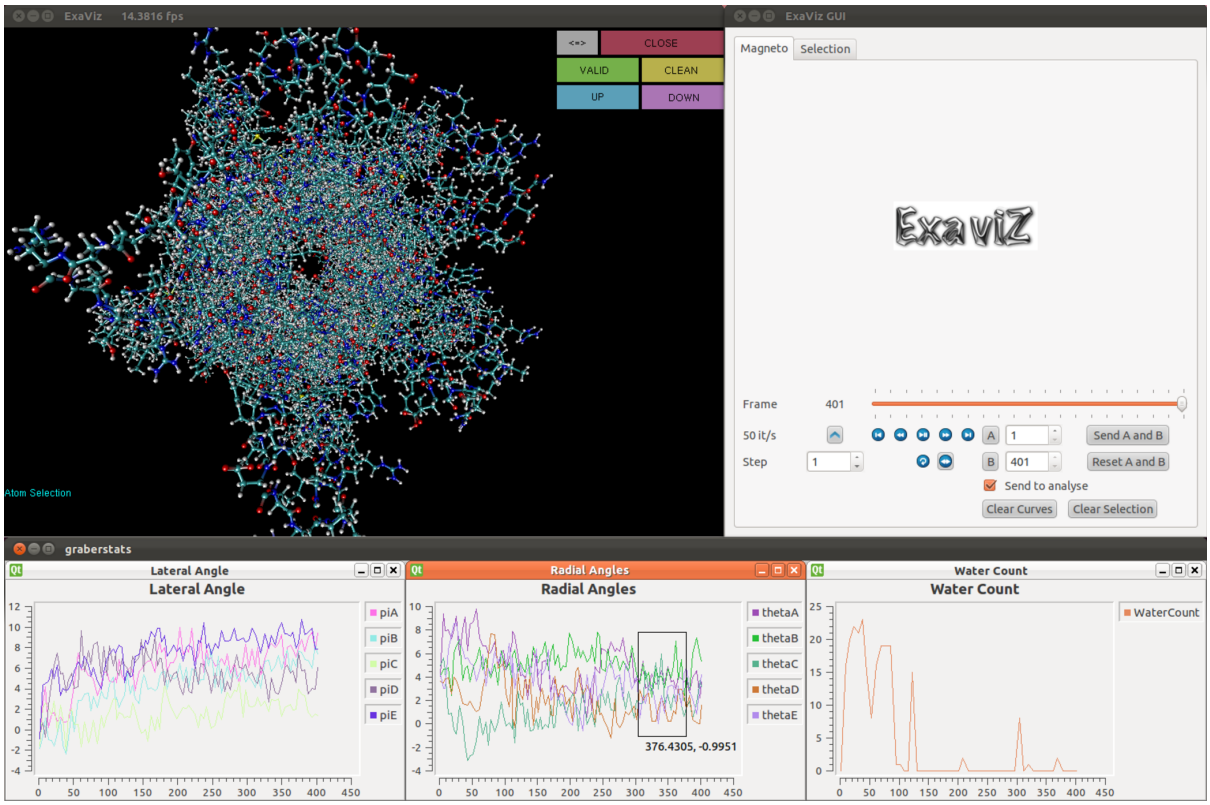


Figure 6.3 – Analyse d'une trajectoire (401 pas de temps) issue d'une simulation utilisant le modèle GLIC. Nous étudions les angles radiaux et latéraux des cinq hélices M2 du canal ainsi que le nombre de molécules d'eau présentes dans le canal.

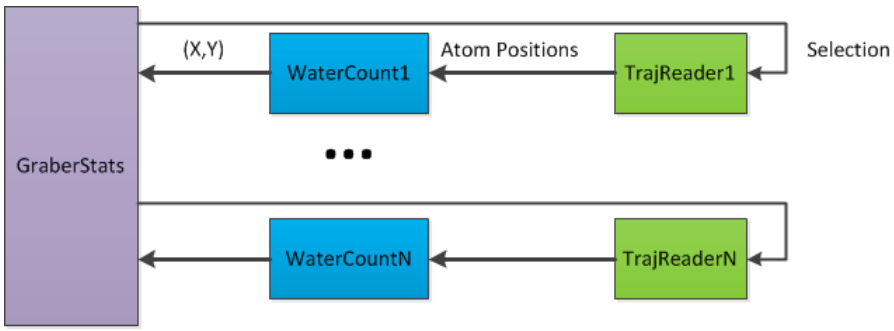


Figure 6.4 – Graphe d'application schématique pour l'analyse de simulations ou trajectoires multiples.

tions, et par extension les différentes trajectoires produites, les biologistes effectuent les mêmes séries d'analyses.

Grâce à notre modèle de programmation par composant, nous pouvons facilement dupliquer un pipeline d'analyse pour l'appliquer à différentes sources (simulation ou lecteur de trajectoire). La Figure 6.4 présente une vue schématique d'un graphe d'application possible. À chaque source de données (simulation ou trajectoire), un ou des modules d'analyse sont instanciés. Les résultats produits sont envoyés vers le module d'affichage des graphiques (GraberStats).

Nous avons testé cette approche en utilisant plusieurs trajectoires issues de simulations du canal GLIC. Au moment de la rédaction de ce mémoire, jusqu'à cinq trajectoires différentes ont pu être analysées simultanément (nombre maximal de trajectoires à notre disposition).

Nous n'avons pas encore pu tester cette approche sur de multiples simulations faute d'avoir un scénario d'utilisation réaliste et utile pour le moment.

6.3 Discussion

6.3.1 Extension vers des pipelines d'analyse plus complexes

Jusqu'à présent, nous avons toujours utilisé des analyses s'appliquant sur l'ensemble d'une trajectoire ou d'une simulation. Le lien entre le module d'analyse et la source de données est alors direct. Cette hypothèse est bien souvent suffisante pour un contexte in-situ et permet déjà de construire des scénarios variés et réalistes. Dans le cas des analyses post-mortem en revanche, les méthodes de travail sont plus complexes.

Les analyses post-mortem forment un processus itératif. Une première série d'analyse est d'abord effectuée sur une partie ou l'ensemble d'une trajectoire. À partir des résultats produits, d'autres analyses peuvent être relancées sur l'ensemble ou une partie de la trajectoire. Cette boucle peut se répéter ainsi pendant plu-

sieurs itérations en générant à chaque fois différentes ramifications dans le processus d'étude.

Le cas où l'ensemble de la trajectoire doit être analysé une seconde fois peut être simplement géré en construisant une seconde application (c'est-à-dire un nouveau graphe d'application). Le second cas est plus compliqué car les deux séries d'analyse ne sont plus indépendantes. Des liens doivent être construits pour savoir quels pas de temps doivent être analysés.

Dans notre approche actuelle, nous construisons une seule application pour gérer toutes les itérations d'analyse. Cela suppose évidemment que le biologiste connaisse les analyses à effectuer à chaque itération. Pour gérer les différentes itérations d'analyse, nous supposons d'abord que l'ensemble des analyses sont connectées à la même source de données (lecteur de trajectoire). Nous nous appuyons ensuite sur deux types de filtrage pour gérer les différentes itérations d'analyse.

Le premier niveau de filtrage est au niveau de la lecture des pas de temps. Les modules d'analyse peuvent envoyer au lecteur de trajectoire une commande pour pour n'envoyer que certains pas de temps de la trajectoire. Pour des raisons de performance, ce filtrage est effectué directement dans le lecteur de trajectoire pour éviter de lire complètement les pas de temps non nécessaires. Ce filtrage pourrait toutefois être fait en dehors du lecteur de trajectoire avec l'aide d'un module similaire au *FilterIteration* utilisé dans notre application in-situ.

Le deuxième niveau de filtrage se fait au niveau du routage des données. Nous plaçons un module intermédiaire entre le lecteur de trajectoire et les modules d'analyse. Son rôle est de ne laisser passer les données que pour certains modules à un moment donné. En pratique, lors de la première lecture de la trajectoire, les données ne sont envoyées qu'aux modules d'analyse de la première itération de traitement, puis aux modules d'analyse de la deuxième itération de traitement lors de la deuxième lecture de trajectoire et ainsi de suite. Pour le moment, il incombe à l'utilisateur de choisir manuellement à quels modules d'analyse vont être envoyées les données à chaque passe de lecture.

6.3.2 Besoin de stockage des données pour les traitements in-situ

Les méthodes d'analyse complexes décrites précédemment sont couramment utilisées par les chercheurs. Il est donc important de pouvoir les supporter. Nous avons vu que, dans un contexte post-mortem, nous pouvons nous rapprocher de ce modèle. Dans une application in-situ en revanche, la problématique est tout autre. En effet, les applications in-situ n'ont accès qu'à l'itération courante de la simulation. Cette contrainte forte s'oppose directement au processus itératif d'analyse que l'on peut souhaiter retrouver également dans un contexte in-situ.

De surcroît, certaines analyses peuvent avoir besoin de plusieurs pas de temps pour produire un résultat. Ces modules peuvent bien évidemment stocker les données

nécessaires au fur et à mesure qu'elle arrivent. Dans le cas où plusieurs modules auraient besoin de ces données, l'empreinte mémoire d'une telle stratégie peut très vite devenir trop importante.

Pour supporter ce type d'analyse, il devient nécessaire de proposer un service d'accès aux données stockées accessible par l'ensemble des modules. Des questions techniques sont alors posées : comment stocker les données (mémoire contre disque dur), à quelle fréquence et comment y accéder. Un premier élément de solution peut être de donner aux modules accès aux données sauvegardées sur disque dur. Cependant, l'accès à ces données peut s'avérer compliqué techniquement alors que les fichiers de trajectoire sont encore en cours d'écriture. Dans certains cas, le format de fichier adopté par une simulation n'est simplement pas prévu pour des accès concurrents. C'est par exemple le cas pour Gromacs qui ne produit qu'un seul fichier pour l'ensemble des données. Une autre solution peut être le modèle proposé par DataSpaces[30]. Dataspaces construit un espace de mémoire partagé dans un contexte distribué sur lequel des clients (ici des analyses) peuvent effectuer des requêtes. Les données sont réorganisées et indexées afin de résoudre plus rapidement les requêtes émises par les clients. Toutefois, à notre connaissance, DataSpaces ne propose pas de méthodes de stockage des données sur disque mais uniquement en mémoire ce qui est insuffisant pour notre cas. Une autre piste intéressante peut être trouvée du côté des bases de données distribuées comme Redis ou Memcache. Celles-ci peuvent servir de plateforme de stockage des données tout en fournissant des moyens d'accès rapides aux données grâce à leur indexation.

Cette problématique est au final complexe et constitue sans doute des pistes de recherche à venir pour résoudre ce problème dans son ensemble.

6.3.3 Avantages et limites d'un environnement unifié

Au cours de cette thèse, nous avons développé une infrastructure unifiée permettant de gérer des simulations interactives, des simulations longues avec des traitements in-situ ainsi que des traitements post-mortem.

Un des principaux avantages de cette approche est un apprentissage plus aisé pour l'utilisateur. En effet, celui ci n'a besoin de se familiariser qu'avec un seul environnement plutôt qu'un environnement par contexte d'exécution. Nous espérons également que l'intégration d'outils usuels aidera l'utilisateur dans cette démarche. Un autre avantage majeur de cette approche est la réutilisation du code. Mis à part quelques composants spécifiques, de nombreux composants sont partagés entre les différents contextes d'exécution. Non seulement nous économisons beaucoup de code mais surtout la création de nouvelle applications devient plus facile car une base de code facilement réutilisable est déjà disponible. Cette réutilisation de code est largement favorisée par FlowVR et le modèle de programmation par composants.

Bien qu'ayant des avantages évidents, avoir un environnement unifié n'a pas que des bénéfices. La gestion des environnements d'exécution peut assez vite devenir

compliquée. En effet, pour des environnements spécifiques, il est souvent nécessaire de créer des composants spécifiques pour certaines fonctionnalités. C'est par exemple le cas pour le module de visualisation. Actuellement nous gérons une version "classique" de visualiseur mais également une version pour CAVE. En démultipliant les contextes, il peut devenir de plus en plus difficile de proposer une interface commune entre tous les composants partageant le même rôle comme la visualisation. La description d'une application disponible dans plusieurs contextes devient alors assez laborieuse pour gérer tous les cas spécifiques à chaque environnement (présence ou non de certains composants, instanciation d'un composant spécialisé si besoin, etc).

Les performances sont un autre enjeu majeur de cet environnement et plus précisément l'utilisation du parallélisme des machines. Les outils d'analyse proposés sont des bibliothèques usuelles couramment utilisées pour traiter des trajectoires. Elles ont cependant le désavantage majeur de ne pas être parallélisées. Nous avons montré que nous pouvons recréer un certain niveau de parallélisme en traitant les pas de temps en parallèle sur plusieurs instances d'un même module d'analyse. L'avantage de cette méthode est qu'elle est portable et peut s'exécuter aussi bien sur un ordinateur de bureau, une station de travail ou un cluster. Dans le dernier cas cependant, la solution n'est clairement pas idéale. En effet, dans un contexte in-situ, il est nécessaire de regrouper les données avant de pouvoir les traiter. Cette solution permet de construire rapidement et facilement des prototypes d'application. Par ailleurs, le support des nœuds dédiés permet de limiter le coût en performance de tels scénarios. Nous pouvons ainsi voir d'une part où se trouve les goulots d'étranglement dans le processus d'analyse et cibler nos efforts de parallélisation. À terme, pour les simulations longues sur de grandes machines parallèles, deux cas peuvent se produire. Soit les outils et algorithmes utilisés peuvent se paralléliser suffisamment et peuvent donc être utilisés dans un contexte in-situ ou *a minima* in-transit. Soit la parallélisation de l'outil n'est pas possible et dans ce cas ces traitements seront effectués dans une phase de post-traitement. Notre infrastructure peut nous permettre de déterminer quels sont les traitements plus adaptés à un contexte in-situ et quels sont ceux qui devront être exécutés en post-traitement.

Chapitre 7

Conclusion

7.1 Bilan des travaux

La quantité de données produite par les simulations devient de plus en plus problématique, à la fois pour les infrastructures qui tentent d'absorber ce déluge de données et les utilisateurs qui doivent ensuite les analyser. De ces analyses peuvent venir les découvertes de demain. Il est donc impératif pour la communauté scientifique de pouvoir les traiter le plus efficacement possible.

Les méthodes in-situ sont apparues comme une solution viable pour aider au traitement des données. Traiter les données alors qu'elles résident encore en mémoire permet d'éviter le goulot d'étranglement lié aux I/Os. Mettre en place de tels traitements n'est cependant pas une tâche aisée si l'on souhaite éviter des solutions *adhocs* et limiter l'impact de ces traitements sur les performances de la simulation. De nombreuses déclinaisons ont été proposées au fil des années, chacune proposant différentes stratégies pour exécuter ces traitements in-situ. Les retours d'expérience de ces différentes infrastructures ont montré qu'il n'existe pas, à priori, de solutions toutes faites permettant de garantir à la fois un coût minimal sur les performances d'une simulation donnée et une flexibilité suffisante pour construire des analyses complexes.

Initialement, les traitements in-situ ont été proposés pour résoudre le goulot d'étranglement lié aux entrées/sorties sur disque rencontré par les simulations de longue durée. Les traitements effectués étaient alors principalement liés à la réduction et réorganisation des données avant écriture ou bien à leur visualisation. Aujourd'hui, la palette des traitements effectués in-situ englobe l'ensemble des traitements habituellement effectués par les scientifiques dans la phase post-mortem (visualisation, filtrage, génération de statistiques, analyses complexes spécifiques, etc). Pour construire des scénarios in-situ réalistes, il devient nécessaire d'intégrer les experts du domaine cible dans le processus de création d'application in-situ.

La solution proposée dans cette thèse repose sur l'intergiciel FlowVR présenté dans le chapitre 3. Grâce à son modèle de programmation par composant, FlowVR permet de coupler des codes hétérogènes parallèles à l'intérieur d'une même application en créant un graphe de communication entre ces différents codes. En l'appliquant au contexte des traitements in-situ, il permet entre autre de coupler un code de simulation parallèle à un ensemble de modules d'analyse.

Gromacs, un des codes usuels de dynamique moléculaire, est notre applica-

tion cible. En collaboration avec des experts en simulation biologique, nous avons construit des applications in-situ interactives (Chapitre 4), de longue durée (Chapitre 5 et 6) ainsi que des applications post-mortems (Chapitre 6). Ces applications s'articulent autour de quatre thématiques principales.

Tout d'abord, nous nous sommes intéressés à l'accélération d'événements biologiques grâce à une méthode interactive de guidage (Chapitre 4). L'utilisateur a la possibilité grâce à un visualiseur en ligne et un bras haptique d'appliquer des forces sur un sous ensemble d'atomes afin de guider le système moléculaire dans une direction désirée. Grâce à notre méthode d'extraction de données asynchrone, nous avons été en mesure de manipuler interactivement des systèmes moléculaires allant jusqu'à 1.7 millions d'atomes en utilisant 512 cœurs. Nous avons utilisé cette méthode pour étudier le passage d'un complexe de fer dans le canal FepA. Ces expériences ont confirmé l'existence des chemins potentiels pour le complexe de fer identifiés dans d'autres travaux. Le niveau de force employé n'a toutefois pas permis de valider biologiquement ces expériences. Pour réaliser des simulations plus longues et permettre un niveau de force plus faible, nous avons proposé un système de guidage automatique basé sur les expériences interactives de l'utilisateur (Chapitre 6). Ce système nous a permis non seulement de reproduire automatiquement les expériences des utilisateurs mais également de les reproduire avec un niveau de force moins élevé.

Notre deuxième thématique concerne la génération de trajectoires. Nous avons déporté la phase d'écriture vers les traitements in-situ libérant ainsi Gromacs de cette tâche. L'écriture de trajectoires est réalisée de manière asynchrone. Ceci nous permet d'utiliser des fréquences d'écriture bien plus élevées pour un coût nettement inférieur à la méthode d'écriture native de Gromacs. Cette méthode a été testée jusqu'à 2048 cœurs (Chapitre 5). Notre approche nous permet également d'aller plus loin en écrivant de multiples trajectoires avec des fréquences et filtrages différents. Ceci permet aux biologistes d'obtenir des trajectoires plus légères, plus précises grâce à une fréquence plus élevée si besoin et enfin des trajectoires directement prêtes à être analysées (Chapitre 6).

La troisième thématique s'articule autour des analyses (Chapitre 6). Nous avons intégré certaines des bibliothèques usuelles du domaine (MDAnalysis, outils Gromacs) dans notre infrastructure. En réimplémentant l'interface des fichiers, nous permettons aux utilisateurs de réutiliser leurs scripts et lignes de commande habituels avec quelques modifications mineures. Ces outils sont accessibles pour tous les contextes d'exécution que cela soit pour des traitements in-situ ou post-mortem. Malheureusement, ces outils ne sont nativement pas ou peu parallélisés. Pour pallier ce problème, nous avons utilisé l'indépendance des pas de temps pour recréer un faible niveau de parallélisme en instanciant plusieurs modules d'analyse et en répartissant de manière cyclique les pas de temps sur ces modules. Ces modules d'analyse nous ont également permis d'observer les méthodes de traitement des trajectoire des biologistes et d'adapter notre workflow en conséquence.

La quatrième thématique se concentre sur la visualisation. Il s'agit d'un élément

clé du processus de compréhension d'un système moléculaire par les biologistes. Nous avons réimplémenté l'algorithme du Quicksurf, une représentation de la surface d'un système moléculaire, dans notre infrastructure (Chapitre 5). La décomposition de l'algorithme en composants ainsi que la flexibilité de notre approche nous ont permis de tester facilement différentes configurations de placement des modules. Nous pouvons ainsi adapter notre algorithme en fonction du coût sur la simulation ainsi que des autres analyses à effectuer.

Lorsqu'une infrastructure in-situ est proposée, la première inquiétude que peuvent émettre les utilisateurs est son coût sur les performances de la simulation. Pour cela, nous avons choisi d'adopter des stratégies visant à séparer au maximum les ressources de calcul allouées à la simulation et aux traitements in-situ (cœur dédié et nœud dédié). À travers différentes expériences et scénarios, nous avons montré que notre infrastructure pouvait passer à l'échelle jusqu'à 2048 cœurs pour un coût modéré de 9% mesuré dans le pire des cas (Chapitre 5) pour effectuer un algorithme de rendu en mode in-situ. De plus, ces scénarios nous ont permis de montrer l'intérêt de la flexibilité proposée par notre infrastructure pour à la fois construire des pipelines de traitement complexes mais aussi placer ces traitements de manière adéquate par rapport aux ressources de calcul à disposition et aux traitements à effectuer.

Le placement fin des modules sur les ressources de calcul ainsi que la gestion manuelle du graphe de l'application nous ont permis de créer des applications complexes pour un coût relativement faible. Le paradigme de *dataflow* permet d'exprimer de manière intuitive la démarche d'analyse itérative qu'effectuent les scientifiques dans leurs travaux de tous les jours. Ces éléments nous ont permis de construire une infrastructure unifiée pour visualiser et analyser des données issues de Gromacs dans un contexte in-situ ou post-mortem.

7.2 La place de l'in-situ à l'ère des machines petascales

La différence entre les capacités de production de données et les débits disponibles pour stocker ces données a toujours été une réalité. Avec l'arrivée des machines petascales, la problématique de la sauvegarde des données s'est faite plus pesante et a motivé en partie l'apparition des traitements in-situ. Pourtant, malgré les possibilités de telles méthodes, les traitements in-situ sont encore loin d'être utilisés couramment par les utilisateurs. Des systèmes sont en train d'émerger comme ADIOS qui est utilisé par une douzaine d'applications à grande échelle en production[18]. Toutefois ce système vise principalement à adresser le problème des I/Os en transformant les données avant écriture mais n'adresse pas le problème global des traitements in-situ.

Le coût d'une infrastructure est l'une des préoccupations majeures des utilisateurs. Effectuer des traitements in-situ nécessite un temps de simulation plus long et/ou des nœuds supplémentaires. L'utilisateur doit donc décider si la valeur ajoutée

produite par les traitements in-situ mérite une allocation de ressources supplémentaires.

Pour faciliter ce compromis, l'ensemble de la communauté a déployé beaucoup d'efforts pour proposer des systèmes minimisant l'impact des traitements in-situ sur les performances des simulations. Globalement, nous retrouvons deux types d'approche : les approches avec concurrence directe sur les ressources et les approches avec ressources dédiées.

Les approches avec ressources concurrentes comme Goldrush[96] visent à utiliser les ressources de la simulation lorsque celle-ci ne les utilise pas complètement. DataStager[7] utilise la même approche pour ordonnancer les communications lorsque la simulation n'utilise pas la carte réseau.

Les approches avec ressources dédiées utilisent soit des cœurs dédiés sur chaque nœud de la simulation, soit des nœuds dédiés alloués en plus, soit les deux. Le principe est de déporter les traitements sur des ressources non utilisées par la simulation afin de réduire les contentions sur les ressources processeurs et réseaux au prix de l'allocation de ressources supplémentaires. Réserver un cœur par nœud ou des nœuds dédiés peut s'avérer coûteux. En effet, sur des machines parallèles, beaucoup de nœuds sont actuellement constitués de huit cœurs. Réserver un cœur par nœud représente alors une perte de 12.5% de puissance de calcul pour la simulation. Ces approches sont toutefois justifiées par le fait qu'une simulation n'utilise pas complètement l'ensemble des cœurs disponibles et que par conséquent l'impact sur les performances de la simulation peut être bien moindre suivant les cas.

À l'heure actuelle, aucune de ces deux approches ne semblent émerger comme prédominante. Chacun de ces systèmes peut proposer des traitements in-situ pour un impact sur les performances de simulation d'environ 10%. Ces systèmes se distinguent alors dans leurs modes d'exécution des traitements in-situ et dans leurs modèles de programmation pour intégrer des traitements in-situ dans leurs infrastructures.

Ces efforts permettent de faciliter le choix de l'utilisateur sur la quantité de ressources à allouer pour des traitements in-situ mais un choix doit toujours être fait. Le nœud du problème reste que les traitements in-situ sont pour le moment optionnels. En effet, dans la très large majorité des cas, les utilisateurs peuvent simplement écrire leurs données sur disque et les traiter par la suite. Suivant les cas, des concessions sont faites sur la quantité de données sauvegardées mais ces concessions sont acceptables dans la majorité des cas. Tant que cette démarche sera possible, il semble difficile de passer outre le problème du coût des traitements in-situ.

7.3 ... et à l'ère des machines exascales

Les contraintes liées aux futures machines exascales pourraient bien changer la donne. Bien qu'il ne soit pas clair encore quelles seront les architectures de ces

machines exascales, nous pouvons déjà établir certains contours en nous basant sur différents rapports et articles [9, 4, 60]. La Table 7.1 issue de [60] synthétise les caractéristiques possibles des futures architectures exascales. Plusieurs de ces caractéristiques vont avoir un impact sur la place de l'in-situ dans le paysage du calcul haute performance.

Paramètre système	Petascale	Exascale 1	Exascale 2	Facteur
Puissance max	2Pf/s	1 Ef/s		500
Consommation	6 MW	$\leq$ 20 MW		3
Mémoire	0.3 Po	32-64 Po		100-200
Nombre de cœurs total	225K	1B x 10	1B x 100	40000-400000
Puissance par nœud	125 GF	1 TF	10 TF	8-80
Nombre de cœurs par nœud	12	1000	10000	83-830
Bande-passante réseau	1.5 Go/s	100 Go/s	1000 Go/s	66-660
Nombre de nœuds	18700	1000000	100000	50-500
Capacité de stockage	15 Po	300-1000 Po		20-67
Bande-passante stockage	0.2 To/s	20-60 To/s		10-30

Table 7.1 – Comparatif des architectures petascales actuelles et des possibles futures architectures exascales.

Tout d'abord, la différence entre les capacités de calcul et les capacités de stockage va atteindre un niveau sans précédent. En effet, la puissance de calcul devrait être multipliée par 500 tandis que la capacité de stockage et la bande associée ne devraient être multipliées que par 10 à 60. On estime qu'à ce niveau, il ne sera possible de sauvegarder que moins de 1% des données produites par les codes de simulation[9]. La quantité de données perdues ne sera alors plus acceptable pour un bon nombre d'applications.

Par conséquent, il va devenir nécessaire de traiter les données en in-situ. Des traitements comme le filtrage et/ou la compression de données vont alors devenir essentiels voir indispensables pour sauvegarder le plus de données possible mais pas seulement. Même avec de tels traitements, la quantité de données sauvegardées peut être insuffisante. Il deviendra alors de plus en plus nécessaire d'incorporer des analyses dans un environnement in-situ.

Il s'agit là d'un changement radical de la place des traitements in-situ pour les simulations car ils passent d'une utilisation optionnelle à une nécessité. Bien que toujours importante, la question du coût des infrastructures devrait devenir moins proéminente car ce coût deviendra nécessaire. Lorsqu'un utilisateur prévoit une simulation, il ne devra alors plus seulement prévoir les ressources et le temps d'exécution pour la simulation mais bien la simulation avec ses traitements in-situ associés. Plus intéressant encore, la principale préoccupation des utilisateurs pourrait devenir de savoir avec une infrastructure donnée quelles sont les analyses qu'il est possible d'exprimer. Dans ce contexte, nous espérons que les approches par *dataflow*

peuvent avoir un impact significatif. En effet, nous avons montré dans cette thèse qu'il est possible de créer des pipelines d'analyse complexes et variés.

Le rapport de l'ASCAC de 2013[4] va encore plus loin dans le rôle de l'in-situ. En effet, les expériences, les grands instruments de mesure ou encore les réseaux de capteurs produisent d'importantes quantités de données qu'il faut également analyser. Des traitements à la volée sont nécessaires de la même manière que pour les simulations. On retrouve de la même manière des besoins d'effectuer des chaînes de traitement sur les données, de transférer des données, de les stocker et de les archiver. Ces synergies font que l'on peut s'attendre à ce que les traitements in-situ prennent de plus en plus d'importance dans notre façon de gérer les données produites par les simulation et autres sources possibles.

Un autre aspect intéressant des futures architectures exascales est la composition des nœuds. Suivant les prédictions, on peut s'attendre à entre 1000 et 10000 cœurs par nœud soit 100 à 1000 fois plus que pour les nœuds des machines petascales. Dans le même temps, la quantité de mémoire embarquée sur chaque nœud devrait être multipliée par un facteur 100. Avec une architecture petascale, il est possible de simuler des grilles d'un trillion de cellules en prenant en compte les contraintes mémoires. En passant sur une machine exascale, on peut s'attendre à des grilles de 100 trillions de cellules. Sur une machine petascale, 1 trillion de cellules sont réparties sur environ 200 000 processus/threads soit 5 millions de cellules par thread. L'expérience montre qu'une telle répartition permet d'occuper efficacement les ressources de calcul. Pour une machine exascale en revanche, il n'y aurait que 1000 cellules par thread ce qui représente une charge de calcul bien trop faible.

En dehors des problèmes que posent ces chiffres pour les simulations elles-même, ce constat peut nous donner une direction pour les futures infrastructures in-situ. Pour les machines petascales, il n'est pas clair de savoir s'il vaut mieux utiliser une stratégie avec ou sans ressources dédiées (cœur ou nœud dédié). Pour les architectures exascales en revanche, les ressources dédiées et en particulier les cœurs dédiés, semblent préférables. En effet, en préservant le même niveau de parallélisme que les simulation, les traitements in-situ souffriraient de la même faible occupation des ressources. Par opposition, l'allocation de ressources dédiées peut permettre de réduire le nombre de threads pour la simulation et donc de mieux utiliser les ressources tout en bénéficiant des ressources libérées pour effectuer des traitements supplémentaires. Actuellement, il est difficile d'allouer un cœur sur huit par nœud car le coût en ressource est important. Avec des nœuds de 1000 cœurs voire plus, il devient beaucoup plus acceptable d'allouer quelques cœurs par nœud sans pénaliser de manière significative la simulation.

Bibliographie

[1] Blue Water Project. Technical report, NCSA.

[2] Interactive Molecular Dynamics with Gromacs. Technical report, Max Planck Institute for Biophyysical Chemistry.

[3] Parallel netcdf home page.

[4] Synergistic Challenges in Data-Intensive Science and Exascale Computing. Technical report, DOE.

[5] Unidata netcdf home page.

[6] H. Abbasi, G. Eisenhauer, M. Wolf, K. Schwan, and S. Klasky. Just in time : Adding value to the io pipelines of high performance applications with jitstaging. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, pages 27–36, New York, NY, USA, 2011. ACM.

[7] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. Datastager : Scalable Data Staging Services for Petascale Applications. In *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, HPDC '09, pages 39–48, New York, NY, USA, 2009. ACM.

[8] S. Ahern, E. Brugger, B. Whitlock, J. S. Meredith, K. Biagas, M. C. Miller, and H. Childs. Visit : Experiences with sustainable software. *arXiv preprint arXiv :1309.1796*, 2013.

[9] S. Ahern, A. Shoshani, K. Ma, A. Choudhary, T. Critchlow, S. Klasky, and V. Pascucci. Scientific discovery at the exascale : Report from the DOE ASCR 2011 workshop on exascale data management, analysis, and visualization. February 2011.

[10] J. Ahrens, B. Geveci, and C. Law. 36 paraview : An end-user tool for large-data visualization. *The Visualization Handbook*, page 717, 2005.

[11] J. Allard, V. Gouranton, L. Lecointre, S. Limet, E. Melin, B. Raffin, and S. Robert. Flowvr : a middleware for large scale virtual reality applications. In *Proceedings of Euro-par 2004*, Pisa, Italia, August 2004.

[12] J. Allard, C. Ménier, B. Raffin, E. Boyer, and F. Faure. Grimage : Markerless 3D Interactions. In *Proceedings of ACM SIGGRAPH 07*, San Diego, USA, August 2007. Emerging Technologies.

[13] J. Allard and B. Raffin. A shader-based parallel rendering framework. In *IEEE Visualization Conference*, Minneapolis, USA, October 2005.

[14] J. Allard and B. Raffin. Distributed physical based simulations for large vr applications. In *IEEE Virtual Reality Conference*, Alexandria, USA, March 2006.

[15] F. Archambeau, N. Méchitoua, and M. Sakiz. Code_saturne : a finite volume code for the computation of turbulent incompressible flows. *International Journal on Finite Volumes*, 1, 2004.

[16] J. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–9, Nov 2012.

[17] J. Biddiscombe, J. Soumagne, G. Oger, D. Guibert, and J.-G. Piccinali. Parallel Computational Steering for HPC Applications Using HDF5 Files in Distributed Shared Memory. *IEEE Transactions on Visualization and Computer Graphics*, 18(6) :852–864, June 2012.

[18] D. Boyuka, S. Lakshminarasimham, X. Zou, Z. Gong, J. Jenkins, E. Schendel, N. Podhorszki, Q. Liu, S. Klasky, and N. Samatova. Transparent I Situ Data Transformations in ADIOS. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 256–266, May 2014.

[19] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. Charmm : A program for macromolecular energy, minimization, and dynamics calculations. *Journal of Computational Chemistry*, 4(2) :187–217, 1983.

[20] B. R. Brooks, C. L. B. III, A. D. M. Jr., L. Nilsson, R. J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. S. D. Caves, Q. Cui, A. R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R. W. Pastor, C. B. Post, J. Z. Pu, M. Schaefer, B. Tidor, R. M. Venable, H. L. W. III, X. Wu, W. Yang, D. M. York, and M. Karplus. Charmm : The biomolecular simulation program. *Journal of Computational Chemistry*, pages 1545–1614, 2009.

[21] F. P. Brooks, Jr., M. Ouh-Young, J. J. Batter, and P. Jerome Kilpatrick. Project GROPEHaptic displays for scientific visualization. In *Siggraph '90*, pages 177–185, 1990.

[22] S. Byna, J. Chou, O. Rübel, Prabhat, H. Karimabadi, W. S. Daughton, V. Roytershteyn, E. W. Bethel, M. Howison, K.-J. Hsu, K.-W. Lin, A. Shoshani, A. Uselton, and K. Wu. Parallel i/o, analysis, and visualization of a trillion particle simulation. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 59 :1–59 :12, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[23] D. Case, V. Babin, J. Berryman, R. Betz, Q. Cai, D. Cerutti, T. Cheatham III, T. Darden, R. Duke, H. Gohlke, et al. Amber 14. 2014.

[24] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods. The amber biomolecular simulation programs. *Journal of Computational Chemistry*, 26(16) :1668–1688, 2005.

[25] M. Chavent, A. Vanel, A. Tek, B. Levy, S. Robert, B. Raffin, and M. Baaden. GPU-accelerated atom and dynamic bond visualization using hyperballs : A unified algorithm for balls, sticks, and hyperboloids. *Journal of Computational Chemistry*, 32(13) :2924–2935, 2011.

[26] J. Dayal, D. Bratcher, G. Eisenhauer, K. Schwan, M. Wolf, X. Zhang, H. Abbasi, S. Klasky, and N. Podhorszki. Flexpath : Type-based publish/subscribe system for large-scale science analytics. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 246–255, May 2014.

[27] O. Delalande, N. Férey, G. Grasseau, and M. Baaden. Complex Molecular Assemblies at Hand via Interactive Simulations. *Journal of Computational Chemistry*, 30(15) :2375–2387, 2009.

[28] C. Docan, M. Parashar, J. Cummings, and S. Klasky. Moving the code to the data - dynamic code deployment using activespaces. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 758–769, May 2011.

[29] C. Docan, M. Parashar, and S. Klasky. Enabling high-speed asynchronous data extraction and transfer using dart. *Concurrency and Computation : Practice and Experience*, 22(9) :1181–1204, 2010.

[30] C. Docan, M. Parashar, and S. Klasky. DataSpaces : an Interaction and Coordination Framework for Coupled Simulation Workflows. *Cluster Computing*, 15(2) :163–181, 2012.

[31] J. Dongarra, P. Luszczek, and A. Petitet. The LINPACK benchmark : past, present and future. *Concurrency and Computation : Practice and Experience*, 15(9) :803–820, 2003.

[32] M. Dorier, G. Antoniu, F. Cappello, M. Snir, and L. Orf. Damaris : How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-free I/O. In *CLUSTER - IEEE International Conference on Cluster Computing*. IEEE, Sept. 2012.

[33] M. Dorier, R. Sisneros, Roberto, T. Peterka, G. Antoniu, and B. Semeraro, Dave. Damaris/Viz : a Nonintrusive, Adaptable and User-Friendly In Situ Visualization Framework. In *LDAV - IEEE Symposium on Large-Scale Data Analysis and Visualization*, Atlanta, United States, Oct. 2013.

[34] G. Eisenhauer, M. Wolf, H. Abbasi, and K. Schwan. Event-based systems : Opportunities and challenges at exascale. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 2 :1–2 :10, New York, NY, USA, 2009. ACM.

[35] A. Esnard, N. Richart, and O. Coulaud. A Steering Environment for Online Parallel Visualization of Legacy Parallel Simulations. In *Distributed Simulation and Real-Time Applications, 2006. DS-RT'06. Tenth IEEE International Symposium on*, pages 7–14, Oct 2006.

[36] J. Esque, M. Sansom, M. Baaden, and C. Oguey. A case study of protein topology : how enterobactin modifies the water network through fepA. *submitted to PLoS One*.

[37] N. Fabian, K. Moreland, D. Thompson, A. Bauer, P. Marion, B. Geveci, M. Rasquin, and K. Jansen. The Paraview Coprocessing Library : a Scalable, General Purpose In Situ Visualization Library. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 89–96, Oct 2011.

[38] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo. Flash : An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *The Astrophysical Journal Supplement Series*, 131(1) :273, 2000.

[39] M. Gamell, I. Rodero, M. Parashar, J. C. Bennett, H. Kolla, J. Chen, P.-T. Bremer, A. G. Landge, A. Gyulassy, P. McCormick, S. Pakin, V. Pascucci, and S. Klasky. Exploring power behaviors and trade-offs of in-situ data analytics. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 77 :1–77 :12, New York, NY, USA, 2013. ACM.

[40] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. GROMACS 4 : Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *Journal of Chemical Theory and Computation*, 4(3) :435–447, 2008.

[41] W. Hsu. Segmented Ray Casting for Data Parallel Volume Rendering. In *Parallel Rendering Symposium, 1993*, pages 7–14, Oct 1993.

[42] W. Humphrey, A. Dalke, and K. Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14 :33–38, 1996.

[43] S. Izrailev, S. Stepaniants, B. Isralewitz, D. Kosztin, H. Lu, F. Molnar, W. Wriggers, and K. Schulten. Steered Molecular Dynamics. In *Computational Molecular Dynamics : Challenges, Methods, Ideas*, pages 39–65. Springer-Verlag, 1998.

[44] C. Johnson, S. Parker, C. Hansen, G. Kindlmann, and Y. Livnat. Interactive Simulation and Visualization. *Computer*, 32(12) :59–65, Dec 1999.

[45] S. Klasky, S. Ethier, Z. Lin, K. Martins, D. Mccune, and R. Samtaney. Grid-Based Parallel Data Streaming implemented for the Gyrokinetic Toroidal Code. In *In Supercomputing Conference (SC 2003*, page 24. IEEE Computer Society, 2003.

[46] M. Koutek, J. van Hees, F. H. Post, and A. F. Bakker. Virtual spring manipulators for particle steering in molecular dynamics on the responsive workbench. In *EGVE'02*, pages 53–ff. Eurographics Association, 2002.

[47] M. Krone, S. Grottel, and T. Ertl. Parallel contour-buildup algorithm for the molecular surface. In *Biological Data Visualization (BioVis), 2011 IEEE Symposium on*, pages 17–22, Oct 2011.

[48] M. Krone, J. E. Stone, T. Ertl, and K. Schulten. Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories. In *EuroVis 2012 Short Papers*, volume 1, 2012.

[49] M. Li, S. S. Vazhkudai, A. R. Butt, F. Meng, X. Ma, Y. Kim, C. Engelmann, and G. Shipman. Functional partitioning to optimize end-to-end performance on many-core architectures. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–12, Washington, DC, USA, 2010. IEEE Computer Society.

[50] P. Liu, D. K. Agrafiotis, and D. L. Theobald. Fast determination of the optimal rotational matrix for macromolecular superpositions. *Journal of Computational Chemistry*, 31(7) :1561–1563, 2010.

[51] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, M. Parashar, N. Samatova, K. Schwan, A. Shoshani, M. Wolf, K. Wu, and W. Yu. Hello ADIOS : the Challenges and Lessons of Developing Leadership Class I/O Frameworks. *Concurrency and Computation : Practice and Experience*, 26(7) :1453–1473, 2014.

[52] K. liu Ma, J. S. Painter, and C. D. Hansen. Parallel Volume Rendering Using Binary-Swap Compositing. *IEEE Computer Graphics and Applications*, 14 :59–68, 1994.

[53] L.-T. Lo, C. Sewell, and J. Ahrens. Piston : A portable cross-platform framework for data-parallel visualization operators. In H. Childs, T. Kuhlen, and F. Marton, editors, *EGPGV*, pages 11–20. Eurographics Association, 2012.

[54] J. Lofstead, R. Oldfield, C. Reiss, and T. Kordenbrock. Extending scalability of collective io through nessie and staging. In *in The Petascale Data Storage Workshop at Supercomputing*, 2011.

[55] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. Flexible IO and Integration for Scientific Codes Through the Adaptable IO System (ADIOS). In *6th international workshop on Challenges of large applications in distributed environments*, pages 15–24, 2008.

[56] B. Lorendeau, Y. Fournier, and A. Ribes. In-situ visualization in fluid mechanics using catalyst : A case study for code saturne. In *Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on*, pages 53–57, Oct 2013.

[57] W. E. Lorensen and H. E. Cline. Marching Cubes : A High Resolution 3D Surface Construction Algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA, 1987. ACM.

[58] J. S. Meredith, S. Ahern, D. Pugmire, and R. Sisneros. Eavl : The extreme-scale analysis and visualization library. In *EGPGV'12*, pages 21–30, 2012.

[59] N. Michaud-Agrawal, E. J. Denning, T. B. Woolf, and O. Beckstein. Mdanalysis : A toolkit for the analysis of molecular dynamics simulations. *J. Comput. Chem.*, 32 :2319–2327, 2011.

[60] K. Moreland. Oh, $#! Exascale! The Effect of Emerging Architectures on Scientific Discovery. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion :*, pages 224–231, Nov 2012.

[61] K. Moreland, U. Ayachit, B. Geveci, and K.-L. Ma. Dax Toolkit : a Proposed Framework for Data Analysis and Visualization at Extreme Scale. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 97–104, Oct 2011.

[62] U. Neumann. Communication Costs for Parallel Volume-Rendering Algorithms. *IEEE Comput. Graph. Appl.*, 14(4) :49–58, jul 1994.

[63] R. Oldfield, P. Widener, A. Maccabe, L. Ward, and T. Kordenbrock. Efficient data-movement for lightweight i/o. In *Cluster Computing, 2006 IEEE International Conference on*, pages 1–9, Sept 2006.

[64] T. Peterka, D. Goodell, R. Ross, H.-W. Shen, and R. Thakur. A Configurable Algorithm for Parallel Image-compositing Applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 4 :1–4 :10, New York, NY, USA, 2009. ACM.

[65] T. Peterka, R. Ross, W. Kendall, A. Gyulassy, V. Pascucci, H.-W. Shen, T.-Y. Lee, and A. Chaudhuri. Scalable parallel building blocks for custom data analysis. In *Proceedings of Large Data Analysis and Visualization Symposium LDAV'11*, Providence, RI, 2011.

[66] B. Petit, T. Dupeux, B. Bossavit, J. Legaux, B. Raffin, E. Melin, J.-S. Franco, I. Assenmacher, and E. Boyer. A 3D Data Intensive Tele-immersive Grid. In *ACM Multimedia (ACMM'10)*. ACM, Oct. 2010.

[67] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. KalÃ©, and K. Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26(16) :1781–1802, 2005.

[68] S. Pickles, R. Haines, R. Pinning, and A. Porter. A practical toolkit for computational steering. *Philosophical Transactions of the Royal Society A : Mathematical, Physical and Engineering Sciences*, 363(1833) :1843–1853, 2005.

[69] S. Pronk, S. Pall, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M. R. Shirts, J. C. Smith, P. M. Kasson, D. van der Spoel, B. Hess, and E. Lindahl. Gromacs 4.5 : a high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics*, 29(7) :845–854, 2013.

[70] F. M. Richards. Areas, Volumes, Packing, and Protein Structure. *Annual Review of Biophysics and Bioengineering*, 6(1) :151–176, 1977.

[71] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit–An Object-Oriented Approach To 3D Graphics*. Kitware, Inc., fourth edition, 2006.

[72] A. Singh, P. Balaji, and W.-c. Feng. Gepsea : A general-purpose software acceleration framework for lightweight task offloading. In *Proceedings of the 2009 International Conference on Parallel Processing*, ICPP '09, pages 261–268, Washington, DC, USA, 2009. IEEE Computer Society.

[73] J. Soumagne and J. Biddiscombe. Computational Steering and Parallel Online Monitoring Using RMA through the HDF5 DSM Virtual File Driver. In *Proceedings of the International Conference on Computational Science, ICCS 2011*, volume 4, pages 479–488, Singapore, June 2011.

[74] J. Soumagne, D. Kimpe, J. A. Zounmevo, and M. Chaarawi. Mercury : Enabling remote procedure call for high-performance computing. In *Cluster 2013*, Indianapolis, IN, 2013.

[75] V. Springel. The Cosmological Simulation Code GADGET-2. *Monthly Notices of the Royal Astronomical Society*, 364, 2005.

[76] J. E. Stone, J. Gullingsrud, and K. Schulten. A system for interactive molecular dynamics simulation. In *Symposium on Interactive 3D graphics*, pages 191–194, 2001.

[77] J. E. Stone, A. Kohlmeyer, K. L. Vandivort, and K. Schulten. Immersive molecular visualization and interactive modeling with commodity hardware. In *ISVC'10*, pages 382–393. Springer-Verlag, 2010.

[78] A. Tek, P. J. Bond, M. S. Sansom, and M. Baaden. Insights into membrane fusion from molecular dynamics simulations of {SNARE} proteins. *Biophysical Journal*, 102(3, Supplement 1) :499a –, 2012.

[79] R. Thakur, W. Gropp, and E. Lusk. Data sieving and collective i/o in romio. In *Frontiers of Massively Parallel Computation, 1999. Frontiers '99. The Seventh Symposium on the*, pages 182–189, Feb 1999.

[80] The HDF Group. Hierarchical Data Format, version 5, 1997-2014. http ://www.hdfgroup.org/HDF5/.

[81] The HDF Group. Parallel Hierarchical Data Format, version 5, 1997-2014. http ://www.hdfgroup.org/HDF5/PHDF5/.

[82] J. Torrellas. Architectures for Extreme-Scale Computing. *Computer*, 42(11) :28–35, Nov. 2009.

[83] G. Torrie and J. Valleau. Nonphysical sampling distributions in monte carlo free-energy estimation : Umbrella sampling. *Journal of Computational Physics*, 23(2) :187 – 199, 1977.

[84] T. Tu, H. Yu, L. Ramirez-Guzman, J. Bielak, O. Ghattas, K.-L. Ma, and D. O'Hallaron. From mesh generation to scientific visualization : An end-to-end approach to parallel supercomputing. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, pages 12–12, Nov 2006.

[85] D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, and H. J. C. Berendsen. Gromacs : Fast, flexible, and free. *Journal of Computational Chemistry*, 26(16) :1701–1718, 2005.

[86] V. Vishwanath, M. Hereld, and M. Papka. Toward simulation-time data analysis and i/o acceleration on leadership-class systems. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 9–14, Oct 2011.

[87] B. Whitlock, J. M. Favre, and J. S. Meredith. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*, EGPGV '11, pages 101–109, Aire-la-Ville, Switzerland, Switzerland, 2011. Eurographics Association.

[88] W. Wriggers, K. A. Stafford, Y. Shan, S. Piana, P. Maragakis, K. Lindorff-Larsen, P. J. Miller, J. Gullingsrud, C. A. Rendleman, M. P. Eastwood, R. O. Dror, and D. E. Shaw. Automated event detection and activity monitoring in long molecular dynamics simulations. *Journal of Chemical Theory and Computation*, 5(10) :2595–2605, 2009.

[89] O. Yildiz, M. Dorier, S. Ibrahim, and G. Antoniu. A performance and energy analysis of i/o management approaches for exascale systems. In *Proceedings of the Sixth International Workshop on Data Intensive Distributed Computing*, DIDC '14, pages 35–40, New York, NY, USA, 2014. ACM.

[90] H. Yu, C. Wang, R. Grout, J. Chen, and K.-L. Ma. In situ visualization for large-scale combustion simulations. *Computer Graphics and Applications, IEEE*, 30(3) :45–57, 2010.

[91] H. Yu, C. Wang, and K.-L. Ma. Massively Parallel Volume Rendering using 2-3 Swap Image Compositing. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–11, Nov 2008.

[92] G. Zhao, J. R. Perilla, E. L. Yufenyuy, X. Meng, B. Chen, J. Ning, J. Ahn, A. M. Gronenborn, K. Schulten, and C. Aiken. Mature HIV-1 Capsid Structure by Cryo-electron Microscopy and All-Atom Molecular Dynamics, 2013.

[93] F. Zheng, H. Abbasi, J. Cao, J. Dayal, K. Schwan, M. Wolf, S. Klasky, and N. Podhorszki. In-situ i/o Processing : A Case for Location Flexibility. In *Proceedings of the Sixth Workshop on Parallel Data Storage*, PDSW '11, pages 37–42, New York, NY, USA, 2011. ACM.

[94] F. Zheng, H. Abbasi, J. Cao, J. Dayal, K. Schwan, M. Wolf, S. Klasky, and N. Podhorszki. In-situ I/O Processing : A Case for Location Flexibility. In

Proceedings of the Sixth Workshop on Parallel Data Storage, PDSW '11, pages 37–42, New York, NY, USA, 2011. ACM.

[95] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. PreDatA - Preparatory Data Analytics on Peta-Scale Machines. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12, 2010.

[96] F. Zheng, H. Yu, C. Hantas, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, and S. Klasky. GoldRush : Resource Efficient in Situ Scientific Data Analytics Using Fine-grained Interference Aware Execution. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 78 :1–78 :12, New York, NY, USA, 2013. ACM.

[97] F. Zheng, H. Zou, G. Eisenhauer, K. Schwan, M. Wolf, J. Dayal, T.-A. Nguyen, J. Cao, H. Abbasi, S. Klasky, N. Podhorszki, and H. Yu. FlexIO : I/O Middleware for Location-Flexible Scientific Data Analytics. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 320–331, May 2013.