HAL
open science

Schémas d'adaptations algorithmiques sur les nouveaux supports d'éxécution parallèles

Sami Achour

To cite this version:

Sami Achour. Schémas d'adaptations algorithmiques sur les nouveaux supports d'éxécution parallèles. Algorithme et structure de données [cs.DS]. Université de Grenoble; Université de Tunis El-Manar. Faculté des Sciences de Tunis (Tunisie), 2013. Français. NNT : 2013GRENM086 . tel-01551787

HAL Id: tel-01551787

https://theses.hal.science/tel-01551787v1

Submitted on 30 Jun 2017

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.

FACULTE DES SCIENCES DE TUNIS

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Préparée dans le cadre d'une cotutelle entre l'Université de Grenoble et l'Université de Tunis El-Manar

Spécialité : **Informatique**

Arrêté ministériel : le 6 janvier 2005 -7 août 2006

Présentée par

Sami ACHOUR

Thèse dirigée par **Denis TRYSTRAM** et **Zaher MAHJOUB**
codirigée par **Wahid NASRI**

préparée au sein des Laboratoires **LIG** et **URAPOP**
et de l'école doctorale : **Mathématiques, Sciences et technologies de l'information, Informatique**

Schémas d'adaptations algorithmiques sur les nouveaux supports d'exécution parallèles

Thèse soutenue publiquement le **« 6 juillet 2013 »**,
devant le jury composé de :

Monsieur Riadh Robbana
Professeur de l'Université de Carthage, Président
Monsieur Laurent PHILIPPE
Professeur de l'Université de Franche-comté, Rapporteur
Monsieur Yahya SLIMANI
Professeur de L'Université de Mannouba , Rapporteur
Monsieur Denis TRYSTRAM
Professeur de l'Université de Grenoble, Membre
Monsieur Zaher MAHJOUB
Professeur de l'Université de Tunis El-Manar, Membre
Madame Sara BOUCHENAK
Maitre de Conférences(HDR) de l'université de Grenoble, Membre
Monsieur Wahid NASRI
Maitre de conférence de l'université de Tunis, Invité

Université Joseph Fourier / Université Pierre Mendès France / Université Stendhal / Université de Savoie / Grenoble INP

Dédicace

À

Mes deux petites : Dorra et Lina

À

Ma femme : Amira

À

Ma Mère : Selma

À

Mon frère : Abderrazak

Aux

Membres de ma famille et de ma belle famille

Remerciements

Pour atteindre ce stade, plusieurs obstacles ont été rencontrés, plusieurs moments difficiles ont été vécus. Le dépassement de ces obstacles et de ces moments n'était pas possible sans le concours de nombreuses personnes.
Je tiens à remercier chaleureusement, en particulier :

- Monsieur Riadh Robbana, professeur à l'université de Carthage, pour l'honneur qu'il me fait en présidant le jury,
- Monsieur Yahya Slimani, professeur à l'université de Manouba, et Monsieur Laurent Philippe, professeur à l'université de Franche-Comté, pour avoir bien voulu accepter de juger mon modeste travail,
- Madame Sara Bouchenak, maitre de conférences (HDR) à l'université de Grenoble, pour avoir bien voulu accepter de participer à ce jury.
- Monsieur Zaher Mahjoub, professeur à l'université de Tunis El Manar, mon directeur de thèse, pour le soutien durant la préparation de mon Mastère et tout au long de celle de la thèse, pour sa disponibilité, pour le temps qu'il m'a consacré afin de rendre ce manuscrit en son état actuel,
- Monsieur Denis Trystram, professeur de l'université de Grenoble, mon directeur de thèse, pour son accueil dans son équipe, pour ses précieux conseils et orientations dans les phases successives de ce travail, pour son soutien moral tout au long de cette thèse.
- Monsieur Wahid Nasri, maitre de conférences l'université de Tunis, mon encadrant, pour son soutien et sa collaboration tout au long de cette thèse, pour ses encouragements pour mener à bien les différentes parties de ce travail,
- Monsieur Luiz Angello Steffenel, maitre de conférences à l'université de Rennes, pour son accueil et sa gentillesse, pour sa collaboration dans les premières parties de ce travail.

Je tiens aussi à remercier ma femme, ma famille, ma belle famille et mes amis pour leurs encouragements et leur confiance sans cesse renouvelés.

Enfin, je tiens à remercier mes collègues de l'URAPOP(ancienne URAPAD), de L'ID-IMAG, de l'ENSIT(ancienne ESSTT) et de l'ISMAIK pour leurs encouragements.

Table des matières

Introduction générale **1**
1 Contexte général . . . 1
2 Contribution . . . 2
3 Organisation du document . . . 3

1 Architectures et programmation à haute performance **5**
1 Introduction . . . 6
2 Architectures à haute performance . . . 6
2.1 Classification . . . 6
2.2 Famille de processeurs . . . 8
2.3 Clusters . . . 9
2.4 Grilles . . . 11
2.5 Clouds . . . 12
2.6 Les réseaux d'interconnexion . . . 12
3 Programmation parallèle . . . 13
3.1 Programmation MPI . . . 13
3.2 Programmation OpenMP . . . 15
3.3 Programmation hybride MPI/OpenMP . . . 16
3.4 Programmation des GPU . . . 17
4 Bibliothèques parallèles de calcul numérique . . . 17
4.1 Bibliothèques et performance . . . 17
4.2 ScaLAPACK . . . 18
4.3 PetsC . . . 18
5 Conclusion . . . 19

2 Adaptativité et modélisation de performance pour des applications à haute performance **20**
1 Introduction . . . 21
2 Performance des applications informatiques . . . 21
2.1 Facteurs affectant la performance . . . 21
2.2 Evaluation de performance . . . 22
2.3 Prédiction de performance . . . 25
3 Applications adaptatives . . . 27
3.1 Définition . . . 28
3.2 Classification . . . 28
3.3 Techniques adaptatives . . . 29
3.4 Caractérisation de l'adaptativité . . . 31
4 Etat de l'art . . . 32
4.1 Environnement séquentiel . . . 32
4.2 Environnement parallèle . . . 34
4.3 Récapitulatif . . . 36
5 Conclusion . . . 38

3 Framework adaptatif **39**
1 Introduction . . . 40
2 Description du Framework . . . 40
2.1 Motivation . . . 40
2.2 Aperçu du framework . . . 41
2.3 Découverte de la plate-forme . . . 41
2.4 Modélisation de performance . . . 43
2.5 Exécution adaptative . . . 44
3 Etude de cas . . . 45
3.1 Problème cible et état de l'art . . . 45
3.2 Alternatives retenues pour la multiplication matricielle (MM) . . . 46
3.3 Modélisation de performance des alternatives de la MM . . . 46
3.4 Performances des alternatives de la MM . . . 48
3.5 Prédiction de performance . . . 50
3.6 Critique . . . 50
4 Modélisation expérimentale de la performance des noyaux de calcul . . . 51
4.1 Mesure de performance des noyaux séquentiels . . . 52
4.2 Génération de modèles . . . 52
4.3 Validation expérimentale . . . 53

4.4 Critique . . . 56
5 Conclusion . . . 56

4 Framework de modélisation de performance 58
1 Introduction . . . 59
2 Environnement et aperçu de notre framework . . . 59
2.1 Plates-formes cibles . . . 59
2.2 Applications cibles . . . 61
2.3 Contraintes fonctionnelles . . . 61
2.4 Éléments de notre solution . . . 62
3 Découverte de programmes . . . 63
3.1 Objectif . . . 63
3.2 Technique de découpage . . . 64
3.3 Critères de découpe . . . 65
4 Découverte de plates-formes . . . 65
4.1 Motivation . . . 65
4.2 Objectif . . . 66
4.3 Détermination des clusters . . . 66
4.4 Validation . . . 67
5 Modélisation de performance . . . 67
5.1 Modélisation des calculs . . . 68
5.2 Modélisation des communications . . . 71
6 Conclusion . . . 76

5 Prédiction de performance d'applications MPI 77
1 Introduction . . . 78
2 Etat de l'art des simulateurs MPI . . . 78
3 MPI-PERF-SIM . . . 79
3.1 Objectifs et démarche . . . 79
3.2 Entrées du module . . . 80
3.3 Génération du graphe de tâches . . . 84
3.4 Prédiction des coûts des tâches . . . 84
3.5 Prédiction du temps de complétion . . . 91
4 Application de MPI-PERF-SIM . . . 95
4.1 Application cible . . . 95
4.2 Plate-forme de validation . . . 95
4.3 Résultats expérimentaux . . . 96

5 Conclusion 98

Conclusion générale et perspectives **99**

Bibliographie **102**

Annexe A : MPI **111**

Annexe B : Régression polynômiale **115**

Annexe C : Découpage de programmes **118**

Table des figures

1 Modèle adaptatif global . . . 3
2 Architecture d'un GPU . . . 10
3 Composition d'un cluster . . . 10
4 Fork/join dans un programme OpenMP . . . 16
5 MPI versus MPI/OpenMP . . . 17
6 Architecture de ScaLAPACK . . . 19
7 Temps mesurés en utilisant la commande *time* . . . 23
8 Classes d'adaptativité . . . 29
9 Framework adaptatif . . . 42
10 Distribution des matrices pour l'algorithme standard . . . 46
11 Distribution des matrices pour l'algorithme Str_Cannon . . . 47
12 Temps d'exécution des différentes alternatives sur Grid'5000 . . . 49
13 Temps d'exécution des différentes alternatives sur le cluster de l'ESSTT . . . 49
14 Erreur de prédiction du noyau *cblas_dgemm* . . . 55
15 Erreur de prédiction du noyau *clapack_dtrtri* . . . 56
16 Plate-forme Multi-clusters . . . 60
17 Modèle global . . . 64
18 Régression simple vs régression raffinée . . . 72
19 Pseudo-Code des benchmarks des routines MPI . . . 74
20 Framework de prédiction de performance de programmes parallèles . . . 81
21 Architecture de MPI-PERF-SIM . . . 82
22 Performances des réseaux . . . 83
23 Performances des noyaux de calcul . . . 83
24 Exemple de génération de graphe de tâches . . . 85

25 Prédiction de performance des routines pt/pt bloquantes 87
26 Prédiction de performance des routines pt/pt non bloquantes 88
27 Poly-algorithme de la routine MPI_Bcast 88
28 Poly-algorithme de la routine MPI_Allgather 88
29 Emulateur de performance des routines collectives 89
30 Prédiction de performance de la routine MPI_Bcast 91
31 Prédiction de performance de la routine MPI_Scatter 92
32 Prédiction de performance de la routine MPI_Gather 93
33 Erreur de prédiction de MPI_PERF_SIM pour le produit matriciel 97
34 Exemple de régression avec R sous Linux 117
35 Exemple de découpe de programme 119
36 Découpage de programme selon une variable 120
37 Découpage de programme selon une procédure 120

Liste des tableaux

1 Machines de Grid'5000 11
2 Comparatif des différentes approches de modélisation de performance 27
3 Caractéristiques de quelques applications adaptatives 37
4 Modèles de performance théoriques des différentes versions de la MM 47
5 Machines de test 48
6 Processeurs des configurations de test 48
7 Meilleurs algorithmes de la MM 50
8 Paramètres de performance des plates-formes de test 51
9 Machines de validation 53
10 Paramètres du modèle polynomial du noyau *cblas_dgemm* 54
11 Paramètres du modèle polynomial du noyau *clapack_dtrtri* 54
12 Temps mesurés (M) vs temps estimés (E) pour le noyau *cblas_dgemm* 54
13 Temps mesurés (M) vs temps estimés (E) pour le noyau *clapack_dtrtri* 55
14 Résultat de la correspondance entre machines et clusters 67
15 Latences intra et inter-clusters 67
16 Modèles des régressions simple et raffinée 71
17 Temps Mesurés vs estimés pour les communications pt/pt 87
18 Caractéristiques des machines d'expérimentation 90
19 Configurations expérimentales : nombre de processeurs de chaque cluster 90
20 Caractéristiques et affiliations des nœuds d'expérimentation 96
21 Nombre de nœuds pour chaque configuration 96
22 Erreur de prédiction du comportement de la MM parallèle 96

Liste des algorithmes

1 Découverte de plate-forme 43
2 Clustering de la plate-forme 66
3 Régression Polynomiale Simple(*Polynomial_Regression*) 70
4 Régression Raffinée 70
5 Régression linéaire progressive 75
6 Prédiction du temps d'une communication pt/pt 86
7 Simulation du comportement d'un programme parallèle 94
8 Procédure d'ordonnancement de tâches (Schedule_Tasks) 94
9 Multiplication Matricielle parallèle 95

Introduction générale

1 Contexte général

Durant ces dernières années, le domaine du calcul parallèle et distribué s'est considérablement développé [73], en particulier au niveau architectures, donnant naissance à une grande variété de supports d'exécution comme les grappes, les grilles et les clouds. Ces plates-formes représentent une alternative raisonnable aux machines parallèles monolithiques traditionnelles, et sont devenues les supports de calcul les plus compétitifs pour résoudre une grande variété d'applications à haute performance et ce, grâce au bon ratio coût/performance garanti par de telles plates-formes. Aujourd'hui, en raison de plusieurs facteurs tels que l'hétérogénéité inhérente, la diversité et l'évolution continuelle des supports d'exécution parallèles existants, il est devenu quasiment impossible d'adopter la méthode traditionnelle pour le développement d'applications parallèles qui consiste à générer manuellement une version optimisée (pour un problème donné) pour chaque type et/ou version de machine. De plus, le grand nombre de paramètres caractérisant les nouvelles architectures et la diversité des réseaux d'interconnexion représente un défi pour modéliser et optimiser les performances, en particulier en ce qui concerne les communications.

L'approche adaptative est une réponse prometteuse à ces challenges [17, 31, 44, 79, 84]. L'idée est d'adapter les algorithmes et leurs exécutions aux architectures cibles. Cette adaptation peut se faire automatiquement dans le programme en considérant plusieurs paramètres pouvant influencer ses performances (données du problème et support d'exécution). Pour que ces techniques garantissent les performances recherchées, il est indispensable de faire recours à des modèles de performance afin d'évaluer l'efficacité d'une application et de représenter correctement les paramètres relatifs au problème à résoudre ainsi qu'à la plate-forme cible. Ces paramètres sont nécessaires pour modéliser au mieux et prédire avec précision les performances des algorithmes utilisés et de là, arrêter les bonnes décisions. En effet, en termes de performance, le fait d'exécuter

un algorithme parallèle sur un ensemble de processeurs ne garantit pas toujours de bonnes performances. La conception d'algorithmes performants passe nécessairement par la compréhension des caractéristiques et le fonctionnement de ces algorithmes ainsi que de l'environnement d'exécution. Pour la prédiction de performances de programmes parallèles, plusieurs méthodes peuvent être utilisées. La méthode la plus simple est celle se basant sur des formules analytiques décrivant le comportement de ces programmes sur la plate-forme considérée [17, 29, 53]. Malgré sa simplicité et sa rapidité, cette méthode souffre de plusieurs inconvénients ayant trait à la précision, la portabilité, le support d'environnements hétérogènes et les programmes de grandes tailles. Une seconde méthode se basant sur des expérimentations consiste à utiliser l'historique des exécutions du programme en question sur la plate-forme considérée [9, 42, 84]. L'exploitation de ces données est généralement effectuée en utilisant des techniques de la statistique, du data-mining ou de l'intelligence artificielle. Toutefois, cette méthode reste toujours liée aux programmes et aux plates-formes. De plus, elle s'avère inutilisable dans le cas d'environnements complexes et hétérogènes et ce, vu le caractère fortement combinatoire des expérimentations à réaliser.

Une troisième méthode de prédiction de performance est la simulation [10, 11, 77, 92]. Cette dernière remédie généralement aux points faibles des deux autres méthodes (analytique et expérimentale). Néanmoins, son inconvénient majeur est le temps requis pour faire une telle simulation, surtout dans le cas de programmes complexes et de plates-formes massives.
Il est à noter ici, que nulle de ces méthodes de prédiction de performance n'est convenable pour être utilisée pour le contexte d'adaptativité, vu les deux contraintes qu'impose ce dernier, à savoir la rapidité et la précision des estimations.

2 Contribution

Le présent travail consiste en la conception d'un framework générique pour le développement d'applications parallèles adaptatives et performantes sur des architectures parallèles hiérarchiques et dédiées. Ce framework doit tirer avantage des différentes approches de modélisation de performance afin de satisfaire les besoins cruciaux imposés par le contexte d'adaptativité. La solution proposée dans cette thèse est schématisée dans la figure 1. Cette solution est composée de deux principales phases, à savoir celle de l'installation de l'application adaptative et celle de l'exécution. Dans la première phase, plusieurs modules sont à exécuter :

- Module de découverte de la plate-forme : il est responsable de la détermination de la hiérarchie de la plate-forme. En d'autres termes, il permet de détecter les clusters de la plate-forme et il associe chaque machine à son cluster.
- Module de découverte de programmes : responsable de la découverte des principales structures que contiennent les différents programmes de l'application adaptative, en particulier

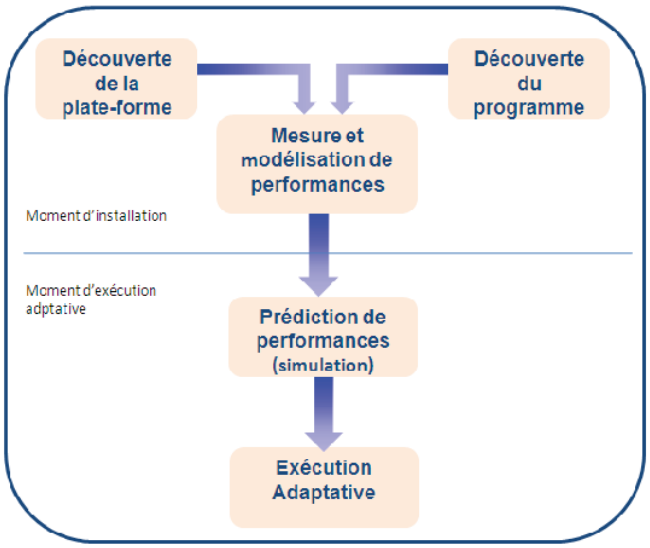


Figure 1 – Modèle adaptatif global

les différents noyaux de calcul et les différentes routines de communication utilisées dans ces programmes.

– Module de modélisation de performance : basé sur les résultats des deux modules précédents, il effectue une phase de mesure de performance (*benchmarking*) pour les routines de communication point à point et les noyaux de calcul. Ces mesures seront utilisées pour générer les différents modèles de communication et de calcul afin de les utiliser au moment de l'exécution.

Précisons qu'au moment de l'exécution, une phase de simulation rapide des programmes parallèles est effectuée. L'objectif de cette phase étant d'estimer la performance de chaque alternative candidate à être utilisée dans l'exécution réelle. Selon la technique adaptative retenue, une variante sera désignée à s'exécuter réellement sur la plate-forme.

3 Organisation du document

Le manuscrit présentant nos travaux comporte cinq chapitres organisés comme suit :

Dans le premier chapitre, nous présentons le contexte matériel et logiciel en relation avec notre problématique. Nous décrivons, en premier lieu, les principales architectures parallèles émergentes telles que les clusters de multi-cœurs, les clusters de GPU et les grilles. Puis, nous présentons les principales approches de programmation parallèle, à savoir MPI pour la programmation des machines à mémoire distribuée, OpenMP pour les machines à mémoire partagée et CUDA pour la programmation des GPU.

Le second chapitre est consacré à l'étude des différentes approches de modélisation de performance (analytique, expérimentale, et par simulation). Nous focalisons aussi notre étude, sur l'approche adaptative tout en essayant de classifier les différentes techniques adoptant cette approche et qui sont appliquées pour le cas des applications informatiques (séquentielles et parallèles). Une étude bibliographique sur plusieurs frameworks et applications informatiques adoptant l'approche adaptative est établie à la fin de ce chapitre.

Dans le chapitre trois, nous détaillons notre solution adaptative initiale basée sur la modélisation analytique des différentes parties du programme parallèle (calcul et communication). Une validation de cette solution est réalisée sur le problème du produit de matrices dans des environnements d'exécution hétérogènes. Une critique de cette solution est ensuite élaborée pour souligner ses défaillances. Dans la dernière section de ce chapitre, nous présentons les éléments d'une solution remédiant à un ensemble de ces défaillances. Cette solution consiste à utiliser des expérimentations dans le processus de génération de modèles pour les parties séquentielles (de calcul) contenus dans un programme parallèle.

Nous proposons dans le quatrième chapitre un modèle de prédiction de performance de programmes parallèles pour le cas de plates-formes hétérogènes et hiérarchiques. Dans ce chapitre, nous précisons l'étendue de cette solution des points de vue matériel et logiciel. Ainsi, nous donnons un aperçu sur cette solution basée sur deux phases (phase d'installation et phase d'exécution). Le reste du chapitre est consacré aux détails des différents modules de la phase d'installation, à savoir les modules de découverte de plate-forme et de programme et le module de modélisation de performance des noyaux de calcul et de communication point à point.

Le chapitre cinq est consacré à la présentation de la deuxième phase de notre framework de prédiction de performance. Cette phase, effectuée au moment de l'exécution, consiste en la simulation de programmes parallèles de type MPI afin de déterminer leurs performances. Dans ce chapitre, nous présentons les étapes et les détails de notre solution nommée MPI-PERF-SIM (*MPI Performance Simulation*) et qui consiste à prédire le coût parallèle de programmes MPI sur des machines hétérogènes hiérarchiques.

Enfin, nous récapitulons les principaux points de notre contribution et décrivons quelques perspectives futures.

Chapitre 1

Architectures et programmation à haute performance

1 Introduction

Les besoins croissants des applications scientifiques en terme de calcul ainsi que les avancées technologiques rapides marquant cette décennie dans les domaines de communication et nanotechnologie constituent une source de progrès pour le domaine du calcul parallèle. En effet, avec les réseaux de communication actuels (Myrinet et Infiniband) offrant des bandes passantes mesurées en termes de Gbps et avec le niveau élevé d'intégration dans les microprocesseurs marqué par l'apparition de processeurs dotés de performances équivalentes aux super-calculateurs des débuts des années 90 (ayant une puissance d'une dizaine de Gflops), la construction de machines parallèles ayant une performance de l'ordre de plusieurs Tflops [1] (voire Pflops [2]), pour traiter des applications très gourmandes en terme d'usage CPU, est devenu une tâche réalisable.
Le domaine des architectures parallèles, ces dernières années, a connu non seulement un progrès fulgurant dans le développement des processeurs et des réseaux d'interconnexion, mais il a connu aussi un enrichissement par les processeurs graphiques (GPU).
Avec cette évolution sur le plan architectural, la programmation des architectures parallèles a connu elle même un progrès afin de suivre cette évolution et surtout afin de proposer des standards pour ce genre de programmation. Plusieurs standards ont été proposés dans ce contexte, les plus importantes sont MPI pour la programmation des architectures à mémoire distribuée, OpenMP pour la programmation des architectures à mémoire partagée et CUDA pour la programmation des GPU.
Dans ce chapitre, nous nous intéressons à la présentation des aspects matériels et logiciels qui ont attribué le libellé performance au domaine de calcul parallèle. Ainsi, nous présentons, dans la section 2, les principales architectures parallèles marquant le début de cette décennie tout en essayant de les classifier. La section 3 décrit les principaux modèles de programmation utilisés par les développeurs d'applications parallèles, à savoir MPI pour les architectures à mémoire distribuée, OpenMP pour les machines mémoire partagée et CUDA pour la programmation des GPUs. Enfin, la section 4 est dévolue à la description de quelques bibliothèques de calcul numérique.

2 Architectures à haute performance

2.1 Classification

Durant les dernières années, et avec l'évolution des architectures parallèles, plusieurs classifications de ces dernières ont été proposées. Ces classifications ont presque les mêmes objectifs qui peuvent être résumés en ce qui suit :

– La prise en considération des différentes architectures présentes sur le marché.

1. 1 Tflops = 10^{12} opérations en virgule flottante par seconde
2. 1 Pflops = 10^{15} opérations en virgule flottante par seconde

– L'efficacité et la simplicité de la caractérisation de chaque architecture.

La classification qui demeura fort longtemps la norme dans le domaine du calcul parallèle est celle de Flynn [39]. Elle se base principalement sur le couple flux d'instructions, flux de données (simples ou multiples) pour distinguer quatres combinaisons :

(a) SISD (Single Instruction stream, Single Data stream) : couvrant les machines séquentielles qui ne supportent pas le "threading" comme les machines à base de processeurs 8080.

(b) SIMD (Single Instruction stream, Multiple Data stream) : ces machines sont composées de plusieurs processeurs (pour effectuer les traitements arithmétiques et logiques) et d'un seul processeur contrôleur. Chaque processeur traite les données qui lui ont été affectées conformément aux directives du processeur contrôleur. Vu leur organisation, ces processeurs sont appelés, selon le cas, tableaux de processeurs ou processeurs vectoriels.

(c) MISD (Multiple Instruction stream, Single Data stream) : le même flux de données est traité simultanément par plusieurs flux d'instructions. Une unité arithmétique et logique utilisant le pipelining est considérée de cette classe d'architectures malgré que ça représente une extension de la définition du flux de données.

(d) MIMD (Multiple Instruction stream, Multiple Data stream) : contient plusieurs processeurs, chacun exécute son propre flux d'instructions pour traiter le flux de données qui lui est affecté. La plus part de ces machines utilisent en réalité le mode SPMD (Single Program Multiple Data) i.e. tous les processeurs démarrent avec le même exécutable et du moment que ces exécutables peuvent avoir plusieurs chemins d'exécution à travers les instructions conditionnelles, ces processeurs ne restent plus en une synchronisation complète et tendent à se comporter comme les processeurs d'une architecture SIMD.

Notons que la majorité des machines parallèles actuelles sont de la classe MIMD, d'où le recours à d'autres types de classifications plus fines qui se basent sur d'autres critères. Notamment, la mémoire a été utilisée comme l'un des plus importants critères de classification de la classe MIMD. Ainsi, quatre sous-classes sont définis [73] :

– Machines à mémoire centralisée partagée : ces machines sont aussi connus sous les noms SMP (*Symmetric Multi-processor*) et UMA (*Uniform Memory Access*). De telles machines sont composées par un faible nombre de processeurs connectés à une seule mémoire. L'accès à cette mémoire est exclusif vu que le bus mémoire est unique.
– Machines à mémoire distribuée partagée : ces machines sont connues sous le nom de DSM (*Distributed Shared Memory*). Une plate-forme de cette classe est composée de machines physiquement distribuées, chacune étant dotée de sa propre mémoire vive. L'ensemble des mémoires constitue un seul espace d'adressage et ceci est réalisé à travers divers mécanismes.
– Machines à mémoire distribuée : dans cette classe, une machine est composée d'un ensemble de nœuds et d'un réseau les interconnectant.

– Machines hybrides à mémoire distribuée-paratgée : ce sont les machines composés de plusieurs nœuds interconnectés par un réseau et où chacune de ces nœuds est composé de plusieurs processeurs partageant une même mémoire. Les grappes de SMPs est le bon exemple de ces machines.

Une classification récente a été proposée par Dongarra et al. [37]. Cette Classification se base sur quatre types d'informations :

(a) Le type de "*clustering*" : cette information peut avoir seulement deux valeurs(cluster commercial ou système monolithique).

(b) Le Nommage : cette information peut avoir trois valeurs différentes(distribué, partagé ou à cache cohérent).

(c) Le type du parallélisme supporté par l'architecture : notons que les types qui peuvent être différenciés sont nombreux (multi-threads, vectoriel, processus séquentiels communicants, passage de message, producteur-consommateur, processus parallèles, . . .).

(d) Latence : ce paramètre peut avoir plusieurs valeurs (cache, vectorielle, multi-thread, mémoire, . . .).

2.2 Famille de processeurs

2.2.1 Processeurs multi-cœurs

Un processeur multi-cœur est composé d'au moins deux cœurs (ou unités de calcul) gravés sur la même puce. Ce type d'architecture est devenu la solution incontournable pour fabriquer des processeurs pouvant atteindre des performances importantes tout en réduisant la quantité de chaleur dissipée par effet Joule. Les premiers processeurs multi-cœurs qui ont été commercialisés sont les POWER4 d'IBM en 2001. Quatre ans plus tard, Intel et AMD introduisent leurs multi-cœurs au marché [3]. Il est évident que les constructeurs de plates-formes parallèles ont bien profité de ce genre de processeurs pour construire des machines parallèles ayant des rapports performance/coût de plus en plus élevés. La preuve peut être vue nettement dans les dernières listes du Top500 [4] des machines les plus puissantes au monde et qui ne comportent que des machines multi-cœurs. Il est à noter que les industriels cherchent toujours à multiplier le nombre de cœurs par processeur, et c'est dans ce contexte que se place le projet Tera-Scale [5] d'Intel qui a comme objectif de construire un processeur ayant une performance de l'ordre du Teraflops.

3. http://fr.wikipedia.org/Microprocesseur_multi-c\T1\oeur
4. www.top500.org
5. http://techresearch.intel.com/ProjectDetails.aspx?Id=151

2.2.2 Processeurs vectoriels

Les processeurs vectoriels sont des processeurs pouvant traiter un vecteur entier en une instruction. Ces processeurs ont été la clé de performance pour les premiers super-ordinateurs du type Cray [6] et même pour des plates-formes assez récentes comme l'Earth-simulator qui a marqué ce début de siècle en détenant la tête de la liste du Top500 pendant plus de 2 ans successifs [7].

2.2.3 Processeurs graphiques

Un processeur graphique communément connu sous le nom GPU (*Graphical Processing Unit*) est un processeur conçu pour décharger les processeurs (CPUs) des traitement graphiques. Cependant, vue que les opérations traitées par ces processeurs ne sont qu'une forme d'arithmétique, les GPUs se sont progressivement développés vers un modèle utilisable aussi pour les calculs non graphiques.
Les GPUs actuels supportent à la fois les deux paradigmes de programmation SIMD et SPMD (cf section 3.1.2). En effet, les processus légers (threads) exécutés sur un GPU ne sont pas complètement indépendants, ils sont ordonnées en blocs de threads, où chaque bloc exécute la même instruction (c'est le SIMD). Il est aussi possible d'ordonnancer sur les GPUs le même flux d'instructions sur plusieurs blocs. Dans ce cas, les blocs de threads sont hors synchronisation (c'est le SPMD).
Cette conception logicielle est apparente sur le matériel. Par exemple, un GPU de type Nvidia comporte 8 clusters de processeurs de texture TPC (*Texture Processor Cluster*). Chacun de ces TPC comporte deux multi-processeurs de flux SM (*Streaming Multiprocessor*). Un SM contient 8 processeurs de flux SP (*Streaming Processor*). (Voir figure 2)

2.3 Clusters

Un cluster (grappe, voir figure 3) est constitué de plusieurs nœuds interconnectés à travers un réseau [59]. Chaque nœud contient un (ou plusieurs) processeur(s), une mémoire, un disque. Les nœuds sont interconnectés à travers un réseau rapide et dédié. Le cluster contient généralement un nœud particulier qui est le nœud frontal. Ce nœud sert pour les utilisateurs distants du cluster (en utilisant Internet) comme nœud d'accès pour compiler leurs programmes et les exécuter sur les nœuds du cluster. Le cluster peut comporter un serveur de fichiers pour faciliter les opérations de partages de ces fichiers.
Le cluster est dit à mémoire distribuée si ses nœuds sont des machines monoprocesseurs. Il est dit à mémoire hybride s'il est composé de nœuds multi-processeurs comme les SMPs (mémoire

6. http://en.wikipedia.org/wiki/Cray-1
7. http://i.top500.org/system/167148

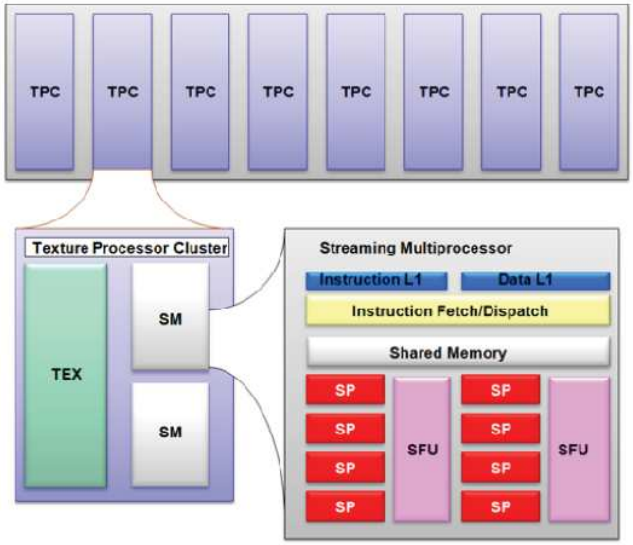


Figure 2 – Architecture d'un GPU

partagée intra-nœud et distribuée inter-nœuds).

Les divers modes de programmation des clusters sont détaillés dans la section 3.

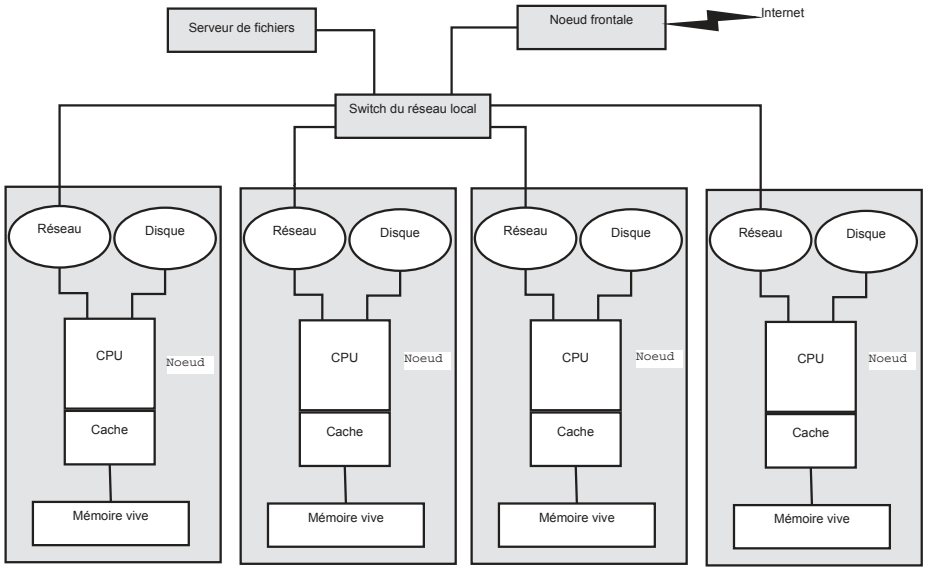


Figure 3 – Composition d'un cluster

Site	AMD Opteron	Intel Xeon	Nbre de processeurs	Nbre de cœurs
Bordeaux	226	102	328	554
Grenoble		232	232	928
Lille	52	148	200	696
Luxembourg		44	44	176
Lyon	270		270	270
Nancy		328	328	1320
Orsay	680		680	680
Reims	88		88	1056
Rennes	80	244	324	1804
Sophia	212	90	302	784
Toulouse	160		160	320
		Total	**2956**	**8580**

Tableau 1 – Machines de Grid'5000

2.4 Grilles

D'un point de vue architectural, une grille est similaire à un cluster, sauf que ses nœuds ne sont pas localisés dans le même emplacement et ne sont pas connectés par le même réseau. En effet, les nœuds d'une grille sont localisés dans plusieurs sites géographiquement éloignés et connectés à travers une combinaison de réseaux locaux et de l'Internet.[59]
Les grilles sont souvent mises en place par des consortium de sociétés, des universités, des établissements de recherche et des agences gouvernementales e.g. l'OSG (*Open Science Grid*) qui est dédiée à la recherche scientifique à large échelle comporte des milliers de processeurs répartis dans 85 instituts de 7 pays dans les continents Américain, Européen et Asiatique.
Il existe aussi un autre type de grilles se basant sur le calcul volontaire des utilisateurs de la toile. Le projet SETI@HOME[7] représente le bon exemple de ce type de grilles.

La plupart des expérimentations présentées dans ce mémoire ont été réalisées sur la grille expérimentale Grid'5000 [8]. Cette plate-forme est actuellement répartie sur onze sites dispersés géographiquement dans toute la France. L'interconnexion de ces sites est assurée par le réseau RENATER (Réseau National d'Enseignement et de Recherche) qui est un réseau à haut débit. Les nœuds de cette grille sont à base de plusieurs versions des processeurs AMD Opteron et Intel Xeon. Les réseaux d'interconnexion intra-sites sont à base des réseaux Infiniband et Myrinet. Actuellement le total des processeurs dans cette grille est environ 3000 soit plus que 8500 cores (voir tableau 1).

8. www.grid5000.fr

2.5 Clouds

Le terme cloud fait référence habituellement à un modèle basé sur Internet et dans lequel les données ne sont pas maintenues par l'utilisateur. Plusieurs définitions ont été attribuées à ce terme selon son utilisation. Parmi ces définitions, on peut citer [20] :

- **Gartner** : c'est un style de calcul offrant à plusieurs clients à travers Internet des capacités massives et scalables relatives aux technologies d'information.
- **Forrester** : c'est une plate-forme composée d'un ensemble de ressources abstraites, hautement scalables et gérées. Cette plate-forme est capable d'accueillir des utilisateurs d'applications qui sont facturés selon leur consommation.
- **IBM** : c'est un paradigme de calcul émergent, dans lequel les données et services résident dans des centres de calcul massivement scalables qui peuvent être accédés à partir de n'importe quel périphérique connecté à Internet.

Le premier cloud qui a été rendu accessible est le EC2 (*Elastic Compute Cloud*) lancé en 2006.

2.6 Les réseaux d'interconnexion

Les constructeurs de plates-formes parallèles savent très bien que la clé pour construire des plates-formes HPC (High Performance Computing) réside non seulement dans des processeurs très puissants, mais aussi dans un réseau d'interconnexion de ces processeurs assurant un débit très élevé et ayant une latence minime. Les réseaux utilisés dans 413 machines du Top500 (liste de novembre 2012) [9] sont le Gigabit Ethernet et l'Infiniband, le reste des machines utilise des réseaux propriétaires ou des réseaux Myrinet.

2.6.1 Gigabit Ethernet

C'est une extension de la norme Ethernet (c'est-à-dire la norme IEEE 802.3) permettant la transmission de trames Ethernet à un débit de 1 Gbps, en utilisant une extension du protocole CSMA/CD et de ses prédécesseurs (Ethernet et Fast Ethernet). Dans sa configuration la plus courante, Gigabit Ethernet utilise le réseau duplex intégral et commuté. En plus de câbles en cuivre, ce type de réseaux permet l'utilisation de la fibre optique.
Dans un sens plus large, plusieurs types de réseaux à taux de transmission plus élevées sont également considérés comme faisant partie de la famille Gigabit Ethernet , comme la norme 10 Gigabit Ethernet, avec une vitesse nominale de 10 Gbps, basé principalement sur l'utilisation de la fibre optique et des connexions full-duplex.

9. http://top500.org/statistics/list/

2.6.2 Myrinet

Myrinet est un standard national américain ANSI/VITA 26-1998, crée par la société Myricom qui construit des commutateurs et des cartes réseaux haut débit. Myrinet est une technologie pour la communication haute performance en HIPPI (*High Performance Parallel Interface*) qui est une spécification de réseau local à haut débit. C'est une technologie qui s'est largement répandue pour relier ensemble des clusters de postes de travail, des PCs, des serveurs ou des ordinateurs.

2.6.3 Infiniband

InfiniBand^{tm} est une spécification qui définit une architecture de communication spécifiquement conçue pour être évolutive et à une faible latence, une bande passante élevée, une tolérance aux pannes et une qualité de Service (*QoS*). Ses spécifications sont développées et maintenues par InfiniBand^{tm} Trade Association, formée par les grandes entreprises dans le domaine de la technologie de l'information, dirigée par IBM, Intel, Mellanox, Oracle et d'autres entreprises. Cette technologie connaît un grand essor ces dernières années chez les constructeurs de plates-formes HPC grâce à ses performances et son bon ratio coût/performance en la comparant avec Myrinet et Gigabit Ethernet.

3 Programmation parallèle

3.1 Programmation MPI

3.1.1 Historique

MPI [83] n'est pas une bibliothèque de communication développée par une université ou une entreprise. C'est un standard qui est né au début des années 90 pour répondre à un besoin de clarification. En effet, à cette époque, la seule façon de faire du calcul parallèle efficace consistait à acheter une machine parallèle propriétaire. Ces dernières étaient généralement livrées avec leur propre bibliothèque de communication qui était rarement compatible avec celle de la machine précédente et jamais avec celle des concurrents. Il était donc très difficile de maintenir un programme et un gros travail était nécessaire à chaque fois que l'on souhaitait changer de machine. Le standard MPI est donc né de la collaboration entre des universitaires et des industriels de divers domaines scientifiques pour résoudre ce genre de problèmes. Les implémentations de MPI sont nombreuses. Parmi les implémentations libres, on peut citer MPICH [10], LAM/MPI [11] et Open MPI [12].

10. http://www.mcs.anl.gov/research/projects/mpich2/
11. http://www.lam-mpi.org/
12. http://www.open-mpi.org/

3.1.2 Modèle SPMD

Un programme MPI consiste en plusieurs processus, chacun s'exécutant sur un processeur. Chaque processus dispose de son propre espace mémoire. Tous les processus exécutent le même programme. Une copie de ce programme est enregistrée dans l'espace mémoire de chaque processus. Les données du programme sont subdivisées en pièces ; une pièce différente réside dans l'espace mémoire de chaque processus. Ce dernier effectue sa portion de calcul et sauvegarde les résultats dans sa mémoire locale. Si un processus a besoin d'une pièce qui réside dans l'espace mémoire d'un autre processus, alors le processus qui dispose de cette pièce doit envoyer un message contenant cette pièce au processus qui en a besoin. Cette approche de programmation (ainsi décrite) est l'approche SPMD (Single Program Multiple Data). Le nombre de processus d'un programme MPI est fixé souvent au début de l'exécution de ce programme. La commande évoquée pour initialiser ce nombre est l'une de deux commandes suivantes :

```
mpiexec -n 7 nom_programme arguments_programme
mpirun -np 7 nom_programme arguments_programme
```

Cette commande commence l'exécution du programme *nom_programme* avec un nombre de processus égale à 7.

3.1.3 Communications point à point

Une communication point à point (pt/pt) doit être évoquée par deux tâches(processus) MPI. Une de ces tâches est la tâche émettrice et l'autre réceptrice. Cette communication peut être effectuée selon l'un de deux modes suivants :

- Mode bloquant : l'appel de la primitive d'envoi ne se termine que lorsque le message a effectivement quitté le processeur et que le récepteur l'a bien reçu. L'indicateur de fin de communication pour le processus destination est l'arrivé du message et sa recopie dans une mémoire tampon locale.
- Mode non bloquant : l'appel d'une primitive d'envoi ou de réception consiste à demander d'expédier ou de recevoir un message dès que possible et de continuer l'exécution du processus aussitôt.

Les paramètres de ces routines sont détaillées dans l'annexe A.

3.1.4 Communications collectives

Les communications se font sur l'ensemble ou une partie des processeurs. Les primitives se basent sur les primitives de la communication point à point. Elles fournissent des solutions pour

un certain nombre de schémas de communication les plus utilisées. Parmi les primitives les plus utilisées, on peut citer :

- La synchronisation (MPI_Barrier) : il n'y a pas d'échange d'information proprement dit, mais les processus sont assurés qu'il ont tous rallié un certain point de leur exécution.
- La diffusion (MPI_Bcast) : un processus envoie un même message à tous les autres processus.
- La distribution (MPI_Scatter) : un processus envoie un message distinct à chacun des autres processus.
- Le rassemblement (MPI_Gather) : à l'inverse de la distribution, un processus reçoit un message distinct de chacun des autres processus.
- Le commérage (MPI_Alltoall) : pour les p processeurs, chaque processus possède une information spécifique. A la fin de la primitive, chaque processeur connaît les p informations.
- La réduction (MPI_Reduce) : on effectue une opération chez l'un des processus sur les données collectées de tous les processus.

Les détails sur les routines MPI associées à ces opérations sont donnés dans l'annexe A.

3.2 Programmation OpenMP

OpenMp [24] est une interface de programmation portable qui permet de faciliter le développement d'applications parallèles pour des machines à mémoire partagée. Il permet de développer rapidement des applications parallèles à petite granularité en restant proche du code séquentiel. Ce mode de programmation définit un ensemble de directives, de routines d'exécution et de variables d'environnement pour exprimer le parallélisme dans le code. Ces directives sont supportées par les langages Fortran, C et C++ et ont pour objectif de changer le flux de contrôle selon le principe du fork & join (cf figure 4). Quand on rencontre dans un code une structure parallèle suivie d' une boucle itérative, le compilateur crée un ensemble de processus esclaves qui vont se partager les itérations de cette boucle et les exécuter en parallèle. Le nombre de processus peut être défini par l'utilisateur en utilisant la directive *omp_set_num_thread* ou en initialisant la variable d'environnement *OMP_NUM_THREAD*. Les variables dans un programme utilisant OpenMP peuvent être déclarées comme privées ou partagées entre les différents processus. Par exemple, l'indice d'une boucle à exécuter en parallèle doit être privé pour chaque processus. Le fait de déclarer une variable comme privée engendre une création de plusieurs copies de cette variable dans l'espace d'adressage de chaque processus.

L'utilisation de OpenMP ne présente un intérêt que sur des environnement composés de plusieurs processeurs (ou cœurs) partageant la même mémoire (comme c'est le cas pour les SMP "*Symmetric Multi Processor*"). Malgré son intérêt dans le cas des machines à mémoires partagées, ce type de programmation ne peut pas être utilisé seul dans le cas des machines à mémoires

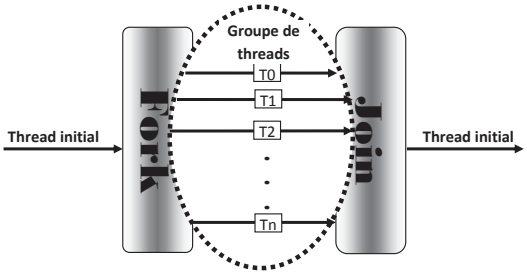


Figure 4 – Fork/join dans un programme OpenMP

distribuées (tel est le cas pour les clusters et les grilles de calcul). En effet, OpenMP ne peut être utilisé sur ces machines qu'en conjonction avec d'autres types de programmation (tels que MPI et PVM).

3.3 Programmation hybride MPI/OpenMP

Comme nous l'avons précisé dans la section précédente, les nœuds composant les nouvelles plates-formes parallèles sont généralement à base de multi-cœurs. Cette spécificité représente la cause de migration des méthodes de programmation parallèle utilisant seulement du MPI en une programmation combinant MPI avec OpenMP. La justification de cette combinaison est de prendre avantage des caractéristiques de chaque type de programmation. Ce genre de parallélisme consiste à raffiner les tâches MPI en utilisant les processus OpenMP. La figure 5 montre la différence entre la décomposition en MPI pur et la décomposition hybride(MPI/OpenMP). En effet, l'occupation de 4 processeurs (ceux du schéma) est réalisée à travers quatre processus MPI dans la première décomposition. Par contre dans la deuxième décomposition, ces processeurs ont été occupés par seulement deux processus MPI. Le gain apporté par la décomposition hybride est clair. Cette technique de décomposition augmente le degré de parallélisme même pour les applications MPI existantes en offrant le bon usage des nœuds à mémoire partagée que peut contenir la plate-forme parallèle. Cette méthode hybride peut être la plus appropriée pour les cas suivants [81] :

- Les applications peu scalables avec l'augmentation des tâches MPI.
- Les codes MPI ayant un faible équilibrage de charges (OpenMP peut faire l'affaire dans ce cas).
- Les codes avec réplication de données (OpenMP fait une seule copie par nœud).
- Les applications MPI avec une contrainte sur le nombre de tâches MPI (exemple une puissance de 2 ou un maximum de tâches).
- Les systèmes où la communication intra-nœud est non optimisée (ce qui peut être résolu par l'accès direct à la mémoire via OpenMP).

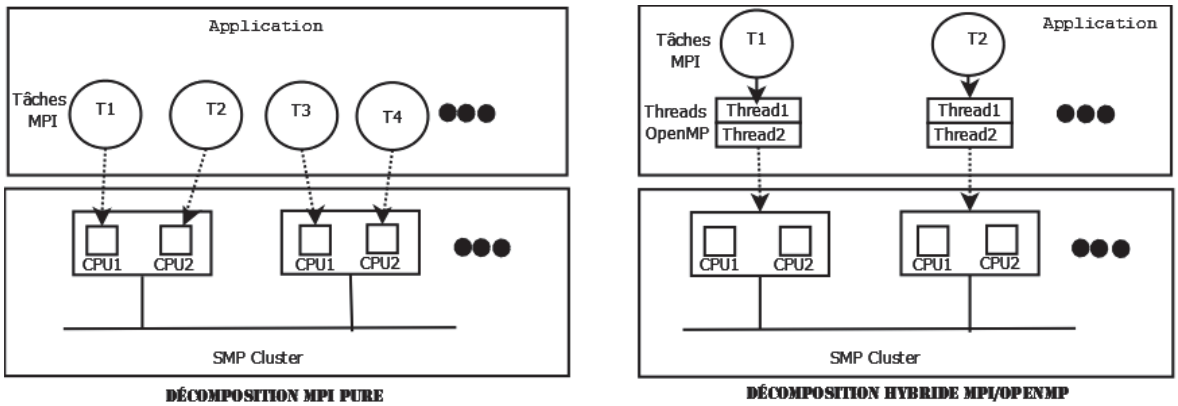


Figure 5 – MPI versus MPI/OpenMP

3.4 Programmation des GPU

Pour la programmation des processeurs graphiques, deux langages ont été proposés récemment, à savoir CUDA [70] et OpenCL [60].
CUDA (*Compute Unified Device Architecture*) est une extension du langage C proposé par NVIDIA pour l'utilisation des cartes graphiques Geforce 8 et ses successeurs. Cet outil sert comme langage à usage général pour la programmation parallèle sur les GPUs. Via ses directives, CUDA permet d'identifier certaines fonctions dans le programme C comme destinées à être exécutées sur le GPU au lieu du CPU. Ces fonctions qui sont appelées aussi *kernels* sont compilées avec le compilateur CUDA pour être ensuite exécutées sur les nombreuses unités de traitement du GPU. CUDA emploie la notion de thread dans son modèle d'exécution. En effet, un kernel démarre des centaines de threads qui sont regroupés en une série de Blocs. Les threads d'un bloc s'exécutent sur le même SM et utilisent la même mémoire pour l'échange de données.
A la différence de CUDA, OpenCL est un standard de programmation parallèle gratuit et ouvert à tous les types de cartes graphiques (NVIDIA, AMD, ...). Ce standard est aussi une extension du langage C permettant d'écrire des kernels.

4 Bibliothèques parallèles de calcul numérique

4.1 Bibliothèques et performance

Dans les programmes parallèles, les parties les plus gourmandes en terme d'usage CPU sont généralement relatives à des opérations d'algèbre linéaire. L'optimisation de ces opérations a connu un grand intérêt par les programmeurs, ce qui a donné naissance à plusieurs bibliothèques englobant chacune certaines de ces opérations. Certaines de ces bibliothèques sont payantes et sont généralement spécifiques à des architectures bien déterminées. La bibliothèque PESSL d'IBM

[55] est un bon exemple de ces bibliothèques. Plusieurs autres bibliothèques ont été conçues pour un usage public telles que la bibliothèque ScaLAPACK et PetsC. Pour ces dernières, le défit (du point de vue performance)est énorme puisqu'elles sont conçues pour s'exécuter sur n'importe quelle architecture.
Il est évident que l'utilisation d'une bibliothèque performante lors du développement d'une application parallèle est une garantie de performance pour cette application. D'où l'importance de choix de bibliothèque lors du développement de son application. Ce choix doit être basé,normalement, sur la performance de la bibliothèque pour le contexte considéré (langage de programmation, compilateur, plate-forme,. . .).

4.2 ScaLAPACK

ScaLAPACK [36](*Scalable Linear Algebra PACKage*) , développé en commun par l'Oak Ridge National Laboratory et l'Université du Tennessee (Knoxville), est une librairie de routines d'Algèbre linéaire très performante destinée aux machines MIMD à mémoire partagée utilisant le passage de messages, et pour les réseaux de PC supportant MPI et/ou PVM (Parallel Virtual Machine) [16]. Elle constitue une continuation du projet LAPACK [8] (*Linear Algebra PACKage*) conçu d'une manière analogue pour fournir des routines d'Algèbre linéaire destinées aux PC, aux machines vectorielles et aux machines à mémoire partagée. ScaLAPACK est construite au dessus de PBLAS (*Parallel Basic Linear Algebra Subroutines*) et BLACS [38] (*Basic Linear Algebra Communication Subprograms*). (voir figure 6).

4.3 PetsC

PetsC [13] (*Portable, Extensible, Toolkit for Scientific Computation*) est une bibliothèque développée par le Laboratoire National d'Argonne (*USA*). Elle peut effectuer les opérations suivantes :

- Les opérations de PBLAS-1 et d'autres opérations du niveau 1 non couvertes par le PBLAS-1.
- Un sous-ensemble de PBLAS-2.
- Factorisation de matrices (méthode de Cholesky et LU).
- Résolution de systèmes d'équations linéaires.
- Résolution de systèmes d'équations linéaires en utilisant une méthode itérative avec préconditionnement.
- Résolution de systèmes d'équations non linéaires.
- Minimisation de fonctions sans contraintes.

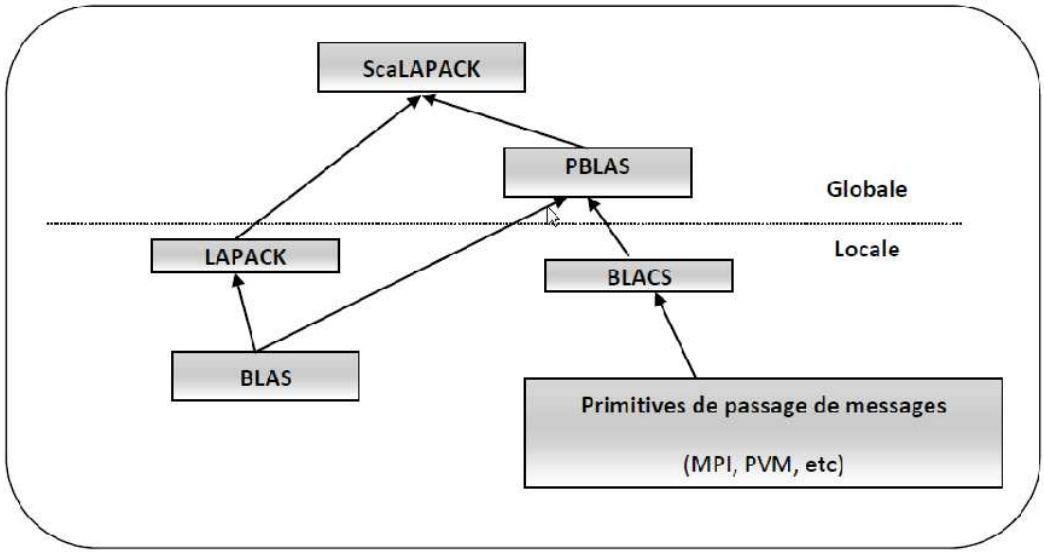


Figure 6 – Architecture de ScaLAPACK

5 Conclusion

Le domaine du calcul parallèle a connu ces dernières années une diversification importante sur les plans matériel et logiciel. Toutefois, malgré cette diversification de plates-formes, les méthodes de programmation se basent généralement sur deux modèles principaux de programmation. Le premier cible principalement les plates-formes distribuées et utilise les messages pour assurer la coopération entre les tâches de la même application. Le second modèle cible les architectures parallèles à mémoire partagée et utilise la notion de thread pour subdiviser le traitement entre les différentes unités de traitement de l'architecture. Les standards les plus utilisés pour ces deux modèles sont respectivement MPI et OpenMP.

Le challenge pour les programmeurs du domaine du calcul parallèle est dans le développement de programmes efficaces et portables pour les architectures actuellement utilisées. Il est bien connu que la performance des programmes dépend d'un très grand nombre de paramètres. En conséquence, il est quasiment impossible de générer manuellement un programme fournissant les meilleures performances pour n'importe quelle plate-forme.

Nous présentons dans le chapitre qui suit une approche favorisant le développement d'applications parallèles (resp. séquentielles) performantes, à savoir l'approche adpatative. Cette approche est basée sur l'utilisation de plusieurs paramètres architecturaux et algorithmiques dans le processus de développement, de compilation et/ou d'exécution afin d'améliorer la performance de l'application considérée.

Chapitre 2

Adaptativité et modélisation de performance pour des applications à haute performance

1 Introduction

Dans la littérature, plusieurs techniques ont été utilisées dans l'objectif de développer des applications performantes sur une multitude de plates-formes. Les deux techniques les plus connues sont l'approche poly-algorithmique et celle utilisant le génération de code. Ces techniques sont des *techniques adaptatives*, c'est-à-dire qu'elles adaptent l'exécution et/ou la compilation de l'application en fonction de valeurs de paramètres architecturaux et/ou algorithmiques afin d'améliorer sa performance sur la plate-forme considérée. Pour atteindre leurs objectifs, ces techniques adaptatives se basent sur des modèles combinant les différents paramètres architecturaux et/ou algorithmiques pour effectuer les différents choix adaptatifs. Pour aboutir à ces modèles, plusieurs techniques de **modélisation de performance** peuvent être utilisées.
Dans ce chapitre, nous abordons en premier lieu (section 2) les principaux concepts qui tournent autour de la performance des applications sur les architectures séquentielles et parallèles. La section suivante est consacrée à l'étude des différentes techniques adoptées par les applications adaptatives. Une étude bibliographique sur plusieurs applications et frameworks adaptatifs est donnée à la fin de ce chapitre (section 4).

2 Performance des applications informatiques

2.1 Facteurs affectant la performance

Les facteurs affectant la performance d'un programme (notamment un programme parallèle) sur une plate-forme donnée sont nombreux. Ces facteurs peuvent être répartis en trois grandes catégories :

(a) Des paramètres architecturaux : ce sont des paramètres reliés à des composantes matérielles pouvant agir sur l'accélération ou le retard d'une application. Parmi ces composantes on trouve :
   - Les processeurs : plusieurs paramètres peuvent être issus de ce composant à savoir le nombre de processeurs (pour le cas d'une plate-forme parallèle), leurs architectures(scalaire, vectoriel), les nombres de cœurs par processeur, leurs fréquences d'horloge, ...
   - La hiérarchie mémoire : la taille des caches processeur et de la mémoire vive ainsi que leurs vitesses d'accès sont des paramètres très importants pour la performance d'une application. La vitesse d'accès aux supports de stockage est aussi un paramètre de performance des applications qui s'exécutent en out-of-core.
   - Le réseau de communication : le délai engendré par une communication inter-processeurs dans une architecture parallèle peut être une source de dégradation de performance. Les paramètres les plus intéressants dans ce contexte sont le débit du réseau, la latence, et la largeur de bissection (cf. Section 2.2.3).

(b) Des paramètres algorithmiques : la performance d'une application peut dépendre de plusieurs paramètres relatifs à l'instance du problème. On peut en citer : la taille du problème lui-même, le type des données (simple ou double précision), la nature des données, ...

(c) Des paramètres relatifs à l'environnement de développement : les plus importants de ces paramètres sont le compilateur et le système d'exploitation.

En principe, pour prédire correctement le comportement d'un programme donné sur une plate-forme donnée, l'ensemble de ces paramètres doit être mis en considération. Mais, pour des raisons de simplification et de généralisation, certains paramètres sont omis par les modèles de prédiction.

2.2 Evaluation de performance

La performance d'une plate-forme d'exécution qu'elle soit séquentielle ou parallèle est l'un des plus importants aspects à évaluer. Pour effectuer une évaluation de performance, différents critères sont à considérer selon le point de vue où l'on se place. En effet, l'utilisateur final est intéressé surtout par le temps de réponse (nommé aussi temps d'exécution) de son programme qui peut être défini par le temps écoulé entre le début et la fin d'exécution du programme. D'autre part, dans un centre de calcul traitant plusieurs tâches de divers utilisateurs, l'intérêt est donné principalement au débit d'exécution des tâches qui peut être défini par la moyenne des tâches unitaires exécutées par unité de temps.

2.2.1 Performance des processeurs

La performance théorique d'un système est généralement quantifiée en FLOPS (*FLoating Point Operations Per Second*). Le calcul de la performance d'une machine multi-cœurs, notée R_{peak} (*peak rate*), peut être déterminé selon l'équation 2.1 :

$$R_{peak} = N_{coeurs} * N_{fpu} * f \tag{2.1}$$

Où N_{coeurs} est le nombre de cœurs de la machine, N_{fpu} le nombre d'unités d'opérations entre flottants par cœur et f la fréquence d'horloge de la machine.
Cette performance théorique ne donne pas la performance réelle d'une machine, elle ne reste que théorique. La performance utilisée pour comparer différentes machines doit être basée sur des tests réels effectués sur ces machines (comme c'est le cas pour effectuer le classement des super-calculateurs dans le Top500). Généralement, pour aboutir à de telles performances (réelles), on

doit se baser sur l'exécution de routines de bibliothèques de référence (*benchmark*) tels que les BLAS(*Basic Linear Algebra Subroutines*). Cette performance est calculée selon l'équation 2.2.

$$R = N_{flop}/temps_d'exécution \tag{2.2}$$

Où N_{flop} est le nombre d'opérations entre flottants dans la routine exécutée ($N_{flop} = 2n^3$ pour la routine *dgemm* du BLAS pour la multiplication de deux matrices carrées d'ordre n).
L'efficacité d'une plate-forme peut être calculée comme suit :

$$E = R/R_{peak} \tag{2.3}$$

2.2.2 Mesure du temps d'exécution

Pour mesurer le temps d'exécution d'un programme donné, plusieurs outils sont à la disposition du programmeur. Le plus connu parmi ces outils est la commande *time* offerte par les systèmes Unix (et Linux). Cette commande est capable de mesurer trois temps différents qui sont les suivants :

- Temps au mur (*wall-clock time*) : c'est le temps mis entre le début et la fin d'un programme.
- Temps utilisateur : temps CPU utilisé par le programme utilisateur
- Temps système : temps utilisé par le système d'exploitation pour faire des allocations mémoires ou disques.

```
sachour@access:~$ time ./prog 1000

real    0m13.442s
user    0m13.413s
sys     0m0.024s
```

Figure 7 – Temps mesurés en utilisant la commande *time*

Cette commande *time* ne peut mesurer que le temps global mis pour l'exécution d'un programme. Pour l'analyse de performance, il est requis d'avoir une idée sur le temps mis par certaines parties du programme. Il existe plusieurs méthodes de mesure de temps dans un programme donné. Ces outils sont fournis par les langages de programmation et par quelques bibliothèques. Les bibliothèques MPI et OpenMP disposent chacune d'elles d'une fonction de mesure de temps qui est indépendante de la plate-forme d'exécution. Ces fonctions sont *MPI_Wtime()* pour MPI et *omp_get_wtime()* pour OpenMP. Ces deux commandes retournent un temps mesuré en secondes. La différence entre les résultats de deux appels à l'une de ces fonctions correspond au temps mis par la partie du programme entre les deux appels.

2.2.3 Performance des réseaux

La performance d'un réseau d'interconnexion d'une plate-forme parallèle peut être résumée par les caractéristiques suivantes :

- La latence : qui fait référence au temps nécessaire avant de commencer l'envoi du message. Indépendamment de la taille du message à envoyer. Elle dépend du matériel et des protocoles logiciels utilisés dans le réseau.
- La bande passante : elle est mesurée en bit par seconde, elle donne une idée sur le taux de transfert des messages.
- La bande passante de bissection : elle est mesurée en bit par seconde et se réfère au taux total de transfert de données si la moitié des nœuds dans le cluster est en train d'envoyer des messages à l'autre moitié.

2.2.4 Benchmarking

La performance d'un système dépend significativement du programme à exécuter. En effet, pour deux programmes A et B et deux systèmes X et Y, la situation suivante peut se produire : le programme A a un temps d'exécution sur le système X plus court que celui sur le système Y, par contre le temps d'exécution du programme B sur le système Y est plus petit que celui sur le système X. En d'autres termes, la plate-forme X (resp. Y) est plus performante pour le programme A (resp. B). D'où l'importance du choix de l'application pour avoir la mesure de performance la plus représentative.
Pour l'utilisateur, il est important de baser le choix de la machine d'exécution sur un ensemble de programmes fréquemment exécutés. Vu que les programmes à exécuter sont généralement non connus au préalable, plusieurs applications standardisées ont été développées pour servir comme programmes de référence (benchmark programs) pour l'évaluation des performances des plates-formes d'exécution. Chacun de ces programmes de référence est basé sur une caractéristique qui va être mesurée sur la machine cible. Ces programmes peuvent être répertoriés en trois grandes catégories [78] :

1. Programmes de référence synthétiques : qui sont typiquement des petits programmes contenant un mixage d'instructions sélectionnées de manière à représenter une large classe d'applications réelles. Généralement, ces programmes n'opèrent pas sur un nombre important de données, ce qui risque d'avoir des résultats loin d'être réalistes vu que les interactions entre calcul et accès mémoire ne sont pas prises en considération. Parmi les programmes de cette classe on trouve le Whetstone [32] et le Drystone [87]. Les métriques de performances mesurées par ces programmes sont respectivement le KWhetstone/s et le KDrystone/s.

2. Noyaux de référence : ce sont des parties pertinentes dans les applications réelles et qui capturent généralement une large portion du temps d'exécution de ces dernières. Parmi les collections de noyaux, on peut citer le Livermore [66] qui consiste en 24 nids de boucles extraits des codes de simulation, et le Linpack [35] qui comporte le code de plusieurs opérations d'algèbre linéaire. Le résultat de ces deux collections est la performance en terme de MFLOPS.
3. Applications réelles de référence : appelées aussi suites de références. Parmi ces suites, on a le NAS Parallel benchmarks [12] qui est une suite développée par la NASA et qui comporte cinq noyaux parallèles et trois applications de simulation. Cette suite est souvent utilisée pour comparer les performances des grappes de calcul. L'avantage de cette suite est qu'elle propose des algorithmes avec des schémas de communication variés, permettant de tester différents comportements. Néanmoins, ces applications n'ont pas été conçues pour des grappes de calcul de très grandes tailles, limitant alors leurs utilisations.
   Une autre suite développée par SPEC (*Standard Performance Evaluation Corporation*) est aussi utilisée dans le domaine d'evaluation de performance des plates-formes parallèles . Actuellement, SPEC n'a pas développé une seule suite mais plusieurs, chacune d'entre elles étant dédiée à l'évaluation de performance pour une architecture ou une caractéristique bien déterminée. Parmi les suites offertes par SPEC on trouve SPEC CPU2006 pour l'évaluation de performance des capacités de calcul des différentes générations de processeurs et SPEC MPI2007 [68] pour l'évaluation de performance pour des clusters de machines.

2.3 Prédiction de performance

2.3.1 Intérêts

La prédiction de performance est la génération d'une estimation du temps d'exécution d'un programme sur une machine donnée [71]. Cette tâche est d'un intérêt certain pour un grand nombre de personnes dans l'industrie du matériel et du logiciel. En effet, trois grandes catégories de personnes ont intérêt à utiliser la prédiction de performance :

- Les constructeurs de machines : les outils de prédiction demandés par ces personnes doivent être détaillés, flexibles et précis vu qu'ils vont être utilisés pour la prédiction de performance de futures machines.
- Les développeurs de compilateurs : le développement de compilateurs est réalisé souvent avant la construction du processeur. Les outils de prédiction de performance doivent donner à ces développeurs la possibilité de déterminer à la fois le bon fonctionnement de leurs compilateurs et la performance des codes compilés avec ces compilateurs.

– Les développeurs d'applications : optimiser une partie du code est devenu une tâche très difficile vu que les processeurs et les environnements actuellement utilisés sont très complexes. Les outils de prédiction de performance donnent la possibilité aux programmeurs de voir les parties gourmandes en terme d'utilisation de la CPU ou de réseau ou encore la détermination des zones d'attente ou de blockages. De ce fait, la tâche d'optimisation du code devient moins complexe.

Il existe principalement trois types de modélisation, à savoir la modélisation analytique, la modélisation expérimentale et la simulation.

2.3.2 Modélisation analytique

Cette technique de modélisation se base sur la génération manuelle de modèles théoriques en faisant abstraction de l'application et/ou de la machine que l'on veut modéliser. En fait, c'est la méthode la plus classique de prédiction de performance, qui nécessite une grande compréhension des caractéristiques de l'application et de l'architecture à modéliser pour développer un tel modèle. Parmi les avantages de cette méthode c'est qu'elle permet d'étudier le comportement des algorithmes avec un temps de réponse réduit. Néanmoins, son inconvénient majeur est l'effort humain investi pour construire de tels modèles et la précision des performances estimées.

2.3.3 Modélisation expérimentale

La modélisation expérimentale évoque généralement des techniques d'analyse des données collectées comme les modèles statistiques ou probabilistes, les techniques de data-mining et les techniques de l'intelligence artificielle, telles que les réseaux de neurones, les chaînes de Markov, etc. Pour pouvoir prédire les performances avec une bonne précision, cette méthode devrait avoir un entrepôt assez important de données relatives aux différentes exécutions. L'avantage majeur de cette technique est qu'elle ne s'intéresse pas aux détails de l'application. Néanmoins, elle est plus couteuse que les méthodes analytiques au sens temps de réponse.

2.3.4 Modélisation par simulation

Cette technique est largement utilisée dans les environnements industriels. Sa principale caractéristique est qu'elle peut produire des prédictions précises de la performance de l'application. Néanmoins, elle présente plusieurs inconvénients e.g. son paramètre de ralentissement (*slowdown*) qui est généralement élevé. Ainsi, un simulateur ayant un *slowdown* de 100 veut dire que la simulation de l'exécution d'une application peut durer cent fois plus que son exécution réelle. De plus, la majorité des outils de simulation est utilisée dans les laboratoires industriels.

	Modélisation Analytique	Modélisation Expérimentale	Simulation
Dépendance de l'architecture	Faible	Élevée	Faible
Dépendance de l'instance de problème	Faible	Élevée	Faible
Coût de développement	Faible	Faible	Élevé
Précision des résultats	Faible	Élevée	Élevée
Coût d'utilisation (temps)	Faible	Élevé	Élevé

Tableau 2 – Comparatif des différentes approches de modélisation de performance

2.3.5 Etude comparative

Pour la comparaison des différentes approches de modélisation de performance, plusieurs facteurs sont à considérer, à savoir, le temps de développement, le coût d'utilisation, la généricité et la précision. Le tableau 2 récapitule les différences entre les trois méthodes de modélisation de performance selon ces différents critères.

3 Applications adaptatives

Pour plusieurs problèmes, divers algorithmes fonctionnellement équivalents peuvent être utilisés pour les résoudre. A titre d'exemple, on peut citer le tri (tri par insertion,tri rapide, tri fusion, etc), le calcul des arbres de recouvrement minimums (les algorithmes de Kruskal et de Prim),la multiplication matricielle (algorithme standard, algorithme de Strassen, algorithme de Winograd, algorithme de Cannon, . . .), le calcul d'enveloppes convexes (incrémentale, emballage cadeau, diviser pour régner, quickhull, . . .).
Quand il existe de nombreuses variantes algorithmiques fonctionnellement équivalentes, le choix entre ces variantes peut dépendre de nombreux facteurs, comme le type et la taille de l'entrée, la facilité d'application et de vérification, etc. Dans un contexte distribué et parallèle, la situation devient encore plus complexe en raison de la grande variété d'architectures. De plus, les performances peuvent varier considérablement pour des tailles différentes (nombre de machines) d'une architecture donnée. Ces facteurs rendent la tâche de détermination du meilleur algorithme pour un contexte déterminé très difficile, et ceci même pour des programmeurs expérimentés. Par ailleurs, et c'est peut-être plus important encore, ces facteurs font qu'il est très difficile d'écrire des programmes portables et qui fonctionnent bien sur de multiples plates-formes ou de données d'entrée de tailles et/ou de types différents.
Idéalement, le programmeur doit simplement spécifier l'opération souhaitée (par exemple le tri) et la décision de l'algorithme à utiliser doit être faite ultérieurement, peut-être même au temps d'exécution, une fois l'environnement et les caractéristiques d'entrée sont connus.

3.1 Définition

un programme est dit adaptatif s'il change de comportement en se basant sur l'état courant de son environnement [44]. La motivation pour changer de comportement est de satisfaire certains critères de performance. En d'autres termes, l'adaptativité dans les programmes informatiques représente une technique d'optimisation de performance pour le cas d'environnements dynamiques.

3.2 Classification

Dans [31], une classification des algorithmes hybrides et adaptatifs est proposée. Sachant qu'un algorithme est dit hybride s'il contient un choix de haut niveau entre au moins deux algorithmes permettant de résoudre le même problème (ces algorithmes sont appelés aussi Poly-algorithmes) [9]. Cette classification proposée a deux volets (cf. figure 8). Le premier volet concerne le nombre d'alternatives dans une solution adaptative qui peut être simple si le nombre de choix est limité (en 0(1) par rapport à la taille du problème). Il est dit baroque quand ce nombre est illimité (ce cas se présente lorsqu'il dépend des entrées comme la taille du problème). La méthode de choix statique entre les alternatives ne peut être adoptée que pour un simple nombre. Par contre, la méthode de choix dynamique peut être la retenue pour n'importe quelle situation.
Le deuxième volet de cette classification concerne la méthode de choix entre les alternatives. En effet, les auteurs ont distingué trois classes d'algorithmes (adaptatifs, mis au point et oublieux). Un algorithme est dit oublieux si son flux de contrôle ne dépend ni de valeurs particulières des entrées ni des propriétés statistiques des ressources.
Les algorithmes mis au point (*tuned*) sont des algorithmes qui comportent une décision stratégique basée sur des paramètres architecturaux statiques. Cette méthode de choix peut être basée sur une conception particulière lorsqu'on combine des connaissances avec une analyse de la plateforme et des entrées. Elle peut être aussi totalement automatisée si le choix est effectué via un algorithme spécifique.
Un algorithme est dit adaptatif s'il évite l'utilisation des paramètres statiques de la machine. Les décisions dans cette classe sont basées sur la disponibilité des ressources et des données d'entrée qui sont découvertes toutes les deux au moment de l'exécution. Dans cette classe, un algorithme est dit introspectif s'il utilise une prédiction des performances des algorithmes pour les données en entrée afin d'en choisir parmi ces derniers.

Dans ce travail, nous ne faisons pas de différenciation entre la classe des algorithmes adaptatifs et celle des algorithmes mis au point. Les deux seront appelés algorithmes adaptatifs. En fait, c'est seulement le moment d'adaptation qui diffère. En effet, pour certains algorithmes l'adaptation s'effectue au cours de l'exécution et pour d'autres elle s'effectue juste avant l'exécution.

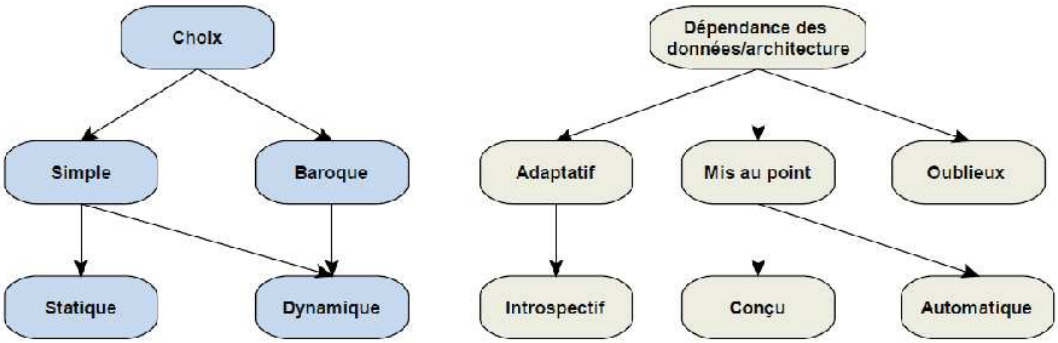


Figure 8 – Classes d'adaptativité

3.3 Techniques adaptatives

Dans la littérature, il existe différentes techniques adaptatives [69]. Les principales sont :

- La technique poly-algorithmique,
- La génération de code,
- Le paramétrage de code,
- Les techniques d'ordonnancement

3.3.1 La technique poly-algorithmique

Plusieurs opérations importantes, telles que le tri et la multiplication matricielle, peuvent être résolues avec une multitude d'algorithmes de fonctionnalités équivalentes. Une solution à ce problème consiste à disposer d'un ensemble d'algorithmes ayant les mêmes fonctionnalités mais de particularités différentes afin de choisir l'un ou les algorithmes les plus performants pour le système cible et les données présentes. Cette approche est appelée poly-algorithmique. Néanmoins, la décision de choix, basée généralement sur des modèles analytiques ou expérimentaux, dépend de plusieurs facteurs dont le type et la taille des données d'entrée et les caractéristiques de l'architecture cible. Les poly-algorithmes disposent donc de mécanismes de décision, qui leur permettent de choisir un ou plusieurs algorithmes suite à leurs propres analyses et en fonction de leurs performances sur le système cible. Deux types de choix peuvent donc être faits par les poly-algorithmes :

- **Sélection d'algorithmes** : le problème consiste à sélectionner l'algorithme le plus performant parmi un ensemble d'algorithmes candidats suivant des critères de choix bien déterminés pour résoudre une instance d'un problème donné sur une architecture particulière.
- **Combinaison d'algorithmes** : l'idée sous jacente de cette technique est d'utiliser, dans le cas classique, plusieurs algorithmes pour résoudre un même problème et ce, afin d'optimiser plusieurs critères de performances où chacun est assuré par un algorithme différent ou en

fonction des données d'entrée, puisque la performance d'un algorithme dépend des données qu'il manipule. La combinaison adaptative d'algorithmes consiste à déterminer et adapter le schéma d'exécution d'algorithmes combinant plusieurs algorithmes pour résoudre l'instance du problème donné en fonction de son contexte d'exécution.

3.3.2 Paramétrage de code

Cette technique est associée généralement à des techniques expérimentales permettant de faire varier les différents paramètres liés au problème et à la machine cibles dans le but de déterminer ceux qui maximisent les performances. L'exemple qui peut être cité dans ce contexte est celui de la taille du bloc de matrice qui représente un paramètre de compilation pour les bibliothèques d'algèbre linéaire telle que ATLAS [9].

3.3.3 Génération de code

Pour la prise en considération de plusieurs paramètres architecturaux dont la modification n'est possible qu'à travers le code source, le paramétrage de code ne pourra pas être envisagé. Parmi ces paramètres on peut citer : la taille du cache d'instructions, la profondeur du pipeline, etc. Ce problème peut être résolu à l'aide de multiples implémentations, où chacune est conçue pour des valeurs particulières de ces paramètres ou pour des types de machines particulières (scalaires, vectorielles, etc.). En effet, le concepteur d'une application adaptative portable et efficace doit développer plusieurs versions de code pour différentes générations de machines. La génération de telles implémentations peut être effectuée manuellement ou d'une façon automatisée. Cependant, vu la diversité des architectures existantes, la génération manuelle ne représente pas la bonne solution. C'est pour cela que plusieurs applications ont choisi de créer des générateurs de code permettant de recouvrir le maximum possible de contextes de travail. Les exemples qui ont adopté cette technique sont multiples, tels que PHiPAC [18], ATLAS [9], FFTW [42], etc.

3.3.4 Techniques de vol de tâches

L'objectif de ces techniques est d'auto-équilibrer la charge de travail des différents processeurs composant une machine parallèle donnée (machine de référence). Ces techniques supportent des allocations de tâches qui dépendent de la disponibilité des processeurs. En effet, à chaque processeur est associée une file de tâches en attente. Dès qu'un processeur termine l'exécution de toutes les tâches qui lui sont allouées, sa file devient vide et il désigne aléatoirement un processeur, appelé processeur victime, et effectue une requête de vol de tâches vers lui (si la file de la victime n'est pas vide)[19].

3.4 Caractérisation de l'adaptativité

Les approches adaptatives peuvent être différenciées par plusieurs critères. Les plus importants sont le niveau et le moment de l'adaptativité.

3.4.1 Niveau de l'adaptativité

L'adaptativité peut être effectuée dans trois niveaux différents :

(i) **Niveau algorithmique :** ici, l'adaptation se fait par rapport aux algorithmes qui peuvent être utilisés pour résoudre un problème donné. Il s'agit donc de choisir le meilleur algorithme ou éventuellement la meilleure combinaison d'algorithmes.
Pour illustrer ce niveau d'adaptativité, prenons à titre d'exemple le problème du produit matriciel. On peut proposer une combinaison de plusieurs algorithmes (standard et rapides).

(ii) **Niveau implémentation :** ce niveau correspond aux techniques de paramétrage de code présenté ci-dessus. ATLAS [9] est une bibliothèque numérique appliquant les approches adaptatives au niveau de l'implémentation. En effet, elle propose différentes implémentations pour une même routine d'algèbre linéaire et choisit au niveau de son installation l'implémentation qui offre la meilleure performance sur la machine cible.

(iii) **Niveau matériel :** ce niveau d'adaptativité se traduit par une adaptation lors de changements des ressources disponibles pour l'exécution d'une application. AppLES [5] adopte ce niveau d'adaptation. Cette méthodologie utilise des techniques d'ordonnancement permettant d'adapter l'application en cours d'exécution aux changements dynamiques du support d'exécution.

3.4.2 Moment de l'adaptativité

On peut distinguer essentiellement deux moments d'adaptativité, c'est-à-dire quand est ce qu'on peut l'appliquer : avant l'exécution (statique) ou durant l'exécution (dynamique) :

- Avant l'exécution : le choix de l'algorithme ou de l'implémentation est connu à l'avance. Ce choix est généralement basé sur des modèles de performance analytiques ou expérimentaux. Il est fait au moment de la compilation ou durant l'installation, comme c'est le cas pour ATLAS et FFTW.
- Juste avant l'exécution : le choix de l'algorithme adéquat est effectué juste après avoir déterminé des paramètres algorithmiques et architecturaux nécessaires pour décrire l'instance du problème et de la plate-forme choisie pour l'exécution. Ce processus de choix utilise ces paramètres pour déterminer l'algorithme qui va traiter le problème considéré.

– Durant l'exécution : ici, le choix peut être influencé par des changements dynamiques de la plate-forme d'exécution. En effet, l'ajout ou le retrait d'un nœud de calcul peut entraîner un réordonnancement des tâches dans le but de garder de bonnes performances pour l'application. Ce type d'adaptativité suppose l'intégration d'outils d'ordonnancement dans l'application.

3.4.3 Evolutivité de l'adaptation

Lorsqu'une application garde trace de toutes ses exécutions antérieures dans le but de garantir les meilleurs choix pour de futures exécutions, on parle d'adaptativité évolutive. L'une des techniques pouvant être utilisée dans ce contexte pour analyser les données relatives aux exécutions est celle appliquant le principe "Utiliser le passé pour prédire le futur".

4 Etat de l'art

L'utilisation des approches adaptatives pour résoudre des problèmes est devenue de plus en plus répandue. Nous présentons dans cette section quelques environnements adaptatifs, incluant des bibliothèques numériques, des méthodologies ou des algorithmes. Nous classons ces environnements selon les architectures pour lesquelles elles sont conçus (architectures séquentielles ou parallèles).

4.1 Environnement séquentiel

4.1.1 ATLAS

ATLAS (*Automatically Tuned Linear Algebra Software*)[9] est une bibliothèque adaptative d'algèbre linéaire se basant sur des techniques expérimentales pour choisir les implémentations adéquates de ses routines. En effet, ATLAS profite d'une phase d'installation qui peut durer longtemps pour tester les différentes implémentations de ses routines afin de générer le code qui offre la meilleure performance pour la plate-forme cible. Durant ces tests, ATLAS fait varier les paramètres algorithmiques et architecturaux qui peuvent affecter la performance de l'application et, en se basant sur des modèles statistiques, elle retient les paramètres qui offrent la performance maximale. Après la phase d'installation d'ATLAS, la décision au moment de l'exécution devient de plus en plus simple et elle ne se base que sur la taille du problème (tailles des matrices à traiter).

4.1.2 PHiPAC

PHiPAC (P*ortable High-Performance, ANSI C*) [18] est une méthodologie pour le développement des noyaux performants et portables d'algèbre linéaire en langage C. Pour atteindre ses objectifs (portabilité et performance), PHiPAC se fonde sur trois principes :

- Une allocation raisonnable des registres offerts par la plate-forme d'exécution,
- Un bon choix des instructions C qui peut améliorer la performance de l'application produite en utilisant PHiPAC,
- L'ordonnancement de ces instructions.

PHiPAC offre un générateur de code paramétré permettant de produire des codes optimisés sur une multitude de plates-formes séquentielles et vectorielles. Elle offre également des scripts pour déterminer les paramètres architecturaux comme le nombre des registres et les tailles des différents niveaux de cache. En fonction de ces paramètres, le générateur de code peut être appelé plusieurs fois. A chaque fois, le code est compilé selon des paramètres spécifiques (comme les tailles des blocs lors d'une multiplication matricielle MM) et des options de compilation spécifiques. L'objet produit à chaque fois est lié puis exécuté et le temps d'exécution est mesuré. Ce traitement est effectué afin de retenir les meilleures valeurs architecturales optimisant le code de l'application. L'exemple qui a montré l'efficacité de PHiPAC est celui de la MM. En effet, l'implémentation du produit matriciel avec PHiPAC a donné des performances équivalentes à celles offertes par les constructeurs des machines. PHiPAC a été le point de départ de plusieurs efforts pour appliquer cette stratégie sur d'autres noyaux d'algèbre linéaire.

4.1.3 FFTW

FFTW (*Fast Fourier Transform in the West*) [42] est une bibliothèque de calcul rapide des transformées de Fourier discrètes écrite en C. FFTW utilise un générateur de code appelé générateur de codelets. Un codelet est un bout de code pour le calcul d'une partie de cette transformée. Dans sa version standard, FFTW contient 150 codelets qui couvrent la majorité des cas couramment utilisés. La combinaison de ces codelets est appelée plan d'exécution. Le plan est déterminé au moment de l'exécution (juste avant le calcul de la transformée) par un planificateur qui utilise la programmation dynamique pour trouver le plan minimisant le temps d'exécution global. Ce planificateur mesure les performances de plusieurs plans pour déterminer le plus rapide. Les plans retenus seront stockés sur disque pour de futurs appels. Le générateur de codelets utilise une variété d'algorithmes de calcul. Parmi ces algorithmes, on peut citer :

- L'algorithme de Cooley et Tukey qui factorise le calcul de la transformée d'ordre n en n1 et n2, avec $n = n1 * n2$ et $n1 \neq 1$ et $n2 \neq$n1
- L'algorithme de Split-Radix pour des ordres multiples de 4
- L'algorithme de Radar utilisable pour les ordres premiers

– L'application directe de la définition de la transformée de Fourier

FFTW3 [43] représente l'extension de FFTW utilisée sur des architectures de type SIMD. Cette version offre également la possibilité de s'exécuter sur des machines à mémoire partagée puisqu'elle dispose d'une implémentation à base de threads.

4.2 Environnement parallèle

4.2.1 STAPL

STAPL (*Standard Template Adaptive Parallel Library*) [84] [79] est une bibliothèque parallèle à base de C++, utilisée dans plusieurs applications scientifiques. Elle permet de résoudre de nombreux problèmes, dont le tri, la recherche dans des listes, des problèmes liés aux graphes, etc. Dans cette bibliothèque, la décision entre les différents choix algorithmiques disponibles se base sur :

– Des modèles de performances, analytiques et expérimentaux,
– Des statistiques,
– Le présent état d'exécution.

Les modèles analytiques se basent sur le modèle BSP [86] et les expérimentaux utilisent les techniques d'apprentissage pour analyser les résultats d'exécutions antérieures.

4.2.2 LFC

LFC (*Lapack For Cluster*) [27] est une bibliothèque d'algèbre linéaire destinée pour des clusters. Elle est basée sur les deux bibliothèques : LAPACK s'il s'agit d'exécuter une routine en séquentielle et ScaLAPACK s'il s'agit de son exécution en parallèle. L'adaptation dans LFC se fait sur deux niveaux : le niveau d'implémentation et le niveau matériel. En effet, LFC utilise comme ATLAS des techniques de génération de code. Elle utilise également des modèles de performances et des modèles statistiques pour déterminer l'ordonnancement des ressources qui convient le mieux pour une routine. Pour effectuer un tel ordonnancement, LFC se base sur des données sur le cluster qui sont continuellement collectées par un service comparable à NWS [89].

4.2.3 AppLeS

AppLeS (*Application Level Scheduling*) [17] est une méthodologie d'ordonnancement et de déploiement d'applications sur grille. Elle est adoptée dans plusieurs applications parallèles de différents domaines. En se basant sur le service NWS, AppLeS collecte continuellement les informations sur la grille pour restituer un nouvel ordonnancement de tâches capable d'améliorer la performance globale de l'application. Pour effectuer ses choix, AppLeS se base sur des modèles analytiques en fonction de l'application.

4.2.4 Framework de Hong et Prasanna

Hong et Prasanna ont proposé une méthodologie adaptative pour implémenter le produit matriciel sur des environnements hétérogènes et dynamiques [53]. Le schéma de l'ordonnancement développé prend en considération des changements au cours de l'exécution des caractéristiques de la machine. L'idée principale est de réserver un nœud appelé coordinateur pour contrôler l'exécution. En effet, lorsqu'un nœud termine sa tâche, il renvoie son statut vers le nœud coordinateur qui rétablit l'ordonnancement en fonction de l'état présent des disponibilités des ressources de calcul et de communication.

4.2.5 Framework de Cuenca et al.

Cuenca et al. [29] ont développé une méthodologie destinée aux noyaux d'algèbre linéaire dans des environnements séquentiels et parallèles. Leur solution repose sur trois phases successives :

- Phase de conception : elle revient à modéliser analytiquement la routine d'algèbre linéaire à implémenter. Le modèle résultant sera la base de tout choix ultérieur.
- Phase d'installation : elle est réservée à la détermination des principaux paramètres systèmes.
- Phase d'exécution : le traitement au niveau de cette phase commence par appeler NWS[89] pour déterminer les paramètres dynamiques (fractions de processeurs libres, fraction de bande passante libre) puis sélectionner les paramètres algorithmiques nécessaires à l'exécution de la routine appelée.

4.2.6 Poly-Librairies

Alberti et al. [5] ont proposé une approche adaptative comparable à l'approche poly-algorithmique. Les alternatives dans cette nouvelle approche sont des librairies au lieu d'algorithmes dans l'approche poly-algorithmique. Les auteurs ont montré que la multitude des bibliothèques d'algèbre linéaire existantes (publiques et privées) peut être utilisée comme étant une méta-bibliothèque efficace sur tout type de support. En effet, en utilisant une telle stratégie, l'appel à une routine d'algèbre linéaire revient à choisir parmi les différentes bibliothèques disponibles l'instance de la routine qui offrira la performance maximale pour le contexte de travail considéré. L'idée proposée pour la mise en œuvre de cette stratégie consiste à créer un fichier d'installation pour chaque librairie permettant de décrire les performances de ses routines sur la machine d'installation pour plusieurs tailles de problèmes. Ces fichiers seront par la suite la base de tout choix dans la poly-librairie.

4.3 Récapitulatif

Le tableau 3 récapitule les principales caractéristiques des applications adaptatives décrites dans cette section. Ainsi, on dégage dans ce tableau pour chacune des applications le domaine d'application, les plates-formes cibles, les techniques adaptatives utilisées et la méthode avec laquelle la modélisation de performance est effectuée.

		ATLAS	PHiPAC	FFTW	FFTW3	STAPL	LFC	AppLeS	Hong et Prasanna	Cuenca et al.
	Domaine d'application	Algèbre linéaire	Algèbre linéaire	Transformée de Fourrier	Transformée de Fourrier	Général	Algèbre linéaire	Général	Multiplication matricielle	Algèbre linéaire
	Plate-formes	Séquentielles	Séquentielles	Séquentielles	Machines SMP	Générales	Clusters	Grilles	Grilles	Clusters
	Techniques Adaptatives	Génération de code	Génération de code	Génération de code	Génération de code	Poly-algorithmique	Génération de code	Ordonnancement	Ordonnancement	Génération de code
Choix	**Modèle**	Expérimental statistique	Expérimental statistique	Expérimental	Expérimental	Analytique Expérimental Apprentissage	Analytique Expérimen-tal	Analytique	Analytique	Analytique
	Dynamique	Non	Non	Non	Non	Non	Non	Oui	Oui	Non
	Evolutif	Non	Non	Non	Non	Oui	Non	Non	Non	Non

Tableau 3 – Caractéristiques de quelques applications adaptatives

5 Conclusion

La performance d'une application sur différentes architectures (parallèles ou séquentielles) est une métrique très importante pour l'utilisateur. En effet, l'utilisateur d'une application donnée cherche toujours à ce que son application soit la plus optimisée pour son contexte d'exécution. Vu que la performance des applications est régie par plusieurs paramètres architecturaux, les développeurs de ces applications ont adopté diverses techniques adaptatives afin de garantir les meilleures performances sur les différentes architectures considérées. Dans ce chapitre, nous avons présenté plusieurs techniques, à savoir la génération de code, la poly-algorithmique et les techniques d'ordonnancement. La principale mission dans toutes ces techniques consiste en le choix de l'alternative (algorithme, implémentation, ordonnancement) qui minimise le temps d'exécution sur la plate-forme considérée. Pour arrêter ce choix, l'application adaptative doit adopter une approche de modélisation de performance pour ses différentes alternatives. Les approches de modélisation de performance sont au nombre de trois (analytique, empirique et simulation), ayant chacune ses propres caractéristiques. Nous avons présenté dans ce chapitre ces différentes approches de modélisation de performance tout en essayant de les comparer.
Nous exposons dans le chapitre suivant une approche de développement d'applications parallèles adaptatives basée sur la modélisation théorique de performance.

Chapitre 3

Framework adaptatif

1 Introduction

Plusieurs travaux de recherche ont été réalisés dans le cadre de l'utilisation de méthodes adaptatives afin d'optimiser les applications séquentielles et parallèles. En effet, pour les plates-formes séquentielles, comme nous l'avons présenté dans le chapitre précédent, les applications adaptatives les plus remarquables sont ATLAS [9] et FFTW [42]. Pour le cas des plates-formes parallèles, les applications adaptatives sont nombreuses [17][43] [53]. . . .
Nous proposons dans ce chapitre un framework de développement d'applications adaptatives combinant les techniques adaptatives, à savoir l'approche poly-algorithmique avec les modèles de performance. Nous utilisons, dans une première phase (section 3), les modèles théoriques pour décrire le comportement des applications parallèles. Dans une seconde phase faisant l'objet de la section 4, nous essayons de dépasser les limites de la méthode analytique en la combinant avec une approche expérimentale.

2 Description du Framework

2.1 Motivation

Pour un problème donné, pouvant être résolu selon plusieurs alternatives, le choix d'une alternative pour traiter une instance de ce problème est difficile vu que la performance de chacune des alternatives est régie par plusieurs paramètres (architecturaux et algorithmiques). Cette difficulté devient accrue avec les plates-formes parallèles hétérogènes. De plus, le temps de traitement d'une instance donnée d'un problème, n'est pas le même d'une exécution à une autre, même si on utilise la même alternative dans les différentes exécutions. Ceci peut être expliqué par le schéma de communication de cette alternative qui varie avec le nombre de processeurs utilisés ou même avec l'ordre de ces processeurs (correspondance processus/processeur) pour un nombre fixé de processeurs. D'où l'importance d'un traitement de choix automatique entre les différentes alternatives pour un contexte d'exécution donné (instance, liste de processeurs d'exécution). Pour tirer avantage du choix (en terme de performance), il faut que la procédure de choix ne rajoute pas un surcoût important au temps de traitement d'un problème donné.
C'est dans ce contexte que nous proposons une méthodologie de couplage entre l'approche poly-algorithmique et la modélisation analytique. Cette méthodologie est conçue pour le développement d'applications s'adaptant d'une exécution à une autre par le choix de la variante qui réduira le temps de résolution de l'instance du problème considéré sur la plate-forme considérée. Cette adaptation se base sur la considération de paramètres reliés à l'instance du problème à traiter et des paramètres de la machine cible par le biais de formules analytiques. L'évaluation de ces formules au moment de l'exécution est la base du choix de l'alternative de résolution à utiliser pour cette exécution.

2.2 Aperçu du framework

Les principales idées derrière la conception de ce framework sont les suivantes :

- L'utilisation de la modélisation théorique des différentes alternatives du problème considéré vue sa généricité et sa rapidité d'utilisation.
- La détermination expérimentale de quelques paramètres algorithmiques et d'autres architecturaux qui peuvent être utilisés dans les modèles théoriques des différentes alternatives sans pour autant rajouter un surcoût important au mécanisme adaptatif.

La solution d'adaptation algorithmique proposée est schématisée dans la figure 9. Cette solution comporte trois principaux modules :

(i) **Module de découverte de la plate-forme :** il est responsable de la collecte des paramètres architecturaux de la plate-forme. La fréquence d'exécution de ce module peut varier entre une seule fois si les ressources de la plate-forme (machines et réseaux) sont statiques ou bien plusieurs fois (lors des changements sur ces ressources). L'exécution de ce module peut être effectuée d'une façon automatique si l'on lui fixe une périodicité.

(ii) **Module de modélisation de performance :** il est capable de générer les modèles à utiliser pour prédire le comportement des alternatives retenues pour le mécanisme adaptatif. Juste à ce niveau, cette tâche est effectuée manuellement par le concepteur de l'application. Ce module va connaître plusieurs améliorations tout au long du travail de thèse.

(iii) **Module d'exécution adaptative :** il est capable de choisir et de lancer la meilleure alternative (en termes de temps d'exécution global) pour exécuter l'instance du problème considérée. Il est à noter que le coût d'exécution de ce module est négligeable devant l'exécution de n'importe quelle alternative choisie surtout pour des instances de grande taille du problème.

2.3 Découverte de la plate-forme

L'objectif principal de ce module est de collecter, d'une façon automatique, quelques paramètres de performance de la plate-forme parallèle. Ces paramètres sont généralement relatifs aux performances des processeurs des différentes machines de la plate-forme ainsi qu'aux performances du réseau interconnectant ces machines.

Ce module requiert comme entrée la liste des machines composant la plate-forme d'exécution. Cette liste est fournie sous forme d'un fichier texte comportant autant de lignes que de nœuds dans la plate-forme (un nom de machine sur chaque ligne du fichier).

La détermination des paramètres relatifs aux performances des processeurs peut être effectuée en utilisant l'un des benchmarks communément utilisés comme LAPACK [8]. Ces paramètres peuvent être utilisés pour deux objectifs. Le premier concerne la détermination des performances

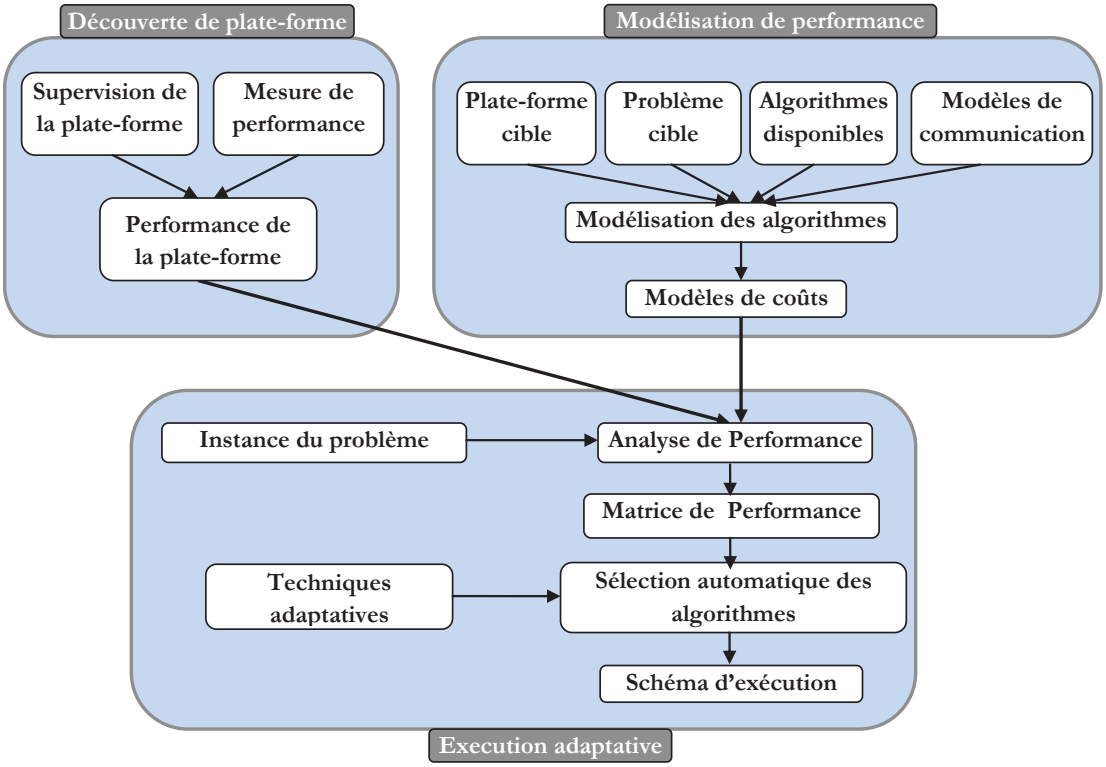


Figure 9 – Framework adaptatif

relatives à chaque type de processeurs de la plate-forme parallèle. Le second est pour leurs utilisation dans l'estimation des temps d'exécution des différentes parties de calcul que comportent les alternatives de résolution du problème considéré.
Pour déterminer les paramètres relatifs au réseau d'interconnexion, plusieurs outils sont susceptibles d'être utilisés. Parmi ces outils, citons NWS [89] qui permet de déterminer des paramètres *online* tels que les bandes passantes entre les nœuds de la plate-forme parallèle, la charge des processeurs et la mémoire disponible sur les nœuds.
Le pseudo code de ce module est représenté par l'algorithme 1. Notons que pour valider notre approche, nous avons développé deux programmes pour la mesure des performances des machines et du réseau. Le premier programme de mesure de performance de machines est un programme C ayant la structure d'un nid de 3 boucles réalisant une opération de produit matriciel. La taille des matrices à traiter est fixée pour ce programme à N=1000, d'où le nombre d'opérations à effectuer est égal à $2 * N^3 - N^2$. A partir du temps d'exécution de ce programme sur une machine donnée, on peut déterminer sa performance en divisant ce temps par le nombre d'opérations effectuées. Le second programme de détermination des paramètres du modèle de communication associés au réseau d'interconnexion de la plate-forme est un programme MPI utilisant la technique du ping-pong (voir annexes) pour mesurer la latence et la bande passante du réseau. Ce programme mesure le temps d'aller/retour d'un message d'un processeur à un autre pour déterminer la latence avec un message vide et la bande passante avec un message d'une taille choisie égale à 1 Mo.

Algorithme 1 Découverte de plate-forme

Entrées: Nodes_file *// fichier contenant les machines de la plate-forme*
Sorties: Perf_file*//fichier contenant les paramètres de performance de la plate-forme*
1: **Pour Tout** $Machine_du_fichier_Nodes_file$ **Faire**
2: $Mesurer_performance_processeur(Machine, Perf_file)$
3: **FinPour**
4: **Pour Tout** $Couple_de_machines_du_fichier_Nodes_file$ **Faire**
5: $Déterminer_parmètres_réseau(Machine1, Machine2, Perf_file)$
6: **FinPour**

2.4 Modélisation de performance

Ce module a pour mission la détermination des modèles théoriques de chaque alternative de l'application adaptative. Ces modèles prennent en considération deux principales composantes, à savoir le coût de calcul et le coût de communication. La modélisation peut être effectuée d'une façon séparée pour chacune de ces deux composantes. En effet, les communications, par exemple, peuvent être modélisées selon l'un des modèles théoriques utilisés comme le modèle de

Hockney[51], LogP [30] ou pLogP [61] (ces modèles de communication sont présentés en détail dans le chapitre 4, section 5.2.1).
Cette tâche de modélisation est confiée au concepteur de l'application parallèle puisqu'elle requiert une expertise par les détails des tâches de cette application. Plusieurs paramètres architecturaux sont à la disposition de ce concepteur dans la tâche de création des modèles. Ces paramètres sont les performances des processeurs et celles du réseau d'interconnexion et qui ont été déterminés par le module de découverte de plate-forme.
Les modèles générés pour les différentes alternatives de l'application parallèle seront la base de tout choix (ou combinaison) entre ces dernières lors de l'exécution.

2.5 Exécution adaptative

Ce module se base sur les résultats fournis par les deux modules précédemment décrits. En effet, le traitement du problème considéré selon l'une des alternatives de résolution n'est effectué qu'après une prédiction de performance des différentes alternatives, sachant que la prédiction de performance d'une alternative de résolution est réalisée à travers l'évaluation du modèle théorique associé à cette alternative.
Pour illustrer le comportement de ce module, considérons un problème disposant d'un ensemble $A = \{A_1, A_2, ..., A_l\}$ composé de l alternatives de résolution (algorithmes ou implémentations parallèles) et d'une plate-forme composée de p nœuds. Considérons aussi le modèle de coût (temps d'exécution) $C(A_k, N_j)$ associé à l'alternative A_k sur le nœud N_j. La matrice de performance qui comporte l lignes et p colonnes peut être calculée selon ce modèle de coût de façon à avoir dans la case d'indices (i,j) la valeur $C(A_i, N_j)$.
Le coût d'exécution parallèle $C_{//}(A_k)$ de l'alternative A_k est déterminé par l'équation 3.1.

$$C_{//}(A_k) = max\{C(A_k, N_j), 1 \leq j \leq p\} \tag{3.1}$$

Si on adopte l'approche poly-algorithmique pour résoudre ce problème, le choix d'une alternative A_i comme étant la meilleure pour ce contexte d'exécution ne pourra être effectuée que lorsque l'équation 3.2 est vérifiée.

$$C_{//}(A_i) = min\{C_{//}(A_k), 1 \leq k \leq l\} \tag{3.2}$$

Notons que dans certaines applications, il est possible de combiner plusieurs alternatives pour traiter les différentes tâches de cette application. Dans ce cas, la matrice de performance est utilisée autrement. Prenons par exemple le cas de tâches indépendantes qui peuvent être exécutées sur les p nœuds, le choix de l'alternative de résolution qui va être retenue à s'exécuter sur le nœud N_i peut être effectué selon la formule $min\{C(A_k, N_i), 1 <= k <= l\}$.

3 Etude de cas

3.1 Problème cible et état de l'art

Pour la validation de l'approche proposée, nous avons choisi comme cas d'étude le problème du produit de matrices sur des environnements parallèles et hétérogènes. Il est à noter que ce problème a connu un grand intérêt durant plusieurs années [15, 62, 72]. La plupart des versions générées pour la résolution de ce problème se basent sur l'algorithme séquentiel standard pour la MM et ne considèrent l'hétérogénéité qu'au niveau des processeurs.
Lastovetsky a étudié dans [62] le problème NP-complet du partitionnement optimal des matrices sur p processeurs pour le compte des opérations d'algèbre linéaire, en particulier pour l'opération de la MM. La résolution de ce problème est réalisée à travers sa réduction à un problème de partitionnement d'un carré unitaire en p rectangles (notons que ce problème est lui-même NP-Complet). L'auteur de ce travail a proposé pour cela trois heuristiques, à savoir :

- La décomposition à base de colonnes : les rectangles forment des colonnes.
- La décomposition à base d'une grille : les rectangles de cette décomposition vérifient que toute ligne horizontale passe par le même nombre de rectangles et toute ligne verticale passe par le même nombre de rectangles.
- La décomposition cartésienne : Les rectangles de cette décomposition forment des lignes et des colonnes.

Dans [72], Ohtaki et al. ont proposé une version parallèle de l'algorithme de Strassen ciblant des clusters hétérogènes. Cette version est basée sur le couplage de l'algorithme de Fox (connu BMR) et l'algorithme de Strassen. Les auteurs ont montré que plus on décompose les matrices (on réduit la taille des blocs) plus on augmente le nombre total d'opérations requis par l'algorithme de Strassen. Pour optimiser l'opération de la MM, les auteurs ont résumé leurs objectifs dans la satisfaction des deux conditions suivantes :

- Assigner les blocs de matrices proportionnellement aux performances des processeurs.
- Ne pas augmenter le nombre de blocs pour que l'algorithme de Strassen se déroule mieux.

Pour répondre à ces deux exigences conflictuelles, Les auteurs ont adopté un modèle de programmation dynamique leur permettant de déterminer la meilleure décomposition des matrices.
Dans [54], les auteurs ont montré que l'utilisation de plusieurs niveaux algorithmiques peut contribuer à des performances meilleures que l'utilisation de chaque algorithme seul. En effet, l'approche proposée consiste à combiner plusieurs implémentations de la MM (Strassen, pdgemm, dgemm,. . .) tout en en constituant trois niveaux d'exécution. Dans le plus haut niveau, l'algorithme de Strassen (à une ou plusieurs niveaux de récursivité) est retenu pour offrir la possibilité de créer des tâches malléables et multi-processeurs. Dans un niveau intermédiaire plusieurs choix sont possibles. Parmi ces choix on trouve l'implémentation *pdgemm* de ScaLAPACK et une autre

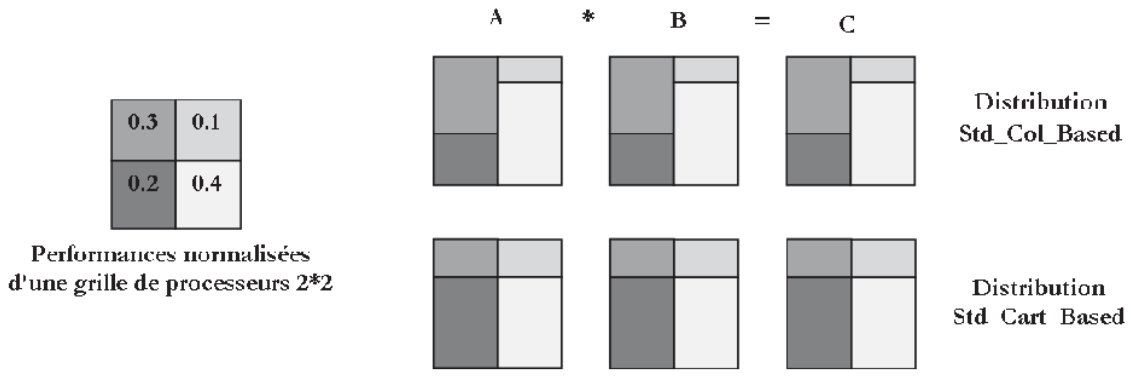


Figure 10 – Distribution des matrices pour l'algorithme standard

décomposition hiérarchique proposée par les mêmes auteurs nommé tpMM. Pour le niveau inférieur, ce sont les routines du BLAS ou d'ATLAS qui sont utilisées.

3.2 Alternatives retenues pour la multiplication matricielle (MM)

Les versions choisies pour jouer le rôle d'alternatives pour le mécanisme adaptatif au sein de notre framework sont les suivantes :

- **Version 1 (Std_col_based) :** c'est un algorithme proposé par Lastovetsky [62] basé sur l'algorithme standard. La décomposition des données est ajustée à la performance des machines d'exécution. Cette décomposition impose que les fragments des matrices à affecter pour les machines forment des colonnes (voir figure 10).
- **Version 2 (Std_cart_based) :** c'est un algorithme proposé aussi par Lastovetsky qui diffère de l'algorithme précédent par la décomposition des données sur les machines. En effet, les fragments de matrices de cet algorithme doivent former non seulement des colonnes mais aussi des lignes. En d'autres termes, les fragments forment une grille (voir figure 10).
- **Version 3 (Str_Cannon_based) :** c'est un algorithme inspiré du travail d'Ohtaky et al. [72]basé sur le couplage de l'algorithme de Cannon [21] au niveau supérieur et l'appel d'une itération de l'algorithme de Strassen dans le niveau inférieur (voir figure 11).

3.3 Modélisation de performance des alternatives de la MM

Pour la modélisation des communications que comporte les alternatives choisies pour la multiplication matricielle, nous avons adopté le modèle de Hockney [51] décrit par l'équation suivante :

$$T(n) = \beta + n/\tau \tag{3.3}$$

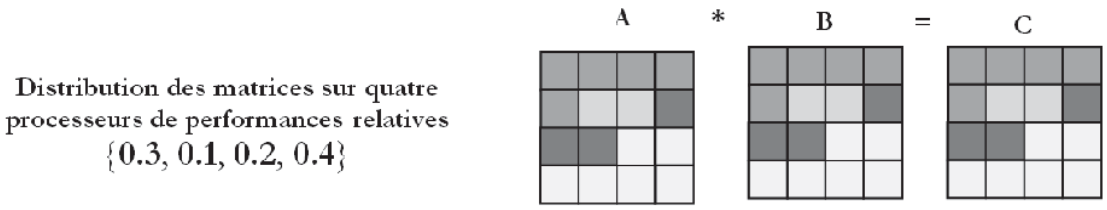


Figure 11 – Distribution des matrices pour l'algorithme Str_Cannon

Avec :

- n est la taille du message en termes de bits.
- $T(n)$ est le temps nécessaire pour le transfert de n bits et exprimé en secondes.
- β est la latence (ou temps nécessaire pour déclencher le transfert) exprimée en secondes
- τ est la bande passante du réseau exprimée en bit par seconde(bps).

Les modèles générés pour les différentes versions de la MM sont présentés dans le tableau 4.

Version de la MM	Modèle de performance
Std_col_based	$Texe(p_i) = Nb_Comm * (\beta + Bloc_Size^2/\tau) + 2 * Nb_Comp * Bloc_Size^3 * Perf(p_i)$
Std_cart_based	$Texe(p_i) = Nb_Comm * (\beta + Bloc_Size^2/\tau) + 2 * Nb_Comp * Bloc_Size^3 * Perf(p_i)$
Str_Cannon_based	$Texe(p_i) = Nb_Comm * (\beta + Bloc_Size^2/\tau) + (7/4) * Nb_Comp * Bloc_Size^3 * Perf(p_i)$

Tableau 4 – Modèles de performance théoriques des différentes versions de la MM

Avec :

- $Texe(p_i)$: le temps d'exécution pour le processeur i,
- Nb_Comm : le nombre de blocs envoyés/reçus par le processeur i ; ce paramètre est déterminé en fonction de l'instance du problème et de la décomposition des matrices.
- Nb_Comp : le nombre de blocs calculés par le processeurs i, déterminé en fonction de l'instance du problème et de la décomposition des matrices.
- $Bloc_Size$: la taille d'un bloc carré, les valeurs que peut prendre ce paramètre sont généralement 32, 64, 128 ou 256 (pour notre cas nous avons fixé ce paramètre à 128).
- $Perf(p_i)$: la performance du processeur i, déterminé lors de la découverte de plate-forme.
- β et τ : paramètres du modèle de Hockney, déterminés dans la phase de découverte de la plate-forme.

Le temps d'exécution parallèle de chacune de ces versions de la MM est décrit par l'équation 3.4.

$$T_{//} = max(Texe(p_i), 0 \leq i < p) \tag{3.4}$$

3.4 Performances des alternatives de la MM

Afin de voir l'intérêt de l'approche poly-algorithmique pour le cas du problème de la MM, nous avons mené des expérimentations sur deux plates-formes différentes (cluster de L'ESSTT et Grid'5000) et ce, pour plusieurs configurations (taille du problème et nombre de machines). Il est à noter que l'ensemble des machines retenues dans chacune de ces plates-formes est composé de deux types différents de machines. Le tableau 5 présente les détails des différents types de machines de ces plates-formes. Ces deux environnements de test opèrent sous le système d'exploitation Linux avec une version du noyau 2.6.13. Nous avons utilisé pour l'implémentation des différentes versions de la MM la version 1.0.7 de MPICH2.

	Type1	**Type2**
Grid'5000	Dual Opteron 2.0GHz, 2GB de RAM	Dual Opteron , Dual Core 2.6GHz, 4GB de RAM
ESSTT	Pentium 4 1.7GHz, 256MB de RAM	AMD Athlon 1.1 GHz, 384MB de RAM

Tableau 5 – Machines de test

La figure 12 représente les temps d'exécution des trois versions de la MM sur la plate-forme Grid'5000 pour respectivement p=7, 9, 12, 14 processeurs. La figure 13 représente les temps d'exécution de ces versions sur 2 sous-ensembles de processeurs de la plate-forme de l'ESSTT de tailles respectives p=6 et p=7. La répartition de ces nombres de processeurs sur les deux types de machines, pour chacune des deux plates-formes, est représentée dans le tableau 6. Les tailles du problème, dans ces expérimentations, varient respectivement entre 400 et 2400 avec un pas de 200 pour la plate-forme de l'ESSTT et entre 500 et 6000 avec un pas de 500 pour Grid'5000.

Grid5000		
p	**Type1**	**Type2**
7	3	4
9	5	4
12	7	5
14	8	6

ESSTT		
p	**Type1**	**Type2**
6	3	3
7	4	3

Tableau 6 – Processeurs des configurations de test

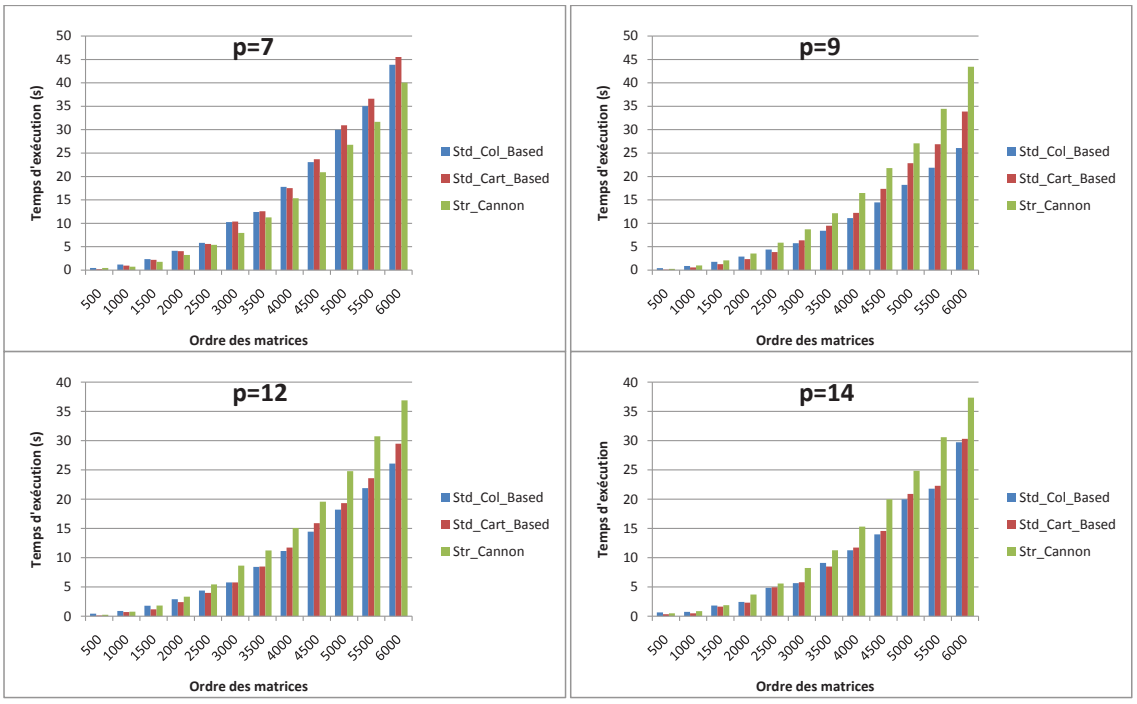


Figure 12 – Temps d'exécution des différentes alternatives sur Grid'5000

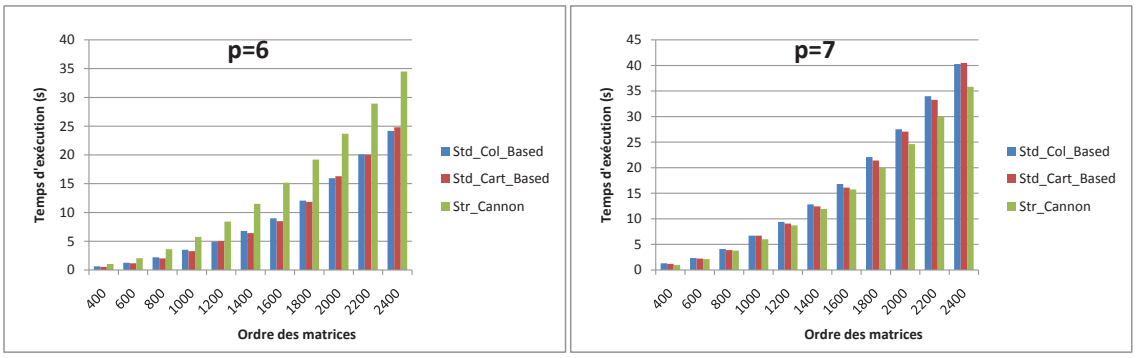


Figure 13 – Temps d'exécution des différentes alternatives sur le cluster de l'ESSTT

Le tableau 7 montre le meilleur algorithme pour les différentes expérimentations. A travers ce tableau, on remarque que le meilleur algorithme (parmi les trois algorithmes considérés) varie avec le nombre de processeurs (p) et la taille du problème (n). En effet, la version basée sur l'algorithme de Strassen domine (en termes de performance) les autres versions pour un nombre de processeurs égal à 7, et ce pour toute instance du problème sur les deux architectures. Pour les autres valeurs du nombre de processeurs (autres que 7), ce sont les deux versions basées sur l'algorithme standard qui dominent. Il est à noter qu'aucune version basée sur l'algorithme standard n'est la meilleure pour toute instance du problème et pour un nombre de processeurs donné. En d'autres termes, il y a toujours une alternance de cette dominance pour les différentes tailles du problème.

Contexte d'exécution		**Meilleur algorithme**	
nombre de processeurs(p)	**taille des matrices(n)**	**Grid'5000**	**ESSTT**
6	$[400, 1000]$ $]1000, 1200]$ $]1200, 1800]$ $]1800, 2400]$		Std_Cart_Based Std_Col_Based Std_Cart_Based Std_Col_Based
7	$\forall n$	Str_Cannon	Str_Cannon
9	$[500, 2500]$ $]2500, 6000]$	Std_Cart_Based Std_Col_Based	
12	$[500, 3500]$ $]3500, 6000]$	Std_Cart_Based Std_Col_Based	
14	$[500, 2000]$ $]2000, 3000]$ $]3000, 3500]$ $]3500, 6000]$	Std_Cart_Based Std_Col_Based Std_Cart_Based Std_Col_Based	

Tableau 7 – Meilleurs algorithmes de la MM

3.5 Prédiction de performance

Pour la prédiction de performance, nous avons évalué les équations de la sous-section 3.3 tout en utilisant les paramètres $Perf(p_i)$, β et τ déduits lors de la phase de découverte de la plateforme via deux benchmarks (de calcul et de communication). Les valeurs de ces paramètres, pour les deux plates-formes de test, sont récapitulés dans le tableau 8.

3.6 Critique

Pour sélectionner les choix idéaux par le mécanisme adaptatif, les modèles de prédiction doivent être très précis. La précision voulue ne peut pas être garantie par des modèles théoriques

		$\tau(Mo/s)$	$\beta(\mu s)$	$Perf(Mflops)$
Grid5000	Type1	98.43	47.92	1679.77
	Type2	98.43	47.92	1804.54
ESSTT	Type1	9.91	67.31	940.52
	Type2	9.91	67.31	782.03

Tableau 8 – Paramètres de performance des plates-formes de test

faisant abstraction de plusieurs phénomènes. En effet, avec de tels modèles, la prédiction de performance pour des applications complexes (en termes de schéma de calcul et de communications) sur des environnements hétérogènes d'exécution devient ardue.
Pour le cas des modèles théoriques générés pour l'application de la MM présentés, plusieurs remarques sont à signaler :

1. La performance (vitesse) d'une machine est supposée constante : utiliser une constante pour décrire la performance d'une machine ne peut faire aboutir à un modèle précis pour toutes les instances du problème. En effet, pour un problème donné, en variant la taille du problème et en mesurant le temps d'exécution pour chacune de ces tailles, on remarque que la performance n'est pas constante et tend généralement vers une valeur asymptotique.
2. La performance d'une machine est supposée indépendante de l'application : pour deux applications différentes, si on prend une instance de chacune de ces applications de sorte que leurs complexités (temporelles) soient égales, le temps mesuré pour ces deux instances n'est pas forcément le même. De ce fait, la performance de la machine varie avec chacune de ces applications.
3. La bande passante entre les nœuds de la plate-forme est supposée constante : la bande passante utilisée par le modèle de Hockney est une bande passante asymptotique.

4 Modélisation expérimentale de la performance des noyaux de calcul

Pour surmonter les deux premiers points faibles (dégagés dans la section précédente) et qui concernent la prédiction de performance des noyaux de calculs, nous avons opté à l'utilisation des modèles de performances basés sur des expérimentations. Ces expérimentations concernent les noyaux de calcul qui vont être utilisés dans les programmes parallèles sur les différents nœuds de la plate-forme. Pour aboutir à de tels modèles, nous nous sommes basés sur deux principales phases :

1. Phase de mesure de performance : cette phase permet de mesurer les temps d'exécution de chaque noyau de calcul sur les différentes machines de la plate-forme. Ces mesures concernent plusieurs instances (taille du problème traité par le noyau).
2. Phase de régression de données : les résultats de la phase précédente seront traitées par un outil de régression polynomiale pour générer les modèles théoriques pour chaque noyau de calcul et ce, pour les différents nœuds de la plate-forme.

4.1 Mesure de performance des noyaux séquentiels

Pour avoir une idée plus réaliste sur le comportement d'un noyau séquentiel sur une machine donnée, l'utilisation d'une série d'expérimentations de ce noyau sur la machine considérée représente la meilleure solution. Dans notre approche, ces expérimentations sont à effectuer lors de la phase d'installation de l'application adaptative. Par conséquent, ces expérimentations ne doivent pas durer longtemps. Pour considérer cette contrainte, nous avons conçu le scénario d'expérimentations suivant :

- On commence les expérimentations à partir d'une taille (du problème) N_0 qui est une valeur pré-définie ($N_0 = 100$ pour notre cas), on double à chaque fois cette taille et on enregistre le couple (taille, temps d'exécution).
- Le traitement se termine lorsque le temps d'exécution du noyau séquentiel dépasse une valeur limite T_{max} mesurée en termes de secondes devant être fixée à l'avance (pour nos expérimentations $T_{max} = 3$)

Il est à noter que, plus N_0 est faible plus le modèle est précis pour les petites instances. De même, plus T_{max} est élevé plus le modèle est précis pour de grandes instances du problème.

4.2 Génération de modèles

Pour la génération des modèles décrivant le comportement des noyaux séquentiels sur les différentes machines d'une plate-forme parallèle, nous avons utilisé la technique de régression de données collectées par la phase de mesure décrite ci-dessus. Cette méthode se base sur l'environnement R (voir Annexe B) pour déterminer les paramètres d'un modèle fourni par le concepteur de l'application.

Considérons un noyau de calcul bien définit, le concepteur doit spécifier une expression paramétrée (qui est le plus souvent sous forme d'un polynôme) en fonction de la taille du problème traité par ce noyau. Les paramètres à déterminer sont les coefficients de ce polynôme. A ce niveau, on va utiliser l'outil R et plus spécifiquement son module de régression pour déterminer les coefficients du modèle spécifié.

4.3 Validation expérimentale

Pour valider cette étape de génération de modèles en utilisant la régression sous le logiciel R, nous avons considéré deux noyaux de calcul scientifique, à savoir :

- Le *cblas_dgemm* pour le calcul de produit de matrices denses faisant partie de la bibliothèque ATLAS [9], pouvant être modélisé par un polynôme de degré 3 en n (n étant l'ordre des matrices) :
  $T(n) = an^3 + bn^2 + cn + d$
- Le *clapack_dtrtri* pour le calcul de l'inverse d'une matrice triangulaire faisant partie de la bibliothèque LAPACK [8], pouvant être modélisé aussi par un polynôme de degré 3 en n :
  $T(n) = a'n^3 + b'n^2 + c'n + d'$

Les machines de validation font partie de quatre clusters différents de la plate-forme Grid'5000. Les caractéristiques de ces machines sont résumées dans le tableau 19. Les paramètres des modèles générés par la régression sont représentés dans le tableau 10 pour le noyau *cblas_dgemm*, et dans le tableau 11 pour le noyau *clapack_dtrtri*.
On peut remarquer à partir de ces deux tableaux que les modèles polynômiaux de degré 3 générés à travers la procédure de régression décrivent bien le comportement des deux noyaux séquentiels associés. Cette remarque est basée sur les valeurs R^2 qui sont égaux à 1 pour la plupart des cas. Notons que ce facteur R^2 varie entre 0 et 1 et plus sa valeur est proche de 1, plus le modèle généré est précis.

Noeud (Cluster/Site)	`Caractéristiques`
Noeud1 (Azur/Sophia)	AMD Opteron 2Ghz, 2CPUs x 2cores, 2GB RAM
Noeud2 (Genepi/Grenoble)	Intel Xeon 2.5 Ghz, 2CPUs x 4 cores, 2GB RAM
Noeud3 (Bordereau/Bordeaux)	AMD Opteron 2.6Ghz, 2CPUs x 2cores, 2GB RAM
Noeud4 (Bordeplage/Bordeaux)	Intel Xeon 3Ghz, 2CPUs x 1core, 2GB RAM

Tableau 9 – Machines de validation

Pour valider cette approche de modélisation pour le cas de deux noyaux séquentiels sur les quatre machines choisies, nous avons exécuté ces noyaux pour différentes tailles des matrices. Ces tailles des matrices retenues pour cette validation sont générés d'une façon quasi-aléatoire. En effet, nous avons opté pour tester des valeurs aléatoires mais représentant une grande plage de tailles des matrices. Pour ce faire, nous avons choisi une valeur aléatoire entre chaque deux

	N œud1	N œud2	N œud3	N œud4
a	5.730e-10	4.443e-10	4.292e-10	5.181e-10
b	1.203e-07	1.635e-08	1.686e-07	3.697e-09
c	-2.274e-04	4.547e-06	-4.392e-04	1.582e-05
d	1.496e-01	-8.576e-03	3.280e-01	-1.440e-02
R^2	1	1	0.9998	1

Tableau 10 – Paramètres du modèle polynomial du noyau *cblas_dgemm*

	N œud1	N œud2	N œud3	N œud4
a'	1.011e-10	7.556e-11	7.862e-11	8.881e-11
b'	4.880e-08	3.356e-08	1.480e-08	3.747e-08
c'	1.852e-04	-2.335e-05	2.469e-04	2.170e-05
d'	-2.821e-01	2.575e-02	-3.020e-01	-5.775e-03
R^2	1	1	0.9998	1

Tableau 11 – Paramètres du modèle polynomial du noyau *clapack_dtrtri*

milliers successifs jusqu'à atteindre une taille des matrices qui ne peut être traitée en mode *"In-Core"*. Les tableaux 12 et 13 représentent les valeurs estimées (en utilisant les modèles générés) et mesurées réellement pour plusieurs tailles de matrices pour respectivement le emphcblas_dgemm et le *clapack_dtrtri*.

	Nœud1		Nœud2		Nœud3		Nœud4	
n	**M**	**E**	**M**	**E**	**M**	**E**	**M**	**E**
1680	2.84	2.82	2.14	2.15	2.14	2.10	2.48	2.48
2330	7.53	7.52	5.71	5.71	5.62	5.65	6.59	6.60
3980	37.23	37.27	28.29	28.28	28.24	28.31	32.80	32.77
4750	63.25	63.19	47.98	48.00	47.45	48.04	55.48	55.67
5160	80.68	80.90	61.52	61.49	61.22	61.52	71.34	71.35
6720	180.02	177.94	135.49	135.59	133.31	135.24	157.26	157.48
7240	225.49	222.26	169.95	169.49	168.17	168.87	197.05	196.91
8100	318.25	310.72	237.50	237.22	233.96	235.93	275.95	275.70
9450	499.61	492.30	377.00	376.44	374.40	373.44	440.80	437.69

Tableau 12 – Temps mesurés (M) vs temps estimés (E) pour le noyau *cblas_dgemm*

La formule utilisée pour déterminer l'erreur relative entre les valeurs estimés et les valeurs mesurés est comme suit :

$$Erreur = |temps_estimé - temps_réel|/temps_réel \tag{3.5}$$

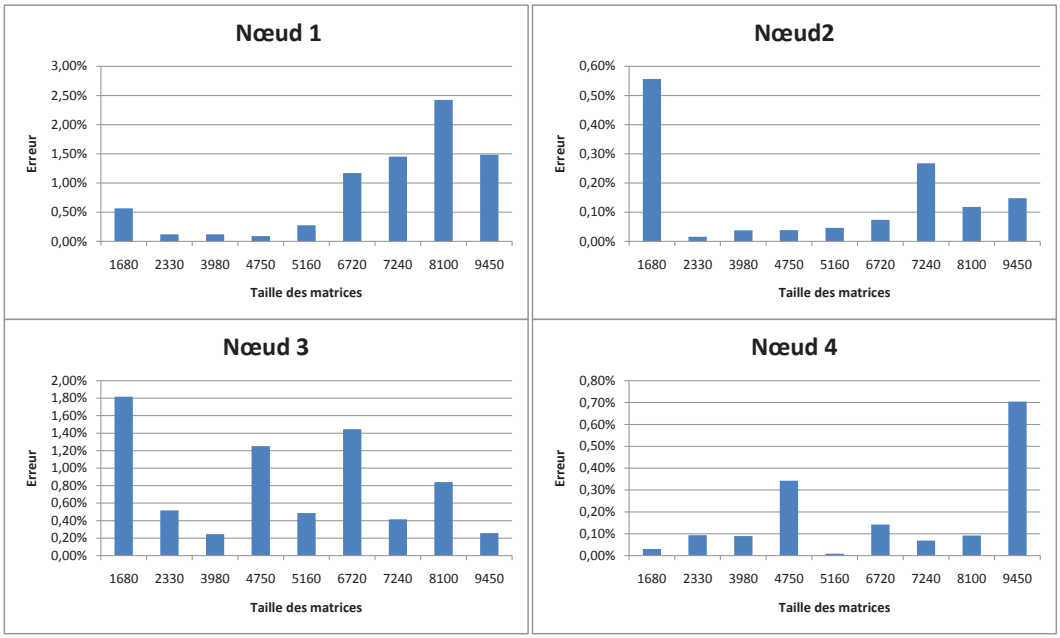


Figure 14 – Erreur de prédiction du noyau *cblas_dgemm*

	Nœud1		Nœud2		Nœud3		Nœud4	
n	**M**	**E**	**M**	**E**	**M**	**E**	**M**	**E**
1680	0.64	0.65	0.44	0.44	0.50	0.53	0.54	0.56
2330	1.57	1.69	1.10	1.12	1.21	1.35	1.33	1.37
3980	7.31	7.60	5.17	5.22	5.67	5.87	6.21	6.27
4750	12.37	12.53	8.77	8.70	9.40	9.63	10.41	10.46
5160	15.79	15.86	11.25	11.34	12.30	12.17	13.30	13.31
6720	33.98	33.85	24.49	24.28	25.56	25.88	29.00	28.78
7240	42.00	41.98	30.36	30.23	32.18	32.10	36.01	35.82
8100	58.30	58.15	41.86	42.11	43.78	44.45	49.43	49.83
9450	91.26	91.15	66.33	66.52	70.08	69.70	78.00	78.49
13450	258.62	257.03	189.95	189.67	193.51	196.99	223.00	223.15

Tableau 13 – Temps mesurés (M) vs temps estimés (E) pour le noyau *clapack_dtrtri*

Les figures 14 et 15 représentent les erreurs relatives entre les valeurs estimées et les valeurs expérimentales pour ces deux noyaux de calcul considéré et ce, pour les quatre machines de validation. A partir de ces résultats, on peut remarquer que les erreurs de prédiction sont minimes et comprises entre 0.01% et 2.42% pour le noyau *cblas_dgemm* et entre 0.13 % et 11.41 % pour le noyau *clapack_dtrtri*. De plus, l'erreur moyenne de prédiction pour les différents nœuds et les

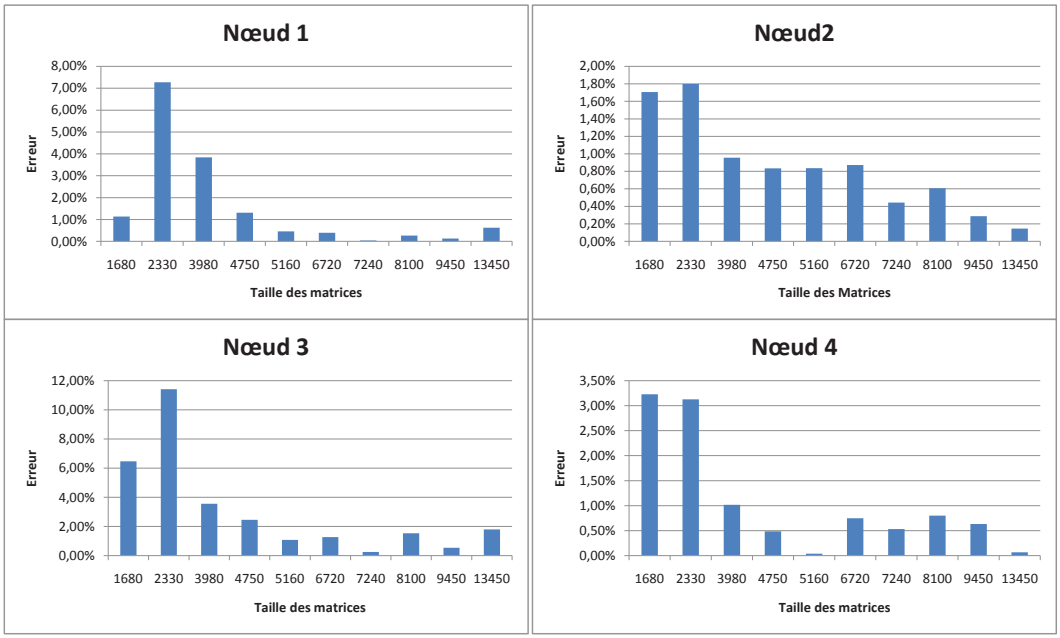


Figure 15 – Erreur de prédiction du noyau *clapack_dtrtri*

différentes valeurs de N est de l'ordre de 0.5% pour le noyau *cblas_dgemm* et de 1.8% pour le noyau *clapack_dtrtri*.

4.4 Critique

Il est à noter que malgré les bons résultats donnés par cette approche de régression, elle présente plusieurs inconvénients, dont :

1. L'intervention humaine lors de la spécification du modèle qui peut être la source d'une fausse prédiction en cas de spécification érronée d'un modèle.
2. La contrainte d'avoir le logiciel R installé sur la machine de prédiction.
3. L'erreur importante pour des problèmes de petites tailles.

5 Conclusion

Dans ce chapitre , nous avons proposé un framework adaptatif destiné au développement d'applications parallèles adaptatives et performantes. Ce framework est basé sur l'utilisation de modèles théoriques dans le mécanisme de choix entre alternatives. Nous avons appliqué la technique poly-algorithmique tout en se basant sur des modèles théoriques pour la description

des performances des différentes alternatives. Le principal objectif via ce choix étant d'avoir une phase rapide de choix d'alternatives. A la base des résultats obtenus lors de l'expérimentation du noyau de multiplication matricielle parallèle, nous constatons que la précision des prédictions d'un programme parallèle donné dépend des modèles utilisés pour décrire les comportements des parties de calcul et des parties de communication. Nous proposons dans le chapitre qui suit un framework automatisant la prédiction de performance tout en garantissant la précision et la rapidité de prédiction de ces deux parties.

Chapitre 4

Framework de modélisation de performance

1 Introduction

Pour appliquer une technique de choix d'alternatives dans le cadre de l'application d'une approche adaptative donnée, deux contraintes sont à considérer, à savoir la **rapidité** du processus de prédiction de performance des différentes alternatives et la **précision** de ces prédictions.
Dans ce chapitre, nous nous concentrons sur le problème de prédiction de performance de programmes parallèles, et nous proposons une solution satisfaisant les deux contraintes. Notons que les hypothèses sur le matériel et les applications, les techniques utilisées pour acquérir les données et les méthodes de prédiction sont les facteurs les plus importants pour déterminer la précision et la portée de l'utilisation d'une technique de prédiction particulière.
La solution proposée est sous la forme d'une méthodologie composée de deux phases (ou moments). La première phase est la phase d'installation de l'application adaptative dans laquelle plusieurs expérimentations sont à mener afin de satisfaire la contrainte de précision des prédictions. Les résultats de ces expérimentations seront synthétisés sous forme de formules analytiques afin d'avoir une évaluation de performance rapide dans la seconde phase du framework qui se déroulera au moment de l'exécution de l'application adaptative.
Les principales missions de ce chapitre étant de présenter globalement le framework proposé ainsi que son environnement architectural et applicatif, puis de décrire en détails les modules de sa phase d'installation. Ainsi, dans la section 2, nous décrivons l'environnement matériel et logiciel que nous ciblons dans notre solution. La section 3 est consacrée à la description d'un module de la phase d'installation du framework, à savoir le module de découverte de programmes. Nous détaillons dans la quatrième section le module de découverte de plates-formes. Dans la section 5, nous présentons en détail notre technique de modélisation de performance et ce, pour les noyaux séquentiels de calcul et les routines de communication pt/pt.

2 Environnement et aperçu de notre framework

2.1 Plates-formes cibles

Ces dernières années, les plates-formes parallèles utilisées dans la plus part des centres de calcul ne sont plus composées par des machines homogènes. En effet, ces plates-formes sont généralement composées de plusieurs types de machines (de différents constructeurs) et même de plusieurs générations du même type de machines. Ces plates-formes peuvent être considérées comme des plates-formes parallèles, hiérarchiques, hétérogènes et dédiées. En effet, leurs principales caractéristiques sont :

- La hiérarchie du réseau de communication : le réseau connectant les différents nœuds de la plate-forme peut être vu comme un arbre. Au plus haut niveau de l'hiérarchie, on peut voir le réseau reliant les différents sites (backbone de la grille) et dans les niveaux les plus

bas, on trouve les réseaux connectant les différents clusters d'un même site et les réseaux connectant les différents nœuds du même cluster.

- L'hétérogénéité des liens réseaux : vu l'utilisation de plusieurs types de réseaux (Myrinet, Infiniband, Giga Ethernet, . . .) dans les différents clusters, et l'utilisation d'un réseau routé pour connecter les différents sites, le réseau est forcément hétérogène.
- L'homogénéité des nœuds appartenant au même cluster : cette homogénéité touche les différents aspects (performance CPU, performance mémoire, et type de la carte réseau).
- L'homogénéité des liens réseaux connectant les différents noeuds d'un cluster (généralement, les nœuds d'un même cluster sont reliés à un même switch réseau).
- Les nœuds à utiliser par une tâche (qui est le plus souvent un processus MPI) sont dédiés à cette dernière. En d'autres termes, aucune autre tâche ne peut être lancée simultanément. Cette hypothèse est fondamentale pour plusieurs raisons. D'une part, la majorité des applications parallèles sont lancées d'une façon exclusive, d'autres part, la non satisfaction de cette hypothèse rend le problème non déterministe .

La figure 16 représente un exemple de ces plates-formes. Cette plate-forme multi-clusters est composée de 4 clusters répartis sur 3 sites distants.

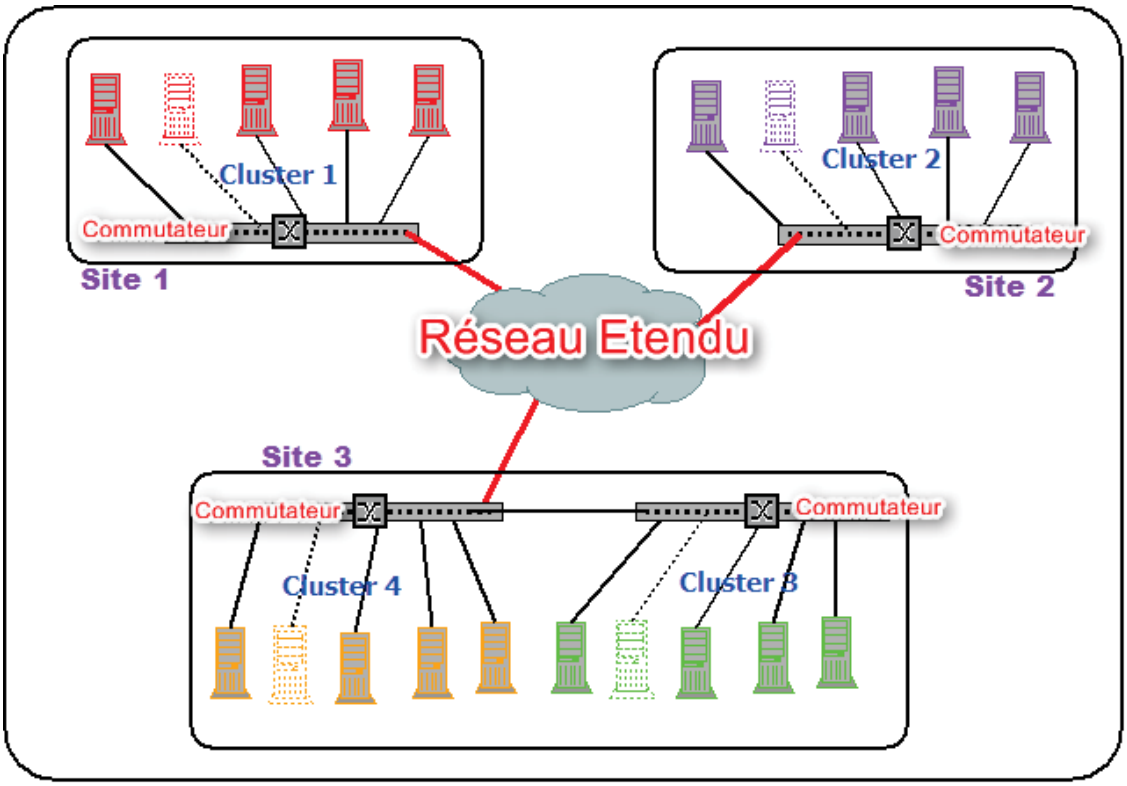


Figure 16 – Plate-forme Multi-clusters

2.2 Applications cibles

Les applications qui peuvent être améliorées en adoptant l'approche adaptative sont nombreuses. Parmi ces applications, on peut citer les bibliothèques parallèles de calcul numérique telles que ScaLAPACK [36] et PETSc [13]. En effet, l'enrichissement de ces bibliothèques par d'autres méthodes de résolution des problèmes numériques traités et l'utilisation de l'approche adaptative pour le choix de la meilleure méthode de résolution d'un problème numérique donné représentent un bon moyen pour améliorer la performance de ces bibliothèques sur les plates-formes parallèles qui ne cessent de changer.
De plus, l'utilisation de l'approche adaptative dans le cadre d'un environnement poly-librairie [5] peut donner de bonnes performances pour une plate-forme donnée.
Dans ce travail, nous focalisons notre étude sur les applications parallèles destinées à être exécutées sur un cluster ou une collection de clusters et adoptant le standard de communication MPI pour faire communiquer ses différentes tâches. Outre les appels aux routines de communication, les routines de ces bibliothèques font généralement appels à des noyaux de calcul séquentiels tels que l'appel des routines du BLAS dans la bibliothèque ScaLAPACK.
Dans ces bibliothèques, on ne s'intéresse qu'aux routines qui peuvent être exécutées selon le modèle adaptatif, c'est-à-dire disposant de plusieurs implémentations pour effectuer le même traitement.

2.3 Contraintes fonctionnelles

Le modèle à proposer dans le cadre de ce travail doit se baser sur les procédures communément utilisées dans l'installation et l'utilisation des ces bibliothèques. En effet, pour l'installation ou l'utilisation d'une bibliothèque donnée, il est fortement recommandée d'avoir des procédures automatisées afin de minimiser les interventions humaines qui peuvent être la source d'éventuelles pertes de temps (ou dégradation de performance).
Vu la nature de l'environnement logiciel et matériel et vu l'approche cible (approche adaptative), plusieurs contraintes sont à considérer lors de la conception de notre solution. Parmi ces contraintes, citons :

- **Une installation rapide** : l'installation de l'application ne doit pas durer longtemps (quelques minutes à la limite). Sachant que cette durée est régie par l'étendu de la plate-forme et la taille de l'application.
- **Des choix fiables** : pour appliquer l'approche adaptative convenablement, il faut baser les mécanismes de choix d'alternatives sur des modèles précis et ce, afin de choisir toujours la "meilleure" alternative en termes de performance.
- **La rapidité du choix** : la procédure de choix de la routine à exécuter doit être effectuée aussi rapidement que possible de sorte que cette durée soit minime devant le gain apporté par la méthode adaptative.

- **Transparence du choix** : lors de l'exécution d'une routine de l'application adaptative, le choix doit être transparent à l'utilisateur.

2.4 Éléments de notre solution

2.4.1 Motivation

Toute application parallèle adaptative impose à la méthode avec laquelle les prédictions de performance sont faites deux contraintes, à savoir la rapidité et la précision de ces prédictions. Ces deux contraintes peuvent être satisfaites comme suit :

- La précision des prédictions peut être satisfaite par des modèles qui se basent sur des expérimentations.
- La rapidité du processus de prédiction de performance peut être satisfaite par des modèles rapides à consulter comme les modèles théoriques.

Ces deux contraintes sont conflictuelles et ne peuvent être satisfaites par une seule méthode de modélisation de performance (expérimentale, théorique ou simulation).
Notre idée pour satisfaire ces contraintes consiste à combiner les différentes méthodes de modélisation de performance, tout en respectant les autres contraintes fonctionnelles. En effet, les expérimentations peuvent être effectuées dans la phase d'installation de l'application adaptative. Toutefois, ces expérimentations ne vont toucher que les briques composant l'application parallèle. Ces briques sont de deux types :

- Noyaux de calculs : ce sont les fonctions séquentielles fréquemment utilisées par l'application parallèle considérée.
- Les routines de communication : seules les routines pt/pt vont être considérées puisque toute autre routine (collective) est basée elle même sur ces routines pt/pt.

Les résultats de ces expérimentations vont être synthétisés sous forme de modèles théoriques pouvant être consultés rapidement au moment où l'on veut prédire la performance de l'application, c'est-à-dire lorsqu'on aura les paramètres d'exécution à notre disposition.
Sur un environnement d'exécution hétérogène, la prédiction de performance de l'application parallèle demande une simulation de son comportement tout en consultant les modèles théoriques des différentes composantes de l'application (routines de calcul et routines de communication).

2.4.2 Aperçu sur le framework

Notre solution schématisée dans la figure 17est basée sur deux principaux moments :

1. Moment de l'installation de l'application adaptative : dans cette phase, plusieurs procédures sont à exécuter afin de déterminer les modèles de performance des noyaux de calcul et des routines de communication pt/pt. Les principaux modules de cette phase sont :

- Module de découverte de plate-forme : qui, à partir de la liste de machines de la plate-forme, permet de déterminer les différents clusters composant cette dernière tout en associant chaque machine à son cluster.
- Module de découverte de programme : qui à partir des programmes sources permet de déterminer les traces de calcul et de communication que comporte ces programmes.
- Module de modélisation de performance : ce module se base sur les résultats des deux autres modules pour générer des modèles de calcul des différentes noyaux de calcul sur les différents clusters de la plate-forme. Cette modélisation touche aussi les communications pt/pt dans/entre les différents clusters.

2. Moment d'exécution : c'est le moment où l'on effectue la prédiction de performance à travers une simulation rapide tout en se basant sur les paramètres d'exécution (paramètres du problème et de la plate-forme) et sur les fichiers obtenus au moment de l'installation. En effet, en se basant sur le nombre de processus d'exécution et sur les traces de calcul et de communication obtenues à travers la phase de découverte de programmes, les tâches de différents processus seront identifiées. La prédiction de performance de chacune de ces tâches est effectuée à travers une évaluation des modèles générés dans la phase d'installation ou bien à travers une émulation rapide pour le cas des communications collectives. La détermination de l'ordonnancement final de ces tâches est effectuée à travers un programme de simulation permettant de déterminer principalement le temps de début et le temps de fin de chaque tâche.

3 Découverte de programmes

3.1 Objectif

Comme le montre la figure 17, ce module fait partie de la phase d'installation de l'application adaptative. Le principal objectif de ce module étant la détermination des différentes structures et instructions que contient l'application en cours d'installation, sachant que les principales structures à déterminer sont les routines de communication et les noyaux de calcul. Ces structures seront les seules considérées et consultées lors de la prédiction de performance d'un programme de cette bibliothèque au moment de l'exécution. Les entrées de ce module sont le code source de l'application qui est généralement écrit en langage C.
Soit, par exemple, un programme MPI écrit en langage C. Si on applique ce module sur ce programme, un fichier résultat du type "*xml*" est à générer. Ce fichier contient les détails de toute instruction du programme considéré.

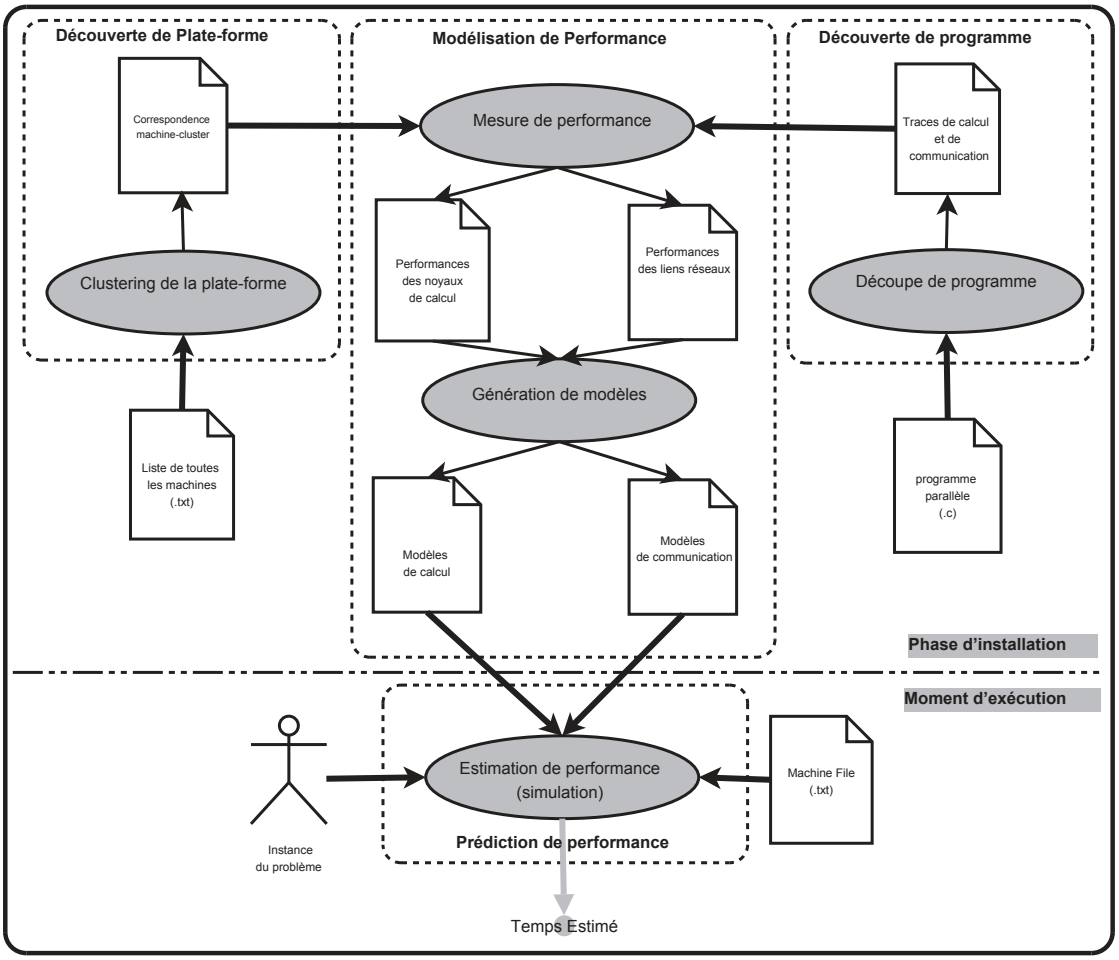


Figure 17 – Modèle global

3.2 Technique de découpage

Pour la découverte de programmes, nous avons basé notre solution sur la technique de découpage de programmes. Cette technique a été proposée pour la première fois en 1979 par Weiser dans sa thèse de doctorat [88].

Conceptuellement, la technique de découpage est simple et consiste à supprimer du programme toutes les lignes (ou instructions) qui n'interviennent pas dans une sémantique bien déterminée. Cette sémantique s'appelle le critère de découpage. Grâce à cette technique, un programme volumineux peut être divisé en n "sous-programmes " de tailles réduites. Chaque sous-programme possède ses propres variables, ses propres contrôles, ses propres instructions et surtout sa propre sémantique. Les fragments ou les blocs trouvés à l'aide de cette technique sont plus simples et plus compréhensibles que le programme original, ce qui permet en retour une manipulation plus

simple du programme.
D'autres détails sur cette procédure de découpage de programmes sont présentés en Annexe C.

3.3 Critères de découpe

Les programmes considérés dans ce travail sont écrits en langage C (ANSI C) et utilisant la bibliothèque de communication MPI. Les principaux critères de découpe de ces programmes sont relatifs aux routines de calcul et de communication utilisées, ainsi que les variables dont elles dépendent.
Les routines de communication de la bibliothèque MPI sont reconnues dans cette étape de découpe de programmes par leurs initiales "MPI_". Les routines de calcul sont aussi reconnues par des initiales tels que "clapack_" si l'on utilise la bibliothèque LaPACK et "cblas_" lors de l'utilisation de la bibliothèque ATLAS.
La première étape dans le processus de découpe de programme est celle de la détection des procédures ou fonctions de calcul et de communication dans le programme. Suite à cette étape, le processus de découpe associé à chacune de ces fonctions est effectué. Dans le processus de découpe, on distingue deux types de variables :

1. Des variables ordinaires : ce sont les variables pour lesquelles les valeurs peuvent être déterminées lors de la phase de compilation du programme. En d'autres termes, ces variables ne dépendent pas des paramètres d'exécution.
2. Des variables spécifiées lors de l'exécution : parmi ces variables, on cite :
   - Le nombre de processus relatif à la variable de la routine $MPI_Comm_size(Communicateur, \&variable)$;
   - Les variables d'exécution : qui peuvent être détectées à travers l'emploi des éléments du tableau $argv$.

4 Découverte de plates-formes

4.1 Motivation

Dans une plate-forme multi-clusters, la détermination de l'étendu et la composition de chaque cluster est une tâche importante pour une programmation performante sur ces plates-formes. Ainsi, en réalisant cette tâche, la tâche du programmeur devient de plus en plus simple vue qu'il va considérer une représentation simplifiée de la plate-forme. Dans cette représentation, les performances des machines et des liens d'interconnexion sont supposées constantes dans un même cluster. Par conséquent, pour étudier un comportement donné sur les machines d'un cluster, il suffit d'effectuer cette étude sur une machine quelconque du cluster puis de généraliser pour le reste des machines. Finalement, pour étudier un comportement sur toute la plate-forme il suffit

de le faire sur chaque cluster s'il concerne un comportement intra-cluster et de faire cette étude entre chaque couple de clusters s'il concerne un comportement inter-clusters.

4.2 Objectif

Notre objectif à travers ce module consiste à déterminer la structuration de la plate-forme parallèle en termes de clusters. Vu que nous ciblons des plates-formes dédiées, nous supposons que les machines de la plate-forme sont connues et la liste de leurs noms peut être spécifiée dans un fichier texte. Cela n'empêche pas que la procédure de détermination des machines soit automatisée si l'on utilise l'un des logiciels de monitoring réseau qui permettent de scanner un réseau de machines tels que Nagios[14], Remos [34] ou TopoMon [33].

4.3 Détermination des clusters

Pour aboutir à la structuration de la plate-forme en termes de clusters, nous nous sommes basés sur la latence du réseau qui est un paramètre résumant plusieurs comportements. En effet, la latence d'une communication entre deux machines dépend à la fois de la performance de ces machines ainsi que de la performance du réseau d'interconnexion. Pour caractériser un cluster d'une plate-forme donnée, il faut que les latences soient plus ou moins les mêmes pour chaque communication entre deux machines de ce cluster.
Notre solution a été implémentée avec un programme parallèle C+MPI(voir algorithme **??**) qui consiste à créer autant de processus (p) qu'il y a de machines dans la plate-forme (fichier Machine_File). Une mesure de latence est ensuite effectuée entre chaque couple de processus. Il est à noter qu'au niveau de cette étape, nous avons garanti qu'il n'y aura pas de chevauchements entre ces communications. Le résultat de cette étape est une matrice d'ordre p contenant les latences entre les différentes machines. En utilisant cette matrice de latences, nous avons déterminé les différents cluster en appliquent l'algorithme de Lowekamp [64].

Algorithme 2 Clustering de la plate-forme

Entrées: Machine_File // fichier contenant la liste des machines
Sorties: Machine_Cluster_File // fichier de correspondance machine/cluster

1: $p \leftarrow Déterminer_nombre_de_machines(Machine_File)$
2: $Mesurer_Latences_réseaux(p, NET - PERF)$ // NET-PERF : matrice d'ordre p contenant les latences inter-machines
3: $ECO(NET_PERF, p, Machine_Cluster_File)$ //clustering selon l'algorithme de Lowekamp

Nom de machine	cluster
bordereau-82.bordeaux.grid5000.fr	C1
genepi-9.grenoble.grid5000.fr	C2
azur-4.sophia.grid5000.fr	C3
genepi-6.grenoble.grid5000.fr	C2
bordeplage-35.bordeaux.grid5000.fr	C4
genepi-7.grenoble.grid5000.fr	C2
azur-7.sophia.grid5000.fr	C3
bordeplage-37.bordeaux.grid5000.fr	C4
bordereau-63.bordeaux.grid5000.fr	C1

Tableau 14 – Résultat de la correspondance entre machines et clusters

	C1	C2	C3	C4
C1	47.08	5391.00	7173.53	74.98
C2	5391.00	62.04	4517.25	5412.28
C3	7173.53	4517.25	47.92	7187.96
C4	74.98	5412.28	7187.96	49.94

Tableau 15 – Latences intra et inter-clusters

4.4 Validation

Pour valider ce module, nous avons appliqué l'algorithme 2 sur un fichier contenant les noms de neuf machines de la plate-forme Grid'5000. Les clusters détectés étaient au nombre de quatre et la correspondance entre ces machines et ces quatre clusters est décrite par le tableau 14. A partir de ce tableau on peut remarquer que les quatre clusters correspondent respectivement aux clusters *bordereau*, *genepi*, *azur* et *bordeplage*.
Les latences intra et inter-clusters sont résumées dans le tableau 15. Ces latences sont mesurées en *microsec*.
Bien que les clusters *bordereau* (C1) et *bordeplage* (C4) fassent partie d'un même site, ils ont été considérés comme différents puisque la latence inter-cluster (74.98 μs) est loin des latences intra-clusters (47.08 μs et 49.94 μs).

5 Modélisation de performance

Due à la particularité structurale et sémantique des programmes MPI (alternance de communications et de calculs), la prédiction de performance de ces programmes nécessite :

1. Des modèles précis décrivant le comportement des régions de calcul que comportent ces programmes.

2. Des modèles de communication décrivant le comportement des routines de communication évoquées par ces programmes.
3. Une modélisation fidèle(simulation) du déroulement des tâches de ces programmes.

Notons bien que la qualité de l'estimation de performance d'un programme MPI dépend des trois modèles sus-cités. Dans ce chapitre, nous nous intéressons aux deux premiers points qui concernent la génération de modèles pour les calculs et les communications. La simulation du déroulement de ces tâches fera l'objet du chapitre 5.

5.1 Modélisation des calculs

5.1.1 Motivation

Dans plusieurs travaux [3, 29, 65], la modélisation théorique du temps d'exécution d'un programme séquentiel est exprimée en fonction (i) du nombre d'opérations dans ce programme et (ii) de la performance de la machine d'exécution (exprimée en terme de Mflops). Ces modèles théoriques restent peu fiables pour décrire le comportement de tout programme séquentiel sur toute machine et pour toute instance du problème.
D'autres recherches ont montré que la performance des programmes séquentiels ou des parties séquentielles dans des programmes parallèles ne dépend pas seulement de ces deux facteurs (complexité du programme et performance théorique de la machine). En effet, plusieurs autres facteurs sont à considérer afin d'avoir une modélisation fiable de la performance de tels programmes. Le plus important de ces facteurs réside dans l'effet de la hiérarchie des mémoires sur l'exécution des ces programmes. Par exemple, dans [22], ce facteur mémoire a été retenu pour résumer le comportement des parties séquentielles d'un programme parallèle. L'étude permettant de voir l'effet de la hiérarchie mémoire sur la performance d'une application donnée nécessite un temps important vu qu'elle se base sur une exécution réelle de cette application ou bien sur une simulation avec un simulateur ayant un facteur de ralentissement important. En outre, cette étude est dépendante de l'instance du problème, d'où la nécessité de la refaire pour chaque instance.
En résumé, considérer uniquement l'approche de modélisation théorique pour la description du comportement d'une application séquentielle donnée ne donne pas une précision élevée lors de la prédiction de performance. D'un autre coté, l'utilisation de procédures de profiling de cette application séquentielle requiert un temps considérable. Par conséquent, l'emploi de l'un de ces deux modes de modélisation n'est pas convenable pour le contexte d'une application adaptative. D'où, la nécessité d'avoir une approche de modélisation garantissant un compromis entre la précision des prédictions et la rapidité de la procédure de prédiction.

5.1.2 Approche de modélisation

Pour établir le compromis entre la précision et la rapidité des prédictions de performance d'un programme séquentiel donné, nous avons basé notre solution sur trois principales tâches :

- Une expérimentation rapide des différents noyaux de calcul utilisés dans l'application adaptative et ce, sur des machines représentatives de chaque cluster.
- Une génération de modèles génériques à travers l'utilisation de la régression polynomiale de chacun des noyaux de calcul sur chaque cluster.
- Un raffinement des modèles obtenus pour garantir un niveau de précision acceptable. En d'autres termes, le comportement d'un noyau de calcul donné sur un cluster donné peut être représenté par plusieurs modèles polynomiaux où chaque modèle est adéquat un intervalle de taille de problème.

Comme le montre la figure 17, les noyaux séquentiels à modéliser sont déterminés par le module de découverte de programmes. Les machines sur lesquelles les expérimentations sont faites sont des machines représentatives de chaque cluster sachant que ces clusters sont déterminés à travers le module de découverte de plate-forme.

A ce niveau, nous avons utilisé la procédure décrite dans le chapitre 3, section 4.1 pour effectuer les expérimentations de la phase d'installation de l'application adaptative. Mais, pour la méthode de régression de données, nous avons opté pour le développement d'une solution automatisée pour la génération de modèles de performance. En effet, cette solution permet de surmonter les limites de la régression semi automatique (présentée dans le chapitre 3, section 4.2). Notre solution est développée en langage C et permet d'effectuer la régression polynomiale associée aux données d'un fichier contenant les résultats expérimentaux acquis lors de la phase de benchmarking. Les caractéristiques de cette méthode de régression sont les suivantes :

- Automatisation de la détermination du meilleur modèle polynomial.
- Pour le même noyau de calcul, on détermine plusieurs modèles pour plusieurs intervalles ayant chacun un niveau de précision acceptable.

Le détails de cette méthode sont donnés dans l'algorithme 3 et l'algorithme4.

L'algorithme 3 est équivalent à la méthode de régression de l'outil R. Cet algorithme permet de déterminer, pour un fichier de mesures, les coefficients du polynôme de degré DEG représentant la fonction d'approximation de ces données. Les entrées de cet algorithme sont : le fichier de mesures (FILE_IN) et le degré du polynôme (DEG). Les coefficients du polynôme résultat sont stockés dans un fichier (FILE_OUT).

L'algorithme 4 donne une idée sur notre solution automatisée pour la détermination des modèles associés à un noyau de calcul donné. La seule entrée de cet algorithme est le fichier contenant les mesures de performance de ce noyau. La première étape de l'algorithme est la détermination du polynôme le plus adéquat pour la régression de l'ensemble de ces données (c'est-à-dire qui

maximise le R^2). Dans une seconde étape, on vérifie si le modèle généré permet de prédire la performance de ce noyau avec la précision voulue. Ce contrôle est effectué sur la base de la comparaison entre l'erreur relative et le seuil accepté (dans notre cas $THRLD = 5\%$). En cas de détection d'un intervalle comportant au moins trois valeurs dont l'erreur est supérieure à $THRLD$, une autre régression polynômiale est effectuée pour cet intervalle. Le résultat de cet algorithme consiste en un fichier comportant plusieurs modèles polynômiaux, où chacun de ces modèles correspond à un intervalle bien déterminé de valeurs.

Algorithme 3 Régression Polynomiale Simple(*Polynomial_Regression*)

Entrées: FILE_IN, DEG // *fichier des mesures collectées lors de la phase de benchmarking, et degré du modèle souhaité*
Sorties: FILE_OUT //*fichier contenant les paramètres du modèle*
1: $lignes \leftarrow Lecture_fichier(FILE_IN, x, y)$ //*lecture des donnes du fichier (les tailles dans x et les mesures dans y), lignes est le nombre de mesures*
2: $Mise_en_eq(lignes, DEG + 1, x, y, matA, matB)$ //*mise en équation pour pouvoir résoudre ce pb comme un système linéaire*
3: $Gauss_pivot_total(matA, matB, DEG + 1, FILE_OUT)$ //*résolution selon la méthode de Gauss, les résultats (les coefficients du polynôme) sont enregistrés dans le fichier de sortie*

Algorithme 4 Régression Raffinée

Entrées: FILE_IN // *fichier des mesures collectées lors de la phase de benchmarking*
Sorties: FILE_OUT //*fichier contenant les intervalles et les paramètres des modèles*
1: $DEG_MAX \leftarrow 5$ // *le degré du polynôme est entre 0 et DEG_MAX*
2: $THRLD \leftarrow 5$ // *seuil = pourcentage d'erreur accepté*
3: $OPT_DEG \leftarrow 0$ // *degré du polynôme optimal*
4: $OPT_R_SQ \leftarrow 0$ // R^2 *optimal*
// *REGRESSION SIMPLE*
5: **Pour** $DEG = 0$ to DEG_MAX **Faire**
6: $Polynomial_Regression(DEG, FILE_IN, FILE_OUT)$
7: $R_SQ \leftarrow R_Squared(DEG, FILE_IN, FILE_OUT, \ldots)$
8: **Si** $R_SQ > OPT_R_SQ$ **Alors**
9: $OPT_R_SQ \leftarrow R_SQ$
10: $OPT_DEG \leftarrow DEG$
11: **FinSi**
12: **FinPour**
13: $Polynomial_Regression(OPT_DEG, FILE_IN, FILE_OUT, \ldots)$
//*RAFFINEMENT DE LA REGRESSION*
14: **Tant que** $Detect_Error_Interval(THRLD, Interval, \ldots) = True$ **Faire**
15: $SIMPLE_REGRESSION(Interval, \ldots)$ //on fait appel à une itération de régression simple pour l'intervalle détecté (de la ligne 5 à la ligne 13)
16: **FinTantque**

Pour illustrer l'avantage de cette nouvelle procédure automatique de génération de modèles correspondants à un noyau de calcul donné, prenons l'exemple du noyau *cblas_dgemm* sur un PC ayant un processeur Intel Dual core. Les modèles générés en utilisant une régression simple et une régression raffinée sont présentés dans le tableau 16. Les erreurs de prédiction selon ces deux types de modélisation sont présentées dans la figure18. A partir de cette figure, on remarque bien que l'erreur de prédiction pour les tailles inférieures à 900 a diminuée considérablement, de sorte que l'erreur maximale pour toutes les valeurs est inférieure à 5% en utilisant cette méthode raffinée de régression.

Type de régression	**Intervalle**	**Modèle**
Régression simple	$\forall n$	$T(n) = 0.5169361978 - 0.0018161448 * n + 0.0000013326 * n^2 + 2 * 10^{-9} * n^3$
Régression raffinée	$n <= 900$	$T(n) = 0.0139757937 - 0.0000241256 * n + 1.44 * 10^{-8} * n^2 + 2.2 * 10^{-9} * n^3$
	$n > 900$	$T(n) = 0.5169361978 - 0.0018161448 * n + 0.0000013326 * n^2 + 2 * 10^{-9} * n^3$

Tableau 16 – Modèles des régressions simple et raffinée

5.2 Modélisation des communications

5.2.1 Etat de l'art

La modélisation théorique des communications a connu un grand intérêt par les chercheurs ces deux dernières décennies [6, 30, 50–52].
Le modèle le plus simpliste est celui proposé par Hockney [51]. En effet, ce modèle assume que le temps requis à l'envoi d'un message, à partir d'un processeur vers un autre, suit l'équation 4.1

$$T = \beta + L/\tau \tag{4.1}$$

Où β représente la latence, τ la bande passante du réseau et L est la taille du message. Malgré les bons résultats d'estimation de ce modèle pour des tailles importantes de messages, son majeur inconvénient réside dans la prédiction du temps de communication pour les petits messages.
Au contraire du modèle de Hockney, le modèle LogP [30] est adéquat seulement aux messages de petites tailles. Ce modèle tire son nom de ses paramètres :

- **L** : latence qui représente le délai du temps entre l'ordre d'envoi et le début effectif d'envoi.
- **o** : surcoût qui représente l'intervalle de temps dans lequel le processeur est occupé par l'envoi ou la réception d'un message, durant ce temps il ne peut effectuer aucune autre tâche.

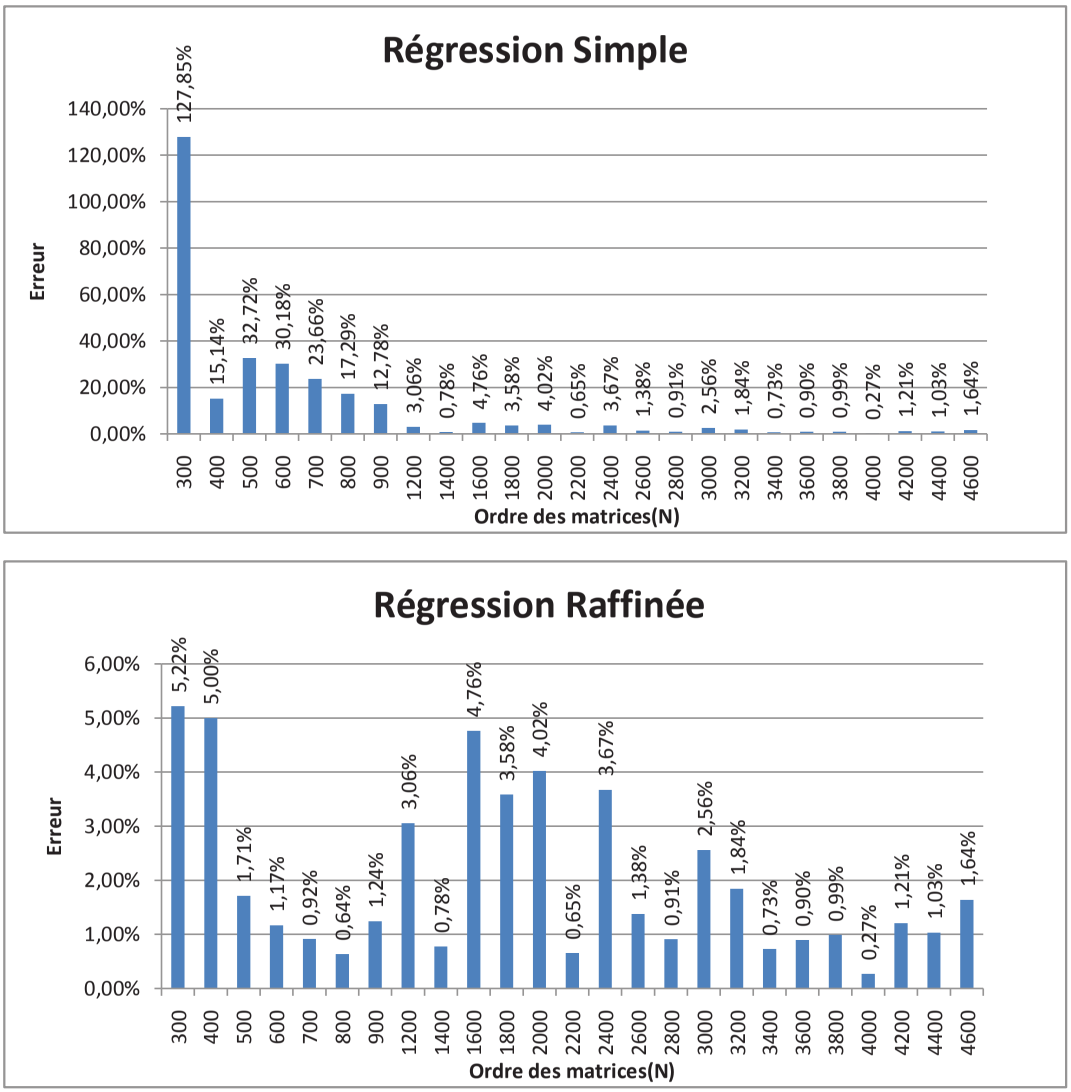


Figure 18 – Régression simple vs régression raffinée

- **g** (gap) : intervalle de temps entre deux envois/rceptions consécutifs .
- **P** : nombre de processeurs.

Le temps requis pour l'envoi d'un message d'un processeur vers un autre en utilisant le modèle LogP est décrit par l'equation 4.2.

$$T = L + 2 * o \tag{4.2}$$

Une généralisation du modèle Logp, pour les différentes tailles de messages, est donnée dans le modèle LogGP [6]. En effet, ce modèle garde les mêmes paramètres du modèle LogP et rajoute un paramètre G (Gap par bit) pour supporter les messages de différentes tailles. La durée d'une communication entre deux processeurs selon ce modèle est déterminée par l'équation 4.3.

$$T = L + 2 * o + G(m - 1) \tag{4.3}$$

Le modèle PlogP (Parametrized LogP) [61] représente une unification du modèle LogP et du modèle LogGP. Ce modèle suppose que tous les paramètres, sauf la latence L, sont des fonctions linéaires dépendantes de la taille du message. Ce modèle a redéfinit la valeur o en considérant le os(m) (surcoût coté émetteur) et or(m) (surcoût coté receveur). Le modèle PlogP assume qu'à tout moment $g(m) >= os(m)$ et $g(m) >= or(m)$.
Dans ce modèle, le temps requis pour envoyer un message d'un processeur vers un autre est décrit par l'équation 4.4.

$$T = L + g(m) \tag{4.4}$$

Une étude comparative de ces quatres modèles est donnée dans [76]. Plusieurs autres modèles ont été proposés comme LogGPO[26] et le modèle proposé par Javadi et al. [58]. Il est à noter que le modèle LogGPO est orienté pour l'amélioration des prédictions pour les routines de communication MPI pt/pt non bloquantes. Le modèle proposé dans [58] est un modèle analytique permettant de décrire le temps de communication dans un environnement multi-clusters. Il est à noter que ces deux modèles utilisent un grand nombre de paramètres pour modéliser les temps de communication.

Outre ces modèles théoriques, il existe plusieurs suites (*Benchmark*) permettant de mesurer expérimentalement les performances des routines MPI. Parmi ces outils on cite, IMB (*Intel MPI Benchmark*) [57], MPIBench [46–48], SKaMPI [80], Mpptest [45] et Netpipe[82]. Certains de ces outils sont maintenus, à savoir IMB et MPIBench. Les autres outils ne sont plus maintenus, mais restent fonctionnels et originaux de par la méthode utilisée pour mesurer les appels, ou la présentation des résultats. Notons que l'ensemble de ces suites mesurent la moyenne du temps pour un ensemble de routines, et ceci pour plusieurs tailles de messages et pour un nombre de processeurs spécifié. Le pseudo-code utilisé par la plupart de ces suites est décrit par la figure 19.

5.2.2 Discussion

Pour plusieurs raisons, l'utilisation des modèles théoriques pour le contexte de plates-formes hétérogènes et hiérarchiques ne représente pas la meilleure solution. Parmi ces raisons, les modèles théoriques sont plus adaptés pour des librairies de communication de bas niveau comme *Active*

```
Boucle sur les routines MPI
    Boucle sur les tailles de messages
        Temps_début =get_time()
        Boucle sur les répétitions
            Si la routine est collective Alors
                Mettre une barrière
            FinSi
            Appel de la routine
        FinBoucle
        Temps_fin=get_time()
        Temps_moy=(Temps_fin - Temps_début)/répétitions
    FinBoucle
FinBoucle
```

Figure 19 – Pseudo-Code des benchmarks des routines MPI

message ou la librairie d'Elan [75]. En d'autres termes ces modèles ne tiennent pas compte des effets rajoutés par les librairies de haut niveau, telle que MPI. De plus, ils ne traitent pas plusieurs autres phénomènes réseau, comme la contention des liens, les délais de routage et la congestion dans les routeurs pour le cas des réseaux routés.
D'autre part, l'utilisation d'une suite de "*Benchmarking*" ne convient pas des architectures étendues et hétérogènes. En effet, ceci est due à la combinatoire importante de tests à réaliser pour les différentes routines de communication, en particulier pour les routines de communication collectives.

5.2.3 Notre approche de modélisation

Pour l'estimation de performance de toute routine de communication (que ce soit pt/pt ou collective), nous avons choisi de baser nos modèles sur les modèles des routines pt/pt. Ce choix est effectué vue que les routines collectives ne sont qu'un ensemble de routines pt/pt organisé selon un schéma bien déterminé.
La modélisation des routines pt/pt est réalisée d'une façon similaire à celle des noyaux de calcul. En effet, cette opération est constituée de deux principales phases, à savoir la phase de mesure

de performance (*benchmarking*) et la phase de génération de modèles.

Mesure de performance

Dans cette phase, on expérimente les routines pt/pt bloquantes (*MPI_Send/MPI_Recv*) et non bloquantes (*MPI_Isend/MPI_Irecv*), et ce entre chaque couple de clusters (en choisissant des machines représentatives de chaque cluster) et entre deux machines du même cluster. Cette opération de "*benchmarking*" est effectuée à travers un programme parallèle optimisé dans l'objectif de réduire la durée de cette phase.
Pour la mesure de performance entre deux machines (inter-clusters ou intra-cluster), ce programme varie la taille du message et mesure le temps mis par cette communication.

Régression Linéaire progressive

Les résultats obtenus dans la phase de *"benchmarking"* seront utilisés par la suite par le module de modélisation de performance des communications en leur appliquant un algorithme de régression (algorithme 5). Le résultat de cet algorithme consiste en un fichier comportant les paramètres des modèles de performance des différentes communications intra et inter-clusters. Pour une communication donnée entre un couple de clusters (ou au sein du même cluster), cet algorithme doit déterminer les paramètres de plusieurs fonctions linéaires lors de la régression des données collectées. Chacune de ces fonctions permet de décrire le comportement de la communication pour un intervalle de taille de messages avec un niveau élevé de précision. Pour contrôler la précision des modèles fournis, le paramètre R^2 est alors utilisé et ne doit pas être inférieur à un seuil fixé à l'avance (0.99 pour notre cas).

Algorithme 5 Régression linéaire progressive

Entrées: FILE_IN // *fichier contenant les mesures du benchmark*
Sorties: FILE_OUT //*fichier contenant les intervalles et les paramètres des modèles*

1: $THRLD \leftarrow 0.99$ // *seuil de* R^2
2: $N \leftarrow Copy_Data_To_Array(FILE_IN, TAB)$ // N =*nombre des mesures*
3: $Start \leftarrow 1$
4: **Pour** $End = 2$ to N **Faire**
5: $Linear_Regression(TAB, Start, End \ldots)$
6: $R_SQ \leftarrow R_Squared(TAB, Start, End, \ldots)$
7: **Si** $R_SQ < THRLD$ **Alors**
8: $Save_Interval(Start, End-1, FILE_OUT, \ldots)$ //*enregistrer les informations sur l'intervalle*
9: $Start \leftarrow End - 1$ //*nouveau intervalle*
10: **FinSi**
11: **FinPour**

6 Conclusion

Dans ce chapitre, nous avons présenté la première phase de notre framework (cf. fig 17) de prédiction de performance de programmes parallèles sur des environnements hiérarchiques. Le principal objectif de cette phase est la détermination des modèles de performance associés aux noyaux de calcul et aux routines de communication MPI pt/pt. Ces modèles constituent la base de toute prédiction de performance de ces programmes au moment de l'exécution (ou lorsqu'on spécifie les paramètres d'exécution). Les modules développés pour le compte de cette phase ont été conçus de sorte à avoir la moindre intervention humaine, et ce afin d'avoir une phase automatisée d'installation des bibliothèques adoptant ce framework. La description de la seconde phase de notre framework qui concerne la prédiction de performance proprement dite fera l'objet du chapitre qui suit.

Chapitre 5

Prédiction de performance d'applications MPI

1 Introduction

La prédiction de performance d'un programme parallèle est une étape primordiale pour plusieurs tâches telles que l'optimisation des programmes, l'équilibrage des charges (load balancing), la comparaison des programmes et la découverte des détails de temps d'exécution de chaque partie du programme. Avec le nombre important de paramètres pouvant influer l'exécution d'un programme parallèle notamment pour le cas de contextes hétérogènes d'exécution, cette tâche de prédiction devient un véritable défi. Cette tâche devient de plus en plus difficile si l'on rajoute une contrainte sur le temps du processus de prédiction.
Nous détaillons dans ce chapitre notre solution intitulé MPI-PERF-SIM de simulation de programmes MPI sur des plates-formes parallèles hiérarchiques. Cette solution s'intègre dans le cadre du framework proposé dans le chapitre 4 pour la prédiction de performances des programmes parallèles. L'objectif principal de ce module de simulation étant de prédire la performance des applications parallèles tout en garantissant une phase rapide de prédiction et une précision satisfaisante des résultats fournis. Ainsi, nous présentons dans la section 2 un état de l'art sur quelques simulateurs de programmes MPI. Dans la section 3, nous détaillons l'architecture, les entrées et les modules de notre simulateur MPI_PERF_SIM. Une application de ce simulateur sur le problème de produit de matrices est présentée dans la section 4.

2 Etat de l'art des simulateurs MPI

La simulation de programmes MPI a connu un grand intérêt par les industriels et les chercheurs ces dernières années [28, 52, 74, 85, 90]. Ceci est due au vaste emploi de ce type de programmes, ainsi qu'à l'importance de ce traitement dans le processus de développement et d'optimisation de ces programmes. Ces simulateurs peuvent être classifiés en deux principales classes, à savoir :

- Les simulateurs en ligne "*online*" qui sont nommés aussi simulateurs par exécution directe. Ces simulateurs se basent sur une exécution réelle de l'application avec une simulation de seulement une partie de cette dernière.
- les simulateurs "*offline*" connu aussi sous le nom de simulateurs en post-mortem. Ces simulateurs utilisent les traces de l'application.

Dans [28], le module SMPI (Simulated MPI) faisant partie du projet SIMGrid [23] est présenté. Ce module adopte une simulation en ligne pour le compte des programmes parallèles du type MPI. La simulation dans ce module concerne les appels aux différentes routines de communication (pt/pt et collectives) toute en considérant l'effet de la contention des liens réseaux. Les calculs dans les programmes simulés sont exécutés directement sur la machine de simulation. SMPI utilise des techniques de réduction de mémoire pour traiter des instances de grandes tailles en termes de nombre de processus et en termes de taille du problème.

Dans [90], les auteurs ont présenté un framework de prédiction de performance hors ligne des programmes parallèles sur des machines parallèles à large échelle et ce, en utilisant une seule machine de simulation. Ce framework, appelé PHANTOM, automatise la détermination des traces de calcul et de communication des programmes parallèles simulés à travers l'utilisation de l'outil FACT [91]. PHANTOM utilise le principe de "record/replay" pour construire sa base de log. Dans ce framework, la modélisation de performance est réalisé à travers des modèles analytiques. La convolution des performances des différentes parties du programme simulé (calcul et communication) est effectuée à l'aide de l'outil SIM-MPI [1].
LogGOPSim [52] est un autre framework de simulation hors ligne des applications parallèles adoptant MPI. Pour la prédiction de performances des communications dans ce framework, plusieurs modèles théoriques qui ont été retenus. Ces modèles sont tous de la famille LogP. Il est à noter que dans ce travail, un nouveau modèle théorique de communication nommé LogGOPS a été proposé et utilisé dans la simulation.
Dans [85], un simulateur hors ligne nommé PSINS est présenté. Ce simulateur accepte en entrée le modèle de communication, les traces du programme et la configuration de la machine pour fournir des statistiques sur les temps d'exécution de chaque partie de ce programme. Les traces de calcul sont obtenus à travers des exécutions du programme à l'aide du "PSINS Tracer". Ce simulateur offre la possibilité de traiter des traces collectés par d'autres simulateurs comme MPIDtrace ou TAU.

Il est à noter que la plupart des simulateurs présentés ou existants sont destinés pour la simulation de programmes parallèles sur des clusters homogènes. Dans notre travail, nous ciblons non seulement les clusters homogènes mais aussi les clusters hiérarchiques. Une autre différence de notre simulateur avec ces simulateurs c'est que nous n'avons pas besoin d'avoir des traces d'exécution de toute l'application, mais nous contenterons de l'exécution des principaux noyaux de calcul et des routines de communication pt/pt.

3 MPI-PERF-SIM

3.1 Objectifs et démarche

Pour prédire la performance d'applications MPI sur des architectures parallèles hiérarchiques, nous avons développé un module intitulé MPI-PERF-SIM (*MPI Performance Simulation*) [1]. Ce module fait partie de la phase d'exécution du framework proposé dans le chapitre précédent (voir figure 20). En effet, MPI-PERF-SIM ne peut être appelé que si l'on dispose de paramètres d'exécution du problème considéré. Ces paramètres sont relatifs à l'instance du problème et à la

1. http://www.hpctest.org.cn/resources/sim-mpi.tgz

plate-forme retenue pour l'exécution de cette instance. Le principal objectif de ce module étant la détermination rapide du comportement de l'application parallèle du point de vue performance. Cette tâche peut être réduite aux deux principaux objectifs suivants :

- La détermination du temps parallèle associé à un programme MPI donné et ce, pour un contexte donné (instance du problème et liste de machines d'exécution). Le temps déterminé peut être utilisé par un mécanisme adaptatif afin de comparer les temps de plusieurs alternatives de résolution d'un problème donné.
- La détermination des temps d'exécution de chaque processus MPI de l'application considérée. Les entités déterminées (temps) peuvent être utilisées pour voir en détails le comportment de l'application parallèle dans l'objectif de détecter les temps libres (idle times) ou toute autre problème qui peut causer la dégradation de performance de l'application considérée.

Pour accomplir ces tâches, MPI-PERF-SIM utilise les résultats de la phase d'installation (décrite dans la chapitre 4) et qui sont sous forme de fichiers.

3.2 Entrées du module

La figure 21 montre les principales entrées et étapes de ce module de prédiction de performances. Les entrées nécessaires pour ce module sont :

- Le fichier comportant *la liste des machines d'exécution* : ce fichier est utilisé pour établir la correspondance entre processus et processeurs. En effet, le premier processus est exécuté sur le processeur de la première ligne de ce fichier, le second processus est affecté au second processeur du fichier, et ainsi de suite. Si le nombre de processus est supérieur au nombre de processeurs, une autre étape d'affectation est effectuée d'une façon similaire. Ce fichier est fourni par l'utilisateur lors de l'exécution.
- Le fichier comportant *les modèles de communication et les paramètres réseaux associés* : dans ce fichier, on trouve les formules calculant les différents paramètres d'un modèle de communication donné et ceci pour toute communication intra et inter-clusters. Pour un paramètre donné, plusieurs formules sont stockées dans ce fichier où chacune d'elles est adéquate à un intervalle donné de tailles de messages. La structure de ce fichier xml est décrite par la figure 23 pour le cas du modèle de Hockney[51]. En effet, dans ce fichier on trouve les modèles de performance de communication entre deux clusters (*Cluster1 et Cluster2*). Cette performance est exprimée par un champ latence (*Latency*) et plusieurs autres champs comportant les modèles de la bande passante (*Bandwidth*)sur plusieurs intervalles(*Interval*). Le modèle de la bande passante sur un intervalle (entre *Start* et *end*) est décrit par une fonction linéaire de paramètres a et b.

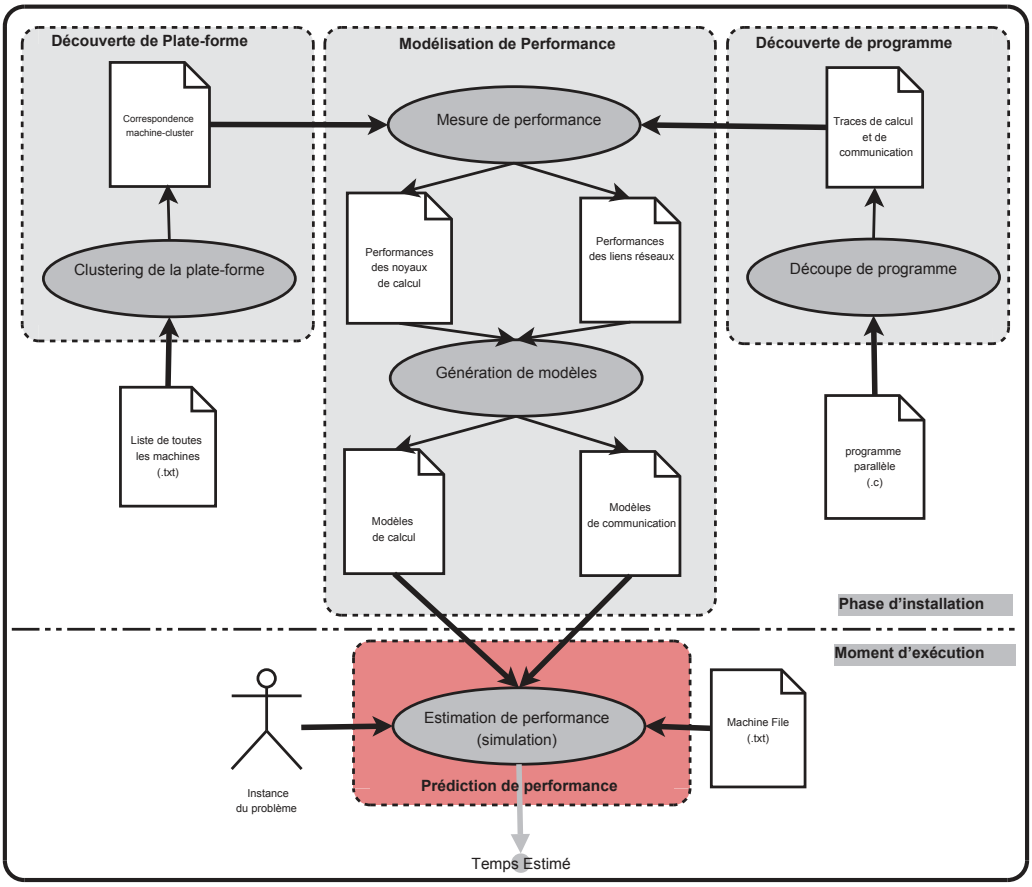


Figure 20 – Framework de prédiction de performance de programmes parallèles

- Le fichier contenant *les modèles de performance des noyaux séquentiels* : on trouve dans ce fichier les valeurs des différents paramètres que comportent les modèles de description de la performance des noyaux séquentiels sur les différentes machines de la plate-forme. Précisons que la description d'un noyau séquentiel donné concerne plusieurs intervalles de taille du problème. La structure de ce fichier xml est décrite par la figure 22. Comme le montre cette figure, chaque champ *SeqKernel* est associé à un noyau donné, le nom de ce noyau est enregistré dans le champ *Name*. Dans Le champ *Cluster*, on trouve les modèles de performance de ce noyau sur les machines de ce cluster, et ce pour plusieurs intervalles. Un intervalle *Interval* est défini par deux valeurs limites (*Start* et *End*), et le modèle de performance dans cet intervalle est de degré *Degree* dont les paramètres sont dans le champ *Params*.
- Le fichier de *correspondance entre machine et clusters* : afin de déterminer l'affiliation d'une machine donnée, ce fichier est consulté. Ce fichier est obtenu avec l'exécution du module de

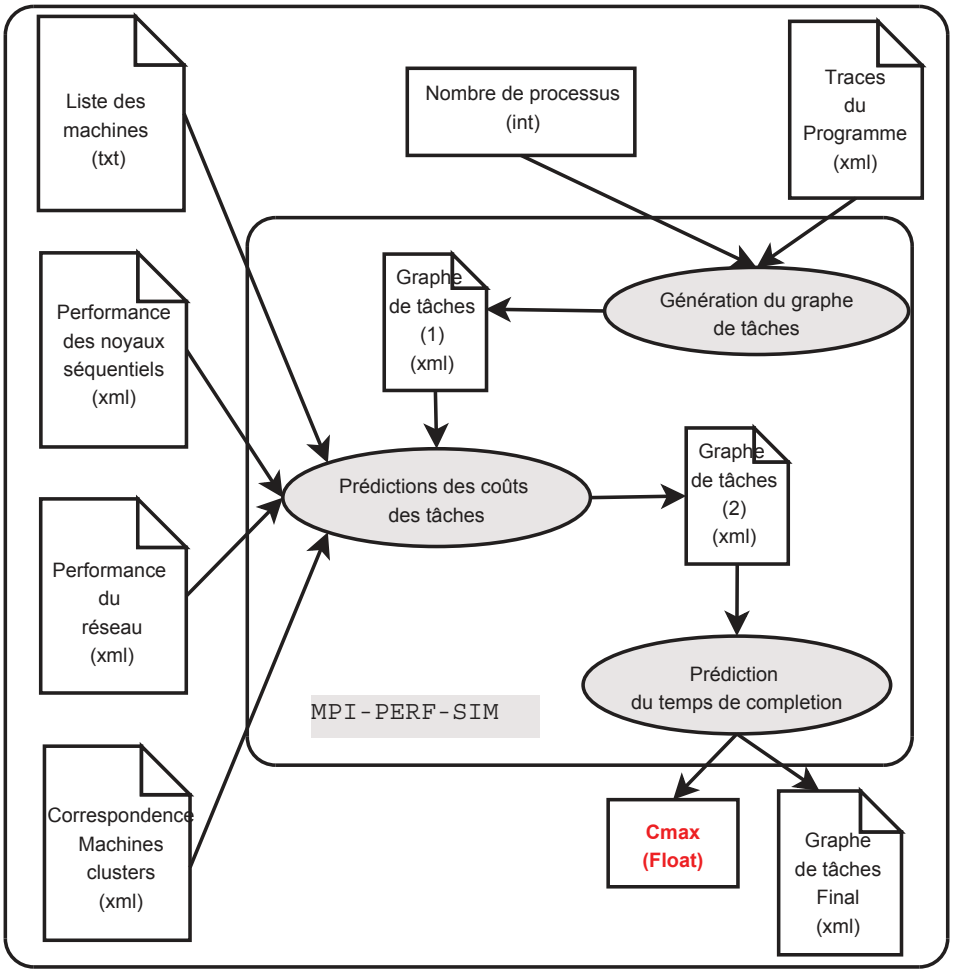


Figure 21 – Architecture de MPI-PERF-SIM

découverte de plate-forme.

- *Le nombre de processus* : Ce nombre est spécifié par l'utilisateur au moment de l'exécution de l'application adaptative.
- *La trace du programme C* : elle comporte la description des différentes tâches (de calcul et de communication) du programme. Ce fichier est obtenu à travers l'exécution du module de découverte de programme.
- *Les paramètres de l'exécution* : ils sont relatifs à l'instance du problème à traiter. Par exemple, pour le cas des problèmes numériques, ces paramètres peuvent consister en les tailles des matrices à traiter (taille du problème).

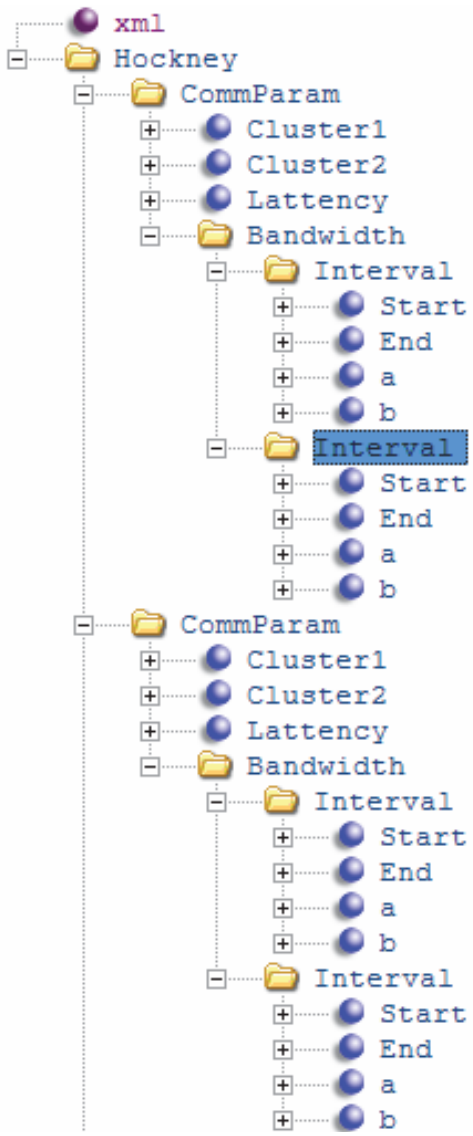


Figure 22 – Performances des réseaux

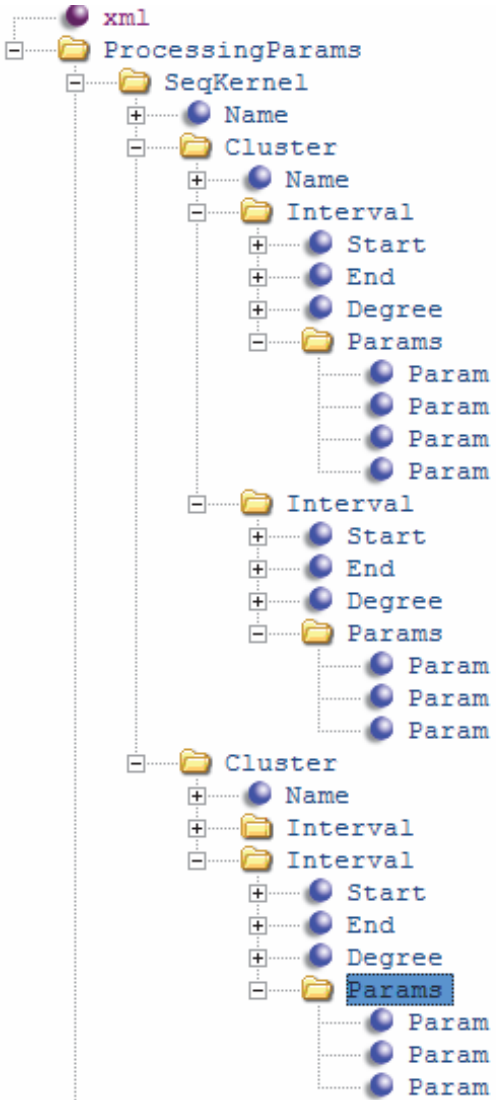


Figure 23 – Performances des noyaux de calcul

3.3 Génération du graphe de tâches

La procédure de génération du graphe de tâches ne peut être abordée que si l'on dispose des paramètres d'exécution spécifiés par l'utilisateur au moment de l'exécution. Ces paramètres sont relatifs au problème (instance) et à la plate-forme choisie pour l'exécution (nombre de processus et fichier de machines). Le but à travers cette génération du graphe de tâches n'est pas la validation du bon comportement du programme MPI et la détection des inter-blockages entre les processus, mais l'estimation du temps d'exécution de ce programme. En d'autres termes, le programme en entrée (représenté par son fichier trace)doit être initialement correct de point de vue logique parallèle. Cette tâche consiste à parcourir le fichier contenant les différentes tâches du programme (fichier trace) et de déterminer pour chaque processus les tâches qui le concernent. Ces tâches diffèrent d'un processus à un autre à cause des structures conditionnelles que comporte le programme MPI considéré.
Pour illustrer le traitement de ce module, considérons l'extrait d'un programme parallèle de la figure 24(a). Les tâches issues de cet extrait sont représentés par la figure 24(b) et qui sont des tâches de calcul (*Comp_func1* et *Comp_func2*) et des tâches de commmunication (*MPI_Send* et *MPI_Recv*). Rappelons que le nombre de processus spécifié par l'utilisateur est égal à trois. Nous remarquons, à partir de cette figure 24, que les tâches de la boucle for ont été sérialisées (tâches T01 et T02). La dépendance entre les tâches de communication pt/pt est spécifiée par le champ *dep*.

3.4 Prédiction des coûts des tâches

3.4.1 Introduction

Pour prédire le temps parallèle d'un programme MPI sur une configuration donnée de la plate-forme d'exécution, on doit tout d'abord estimer les coûts des différentes tâches composant ce programme. On suppose que ces tâches sont non préemptifs, c'est-à-dire, si une tâche est lancée pour être exécutée sur un processeur donné, on ne peut l'interrompre avant qu'elle ne se termine. Ces tâches sont de deux types : des tâches de calcul et des tâches de communication. Les tâches de communication peuvent être elles-mêmes classées en deux types : des communications pt/pt (évoquant deux processus) et des communications multi-processus ou collectives.

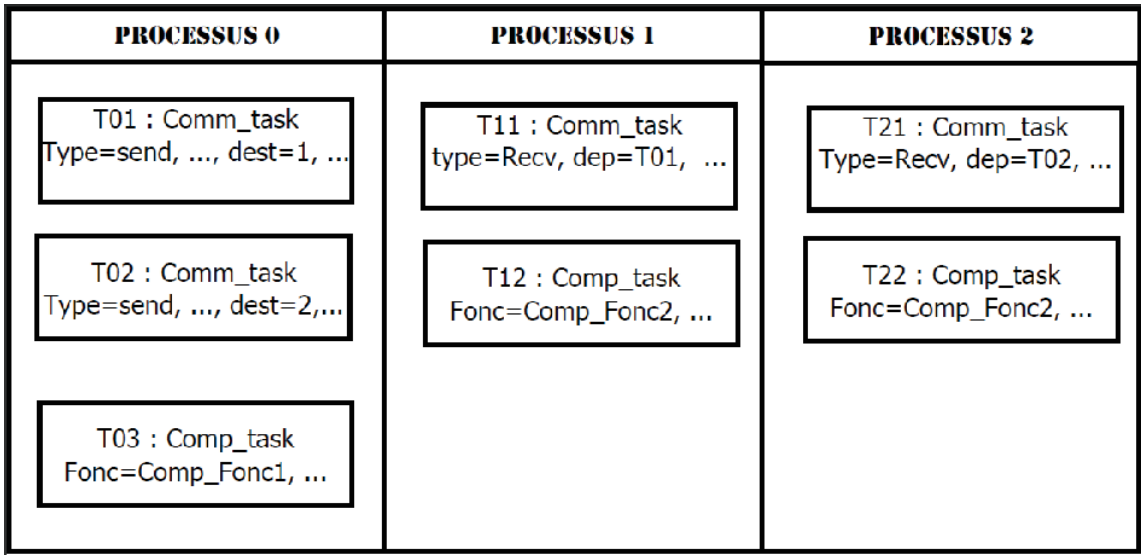
3.4.2 Communications pt/pt

Les communications pt/pt sont de deux types :

- Communications bloquantes : **MPI_Send** pour l'envoi et **MPI_Recv** pour la réception (plus de détails sur ces routines sont dans la partie annexes).

```
if (proc==0)
{
        for (p=1 ; p<procs ;p++)
                MPI_Send(...,p,...) ;
        Comp_fonc1(...) ;
}
else
{
        MPI_Recv(...,0,...) ;
        Comp_fonc2(...) ;
}
```

(a) Extrait d'un programme MPI



(b) Tâches de l'extrait

Figure 24 – Exemple de génération de graphe de tâches

– Communications non bloquantes : **MPI_Isend** pour l'envoi et **MPI_Irecv** pour la réception.

La détermination du coût de l'une des ces routines de communication pt/pt est effectuée selon l'algorithme 6. Considérons une communication pt/pt donnée dont on veut estimer son temps d'exécution. En utilisant les rangs des processus en communication et le fichier de machines d'exécution, les nœuds en communication sont déterminées. En utilisant le fichier de correspondance machine-cluster, les clusters impliqués dans cette communications sont aussi déterminés. Par la suite, la détermination du temps de communication peut être effectuée aisément après la consultation du fichier de performance du réseau d'interconnexion.

Algorithme 6 Prédiction du temps d'une communication pt/pt

Entrées: $Nature, msg_size, msg_type, src, dest, perf_file, hostfile, corr_mach_cluster$
Sorties: est_time

1: Déterminer_les_clusters_en_communication (src,dest, hostfile, corr_mach_cluster, cluster_src, cluster_dest)
2: Déterminer_paramètres_Hockney (cluster_src, cluster_dest, msg_size, msg_type,perf_file,latency,bandwidth)
3: retourner (latency + msg_size * taille(msg_type)/bandwidth)

Pour la validation de ce module de prédiction de performance des routines pt/pt, nous avons considéré les deux types de communication pt/pt (bloquantes et non bloquantes), et nous avons déterminé les modèles de communication pour plusieurs tailles de problèmes choisies d'une façon quasi-aléatoire.

Les figures 25 et26 représentent respectivement les prédictions de performance des routines de communication bloquantes *MPI_Send /MPI_Recv* et les routines de communication non bloquantes (MPI_Isend / MPI_Irecv), et ce pour une communication entre deux nœuds appartenant à deux clusters différents(*genepi* et *sol*). A travers ces prédictions, on remarque que l'erreur relative de prédiction ne dépasse pas les 14 % pour les routines bloquantes et 17% pour les routines non bloquantes. L'erreur moyenne est de l'ordre de 5% (resp. 8%) pour les communications bloquantes (resp. non bloquantes). On remarque aussi que l'erreur de prédiction pour les communications bloquantes diminue avec l'augmentation de la taille du message. Par contre, il n'y a aucune régularité dans les résultats de prédiction de performance des routines non bloquantes.

3.4.3 Communications collectives

Une routine de communication collective ne représente qu'un schéma optimisé pour effectuer une opération de communication multi-processus. On trouve dans la littérature plusieurs schémas pour les différentes routines de communication collectives. Chacun de ces schémas représente la

N(Ko)	*comm pt/pt bloquantes*			*comm pt/pt non bloquantes*		
	M(s)	**E(s)**	**Erreur(%)**	**M(s)**	**E(s)**	**Erreur(%)**
890	0.06824539	0.077800101	14.00	0.043924404	0.048428934	10.26
1452	0.098747109	0.110289107	11.69	0.079031634	0.092743624	17.35
4367	0.247734087	0.263854112	6.51	0.22028205	0.2286759	3.81
6011	0.333842103	0.35140956	5.26	0.235993551	0.266948819	13.12
8383	0.45690542	0.478401292	4.70	0.373106378	0.388186698	4.04
28853	1.501352195	1.534975921	2.24	1.48581822	1.615973733	8.76
34531	1.788608638	1.788910661	0.02	1.765750455	1.932249802	9.43
66666	3.426077317	3.453473568	0.80	2.604228619	2.61940029	0.58
132222	6.763361529	6.848791333	1.26	5.120545944	5.612704899	9.61

Tableau 17 – Temps Mesurés vs estimés pour les communications pt/pt

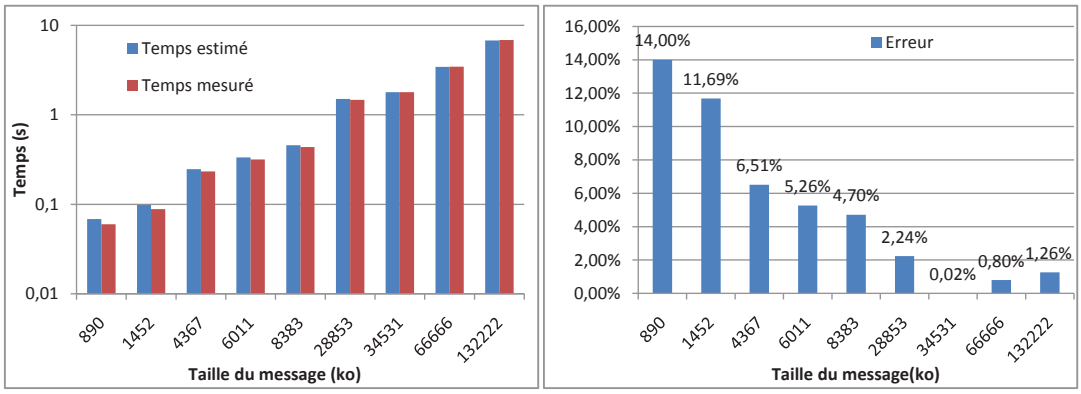


Figure 25 – Prédiction de performance des routines pt/pt bloquantes

façon la plus optimisée pour réaliser l'opération collective et ce pour un contexte particulier en termes du nombre de processus impliqués et en termes de tailles de message à communiquer. Les implémentations MPI ont bien profité de ces schémas pour fournir des routines bien optimisées. Autrement dit, ces routines utilisent une approche adaptative dans leurs codes, à savoir l'approche poly-algorithmique. A titre d'exemple, les pseudo-codes des figures 27 et 28 décrivent cette approche poly-algorithmique pour, respectivement, les routines *MPI_Bcast* et *MPI_Allgather* de la bibliothèque MPICH2.

La prédiction du temps de communication d'une routine de communication collective donnée ne peut pas être dissociée de son implémentation, vu que le comportement ainsi que les schémas adoptés ne sont pas les mêmes d'une implémentation MPI à une autre.
Pour déterminer le coût d'une communication collective sur l'ensemble des processus participant

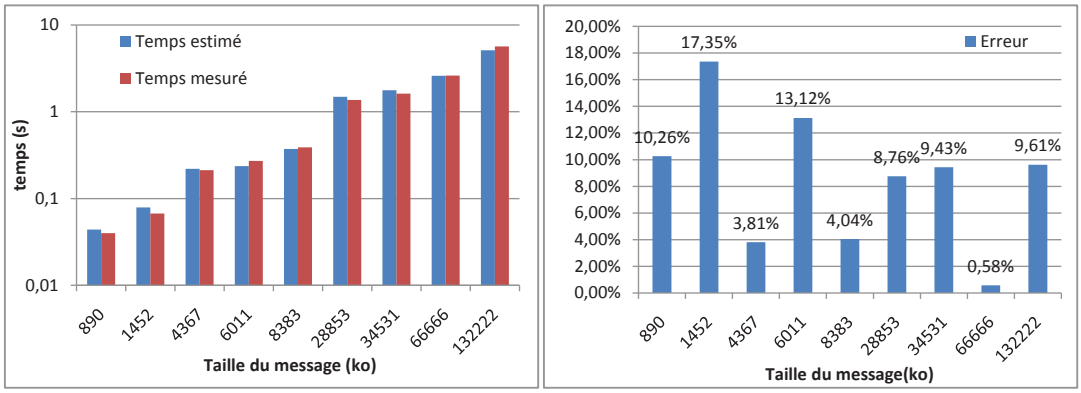


Figure 26 – Prédiction de performance des routines pt/pt non bloquantes

```
If ((msg_size<BCAST_SHORT_MSG)OR(comm_size<MIN_PROCS))Then
     Algorithm : Binomial_Bcast(…)
Else
     If((msg_size< BCAST_LONG_MSG)AND(is_pof2(comm_size))Then
          Algorithm : Bcast_Scatter_Doubling_Allgather(…)
     Else
          Algorithm : Bcast_Scatter_Ring_Allgather(…)
     EndIf
EndIf
```

Figure 27 – Poly-algorithme de la routine MPI_Bcast

```
If ((msg_size<AllGATHER_SHORT_MSG)OR(is_pof2(comm_size)=FALSE)Then
     Algorithm : Bruck_Algo(…)
Else
     If((msg_size< ALLGATHER_LONG_MSG)AND(is_pof2(comm_size))Then
          Algorithm : Recursive_Doubling_Algo(…)
     Else
          Algorithm : Ring_Algo(…)
     EndIf
EndIf
```

Figure 28 – Poly-algorithme de la routine MPI_Allgather

dans cette communication, une étape d'émulation de cette routine est effectuée. En effet, pour une routine de communication collective donnée, l'émulation est effectuée à travers un code C qui émule le code de cette routine. Le principal objectif de ce code étant de déterminer le temps d'exécution de la routine collective, et ce pour les différents processus participant dans une telle

opération. Pour ce faire, notre approche consiste à modifier les codes de ces routines de sorte à changer les appels aux routines de communication pt/pt composant cette routine en des appels au module de prédiction de performance des routines pt/pt [2]. La figure 29 illustre le comportement de notre émulateur de performance pour les routines de communications collectives.

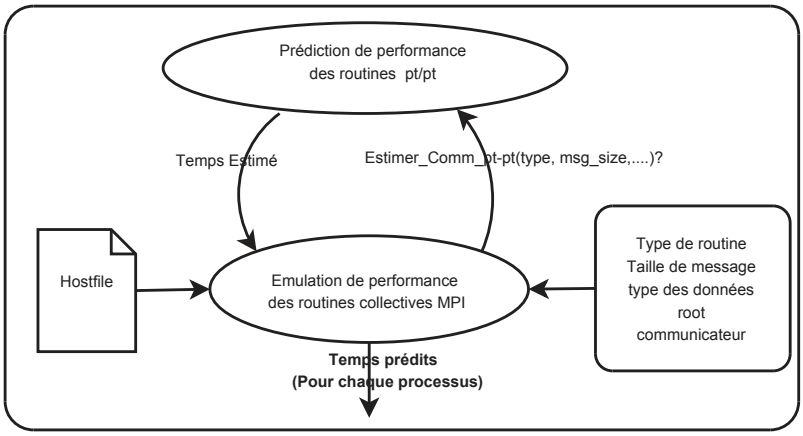


Figure 29 – Emulateur de performance des routines collectives

Pour la validation de ce module de prédiction de performance, nous avons choisi trois routines très utilisées par les programmes parallèles, à savoir *MPI_Bcast*, *MPI_Scatter* et *MPI_Gather*. Cette validation a été effectuée dans un environnement hétérogène composé de plusieurs machines affiliées à cinq clusters différents de la plate-forme Grid'5000. Les détails sur ces machines sont récapitulés dans le tableau 18. Les mesures de performance ainsi que les prédictions de performance de chacune de ces routines ont été élaborées pour six configurations différentes composées respectivement par 16, 32, 49, 64, 81 et 128 processeurs. La répartition des processeurs pour chacune de ces configurations est représentée par le tableau 19.

Les résultats de cette validation sont représentés par les figures 30,31 et 32 et qui correspondent respectivement aux routines *MPI_Bcast*, *MPI_Scatter* et *MPI_Gather*. Dans ces schémas, les temps prédits par notre module de prédiction de performance sont représentés par des barres de couleur bleue, les temps d'exécution réels de ces routines sont représentés par des barres en rouge, l'erreur relative est représentée par une courbe verte.

– La figure 30 montre que l'erreur relative varie entre 0% et 87% pour toutes les expérimentations de la routine *MPI_Bcast*. Mais, l'erreur moyenne de ces expérimentations est de l'ordre de 10%. L'erreur moyenne par configuration varie entre 6% et 23%.

Cluster	Site	Caractéristiques
Chuque	Lille	cputype : AMD opteron, cpufreq : 2.2Ghz, nodecpu : 2, cpucore : 2, memnode : 4096Mo
Genepi	Grenoble	cputype : Intel Xeon, cpufreq : 2.5Ghz, nodecpu : 2, cpucore : 8, memnode : 8192Mo
Helios	Sophia	cputype : AMD Opteron, cpufreq : 2.2Ghz, nodecpu : 2, cpucore : 4, memnode : 4096Mo
Sol	Sophia	cputype : AMD Opteron, cpufreq : 2.6Ghz, nodecpu : 2, cpucore : 4, memnode : 4096Mo
Paramount	Rennes	cputype : Intel Xeon, cpufreq : 2.33Ghz, nodecpu : 2, cpucore : 4, memnode : 8192Mo

Tableau 18 – Caractéristiques des machines d'expérimentation

Configuration \ Cluster	Genepi	Paramount	Helios	Sol	Chuque	Total nœuds
Hetero16	3	4	3	3	3	16
Hetero32	6	8	7	5	6	32
Hetero49	10	10	10	10	9	49
Hetero64	13	16	13	12	10	64
Hetero81	15	20	15	16	15	81
Hetero128	20	25	20	28	35	128

Tableau 19 – Configurations expérimentales : nombre de processeurs de chaque cluster

- A partir de la figure 31, on peut remarquer que l'erreur relative pour l'ensemble des tests de la routine *MPI_Scatter* varie entre 1% et 31% , mais avec une erreur moyenne de l'ordre de 11%. L'erreur moyenne par configuration varie entre 5% et 15%.

- L'erreur d'estimation de la performance de la routine *MPI_Gather*, représentée par la figure 32, varie entre 1% et 35% et avec une erreur moyenne de 9%. L'erreur moyenne par configuration varie entre 5% et 18%.

En résumé, malgré la valeur élevée pour certains cas, cette erreur reste raisonnable vu le contexte de prédiction de performance (mode hors ligne) qui ne considère pas plusieurs facteurs tel que la congestion réseau.

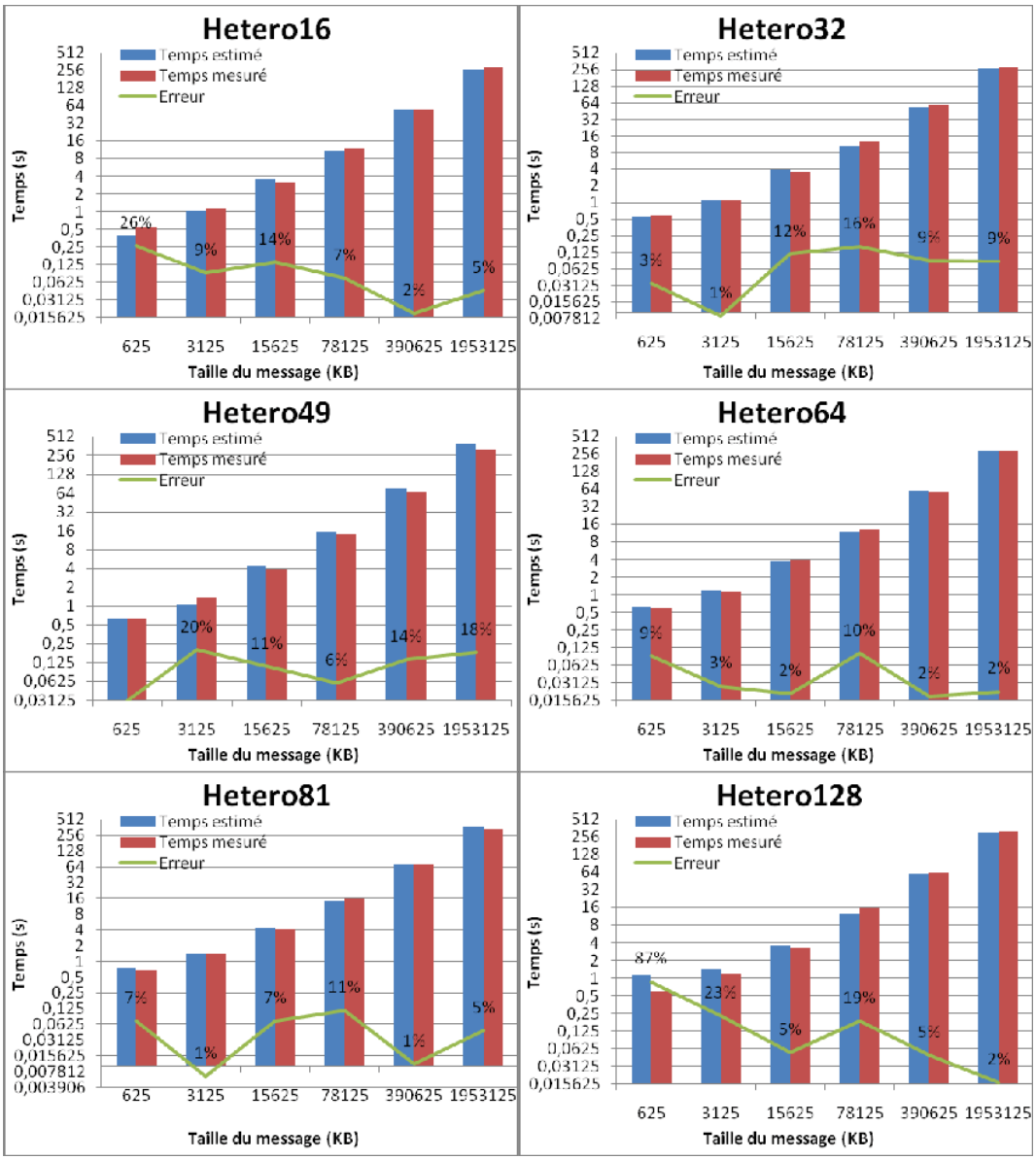


Figure 30 – Prédiction de performance de la routine MPI_Bcast

3.5 Prédiction du temps de complétion

La simulation du comportment d'un programme parallèle donné est un moyen permettant de déterminer son temps d'exécution. En effet, dans un programme parallèle, le temps d'exécution n'est pas constitué seulement de calcul et de communications, mais il comporte aussi des temps

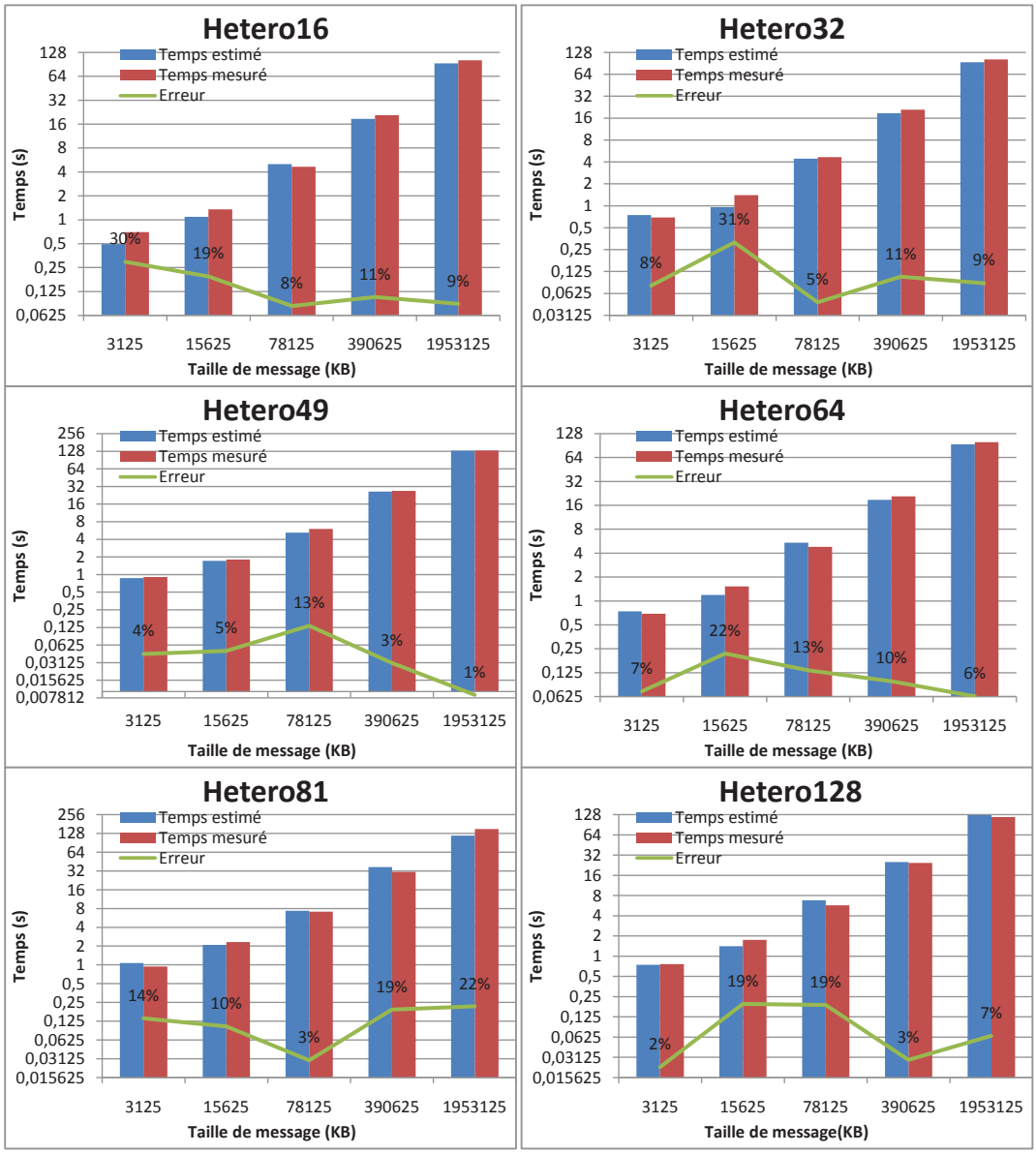


Figure 31 – Prédiction de performance de la routine MPI_Scatter

libres qui sont dues aux attentes générés par les communications. Ces attentes peuvent être déterminées en simulant simultanément le comportement des différents processus.

L'algorithme 7 donne une idée sur notre solution de simulation du comportment d'un programme MPI représenté par ses traces de calculs et de communications. Notre procédure de simulation commence avec la génération du graphe de tâches initial comportant les tâches de p processus

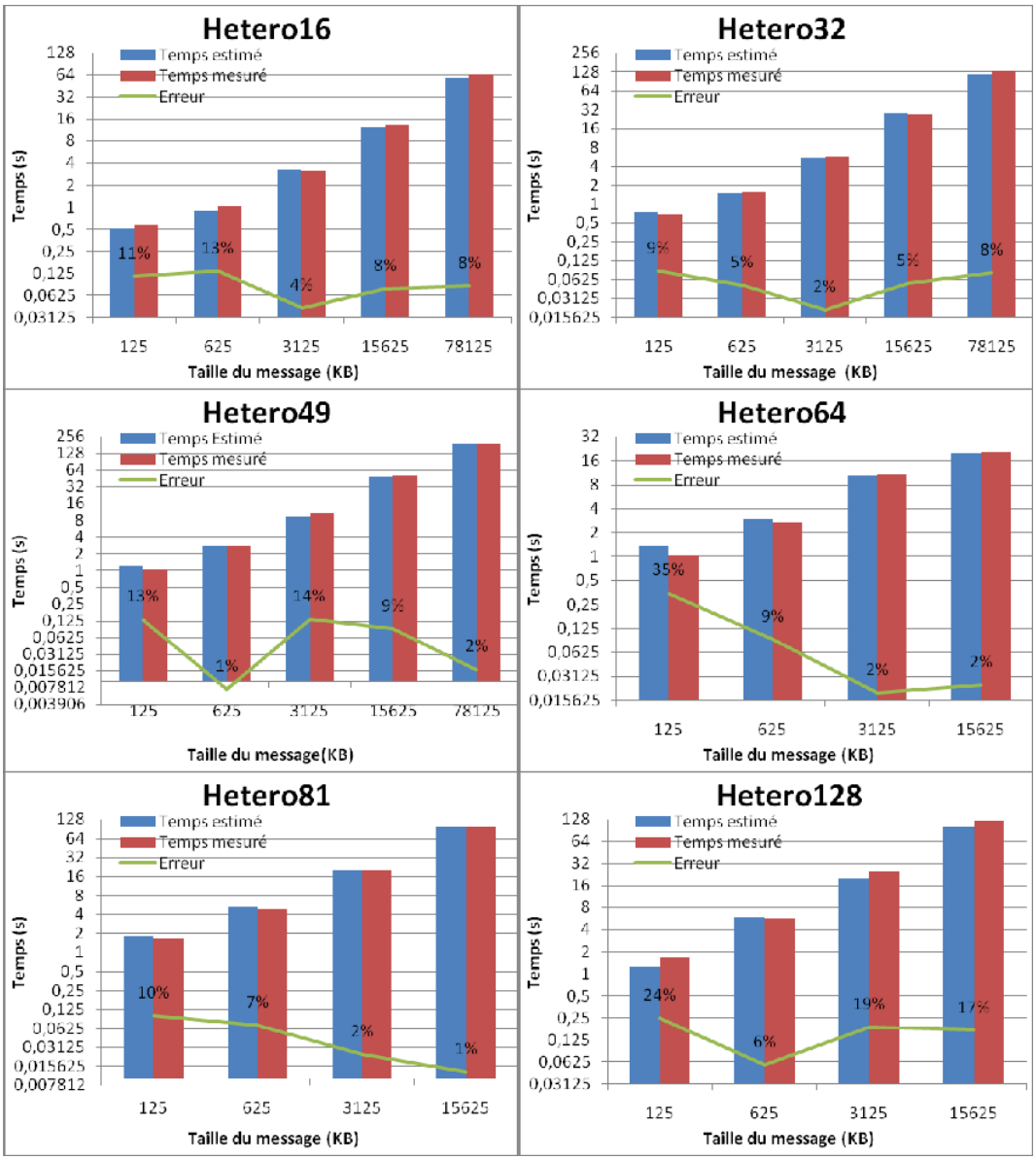


Figure 32 – Prédiction de performance de la routine MPI_Gather

(paramètre fourni par l'utilisateur). Une boucle sur les processus et leurs tâches est ensuite parcourue afin de déterminer pour chacune de ces tâches le temps de début et le temps de fin de cette dernière. Dans cette boucle on fait appel à une fonction récursive (*Schedule_Tasks*) pour déterminer les temps de début et de fin d'une tâche donnée. Cette fonction (voir algorithme 8) permet de gérer les dépendances entre les tâches des différents processus surtout particulièrement

entre les tâches de communication.

Algorithme 7 Simulation du comportement d'un programme parallèle

Entrées: $Fichiertrace, nombredeprocessus, Fichierdemachines, ...$
Sorties: $GraphedeT\hat{a}ches$//Graphe de tâches avec les temps de début et de fin

1: $Génerer_Graphe_T\hat{a}ches(Task, max_indices)$ //$Task$ est une matrice de tâches, $max_indices$ est un tableau contenant le nombre de tâches de chaque processus
2: **Répéter**
3: $proc \leftarrow Début(indices, max_indices)$ //$indices$ est un tableau contenant les compteurs de simulation de chaque processus, $proc$ est le processus à simuler maintenant
4: $deb \leftarrow indices[proc]$//tâche de début de simulation pour le processus $proc$
5: $fin \leftarrow max_indices[proc]$//tâche de fin de simulation pour le processus $proc$
6: $Schedule_Tasks(Task, proc, indices, max_indices, deb, fin)$
7: **Jusqu'à** $(Fin_taches(indices, max_indices))$
8: $Sauvegarder_info_taches(task, max_indices, GT_Final)$

Algorithme 8 Procédure d'ordonnancement de tâches (Schedule_Tasks)

Entrées: $Graphedet\hat{a}ches$
Sorties: $Graphedet\hat{a}chescomplété$

1: $cpt \leftarrow deb$
2: **Tant que** (cpt<=fin) **Faire**
3: **Si** $Task[cpt].type = comp$ **Alors**
4: **Si** (cpt=1) **Alors**
5: $Task[proc][cpt].Start \leftarrow 0$
6: **Sinon**
7: $Task[proc][cpt].Start \leftarrow Task[proc][cpt-1].End$
8: **FinSi**
9: $Task[proc][cpt].End \leftarrow Task[cpt].Start + Déterminer_Cout_tache(...)$
10: **FinSi**
11: **Si** $(Task[cpt].type = Comm_Send)$ **Alors**
12: $dep \leftarrow Task[proc][cpt].params.dep$
13: $src \leftarrow Task[proc][cpt].params.src$
14: **Si** (indices[src]<dep-1) **Alors**
15: $Schedule_Tasks(Task, src, indices, max_indices, indices[src], dep-1)$
16: $Task[cpt].Start \leftarrow max(Task[proc][cpt-1].End, Task[src][dep-1].End$
17: **FinSi**
18: **FinSi**
19: $cpt \leftarrow cpt + 1$
20: $indices[proc] \leftarrow cpt$
21: **FinTantque**

4 Application de MPI-PERF-SIM

4.1 Application cible

Afin de valider MPI-PERF-SIM, nous avons choisi de comparer les temps estimés avec les temps réels et ce, pour une application parallèle comportant des tâches de calcul et des tâches de communication. L'application choisie pour cette validation est relative au traitement de la multiplication matricielle parallèle(MM) selon le schéma de l'algorithme de Fox [40] communément connu sous le nom BMR(*Broadcast Multiply Roll*). Le pseudo-code de cette application est représenté par l'algorithme 9.

Algorithme 9 Multiplication Matricielle parallèle

Entrées: A, B *//les matrices A et B initialement sur le processus 0*
Sorties: C=A*B *//La matrice C sur le processus 0*
*//**Etape 1 :** Distribution des matrices*
1: **Pour** $p = 0$,$procs - 1$ **Faire**
2: //*procs* est le nombre de processus
3: Envoyer un bloc de lignes de la matrice A vers le processus p
4: Envoyer un bloc de colonnes de la matrice B vers le processus p
5: //Ces deux blocs sont sauvegardés dans localA et localB
6: **FinPour**
*//**Etape 2 :** Produit Matriciel*
7: **Pour** $step = 0$,$procs - 1$ **in parallel Faire**
8: cblas_dgemm(localA, localB, localC,. . .)
9: Rotation Circulaire vers le haut de localA
10: **FinPour**
*//**Etape 3 :** Collecte des blocs de la matrice C*
11: **Pour** $p = 0$, $procs - 1$ **Faire**
12: Recevoir de chaque processus p un bloc de colonnes de la matrice C
13: **FinPour**

4.2 Plate-forme de validation

La plate-forme de test est composée d'un ensemble de machines de Grid'5000 affiliées à quatre clusters de trois sites différents (voir tableau 20). Les expérimentations ont été traitées sur trois configurations matérielles différentes. Ces configurations sont composées par respectivement 9, 15 et 27 processeurs. La répartition de ces nœuds pour chacune de ces configurations sur les quatre clusters considérés est représentée par le tableau 21.

Cluster	Site	Caractéristiques des nœuds
Chti	Lille	cputype : opteron, nodecpu : 2, cpucore : 1, memnode : 4096Mo
Genepi	Grenoble	cputype : xeon-harpertown, nodecpu : 2, cpucore : 4, memnode : 8192Mo
Griffon	Nancy	cputype : Intel Xeon, nodecpu : 2, cpucore : 4, memnode : 16384Mo
Grelon	Nancy	cputype : Intel Xeon, nodecpu : 2, cpucore : 2, memnode : 2048Mo

Tableau 20 – Caractéristiques et affiliations des nœuds d'expérimentation

	Chti	Genepi	Griffon	Grelon	Total nœuds
Config1	3	2	3	1	9
Config2	4	5	3	3	15
Config3	7	7	7	6	27

Tableau 21 – Nombre de nœuds pour chaque configuration

4.3 Résultats expérimentaux

Les expérimentations ont été menées pour différentes tailles du problème (taille des matrices) variant entre 5000 et 17000 avec un pas de 3000. Les résultats sont résumés dans le tableau 22. Ce tableau contient les temps estimés (**E**) par MPI-PERF-SIM et les temps mesurés réellement (**M**) pour les différentes configurations et tailles. Ces temps sont mesurés en secondes. Le taux d'erreur dans ce tableau représente l'erreur relative entre les deux temps. La figure 33 représente ces erreurs pour les différentes configurations. A partir de cette figure, nous constatons que l'erreur de prédiction de performance de MPI-PERF-SIM pour cette application varie entre 0% et 17%. L'erreur moyenne de l'ensemble de ces expérimentations est de l'ordre de 5%. L'erreur moyenne par configuration varie entre 3% pour Config1 et 6% pour Config2.

N	***Config1***			***Config2***			***Config3***		
	M	**E**	**Erreur**	**M**	**E**	**Erreur**	**M**	**E**	**Erreur**
5000	31.94	29.17	8.67	26.53	21.93	17.34	25.91	23.33	9.96
8000	104.15	107.25	2.98	77.56	72.36	6.70	60.70	60.94	0.40
11000	260.96	261.82	0.33	174.99	176.04	0.60	126.63	122.59	3.19
14000	538.23	518.16	3.73	362.16	345.04	4.73	267.86	258.40	3.53
17000	891.21	863.36	3.12	609.81	581.07	4.71	418.37	400.63	4.24

Tableau 22 – Erreur de prédiction du comportement de la MM parallèle

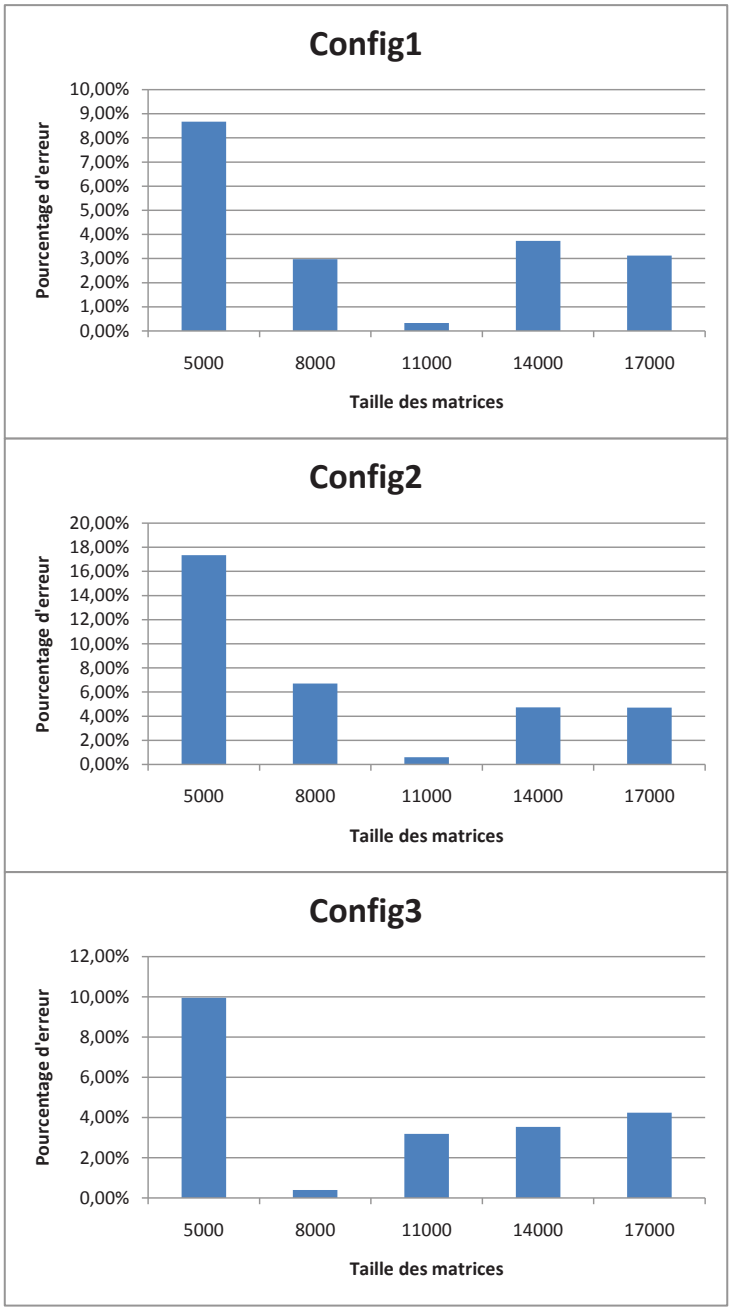


Figure 33 – Erreur de prédiction de MPI_PERF_SIM pour le produit matriciel

5 Conclusion

La prédiction de performance d'une application parallèle sur une plate-forme parallèle hétérogène est une tâche difficile surtout si elle est effectuée en mode hors ligne (off-line). Cette difficulté est due à plusieurs facteurs, les plus pertinents sont l'imprécision des modèles utilisés et la non considération de plusieurs paramètres (pouvant varier au cours de l'exécution) telles que la contention des liens réseau et l'imprécision des modèles de performance qui est très nette pour le cas des routines de communication pt/pt non bloquantes et des routines collectives. Dans ce chapitre, nous avons présenté le module MPI-PERF-SIM permettant de simuler le comportement (du point de vue temps d'exécution) d'une application parallèle sur une architecture parallèle hiérarchique. Pour accomplir sa tâche, ce simulateur se base sur les paramètres fournis par l'utilisateur ainsi que les fichiers générés par la phase d'installation de l'application parallèle. L'application de ce module sur le problème de MM a donné des résultats satisfaisants malgré la complexité de l'environnement hétérogène considéré.

Conclusion générale et perspectives

La programmation sur les plates-formes parallèles actuelles, notamment les plates-formes hétérogènes et hiérarchiques, constitue un défi pour les programmeurs. Cela s'explique par le nombre important de facteurs agissant sur la performance des programmes parallèles produits. En conséquence, l'adoption de l'ancienne technique de programmation consistant à produire des versions optimisées pour les différents types et versions de machines parallèles ne peut plus résoudre le problème de la programmation sur ces plates-formes d'exécution.

Par analogie avec le contexte séquentiel, l'approche adaptative qui consiste à considérer plusieurs variantes algorithmiques résolvant le même problème afin de choisir celle qui donne la performance maximale pour un contexte d'exécution donné, représente la solution pour une programmation efficace sur les plates-formes parallèles actuelles. Pour la mise en oeuvre d'une approche adaptative donnée, des techniques de prédiction de performance doivent être utilisées. Le choix de l'adoption d'une technique de modélisation de performance particulière doit obéir aux contraintes imposées par le domaine adaptatif. Ces contraintes sont relatives à la précision des prédictions et la rapidité du processus permettant d'aboutir à ces prédictions.

Nous avons commencé dans ce travail par proposer un framework de développement d'applications parallèles adaptatives. Ce framework a été basé, dans une première étape, sur la modélisation théorique de performance. Il a été validé à travers des tests effectués sur un noyau important dans les applications scientifiques, à savoir la multiplication matricielle pour le cas d'environnements hétérogènes. Malgré la rapidité du processus de prédiction dans ce framework, nous avons constaté que l'utilisation des modèles théoriques ne peut satisfaire la contrainte de précision surtout pour le cas d'architectures et applications complexes.

Pour remédier aux insuffisances rencontrées avec la modélisation analytique, nous nous sommes concentrés sur le problème de prédiction de performance et avons conçu un framework automatisant cette tâche. Notre solution de prédiction de performance pour une application donnée est basée sur la combinaison des différentes techniques de modélisation (théorique, empirique et simulation). En effet, notre framework se base sur un module (MPI-PERF-SIM) de simulation rapide

et hors-ligne du comportement de l'application considérée. Une telle simulation se base principalement sur la consultation des traces de l'application et des modèles de performance relatifs aux communications pt/pt et aux noyaux de calcul de cette application. Les modèles théoriques utilisés par nos simulations sont générés à travers une méthode statistique (la régression) appliquée sur plusieurs mesures réelles (expérimentations) de performance. Ces expérimentations sont effectuées au moment de l'installation de l'application adaptative et concernent les noyaux de calcul de l'application et les routines de communication MPI pt/pt.

En guise de récapitulation, nous pouvons dire que les principales contributions de cette thèse concernent les aspects suivants :

- *L'intégration de techniques de modélisation de performance avec des techniques adaptatives.*
- *La détermination de la structure de la plate-forme* : en partant de la liste des machines de la plate-forme et en utilisant un algorithme de clustering, nous obtenons la correspondance entre machine et cluster, ce qui simplifie les traitements dans plusieurs modules de notre framework.
- *La détermination des traces de calcul et de communication* : en se basant sur la technique de découpe de programmes, nous déterminons les différentes tâches composant un programme écrit en langage C et utilisant la bibliothèque de communication MPI.
- *La modélisation des noyaux de calcul* : en développant une méthode de régression polynomiale permettant de déterminer d'une façon automatique le polynôme qui représente le mieux le comportement d'un noyau de calcul considéré. Notre régression permet également de fournir plusieurs modèles polynomiaux pour le même noyau de calcul et pour la même machine. L'utilisation d'une telle modélisation pour un noyau de calcul permet d'aboutir à des erreurs de prédiction de performance minimes pour toute taille du problème.
- *La modélisation des routines pt/pt* : en se basant sur des expérimentations entre chaque couple de clusters et entre deux machines de chaque cluster, on est capable de générer, à l'aide d'une technique de régression de données, les modèles théoriques relatifs aux communications pt/pt de la bibliothèque MPI intra et inter-clusters. Ces modèles sont définis sur plusieurs intervalles des tailles de messages et définis comme des fonctions linéaires sur les différents intervalles. Cette stratégie de modélisation est assez précise pour des contextes stables.
- *L'émulation du comportement des routines collectives* : la prédiction de performance d'une routine collective est effectuée dans notre framework au moment de l'exécution (ou au moment où on connaît les paramètres d'exécution). Cette prédiction est basée sur des codes émulant le comportement de ces routines et sur les modèles de performances des routines MPI pt/pt.
- *La simulation rapide du comportement de l'application parallèle* : une fois les paramètres d'exécution connus, le graphe de tâches de l'application peut être établi et la simulation du

comportement de chaque processus peut être effectuée. Cette simulation a pour principal objectif la détermination des temps de début et de fin de chaque tâche. La précision des prédictions fournies par notre simulateur dépend non seulement de la précision de nos modèles correspondant aux noyaux séquentiels et aux routines MPI pt/pt, mais aussi des facteurs en ligne qui ne peuvent pas être considérés dans notre solution.

Il faut toutefois ajouter que notre solution peut être améliorée par plusieurs mesures. Nous en citons les suivantes comme perspectives futures :

- La considération de modèles supportant la modélisation des contentions des liens réseaux et ce, en utilisant le modèle LoPC [41] ou LoGPC [67] afin d'améliorer la précision des prédictions relatives aux routines de communications collectives.
- La considération des changements sur la plate-forme en mettant à jour les fichiers comportant la description des performances de la plate-forme.
- La modélisation des parties de calcul adoptant OpenMP en considérant les travaux de Chapman et al. [25, 63] et Adhianto [4].
- La considération des noyaux CUDA dans les différents modules de notre framework de prédiction de performance.
- L'adoption du modèle LogGPO [26] pour améliorer la prédiction des routines pt/pt non bloquantes.

Bibliographie

[1] S. Achour, M. Ammar, B. Khmili, and W. Nasri. MPI-PERF-SIM : Towards an Automatic Performance Prediction Tool of MPI Programs on Hierarchical Clusters. In *19th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'11), Ayia Napa, Cyprus.*, 2011.

[2] S. Achour and W. Nasri. A Performance Prediction Approach for MPI Routines on Multi-clusters. In *20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'12), Munich, Germany,*. IEEE Computer Society, 2012.

[3] S. Achour, W. Nasri, and L. A. Steffenel. On the Use of Performance Models for Adaptive Algorithm Selection on Heterogeneous Clusters. In *17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'09), Weimar, Germany*, 2009.

[4] L. Adhianto. *A New Framework for Analyzing, Modeling and Optimizing Mpi and/or Openmp Applications*. PhD thesis, University of Houston, Houston, TX, USA, 2007.

[5] P. Alberti, P. Alonso, A. M. Vidal, J. Cuenca, and D. Gimenez. Designing Polylibraries to Speed up Linear Algebra Computations. *International Journal of High Performance Computing and Networking*, 1 :75–84, 2004.

[6] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP : Incorporating Long Messages into the LogP Model — One step closer towards a realistic model for parallel computation. Technical report, University of California at Santa Barbara, Santa Barbara, CA, USA, 1995.

[7] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home : An Experiment in Public-resource Computing. *ACM Communication*, 45 :56–61, 2002.

[8] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammerling, J. Demmel, C. Bischof, and D. Sorensen. LAPACK : a Portable Linear Algebra Library for High-performance Computers. In *ACM/IEEE Conference on Supercomputing*, New York, USA, 1990.

[9] C. W. Antoine, A. Petitet, and J. J. Dongarra. Automated Empirical Optimization of Software and the ATLAS Project. *Parallel Computing*, 27 :3–35, 2000.

[10] R. Aversa, B. Di Martino, M. Rak, S. Venticinque, and U. Villano. Performance Prediction Through Simulation of a Hybrid MPI/OpenMP Application. *Parallel Computing*, 31 :1013–1033, 2005.

[11] R. Bagrodia, E. Deelman, and T. Phan. Parallel Simulation of Large-Scale Parallel Applications. *International Journal of High Performance Computing and Applications*, 15 :3–12, 2001.

[12] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks : Summary and Preliminary Results. In *ACM/IEEE Conference on Supercomputing*, 1991.

[13] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc Users Manual. Technical report, Argonne National Laboratory, 2010.

[14] W. Barth. *Nagios. System and Network Monitoring.* No Starch Press, 2006.

[15] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. Matrix-Matrix Multiplication on Heterogeneous Platforms. In *International Conference on Parallel Processing*, Toronto, Canada, 2000.

[16] A. Beguelin, J. J. Dongarra, A. Geist, R. Manchek, and V. Sunderam. A User's Guide to PVM Parallel Virtual Machine. Technical report, University of Tennessee, Knoxville, TN, USA, 1991.

[17] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. Adaptive Computing on the Grid Using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14 :369–382, 2003.

[18] J. Bilmes, K. Asanovic, J. Demmel, and C. Chin. Optimizing Matrix Multiply using PHiPAC : a Portable, High-Performance, ANSI C Coding Methodology. Technical report, University of Tennessee, Knoxville, TN, USA, 1996.

[19] R. D. Blumofe and C. E. Leiserson. Scheduling Multithreaded Computations by Work Stealing. *ACM Journal*, 46(5) :720–748, 1999.

[20] M. Cafaro and G. Aloisio. *Grids, Clouds and Virtualization*. Computer Communications and Networks. Springer, 2010.

[21] L. E. Cannon. *A Cellular Computer to Implement the Kalman Filter Algorithm*. PhD thesis, Montana State University, Bozeman, MT, USA, 1969.

[22] L. Carrington, A. Snavely, X. Gao, and N. Wolter. A Performance Prediction Framework for Scientific Applications. In *ICCS Workshop on Performance Modeling and Analysis (PMA03)*, Melbourne, Australia, 2003.

[23] H. Casanova, A. Legrand, and M. Quinson. Simgrid : A generic framework for large-scale distributed experiments. In *10th EUROS/UKSim International Conference on Computer Modelling and Simulation,*, pages 126–131, Cambridge University, Emmanuel College, Cambridge, UK, 2008.

[24] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[25] B. Chapman, G. Jost, and R. v. d. Pas. *Using OpenMP : Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press, 2007.

[26] W. Chen, J. Zhai, J. Zhang, and W. Zheng. LogGPO : An Accurate Communication Model for Performance Prediction of MPI Programs. *Science in China Series F : Information Sciences*, 52 :1785–1791, 2009.

[27] Z. Chen, J. J. Dongarra, P. Luszczek, and K. Roche. Self Adapting Software for Numerical Linear Algebra and LAPACK for Clusters. *Parallel Computing*, 29 :1723–1743, 2003.

[28] P.-N. Clauss, M. Stillwell, S. Genaud, F. Suter, H. Casanova, and M. Quinson. Single Node On-Line Simulation of MPI Applications with SMPI. Research Report RR-7426, INRIA, 2010.

[29] J. Cuencal, D. Giminez, J. Gonzalez, J. J. Dongarra, and K. Roche. Automatic Optimisation of Parallel Linear Algebra Routines in Systems with Variable Load. In *11th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'03), Genova, Italy*, 2003.

[30] D. Culler, R. Karp, D. Patterson, A. Sahay, R. Subramonian, and T. V. Eicken. LogP : Towards a realistic model of parallel computation. In *4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, New Orleans, LA, USA, 1993.

[31] V.-D. Cung, V. Danjean, J.-G. Dumas, T. Gautier, G. Huard, B. Raffin, C. Rapine, J.-L. Roch, and D. Trystram. Adaptive and Hybrid Algorithms : Classification and Illustration on Triangular System Solving. In J.-G. Dumas, editor, *Transgressive Computing*, Grenade, Espagne, 2006.

[32] DEC. The Whetstone Performance. Technical report, Digital Equipment Corporation, 1986.

[33] M. den Burger, T. Kielmann, and H. E. Bal. TOPOMON : A Monitoring Tool for Grid Network Topology. In *International Conference on Computational Science*, Amsterdam, The Netherlands, 2002.

[34] P. A. Dinda, T. Gross, R. Karrer, B. Lowekamp, N. Miller, P. Steenkiste, and D. Sutherland. The Architecture of the Remos System. In *10th IEEE International Symposium on High Performance Distributed Computing*, Washington, DC, USA, 2001.

[35] J. J. Dongarra. Performance of Various Computers Using Standard Linear Equations Software. *Sigarch Computer Architecture News*, 20 :22–44, 1992.

[36] J. J. Dongarra, L. S. Blackford, J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK User's Guide*. Society for Industrial and Applied Mathematics, 1997.

[37] J. J. Dongarra, T. Sterling, H. Simon, and E. Strohmaier. High Performance Computing Clusters, Constellations, MPPs, and Future Directions. *Computing in Science and Engineering*, 7 :51– 59, 2005.

[38] J. J. Dongarra and R. C. Whaley. A User's Guide to the BLACS v1.1. Technical report, LAPACK Working Note, 1997.

[39] M. J. Flynn. Very High-speed Computing Systems. *Proceedings of The IEEE*, 54 :1901–1909, 1966.

[40] G. Fox, S. W. Otto, and A. J. G. Hey. Matrix Algorithms on a Hypercube I : Matrix Multiplication. *Parallel Computing*, 4 :17–31, 1987.

[41] M. Frank, A. Agarwal, and M. K. Vernon. LoPC : Modeling Contention in Parallel Algorithms. In *6th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPOPP), Las Vegas, Nevada, USA*, pages 276–287, 1997.

[42] M. Frigo and S. G. Johnson. FFTW : An Adaptive Software Architecture For The FFT. In *International Conference on Acoustics, Speech and Signal Processing*, pages 1381–1384, Seattle, Washington, USA, 1998.

[43] M. Frigo and S. G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93 :216–231, 2005.

[44] M. G. Gouda and T. Herman. Adaptive Programming. *IEEE Transactions on Software Engineering*, 17(9) :911–921, 1991.

[45] W. Gropp and E. L. Lusk. Reproducible Measurements of MPI Performance Characteristics. In *6th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, London, UK, 1999.

[46] D. Grove and P. Coddington. Precise MPI Performance Measurement Using MPIBench. In *High Performance Computing Asia*, Gold Coast, Queensland, Australia, 2001.

[47] D. A. Grove and P. D. Coddington. Communication Benchmarking and Performance Modelling of MPI Programs on Cluster Computers. *Journal of Supercomputing*, 34 :201–217, 2005.

[48] D. A. Grove and P. D. Coddington. Modeling Message-passing Programs with a Performance Evaluating Virtual Parallel Machine. *Performance Evaluation*, 60 :165–187, 2005.

[49] M. Harman, S. Danicic, and S. O. Computing. Using Program Slicing to Simplify Testing. *Journal of Software Testing, Verification and Reliability*, 5 :143–162, 1995.

[50] T. K. Henri and H. E. Bal. Fast Measurement of LogP Parameters for Message Passing Platforms. In *Lecture Notes in Computer Science*, pages 1176–1183, 2000.

[51] R. W. Hockney. The Communication Challenge for MPP : Intel Paragon and Meiko CS-2. *Parallel Computing*, 20(3) :389–398, 1994.

[52] T. Hoefler, T. Schneider, and A. Lumsdaine. LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model. In *19th ACM International Symposium on High Performance Distributed Computing*, pages 597–604, 2010.

[53] B. Hong and V. K. Prasanna. Adaptive Matrix Multiplication in Heterogeneous Environments. In *International Conference on Parallel and Distributed Systems*, Los Alamitos, CA, USA, 2002.

[54] S. Hunold, T. Rauber, and G. Rünger. Multilevel Hierarchical Matrix-Matrix Multiplication on Clusters. In *18th International Conference of Supercomputing (ICS'04)*, pages 136–145, 2004.

[55] IBM. *Parallel Engineering and Scientific Subroutine Library : Guide and Reference*. New York, USA, 1995.

[56] R. Ihaka and R. Gentleman. R : A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics*, 5(3) :299–314, 1996.

[57] Intel. Intel mpi benchmark. http ://software.intel.com/en-us/articles/intel-mpi-benchmarks/.

[58] B. Javadi, J. H. Abawajy, and M. K. Akbari. Analytical Modeling of Interconnection Networks in Heterogeneous Multi-cluster Systems. *The Journal of Supercomputing*, 40 :29–47, 2007.

[59] A. Kaminsky. *Building Parallel Programs : SMPs, Clusters & Java*. Course Technology Press, Boston, MA, United States, 1st edition, 2009.

[60] Khronos OpenCL Working Group. *The OpenCL Specification, version 1.0.29*, 8 December 2008.

[61] T. Kielmann, H. E. Bal, and K. Verstoep. Fast Measurement of LogP Parameters for Message Passing Platforms. In *International Parallel and Distributed Processing Symposium/International Parallel Processing Symposium*, 2000.

[62] A. Lastovetsky. On Grid-based Matrix Partitioning for Heterogeneous Processors. In *Sixth International Symposium on Parallel and Distributed Computing*, Washington, DC, USA, 2007.

[63] C. Liao and B. M. Chapman. Invited Paper : A Compile-time Cost Model for OpenMP. In *21th International Parallel and Distributed Processing Symposium (IPDPS 2007), Long Beach, California, USA*, pages 1–8, 2007.

[64] B. Lowekamp, D. O'Hallaron, and T. Gross. Topology Discovery for Large Ethernet Networks. *SIGCOMM Comput. Commun. Rev.*, 31 :237–248, 2001.

[65] D. R. Martínez, V. B. Pérez, M. Boullón, J. C. Cabaleiro, and T. F. Pena. Analytical Performance Models of Parallel Programs in Clusters. In *Parallel Computing : Architectures, Algorithms and Applications, ParCo 2007, Forschungszentrum Jülich and RWTH Aachen University, Germany,*, 2007.

[66] F. McMahon. *The Livermore Fortran Kernels : a Computer Test of the Numerical Performance Range*. Lawrence Livermore National Laboratory, 1986.

[67] C. A. Moritz and M. I. Frank. LoGPC : Modeling Network Contention in Message-Passing Programs. *IEEE Transactions on Parallel and Distributed Systems*, 12 :254–263, 1998.

[68] M. S. Müller, M. van Waveren, R. Lieberman, B. Whitney, H. Saito, K. Kumaran, J. Baron, W. C. Brantley, C. Parrott, T. Elken, H. Feng, and C. Ponder. SPEC MPI2007 : an Application Benchmark Suite for Parallel Systems Using MPI. *Concurrency and Computation : Practice & Experience*, 22 :191–205, 2010.

[69] Y. Ngoko. Poly-algorithmes Pour une Programmation Efficace des Problèmes Numériques. Master's thesis, Université de Yaoundé I, 2005.

[70] NVIDIA. The CUDA Homepage. http ://www.nvidia.com/object/cuda_home.html, 2007.

[71] D. Ofelt and J. L. Hennessy. Efficient Performance Prediction for Modern Microprocessors. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Santa Clara, CA, USA, 2000.

[72] Y. Ohtaki, D. Takahashi, T. Boku, and M. Sato. Parallel Implementation of Strassen's Matrix Multiplication Algorithm for Heterogeneous Clusters. In *International Parallel and Distributed Processing Symposium*, Los Alamitos, CA, USA, 2004.

[73] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design, Fourth Edition : The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2008.

[74] B. Penoff, A. Wagner, M. Tüxen, and I. Rüngeler. MPI-NeTSim : A Network Simulation Module for MPI. In *The Fifteenth International Conference on Parallel and Distributed Systems (ICPADS'09)*,, Shenzhen, China, 2009.

[75] F. Petrini, W. chun Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network : High-Performance Clustering Technology. *IEEE Micro*, 22 :46–57, 2002.

[76] J. Pješivac-Grbović, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra. Performance Analysis of MPI Collective Operations. *Cluster Computing*, 10 :127–143, 2007.

[77] S. Prakash and R. L. Bagrodia. MPI-SIM : using parallel simulation to evaluate MPI programs. In *30th Conference on Winter Simulation (WSC '98)*, Los Alamitos, CA, USA, 1998.

[78] T. Rauber and G. Rünger. *Parallel Programming for Multicore and Cluster Systems*. Springer Verlag, 2010.

[79] L. Rauchwerger, F. Arzu, and K. Ouchi. Standard Templates Adaptive Parallel Library (STAPL). In *Selected Papers from the 4th International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers*, London, UK, 1998.

[80] R. Reussner, P. Sanders, L. Prechelt, and M. Muller. SKaMPI : A Detailed, Accurate MPI Benchmark. In *Recent advances in Parallel Virtual Machine and Message Passing Interface : 16th European PVM/MPI Users' Group Meeting*, Espoo, Finland, 1998.

[81] L. Smith and M. Bull. Development of Mixed Mode MPI / OpenMP Applications. *Scientific Programming*, 9(2-3) :83–98, 2001.

[82] Q. O. Snell, A. R. Mikler, and J. L. Gustafson. NetPIPE : A Network Protocol Independent Performance Evaluator. In *International Conference on Intelligent Information Management and Systems*, 1996.

[83] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. J. Dongarra. *MPI : The Complete Reference*. MIT Press, Cambridge, MA, 1996.

[84] N. Thomas, G. Tanase, O. Tkachyshyn, J. Perdue, N. M. Amato, and L. Rauchwerger. A Framework for Adaptive Algorithm Selection in STAPL. In *tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '05, pages 277–288, New York, USA, 2005.

[85] M. Tikir, M. Laurenzano, L. Carrington, and A. Snavely. PSINS : An Open Source Event Tracer and Execution Simulator for MPI Applications. In *15th International Euro-Par Conference on Parallel Processing*, 2009.

[86] L. G. Valiant. A Bridging Model for Parallel Computation. *ACM Communications*, 33(8) :103–111, 1990.

[87] R. Weicker. DHRYSTONE : A Synthetic Systems Programming Benchmark. *Communications of the ACM*, 27(10) :1013–1030, 1984.

[88] M. D. Weiser. *Program Slices : Formal, Psychological, and Practical Investigations of an Automatic Program Abstraction Method*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1979.

[89] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service : A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15 :757–768, 1999.

[90] J. Zhai, W. Chen, and W. Zheng. PHANTOM : Predicting Performance of Parallel Applications on Large-scale Parallel Machines Using a Single Node. In *PPoPP '10 : Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 305–314, New York, NY, USA, 2010.

[91] J. Zhai, T. Sheng, J. He, W. Chen, and W. Zheng. FACT : Fast Communication Trace Collection for Parallel Applications Through Program Slicing. In *23th International Conference on Supercomputing, IBM T.J. Watson Research Center, Yorktown Heights, NY, USA*, 2009.

[92] G. Zheng, T. Wilmarth, P. Jagadishprasad, and L. V. Kalé. Simulation-based Performance Prediction for Large Parallel Machines. *International Journal on Parallel Programming*, 33(2) :183–207, 2005.

Annexe A : MPI

Structure d'un programme C + MPI

```
#include <mpi.h>
int main(int argc, char *argv[])
{
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (rank==0)
        //{Code du premier processus}

        //.
        //.
        //.

    if (rank==size-1)
        //{Code du dernier processus}
    MPI_Finalize ();
}
```

Routines pt/pt(Syntaxe)

Processus source :
MPI_Send(void * buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm) ;
buf : adresse de départ du buffer d'envoi
count : nombre d'éléments à envoyer
datatype : type de données à envoyer
dest : le rang du processus destination
tag : entier identifiant le message
comm : Communicateur MPI

Processus destination :
MPI_Recv(void * buf, int count, MPI_Datatype datatype, int src, int tag, MPI_Comm comm, MPI_Status * status) ;
buf : adresse de départ du buffer de réception
count : nombre d'éléments à recevoir
datatype : type de données à recevoir
src : rang du processus source
tag : entier identifiant le message *comm :* Communicateur MPI
status : Pointeur à une structure du type MPI Status.

La structure MPI_Status contient des informations sur le tag et sur le processus source. Les jokers : MPI_ANY_TAG et MPI_ANY_SOURCE acceptent respectivement tout tag et toute source.

Exemple de codes utilisant les routines pt/pt : Ping-pong

```
#include <mpi.h>
int main (int argc , char** argv)
{
    int rank, size, value ;
    MPI_Status st ;
    MPI_Init (& argc, &argv) ;
    MPI_comm_rank (MPI_COMM_WORLD, & rank) ;
    MPI_comm_size (MPI_COMM_WORLD, & size) ;
    if (rank==0)
    {
        MPI_Send (& rank, 1, MPI_INT, 1, 0, MPI_COMM_WORLD) ;
        MPI_Recv (&value, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, & st) ;
    }
    else
        if (rank==1)
    {
        MPI_Recv (&value,1, MPI_INT, 0, 0, MPI_COMM_WORLD, & st) ;
        MPI_Send (&value,1, MPI_INT, 0, 1, MPI_COMM_WORLD) ;
    }
    MPI_Finalize() ;
    return 0 ;
}
```

Routines collectives

Diffusion : MPI_Bcast
MPI_Bcast(void * buf, int count, MPI_Datatype datatype, int root, MPI_Comm comm) ;

buf : adresse de départ du buffer d'envoi
count : nombre d'éléments à envoyer
datatype : type de données à envoyer
root : le rang du processus source d'information
comm : Communicateur MPI

Distribution : MPI_Scatter

MPI_Scatter(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm) ;

sendbuf : adresse de départ du buffer d'envoi

sendcnt : nombre d'éléments à envoyer

sendtype : type de données à envoyer

recvbuf : adresse de départ du buffer de réception

recvcnt : nombre d'éléments à recevoir

recvtype : type de données à recevoir

root le rang du processus source d'information

comm : Communicateur MPI

Collecte : MPI_Gather

MPI_Gather(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm) ;

sendbuf : adresse de départ du buffer d'envoi

sendcnt : nombre d'éléments à envoyer

sendtype : type de données à envoyer

recvbuf : adresse de départ du buffer de réception

recvcnt : nombre d'éléments à recevoir

recvtype : type de données à recevoir

root le rang du processus source d'information

comm : Communicateur MPI

Annexe B : Régression polynômiale

Régression polynômiale : principe

Considérons un ensemble de N points expérimentaux (x_i, y_i) $i = 1..N$.
Le but avec la régression polynômiale est de trouver un polynôme R de degré n et de coefficients $a_0, a_1, \ldots, a_n$ (pour que la méthode marche, il faut bien sûr que N>n).
Avec $R(x_i) = \sum_{p=0}^{n} a_p x_i^p$

Le but est de minimiser la différence entre les valeurs prises par R aux points x_i et les valeurs expérimentales y_i. Pour ceci on peut utiliser la méthode des moindres carrés, c'est-à-dire minimiser la somme des carrés des écarts entre valeurs expérimentales et valeurs calculées par le polynôme. Les seuls paramètres variables sont les coefficients du polynôme. On minimise donc la quantité :
$E = \sum_{i=1}^{N} (y_i - R(x_i))^2 = \sum_{i=1}^{N} [y_i - \sum_{p=0}^{n} a_p x_i^p]^2$
Cette quantité peut être considérée comme une fonction des variables $a_0, a_1, \check{\mathrm{E}}, a_n$. Et pour minimiser E, il suffit de trouver pour chaque a_p quelle valeur annule la dérivée partielle de E par rapport à a_p. La résolution de ce problème peut être rendu à la résolution d'un système de Kramer de n+1 équations à n+1 inconnus. Ce système peut s'écrire sous forme matricielle :
B=MA
avec :
- A est un vecteur de n éléments et $A(q) = a_q$
- B est un vecteur de n éléments et $B(q) = \sum_{i=1}^{N} y_i x_i^p$
- M est une matrice (n,n) et $M(p,q) = M(q,p) = \sum_{i=1}^{N} x_i^{p+q}$

Il est bien clair que la solution de ce problème est $A = M^{-1}B$.

L'environnement R [1]

R est un logiciel de statistique qui représente à la fois un langage informatique et un environnement de travail. Ce logiciel est multi plates-formes(Windows, Linux, Mac), gratuit et à code source ouvert(open source). Le projet R est initialement écrit par Ross Ihaka et Robert Gentleman [56]connus aussi par "R&R". Actuellement, les sources de R sont accessibles en écriture par une vingtaine de personnes qui font le cœur de la communauté de ce logiciel.
Dans ce logiciel, plusieurs techniques statistiques modernes et classiques qui ont été implémentées. Les méthodes les plus courantes sont :

- Statistique descriptive,
- Tests d'hypothèses,
- Analyse de la variance,
- Méthodes de régression linéaire (simple et multiple).

L'environnement R offre également la possibilité de générer des graphiques de qualité à l'aide des commandes de son langage.

Régression polynômiale avec R sous Linux

Lecture des données

data<-read.table("chemin_du_fichier_de_données",header=TRUE,dec=",")

Enregistrement des variables dans les entê tes

attach(data)

Déclaration d'une formule

$t < -lm(t\ I(n) + I(n^2) + I(n^3))$

Régression selon la formule

summary(t)

1. http://www.r-project.org/

Exemple de Résultats :

```
Call:
lm(formula = t ~ I(N) + I(N^2) + I(N^3))

Residuals:
      Min        1Q    Median        3Q       Max
-0.361705 -0.055294 -0.005817  0.062735  0.580399

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.496e-01  2.037e-01   0.735    0.466
I(N)        -2.274e-04  2.364e-04  -0.962    0.341
I(N^2)       1.203e-07  7.908e-08   1.522    0.135
I(N^3)       5.730e-10  7.915e-12  72.388   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1617 on 49 degrees of freedom
Multiple R-squared:     1,      Adjusted R-squared:     1
F-statistic: 8.451e+05 on 3 and 49 DF,  p-value: < 2.2e-16
```

Figure 34 – Exemple de régression avec R sous Linux

Annexe C : Découpage de programmes

Intérêts

Au début, le découpage est limité à des objectifs basiques qui ne dépassaient pas le terme de simplification et compréhension des programmes. Mais, des études ultérieures [49] ont permis d'élargir ses horizons, principalement, en introduisant cette technique dans la construction des graphes de dépendances. Le découpage peut être appliqué pour plusieurs autres objectifs, telle que :

- La détermination du degré de cohésion d'un programme.
- La maintenance et le débogage des programmes.
- La retro-ingénierie.
- La réutilisation des composants.
- La parallélisation automatique.
- La vérification assistée des applications.
- Le calcul des performances et des complexités.

Principe

La question qui se pose maintenant est : comment peut-on juger quelles instructions à garder et quelles instructions à supprimer ? La réponse à cette question est la suivante : la sémantique qu'on va utiliser est basée sur les instructions dans le programme (routines de communication, fonctions, procédures ou variables) et ensuite construire son graphe de dépendances des données, ce qui rend une instruction dépendante d'une autre, c'est la façon dont ses deux instructions utilisent les différentes variables du programme.

```
1   <type> A ;
2   <type> B ;
3   A = 2 ;
4   B = 9 ;
5   I1 ( A ) ;
6   I2 ( B ) ;
7   B = 10;
```

(a) Programme à découper

```
2   <type> B ;
4   B = 9 ;
6   I2 ( B ) ;
```

(b) Programme découpé

Figure 35 – Exemple de découpe de programme

Pour éclaircir l'idée, soit le programme P possédant deux variables A et B et deux instructions I1 et I2. L'instruction I1 utilise la variable A et I2 utilise la variable B (voir figure 35(a)).

Supposons que nous voulons focaliser notre étude sur l'instruction I2 de ce programme. Nous devons, dans un premier temps, extraire les variables figurant dans les paramètres de cette instruction, dans notre exemple c'est la variable B. Ensuite, nous devons faire le suivi de chaque paramètre extrait. Nous parcourons le programme et nous gardons les numéros des lignes où la variable B a subi un changement. On peut remarquer que B a subi des changements dans les lignes 4 et 7. La ligne 4 est gardée alors que la ligne 7 est rejetée puisque ce changement n'affecte pas I2. Les numéros des lignes finalement gardés sont 2, 4 et 6. Le fragment final est représenté par la figure 35(b).

Classification

Il existe deux principaux paradigmes de découpage : un découpage sémantique et un découpage syntaxique. Le découpage sémantique possède plusieurs procédés de découpage. Les plus utilisés sont :

- Le découpage statique qui néglige les entrées.
- Le découpage dynamique : une seule entrée est prise en charge.
- Le découpage quasi-statique : plusieurs entrées peuvent être prises en charge.

```
1   X = 1;
2   Y = 2;
3   Z = Y −2;
4   R = X;
5   Z = X + Y;
```

(a) Code à découper

```
1   X=1 ;
2   Y=2 ;
3   Z=X+Y ;
```

(b) Code découpé

Figure 36 – Découpage de programme selon une variable

```
1   X=100 ;
2   Y=2+X ;
3   F=Factoriel(X)
4   X=Y+2
```

(a) Code à découper

```
1   X=100
2   F=Factoriel(X)
```

(b) Code découpé

Figure 37 – Découpage de programme selon une procédure

- Le découpage conditionnel : bascule entre le découpage statique et le découpage dynamique suivant les conditions.

Le découpage syntaxique possède deux principaux types de procédés :

- Le découpage amorphe (Amorphous).
- Le découpage par préservation de syntaxe.

Pour clarifier le concept de découpage statique, soit le fragment de code C de la figure 36(a).

Supposons que nous nous intéressons seulement à la variable Z et que nous voulons savoir l'effet de ce fragment de code sur la valeur de la variable Z à la fin de son exécution. L'application de la technique de découpage sur ce fragment du code Z aboutira au résultat donné dans la figure 36(b). Il est remarquable que la ligne 4 (R = X) n'affecte pas la valeur finale de Z, c'est pour cela qu'elle n'est pas inclue dans le fragment final. On peut remarquer aussi que l'instruction de la ligne 3 (Z=Y- 2) n'a pas d'influence sur la valeur finale de Z, c'est pour cette raison qu'elle ne figure pas dans le fragment final. Dans l'exemple de la figure 37 au lieu de nous intéresser à une variable, nous nous intéressons à une procédure. La procédure *Factoriel* ne dépend que de la variable de la ligne 1(X =100), donc l'instruction de la ligne 2 (Y = 2 + X) sera supprimée. L'instruction de la ligne 4 (X = Y + 2) n'a pas aussi d'effet sur la procédure *Factoriel* puisqu'elle s'exécute après.

Schémas d'adaptations algorithmiques sur les nouveaux supports d'exécution parallèles

Résumé. Avec la multitude des plates-formes parallèles émergentes caractérisées par une hétérogénéité sur le plan matériel (processeurs, réseaux, …), le développement d'applications et de bibliothèques parallèles performantes est devenu un défi. Une méthode qui se révèle appropriée pour relever ce défi est l'approche adaptative consistant à utiliser plusieurs paramètres (architecturaux, algorithmiques,…) dans l'objectif d'optimiser l'exécution de l'application sur la plate-forme considérée. Les applications adoptant cette approche doivent tirer avantage des méthodes de modélisation de performance pour effectuer leurs choix entre les différentes alternatives dont elles disposent (algorithmes, implémentations ou ordonnancement). L'usage de ces méthodes de modélisation dans les applications adaptatives doit obéir aux contraintes imposées par ce contexte, à savoir la rapidité et la précision des prédictions. Nous proposons dans ce travail, en premier lieu, un framework de développement d'applications parallèles adaptatives basé sur la modélisation théorique de performances. Ensuite, nous nous concentrons sur la tâche de prédiction de performance pour le cas des milieux parallèles et hiérarchiques. En effet, nous proposons un framework combinant les différentes méthodes de modélisation de performance (analytique, expérimentale et simulation) afin de garantir un compromis entre les contraintes suscités. Ce framework profite du moment d'installation de l'application parallèle pour découvrir la plate-forme d'exécution et les traces de l'application afin de modéliser le comportement des parties de calcul et de communication. Pour la modélisation de ces deux composantes, nous avons développé plusieurs méthodes s'articulant sur des expérimentations et sur la régression polynômiale pour fournir des modèles précis. Les modèles résultats de la phase d'installation seront utilisés (au moment de l'exécution) par notre outil de prédiction de performance de programmes MPI (MPI-PERF-SIM) pour prédire le comportement de ces derniers. La validation de ce dernier framework est effectuée séparément pour les différents modules, puis globalement pour le noyau du produit de matrices.

Mots clés : adaptativité, modélisation et prédiction de performance, MPI, plates-formes hiérarchiques, régression, simulation.

Algorithmic adaptation schemata for recent parallel platforms

Abstract. With the multitude of emerging parallel platforms characterized by their heterogeneity in terms of hardware components (processors, networks, ...), the development of performant applications and parallel libraries have become a challenge. A method proved suitable to face this challenge is the adaptive approach which uses several parameters (architectural, algorithmic, ...) in order to optimize the execution of the application on the target platform. Applications adopting this approach must take advantage of performance modeling methods to make their choice between the alternatives they have (algorithms, implementations or scheduling). The use of these modeling approaches in adaptive applications must obey the constraints imposed by the context, namely predictions speed and accuracy. We propose in this work, first, a framework for developing adaptive parallel applications based on theoretical modeling performance. Then, we focuse on the task of performance prediction for the case of parallel and hierarchical environments. Indeed, we propose a framework combining different methods of performance modeling (analytical, experimental and simulation) to ensure a balance between the constraints raised. This framework makes use of the installing phase of the application to discover the parallel platform and the execution traces of this application in order to model the behavior of two components namely computing kernels and pt/pt communications. For the modeling of these components, we have developed several methods based on experiments and polynomial regression to provide accurate models. The resulted models will be used at runtime by our tool for performance prediction of MPI programs (MPI-PERF-SIM) to predict the behavior of the latter. The validation of the latter framework is done separately for the different modules, then globally on the matrix product kernel

Key words : adaptive, hierarchical clusters, MPI, performance modeling and prediction, regression, simulation.

أنماط تكيف خوارزمي للمنصات الموازية الحديثة

الملخص. في ظل تعدد المنصات الموازية المتسمة بعدم التجانس في معداتها (معالجات ، شبكات)، أصبح من التحدي تطوير تطبيقات ذات أداء جيد. يبدو أن أفضل وسيلة لرفع هذا التحدي يكمن في إتباع نهج التكيف عند الأداء. علما و أن هذه الطريقة تستعمل عدة بيانات عن المنصة و عن الخوارزميات لتحسين الأداء عند الاستخدام. و للتمييز بين الخيارات المتاحة لإحدى التطبيقات المكيفة يجب استعمال أساليب النمذجة في حساب الأداء. و الجدير بالذكر أن استعمال هذه الأساليب يجب أن يخضع لمقياسين و هما السرعة و الدقة في التنبؤ. نقترح في أول هذا العمل منظومة لتطوير البرمجيات الموازية المعتمدة على التكيف عند الأداء و ذلك عبر استعمال النمذجة النظرية. وفي باقي هذا العمل نركز جهودنا على نمذجة التطبيقات خصوصا في ظل المنصات الموازية و الطبقية. و قد قدمنا في هذا الصدد منظومة معتمدة على المزج بين عدة أساليب نمذجة (نظرية و تطبيقية ومحاكاة) وذلك من أجل التوفيق بين المقياسين المذكورين أعلاه. هذا وتستغل هذه المنظومة فترة التنصيب لاكتشاف عدة معلومات عن المنصة و عن البرنامج وذلك من أجل نمذجة الاتصالات و الأجزاء المخصصة للحسابات. وقد قمنا لنمذجة هذين المكونين ببرمجة طريقة مرتكزة على التجارب وعلى الانحسار متعدد الحدود و ذلك من أجل الحصول على نماذج دقيقة. و تستغل النماذج المتحصل عليها للتنبؤ بأداء البرمجية عبر الأداة (MPI-PERF-SIM) التي تستعمل للمحاكاة السريعة للبرمجيات . و للتثبت من جدارة المنظومة المقترحة قمنا بالتحقق من جميع وحداتها كل على حدة ثم التحقق من المنظومة كاملة و ذلك بالنسبة لنواة ضرب المصفوفات.

الكلمات المفاتيح: انحسار، تكيف ، محاكاة ، م.ب.آي ، منصات طبقية ، نمذجة و تنبؤ الأداء.