Prédiction de performances d'applications de calcul distribué exécutées sur une architecture pair-à-pair

Bogdan Florin Cornea

▶ **To cite this version:**

Bogdan Florin Cornea. Prédiction de performances d'applications de calcul distribué exécutées sur une architecture pair-à-pair. Autre [cs.OH]. Université de Franche-Comté, 2011. Français. NNT : 2011BESA2012 . tel-00800314

HAL Id: tel-00800314
https://theses.hal.science/tel-00800314

Submitted on 13 Mar 2013

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

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

Université de Franche-Comté **U.F.R. Sciences,Techniques et Gestion de l'Industrie**

Année : 2011 **École doctorale SPIM**
N° d'ordre : 103

Prédiction de performances d'applications de calcul distribué exécutées sur une architecture pair-à-pair

THÈSE

présentée et soutenue publiquement le 8 décembre 2011

pour l'obtention du

Grade de Docteur de l'Université de Franche-Comté

(spécialité : Informatique)

par

Bogdan Florin CORNEA

Composition du jury

Directeur de thèse :	Julien BOURGEOIS	Professeur à l'Université de Franche-Comté
Rapporteurs :	Pascal FELBER	Professeur à l'Université de Neuchâtel
	Jean-François MÉHAUT	Professeur à l'Université Joseph Fourier
Examinateurs :	Didier EL BAZ	Chargé de Recherche HDR au LAAS-CNRS
	Martin QUINSON	Maître de Conférences à l'Université de Nancy
	Jean-Marc VINCENT	Maître de Conférences à l'Université Joseph Fourier

Laboratoire d'Informatique de l'Université de Franche-Comté — LIFC EA 4269
Projet ANR CIP N° ANR-07-CIS7-011-01

Mis en page avec la classe thloria.

Ces travaux doctoraux ont été soutenus par un nombre important de personnes, ils ont été encadrés avec beaucoup de professionnalisme, et ils ont été évalués et validés par des spécialistes du domaine. À toutes ces personnes je tiens à leur remercier sincèrement.

En particulier, je remercie :

Pascal FELBER, Professeur à l'Université de Neuchâtel (Suisse), le Président du Jury de thèse, qui a évalué mon travail et a également apprécié la complexité de ma thèse et la difficulté d'implémenter cette contribution.

Jean-François MÉHAUT, Professeur à l'Université Joseph Fourier, d'avoir évalué ces travaux doctoraux, et d'avoir proposé des idées d'amélioration de cette contribution.

Didier EL BAZ, Chargé de la Recherche HDR au LAAS-CNRS, et Martin QUINSON, Maître de Conférences à l'Université de Nancy, ainsi que Jean-Marc VINCENT, Maître de Conférences à l'Université Joseph Fourier, qui ont accepté de faire partie du Jury de thèse et qui ont partagé leur maîtrise du domaine HPC. Leur feedback m'a permis d'identifier de nouvelles perspectives de recherche, qui permettent de faire des prédictions de performances dans un contexte réel plus complexe que celui pris en compte pendant ces travaux.

Julien BOURGEOIS, Professeur à l'Université de Franche-Comté, qui a dirigé et encadré mon activité de recherche pendant ces trois années de doctorat. Sa maîtrise du domaine HPC et particulièrement P2P ainsi que sa vision sur les tendances futures des scientifiques du domaine ont représenté des éléments essentiels pour mes travaux. Sans sa confiance dans mon potentiel et sans ses conseils précieux et permanents, je n'aurais pas pu atteindre les objectifs de cette thèse. Je lui remercie pour l'énergie et le temps investis dans ma formation en tant que docteur ainsi que pour les connaissances acquises grâce à lui pendant ces années.

L'ensemble des Professeurs, Maîtres de Conférences, doctorants et personnel administratif du LIFC/DISC pour leur soutien, leurs conseils, leurs critiques constructives et l'ambiance crée au sein du laboratoire.

Les enseignants de l'IUT Belfort-Montbéliard du département Services et Réseaux de Communication ainsi que celui des Réseaux et Télécommunications qui m'ont permis de débuter dans l'enseignement supérieur.

Mes amis à Montbéliard (France), à Brasov (Roumanie), et dans d'autres villes ou pays, qui ont crée des moments de détente très variés et agréables tout au long de la période de mes études doctorales.

Je tiens à remercier ma mère Lucia, mon frère Ciprian pour leur amour et leur confiance en moi, pour tous leurs conseils, leur soutien moral, leurs partages de connaissances, et leur partage des sentiments optimistes. À mes belles-sœurs, ma belle-famille et aux grands-parents pour leur intérêt permanent sur mon état d'esprit, ainsi que pour tous les moments encourageants qu'ils sont crées pendant cette période. Ceux-ci m'ont encouragé et aidé à persévérer pour atteindre les buts de ces travaux.

Je remercie à ma chère Madalina, qui est à mes côtés depuis le début de cette de thèse. Elle m'a soutenu dans les moments difficiles, m'a conseillé dans les moments de prise de décisions, m'a poussé à avancer pour arriver au terme de ces travaux et m'a supporté pendant toutes mes démarches administratives que j'ai eues à préparer durant ces années. Je lui remercie pour tout son amour, sa confiance, son optimisme et sa patience.

À toutes les autres personnes qui m'ont soutenu et qui m'ont encouragé afin de réussir à défendre cette thèse, j'espère qu'à mon tour je leur ai partagées de l'expérience acquise pendant la période 2008-2011, et je leur remercie encore une fois.

Table des matières

Liste des tableaux ix

Table des figures xi

Introduction 1

I État de l'art des outils de prédiction de performances 9

Introduction 11

1 Les outils analytiques de prédiction de performances 13

Introduction . . . 13

1.1 LogP - 1993 . . . 13

1.2 LogGP - 1995 . . . 14

1.3 Uni-Paderborn - 1996 . . . 14

1.4 LoGPC - 1998 . . . 15

1.5 UCHICAGO - 2001 . . . 15

1.6 LANL - 2001 . . . 16

1.7 TUDELFT - 2003 . . . 16

Vue comparative . . . 17

2 Les outils de prédiction de performances basés sur des mesures 21

Introduction . . . 21

2.1 P^3T - 1993 . . . 21

2.2 USC-UC - 1996 . . . 22

2.3 NWS - 1999 . . . 22

2.4 ARM - 1999 . . . 23

2.5 FAST - 2003 . . . 23
2.6 PEVPM - 2003 . . . 24
2.7 APAPS - 2004 . . . 24
2.8 NCSU - 2005 . . . 25
2.9 GPRES - 2005 . . . 26
2.10 UPB-2009 . . . 26
2.11 Leiden - 2010 . . . 27
2.12 PAS2P - 2010 . . . 27
2.13 NCSU (Wu) - 2011 . . . 28
Vue comparative . . . 29

3 Les outils hybrides de prédiction de performances 33
Introduction . . . 33
3.1 Dimemas - 1996-2004 . . . 33
3.2 MPI-Sim - 1998 . . . 34
3.3 ChronosMix - 2000 . . . 34
3.4 PACE - 2000 . . . 35
3.5 POEMS - 2000 . . . 36
3.6 BigSim - 2004 . . . 37
3.7 POSE - 2005 . . . 37
3.8 P2PPerf - 2006 . . . 38
3.9 Susukita - 2008 . . . 38
3.10 PSINS - 2009 . . . 39
3.11 WARPP - 2009 . . . 40
3.12 PHANTOM - 2010 . . . 40
3.13 MPI-PERF-SIM - 2011 . . . 41
3.14 SMPI - 2011 . . . 42
Vue comparative . . . 42

4 Les outils d'analyse et de visualisation des performances 47
4.1 Paradyn et DynInst . . . 47
4.2 *HPCToolkit* . . . 48
4.3 TAU . . . 48
4.4 Vampir . . . 49
4.5 *HPM Toolkit* . . . 49

4.6 Jumpshot . . . 49
4.7 Paraver . . . 50
4.8 mpiP . . . 50
4.9 KOJAK . . . 50
4.10 *Scalasca* . . . 51
4.11 *ScalaTrace* . . . 51
4.12 *Visual Studio Profiling Tools* . . . 52
4.13 D'autres outils d'analyse ou de transformation de code source . . . 52

Conclusion **55**

II Contribution - Prédiction de performances **57**

Introduction . . . 59

5 Pré-requis **63**

5.1 Les compteurs matériels de performance (*Hardware Performance Counters*) . . . 63
5.2 GNU/Linux et l'accès aux compteurs matériels . . . 64
5.3 PAPI - accéder aux compteurs matériels . . . 65
5.3.1 Les fonctions PAPI de haut niveau . . . 66
5.3.2 Les fonctions PAPI de bas niveau . . . 66
5.4 ROSE . . . 68
5.4.1 Une vue d'ensemble . . . 68
5.4.2 Des représentations intermédiaires (IR) . . . 70
5.4.2.1 L'*Abstract Syntax Tree* (AST) . . . 70
5.4.2.2 Le *Data Dependence Graph* (DDG) . . . 71
5.4.2.3 Le *Control Dependence Graph* (CDG) . . . 71
5.4.2.4 Le *System Dependence Graph* (SDG) . . . 75
5.5 L'outil de simulation Simgrid . . . 75
5.5.1 Généralités . . . 75
5.5.2 Gestion de MPI, de P2P-SAP ou d'une autre bibliothèque de communication . . . 77

6 Terminologie, méthodologie et implémentation **79**

6.1 La terminologie . . . 80

6.1.1 Les temps mesurés ou estimés 80
6.1.2 L'application d'entrée 82
6.1.2.1 Des multiples langages de programmation : C, C++, Fortran 82
6.1.2.2 Des multiples formalismes de communication : MPI, P2P-SAP 82
6.1.3 Le *slowdown* 83
6.1.4 Le coût de la prédiction 84
6.1.4.1 Technique simple de *benchmarking* par bloc d'instructions 84
6.1.4.2 Technique optimisée de *benchmarking* par bloc d'instructions 85
6.1.5 La précision 85
6.1.6 Les systèmes et les topologies ciblés par dPerf 85
6.2 La méthodologie 87
6.2.1 Réduction du *slowdown* 87
6.2.1.1 Choisir le code d'entrée 88
6.2.1.2 Les niveaux d'optimisation du compilateur 88
6.2.1.3 Benchmarking simple par blocs d'instructions 91
6.2.1.4 Benchmarking optimisé par blocs d'instructions 93
6.2.1.5 Rendre le code transformé 99
6.2.2 Extrapolation de performances 100
6.2.2.1 L'identification de la topologie logique du code d'entrée 100
6.2.2.2 Simulation avec Simgrid MSG et les résultats de prédiction 103
6.2.3 Le coût pour accéder à une prédiction faite avec dPerf 106
6.3 L'implémentation 106
6.3.1 Réduction du *slowdown* 107
6.3.2 Simulation basée sur fichiers de traces 110
6.3.2.1 Obtenir les fichiers de trace 110
6.3.2.2 Ré-jouer les traces 111
6.4 La disponibilité publique de dPerf 111

III Étude de cas 113

7 Étude de la fonctionnalité et de la précision 115
7.1 Précision de la prédiction du temps de calcul 116
7.2 Prédiction de performance de l'application NAS Integer Sort 116
7.3 Prédiction de performance du problème de l'obstacle 126

8 Étude d'une application : le problème de l'obstacle 129
8.1 Identification de la topologie logique 129
8.2 Prédiction de performances sur un réseau cible 132
8.3 Extrapolation du nombre de nœuds 133

Conclusion 137

Publications 141
1 Des conférences internationales avec comité de lecture 141
2 Des revues internationales avec comité de lecture 141
3 Des rapports techniques internes 141

Bibliographie 143

Liste des tableaux

1.1 Les caractéristiques les plus importantes des outils analytiques de prédiction de performances, et les caractéristiques de l'outil dPerf développé durant ce travail de recherche. (1)Prendre en compte le niveau d'optimisation du compilateur ; (2) La communication ; (3)Voir chapitre 6 pour la méthodologie et l'implémentation de dPerf ; (4)Le coût de la résolution du modèle ; 19

2.1 Les caractéristiques les plus importantes des outils de prédiction de performances basés sur des mesures, et les caractéristiques de l'outil dPerf. (1)Prendre en compte le niveau d'optimisation du compilateur ; (2) La communication ; (3)Voir chapitre 6 pour la méthodologie et l'implémentation de dPerf ; (4)La topologie logique de communication ; (5)Le coût de la résolution du modèle ; 31

3.1 Les caractéristiques les plus importantes des outils hybrides de prédiction de performances, et les caractéristiques de l'outil dPerf. (1)Prendre en compte le niveau d'optimisation du compilateur ; (2) La communication ; (3)Voir chapitre 6 pour la méthodologie et l'implémentation de dPerf ; (4)La topologie logique de communication ; (5) Toute application qui communique dans un formalisme *Message Passing* ; (6) *Speedup*, et non pas *slowdown* ; (7)Le coût de la résolution du modèle ; (8) Le *speedup* du simulateur 45

4.1 Liste des outils pertinents d'analyse, de transformation ou de compilation de codes source. (1) Voir la section 5.4 ; 53

6.1 Vue sur les principaux niveaux d'optimisation des compilateurs G++ et ICPC. 1 Le compilateur ; 89
6.2 Le résultat de l'identification faite par dPerf dans la phase statique. Ce résultat était obtenu pour un programme simple en MPI. 1 D'autres cas non gérés par dPerf ou non identifiables ; (2)Temps pour obtenir la décision ; 101
6.3 Les événements reconnus et gerés par dPerf. 110

7.1 Les applications distribuées utilisées pour étudier la précision des prédictions de performances faites par dPerf. (1) NAS Integer Sort ; 116

8.1 Comparaison des puissances de calcul entre le système hôte et les prédictions faite par dPerf pour le code de l'obstacle. 133

Table des figures

1 L'intérêt des outils de prédiction de performances pour le développement et l'adaptation des application aux systèmes cibles, ainsi que dans l'évaluation des systèmes existants et futurs en vue d'augmenter leur puissance de calcul. 2

4.0.1 L'évolution des architectures de calcul (concernant le domaine de l'*HPC*) à partir des nœuds indépendants, vers les grappes de calcul, les grille de calcul, le *cloud* ou les super-ordinateurs. 59

5.1 La hiérarchie nécessaire pour accéder les compteurs matériels de performances. Cet accès est possible directement à travers les bibliothèques *perfmon* et *perfctr*, ou en utilisant l'API de PAPI pour accéder aux mêmes bibliothèques 65
5.2 Vue comparative entre la prise de mesure avec PAPI en mode "réel" et en mode "virtuel" 67
5.3 Les trois parties de ROSE. 68
5.4 La structure générale d'un outil personnalisé basé sur ROSE 69
5.5 L'AST correspondant au code exemple présenté dans la Fig. 5.1 72
5.6 Les DDG correspondant au code exemple présenté dans la Fig. 5.1. À gauche, le DDG de la fonction "main", et à droite, le DDG de la fonction "addition" 73
5.7 Les CDG correspondant au code exemple présenté dans la Fig. 5.1. À gauche, le CDG de la fonction "main", et à droite le CDG de la fonction "addition" 74
5.8 Un extrait du SDG correspondant au code exemple présenté dans la Fig. 5.1. 76
5.9 La structure du *framework* Simgrid. dPerf utilise le module MSG de Simgrid. 77

6.1 L'outil dPerf. En gris, la bibliothèque PAPI et les outils ROSE et SimGrid développés en externe. En blanc, l'élargissement (en vue de la compatibilité avec nos besoins) et le développement faisant partie de dPerf. 80
6.2 Le processus de prédiction avec dPerf. La durée de chaque étape est montrée sur la ligne du temps. 81
6.3 La classification des systèmes HPC en fonction de leur disponibilité. Présentation de la topologie de communication employée par les applications HPC ainsi que les systèmes potentiels acceptant chacune des topologies. . . 86

6.4 Le flux de travail dans dPerf. Le code source d'une application est reçu en entrée, et une prédiction de performances est disponible à la sortie de dPerf. 88
6.5 Résultats d'une étude sur l'impact des optimisations du compilateur au niveau du temps moyen par itération d'une boucle. Expériences faites avec un code simple contenant une structure répétitive avec réutilisation des variables en fin du programme. Compilateur utilisé : ICPC 90
6.6 La décomposition d'un bloc d'instructions (celui à gauche) en sous-blocs (voir l'image droite) : un bloc contient des instructions séquentielles, et l'autre contient uniquement l'appel de la communication. L'identification des communication P2P-SAP ou MPI se fait de la même façon, le deux étant gérés par dPerf 91
6.7 La décomposition d'un bloc d'instructions (celui à gauche) en sous-blocs (voir l'image droite) avec la technique optimisée de *benchmarking* : un bloc contient des instructions séquentielles, et l'autre uniquement l'appel de la communication ; Une modification du nombre d'itérations a lieu. L'identification des communication P2P-SAP ou MPI se fait de la même façon, le deux étant gérés par dPerf 93
6.8 La courbe du temps moyen par itération d'une boucle de test. Les différents niveaux d'erreur que l'utilisateur pourra imposer, peuvent influencer le *threshold* 96
6.9 Pour un *benchmark* de NAS, le coût de prédiction avec la technique simple de *benchmarking* est très proche du temps de l'exécution normale. Contrairement à ces temps rapprochés, le coût de prédiction, quand dPerf utilise la technique de *threshold* , est fortement réduit. Pour ces résultats, le compilateur n'utilise pas d'optimisations (le niveau choisi est 0). 97
6.10 Expériences avec un code d'exemple en vue d'étudier le temps moyen par itération d'une boucle. Code compilé avec G++ et puis ICPC. Les niveaux potentiels d'erreur de 0, 5 et 10% sont indiqués. 98
6.11 Trois topologies logiques sont identifiables. La topologie maître-travailleur (le processus maître en gris) dans 6.11a ; une communication de type maille en 2 dimensions dans 6.11b ; le tore en 3 dimensions dans 6.11c. 100
6.12 Extrapolation de N à N' pour une topologie maître-travailleur (a,b), et maille en 2d (c,d). (a,c) Les topologies identifiées par dPerf. (b,d) La topologie $dPerf_{toplog}$ d'un nombre augmenté de nœuds. Extrapolation faite par dPerf. 105
6.13 Exemple de *slowdown* , gain et temps de simulation ($t_{simulation}$) obtenus avec dPerf pour l'application NAS IS sur un nombre de 2, 4, 8 et 16 processus, et pour un problème de classe A. 107

7.1 Exécution, prédiction et erreur à la prédiction pour la transformée de Laplace, implémentation séquentielle. 117
7.2 Le pourcentage de calcul et de communication, en fonction du niveau d'optimisation de GCC, pour l'exécution de NAS IS. 118
7.3 La topologie réseau de la plate-forme *IS-Exp1*. Les nœuds identiques sont représentés avec une même couleur. 119

7.4 La topologie réseau de la plate-forme *IS-Exp2*. Les nœuds identiques sont représentés avec une même couleur. 120
7.5 Le scenario *IS-Exp1*. Le temps prédit est comparé au temps d'exécution. L'erreur montre la précision de dPerf par rapport à la réalité. Les résultats sont étudiés pour tous les niveaux d'optimisation pertinentes disponibles dans GCC. 121
7.6 Le *slowdown* et le gain pour le scénario *IS-Exp1*. 122
7.7 Le scénario *IS-Exp2*. Le temps prédit est comparé au temps d'exécution. L'erreur montre la précision de dPerf par rapport à la réalité. Les résultats sont étudiés pour tous les niveaux d'optimisation pertinentes disponibles dans GCC. 124
7.8 Le *slowdown* et le gain pour le scénario *IS-Exp2*. 125
7.9 Le pourcentage de calcul et de communication, en fonction du niveau d'optimisation de GCC, pour l'exécution du code de l'obstacle. 126
7.10 Le temps d'exécution, la prédiction et l'erreur de dPerf pour le code de l'obstacle. 128

8.1 Identification d'un appel de communication à l'aide de l'AST du code de l'obstacle. 131
8.2 La topologie réseau de la marguerite (*daisy*). 132
8.3 Vue comparative des performances du code de l'obstacle pour des différentes topologies réseau testées avec dPerf. 133
8.4 Prédiction de performance du code de l'obstacle avec dPerf, à base des traces pour 2 nœuds 135

Résumé

Dans le domaine du calcul de haute performance, les architectures d'exécution sont en continuelle évolution. L'augmentation du nombre de nœuds de calcul, ou le choix d'une topologie réseau plus rapide représentent un investissement important tant en temps qu'en moyen financier. Les méthodes de prédiction de performances permettent de guider ce choix. En parallèle à ce développement, les systèmes HPC pair-à-pair (P2P) se sont également développés ces dernières années. Ce type d'architecture hétérogène permettrait la résolution des problèmes scientifiques pour un coût très faible par rapport au coût d'une architecture dédiée.

Ce manuscrit présent une méthode nouvelle de prédiction de performances pour les applications réelles de calcul distribué, exécutées dans des conditions réelles. La prédiction prend en compte l'optimisation du compilateur. Les résultats sont extrapolables et ils sont obtenus pour un ralentissement réduit. Ce travail de recherche est implémenté dans un logiciel nouveau nommé dPerf. dPerf est capable de prédire les performances des applications C,C++ ou Fortran qui communiquent en utilisant les normes MPI ou P2P-SAP et qui s'exécutent sur une architecture cible pair à pair, hétérogène et décentralisée. La précision de cette contribution a été étudiée sur (i) la transformée Laplace, pour l'aspect séquentiel, (ii) le *benchmark* IS de NAS, pour l'aspect MPI, (iii) et le code de l'obstacle pour l'aspect calcul P2P décentralisé et l'extrapolation du nombre des nœuds.

Mots-clés: Réduction du *slowdown*, analyse statique, instrumentation automatique, extrapolation de performances, simulation basée sur fichiers de trace, prédiction de performances, langages C, C++, Fortran, systèmes hétérogènes, environnement décentralisé, calcul pair-à-pair de haute performance.

Abstract

In the field of high performance computing, the architectures evolve continuously. In order to increase the number of computing nodes or the network speed, an important investment must be considered, from both temporal and financial point of view. Performance prediction methods aim at assisting in finding the best trade-off for such an investment. At the same time, P2P HPC systems have known an increase in development. These heterogeneous architectures would allow solving scientific problems at a low cost, with respect to dedicated systems.

This manuscript presents a new method for performance prediction. This method applies to real applications for distributed computing, considered in a real execution environment. This method uses information about the different compiler optimization levels. The prediction results are obtained with a reduced slowdown and are scalable. This thesis took shape in the development of the dPerf tool. dPerf predicts the performances of C,

C++, and Fortran application, which use MPI or P2P-SAP to communicate. The applications modelled by dPerf are meant for execution on P2P heterogeneous architectures, with a decentralized communication topology. The accuracy of dPerf has been studied on three applications : (i) the Laplace transform, for sequential codes, (ii) the NAS Integer Sort benchmark, for distributed MPI programs, (iii) and the obstacle problem, for the decentralized P2P computing and the scaling of the number of computing nodes.

Keywords: Reducing the slowdown, static analysis, automatic instrumentation, performance scaling, trace-based simulation, performance prediction, C, C++, Fortran languages, heterogeneous systems, decentralized environment, peer-to-peer HPC.

Introduction

Afin de mener à bien leurs travaux, les scientifiques ont besoin de capacités de calcul toujours croissantes. En effet, celles-ci leur permettent la résolution de problèmes de plus en plus complexes. C'est en réponse à cette demande que s'est formée la communauté des chercheurs en calcul distribué. L'idée en est simple, il s'agit de connecter plusieurs machines en parallèle afin de fournir une puissance de calcul supérieure à une machine seule. Il y a, bien évidemment, un lien fort entre l'application parallèle et l'architecture d'exécution. Ce lien entraîne la dépendance entre (i) le développement de nouvelles architectures, (ii) des pilotes de celles-ci, (iii) des systèmes d'exploitations qui tiennent compte des dernières architectures, (iv) et le développement de nouvelles applications adaptées aux architectures de calcul de dernière génération.

Il y a plusieurs décennies que les acteurs du milieu du *HPC* (High Performance Computing) s'intéressent aux différentes architectures de calcul parallèles et distribuées. L'avantage de ces systèmes est la disponibilité d'une importante puissance de calcul. Un exemple de système distribué est une grappe de calcul. Dans ce type d'architecture, une augmentation des performances peut se traduire par une augmentation de la précision et/ou un temps plus court d'accès au résultat du calcul. Les paramètres qui conditionnent les performances d'une application sont : (i) la puissance de calcul, ou le nombre et la puissance des nœuds de calcul, (ii) la taille de la mémoire disponible, (iii) la capacité de stockage, (iv) et les caractéristiques du réseau.

Une architecture *HPC* homogène et dédiée au calcul implique un haut coût d'investissement du point de vue de l'installation, de la maintenance et de l'utilisation (ou de l'exploitation). Contrairement à ce type d'architecture, un système partagé et hétérogène est disponible pour un coût plus modique. Un exemple d'architecture homogène est le super-ordinateur. Celui-ci est composé de plusieurs nœuds de calcul identiques connectés par un réseau de communication dédié. Le regroupement des ressources de différents types a généré des systèmes hétérogènes tels que la grille des postes de travail. Les nœuds de celle-ci sont connectés par un réseau pair-à-pair. Les architectures pair-à-pair connaissent depuis plusieurs années un intérêt croissant. En général, les systèmes homogènes sont stables, signifiant que les ressources sont fiables. La volatilité des ressources est une caractéristique des systèmes pair-à-pair (*peer-to-peer*), abrégé P2P.

Les modifications du nombre de nœuds ou de l'infrastructure réseau impliquent une adaptation des techniques de développement logiciel utilisées. Dans le *HPC*, ceci implique la modification des applications pour les adapter à la topologie de communication et à l'architecture cible.

Les performances d'une application *HPC* dépendent du système cible et de la façon dont le programme utilise ses ressources de calcul. La prédiction de performances est née afin d'évaluer les performances d'une application exécutée sur une architecture de calcul donnée.

L'existence d'un nombre important de types d'architectures de calcul a mené au développement de nombreuses méthodes et outils de prédiction des performances. Le développement de ces outils est adapté aux architectures cibles ou aux applications, et est donc conditionné par les besoins des scientifiques. Les outils de prédiction de performances sont destinés à donner une vue d'ensemble sur le comportement des applications parallèles et distribuées quand un ou plusieurs parmi les facteurs suivants ne sont pas connus ou disponibles :

- l'architecture de calcul cible est en cours de création ;
- la topologie réseau est peut-être indisponible, ou une nouvelle infrastructure réseau dont les performances rendues ne sont pas connues ;
- l'application est en cours de développement et les développeurs ont besoin d'étudier les performances du code développé.

L'utilité des méthodes de prédiction de performances est présentée dans la figure 1.

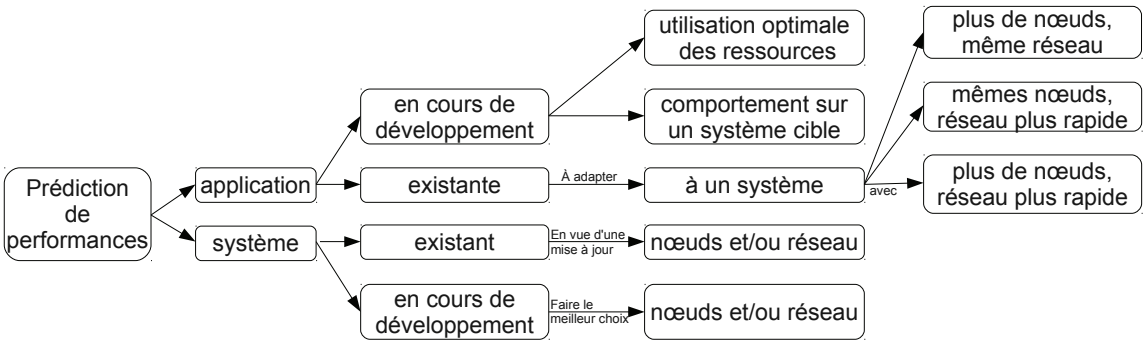


FIG. 1: L'intérêt des outils de prédiction de performances pour le développement et l'adaptation des application aux systèmes cibles, ainsi que dans l'évaluation des systèmes existants et futurs en vue d'augmenter leur puissance de calcul.

Les outils de prédiction de performances développés jusqu'à présent sont limités par un ou plusieurs des facteurs suivants :

- le *slowdown* (facteur de ralentissement) ;
- le support pour les systèmes P2P (décentralisés) ;
- le support pour de multiples langages de programmation *HPC* ;
- le support pour de multiples formalismes de communication ;
- le manque de support pour l'hétérogénéité.

L'efficacité d'un outil de prédiction de performances est donnée par le *slowdown* . Il est le facteur de ralentissement du processus de prédiction de performance par rapport à une exécution réelle. Autrement dit, le *slowdown* est défini comme :

$$slowdown_{global} = \frac{temps_{prediction}}{temps_{execution\,reelle}} \tag{1}$$

$$slowdown_{par\,processus} = \frac{temps_{prediction}}{temps_{execution\,reelle} \times Nombre\,processus} \quad (2)$$

Dans la majorité des cas, le $slowdown_{global}$ n'est pas utilisé, et toute utilisations du terme *slowdown* sans indice fera référence au $slowdown_{\ par\,processus}$. Étant donné une architecture de calcul "S", et une application "A" , un *slowdown* inférieur à l'unité montre le fait que l'exécution réelle de "A" sur le système "S" prend plus de temps que pour obtenir une prédiction de performance. D'une même façon, un *slowdown* supérieur à l'unité montre que l'outil prend plus de temps pour obtenir une prédiction que l'exécution complète de "A" sur la machine "S". Un *slowdown* égal à 100 se traduit par un temps d'obtention de la prédiction égal à 100 fois le temps d'exécution de "A", ce qui est très souvent une particularité des outils de prédiction. Jusqu'à présent, très peu d'outils offrent des prédictions avec un *slowdown* proche de 1.

Le développement des méthodes de prédiction de performances qui modélisent les *systèmes HPC P2P centralisés* a commencé il y a quelques années. Ces efforts, bien évidemment appréciés par la communauté de *HPC* , n'ont pas été suffisants car ils restent plusieurs aspects qui ne sont pas couverts. L'un des problèmes est le support de multiples langages de programmation utilisés couramment dans le domaine de *HPC* : Fortran, C, C++ ou Java. À l'heure actuelle il n'y a pas d'outil qui gère plus d'un ou deux langages. De nouveaux mécanismes de communication sont optimisés pour les réseaux P2P décentralisés qui, contrairement aux systèmes P2P classiques, communiquent de façon décentralisée. Ce type de développement implique le passage des formalismes de communication traditionnels tels que la norme Message Passing Interface (MPI) par exemple, vers ces nouvelles bibliothèques comme P2P Self Adaptive Protocol (P2P-SAP)[49], par exemple.

Malheureusement, jusqu'à aujourd'hui, les outils de prédiction de performances ne gèrent qu'un nombre limité de formalismes de communication.

L'inconvénient d'une majeure partie des outils de prédiction de performance est qu'ils ne prennent pas en compte l'*hétérogénéité* des systèmes. Si néanmoins un outil modélise les architectures de calcul hétérogènes, il n'adresse pas un ou plusieurs aspects présents ci-dessus.

À ce rythme et sans efforts pour améliorer le support pour les aspects mentionnés ci-dessus, les outils de prédiction risquent d'être en retard du point de vue du progrès technologique.

Comment prédire les performances des applications de calcul distribué dans des conditions réelles sur des systèmes hétérogènes P2P ou la communication a lieu de façon décentralisé ?

Afin de répondre à cette problématique, il faut impérativement répondre aux questions suivantes :

1. Quel est l'état de l'art des outils et méthodes qui prédissent les performances des applications distribuées et quels sont leur limitations ?

2. Si les outils et les méthodes qui existent ne sont pas suffisants pour rendre des résultats de prédiction dans les systèmes distribués pair-à-pair, quelle est la méthode qui pourrait faire la différence ?

La thèse que je défends est qu'il est possible de répondre aux questions présentées ci-dessus tout en ayant une grande précision et en gardant un *slowdown* le plus proche de 1 possible. Cet objectif s'est concrétisé dans une méthodologie de calcul de prédiction de performance pour les applications de calcul distribué dans des conditions réelles. Un premier pas vers ces conditions est de calculer des prédictions de performances précises en prenant en compte les optimisations au niveau du compilateur. Ceci implique une exécution réelle de l'application analysé, fait qui, en général, est problématique du point de vue du temps pour obtenir une prédiction et de son extrapolabilité. Pour améliorer ces deux aspects, cette thèse propose une méthodologie qui réduit le temps de calcul d'une prédiction ainsi que l'extrapolation de la prédiction de performances. L'économie de temps est obtenue grâce aux méthodes de *benchmarking* simple et *benchmarking* optimisé par bloc d'instructions. L'extrapolation des résultats expérimentaux est réalisée en utilisant un simulateur paramétrable permettant de (i) tester des différentes configurations réseau, et (ii) d'extrapoler le nombre de nœuds d'un système. La méthodologie issue de cette thèse est détaillée tout au long de ce document.

Ces nouvelles méthodes ont été intégrées dans un outil de prédiction de performances que nous avons appelé **dPerf** (distributed Performance prediction) et que nous proposons afin d'offrir à la communauté de *HPC* une solution aux défis actuels. Les caractéristiques principales de dPerf sont :

- Un temps réduit nécessaire à l'obtention du résultat de prédiction de performances grâce à la réduction du *slowdown* des applications évaluées ;
- La prise en compte des niveaux d'optimisation du compilateur ;
- Le support pour différents *langages de programmation HPC* ;
- Le support pour différents *formalismes de communication* standardisés, comme le MPI, ou non-standardisés, comme c'est pour le moment le P2P-SAP ;
- Le support pour des *systèmes hétérogènes* grâce (i) à la prédiction basée sur la réduction du *slowdown* , (ii) et à la simulation basée sur des fichiers de traces ;
- Le support pour les *systèmes P2P décentralisés* utilisés pour les applications de calcul haute performance.

Afin de calculer une prédiction de performances dans le temps le plus court possible, le rôle principal est attribué à la réduction du *slowdown* . En utilisant cette technique, il est possible d'obtenir très rapidement une prédiction pour les applications simples, ou, pour un code complexe, il est également possible de le simplifier tout en conservant sa fonctionnalité. De cette façon, la prédiction de performances basée sur une telle méthode se fait avec un *slowdown* réduit. A part ces avantages, une analyse statique fournit aussi les moyens d'extraire les éléments essentiels du code d'entrée afin de pouvoir mettre en place des interfaces avec d'autre mécanismes qui aident à la prédiction de performances.

L'outil que nous avons développé peut calculer la performance d'une application si elle est écrite en utilisant l'un des *langages le plus souvent utilisés en HPC* , parmi lesquels on retrouve C, C++ et Fortran. Le code d'entrée de dPerf est analysé, transformé selon le résultat souhaité et à la fin, rendu en tant que nouveau code parfaitement compilable, tout se faisant d'une façon entièrement automatique et sans l'intervention de l'utilisateur.

dPerf est conçu de manière à reconnaitre *plusieurs formalismes de communication* et pouvoir l'étendre par ajout de nouveaux formalismes. Généralement, une application destinée à l'HPC communique utilisant MPI. Cette norme a été conçue à l'époque du parallélisme sur des systèmes homogènes, mais le potentiel des systèmes P2P pose des nouveaux défis liés à la topologie réseau. Si des environnements de calcul ont été développés pour les topologies centralisées, ceci n'est pas le cas pour les topologies décentralisées.

Les chercheurs font des efforts pour développer et proposer la standardisation de nouveaux formalismes de communication. Nous développons donc l'outil dPerf afin qu'il accepte aussi bien les applications *HPC* écrites pour un système *homogène* ou *hétérogène*. Ces systèmes sont modélisés par dPerf en séparant une application *HPC* en deux parties : la communication et le calcul. Ce dernier est extrait du code d'origine et ses caractéristiques sont mises en valeur par une technique que nous avons appelée *benchmarking par blocs d'instructions*.

Les solutions de calcul *HPC* repartis, telles que le *P2P computing* ou le *cloud computing*, utilisent leurs propres bibliothèques de communication qui sont disponibles ni pour la communauté de *HPC* , ni pour les développeurs d'outils de prédiction de performances. Pour cela, plusieurs laboratoires français se sont réunis afin de développer (i) *P2P-SAP*, un formalisme de communication pour le *HPC* pair-à-pair décentralisé, et (ii) dPerf, un outil pour prédire les performances des application distribuées P2P qui utilisent P2P-SAP pour communiquer. Ces efforts seront diffusés publiquement afin de montrer leur applicabilité et leurs fonctionnalités.

Plan du document

La première partie présente les méthodes les plus pertinentes de visualisation et de prédiction de performances pour les systèmes et les applications parallèles ou distribuées. Ces méthodes ont été développées, en majorité, pendant cette dernière décennie. Selon la caractéristique dominante de chaque outil mentionné dans cette partie, nous allons classifier l'état de l'art des travaux de recherche et le développement dans :

1. des outils analytiques ;
2. des outils basés sur des mesures ;
3. des outils hybrides ;
4. des méthodes pour visualiser le comportement d'un système ou d'une application *HPC* .

Le chapitre 1 contient les techniques de prédiction de performances entièrement basées sur des formules mathématiques. En général, ces méthodes calculent la prédiction avec

une haute précision, mais le coût total pris par la conception du modèle mathématique est très important (élevé).

Dans le 2ème chapitre nous présentons les outils qui font une prédiction de performances en se basant sur des mesures prises soit au moment de la conception du modèle système-application, soit au moment de la résolution du modèle décrivant l'ensemble système-application.

Les méthodes hybrides sont énumérées dans le chapitre 3. Ce sont des techniques qui combinent d'une façon efficace les modèles mathématiques avec la prise des mesures, afin de fournir des prédictions plus précises et pour une plage des systèmes ou applications beaucoup plus répandue que celle des techniques précédentes.

Dans le chapitre 4 sont présentés tous les outils qui ne fournissent pas de prédiction de performances mais qui aident à la visualisation du comportement d'un système ou d'une application *HPC* .

La thèse défendue ici utilise une approche hybride qui sera présentée dans la troisième partie du document, avec une comparaison directe entre notre méthode et l'état de l'art des outils hybrides de prédiction de performances.

La deuxième partie de ce manuscrit contient la contribution amenée par cette thèse au domaine du calcul de haute performance. Nous présenterons les outils et les bibliothèques requises par notre outil dPerf. Les termes les plus utilisés dans la prédiction de performances avec dPerf seront définis. Une présentation détaillée sera faite sur la méthodologie de dPerf, ainsi que la façon dont celle-ci est implémentée.

Dans une première phase, nous présentons les outils nécessaires au développement et au fonctionnement de dPerf (voir le chapitre 5). Le but de dPerf est de fournir un outil de prédiction de performances adapté aux systèmes et aux applications *HPC* en général, et aux applications *HPC* P2P en particulier. Pour atteindre ce but, nous avons cherché les outils et les bibliothèques les plus adaptés. Le système d'exploitation utilisé est le GNU/Linux parce qu'il permet l'accès aux compteurs hardware, accès nécessaire à la prise de mesure des événements. Nous présentons l'utilité de la bibliothèque PAPI ainsi que les deux frameworks, c'est-à-dire ROSE Compiler et Simgrid, qui fournissent les méthodes nécessaires à la réduction du *slowdown* , aux transformations du code d'entrée, si besoin, et à la simulation de la communication dans le système *HPC* cible.

Le chapitre 6 introduit le lecteur à la méthodologie que nous proposons et que nous avons appliquée dans dPerf pour qu'il soit un outil adapté aux systèmes et applications *HPC* courantes. Nous expliquons l'approche de dPerf qui permet la réduction automatique du *slowdown* d'une application *HPC* distribuée, qui utilise une topologie logique décentralisée, conçue pour être lancée dans un environnement P2P. Ce chapitre présent également comment nos efforts de recherche sont présentés publiquement. Il s'agit de la façon dont les sources de dPerf sont accessibles pour tous les développeurs d'applications, ou de systèmes *HPC* . Nous indiquons la façon dont ceux intéressés peuvent accéder à la documentation leur permettant de bien installer et utiliser notre outil, y compris toutes les dépendances.

La troisième partie présente les résultats expérimentaux qui montrent la précision de l'outil dPerf. Deux étapes peuvent être identifiées :

1. l'analyse de trois applications différentes pour étudier la précision des prédictions de dPerf. Notre outil analyse une implémentation séquentielle de la transformée de Laplace, puis le *benchmark* IS de NAS écrit en C/MPI, et finissant par des expériences avec le problème de l'obstacle implémenté pour le réseau pair-à-pair décentralisé. Ces études montrent (i) la prédiction précise du temps de calcul d'un code séquentiel, (ii) la prédiction précise des performances d'un code distribué MPI, (iii) puis la prédiction des performances d'un code distribué P2P. Ces tests ont le but de montrer que l'erreur de la prédiction faite par dPerf reste acceptable.
2. une étude de cas présentant la prédiction de performance pour le code mathématique du problème de l'obstacle. Cette étape s'appuie sur la précision étudié à l'étape précédente pour montrer l'utilité de dPerf. Notre outil prédira les performances de l'application pour une topologie réseau potentielle, ou bien pour un nombre différent de nœuds de calcul. Les résultats sont obtenus pour un faible coût.

Première partie

État de l'art des outils de prédiction de performances

Introduction

Cette partie présente l'évolution de la recherche sur les outils qui aident l'optimisation des applications parallèles et distribuées, ainsi que les systèmes HPC qui veulent fournir une puissance de calcul augmentée pour un faible coût.

L'état de l'art des outils de prédiction de performances présente d'un côté les outils qui calculent et rendent une prédiction de performances, et de l'autre côté les méthodes d'extraction des caractéristiques les plus importantes d'une application ou d'une architecture de calcul haute performance. Ces dernières ne fournissent pas une estimation de performance, mais présentent seulement le comportement de l'application et du système pendant une exécution de l'application. Du point de vue de la thèse défendue ici, les outils qui fournissent le résultat final d'une prédiction de performances détiennent une place prioritaire par rapport aux méthodes de visualisation du comportement d'applications. Pour cette raison, les outils de la première catégorie bénéficieront d'une présentation plus détaillée dans les chapitres 1, 2, 3, suivis par une présentation, plus restreinte, des outils de visualisation, dans le chapitre 4.

Concernant les outils de prédiction qui seront présentés dans cette partie, il y a deux aspects à définir, qui vont permettre de classifier les méthodes développées ces dernières années :

1. La façon de modéliser le système et (ou) l'application. Afin de pouvoir donner des résultats sur le système et/ou sur l'application, l'outil de prédiction doit les "connaitre". Pour cela, avant de se prononcer sur les performances, il faut être en possession de modèles qui représentent le mieux possible l'architecture de calcul - nœuds de calcul plus infrastructure réseau - et le comportement de l'application. De cette façon, en modifiant à leur tour les caractéristiques les plus importantes du système -nombre de nœuds, type du réseau- ou de l'application -taille du problème, nombre de nœuds utilisés, type de communication synchrone ou asynchrone- on peut étudier l'impact de ces modification sur les performances. Pour la conception d'un modèle, de nombreuses techniques de modélisation existent parmi lesquelles :
   - des graphes déterministes ;
   - des graphes stochastiques ;
   - des réseaux de fils d'attente ;
   - des réseaux Petri.
2. La façon de résoudre le modèle du système ou de l'application pour donner des prédictions des performances. Pour estimer la performance d'une application ou la puissance de calcul d'un système HPC, ce sont les outils de prédiction de performance qui donnent le résultat. Ils utilisent des modèles mathématiques, des mesures cap-

turées au moment d'une exécution (partielle ou complète), ou des combinaisons modèle-mesure. La résolution de ceux-ci donne la prédiction de performances. Cette résolution peut être faite soit de façon analytique, soit par la simulation.

La suite de cette partie présente l'état de l'art des outils les plus pertinents pour la prédiction et la visualisation des performances, classifiés selon l'aspect dominant lui donnant la fonctionnalité (entre la technique de modélisation et la résolution du modèle).

Plusieurs chercheurs [104],[90],[31],[88] ont présenté à leur façon l'état de l'art des outils du domaine, en présentant les points forts et ceux faibles selon une classification différente de celle utilisée dans ce document. Dans leurs publications, les auteurs ont pris en compte des outils ou méthodes qui peuvent être moins pertinentes dans le cadre de la thèse défendue ici, et qui ne seront pas mentionnés dans notre classification sur l'état de l'art des outils. Nous avons choisi de grouper les outils de prédiction de performances selon la façon dont le modèle de l'application et du système ont été conçus, construits et résolus. Dans le cas des outils où la conception, la construction et la résolution sont faites l'une de façon analytique, l'autre basée sur de mesures, etc. l'outil est classifié selon l'aspect le plus pertinent.

Chapitre 1

Les outils analytiques de prédiction de performances

Introduction

Les outils analytiques de prédiction de performances sont entièrement mathématiques, qui définissent une architecture de calcul, qui est composée de machines - ou nœuds de calcul - et le réseau utilisé.

Même si ces outils sont très rapides grâce à une résolution analytique du modèle, leur précision est proportionnelle au coût d'obtention du modèle. La modélisation du système et de l'application implique la description détaillée de ceux-ci. Une haute complexité du modèle nécessite d'importantes ressources matérielles et humaines. Pour décrire un système ou une application dans le plus grand détail, le coût d'obtention du modèle est, généralement, élevé.

Ces méthode nécessitent une bonne compréhension du système ciblé et de l'application. Ceci implique donc que l'utilisateur apprenne les détails liés aux système et aux applications, tâche non-triviale qui nécessite du temps et des ressources. Comme les architectures de calcul gagnent très rapidement de la puissance et de la complexité, l'intérêt des scientifiques vers le développement et l'utilisation des méthodes purement analytiques, a baissé, lui aussi, rapidement. À partir de 2005, il existent encore très peu d'outils de prédiction de performances analytiques.

Parmi les méthodes analytiques les plus représentatives, on retrouve LogP, LogGP, LoGPC, RSIM, ainsi que les travaux de recherche effectués à : l'Université de Paderborn en Allemagne (Uni-Paderborn), l'Université de Chicago aux États-Unis (UCHICAGO), le Laboratoire National Los Alamos aux États-Unis (LNLA) et l'Université Technique de Delft aux Pays-Bas (TUDELFT).

1.1 LogP - 1993

LogP [45] est un outil pour modéliser les machines parallèles, le développement de cet outil se faisant à l'Université de Californie. Le but de LogP est d'offrir une base pour le développement des algorithmes parallèles rapides et portables, mais aussi pour assister et

guider les concepteurs de machines parallèles. LogP fonctionne avec des informations sur la bande passante, la puissance de calcul, la latence, et l'efficacité dans l'association entre le calcul et la communication.

Pendant la période de développement de LogP, les fabricants de super ordinateurs souhaitaient pouvoir envisager les performances des applications sur les futures architectures estimées être composées de plus de 1000 nœuds de calcul. Comme chaque nœud était prévu de contenir un processeur puissant et une mémoire de grande capacité, le point faible restait la connexion réseau qui avait une bande passante réduite et une latence importante. Pour ces raisons, LogP est apparu au moment idéal, avec une solution analytique de prédiction de performances des applications parallèles, qui doivent s'exécuter d'une manière synchrone sur une architecture de calcul haute performance homogène. Le nom de l'outil, LogP, signifie la prise en compte de la Latence, l'indisponibilité du processeur pendant l'envoi ou la réception des messages (overhead), de la bande passante (gap), et enfin, du nombre des modules Processeur/mémoire.

Culler et al.[45] ont développé et proposé cet outil analytique qui demande beaucoup de ressources afin de mettre en place le modèle du système et de l'application. Sa précision varie entre 14 et 95%, en fonction des modèles, ce qui à posé les bases de nouveaux outils mais qui aurait eu besoin de développements supplémentaires.

1.2 LogGP - 1995

Avec l'avancement technologique, Alexandrov et al. ont proposé dans [30] une version améliorée du LogP. Cette version apporte un nouveau modèle linéaire pour prendre en compte les messages de grande taille échangés entre les ressources de calcul employées par l'application évaluée. La lettre G majuscule représente, comme le g minuscule, la bande passante mais apporte plus de précision pour les messages d'une grande taille.

Les résultats expérimentaux réalisés par Alexandrov et al. montrent que la prise en compte des messages de grande taille, en plus des ceux gérés par LogP augmente la précision. En effet, LogGP donne des résultats de prédiction de performances avec une précision de plus de 92%.

1.3 Uni-Paderborn - 1996

Au Centre de Calcul Parallèle de Paderborn, Simon et al. ont développé un outil de prédiction de performances des architectures de calcul, guidés par la conviction que l'évaluation des performances est un aspect essentiel à chaque étape du cycle de vie d'un système [98].

La méthode proposée s'adresse aux systèmes homogènes qui ne peuvent plus être évalués d'une façon précise avec des outils tels que LogP [45]. LogP utilise un modèle matériel d'une complexité inférieure aux systèmes des années 96. De plus, les applications ayant un niveau de parallélisme élevé rendent cet outil inefficace. Motivés par la conception d'une nouvelle approche de prédiction de performances mieux adaptée aux architectures de l'année 1996, Simon et al. développent cette méthode analytique qui est basée sur trois

modèles : (i) un modèle pour le graphe de tâches, (ii) un modèle de file d'attente, et (iii) un modèle de la mémoire. La solution d'Uni-Paderborn est capable de gérer n'importe quel langage de programmation si les communications sont effectuées avec MPI et si les applications sont exécutées.

La concurrence des communications est décrite par le graphe de tâches, et chaque communication est caractérisée par une taille et une distance d'envoi. Le support multiprocesseur est modélisé en appliquant une technique de transfert de tâches à latence zero. La contention et la disponibilité des ressources est modélisée par un réseau de files d'attente. L'aspect séquentiel de l'application est modélisé à l'aide des modèles du processeur et de l'accès mémoire. La prédiction de performances utilisant la méthode de Simon et al. atteint une précision de 93,5 à 98%.

1.4 LoGPC - 1998

À partir de LogP et LogGP, Moritz et al. proposent LoGPC [77]. C'est une amélioration des travaux précédents avec la création du module gestionnaire de la contention réseau et du comportement des interfaces réseaux. Ces éléments additionnels donnent plus de précision dans l'évaluation des applications qui communiquent en échangeant des messages.

Moritz et al. ont comme but d'appliquer LoGPC pour l'identification du meilleur compromis entre l'échange synchrone ou asynchrone de messages. Les expériences relèvent que, pour une application utilisant le mode synchrone, LoGPC a une précision à la prédiction de 97%, comparé à LogP et LogGP qui rendent la prédiction respectivement avec 93 et 65% de précision pour une même application.

LoGPC à été validé sur une machine multiprocesseurs MIT-Alewife [29], avec les modèles correspondant aux applications Diamond DAG [82] et EM3D [35], l'échange de messages se faisant par des messages actifs (Active Messages) [116]. Diamond DAG est un programme scientifique utilisé pour la comparaison des chaines d'ADN. EM3D est une application développée à l'Université Berkeley (États-Unis) et écrite en Split-C. Elle modélise, selon les algorithmes présentés dans [73], la propagation des ondes électromagnétiques à travers des solides.

1.5 UCHICAGO - 2001

La méthode développée à l'Université de Chicago [94] utilise un modèle mathématique très complexe du réseau, des machines et de l'application.

Ripeanu et al. expliquent la façon dont la modélisation d'une application implique une étude de celle-ci en préalable. Cette étude est réalisée sur des systèmes de calcul homogènes et donne des résultats théoriques qui sont ensuite validés par des expériences. Après avoir étudié et modélisé l'application, une prédiction de performances est faite pour le cas où la même application s'exécute sur une grille de calcul, soit un système de calcul distribué composé de plusieurs sous-systèmes homogènes distribués sur plusieurs sites.

Les développeurs de cette méthode ont défini un seul modèle mathématique pour

valider leur approche. Ils ont modélisé le logiciel parallèle Cactus [94], qui est un cadre de calcul parallèle développé au sein du département d'informatique de l'Université de Chicago. Cactus était initialement un cadre pour le calcul des équations d'Einstein. La décision de modéliser uniquement l'application Cactus limite l'applicabilité de la méthode proposé par Ripeanu et al.

Les résultats expérimentaux sont obtenus au coût d'un *slowdown* réduit car pour obtenir une prédiction, Ripeanu et al. n'ont pas la contrainte d'exécuter l'application. La méthode est efficace et sa précision est de plus de 95%. Le point faible de cette méthode analytique reste le cout très élevé pour pouvoir développer le modèle mathématique du système et de l'application.

1.6 LANL - 2001

Kerbyson et al. proposent dans [66] une méthode de prédiction de performances et de modélisation de l'évolution (*scalability modelling*) pour les applications à grande échelle. Cette méthode utilise un modèle mathématique pour l'application évaluée et un autre pour l'architecture de calcul cible. Son but est d'offrir une vue générale sur les performances de l'application modélisée, sur le système cible, et de révéler les causes des éventuelles congestions.

Sachant qu'une méthode analytique est dépendante des caractéristiques de l'application et de l'architecture de calcul modélisée, les chercheurs du Laboratoire National Los Alamos (LANL) adressent uniquement SAGE, une importante application ASCI [24]. Le système ciblé est une architecture parallèle et homogène de calcul haute performance. La méthode de Kerbyson et al. donne des résultats de prédiction avec une précision de 89-95%. Même si le coût de conception des modèles de l'application et du système est élevé, les avantages se trouvent dans le *slowdown* réduit et le fait qu'un modèle pourrait, éventuellement, être crée pour d'autres applications ou d'autres façons de communiquer.

1.7 TUDELFT - 2003

À l'Université Technique de Delft, une méthode analytique de prédiction de performances a été développée. Elle est présentée par Van Gemund et al. dans [111] comme une méthode très efficace dans sa consommation de ressources. Le résultat de la prédiction est très précis dès les premières étapes dans le processus de conception des programmes parallèles. Du point de vue de Van Gemund et al., la prédiction de performances est un outil d'ingénierie qui fournit un retour important sur les choix de conception concernant la synthèse d'un code ou le développement de la machine.

L'équipe de chercheurs de TUDELFT proposent cette méthode analytique pour prédire les performances d'applications parallèles qui communiquent en utilisant MPI, quel que soit le langage de programmation utilisé. Ces applications sont conçues pour fonctionner d'une manière centralisée sur des architectures de calcul haute performance homogènes. Étant une méthode analytique, l'obtention de la prédiction prend beaucoup moins de temps qu'une exécution complète du code évalué, fait qui implique une valeur très réduite

du *slowdown* . Les expériences faites par Van Gemund et al. montrent une précision de la méthode de 50 à 90%, mais le coût pour obtenir la prédiction reste encore assez important.

Cette méthode développée à TUDELFT est basée sur le formalisme de modélisation PAMELA (Performance Modeling Language)[110], ayant comme but de minimiser le coût pour obtenir la prédiction tout en augmentant l'efficacité.

Vue comparative

Les outils qui analysent et prédisent les performances d'une application parallèle ou distribuée et qui utilisent une approche entièrement analytique ont été développés avec succès jusqu'en 2003. Pendant cette période, ces outils ont représenté d'une façon très précise les performances d'une application parallèle. L'activité de développement la plus importante se trouve dans la conception des modèles mathématiques décrivant le système *HPC* , l'infrastructure du réseau utilisé entre les machines -ou les nœuds- de calcul, ainsi que la description du comportement de l'application à analyser. À cause des importants sauts technologiques dans le domaine du calcul haute performance, ces méthodes analytiques ont commencé à demander trop de ressources - d'analyse du système et de l'application à modéliser, et de développement du modèle mathématique - par rapport au résultat fourni. Autrement dit, ce n'était plus efficace de concevoir un modèle - chose très laborieuse- pour chaque application différente, ou bien pour une nouvelle architecture de calcul ou réseau. Les chercheurs se sont rendu compte de l'inefficacité des outils de prédiction analytique face à la quantité impressionnante de moyens de calcul de plus en plus complexes et variés qui devenaient disponibles sur le marché. Face à la difficulté de l'utilisation des méthodes analytiques de prédiction de performances, les outils basés sur des mesures ont commencé leur développement.

Le tableau 1.1 présent une vue comparative de principaux outils de prédiction de performances analytiques et de dPerf, l'outil développé afin de soutenir cette thèse. La méthodologie utilisée pour dPerf et l'implémentation sont expliquées dans le chapitre 6. Nous mentionnons que dPerf n'est pas un outil analytique, mais hybride.

À part l'outil de l'UCHICAGO, toutes les méthodes ci-dessus ne proposent que des solutions adaptées aux systèmes homogènes. La plupart des architectures de calcul d'aujourd'hui ne sont plus homogènes, mais très souvent distribuées sur des sites géographiquement éloignés et composées de différents éléments. Dans ce contexte, un outil analytique a peu de possibilité de survie et ne pourra être appliqué que pour des grappes de calcul situées sur un seul site. La topologie logique des applications de calcul haute performance est, en général, centralisée, c'est à dire que les tâches à effectuer par chacun des nœuds participants sont distribuées par un nœud central et le résultat est récupéré, toujours par ce nœud central, à la fin du calcul. L'intérêt des systèmes actuels est de pouvoir donner des résultats de prédiction pour une application qui emploie la communication décentralisé, comme c'est le cas de l'architecture P2P (pair-à-pair). Jusqu'à maintenant, il n'y a pas eu de solution non-commerciale disponible afin que le milieu scientifique puisse développer de vraies applications P2P décentralisées, et donc pas d'outil pour pouvoir prédire les performances de ce type d'application. En parlant des applications analysées par les outils analytiques du Tableau 1.1, seul l'approche développé à l'Uni-Paderborn

peut analyser une application distribuée, le reste étant limité aux applications parallèles. Du point de vue des langages et bibliothèques de communication utilisés, Uni-Paderborn, RSIM et TUDELFT peuvent prédire les performances des applications quel que soit le langage, communiquant en MPI ou en utilisant ILP (*instruction-level parallelism*). Les autres méthodes modélisent uniquement une application telle que SAGE, CACTUS, Diamond DAG, Sweep3D ou encore FFT, fait qui limite considérablement leur applicabilité dans le domaine de *HPC* . Le *slowdown* de RSIM est le plus important, et la plupart des outils, étant analytiques, comportent un *slowdown* inférieur à l'unité car une fois le modèle conçu, l'évaluation de performances prend beaucoup moins qu'une exécution normale de l'application analysée. Comme mentionné au début, les outils analytiques fournissent les prédictions de performances avec une précision dépassant facilement 90%, mais le coût pour obtenir cette prédiction - soit le temps pour concevoir le modèle nécessaire - reste assez important et parfois constituant un vrai inconvénient.

TAB. 1.1: Les caractéristiques les plus importantes des outils analytiques de prédiction de performances, et les caractéristiques de l'outil dPerf développé durant ce travail de recherche. (1)Prendre en compte le niveau d'optimisation du compilateur ; (2) La communication ; (3)Voir chapitre 6 pour la méthodologie et l'implémentation de dPerf ; (4)Le coût de la résolution du modèle ;

Méthode ou Lab	Année	Système cible	Topologie logique	Code d'entrée accepté			Optim.(1)	Valeurs		
				Type	Comm.(2)	Langage		Slowdown ou ralentissement	Précision [%]	Coût de prédiction(4)
dPerf(3)	2011	hétérogène	P2P décentralisée	distribué	P2P-SAP, MPI, autre	C, C++, Fortran	OUI	0,2 - 1,1	> 80	˜1
TUDELFT	2003	homogène	centralisée	parallèle	MPI	tous	NON	«1	50-90	»1
LANL	2001	homogène	centralisée	parallèle	SAGE (benchmark ASCI)		NON	<1	89-95	»1
UCHICAGO	2001	hétérogène	centralisée	parallèle	CACTUS		NON	<1	>95	très haut
LoGPC	1998	homogène	centralisée	parallèle	Diamond DAG, EM3D		NON	«1	>78	»1
Uni-Paderborn	1996	homogène	centralisée	distribué	MPI	tous	NON	«1	93.5-98	»1
LogGP	1995	homogène	centralisée	parallèle	Sweep3D		NON	«1	>92	»1
LogP	1993	homogène	centralisée	parallèle	FFT, Linpack LU		NON	«1	14-95	»1

Chapitre 2

Les outils de prédiction de performances basés sur des mesures

Introduction

Les outils de prédiction de performances de cette catégorie sont basés soit sur des informations obtenues suite à une exécution partielle ou complète de l'application analysée, soit sur un historique de l'exécution de l'application dans différentes conditions environnementales. En général, dans cette catégorie il n'y a pas d'outils basés sur un modèle mathématique. Si un modèle mathématique est utilisé pour modéliser ou pour prédire les performances, il ne représente pas la partie la plus importante de l'outil.

2.1 P^{3}T - 1993

Fahringer et al. présentent dans [54] l'outil P^{3}T (*Parameter-based Performance Prediction Tool*). Cet outil fait partie du système VFCS (*Vienna Fortran Compilation System*). P^{3}T modélise les applications écrites en Fortran et compilées avec VFCS. Au moment de la compilation, P^{3}T fournit des informations de prédiction qui peuvent servir aux développeurs à améliorer leur code source. P^{3}T ne prédit pas le temps d'exécution, mais donne des détails sur les paramètres clés de l'application. Ceux-ci sont classifiés en (i) distribution des tâches, (ii) coûts de communication, et (iii) traces des données à travers la mémoire cache. Une version améliorée de P^{3}T est proposée dans [53]. L'outil est doté d'un mécanisme pour identifier si l'un des processeurs est chargé du calcul outil ou des tâches inutiles.

P^{3}T modélise les systèmes distribués multi-processeurs et les applications parallèles, avec un intérêt particulier pour les paramètres des applications. Une interface graphique permet la visualisation des paramètres des applications qui peuvent influencer les performances d'exécution. La contention réseau ainsi que le nombre de défauts de cache sont calculés de façon statique, les autres paramètres nécessitant une exécution du code source. À cause de cette exécution, le ralentissement (*slowdown*) de cette méthode est égal à 1, et toute analyse supplémentaire augmentera encore plus cette valeur. Pourtant, la précision de P^{3}T à la prédiction est de 98%, le processus de prédiction n'étant pas très coûteux.

2.2 USC-UC - 1996

Une collaboration entre l'USC (*University of Southern California*) et l'UC (*University of California*) est présentée par Saavedra et al. dans [95]. Une technique est proposée pour modéliser, indépendamment du système, l'exécution des applications séquentielles. Ce modèle contient les caractéristiques de l'application ainsi que celles du système. À l'aide des caractéristiques mentionnées, Saavedra et al. montrent l'efficacité de leur technique dans la prédiction du temps d'exécution sur une combinaison aléatoire d'application et de système. La méthode proposée identifie les instructions clés du code source dont leur modification augmente l'efficacité de l'application analysée.

L'approche proposée par Saavedra et al. modélise les applications séquentielles, et leur méthode est une inspiration pour l'outil *ChronosMix* (voir la section 3.3) qui applique aussi un principe basé sur un modèle d'exécution indépendant de l'architecture de calcul.

2.3 NWS - 1999

Un effort commun des Universités de Tennessee, Washington et California, est présenté dans [118]. Un outil appelé *Network Weather Service* (NWS), a été proposé pour prédire l'utilisation des ressources de calcul distribuées pour un ensemble d'architectures de calcul.

Les prédictions obtenues avec NWS peuvent être utilisées afin de mieux ordonner l'utilisation des ressources d'une architecture de calcul cible. Les systèmes qui peuvent être analysés sont les grappes et les grilles de calcul. Ces systèmes peuvent être homogènes ou hétérogènes, sachant que la prédiction se fait pour le même système que celui utilisé pendant la modélisation.

Wolski et al. ont développé une méthode de prédiction des caractéristiques de performance qui changent dynamiquement. NWS adresse les architectures de calcul uniquement, les applications n'étant pas prises en compte. NWS contrôle et fournit dynamiquement des prédictions en fonction du comportement des ressources réseau et calcul. L'outil est composé de deux parties : une pour la capture et la supervision, et l'autre pour la prédiction. La modélisation des ressources est faite en prenant des mesures de performance du CPU, de la mémoire et du réseau (latence et bande passante). Ces caractéristiques sont stockées dans une base de données afin de constituer un historique qui servira à mieux décrire le système analysé. Dans la deuxième partie de NWS, les caractéristiques accumulées pendant la première phase sont utilisées pour faire différentes prédictions de performances de l'architecture de calcul distribué antérieurement analysée. Toutes les fonctionnalités de NWS sont accessibles à travers l'*API* crée pour ce but. L'*API* est utilisable en C, C++ ou Fortran, les langages les plus utilisés dans le domaine du *HPC*, permettant ainsi l'intégration de la prédiction de performances avec NWS directement dans les applications HPC. Même si NWS est un outil de prédiction de performances de l'architecture de calcul, nous pouvons quand même nous prononcer sur une valeur du *slowdown* . La méthode proposée par Wolski et al. rend la prédiction, selon les graphiques présentés dans [118], avec une haute précision, mais cette prédiction est obtenue suite à l'historique d'utilisation des ressources système. Ce fait relève la nécessité d'au moins une exécution d'une application. NWS nécessite au moins une exécution complète, ceci se reflétant dans un

slowdown supérieur à 1, mais s'il s'agit d'une application très lourde -avec une exécution durant plusieurs jours- alors il est possible d'obtenir le minimum nécessaire à la prédiction bien avant la fin d'une exécution, soit pour un *slowdown* inférieur à l'unité.

2.4 ARM - 1999

Une méthode issue de la collaboration entre UCSD (*University of California San Diego*) et UTK (*University of Tennessee Knoxville*) est présentée par Faerman et al. dans [52]. ARM (*Adaptive Regression Modeling*) constitue une approche de prédiction des performances des applications orientées vers le calcul et exécutées sur des grilles *HPC* .

La prédiction de la durée de transfert de données dans une application de calcul intensif est utilisée dans les environnements distribués aux utilisateurs multiples. Cette prédiction utilise le *benchmarking* du réseau disponible dans l'outil NWS (voir la section 2.3). Dans ARM, des méthodes statistiques de régression sont utilisées pour améliorer les performances d'une application sur un système cible.

Faerman et al. présentent une méthode qui fournit des prédictions avec une précision de plus de 88% sur deux applications : SARA et SRB.

2.5 FAST - 2003

La bibliothèque FAST (*Fast Agent's System Timer*)[93] est développée à l'ÉNS-Lyon (France). Cet outil a pour but de fournir des informations sur les besoins des routines et sur la disponibilité de la plate-forme de calcul. FAST est basé sur NWS (voir la section 2.3). L'outil développé à l'ÉNS-Lyon par Martin Quinson utilise pour ces prédictions des prédictions obtenues par NWS concernant la disponibilité des ressources d'un système. L'utilisation de la bibliothèque FAST dans des différents cadre, permet d'utiliser ses prédictions pour (i) l'ordonnancement des routines séquentielles sur une grille de calcul, (ii) pour visualiser l'état courant d'une plate-forme, (iii) pour vérifier les performances d'une code pour un jeu particulier de données, ou (iv) pour l'étude du traitement des routines parallèles.

La structure de la bibliothèque FAST contient deux parties. L'une de celles-ci est chargée de l'identification automatique des besoins des routines sur chaque machine, et l'autre partie fournit des prédictions interactives. Ces prédictions sont utilisées par les application clientes de FAST.

Les prédictions fournies par FAST, qui tiennent compte des besoins des routines ainsi que des information sur la disponibilité des ressources, avec la charge utilisable par l'ordonnanceur, se concrétisent dans une prédiction du temps de calcul t_{calcul}. Ce temps ne prend pas en compte des charges extérieures, ni les machines indisponibles. L'auteur de la bibliothèque montre que l'utilisation de FAST implique un surcoût inférieur a 1%, dans le cas d'une interrogation avec un default de cache, ou 4000 fois plus rapide, sans default de cache. L'erreur de la prédiction avec FAST est, en général, inférieure à 16%, et dans certain cas restreints elle pourra atteindre 35%.

2.6 PEVPM - 2003

Dans le cadre de sa thèse sur la prédiction de performances des applications parallèles [62], Duncan A. Grove a développé en 2003, un système de modélisation appelé PEVPM (*Performace Evaluating Virtual Parallel Machine*). PEVPM était conçu pour être extrêmement précis, facile à utiliser et avec un faible coût pour obtenir la prédiction. Cet outil est composé de modèles décrivant chacun des événements de calcul et de communication. Ces modèles sont élaborés à partir des dépendances de données, les niveaux de contention et la répartition de la performance parmi les opérations bas-niveau. Concernant la résolution du modèle qui vient d'être crée, la répartition de la performance de chaque modèle est échantillonnée selon la méthode *Monte Carlo*, réussissant ainsi de prendre en compte l'effet de la contention.

La première phase dans la prédiction calculée avec PEVPM est la conception du modèle. L'application à analyser nécessite une instrumentation manuelle, fait qui implique la compréhension du code à instrumenter. Le modèle n'a pas de connaissance sur les aspects liés à la mémoire et à la topologie réseau. Celui-ci constitue un point faible de l'outil. PEVPM utilise la prise de mesures pour modéliser le système et l'application, ensuite il utilise ce modèle pour obtenir la prédiction de performances. Du point de vue du travail de recherche effectue par D. A. Grove, nous avons classifié son travail comme faisant partie des méthodes basées sur des mesures, car c'est cet aspect qui représente la partie dominante de son travail. Le code qui sera modélisé et analysé est impérativement une application parallèle écrite en quel que soit le langage de programmation procédural -le C, C++ et Fortran faisant partie- tant qu'il communiquent en MPI. L'architecture de calcul est caractérisée par le processeur uniquement.

Pendant la deuxième phase, la résolution du modèle, Grove a utilisé des informations statistiques acquises avec l'outil MPIBench -développé par Grove et al. et présenté pour la première fois dans [61]- pour modéliser l'infrastructure réseau.

Avec PEVPM, D. A. Grove réussit à proposer une méthode assez efficace de prédiction de performances nécessitant un effort important pour la conception du modèle, mais ayant un *slowdown* inférieur à l'unité grâce à la technique de modélisation basée sur des mesures, et à l'évaluation très rapide du modèle. Malgré la précision de 95% présentée par D. A. Grove sur l'implémentation C/MPI de l'algorithme Jacobi, un point faible de PEVPM reste la dépendance entre la méthode de prédiction et la plateforme utilisée pendant la conception du modèle. Si une modification se produit dans le système cible, un nouveau modèle doit être créé. De la même manière, si une modification se produit au niveau de l'application analysée, elle doit être, à nouveau, instrumentée manuellement.

2.7 APAPS - 2004

APAPPS est un outil de prédiction de performance pour les applications parallèles qui communiquent en utilisant MPI. L'approche présentée par Khan et al. dans [67] est inspirée de l'outil Paradyn (voir la section 4.1). APAPS modélise les grappes de calcul homogènes, ayant un seul cœur par processeur et par nœud .

Le principe utilisé dans APAPS est de diviser l'application analysée dans une partie

de calcul et une partie de communication MPI. Pour mesurer le temps total de calcul pris par les lignes d'instructions, une méthode de *benchmarking* de celles-ci est utilisée. Pour les communications, le *benchmarking* de toutes les fonctions MPI est fait, suivi par le *benchmarking* de la bande passante et de la latence. Khan et al. modélisent les performances d'une application en tenant compte de la contention de communication. Le temps total d'exécution de l'application analysée est donné par :

$$t_{total} = t_{calcul} + t_{communication} = \sum_{i=1}^{Nb\ operations} (C_i \times N_{op(i)}) + \sum_{i=1}^{N_c} (\alpha + \frac{\lambda \times B_i}{\beta}) \qquad (2.1)$$

où C_i est le coût associé à chacune des opérations, $N_{op(i)}$ est le nombre d'itérations d'une même opération i pendant l'exécution du programme, N_c est le nombre de communications par processus, α représente la latence, λ = nombre de processus, et représente le taux de contention, B_i est la taille du message i, β est la bande passante. Ce modèle dépend de l'architecture de calcul.

Même si les prédictions faite avec cette méthode sont susceptible d'être influencées par la latence du réseau, la précision dans l'évaluation des performances d'une application reste un point fort de cette approche.

2.8 NCSU - 2005

Yang et al. proposent dans [124] une méthode qui utilise des résultats obtenus d'une exécution partielle sur un système hôte, pour faire une prediction sur un système cible.

Cette approche développée à l'Université de North Carolina (NCSU), montre que la prédiction relative des performances entre deux plateformes *HPC* est observée avec précision sans qu'une exécution complète de l'application soit nécessaire. Yang et al. prouvent qu'une exécution partielle suffit pour prédire avec précision le comportement d'une exécution complète.

Cette technique ne modélise pas l'application, elle n'analyse pas le code source, et elle ne simule pas l'architecture de calcul. Cette méthode utilise la prédiction basée sur la prise partiale de mesures et l'observation du comportement pendant ce temps. Des expérimentations d'une courte durée sont faites sur des multiples systèmes cible potentiels. Dans une première phase, une prédiction du $t_{predit_{execution}}$ est faite sur une architecture cible. Ceci est une combinaison entre :

- les performances de l'application calculées sur un système de référence ;
- les performances de l'application calculées sur un système cible. Ces performances sont obtenues en fonction du rapport entre l'architecture de référence et celle cible. Sur le système cible, une exécution partielle est requise.

Cette approche a une précision réduite dans le cas des applications dont le comportement change de façon dynamique. La méthode proposée par Yang et al. perd en précision quand la taille du problème ou le nombre de nœuds du système varient entre le moment de l'obtention des valeurs de référence et le moment de la prédiction. La méthode proposé par Yang et al. est utilisée pour toutes les applications parallèles, sans contrainte de bibliothèque de communication ou langage de programmation. Les systèmes ciblés sont des

architectures parallèles sans mention sur le dégrée d' homogénéité. Grâce à l'exécution partielle, le ralentissement de cette méthode est fortement réduit. La précision varie de 62 à 94% et le coût pour obtenir la prédiction reste acceptable. À part le temps pour modéliser l'architecture de référence, le processus de prédiction prend moins de 1% du temps total d'exécution pour rendre le résultat.

2.9 GPRES - 2005

Le GPRES (*Grid performance PREdiction System*) [69] est une méthode de prédiction de performances des systèmes qui est développée dans le cadre des projets SGIgrid [25] et Clusterix [3]. GPRES ne modélise pas les applications, et les systèmes concernés sont des grilles de calcul de haute performance.

GPRES sert à fournir des informations liées à l'utilisation des ressources d'un système. La prédiction est faite en fonction d'un historique d'utilisation des ressources par une application. Même si GPRES ne prédit pas les performances de l'application, il dépend de celle-ci afin de fournir la prédiction du système. Cette prédiction est basée sur des groupes de similarités. GPRES estime qu'une certaine tâche, antérieurement observée, prendra le même temps pour être exécutée pour quelle que soit l'application qui la contiendra. L'architecture de GPRES sépare le processus d'acquisition de celui de prédiction.

L'approche utilisée dans GPRES a le but d'augmenter l'efficacité du mécanisme de distribution de ressources dans un système *HPC* . Une telle augmentation conduit à une meilleure performance de l'application et du système utilisés. Kurowski et al. ont validé dans [70] l'efficacité de GPRES pour les grilles de calcul. Le coût pour accéder à une prédiction faite par GPRES est élevé car la méthode proposée par Kurovski et al. utilise un historique d'exécutions, ceci impliquant plusieurs exécutions complètes d'une application. Le coût de cet apprentissage est amorti si un grand nombre de prédictions sont basées sur le même historique.

2.10 UPB-2009

Dans les systèmes distribués de grande taille, la gestion des ressources devient difficile à cause des fortes variations dans les paramètres qui définissent certaines composantes. Dans [100], une plateforme de prédiction des performances est présentée. Son but est d'augmenter l'efficacité de la gestion des ressources. Cette plateforme est une partie composante du système MonALISA [9] et elle permet d'employer l'un parmi plusieurs algorithmes de prédiction de performances prédéfinis, ou bien de définir de nouveaux algorithmes.

Le but de cette méthode est d'être évolutive (EN : *scalable*), flexible et facile à utiliser. Pour atteindre ces objectifs, elle est conçue en deux parties, une partie intégrée dans monALISA et une autre partie qui exécute l'algorithme de prédiction.

Cette plateforme rend le résultat de prédiction de performance en utilisant une méthode basée sur un historique de l'exécution des applications. Comme c'est une prédiction qui nécessite l'exécution du code, ce modèle fait partie de la catégorie des outils de type *profiling*, ou basés sur des mesures.

Cette méthode permet de faire une prédiction de performances uniquement pour le système, et non pas pour l'application. L'analyse de prédiction prend en compte les aspects liés au CPU, liens de communication et l'accès mémoire. Toutes les applications sont acceptées, quel que soit le langage de programmation et la façon de communiquer utilisés pour les développer.

La méthode proposée par Stratan et al. donne une prédiction des performance du système analysé avec un *slowdown* supérieur à 1, une précision comprise entre 70 et 80%, et un coût, supposé faible, pour obtenir la prédiction.

2.11 Leiden - 2010

Minh et al. présentent dans [76] leur activité de recherche à l'Université de Leiden (aux Pays-Bas). L'approche proposée par Minh et al. est basée sur un historique d'exécution des applications et de l'utilisation des ressources du système. Les limitations de cette méthode sont celles de l'historique. L'application, le langage et l'architecture utilisés dans la création de l'historique représentent les caractéristiques des applications ou des systèmes à modéliser. La méthode proposée par Minh et al. modélise les applications sur des systèmes parallèles de *backfilling* en vue d'un meilleur ordonnancement des tâches.

Le ralentissement de cette méthode est donné par la durée d'obtention de l'historique. Quand un historique existe, le ralentissement est inférieur à un. Dans certains cas, il n'y a pas d'historique (de l'exécution), et ceci implique au moins deux exécutions. Ceci représente un ralentissement supérieur ou égale à 2. L'approche développée a l'Université de Leiden donne des prédiction avec jusqu'à 32% plus de précision que d'autres techniques similaires mentionnées par Minh et al. dans [76]. Le coût pour accéder au résultat de la prédiction varie en fonction de l'existence ou non d'un historique d'exécution.

2.12 PAS2P - 2010

Pour caractériser le comportement et prédire les performances des application parallèles, l'outil PAS2P (*Parallel Application Performance Prediction*) est développé à l'Université Autonoma de Barcelone (en Espagne). La présentation et l'évolution du développement de PAS2P sont présentées par Wong et al. dans [119, 120, 121, 122]. L'approche de Wong et al. est de décrire une application à partir de son comportement, puis analyser les communications implémentées dans le code source de l'application pour extraire uniquement les phases pertinentes de la communication.

PAS2P demande qu'un code source soit soumis à une instrumentation, à une exécution sur une machine locale, et à une exécution du résultat des deux étapes antérieures à celle-ci pour donner une prédiction de performances. Cet outil modélise les applications parallèles qui communique en utilisant une bibliothèque d'échange de messages telle que MPI. Ces applications doivent être écrites en C, en C++ ou en Fortran, et exécutées sur des systèmes de calcul homogènes. Le choix des langages est limité par la bibliothèque MPE qui ne gère pas d'autres langages a part ceux-ci.

L'instrumentation du code source est faite avec la bibliothèque MPE. Cette étape

implique d'exécuter le code source pour générer des fichiers de trace, d'identifier les événements de communication, puis d'identifier des blocs d'instructions étendus. La deuxième étape est l'obtention d'un modèle de l'application qui soit indépendant de l'architecture de calcul. Inspirés de l'algorithme de Lamport, Wong et al. implémentent l'obtention d'un modèle en utilisant une horloge globale et le *benchmarking* des blocs d'instructions qui sont parallélisés. La troisième étape modélise les blocs pertinents, c'est-à-dire, les blocs d'instructions qui sont parallélisés et dont la pertinence est donnée par :

$$pertinence_{bloc\,instructions} = cout_{bloc\,instructions} \times t_{execution_{bloc\,instructions}} \tag{2.2}$$

La quatrième étape correspond à la construction de la signature de l'application. Celle-ci nécessite une re-exécution du code source instrumenté. La prédiction est obtenue pendant la dernière étape quand la signature de l'application est exécutée sur l'architecture cible. Les architectures modélisées avec PAS2P devront être homogènes, une exécution de la signature est demandée pour chaque type de système différent. La formule selon quelle est calculé le résultat de la prédiction est :

$$t_{predit} = t_{instrumentation} + t_{obtention\,modele} + t_{execution\,signature} + \sum pertinence_{blocs} + t_{re-execution}; \tag{2.3}$$

Les prédictions de PAS2P ne sont pas évolutives. Si un nombre N des nœuds est utilisé pour modéliser l'application et pour obtenir la signature de celle-ci, la prédiction sera faite pour un nouveau système de N nœuds de calcul.

Le ralentissement de PAS2P par rapport à une exécution complète d'un code source est de deux, car l'outil nécessite minimum deux exécutions du code source sur une machine locale avant de pouvoir prédire les performance sur une autre architecture. La précision à la prédiction est de plus de 97%, avec un coût important pour obtenir ce résultat.

2.13 NCSU (Wu) - 2011

Les chercheurs à l'Université de North Carolina (NCSU) proposent dans [123] une méthode pour faciliter l'analyse des application parallèles. Étant donnée une application parallèle, Wu et al. ont développé une approche qui permet de générer un *benchmark* de performances qui sera spécifique à cette application passée en entrée. Le but d'un *benchmark* et de pouvoir permettre aux scientifiques de réaliser des prédictions de performances qui adressent directement leurs applications.

Sachant que généralement les outils de prédiction de performances sont développés en se basant sur un *benchmark* général, il y a un risque que ces outils ne puissent pas représenter certains aspects essentiels des applications scientifiques qui seront analysées avec ces méthodes. C'est pour ceci que l'équipe des chercheurs à NCSU propose une méthode de création de *benchmark* bien spécifique à quelle que soit l'application d'entrée.

Même si il ne s'agit pas tout à fait d'une méthode de prédiction de performances, elle en est très proche, et elle contribue visiblement à la prédiction de performances. C'est pour cela que NCSU justifie sa place parmi les méthodes basées sur des mesures.

Wu et al. expliquent qu'une application parallèle de calcul haute performance, écrite dans n'importe quel langage de programmation, est reçue en entrée et un outil développé

à NCSU, ScalaTrace (voir la section 4.11), effectue l'analyse du code, exécute le code, puis rend des traces d'exécution. Ces traces contiennent les temps de calcul des blocs séquentiels, et les paramètres de communication, et ils ont un avantage de taille par rapport aux autres mécanismes générateurs de traces : les traces sont évolutives (peuvent être extrapolées). Les paramètres de communication sont extraits des appels MPI, l'interface d'échange des messages, utilisée par les processus employés par l'application pour communiquer. Les traces sont écrites dans un langage propriétaire de NCSU. Ces traces servent d'entrée pour coNCePTuaL, un générateur de benchmaks écrit en C/MPI, lui aussi développé à NCSU, qui va générer un *benchmark* spécifique à l'application analysée antérieurement avec ScalaTrace, et qui va utiliser les traces pour remplacer le comportement réel de l'application. De cette façon, le *benchmark* gardera toutes les caractéristiques de l'application initiale, mais d'une manière très compacte.

La méthode proposée par Wu et al. est appliquée avec un *slowdown* supérieur à l'unité, car ScalaTrace nécessite au moins une exécution complète de l'application MPI analysée afin de générer les traces. Avec une précision de plus de 78%, cet approche a un faible coût d'analyse car le *benchmark* rendu en sortie décrit l'application initiale sans éléments dynamiques, ceux-ci étant remplacés par des traces extensibles (*scalable*). Selon les auteurs, ils proposent une méthode inédite qui permet de générer un *benchmark* à partir d'une application MPI.

Vue comparative

Dans le tableau 2.1 nous présentons les outils de prédiction de performances basés sur des mesures, décrites dans ce chapitre. Ceux-ci peuvent utiliser aussi des modèles analytiques mais il n'utilisent pas des simulateurs. L'outil issu de cette thèse, dPerf, est aussi mentionné dans le tableau pour pouvoir comparer ses caractéristiques avec celles des outils basés sur des mesures. Nous mentionnons que dPerf n'est pas un outil basé sur des mesures, mais il est un outil hybride.

Un nombre assez réduit des outils de ce type a été développé dans les années 90, mais contrairement aux méthodes du chapitre 1, ceux basés sur des mesures ont continué d'être proposés. Parmi ceux-ci, près de la moitié modélisent les architectures hétérogènes. Le type de communication préféré par les développeurs reste le MPI, à l'exception de NWS, GPRES, UPB et Leiden. L'avancement technologique entre 1990 et 2011 impose la proposition des approches qui modélisent des multiples langages de programmation tels que le C, le C++ et le Fortran ou le Java. Cet intérêt des scientifiques pour les outils plus génériques de prédiction de performances indique la présence des contraintes d'ordre économique. Les outils de prédiction suivent le degré de complexité des application et proposent des résultats précis et compétitifs. Nous pouvons traduire leur caractère compétitif par le faible coût de prédiction par rapport à l'investissement requit par l'optimisation des applications. La ré-conception des application peut être envisagé, par certains, comme une solution à une meilleur utilisation des ressources, soit à une augmentation de la puissance de calcul. Contrairement à cette idée, l'état de l'art des outils basés sur des mesures montre que de nombreuses méthodes donnent des prédictions avec un *slowdown* inférieur à 1. Ces valeurs indiquent que des alternatives d'architecture de calcul peuvent être tes-

tés plus rapidement que toute optimisation d'application analysée. En contre-partie, la préparation qui précède la prédiction peut, pour la plupart des outils dans cette catégorie peut devenir un point négatif qui les caractérisent. Cet aspect reste moins perturbant que dans le cas des outils analytiques. Du point de vue de l'efficacité des outils présentés ici, nous constatons que la majorité ont un ralentissement proche de la valeur de 1 et leur précision est élevée, avec une exception qui est l'approche *Leiden*.

Les méthodes de prédiction de performances basées sur des mesures fournissent le résultat avec plus de rapidité et pour un moindre coût par rapport aux outils analytiques. Pourtant, le désavantage majeur des approches présentées dans ce chapitre est leur dépendance de l'architecture modélisée, ainsi que de la disponibilité physique de celle-ci.

TAB. 2.1: Les caractéristiques les plus importantes des outils de prédiction de performances basés sur des mesures, et les caractéristiques de l'outil dPerf. (1)Prendre en compte le niveau d'optimisation du compilateur ; (2) La communication ; (3)Voir chapitre 6 pour la méthodologie et l'implémentation de dPerf ; (4)La topologie logique de communication ; (5)Le coût de la résolution du modèle ;

Méthode ou Lab	Année	Système cible	Topologie logique(4)	Code d'entrée accepté			Optim.(1)	Valeurs		
				Type	Comm.(2)	Langage		Slowdown ou ralentissement	Précision [%]	Coût de prédiction(5)
dPerf(3)	2011	hétérogène	P2P décentralisée	distribué	P2P-SAP, MPI, autre	C,C++, Fortran	OUI	0,2 - 1,1	> 80	˜1
P^{3}T	1993	homogène	centralisée	parallèle	MP	Fortran	non	<1	98	>1
USC-UC	1996	-	-	séquentiel	sans	Fortran, autre	non	<1	75-95	-
NWS	1999	hétérogène	décentralisée	distribué	-	C,C++, Java	non	≥ 1 ou <1	haute	-
ARM	1999	homogène	centralisée	distribué	SARA, SRB		non	non précisé	88	non précisé
FAST	2003	hétérogène	décentralisée	distribué	-	Tout	non	«1	<16	faible
PEVPM	2003	homogène	centralisée	parallèle	MPI	C,C++, Fortran	non	<1	95	»1
APAPS	2004	homogène	centralisée	distribué	MPI	C,C++	non	>1	85	non précisé
NCSU	2005	homogènes	centralisée	distribué	Tout	Tout	non	<1	62-94	>1
GPRES	2005	hétérogène	centralisée	-	-	-	-	-	efficacité prouvée	élevé
UPB	2009	hétérogène	centralisée	distribué	-	-	-	»1	70-80	<1
Leiden	2010	-	-	-	-	-	-	<1, ou ≥2	32	variable
PAS2P	2010	homogène	centralisée	parallèle	MPI	C,C++, Fortran	non	>2	97	élevé
NCSU (Wu)	2011	Tout	centralisée	parallèle	MPI	Tout	non	>1	< 78	faible

Chapitre 3

Les outils hybrides de prédiction de performances

Introduction

Dans cette catégorie, nous mentionnons tous les outils pertinents de prédiction qui utilisent une approche mixte pour obtenir le modèle de l'application ou du système, ou pour le résoudre. Ces outils ont un intérêt particulier car la complexité des architectures et des systèmes de nos jours sont modélisés avec moins d'effort par rapport aux outils analytiques - qui nécessitent des connaissances avancées sur le système et l'application- et avec une résolution plus rapide des modèles par rapport aux outils basés sur des mesures -grâce à la simulation complète ou basée sur des traces. Les outils dans cette catégorie possèdent une meilleure capacité d'extrapolation des résultats par rapport aux outils présentés dans les chapitres 1 et 2.

3.1 Dimemas - 1996-2004

Un outil de prédiction de performances pour les applications communicant en utilisant MPI a été proposé par Labarta et al. dans [71]. Cet outil, nommé Dimemas, fonctionne à base de traces d'exécution. Le résultat de la simulation est une trace de visualisation qui sert d'entrée pour l'outil Paraver [71]. Labarta et al. présentent Dimemas comme un simulateur de machines à mémoire distribuée. L'approche de Labarta et al. suit la reconstruction sur un système cible du comportement de l'application analysée. À la sortie de Dimemas, ce comportement est donné sous forme d'un nouveau fichier de trace qui peut être visualisé avec Paraver.

Le développement de Dimemas a été actif jusqu'au début des années 2000 [34] [33]. Les nouvelles versions de Dimemas ont étendu le support de l'outil aux applications MPI ou OpenMP exécutées sur des grilles de calcul. Le choix du langage de développement est limité aux trois langages qui sont les plus utilisés dans le domaine de l'*HPC* , c'est-à-dire le C, le C++ et le Fortran. L'approche de Dimemas a évolué depuis [71]. Le code source analysé est instrumenté d'une façon automatique à l'aide de l'outil VampirTrace [86]. Le résultat de l'instrumentation est un fichier en format OTF (*Open Trace Format*) [84].

Ce fichier OTF ainsi qu'un fichier de configuration de la plateforme cible sont les deux éléments requis en entrée par Dimemas. Avec ceux-ci, Dimemas lance une simulation et donne le résultat de prédiction.

Les inconvénients de l'approche utilisée dans Dimemas sont liés au contenu des fichiers de traces (OTF). Ces fichiers ne contiennent pas les informations nécessaires à une prédiction pour :

- une taille du problème différente de celle utilisée pour l'obtention des traces ;
- un nombre de processus parallèles différent de celui utilisé pendant l'instrumentation avec VampirTrace.

La précision à la prédiction de Dimemas est d'environ 80% mais le ralentissement du simulateur par rapport a une exécution complète reste important. L'utilisation de VampirTrace pour instrumenter et obtenir les traces permet à Dimemas de générer les fichiers en format OTF soit sur une machine soit sur l'architecture cible. Le premier choix implique un ralentissement de plusieurs fois le temps d'exécution réelle, et la deuxième solution est plus rapide mais elle implique la disponibilité de l'architecture de calcul ciblée.

3.2 MPI-Sim - 1998

MPI-SIM [91] est une bibliothèque pour faire des simulations parallèles basées sur l'exécution (*execution-driven parallel simulations*). Cet outil modélise les applications MPI. *MPI-SIM* est développé à partir de l'outil MPI-LITE pour assurer la parallélisation de la simulation. *MPI-SIM* applique une technique avancée de synchronisation des messages échangés entre les nœuds . Le processus de prédiction des performances se fait en fonction de deux caractéristiques de l'architecture de calcul : nombre de nœuds , et la latence des messages.

L'efficacité de l'outil proposé par Prakash et al. est montrée par les prédictions faites pour les *benchmark* de NAS. La précision de *MPI-SIM* est de plus de 80%, et le résultat est rendu rapidement grâce au *speedup* important. Ce *speedup* varie en fonction du problème traité pouvant aller jusqu'à douze fois le temps d'une exécution normale.

3.3 ChronosMix - 2000

La recherche de Saavedra, qui proposait dans [95] une méthode de prédiction de performances des applications séquentielles, basée sur un modèle indépendant de l'architecture de calcul, était reprise et élargie à l'Université de Franche-Comté par Bourgeois et al. [37],[38] pour les applications parallèles et réparties.

L'outil développé, appelé ChronosMix, adresse les applications parallèles écrites en C qui échangent des messages en utilisant MPI. L'une des nouveautés proposées dans ChronosMix est la prédiction de performances des applications qui sont destinées non pas aux systèmes homogènes -les plus utilisés jusqu'à l'an 2000- mais aux systèmes repartis hétérogènes. Pour ce type d'architecture de calcul qui implique une complexité élevée des nœuds et du réseau, très peu des études étaient proposées à la fin des années 90. Les auteurs de ChronosMix proposent un outil dont les résultats de prédiction sont extrapolables.

Ceux-ci sont possibles grâce à la technique de *micro-benchmarking* des instructions.

ChronosMix a été développé pour fournir des prédictions de performances qui aident à la conception et à l'optimisation des applications parallèles et reparties. L'outil combine la modélisation avec la prise de mesures et la simulation. ChronosMix fait des prédictions en utilisant l'analyse statique d'un code d'entrée. À la base de l'analyse statique se trouve l'environnement Sage++.

ChronosMix modélise l'architecture parallèle de calcul et le programme (en C/MPI). Cette modélisation est basée sur une technique de *micro-benchmarking*, soit le *benchmarking* des instructions élémentaires. Le *micro-benchmarking* implique la mesure du temps d'exécution de chacune des 177 instructions basiques reconnues par ChronosMix, qu'il classifie en trois catégories : (i) les opérations sur les types de données, (ii) les structures de contrôles et les fonctions systèmes, et (iii) les fonctions mathématiques. Dans ChronosMix, l'identification des instructions élémentaires nécessite la présence (ou l'existence) de l'architecture cible. La suite de cette identification ne requiert plus l'architecture cible et aucune exécution n'est plus lancée sur cette architecture. À part le *micro-benchmarking*, ChronosMix fait des simulations de calcul et de communication, il présente les résultats d'une façon graphique, et enregistre les détails de chaque prédiction dans une base de données en vue de leur réutilisation.

L'approche de ChronosMix rend des prédictions pour un *slowdown* très réduit -inférieur à l'unité en moyenne car seul un jeux de 177 instructions est exécuté une fois-, avec peu des contraintes par rapport aux outils *HPC* de l'année 2000. Bourgeois et al. ont développé un outil de prédiction qui nécessite un exemplaire de nœud de l'architecture cible et non pas le système *HPC* complet. Pour élargir ce type de prédiction, à partir des modèles obtenus pour ChronosMix, de nouvelles architectures de calcul -non disponibles- peuvent être crées par transformation.

3.4 PACE - 2000

Une méthode de prédiction de performances destinée au cycle de vie d'une application -conception et implémentation- a été développée au sein du département d'informatique de l'Université de Warwick (en Angleterre). L'outil développé à Warwick [83] est une hiérarchie de modèles de performance définissant une architecture de calcul en fonction de ses composantes logicielles et matérielles, et les caractéristiques de parallélisme. Cet outil se nomme PACE (*Performance Analysis and Caracterisation Environment)*) et il fournit des informations sur le temps d'exécution, l'utilisation des ressources, et l'aspect évolutif de l'architecture de calcul ou de l'application. Le but de PACE est de fournir des informations sur les performances tout en conservant la précision du système et de l'application.

L'outil développé par Nudd et al. aide à la conception et au dimensionnement des systèmes *HPC* , en fonction des performances obtenues pour les applications destinées à l'exécution sur les systèmes concernés. Deux types d'analyse sont disponibles dans PACE : (i) la pré-implémentation, et (ii) la post-implémentation. Les développeurs de PACE envisagent même une analyse en temps-réel grâce au coût très faible de PACE pour fournir une prédiction.

L'approche de l'outil à Warwick utilise les mesures et la modélisation des performances. PACE combine avec succès le *micro-benchmarking* avec l'historique de l'utilisation des ressources système pour obtenir un modèle analytique système-application d'une complexité variable. L'accès au résultat de la prédiction faite par PACE a un faible coût, mais l'obtention du modèle matériel plus logiciel reste assez élevé. L'analyse de performances faite par PACE a une précision de 90-95 %. L'approche est basée sur un nombre restreint des modèles -des logiciels et des composantes matérielles- et des mesures pour modéliser le parallélisme.

PACE adresse uniquement les applications MPI/PVM quelles que soient les architectures de calcul. Les prédictions de PACE trouvent une utilisation fréquente dans le cycle de vie d'une application, à partir de la conception et jusqu'à la maintenance.

3.5 POEMS - 2000

POEMS (Performance Oriented End-To-End Modeling System)[28] est un projet très ambitieux de l'Université d'Illinois car c'est à la fois une méthode de prédiction de performances analytique et hybride.

Adve et al. proposent POEMS en tant qu'environnement de modélisation de performances pour les systèmes parallèles et distribués complexes. Si POEMS a pour but la modélisation des architectures de calcul, ceci est atteint en utilisant une approche analytique, basée sur des mesures, ou hybride. L'approche analytique de POEMS est basée sur l'outil LogP [46], tout en tenant compte des améliorations présentées dans LogGP [101] et LoPC [56]. Cet approche peut être employée avec succès pour toutes les applications parallèles, quel que soit le langage utilisé dans le développement, et avec le coût de développement du modèle. La partie dans POEMS qui est chargée de la modélisation hybride utilise l'un des trois simulateurs : SimpleScalar pour modéliser le processeur et la hiérarchie mémoire, PARSEC pour la simulation des interconnexions réseau, ou MPI-SIM - un simulateur des systèmes parallèles de grande échelle. La prédiction qui utilise MPI-SIM sera limitée à l'analyse des applications écrites en C ou Fortran.

POEMS sauvegarde les résultats de la modélisation et de l'évaluation dans une base de connaissances. Pour tout modèle, l'intervention d'un analyste de performances est requise. C'est à sa charge de donner la charge et l'architecture des systèmes ciblés.

Les applications parallèles analysées avec POEMS utilisent obligatoirement MPI pour l'échange de messages. Le choix du langage de programmation utilisé pour le développement de l'application analysée est limité à Fortran, car POEMS utilise un générateur de graphes des taches (*Task Graph Generator*) capable d'interpréter seulement ce langage. Cette décomposition en graphe des taches est inspirée par LoPC et LogGP, fait qui ralentit considérablement la modélisation d'une application.

Adve et al. ont développé un outil qui calcule la prédiction de performances d'un système pour un coût du *slowdown* inférieur à l'unité, avec une précision qui varie sur une plage assez large, en fonction de l'approche choisie. Malgré les bons résultats de ce projet, le coût pour obtenir une prédiction reste élevé à cause de l'effort de développement du modèle mathématique. POEMS n'a plus bénéficié de développement après l'an 2000 principalement à cause du manque de support pour les outils qui font partie du projet.

3.6 BigSim - 2004

Zheng avec son équipe de chercheurs à l'Université d'Illinois ont présenté dans [128] un outil de prédiction de performances de systèmes homogènes *HPC* contenant un grand nombre de processeurs. Leur outil se nomme *BigSim* et il donne des prédictions pour des super-ordinateurs tels que le BlueGene/L. Les prédictions sont basées sur des exécutions complètes d'une application sur une architecture de calcul existante. Pendant la phase de simulation, BigSim peut, soit utiliser des fichiers de traces, soit paralléliser les événements d'une application. Dans les deux cas, BigSim prédit les performances des applications parallèles écrites dans n'importe quel langage, la communication se faisant en utilisant MPI ou Charm++. Le simulateur modélise les systèmes homogènes multi-processeurs, et trois catégories d'applications : (i) celles linéaires déterministes, (ii) celles déterministes basées sur l'échange des messages, et (iii) celles parallèles non-linéaires. Les prédictions faites par BigSim ont une précision de maximum 94%, mais le résultat est obtenu avec un ralentissement important, car l'exécution complète de l'application analysée est requise.

Le choix des langages de programmation et des formalismes de communication limite l'applicabilité de BigSim dans le contexte *HPC* d'aujourd'hui.

3.7 POSE - 2005

Un environnement de simulation parallèle orienté-objet (POSE)[117] est le résultat de la recherche menée à l'Université d'Illinois (à *Urbana-Champaign* aux États-Unis) dans le domaine de la prédiction de performances. POSE est développé pour prédire les performances des applications dans des architectures de calcul parallèles et homogènes. Le but des prédictions faites avec POSE est de pouvoir mieux choisir la configuration de nouveaux systèmes ou d'indiquer les endroits dans une application qui peuvent être modifiés pour augmenter ses performances.

POSE est un environnement qui utilise une technique de simulation réseau en parallèle, avec un intérêt particulier pour la contention. Le langage utilisé pour le développement de POSE est *Charm++*, un langage de programmation parallèle dérivé du C++. L'application analysée doit être parallèle et peut être écrite en Charm++ ou en tout autre langage si la communication est faite en MPI. Le système cible est une grappe de calcul -et donc homogène- avec support pour les nœuds de calcul multi-processeurs. Pour la prédiction de performances, POSE fait appel à l'émulateur BigSim, présenté antérieurement dans cette section. BigSim est chargé de la mise-à-disposition d'un environnement d'exécution parallèle de l'ordre des Peta-Flops. Avec le système virtuel crée par BigSim, POSE ajoute son propre mécanisme de simulation des topologies réseau. L'activité de recherche à l'Université d'Illinois se concentre sur la simulation des infrastructures réseau en étudiant l'impact de la contention dans les systèmes tels que BlueGene.

Le résultat de la prédiction avec POSE est obtenu avec un *slowdown* très réduit. La précision à la prédiction peut aller jusqu'à 90%, mais cette valeur peut varier en fonction de la complexité du modèle de système. En ce qui concerne le coût pour accéder à une prédiction, POSE est très coûteux car beaucoup des ressources -temporaires, matérielles et humaines- sont affectées à la création du modèle système et aux modifications de

l'application avant le processus de simulation.

3.8 P2PPerf - 2006

Parmi le nombre restreint d'outils de prédiction qui modélisent les architectures *HPC* pair-à-pair, il y a P2PPerf [50, 51]. Celui-ci est développé au LIFC (en France) par Ernst-Desmulier et al. qui proposent un outil qui simplifie l'étude et l'évaluation du comportement des applications distribuées, exécutées sur un système P2P. Ce système se nomme JNGI[112], il est basé sur le réseau virtuel JXTA[108] et il permet l'exécution des applications parallèles sur un nombre important de pairs. P2PPerf permet de récréer un scénario si celui-ci pourra contribuer à l'optimisation des performances, qu'il s'agit du comportement de l'application ou du chargement du système (*system load*). Ernst-Desmulier et al. évoquent l'utilité de P2PPerf dans la simulation des scénarios ou le nombre de nœuds , ou pairs est évolutif. Ainsi, une étude peut être faite sur l'impact de différentes configurations du système cible sur les performances de celui-ci.

Cet outil requit en entrée deux paramètres : (i) un code source, et (ii) les caractéristiques de l'architecture cible. Le code source est analysé de façon statique an utilisant l'outil *RECODER*[32], qui contient des fonctions pour transformer une application Java. Deux méthodes pour mesurer le temps de calcul de l'application, sans la communication, sont disponibles. L'une nécessite l'exécution complète de l'application, et l'autre exécute que les instructions basiques une seule fois. La communication est simulée avec NS2 [11], et le résultat de la prédiction est, enfin, calculé par P2PPerf à partir des résultats intermédiaires fournis par le simulateur réseau, les caractéristiques du système cible, et les temps de calcul du code source séquentiel.

La précision à la prédiction est, dans le cas de P2PPerf, répartie en calcul et communication. L'outil a une précision de 85% pour le calcul séquentiel, et une précision de 95% dans la prédiction du temps de communication.

3.9 Susukita - 2008

Dans [103], Susukita et al. proposent une méthode de prédiction basée sur (i) la modélisation des performances séquentielles sur un processeur, et (ii) la simulation à macro-niveau (*macro-level simulation*) des applications. Les systèmes ciblés sont du type homogène.

Les étapes dans la prédiction de performances avec cet outil sont les suivantes :

1. la modélisation séquentielle des blocs de calcul. Cette étape nécessite soit des simulations au niveau du processeur, soit des exécutions réelles ;
2. le remplacement des blocs de calcul par leur performances. Cette étape est critique pour l'applicabilité de la méthode. Si des dépendances de données empêchent le remplacement des blocs par leur temps de calcul, la méthode proposée par Susukita et al. ne peut pas être appliquée et la prédiction ne peut pas être faite.
3. la modélisation des performances des communications. Un *benchmarking* des communications MPI est faite sur le système cible pour fournir au simulateur, dans

l'étape suivante, un modèle de communication de l'architecture cible ;

4. un squelette décrivant l'application et son comportement est généré à partir du modèle de l'application ;
5. la simulation dans BSIM[103] du squelette de l'application obtenu précédemment. BSIM prend en compte le modèle de communication du système cible pendant la simulation. Le temps prédit est obtenu selon la formule :

$$t_{predit} = \sum_{i=1}^{Nb\,blocs} (t_{calcul_{bloc\,i}}) + t_{latence\,memoire} + t_{communication} \tag{3.1}$$

La méthode présentée par Susukita et al. donne les prédictions avec une précision supérieure à 80%. Cette approche est très rapide, la simulation étant obtenue avec une accélération de plus de 39 fois par rapport à l'exécution réelle. Cette accélération de la simulation, nommée *speedup* montre le faible coût pour accéder la prédiction.

3.10 PSINS - 2009

Les chercheurs à San Diego Supercomputer Center, aux États-Unis, ont proposé dans [107] un outil efficace, précis et flexible de modélisation et prédiction de performances. Cet outil s'adresse aux applications MPI, et il fournit des traces d'événements de calcul et de communication. Ces traces sont stockées d'une manière compacte et soluble (ou malléable). Cet outil développé par Tikir et al. se nomme PSINS (*PMaC's open source Interconnect and Network Simulator*) et ses résultats sont accessibles rapidement, ils sont évolutifs et extensibles. PSINS est conçu avec un API pour les différents outils de visualisation et simulation -tels que MPIDtrace, Dimemas, TAU ou Vampir- basés sur des fichiers de trace.

L'approche de PSINS consiste en un générateur de traces -nommé *PSINS tracer*- et un simulateur -*PSINS Simulator*. *PSINS tracer* instrumente une application MPI et génère des traces compactes et solubles. Ces traces, ainsi que la configuration des machines et le modèle de communication sont utilisées dans *PSINS Simulator*. La configuration ou l'architecture de calcul sont définies dans une classe C++. Un modèle d'architecture contient le détails d'ordonnancement des événements, de l'utilisation des ressources, et du temps de computation des événements. Chaque architecture configurée peut contenir un modèle de contention parmi ceux pré-définis dans PSINS. Le simulateur est jusqu'à 7 fois plus rapide qu'une exécution complète de l'application originale, et la prédiction de PSINS est faite avec une précision superieure à 83%.

PSINS est un outil de prédiction rapide et efficace qui fournit des résultats uniquement pour les applications communicant en MPI. Il contient un nombre généreux de modèles d'architectures et de communications, avec la possibilité d'ajouter des nouveaux sans un effort considérable. L'utilisation des traces pour la phase de simulation réduit beaucoup le temps pour accéder à une prédiction, fait qui représente un avantage de cet outil face aux autres activités de recherche du même domaine.

3.11 WARPP - 2009

Le simulateur WARPP (*WARwick Performance Prediction*)[63] est un outil qui modélise les performances des applications parallèles d'une complexité élevée, destinées au domaine scientifique ou industriel. Cet outil est développé à l'Université de Warwick (en Angleterre) par Hammond et al.

L'approche utilisée dans WARPP est basée sur la prise des mesures pour l'application et le système, et la simulation du réseau. Cet outil prédit les performances des applications parallèles communicant en utilisant MPI, celles-ci étant destinées à l'exécution sur des systèmes homogènes. Les architectures modélisées par WARPP sont du type grappes de calcul multi-cœur. WARPP est basé sur le *framework* PACE (voir la section 3.4), et il est développé en Java. Tout comme PACE, l'outil présenté par Hammond et al. utilise la technique de simulation des événements distincts (*Discrete Event Simuation*, ou DES) pour la modélisation d'une application et la façon dont celle-ci utilise les ressources du système. Les étapes dans la prédiction de performances de WARPP sont :

1. l'analyse et l'instrumentation automatiques d'un code source ;
2. la prise de mesures sur la machine à modéliser. Cette prise de mesures implique l'exécution du code source instrumenté pour obtenir les temps de computation, suivi par l'exécution d'un *benchmark* MPI.
3. le simulateur des événements distincts (DES).

WARPP est capable de prendre des mesures sur un nombre réduit de processus parallèles -tel que 4 processus- et de prédire les performances pour un système composé de jusqu'à 65000 nœuds de calcul.

Le ralentissement de l'outil développé par Hammond et al. est supérieur à l'unité car une exécution du code instrumenté est requise. La précision de WARPP est de plus de 90% avec un effort réduit pour modéliser l'application et le système.

3.12 PHANTOM - 2010

Il est un outil qui modélise les applications MPI écrites en Fortran. Proposé par Zhai et al. dans [126], PHANTOM utilise une technique qui sépare le calcul et la communication. Le temps de calcul du code séquentiel est obtenu après une exécution complète de celui-ci, et il est enregistré dans un fichier de trace. Pour mesurer les communications, l'outil développé par Zhai et al. applique une technique nommée FACT[127] pour générer des traces des événements MPI. Les traces de calcul et de communication sont passées en entrée du simulateur SIM-MPI[109] qui donne le résultat de la prédiction.

Les applications modélisées avec PHANTOM sont destinées à l'exécution sur des systèmes hétérogènes, mais l'approche était validée pour ceux homogènes uniquement. Dans le cas des grappes de calcul, le système cible entier n'est pas requis pour la prédiction, mais qu'un seul nœud qui fera partie de celui-ci. Avec ce nœud , PHANTOM initialise la première phase de la prédiction, soit la mesure du $t_{calcul\ sequentiel}$ du code source. Dans cette phase, le compilateur Open64[13] est utilisé. Même si celui-ci gère plusieurs langages de programmation -Java, C, C++, Fortran- le choix du langage est limité par l'outil FACT,

et par la maturité du projet PHANTOM , celui en dernier acceptant que le Fortran. La deuxième phase, celle de la capture des paramètres de communication, est suivie par le démarrage du simulateur qui calcul le temps d'exécution

La précision présentée par Zhai et al. pour les architectures homogènes est supérieure a 83%. PHANTOM est un outil où les trois phases du processus de prédiction s'enchainent avec très peu d'intervention de l'utilisateur. Une prédiction sur un système homogène a un coût faible, mais celui-ci peut beaucoup augmenter pour les architectures hétérogènes, où le code séquentiel doit être exécuté sur chaque type de nœud différent qui sera contenu par le système.

3.13 MPI-PERF-SIM - 2011

Cet outil est un effort commun de l'École Supérieure de Sciences et Techniques de Tunis (en Tunisie), et du Laboratoire d'Informatique de Grenoble (en France). Achour et al. présentent dans [27] l'outil MPI-PERF-SIM pour la prédiction de performances des applications parallèles MPI exécutées sur des systèmes hétérogènes. MPI-PERF-SIM est une approche basée sur la modélisation analytique, l'analyse statique du code, et la prise de mesures sur les nœuds de calcul et le réseau. MPI-PERF-SIM modélise les applications parallèles MPI destinées à des systèmes hétérogènes tels que les grappes de calcul. Le code source à analyser doit impérativement être écrit en C.

MPI-PERF-SIM implique une phase qui se déroule avant la prédiction -nommée phase d'installation- et une autre au moment du lancement de la simulation. Pendant la phase d'installation, MPI-PERF-SIM découvre les composantes de la plateforme, ainsi que les caractéristiques de l'application. L'application est analysée à l'aide de l'outil FACT [127] qui décompose un code source en représentation intermédiaire en utilisant WHIRL. Même si WHIRL gère les langages C, C++, Java et Fortran, MPI-PERF-SIM n'offre aucun support pour un autre langage que le C. Le résultat de l'analyse sont des fichiers de trace contenant des événements de calcul et de communication, sans aucune indication sur le temps de ceux-ci. Le processus d'analyse statique du code est suivi par une modélisation des performances du système. Cette modélisation nécessite la prise de mesures des instructions séquentielles calculées par le kernel du système ainsi que les caractéristiques du réseau - obtenues grâce aux outils de *benchmarking* . La modélisation des architectures de calcul cibles requit :

- un historique de l'application analysée. Celui-ci doit être acquis pour différentes tailles de l'application ;
- un polynôme décrivant la variation du temps d'exécution pour l'historique des différentes tailles de l'application ;
- les fichiers de trace obtenus antérieurement ;
- les performances du kernel des nœuds ;
- les caractéristiques réseau.

Après avoir obtenu toutes les informations nécessaires à la construction du modèle système, la phase de la prédiction est initialisée. Celle-ci fait la correspondance entre le modèle du système et les événements dans les fichiers de trace. La formule implémentée dans

MPI-PERF-SIM pour obtenir le temps prédit est :

$$t_{predit} = t_{communication} + t_{computation} + t_{attente} \tag{3.2}$$

MPI-PERF-SIM est uniquement un simulateur qui est chargé de faire cette correspondance correctement. Les fonctionnalités de MPI-PERF-SIM sont similaires avec celles du module MSG de Simgrid (voir la section 5.5).

L'efficacité de MPI-PERF-SIM est prouvée à travers les expériences présentées par Achour et al. Cet outil donne des prédictions pour un ralentissement réduit car le processus de prédiction ne nécessite pas l'exécution du code source. La précision à la prédiction est de minimum 87%. Le coût pour accéder aux prédictions faites par MPI-PERF-SIM est variable car une étape préparatoire avant le processus de prédiction requit un historique d'exécution.

3.14 SMPI - 2011

Dans [41], Clauss et al. présentent SMPI (*Simulated MPI*), un outil de simulation *on-line*, soit l'exécution d'une application sur un système réel où la communication à lieu dans un simulateur et non pas entre des nœuds réels. SMPI fait partie de SIMGRID [40] (voir la section 5.5), un *framework* pour le développement des simulateurs de systèmes hétérogènes.

SMPI simule les communications de façon rapide et évolutive. L'outil contient aussi un nouveau modèle linéaire pour l'échange de données entre les nœuds . Une autre caractéristique de SMPI est qu'il permet de simuler un système *HPC* de grande-échelle à partir d'un seul nœud réel. Ceci est atteint grâce à la technique de réduction du temps de calcul de l'application ainsi que l'empreinte de l'utilisation de la mémoire. Dans SMPI, les développeurs ont implémenté la modélisation de la congestion réseau d'une façon analytique. La précision à la prédiction est validée par rapport à l'exécution réelle sur une grappe de calcul, et puis le caractère évolutif du résultat est montré, avec une erreur de 18%.

La précision des prédictions faites par SMPI est, en moyenne, entre 71 et 90%, en fonction du type des communication utilisée dans le code source. L'approche proposée e par Clauss et al. fournit la prédiction plus rapidement que le temps prédit, où la simulation est plus rapide que le temps simulé. Pour SMPI, il ne s'agit pas d'un *slowdown* du simulateur par rapport au temps d'exécution réelle, mais d'un gain qui est proportionnel à la taille des messages échangés. Le coût pour accéder à une prédiction de SMPI est faible car la méthode ne nécessite ni des exécutions antérieures, ni la modification du code source.

Vue comparative

L'effort de développer des outils hybride de prédiction de performances a commencé à la fin des années 90. Depuis, un nombre très important d'outils de ce type ont été proposés, mais la généralité de ceux-ci reste à analyser. Nous avons réuni dans le tableau

3.1 les caractéristiques des outils les plus pertinents qui sont basés sur une approche mixte de prédiction de performances, y compris dPerf, l'outil issu de cette thèse.

La complexité des architectures de calcul augmente en permanence et les attentes des scientifiques sont de plus en plus élevées. Pour répondre à leurs besoins, les développeurs des outils de prédiction proposent ces solutions hybrides qui combinent la prise de mesures et la modélisation de l'architecture de calcul dont la résolution donne la prédiction de performances. Un grand nombre de développeurs de ce type d'outils ont opté pour l'approche hybride pour deux raisons :

1. celle-ci possède une capacité de modélisation des nœuds et du réseau facile à employer, par rapport aux outils des chapitres 1 et 2 ;
2. ces approches permettent une meilleure extrapolation des résultats, grâce à la modélisation des ressources. La résolution des modèles fait partie souvent d'un processus de simulation.

Les scientifiques ont observé le potentiel des méthodes hybrides. Pour ceci, un processus continu de développement se déroule, jusqu'à l'obtention d'un outil générique capable de couvrir une gamme variée des applications et systèmes. L'état de l'art des approches dans cette catégorie montre que ces performances, cette fois-ci pour les outils de prédictions, n'étaient pas encore atteintes malgré les nombreuses propositions (voir le tableau 3.1 pour celles pertinentes).

Avec une seule exception, c'est-à-dire RSIM, une moitié des outils modélisent les architectures homogènes, et l'autre modélise celles hétérogènes. Ce rapport indique la confiance que les développeurs font aux méthodes hybrides, grâce à leur capacité de prendre en compte les aspects liées à la mémoire et au multi-cœurs.

Du point de vue des langages de programmation utilisés pour le développement des applications, nous observons que les méthodes hybrides proposent une variété allant de C, C++, Fortran et jusqu'au Java. Dans le cas des outils qui supportent le C, le C++ et le Fortran, la communication se fait, presque sans exception, avec MPI ou une bibliothèque d'échange de message dérivée de celle-ci. Cet aspect limite beaucoup la plage d'applications à analyser avec ces outils.

La nouveauté parmi les outils hybrides est constitué par P2PPerf qui modélise les applications P2P en Java communiquant avec JXTA. Contrairement à celui-ci, les outils BigSim et POSE sont les plus limités car ils modélisent uniquement les applications développées en Charm++, une choix moins populaire que les autres langages qu'on vient de mentionner.

Grâce aux combinaisons entre les modèles analytiques, la prise de mesures et la simulation, les outils dans cette catégorie rendent les résultats avec une précision plus élevée que celle des outils présentés dans les chapitres 1 et 2. En ce qui concerne le coût pour accéder au résultat de prédiction, celui-ci est faible pour la majorité des outils hybrides.

Le *slowdown* , ou le ralentissement quand il ne s'agit pas d'un *slowdown* , est inférieur à l'unité dans la plupart des cas, sauf RSIM et, sous certains conditions, Dimemas, Bigsim, PSINS, WARPP et PHANTOM. En général, nous remarquons un faible coût pour obtenir la prédiction, ceci étant l'un des facteurs critiques dans le choix de l'outil de prédiction dans le domaine de l'*HPC* .

Malgré les nombreux outils hybrides, les scientifiques manquent un outil plus générique

qui pourra réunir les caractéristiques de la majeure partie des méthodes proposées dans ce chapitre. À cause de ceci, des nouvelles propositions sont attendues et des nouveaux méthodes sont en permanence présentées par les chercheurs.

TAB. 3.1: Les caractéristiques les plus importantes des outils hybrides de prédiction de performances, et les caractéristiques de l'outil dPerf. (1)Prendre en compte le niveau d'optimisation du compilateur ; (2) La communication ; (3)Voir chapitre 6 pour la méthodologie et l'implémentation de dPerf ; (4)La topologie logique de communication ; (5) Toute application qui communique dans un formalisme *Message Passing* ; (6) *Speedup*, et non pas *slowdown* ; (7)Le coût de la résolution du modèle ; (8) Le *speedup* du simulateur

Méthode ou Lab	Année	Système cible	Topologie(4)	Code d'entrée accepté			Optim.(1)	Valeurs		
				Type	Comm.(2)	Langage		Slowdown ou ralentissement	Précision [%]	Coût de prédiction(5)
dPerf(3)	2011	hétérogène	P2P décentralisée	distribué	P2P-SAP, MPI, autre	C,C++, Fortran	oui	0,2 - 1,1	> 80	˜1
Dimemas	1996 - 2004	hétérogène	centralisée	distribué	MP (5)	C,C++, Fortran	non	≥ 1	80	approx. 1
MPI-SIM	1998	homogène	centralisée	parallèle	MPI	Fortran	non	12(8)	80	-
ChronosMix	2000	hétérogène	centralisée	distribué	MPI	C	non	0,02-0,25	> 85	« 1
PACE	2000	hétérogène	centralisée	distribué	MPI/PVM	Tout	non	« 1	95	» 1 (haut)
POEMS	2000	homogène	centralisée	distribué	MP 5	Fortran	non	réduit	variable	» 1 (haut)
RSIM	2002	CC-NUMA,	centralisée SPARC ou Solaris	parallèle	ILP	Tout	non	>1000	-	»1
BigSim	2004	homogène	centralisée	parallèle	Charm++ , MPI		non	>1	94	inconnu
POSE	2005	homogène	centralisée	parallèle	Charm++ , MPI		non	«1	<90	»1 (haut)
P2PPerf	2006	hétérogène	P2P centralisée	distribué	JXTA	Java	non	<1 ou >1	14	variable
Susukita	2009	homogène	centralisée	distribué	MPI	inconnu	non	39(8)	80	faible
PSINS	2009	hétérogène	centralisée	distribué	MPI	Tout	non	7(6)	17	faible
Warpp	2009	homogène	centralisée	parallèle	MPI	Tout	non	>1	90	faible
PHANTOM	2010	hétérogène	centralisée	parallèle	MPI	Fortran	non	«1 ou >1	95	faible
MPI-PERF-SIM	2011	hétérogène	centralisée	parallèle	MPI	C	non	«1	> 83	faible
SMPI	2011	homogène	centralisée	distribué	MPI, OpenMPI	C	non	<1	71-90	faible

Chapitre 4

Les outils d'analyse et de visualisation des performances

Ce chapitre présente les outils les plus pertinents de visualisation des performances des applications ou systèmes *HPC*. Les outils présentés dans la suite ne donnent pas des prédictions de performances, mais seulement des informations sur le comportement des applications sur les systèmes *HPC*. Ces outils ne sont pas directement comparables avec dPerf, ni avec ceux présentés dans le chapitres 1, 2, et 3.

4.1 Paradyn et DynInst

Proposé dans [75], Paradyn est un outil qui instrumente et mesure les performances des applications et systèmes *HPC*. Le but de Paradyn est de guider les développeurs d'applications et les concepteurs des systèmes pour résoudre les aspects liés à la programmation ou à la conception du matériel. Les deux composants principaux de Paradyn sont le module d'instrumentation automatique et le module de prise des mesures. À part ceux-ci, Paradyn contient aussi une interface de visualisation, nommée VisiLab, qui peut servir aux développeurs d'outils de prédiction pour accéder aux résultats de performances de Paradyn, et pour le développement des outils de visualisation personnalisés.

Paradyn modélise les applications PVM et MPI destinées à l'exécution sur des systèmes homogènes ou hétérogènes. Cet outil identifie les points faibles dans la conception des logiciels et des systèmes. Les résultats de Paradyn sont évolutifs (*scalable*).

À partir de 1997, et sans mettre fin au développement de Paradyn, l'équipe de chercheurs à l'Université de Wisconsin (aux États-Unis) qui proposait en 1995 cet outil, présentent de nouvelles interfaces de programmation (API) plus spécifiques que Paradyn. L'une de ces interfaces est DynInst [87], une API chargée de l'instrumentation automatique du code source. Cet outil est activement développé.

Pour Paradyn et DynInst, le temps pour accéder au résultat de l'analyse n'est pas critique, car ils ne fournissent pas de prédictions de performance des applications, mais ils montrent le comportement de celles-ci.

4.2 *HPCToolkit*

C'est une collection d'outils pour réaliser des analyses de performances basées sur des mesures. Depuis la présentation publique de *HPCToolkit* en 2003 [106, 74], l'outil est actif et soutenu. *HPCToolkit* est composé de plusieurs bibliothèques écrites en Fortran, C, ou C++. Cet ensemble d'outils est conçu pour automatiser le processus d'instrumentation d'un code source, et pour minimiser le nombre de compilation des binaires rendus de l'instrumentation. Les résultats obtenus avec *HPCToolkit* sont évolutifs, modélisant des applications séquentielles ou parallèles. Plus récemment [105], *HPCToolkit* analyse et modélise les applications destinées aux systèmes homogènes d'un haut degré de parallélisme. Les principales actions de *HPCToolkit* sont :

- l'analyse et la prise des mesures des codes sources multi-langage qui utilisent des bibliothèques binaires externes ;
- prendre des mesures avec un minimum de (ou sans) instrumentation du code source sous évaluation ;
- l'enregistrement et la corrélation entre les mesures prises pour des multiples types d'événements ;
- le calcul de l'utilisation des ressources telles que la bande passante ou la mémoire, en vue de l'optimisation de l'application et du système analysés ;
- attribuer un coût très précis aux mesures, en fonction du contexte dynamique ou des boucles.

Les outils disponibles dans *HPCToolkit* ont une efficacité validée de façon expérimentale dans l'analyse des performances des grappes de calcul de très grande taille.

4.3 TAU

Cet outil a comme but d'évaluer les performances des applications parallèles qui communiquent en utilisant MPI. TAU (*Tuning and Analysis Utilities*)[97] est un effort commun du DOE (*Departement Of Energy*), du LLNL (*Lawrence Livermore National Laboratory*), de l'ANL (*Aragonne National Laboratory*), et du LANL (*Los Alamos National Laboratory*) aux États-Unis. Initialement, TAU ciblait les applications écrites en pC++ (la version parallélisée de C++), mais le développement a intégré aussi le support pour le C, le C++, le HPF (*High Performance Fortran*), le Java, et le Python.

TAU fournit des méthodes d'analyse statiques ainsi que dynamiques d'un code source. Il contient des mécanismes pour l'instrumentation automatique, la prise des mesures des performances et la visualisation des résultats pour un code source d'entrée. Les mesures prises avec TAU sont portables et évolutives grâce aux traces obtenues en sortie et écrites en format OTF (*Open Trace Format*)[84].

Les deux parties les plus importantes de TAU sont (i) le paquet de prise de mesures portables, et (ii) le paquet d'analyse de code source.

Le format OTF était proposé pour uniformiser les formats des fichiers de trace, et pourtant ceci, le format OTF ne peut être lu que par Vampir [85] (voir la section 4.4), un outil commercial payant. Cet aspect réduit la popularité du format OTF, car le développement des outils personnalisés de prédiction de performances qui intègrent un outil

commercial comme Vampir, est soumis aux conditions plus restreintes que dans le cas des outils libres.

4.4 Vampir

Vampir [85, 102] est un outil de visualisation pour les fichiers de trace écrits en format OTF (*Open Trace Format*)[84]. Les traces en format OTF sont générées par VampirTrace[86], une bibliothèque qui identifie les événements de calcul et de communication MPI ou OpenMP. Les événements contenus par un fichier OTF sont du type calcul ou communication. Seuls les événements de communication des bibliothèques MPI ou OpenMP apparaissent dans un fichier de trace OTF. Vampir met à la disposition de l'utilisateur de nombreuses façons de visualiser le comportement des applications, selon les traces. Pour les applications avec un grand nombre d'événements, une version parallèle de l'outil est disponible et se nomme VampirServer, .

Cet outil fournit un moyen très rapide pour visualiser en vue d'une analyse, les traces d'exécution d'une application parallèle. Vampir permet d'atteindre plusieurs niveaux de détail de la visualisation. Vampir implémente l'analyse optimisée des événements.

4.5 *HPM Toolkit*

Une équipe des chercheurs conduite par Luiz de Rose, développe *HPM Toolkit* (*Hardware Performance Monitor Toolkit*) [48], un ensemble d'outils pour mesurer les performances matérielles des applications exécutées sur des systèmes IBM. Le développement de *HPM Toolkit*a été fait entre les années 2001 et 2004, et il a quatre modules importants :

- *hpmcount* est chargé du démarrage d'une application qui fournit le temps de référence des mesures, les informations lues des compteurs matériels de performances, ainsi que des informations sur l'utilisation des ressources du système ;
- *libhpm* est la bibliothèque utilisée par les applications instrumentées pour accéder aux données mises à disposition à travers *hpmcount*. Cette bibliothèque est celle qui impose que les applications *HPC* soient écrites en C, en C++ ou en Fortran, et qu'elles communiquent en utilisant MPI ;
- *PeekPerf* est l'interface graphique pour visualiser les informations de performances générées par *libhpm* ;
- *hpmstat* a le rôle de ressembler les informations des compteurs matériels de performances par rapport à l'état du système.

L'applicabilité de *HPM Toolkit* reste limitée aux systèmes IBM, pourtant, les informations analysées permettent aux concepteurs des systèmes de bien analyser le comportement de différentes applications en fonction des modifications au niveau matériel.

4.6 Jumpshot

Cet outil est développé en Java et il est utilisé pour la visualisation post-mortem des performances des applications parallèles. *Jumpshot* [5], actuellement à la quatrième

version, utilise des fichiers journaux en format SLOG-2 pour avoir des informations évolutives. Ce type de journal permet à *Jumpshot* d'utiliser des niveaux de détails très variés pendant l'analyse des performances. Les principales fonctionnalités de *Jumpshot* sont disponibles à travers l'interface graphique. Celle-ci sont : (i) le convertisseur des journaux, (ii) la légende, (iii) le fil du temps, (iv) l'histogramme, et (v) la possibilité de personnaliser l'outil. *Jumpshot* est distribué *open source* en tant que composant du logiciel MPE (*MPI Parallel Environnement*). À part *Jumpshot*, MPE contient d'autres outils pour la prise de mesures des applications parallèles MPI, mais sans faire des prédictions de performances.

4.7 Paraver

Il est un outil de visualisation des performances développé à Barcelone (*Barcelona Supercomputing Center*) en Espagne. Paraver [17, 102]. Une des caractéristiques importantes de Paraver est l'interface graphique *Motif* aisée à utiliser. Le développement de Paraver cherche à mettre en priorité la qualité élevée de la visualisation du comportement des applications. Paraver est capable d'afficher une grande quantité d'informations sur un code exécuté et mesuré préalablement. Le but de Paraver est de fournir ces informations qui servent à prendre les meilleures décisions dans le développement et l'optimisation des applications. Paraver utilise l'outil OMPItrace [12] pour générer des fichiers de trace pour les applications communiquant en utilisant les bibliothèques OpenMP, MPI, ou les deux.

4.8 mpiP

mpiP [113] est une bibliothèque légère de modélisation des applications MPI développée par Vetter et al. au LLNL (*Lawrence Livermore National Laboratory*) aux États-Unis. Cet outil ne modélise pas le comportement entier d'une application mais uniquement les communications MPI. mpiP modélise les applications écrites en C, en C++, et en Fortran. L'utilité principale de mpiP est de faire connaître aux utilisateurs -de façon automatique et précise- les raisons qui empêchent leurs applications d'évoluer vers un nombre plus important de nœuds . Les informations sont capturées localement et les captures de tous les processus sont, à la fin, fusionnées. mpiP fournit le pourcentage de temps, par rapport à l'exécution totale, attribué a chacune de tâches, ainsi que des informations sur les moments et les processus qui initient les communications. L'analyse de performance avec mpiP est aisée grâce à l'interface graphique nommée *mpipview*.

4.9 KOJAK

Il est un outil d'analyse des performances des applications parallèles, communiquant en utilisant MPI, OpenMP, SHMEM, ou une combinaison de ceux-ci. KOJAK (*Kit for Objective Judgement and Knowledge-based Detection of Performance Bottlenecks*) [6] fournit des méthodes d'analyse qui varient de l'instrumentation, l'analyse post-mortem des données, et jusqu'à la présentation des résultats. KOJAK identifie, de façon automatique, les événements de communication parmi les traces d'exécution qui lui sont fournies. L'analyse

des événements de communication montrent si ceux-ci sont utilisés d'une façon efficace ou non, guidant ainsi les développeurs dans le processus d'optimisation du code source. Parfois, ce n'est pas l'optimisation du code source qui peut améliorer les performances, mais ce sont les ressources du système utilisé. Tous ces détails sont visibles à travers une interface graphique disponible dans KOJAK. À partir de l'année 2008, KOJAK reste un projet actif mais distribué publiquement avec l'ensemble d'outils nommé *Scalasca* (voir la section 4.10).

4.10 *Scalasca*

Cet outil est un effort commun de JSC (*Jülich Supercomuting Center*) et de *German Research School for Simulation Sciences* pour l'optimisation des performances des applications parallèles. *Scalasca*[58] permet la prise des mesures et l'analyse du comportement des applications au moment de l'exécution. Grâce aux détails fournis par *Scalasca*, les parties du code source qui manquent d'optimisation sont découvertes. Ces parties représentent, en général, des événements de communication et de synchronisation. *Scalasca* met à la disposition des développeurs d'applications et des concepteurs des systèmes *HPC*, des méthodes pour les assister dans le processus de prise de mesures, d'analyse des performances, d'amélioration du code source. Le but de cette assistance est d'obtenir une application optimisée.

Scalasca modélise les applications scientifiques qui communiquent en utilisant MPI, OpenMP, ou une combinaison de ceux-ci. L'analyse de performances faite par *Scalasca* nécessite l'instrumentation préalable du code source à analyser. Cette analyse peut être effectuée au niveau du code source, ou des bibliothèques. La façon dont le résultat sera rendu est choisie par l'utilisateur. Ainsi, ce résultat peut être visualisé en tant que rapport, à la fin de l'exécution du code source instrumenté, ou en utilisant des fichiers de trace. Dans les deux cas, un outil de visualisation est disponible dans *Scalasca*. Dans le cas des fichiers de trace, il y a un fichier pour chacun des processus parallèles. La visualisation est faite en parallèle, sur un nombre de nœuds égal à celui utilisé pour obtenir les traces. L'outil de visualisation permet de comparer les performances de différentes exécutions, ou d'observer l'impact de la suppression des événements de communication de la liste de traces.

Parmi les outils distribués avec *Scalasca*, nous mentionnons KOJAK (voir la section 4.9). Un outil alternatif pour la visualisation et l'analyse des traces générés par *Scalasca* est Vampir (voir la section 4.4).

4.11 *ScalaTrace*

ScalaTrace [81] est un outil proposé par Noeth et al. pour créer des fichiers de traces évolutifs. Cet outil ne modélise pas l'aspect séquentiel d'un code source. L'outil utilise une technique pour représenter d'une façon concise et évolutive les événements de communication de la bibliothèque MPI. *ScalaTrace* combine l'utilisation de la couche de prise de mesures du MPI (PMPI), avec le mécanisme déterministe proposé par Noeth et al. pour

réjouer les traces. L'approche est validée sur des applications C++ et Fortran.

ScalaTrace modélise les communications des systèmes homogènes et nécessite l'exécution de l'application analysée.

4.12 *Visual Studio Profiling Tools*

Cet ensemble d'outils est développé par *Microsoft* et son but est de mesurer, d'évaluer et d'identifier les problèmes liés aux performances des applications. Tous les outils dans ce paquet sont accessibles dans l'environnement de développement *Visual Studio*[1]. Dans [60], Goldin présente les fonctionnalités de ce paquet par rapport aux applications *HPC*.

L'analyse d'une application *HPC* avec Visual Studio implique l'utilisation de son propre IDE (*Integrated Development Environment*). La prise de mesures est démarrée automatiquement sur tous les nœuds du système, et l'application est exécutée. Les nœuds étant identiques, la prise de mesures est réalisée sur un seul nœud . Les résultats sont accessibles tout de suite depuis l'IDE de Visual Studio. *Visual Studio Profiling Tools* permet de capturer, afficher, et analyser les performances d'un code source selon les préférences de l'utilisateur. Ces préférences sont choisies avant la prise de mesure.

Un système basé sur *Windows HPC Server* est une grappe logique de calcul. L'optimisation de ce type d'architecture devient plus critique que dans le cas des grappes physiques, soit les systèmes localisés au même endroit. L'utilisation d'un système d'exploitation *Microsoft* avec *Visual Studio Profiling Tools* est exigée. Celle-ci réduit l'applicabilité de cet outil d'analyse des performances, car *Windows HPC Server* ne peut pas être déployé sur toute architecture de calcul disponible dans le domaine *HPC*.

4.13 D'autres outils d'analyse ou de transformation de code source

A part les outils de prédiction présentés dans les chapitres 1 - 3, ou ceux de visualisation des performances (voir les sections 4.1 - 4.12), il y d'autres outils qui sont principalement des compilateurs ou des outils de transformations source-à-source. Le tableau 4.1 résume ces outils avec les langages acceptés en entrée.

[1] Valable pour les versions *Visual Studio Ultimate* et *Visual Studio Premium*

TAB. 4.1: Liste des outils pertinents d'analyse, de transformation ou de compilation de codes source. (1) Voir la section 5.4 ;

Nom de l'outil	Caractéristiques principales	Développeur	Langages gérés	Projet actif
PIPS[19]	compilateur (parmi d'autres fonctionnalités)	partenariat entre la France, l'Angleterre et les États-Unis	C, Fortran	oui
Polaris[21]	compiler	Univ. A&M Texas (États-Unis)	Fortran	non
SUIF[26]	compiler	Univ. Standford (États-Unis)	C, Fortran	non
Open64[13]	compilateur	CAPSL (États-Unis)	C, C++, Fortran	oui
CETUS[2]	compilateur, transformations source-à-source	Univ. Purdue (États-Unis)	C, C++	oui
OSCAR[14]	compilateur	Univ. Waseda (Japon)	Fortran	oui
PoCC[20]	compilateur source-à-source ou basé sur un modèle	INRIA (France)	C	oui
Rose(1)[96]	compilateur, transformations source-à-source	LLNL (États-Unis)	C, C++, Fortran, OpenMP, UPC	oui
LLVM[7]	compiler	Univ. Illinois (États-Unis)	C, C++, objective-C, Fortran	oui

Conclusion

Cette partie présente l'état de l'art des outils de prédiction de performances ainsi que des outils de visualisation des performances des applications ou des systèmes HPC.

La catégorie des outils présentés dans le chapitre 1 est caractérisé par un coût élevé pour obtenir la prédiction de performances. Ce coût est en grande partie causé par la conception du modèle de l'application ou du système cible. Une fois le modèle réalisé, ces outils fournissent le résultat de prédiction dans un délai très court. Leur principal inconvénient est de devoir investir dans la ré-conception du modèle utilisé pour toute modification du système cible ou de l'application analysée. Contrairement à ces outils, nous présentons dans ce manuscrit une méthodologie dont le temps pour obtenir une prédiction est inférieur ou égal au temps d'une exécution normale, ce qui est plus rapide que les outils analytique de prédiction. Notre méthodologie utilise (i) deux techniques de *benchmarking* par bloc d'instructions, ainsi que (ii) la simulation basée sur des fichiers de trace, celle-ci permettant d'économiser du temps et de donner le résultat rapidement.

Les outils présentés dans le chapitre 2 sont caractérisés par une modélisation basée sur une exécution complète, une exécution partielle ou un historique d'exécutions précédentes. Les outils de cette catégorie couvrent une gamme très vaste de cas, mais prises individuellement, ceux-ci restent limités dans l'application acceptée ou dans le système modélisé. Si certains outils acceptent des applications écrites en C et C++, d'autres outils modélisent C, C++ et Fortran, mais sur des architectures homogènes uniquement. Si certaines des méthodes de cette catégorie prédisent les performances des applications indépendamment de leur langaueg de programmation, la communication doit se faire en utilisant MPI. La méthodologie que nous décrivons dans cette thèse prédit les performances des applications écrites en C, C++ ou Fortran, qui communiquent en utilisant MPI, P2P-SAP, ou une autre bibliothèque de communication qui pourra être configuré, et les systèmes modélisés sont homogènes ou hétérogènes. Cette solution que nous proposons par la suite calcule des prédictions précises dans un **temps réduit** grâce aux techniques de *benchmarking* par bloc d'instructions et de la simulation basée sur des fichiers de trace.

Dans la catégorie des outils hybrides (voir le chapitre 3), les caractéristiques les plus importantes des outils présentés sont l'utilisation des modèles mathématiques et l'utilisation des simulateurs pour le calcul de la prédiction de performances. La précision des résultats fournis par les outils de ce groupe dépasse pour la plupart la valeur de 80%. Ces outils modélisent, a part quelques exceptions, les systèmes hétérogènes mais ils restent limités dans le choix du langage de programmation. La majeure partie des outils n'acceptent qu'un seul langage de programmation : soit le C, soit le Fortran, soit le Java, soit (dans le cas de BigSim et POSE) Charm++. Il y a des exceptions qui modélisent

les applications quel que soit le langage, mais une contrainte reste au niveau de la bibliothèque de communication. Tous les outils ne modélisent que les communications MPI, a part P2PPerf qui est conçu pour les applications communiquant en JXTA et RSIM pour celles en ILP (*Instruction Level Parallelism)*. Par rapport aux outils dans cette catégorie, notre méthode modélise des applications C, C++ ou Fortran, qui communiquent en utilisant MPI ou P2P-SAP, tout en prenant en compte les différentes niveaux d'optimisation du compilateur. La prise en compte de l'optimisation du compilateur place les résultats de prédiction issus de cette thèse dans un **contexte réel**. De plus, notre méthodologie propose des technique de modélisation des applications qui sont exécutées dans un environnement pair-à-pair décentralisé.

L'outil issu de la méthodologie présentée dans cette thèse fournit de prédictions de performances et ne propose pas de méthode de visualisation des résultats. Les traces qui sont utilisées par notre outil afin de calculer les prédictions peuvent être visualisé à l'aide des outils de visualisation de performances standard. La contribution de cette thèse ne propose donc pas de nouveautés par rapport aux outils de visualisation décrits dans le chapitre 4.

Deuxième partie

Contribution - Prédiction de performances

Introduction

Les systèmes *HPC* deviennent de plus en plus performants mais cette performance requiert une architecture de plus en plus complexe. Les fabricants de composants dédiés à ces systèmes proposent continuellement des technologies nouvelles afin de satisfaire une demande de la part des scientifiques qui ont besoin d'une importante puissance de calcul. Le domaine de *HPC* - calcul haute performance - connait donc un progrès permanent liant les acteurs énumérés ci-dessous :

1. les chercheurs ;
2. les concepteurs et fabricants des architectures de calcul (en général de haute performance, mais pas seulement) ;
3. les développeurs d'applications destinées à l'*HPC* .

Les chercheurs demandent la mise à disposition de ressources de calcul, idéalement infinies, pour avancer leurs activités de recherche. Ce besoin a, forcément, un coût qui est assez élevé, comparé aux coûts de ressources "'généralistes" de calcul (des ordinateurs de bureau ou personnels). La nécessité de résoudre des problèmes scientifiques de plus en plus complexes pour un coût du système *HPC* réduit au maximum, a permis la création des outils de prédiction de performances. Ces outils ont, en général, un but commun : fournir les informations sur la performance d'une application destinée à l'exécution sur un système *HPC* nouveau, ou mis à niveau.

La contribution de cette thèse vient à aider les chercheurs en leur fournissant dPerf, un nouvel outil de prédiction de performances basé sur des méthodes innovantes et utilisant les dernières techniques d'analyse du code source tout en proposant des solutions aux multiples questions liées au calcul de haute performance. dPerf, l'implémentation de ces travaux de thèse aidera les scientifiques à faire un meilleur choix pour augmenter les performances de leurs applications suite à une analyse menée sur une architecture de calcul hôte, et une prédiction sur une architecture cible.

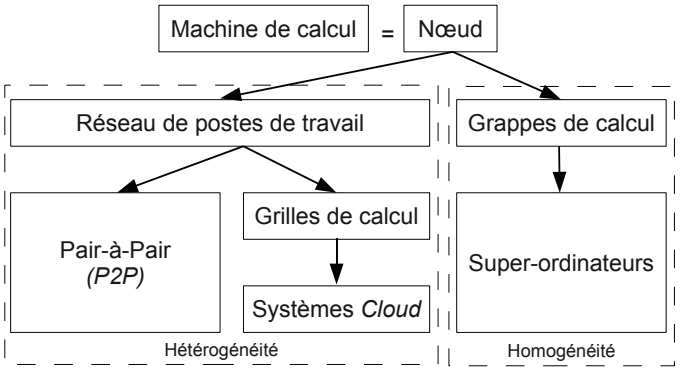


FIG. 4.0.1: L'évolution des architectures de calcul (concernant le domaine de l'*HPC*) à partir des nœuds indépendants, vers les grappes de calcul, les grille de calcul, le *cloud* ou les super-ordinateurs.

Le rôle des concepteurs et des fabricants des architectures de calcul est de proposer des solutions qui répondent aux demandes des scientifiques. Pourtant, les efforts dans la conception et la fabrication viennent, très souvent, avec un coût important. Afin que leurs efforts soient récompensés, il est impératif que les solutions qu'ils proposent restent accessibles pour le client. À cet effet, les concepteurs ont fait évoluer les architectures de calcul (voir Fig. 4.0.1) à partir des machines indépendantes. Peu après, les machines de calcul étaient nommées des "nœuds" de calcul, car très vite les concepteurs d'architectures de calcul ont observé qu'ils peuvent grouper plusieurs machines (ou nœuds) pour fournir une puissance nettement supérieure à celle d'une machine indépendante. Comme on peut voir dans la figure 4.0.1, l'évolution a créé, d'un côté, les architectures de calcul basées sur une homogénéité des nœuds et de l'infrastructure réseau, et, de l'autre côté, les systèmes hétérogènes, contenant plusieurs types de nœuds et de composants réseau. Inspiré par les grilles de calcul et le *cloud*, l'intérêt pour les réseaux P2P n'a cessé d'augmenter, grâce à leur potentiel de calcul pour un coût très faible.

dPerf est conçu pour prédire des performances non seulement pour les codes *HPC* parallèles ou distribués, mais surtout pour les applications destinées à des environnements de calcul haute performance pair-à-pair (centralisé ou décentralisé). Plus particulièrement, dPerf permet de donner des résultats de performances pour un nouvel environnement qui met en valeur le potentiel *HPC* de l'infrastructure P2P. Les prédictions rendues par dPerf aideront les concepteurs et les fabricants des architectures de calcul à mieux configurer leurs produits, quel que soit le type du système (hétérogène ou homogène).

L'outil que nous proposons s'adresse également aux développeurs d'applications de calcul haute performance. Les langages utilisés le plus souvent pour la programmation d'applications *HPC* sont le C, le C++ et le Fortran. Ces développeurs ont toujours pu bénéficier de l'aide fournie par les outils de prédiction de performances, mais ils se trouvent en difficulté face aux systèmes nouveaux dont la complexité ne peut plus être gérée par les outils développés jusqu'à aujourd'hui. La complexité des applications *HPC* de nos jours implique au moins la reconnaissance des systèmes hétérogènes, avec la topologie logique centralisée ou décentralisée (comme c'est le cas du réseau P2P), où les applications de calcul peuvent être parallèles ou distribuées, et la communication pouvant se faire dans MPI ou dans d'autres formalismes de communication mieux adaptés au système ciblé (comme c'est le cas de P2P-SAP pour le système P2P). En plus des aspects qu'on vient de mentionner, le développeur d'une application destinée à l'*HPC* pourra bénéficier encore plus des outils de prédiction de performance s'il peut obtenir des résultats de prédiction tout au long du cycle de développement de son application.

Dans le paragraphe ci-dessus, nous avons mentionné les nombreux aspects nécessaires pour qu'un outil de prédiction de performance soit général et qu'il soit adapté aux systèmes et aux applications *HPC* d'aujourd'hui. Pour couvrir tous ces aspects, nous proposons un outil nouveau pour une prédiction de performances qui amène des réponses aux exigences des scientifiques, des concepteurs et des développeurs du domaine de l'*HPC* . Notre outil est appelé dPerf.

Cette partie du document présente toutes les conditions nécessaires pour le bon fonctionnement de dPerf, ainsi que tous les détails sur les techniques utilisées (la méthodologie) et leur implémentation dans dPerf à partir d'un code *HPC* reçu en entrée, et jusqu'au moment de l'obtention de sa prédiction de performances.

Chapitre 5

Pré-requis

La thèse défendue est la prédiction de performances des applications distribuées réelles exécutés dans des conditions réelles. Ces travaux présentent une approche de prédiction de performances qui prend en compte les différentes niveaux d'optimisations utilisés par le compilateur, tout en réduisant le ralentissement du processus de prédiction et en fournissant des résultats extrapolables. Ce chapitre présente toutes les conditions nécessaires au développement et au bon fonctionnement de notre outil dPerf, une implémentation de la contribution de ces travaux. Ce chapitre présente les méthodes de dernière génération utilisées dans dPerf afin que celui-ci rende possible (1) la réduction du *slowdown* et (2) la prédiction de performances d'un code C, C++ ou Fortran dans de conditions réelles. Ce code communique dans P2P-SAP ou MPI, et il est destiné aux systèmes de calcul *HPC* pair-à-pair décentralisés.

Le langage que nous avons choisi pour le développement de dPerf est le C++. Ce choix été fait pour intégrer plusieurs outils existants -qui n'étaient pas développés essentiellement pour le domaine de la prédiction de performance- et parce-que le C++ est parmi les langages les plus utilisés dans l'*HPC* . De plus, notre choix pour le langage C++ est confirmé également par les conditions à remplir, présentées dans la suite.

5.1 Les compteurs matériels de performance (*Hardware Performance Counters*)

La prédiction de performance basée sur des mesures, comme c'est le cas de notre outil, est dépendante de la technique de prise de mesure. Nous avons cherché la meilleure façon de mesurer rapidement et avec précision les cycles de calcul d'un système, sachant que notre prise de mesure doit réduire au maximum le bruit introduit dans le système mesuré. À cet effet, nous avons décidé de prendre les mesures à l'aide des compteurs hardware. Ces compteurs sont des registres d'une utilité bien spécifique, présents dans la plupart des microprocesseurs d'aujourd'hui. Les compteurs hardware peuvent mesurer toute activité du microprocesseur. L'information fournie par les compteurs hardware concerne deux types d'événements :

1. les créneaux de temps des cycles de calcul initiés par, et appartenant à, l'utilisateur prenant les mesures. Ceux-ci sont sont appelés des "cycles utilisateur". Une mesure

de la durée des cycles utilisateur donne un "temps utilisateur" ;

2. les créneaux de temps des cycles calcul initiés par, et appartenant à, le système d'exploitation.On appelle ceux-ci des "cycles système". Une mesure de la durée des cycles système donne un "temps système".

Bien évidement, une durée qui correspond à un type d'événement parmi ceux mentionnés ci-dessus, est exprimée toujours relativement à la mesure prise au moment zéro, ou le moment de début.

Une troisième catégorie existe, qui n'est pas liée aux compteurs de performance, mais à l'horloge interne de la machine (ou du nœud). Il s'agit du temps réel, connu en tant que *wall time* ou *wall clock time* et qui représente l'unité de temps du monde réel.

La précision de la mesure prise en utilisant les compteurs hardware dépendra directement de l'implémentation de la méthode faisant la lecture des compteurs. Ces aspects sont présentés dans la section 6.2.1.

5.2 GNU/Linux et l'accès aux compteurs matériels

Les compteurs hardware sont facilement accessibles depuis GNU/Linux. Dans ce système d'exploitation, quelle que soit la distribution choisie, on peut activer de deux façons le module *Performance Counters*, qui est chargé de l'accès aux registres compteurs de performance. La première façon d'activer ce module est de compiler soi-même une version personnalisée du noyau Linux (ou *Linux kernel*), si ce noyau est une version plus ancienne que celle 2.6.31. À partir de cette version, le module *Performance Counters* est activé par default au moment de l'installation du Linux. Une étude effectuée à l'Université de Lugano par Zaparanuks et al. présentent dans [125] les infrastructures qu'on peut activer dans le module *Performance Counters*, pour la prise de mesures. Ces infrastructures sont : *perfctr* [89] et *perfmon* [18].

La partie expérimentale décrite par Zaparanuks et al. montre que l'utilisation de *perfmon* et de *perfctr* pour lire les compteurs matériels de performances sont plus efficaces que les mesures qui n'utilisent pas ces compteurs, mais les différences entre les deux bibliothèques apparaissent en fonction du type de la mesure prise. L'ordre hiérarchique est visible dans la Fig. 5.1. Les deux situations possibles sont :

1. l'utilisation directe des fonctions des bibliothèques *perfmon* ou *perfctr* pour la prise de mesures. L'analyse automatique qui fait partie du processus de prédiction de performances de dPerf nécessite une API (Application Programming Interface) facile à intégrer et utiliser. Pour cette raison, l'utilisation directe des bibliothèques *perfmon* ou *perfctr* dans dPerf n'est pas efficace.
2. l'utilisation des fonctions PAPI (voir section 5.3) pour accéder les fonctions des bibliothèques ci-dessus. Dans la section suivante, nous expliquons les principales différences entre les fonctions PAPI de haut et de bas niveau. dPerf utilise les fonctions PAPI de bas niveau, et donc la meilleure precision des mesures, selon Zaparanuks et al., est obtenue en utilisant PAPI sur *perfctr*.

Selon les expériences réalisées par Zaparanuks et al. nous avons choisi d'activer le module des compteurs matériels avec *perfctr*.

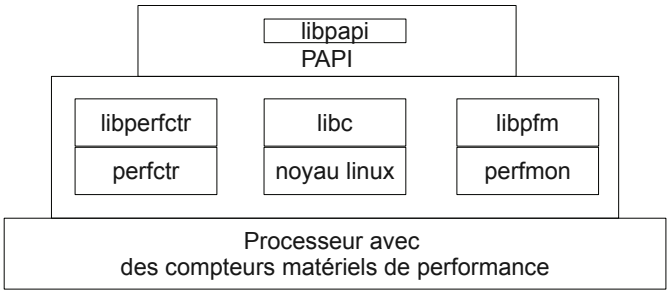


FIG. 5.1: La hiérarchie nécessaire pour accéder les compteurs matériels de performances. Cet accès est possible directement à travers les bibliothèques *perfmon* et *perfctr*, ou en utilisant l'API de PAPI pour accéder aux mêmes bibliothèques .

5.3 PAPI - accéder aux compteurs matériels

La plupart des méthodes qui utilisent une approche basée sur des mesures, utilisent des fonctions telles que *"gettimeofday"* pour lire le numéro du cycle courant du processeur. Pour connaître le nombre écoulé de cycles de calcul, on prend une deuxième mesure et on fait la différence. Les fonctions utilisées pour mesurer la durée sont basées sur la lecture de l'état d'un processeur. La façon dont une fonction est implémentée, sera caractérisée (au moins) par :

- le coût, en terme de cycles du processeur, nécessaire pour (1) appeler la fonction prenant la mesure, (2) lire l'état du processeur, et (3) rendre le résultat ;
- la résolution de la méthode utilisée par la fonction pour lire l'état du processeur, celle-ci pouvant être de l'ordre des centaines de millisecondes -soit une faible résolution- et jusqu'aux nanosecondes -soit une résolution élevée.

La bibliothèque MPI contient une fonction pour prendre des mesures de temps, appelée *MPI_ Wtime*. Cette fonction lit le temps réel instantané avec une résolution de l'ordre de la millisecondes. Bien sûr, cette fonction ne peut être utilisée qu'avec une application communiquant avec MPI, et notre outil doit modéliser les applications communiquant avec P2P-SAP (pour les architectures P2P ou la communication se fait d'une manière de-centralisée).

La syntaxe des langages C et C++ contient la fonction *gettimeofday*. Elle est très souvent utilisée par les développeurs pour mesurer la durée. Nous avons décidé de ne pas l'utiliser à cause du coût pour son appel, et à cause de la résolution limitée aux millisecondes. Sa résolution est de l'ordre des millisecondes, mais la prise de mesures dépend du créneau de temps qui lui était attribué par le microprocesseur. Conforme à l'étude menée par Finney dans [55], certains systèmes basés sur UNIX mettent à jour la valeur de *gettimeofday* toutes les 10 millisecondes. Cet aspect rend les performances de *gettimeofday* inadéquates pour notre outil qui nécessite la meilleur résolution pour mesurer même les plus courts calculs du processeur.

Nous développons un outil qui analyse des applications écrites en C, C++ ou Fortran. Nous avons cherché une façon rapide et précise pour la prise de mesures que nous pouvons utiliser avec ces trois langages de programmation. *Performance Application Programing Interface* (ou PAPI)[15],[16] est la bibliothèque que nous avons choisi d'utiliser car elle

le sur-coût ajouté aux mesures prises est négligeable, sa précision est très haute, et les durées sont exprimées en nanosecondes.

PAPI est une interface pour la lecture des compteurs matériels développée à l'Université de Tennessee (États-Unis). Au lancement du projet, son but était de concevoir, de standardiser, et d'implémenter PAPI, une interface pour accéder aux registres du processeur afin de lire son état. PAPI a beaucoup évolué entre la première présentation faite par Mucci et al. [78], et aujourd'hui, car ce projet est toujours actif.

Cette bibliothèque fonctionne sur une grande plage d'architectures de calcul car elle est standardisée, et elle comprend un grand nombre de fonctions -autour de 100- pour lire les compteurs de performance du processeur. L'appel des fonctions peut se faire en C ou en Fortran. En utilisant PAPI, les développeurs ont accès aux compteurs matériels avec un minimum de bruit introduit dans le système mesuré. Ceci augmente la précision des mesures prises dans le cadre du processus de prédiction de performances. Le nom de toutes les fonctions disponibles dans PAPI commence par "PAPI_". Elles sont divisées dans deux catégories :

1. des fonctions de haut niveau (*High Level Functions*) ;
2. des fonctions de bas niveau (*Low Level Functions*).

5.3.1 Les fonctions PAPI de haut niveau

Elles représentent un groupe d'interfaces très basiques qui peuvent aider l'utilisateur à instrumenter une application. Parmi ces fonctions on trouve :

- "PAPI_num_counters". Cette fonction lit le nombre total de compteurs de performance disponibles dans l'unité de traitement.
- "PAPI_flops" est une fonction qui permet, d'une façon simplifiée, la lecture des megaflops par seconde (Mflops/s) depuis la dernière lecture avec PAPI_flops, où "flops" représente le nombre d'opérations en virgule mobile effectuées par le processeur dans une seconde.

5.3.2 Les fonctions PAPI de bas niveau

Elles sont moins restrictives et elles permettent d'accéder aux compteurs en mode avancé. Nous allons mentionner ci-dessous tous les fonctions disponibles dans PAPI, que nous utilisons :

- "PAPI_library_init". Cette fonction, nous initialise la bibliothèque PAPI.
- "PAPI_get_real_nsec" : lit le nombre total de nanosecondes ($seconde * 10^{-9}$) depuis un temps de réference arbitrairement choisi. En général, au début d'une application à mesurer, nous faisons appel à la méthode *PAPI_get_real_nsec*. La valeur rendue constituera le temps de référence (t0). Après, tous les nouveaux appels de cette fonction vont rendre des temps instantané que nous allons utiliser pour faire la différence entre le temps de référence et un temps ultérieur. Le temps lu par cette fonction est le temps réel (*wall time*). Il correspond à la durée totale depuis le premier créneau de temps attribué par le processeur à l'application mesurée, jusqu'au dernier créneau. La mesure prend en compte tous les créneaux de temps de tous les processus

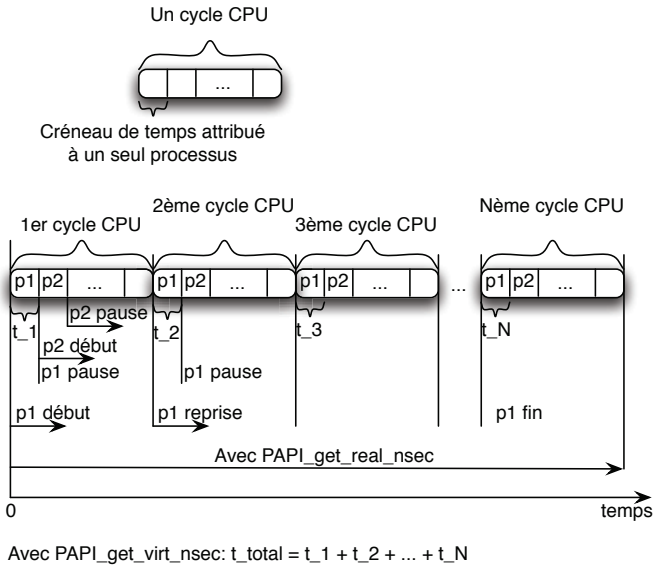


FIG. 5.2: Vue comparative entre la prise de mesure avec PAPI en mode "réel" et en mode "virtuel"

de cet intervalle, quel que soit le nombre de processus exécutés, comme on peut le voir dans la Fig.5.2 . Nous avons choisi d'utiliser la résolution de nanosecondes dans dPerf.

– "PAPI_get_virt_nsec" lit, depuis les compteurs matériels, le temps virtuel. Ce temps correspond à la durée totale des créneaux de temps affectés par le processeur à l'application mesurée. Autrement dit, il s'agit de la somme des cycles utilisateur. Pour les expériences réalisées durant cette thèse, nous avons choisi d'utiliser seulement les fonctions qui fournissent les temps "utilisateur", afin d'éviter tout bruit possible généré par les applications système tournant en arrière plan sur les systèmes mesures.

L'appel de *PAPI_library_init* est obligatoire avant de pouvoir utiliser toute autre fonction PAPI.

Prédire les performances est un processus délicat car la précision doit être d'un très haut niveau. Cela implique la prise en compte de tous les facteurs externes pouvant influencer la prédiction. Pour cette raison, nous avons utilisé la bibliothèque PAPI, car c'est, de notre point de vue, la meilleure interface pour lire les compteurs matériels :

- PAPI fournit, à travers ses nombreuses fonctions, des mesures de temps exprimées en micro- ou nanosecondes ;
- le bruit introduit par l'appel d'une fonction PAPI dans le système mesuré est très réduit [125]. L'erreur de prise de mesure est ainsi réduite par 66% par rapport à d'autre méthodes.

La façon dont nous avons utilisé les fonctions de PAPI dans notre outil dPerf est présentée dans le chapitre 6.

5.4 ROSE

Nous proposons un outil capable de fournir des prédictions de performances pour une gamme variée d'applications *HPC* . Pour cela, nous avons développé dPerf en dessus du framework *ROSE Compiler* (voir Fig. 6.1), un outil d'analyse statique d'application développé à Lawrence Livermore National Laboratory (États-Unis)[96],[23].

5.4.1 Une vue d'ensemble

ROSE, développé par Quinlan et al., est un environnement de compilation disponible *open-source*. Cet environnement fournit une gamme très large de fonctions pour la création des outils de transformation source-à-source, et d'analyse pour les applications à grande échelle (applications *HPC*). ROSE est conçu pour servir comme base pour le développement des outils personnalisés, destinés à :

- la réduction du *slowdown* ;
- la transformation automatisée d'un code source reçu en entrée ;
- l'optimisation d'un code source suite à une analyse statique ;
- l'optimisation des boucles complexes.

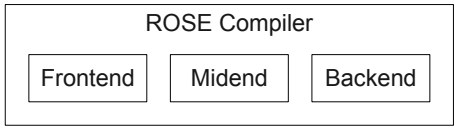


Fig. 5.3: Les trois parties de ROSE.

ROSE est structuré en trois parties (voir Fig. 5.3), le *frontend* étant chargé de la décomposition syntaxique d'un code d'entrée, le *midend* qui fournit toutes les fonctions nécessaires au parcours, à son analyse et à la transformation d'un code reçu en entrée, et enfin le *backend* qui est chargé de rendre le code d'entrée avec la prise en compte de toutes les modifications faites dans le *midend* . Une vue détaillée de la structure de ROSE est présentée dans la figure 5.4.

En choisissant ROSE comme base de départ pour le développement de dPerf, nous avons la possibilité de choisir parmi les langages de programmation reconnus par ROSE, ceux qui pourront être analysés par notre outil. En ce moment, ROSE reconnait les langages suivants :

- le C
- le C++
- le Fortran, versions Fortran4, F66, F77, Fortran 90/95, et Fortran 2003
- l'OpenMP
- l'UPC

Dans cette liste, nous nous sommes orientés vers le C, le C++ et le Fortran (voir Fig. 6.1), car ils sont les plus utilisés pour développer des applications *HPC* .

Ce choix représente un premier avantage de dPerf par rapport aux outils de prédiction de performance.

Pour le développement de dPerf, certaines caractéristiques de ROSE ont attiré notre attention :

- il est développé à partir du framework SAGE++ [36], un ensemble d'outils (un *toolkit*) pour le préprocesseur de compilateur. Le guide d'utilisateur de ROSE explique (voir [22]) l'évolution de ROSE à partir de SAGE++, en passant par SAGE II et SAGE III. Le framework Sage++ a été utilisé pour le développement d'autres outils de prédiction de performances tel que ChronosMix [38] qui a démontré la puissance de la décomposition syntactique avec SAGE++ appliquée sur des codes parallèles. Depuis, grâce au développement continu, ROSE est devenu un outil idéal pour l'analyse de code source.
- son *frontend* crée la décomposition syntactique d'un code d'entrée, décomposition appelée "représentation intermédiaire" (*IR*). Il y a plusieurs *IR* qui varient selon la complexité de l'analyse poursuivie. Ces *IR* sont présentées dans la suite.
- les décompositions crées par le *frontend* de ROSE contiennent toutes les informations existantes dans le code original reçu en entrée. Cet aspect garantit la concordance entre le code original et le code transformé (l'output de l'analyse).
- il contient un nombre très important de méthodes pour analyser et modifier les *IR* . ROSE contient aussi les mécanismes pour vérifier les éventuelles violations de la syntaxe du langage utilisé.
- À la fin de l'analyse et de la transformation, ROSE rend en sortie, à travers son module *backend* , un code nouveau écrit dans le même langage que le code d'entrée.

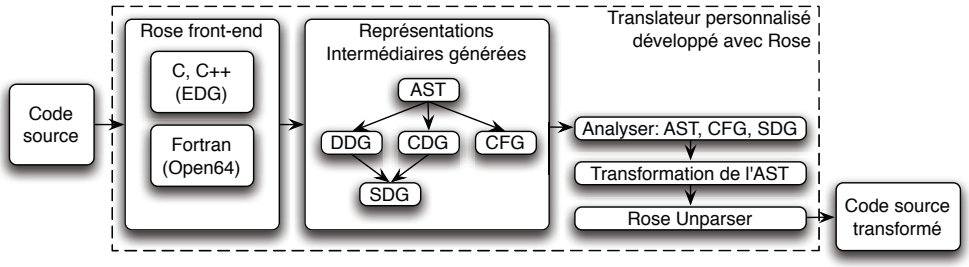


FIG. 5.4: La structure générale d'un outil personnalisé basé sur ROSE

La Fig. 5.4 montre qu'avec ROSE on peut créer, à travers son module *frontend* , un nombre important d'*IR* qui peuvent être utilisées dans la prédiction de performances. L'accès aux *IR* est fait grâce au module *midend* de ROSE. Le graphe syntactique d'un code est présenté en tant qu'*Abstract Syntax Tree*. Analyser les graphes des dépendances de contrôle et des dépendances de donnée est possible grâce au *Control Dependence Graph* et au *Data Dependence Graph*. Une analyse commune des deux graphes qu'on vient de mentionner peut être faite sur le *System Dependence Graph*.

En fonction de la complexité du code source, dPerf utilisera les différentes *IR* de ROSE pour faire l'analyse de code. Cet aspect représente un deuxième avantage de dPerf par rapport aux outils de prédiction de performances existantes. Une des raisons d'avoir choisir ROSE comme base pour le développement de dPerf est le fait d'être un projet

activement développé et soutenu. Ce fait aide à utiliser ROSE dans le développement d'un outil nouveau qui dépasse l'applicabilité prévue par Quinlan et al. et l'emmène dans le domaine de l'*HPC* . Son développement actif implique des réponses de la part des développeurs de ROSE à nos questions liées à notre façon d'utiliser leur outil.

5.4.2 Des représentations intermédiaires (IR)

Dans la suite, nous présentons les *IR* de ROSE et nous expliquons leur utilité dans la réduction du *slowdown* , l'étape initiale du processus de prédiction de performances.

Listing 5.1: Le code d'entrée (en C) d'exemple utilisé pour obtenir l'AST, le DDG, le CDG et le SDG expliqués dans cette section.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int addition(){
5          int x,y;
6          x=rand()%10+1;
7          y=rand()%10+1;
8          return (x+y);
9  }
10
11 int main(){
12         int a,b;
13         a = addition();
14         b = addition();
15         if (a>=b){
16                 printf("max=%d \n",a);
17         }
18         else{
19                 printf("max=%d \n",b);
20                 }
21         return 0;
22 }
```

5.4.2.1 L'*Abstract Syntax Tree* (AST)

C'est la représentation syntactique fondamentale d'un fichier source. Cette représentation a une structure arborescente abstraite, d'où le nom d'arbre syntactique abstrait. Analyser l'AST est aisé grâce aux méthodes prévues à cet effet par les développeurs de ROSE. Si on prend comme exemple un code simple (voir Fig. 5.1), on peut voir son AST dans la Fig. 5.5. Dans l'AST, on retrouve chaque instruction du code et chacun de ses éléments représentés par un nœud . L'analyse de cet arbre syntactique est, en fait, un ensemble de recherches et de modifications effectuées sur l'AST. Les différentes fonctions

pour modifier l'AST varient à partir d'une simple recherche ou identification des mots clés, pouvant aller jusqu'à l'insertion de nouvelles instructions de programmation dans l'AST.

Une fois l'analyse et la transformation finies, on peut appeler, dans le *backend* de ROSE, la fonction chargée de transformer l'AST en lignes d'instructions du même langage que le code original utilisé au début pour générer l'AST. Cette action est appelée *unparsing*. Le code nouveau contiendra toutes les transformations souffertes par l'AST.

Jusqu'à maintenant, nous avons parlé de l'importance de l'AST quand il y a un seul fichier source reçu en entrée. Si nous voulons analyser d'une façon statique un projet contenant plusieurs fichiers, la procédure est celle proposée par Kulkarni et al. dans [68], puis définie par Quinlan et al. dans [92]. Ils présentent une méthode, disponible aussi dans le framework ROSE, pour obtenir l'AST de plusieurs fichiers appartenant au même projet, représentation appelée *merged AST*, ou l'AST fusionné.

Comme décrit dans le chapitre 6, dans dPerf nous utilisons l'AST d'un code d'entrée afin d'identifier et instrumenter des éléments clés comme les déclarations (IF, FOR, DO WHILE, SWITCH), les blocs d'instructions, et les appels de communication (MPI ou P2P-SAP). Pour dPerf, si le niveau de dépendances est élevé dans le code analysé, l'AST ne suffit pas et la réduction du *slowdown* a besoin d'un complément pour les dépendances de données. Ce complement peut être réalisé par exécution ou par la l'analyse d'au moins une *IR* de plus que l'AST : le DDG, le CDG ou le SDG.

5.4.2.2 Le *Data Dependence Graph* (DDG)

Ce graphe fournit des informations sur les dépendances de donnés. Si le code d'entrée contient plusieurs fonctions, une représentation DDG existera à la sortie pour chacune d'elles. Prenons le code de la Fig. 5.1. Après avoir appelée la méthode de ROSE chargée de la création de DDG, deux représentations seront obtenues. La Fig. 5.6 contient le DDG de la fonction "main" en haut, el celui de la fonction "addition" en bas.

Les valeurs rendues par des fonctions sont des nœuds représentés par des rectangles. On retrouve chaque ligne d'instructions, sans modification et sans être décomposée, dans une ellipse. Les arcs étiquetés "DATA" relient chaque deux nœuds dépendants l'un sur l'autre. Le nœud se situant au sommet de l'arc, est dépendant de la valeur calculée dans le nœud se trouvant à la base de l'arc. Cette *IR* est utile à la résolution des dépendances de données dans le but de faire une analyse statique des codes plus complexes.

5.4.2.3 Le *Control Dependence Graph* (CDG)

Similaire au DDG, le CDG est un graphe montrant les dépendances de contrôle. La création du CDG d'un code d'entrée implique la génération d'un CDG pour chaque fonction définie dans le code, et donc sans détailler les fonctions déclarées et définies externes. La Fig. 5.7 montre le CDG de chacune des fonctions ("main" et "addition") définies dans le code d'exemple présenté dans la Fig. 5.1.

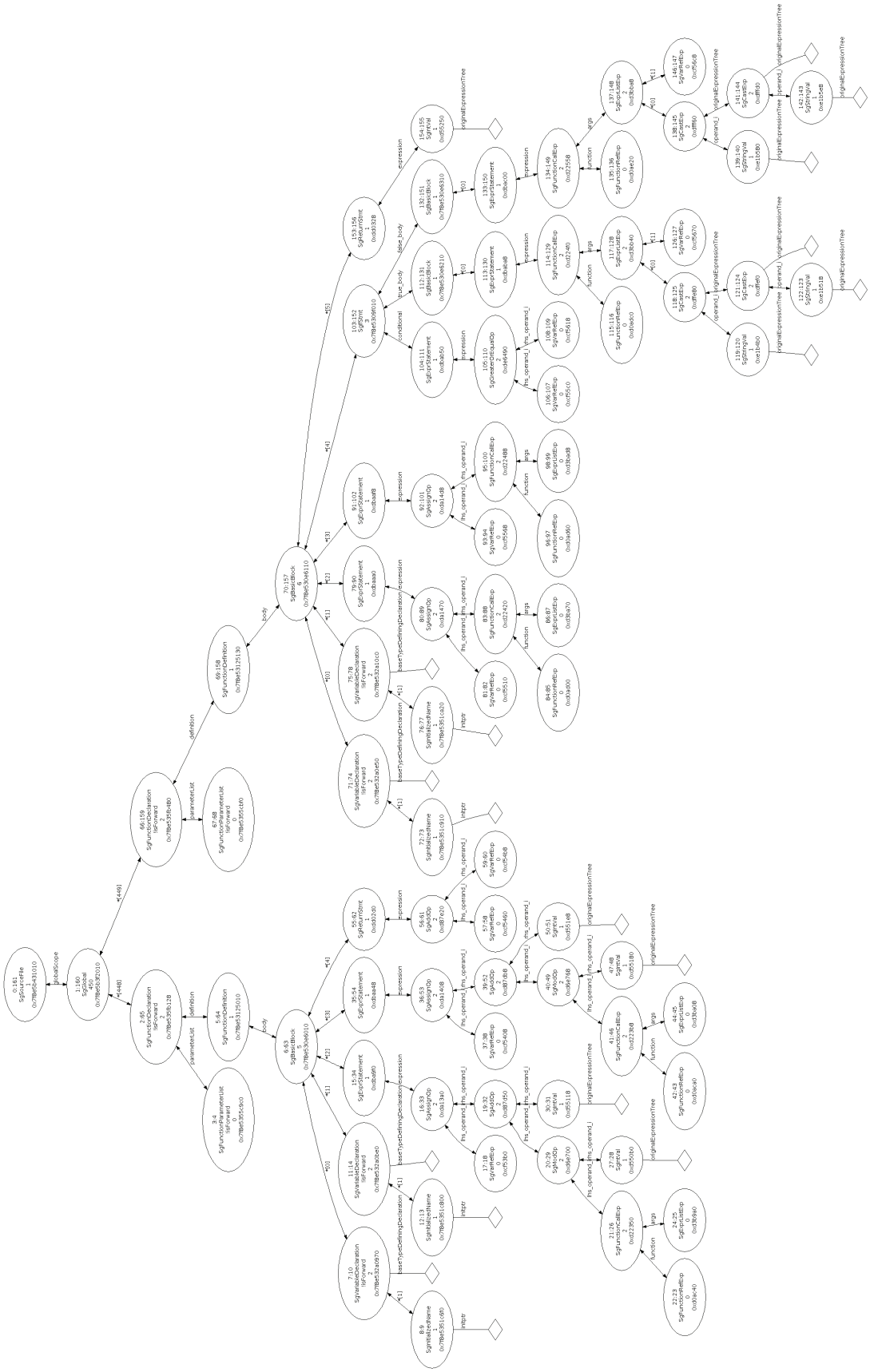


FIG. 5.5: L'AST correspondant au code exemple présenté dans la Fig. 5.1

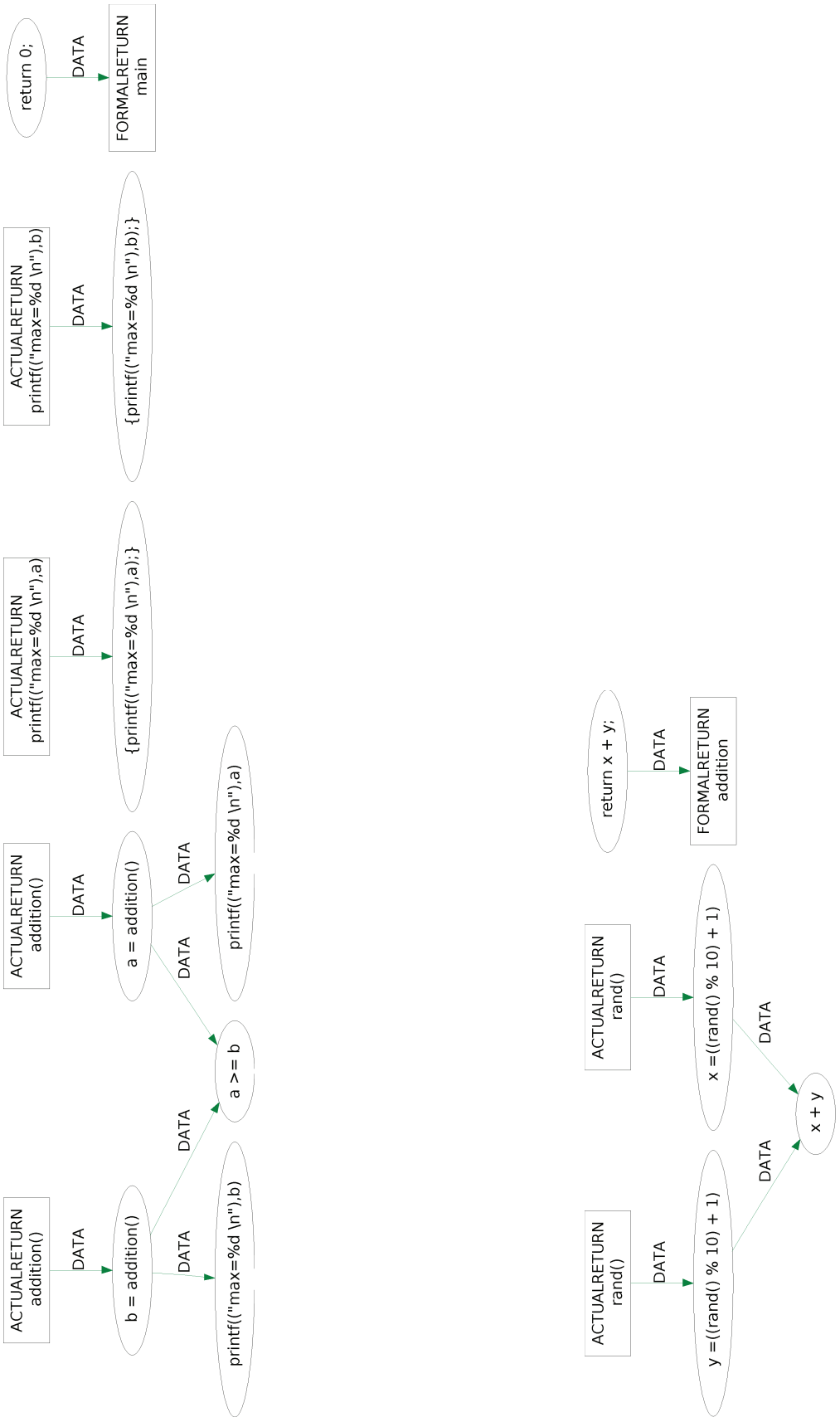


FIG. 5.6: Les DDG correspondant au code exemple présenté dans la Fig. 5.1. À gauche, le DDG de la fonction "main", et à droite, le DDG de la fonction "addition"

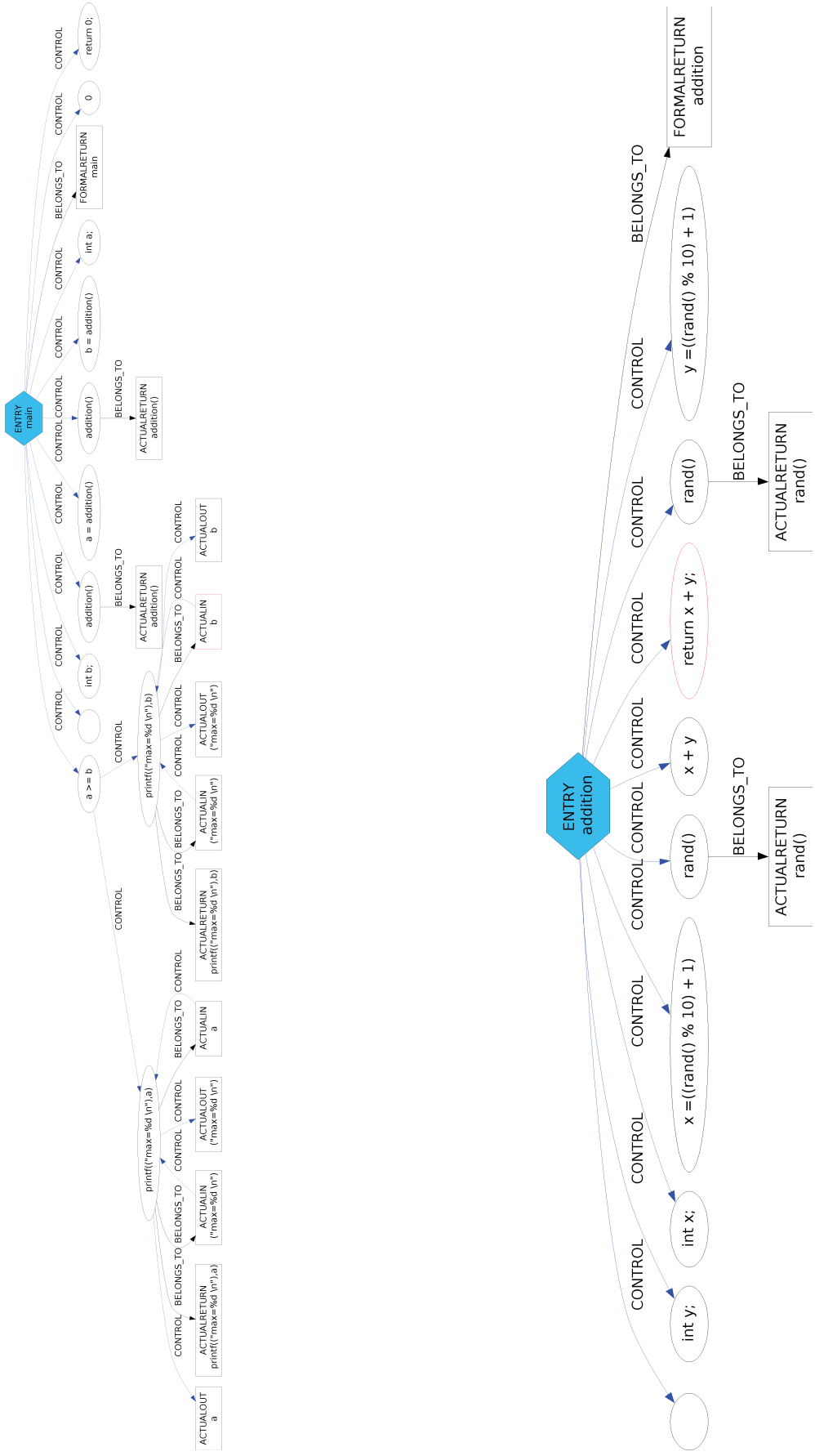


Fig. 5.7: Les CDG correspondant au code exemple présenté dans la Fig. 5.1. À gauche, le CDG de la fonction "main", et à droite le CDG de la fonction "addition"

5.4.2.4 Le *System Dependence Graph* (SDG)

Le SDG est un super-graphe, c'est-à-dire un graphe qui contient toutes les informations de DDG et de CDG dans une seule représentation intermédiaire. Nous nous intéressons à cette *IR* , car elle peut fournir à dPerf des informations liées aux dépendances de données. L'étude du SDG détient une place importante dans la réduction du *slowdown* d'une application. Par exemple, dans le cas d'une structure conditionnelle de type IF, où la condition dépend des paramètres, et non pas des constantes, dPerf utilise les informations dans SDG afin de suivre la propagation des variables à travers l'application. Parfois, les dépendances ne sont pas résolubles qu'avec le SDG, comme c'est le cas des structures *IF* où les paramètres sont dépendants des valeurs reçues d'un autre nœud de calcul. Dans cette situation, une analyse statique suivie par une exécution est nécessaire.

Pour le code d'exemple montré dans la Fig. 5.1, nous incluons seulement un extrait de SDG. Cet extrait est présenté dans la Fig. 5.8, et le SDG complet est disponible en ligne (voir [42] , sample-code-SDG-full.png).

Si le SDG n'est pas pris en compte, et si l'application analysée contient des dépendances de données, la prédiction avec dPerf nécessitera l'exécution du code instrumenté.

Toute transformation du code est réalisée dans l'AST uniquement. Les autres *IR* servent seulement à décrire la relation entres les structures syntactiques. En grandes lignes, dPerf copie l'adresse du nœud -il s'agit du nœud en tant qu'élément syntactique, et non pas en tant que machine de calcul- qui sera soumise aux modifications, puis cette adresse est recherchée dans l'AST afin de faire les modifications.

5.5 L'outil de simulation Simgrid

5.5.1 Généralités

Dans le processus de prédiction de performances avec dPerf, il existe une simulation basée sur des fichiers de trace. Pour cette simulation, dPerf utilisera les méthodes disponibles dans le *framework* Simgrid [40], un outil qui permet de simuler des topologies réseau distribuées ou bien des architectures de calcul hétérogènes. Les méthodes disponibles dans Simgrid sont également chargées de rendre le résultat final de la prédiction.

Simgrid est un projet ayant comme but d'offrir aux développeurs la possibilité de construire des simulateurs personnalisés. Avec un tel simulateur personnalisé, on peut essayer une multitude de systèmes de calcul parallèles ou distribués avec un minimum d'effort. Pour dPerf, nous avons choisi d'utiliser le module MSG, disponible dans Simgrid. Dans la structure de Simgrid (voir Fig. 5.9), MSG se trouve en dessous de la couche réservée à l'utilisateur. MSG est chargé de la simulation basée sur des fichiers de trace et dPerf contient le mécanisme nécessaire pour interagir avec ce module de Simgrid.

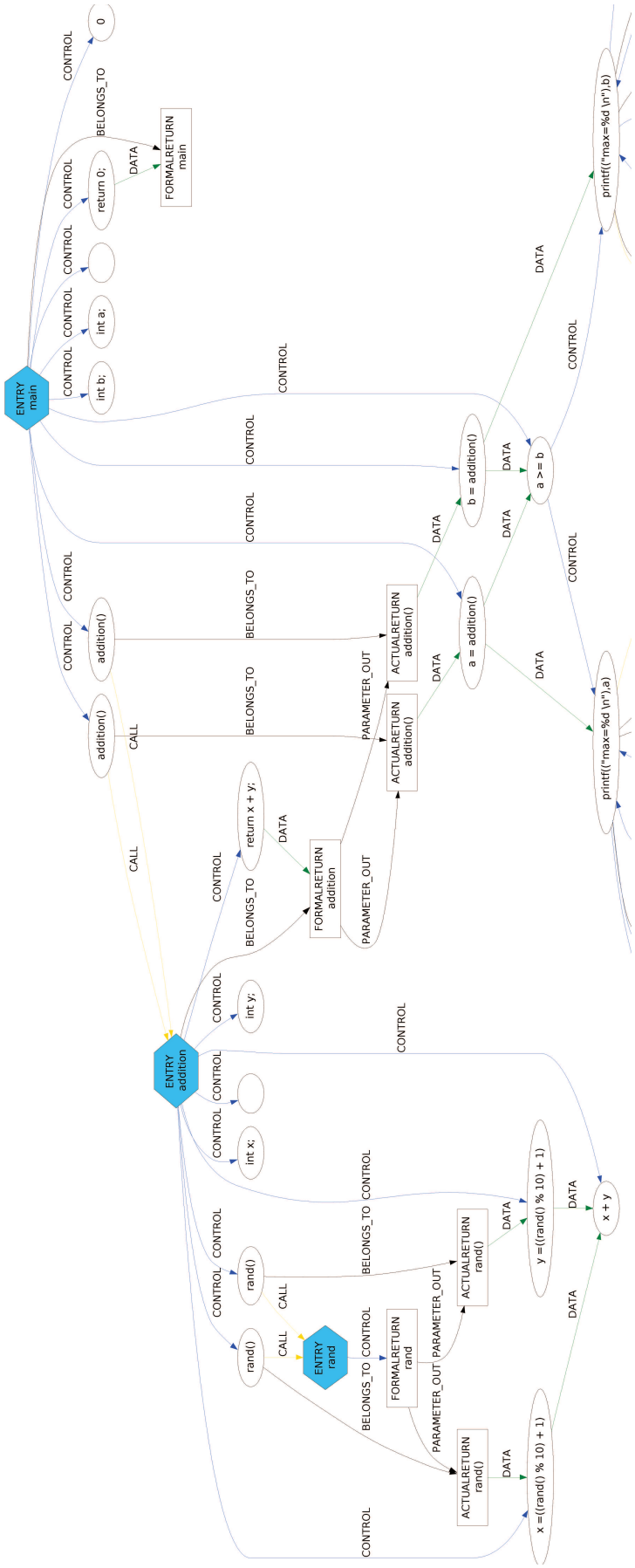


FIG. 5.8: Un extrait du SDG correspondant au code exemple présenté dans la Fig. 5.1.

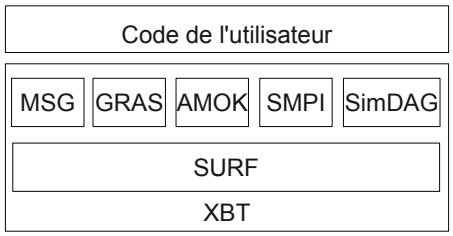


FIG. 5.9: La structure du *framework* Simgrid. dPerf utilise le module MSG de Simgrid.

Parmi les caractéristiques qui font de Simgrid un *framework* d'actualité et très puissant, nous mentionnons :

- la possibilité de définir des plateformes distribuées d'une haute complexité. Un fichier descriptif correspond à une plateforme. Ce fichier contient *la définition des machine*s et la définition des topologies réseau.
- la personnalisation du module MSG. Grâce aux actions reconnues par le module MSG, tout type de fichier de trace peut être reçu en entrée. dPerf utilise des actions déjà définies dans MSG mais aussi des nouveaux événements qui ne sont pas gérés par Simgrid par default. La puissance de calcul des nœuds est aussi personnalisable. Par exemple, nous pouvons définir la puissance de calcul en FLOPS (FLoating-point Operations Per Second), ou en une autre unité qui assurera la compatibilité entre les mesure prises et la simulation.
- le coût très faible pour accéder le résultat d'une simulation.

5.5.2 Gestion de MPI, de P2P-SAP ou d'une autre bibliothèque de communication

dPerf est développé pour prédire de performances pour les application *HPC* . Les applications concernées varient à partir de celles utilisant l'implémentation parallèle la plus simple, s'exécutant dans un système de calcul homogène. Celles-ci peuvent aller jusqu'à l'implémentation distribuée en vue d'une exécution dans un système P2P décentralisé. Pour couvrir une gamme si large des architectures système, des applications et des bibliothèques de communication, nous avons choisi d'utiliser le module MSG de Simgrid. C'est ce module qui prendra en charge l'aspect des communication entre les nœuds participants.

Le module MSG contient une liste d'événements reconnus, et de leur définition. Une définition d'événement est l'ordre des actions, et les actions elles-mêmes que Simgrid fera, afin de simuler le comportement réel d'un événement. Ces actions sont décrites comme échanges de messages entre les processus participants. Toute information utilisée dans la description d'un événement est lue du fichier de trace. Une telle ligne de trace correspond uniquement à une seule action point-à-point ou collective.

Grâce à la description des événements dans MSG, nous pouvons facilement définir des événements correspondant à des nouvelles bibliothèques de communication. De cette façon, nous avons la possibilité d'augmenter la liste d'actions MPI gérées par Simgrid et d'ajouter à la même liste les actions nouvelles correspondant au protocole P2P-SAP développé au LAAS-CNRS. L'avantage de la personnalisation du module MSG donne à

notre outil -dPerf- l'opportunité de gérer non seulement le MPI et le P2P-SAP, mais aussi de nouveaux protocoles et des bibliothèques de communication.

En utilisant les fonctions du *framework* Simgrid, dPerf peut prédire les performances des applications destinées aux systèmes *HPC* homogènes ou hétérogènes. Avec Simgrid et son module MSG, un système de calcul est définit à l'aide de deux fichiers XML. Un fichier contient une liste de nœuds et de leur identificateur qui sera utilisée pendant la simulation. Le deuxième fichier XML définit la puissance de chaque nœud ainsi que la topologie réseau utilisée. Les développeurs de Simgrid ont mis à la disposition des utilisateurs un outil pour générer automatiquement quelques systèmes *HPC* . Cet outil est particulièrement utile pour ceux qui n'ont pas besoin de gérer tous les détails du système utilisé.

Chapitre 6

Terminologie, méthodologie et implémentation

Le domaine de l'*HPC* a besoin d'outils polyvalents de prédiction de performances. Pour ceci, nous allons présenter notre façon de répondre à la problématique posée tout au long de ce travail de recherche : "Comment prédire les performances des applications de calcul distribué dans des conditions réelles sur des systèmes hétérogènes P2P ou la communication a lieu de façon décentralisé ?"

La polyvalence d'un outil de prédiction de performances s'exprime par la prise en compte de la multitude d'applications et d'architectures de calcul existantes et émergentes. Les applications *HPC* se différencient selon :

- le langage utilisé pour les développer, ;
- le formalisme de communication utilisé pour tout échange de messages ;
- l'architecture de calcul prévue pour leur déploiement (voir le paragraphe suivant) ;
- la topologie logique : centralisée ou décentralisée.

À notre connaissance, il n'existe pas d'outil de prédiction qui prenne en compte plus d'un langage et qui utilise plusieurs types de formalismes de communication.

En ce qui concerne les architectures de calcul de haute performance, elles varient, en général, selon :

- le degré d'homogénéité des nœuds et/ou de la topologie réseau ;
- le degré d'hétérogénéité des nœuds et/ou de la topologie réseau ;
- la topologie logique :
  - centralisée ;
  - décentralisée ;
- l'architecture de calcul :
  - parallèle ;
  - distribuée ;

Un outil de prédiction de performance doit trouver en permanence un compromis entre tous les facteurs liés à l'application et au système. Pour le développement de dPerf, nous avons recherché le meilleur compromis pour que dPerf gère :

- les applications écrites en C, C++, ou Fortran ;
- les applications communicants avec :
  - MPI,
  - P2P-SAP,
  - ou un autre formalisme de communication, à définir dans dPerf ;
- les applications censées s'exécuter sur une architecture de calcul homogène ou hétérogène ;
- les applications développées pour un environnement de calcul parallèle ou distribué, utilisant une topologie logique centralisée ou décentralisée ;

La façon dont dPerf est conçu et implémenté, assure la production (ou le calcul) précis et rapide de prédictions de performances, grâce à deux techniques de *benchmarking* par blocs d'instruction[43], qui permettent d'avoir un *slowdown* très réduit et donc un gain de temps important.

6.1 La terminologie

Cette section décrit la terminologie employée dans le processus de prédiction de performances de dPerf.

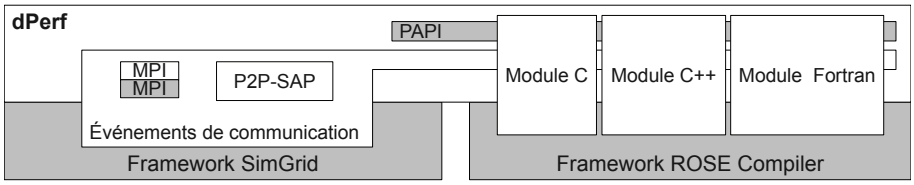


FIG. 6.1: L'outil dPerf. En gris, la bibliothèque PAPI et les outils ROSE et SimGrid développés en externe. En blanc, l'élargissement (en vue de la compatibilité avec nos besoins) et le développement faisant partie de dPerf.

6.1.1 Les temps mesurés ou estimés

À travers le processus de prédiction de performances avec dPerf, les temps suivants, dont une partie est montrée figure 6.2, sont, soit mesurés, soit calculés :

- $t_{execution\,normale}$ est le temps nécessaire à une exécution complète, non-modifiée, de l'application analysée. La valeur mesurée est celle correspondante à la durée de l'exécution de l'application du début et jusqu'à la fin. Pendant la phase expérimentale, cette valeur sert de référence pour valider les prédictions faites avec dPerf.
- $t_{computation}$ est le temps de calcul pris par le processeur pour calculer un bloc d'instructions. La valeur de ce temps est le résultat d'une mesure effectuée avec dPerf en utilisant les compteurs hardware et la bibliothèque PAPI, présentés antérieurement

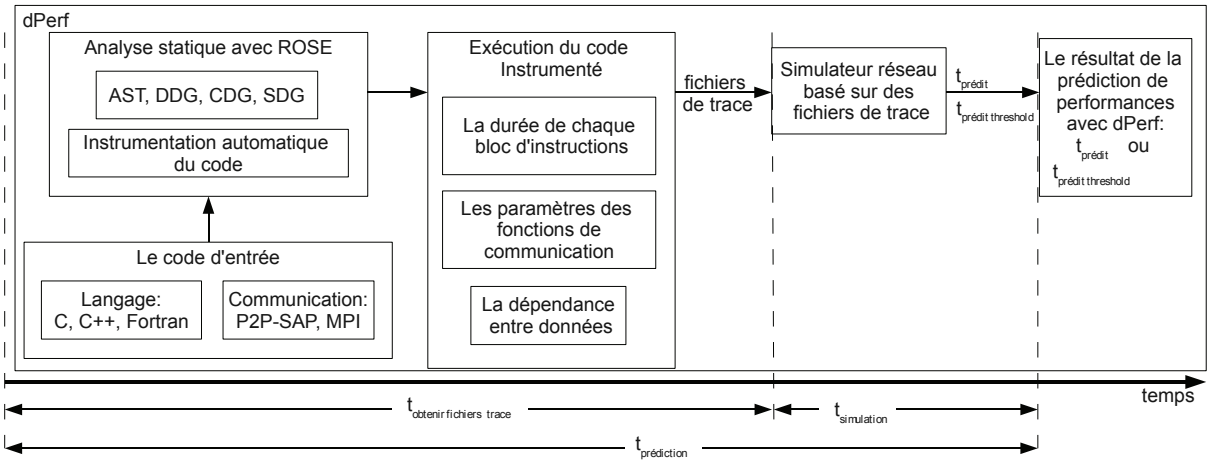


FIG. 6.2: Le processus de prédiction avec dPerf. La durée de chaque étape est montrée sur la ligne du temps.

(voir chapitre 5). Ce temps correspond à l'ensemble des instructions contenues entre deux lectures des compteurs hardware : une lecture représentant le commencement de la prise de mesure, et l'autre lecture représentant la fin de la prise de mesure. $t_{computation}$ contient la durée de calcul prise par le processeur dans le créneau réservé uniquement au traitement de l'application analysée, et non pas au cycle réel du système. $t_{computation}$ est une valeur mesurée en mode "virtuel" (voir figure 5.2). De cette manière, $t_{computation}$ ne contiendra pas le temps que le processeur attribuait au traitement des autres applications ou processus du système.

- $t_{communication}$ est le temps nécessaire aux échanges d'information entre les processus employés par l'exécution de l'application, soit les processus participants au calcul.
- $t_{prediction}$ est le temps requis pour obtenir un résultat de prédiction (voir figure 6.2). Autrement dit, $t_{prediction}$ est le temps nécessaire pour parcourir toutes les étapes du processus de prédiction, qui permettent d'accéder à la valeur t_{predit} (voir la définition ci-dessous).
- $t_{prediction\,threshold}$; similaire à $t_{prediction}$, sauf que dPerf applique la technique optimisée de *benchmarking* , qui réduit $t_{prediction}$. Donc, $t_{prediction\,threshold} < t_{prediction}$.
- $t_{simulation}$ est le temps d'une simulation à partir des fichiers de trace. Ce temps ne prend pas en compte la durée de la réduction du *slowdown* . $t_{simulation}$ est donné par la durée d'exécution du module MSG de Simgrid pour une plateforme spécifique et pour un jeux de fichiers de trace (voir figure 6.2).
- $t_{obtenir\,fichiers\,trace}$ est le temps nécessaire pour obtenir les fichiers de trace. Pour obtenir ce temps, les compteurs hardware sont lus au début et à la fin de l'exécution du code instrumenté, moment qui représente l'obtention des fichiers de trace (voir figure 6.2).
- t_{predit} est le résultat de la prédiction faite par dPerf, c'est-à-dire le temps d'exécution du code d'entrée prédit par dPerf quand la méthode simple de *benchmarking* des blocs d'instructions est utilisée.

- $t_{predit\,threshold}$ ou le temps prédit par dPerf selon la technique optimisée de *benchmarking* des blocs d'instructions, qui utilise un seuil suffisant pour avoir une estimation précise du temps d'un bloc.

6.1.2 L'application d'entrée

L'application d'entrée est le code source de l'application à analyser afin de rendre une prédiction sur ses performances d'exécution. Dans dPerf nous avons utilisé les outils et les bibliothèques d'analyse de codes source les plus adaptés au développement d'un nouveau notre outil de prédiction de performance qui répond à la problématique de ces travaux de thèse. Ceux-ci sont présentés dans le chapitre 5. Le principal but de ce travail de recherche est de proposer une technique de prédiction de performances des applications de calcul distribuées exécutées dans un environnement décentralisé d type pair-à-pair. Les prédictions de performances qui seront fournies doivent doivent être faites par rapport aux conditions réelles de calcul intensif pair-à-pair, pour un *slowdown* diminué et avec la possibilité d'extrapolation des résultats.

6.1.2.1 Des multiples langages de programmation : C, C++, Fortran

Pour modéliser une gamme variée d'applications, nous avons développé dPerf en utilisant l'outil ROSE (voir chapitre 5.4 pour les fonctions de ROSE utilisées dans le développement de dPerf). De cette façon, nous fournissons des résultats de prédiction pour toute application écrite en C, C++ ou Fortran. Les nombreux langages de programmation acceptés pourra être plus large, mais nous n'avons ciblé que les trois langages mentionnés précédemment car ils sont les plus utilisés en *HPC* . Dans la figure 6.1, la vue empilée de dPerf montre les trois modules de dPerf qui seront utilisés en fonction du langage détecté en entrée. Chacun des modules - Module C, Module C++ et Module Fortran - correspond à l'un des trois langages.

Le fait d'accepter en entrée et d'analyser trois langages parmi ceux les plus utilisés dans la programmation *HPC* donne un important avantage à notre outil car à notre connaissance, il n'y a pas d'outil de prédiction de performances qui gère plus d'un langage. Il existe parfois des outils qui sont conçus non pas pour toute application utilisant un langage particulier, mais uniquement pour une application d'intérêt pour l'équipe développant l'outil. Cette caracteristique de dPerf est visible aussi dans la colonne 7 ("Langage") des tableaux 1.1, 2.1, et 3.1.

6.1.2.2 Des multiples formalismes de communication : MPI, P2P-SAP

La diversité d'applications pouvant être reçues en entrée par dPerf est augmentée grâce à la manière que nous avons choisie pour développer dPerf : en combinant des méthodes de ROSE et de Simgrid. De cette façon, dPerf a pu être programmé pour qu'il accepte non seulement différents langages de programmation, mais aussi différents formalismes de communication entre les processus participants. En ce qui concerne le formalisme de communication, que celui-ci soit standardisé ou non, le principe de prédiction est le suivant :

1. dPerf analyse d'une façon statique le code d'entrée pour identifier les instructions de communication. Les appels des fonctions de communication sont "isolées" du code séquentiel. Cette séparation rendra un fichier de trace, résultat d'une exécution du code instrumenté.

2. Le fichier de trace est passé en entrée de Simgrid. Le module MSG de Simgrid est utilisé pour pouvoir lire les détails de chaque fonction de communication identifiée auparavant par dPerf. À ce point, nous pouvons définir dans le module MSG n'importe quel formalisme de communication, que Simgrid gèrera sans difficulté.

dPerf est conçu pour rendre des résultats de prédiction pour des applications anciennes -séquentielles, parallèles sur des systèmes homogènes ou hétérogènes- et pour adresser les nouvelles applications qui sont destinées aux environnements de calcul P2P. Autrement dit, l'objectif de dPerf inclut le support pour le calcul de haute performance dans un environnement P2P, et dans notre cas ceci est réalisé par l'environnement P2Pdc (le *P2P decentralized computing environment*) développé au laboratoire LASS-CNRS à Toulouse, France. Cet environnement de calcul distribué de haute performance en P2P utilise son propre protocole de communication nommé P2P-SAP (*P2P Self Adaptive Protocol*) [65], [49], [80]. P2P-SAP est un protocole qui fonctionne dans l'environnement P2Pdc ayant comme but la réalisation de calculs de haute performance d'une façon décentralisée en utilisant les ressources - la puissance de calcul- d'une infrastructure pair-à-pair.

A part une exception, faire de prédictions de performances pour les applications *HPC* destinées aux architectures de calcul P2P est, selon nos connaissances, sans précédent. L'exception est l'outil P2PPerf[50, 51] (voir section 3.8), qui est développé au laboratoire LIFC (à Montbéliard, en France) et qui modélise les applications écrites en Java communicant en utilisant JNGI. Ces faits confient à dPerf un avantage considérable face aux outils d'aujourd'hui.

dPerf fonctionne également avec les bibliothèques au standard MPI (*Message Passing Interface*) [8]. Dans la figure 6.1, nous pouvons remarquer que dPerf contient les modules pour le C, le C++ et le Fortran. Nous avons développé ces modules pour qu'ils puissent être "visibles" et accessibles depuis les événements de communications reconnus par Simgrid et depuis les méthodes de Rose que dPerf utilise pendant la réduction du *slowdown* . En effet, tout événement de communication reconnu par dPerf au moment de la réduction du *slowdown* doit être impérativement reconnu par Simgrid, quels que soient la bibliothèque ou le protocole de communication utilisés.

6.1.3 Le *slowdown*

Il constitue l'unité de mesure de l'efficacité des outils de prédiction. Le *slowdown* est défini comme étant le rapport entre (i) le temps nécessaire pour obtenir une prédiction, et (ii) le temps d'une exécution réelle d'une application. Le *slowdown* est exprimé par processus simulé, à partir des mesures prises sur une même architecture.

$$slowdown = \frac{t_{prediction}}{t_{execution\,normale} \times Nb_Processus} \tag{6.1}$$

où

$$t_{prediction} = t_{obtenir\ fichiers\ trace} + t_{simulation} \tag{6.2}$$

Dans les chapitres 1 à 3, nous avons classifié les outils de prédiction de performances en (i) analytiques, (ii) basés sur des mesures, et (iii) hybride. La majorité des outils analytiques ont un *slowdown* inférieur à l'unité, et nous pouvons parler d'un gain plutôt que d'un *slowdown* . Les méthodes dans les deux catégories suivantes, (ii) et (iii), ont un *slowdown* supérieur à 1, et donc n'ont pas de gain.

Le gain est l'inverse du *slowdown* , et il représente la rapidité du processus de prédiction par rapport à l'exécution normale d'une application, soit la même application pour laquelle un outil fournit la prédiction.

$$gain = \frac{1}{slowdown} \tag{6.3}$$

Quand un outil de prédiction fournit la prédiction (pour une application d'entrée donnée) plus rapidement que le temps pris par l'exécution normale (de la même application), il est caractérisé par un gain. Contrairement à cette caractéristique, quand une application prend moins de temps pour s'exécuter que l'outil prend pour fournir la prédiction, cet outil sera caractérisé par un *slowdown* .

6.1.4 Le coût de la prédiction

Il s'agit du coût, mesuré en unités de temps, pour accéder au résultat de la prédiction. Le coût de la prédiction - $t_{prediction}$ - est la somme des temps $t_{obtenir\ fichiers\ trace}$ - la transformation du code d'entrée, l'exécution du code transformé (afin d'obtenir les fichiers de traces)- et $t_{simulation}$ - la simulation basée sur des fichiers de trace (en vue de l'obtention du résultat de la prédiction). Selon les termes définis dans le chapitre 6.1.1 et présentés dans la figure 6.2, le coût d'une prédiction faite par dPerf est :

$$t_{prediction} = t_{obtenir\ fichiers\ trace} + t_{simulation} \tag{6.4}$$

6.1.4.1 Technique simple de *benchmarking* par bloc d'instructions

Elle représente la méthode pour mesurer le temps pris par un micro-processeur afin de traiter, ou calculer, un ensemble de lignes de code. Sur l'ensemble d'une application, le temps total de calcul séquentiel - soit sans communication avec d'autres processeurs - est :

$$t_{computation} = \sum_{i=0}^{n} t_{computation\ bloc_i} \tag{6.5}$$

où $t_{equation\ bloc_i}$ correspond au temps de chaque bloc d'instructions délimité par deux éléments parmi ceux-ci : le début du code analysé, le début d'une communication, la fin d'une communication, ou la fin du code analysé.

Cette technique implique l'ajout de lignes de code afin de pouvoir prendre les mesures de blocs d'instructions. Les instructions du code d'entrée ne sont pas modifiées, ce qui n'est pas le cas dans la technique optimisée, présentée ci-dessous.

6.1.4.2 Technique optimisée de *benchmarking* par bloc d'instructions

Cette technique que nous appelons la méthode du seuil ou du *threshold* a le même but que la technique simple de *benchmarking* que nous proposons, avec cependant une différence : la technique optimisée, si elle est applicable, réduira considérablement le coût de la prédiction.

La méthode du seuil implique la modification des lignes d'instructions originales. Plus précisément, cette méthode cherche les structures répétitives principales non- interdépendantes, les boucles les plus extérieures, afin de réduire leur nombre d'itérations en dessous d'un seuil. Ce seuil doit garantir que dPerf puisse estimer avec précision le temps total de la boucle, soit le temps de la boucle si elle avait fini son exécution sans aucune modification.

Nous avons présenté ces deux techniques, simple et optimisée, de *benchmarking* par bloc d'instructions précédemment dans [43]. Nous avons présenté plus de détails concernant ces méthodes ainsi que la façon dont nous les avons implémentées dans les chapitres 6.2.1.3 et 6.2.1.4. Nous nous sommes inspirés pour ces deux techniques de ChronosMix [38] et de P2PPerf [50, 51], mais nous les avons développées en concordance avec les tendances du domaine de l'*HPC* .

6.1.5 La précision

Cette thèse a le but de prédire les performances des applications réelles de calcul distribué. Les performances doivent être calculées avec une précision augmenté afin que les scientifiques puissent les utiliser pour améliorer leurs applications.

Pour fournir des prédictions très précises, cette thèse propose une technique de *benchmarking* simple par bloc d'instructions. Pour certains applications qui le permet, nous proposons également une technique de *benchmarking* optimisé par bloc d'instructions. Ces technique sont intégrées dans l'outil dPerf, une implémentation de la contribution présenté dans ce manuscrit. Un des objectifs de dPerf est de prédire les performances des applications analysées avec un taux d'erreur assez faible, dans des conditions réelles. Pour ceci, notre contribution prend en compte les différentes optimisations faites par le compilateur, puis une des techniques de *benchmarking* par bloc d'instructions est appliquée et des résultats extrapolable sont en fin obtenues. L'approche de la prédiction de performances ainsi que la précision des cette contribution sont présenté dans les sections suivantes.

6.1.6 Les systèmes et les topologies ciblés par dPerf

L'architecture de calcul est composée de l'ensemble de machines de calcul, ou de nœuds, employés dans un processus de calcul séquentiel, parallèle, distribué.

Toute application exécutée représente un calcul que la machine doit réaliser, e.g. un calcul mathématique, une image à transformer, un fichier multimédia à rendre à la sortie (audio ou vidéo), etc. On peut avoir plusieurs types d'architectures de calcul. La classi-

fication - en version étendue par rapport à la figure 4.0.1- peut être vue dans la figure 6.3. Les entreprises et les scientifiques demandent de plus en plus de puissance de calcul. C'est la raison pour laquelle les chercheurs et les développeurs de systèmes et outils *HPC* se sont orientés vers les réseaux P2P et, plus récemment, le *cloud*. Ces deux architectures de calcul peuvent faire partie de deux environnements de calcul complètement différents :

- a) Les environnements stables, comme c'est le cas des systèmes *HPC* classiques. Les systèmes faisant partie de cette catégorie, étaient présentés dans l'introduction de la deuxième partie de ce document. Une application exécutée dans un environnement stable peut utiliser une topologie logique centralisée ou décentralisée. Celle centralisée implique la présence d'un nœud "maitre" -ou central, d'où le terme "centralisé"- parmi les nœuds participants au calcul. Ce nœud maître a pour rôle de distribuer les tâches aux nœuds participants -souvent appelés travailleurs (ou *workers*)- et de centraliser le résultat du calcul effectué par chacun des nœuds. La topologie décentralisée est utilisé pour diverses configurations ne nécessitant pas cet élément centralisateur, comme une topologie en maille en 2 dimensions, par exemple. Dans le domaine de l'*HPC* , le terme "décentralisé" représente la hiérarchie de dépendances entre tous les nœuds participants au calcul. Un calcul décentralisé se fait sans la présence d'un nœud "maître".
- b) l'environnement volatile P2P. Les systèmes de ce type utilisaient, jusqu'à présent, une topologie de communication centralisée. Une nouvelle approche de calcul de haute performance commence à exploiter le potentiel des réseaux P2P d'une manière complètement décentralisée. Cet environnement se nomme P2Pdc[80, 44] et il est développé, au même temps que le protocole P2P-SAP[49, 44], au laboratoire LAAS-CNRS à Toulouse par The Tung NGUYEN et Didier EL-BAZ dans le cadre du projet ANR-CIP[1].

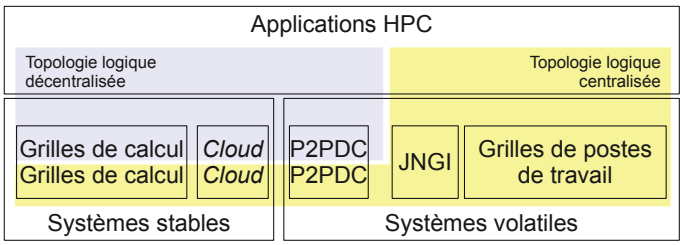


FIG. 6.3: La classification des systèmes HPC en fonction de leur disponibilité. Présentation de la topologie de communication employée par les applications HPC ainsi que les systèmes potentiels acceptant chacune des topologies.

Le but de cette thèse est de réaliser un outil de prédiction de performances pour les applications exécutées dans une architecture de calcul contenant plusieurs nœuds. Bien évidemment, l'approche que nous présentons ici fonctionne également pour les architectures de calcul exécutant du code séquentiel sur une seule machine, mais cet aspect ne fait pas l'objet de notre étude.

Dans le domaine de calcul de haute performance, on exclut l'idée de résoudre des problèmes scientifiques sur des postes de travail indépendants qui n'ont pas d'échanges de données avec d'autres machines. Nous appelons ces machines indépendantes car, en général, elle vont exécuter des applications séquentielles. Avec le développement des microprocesseurs multi-cœurs, nous pouvons exécuter, bien évidemment, des applications parallèles sur une même machine, mais les performances restent très réduites. Comme nous sommes intéressés particulièrement par les applications qui font du calcul intensif, e.g. les applications scientifiques, les applications commerciales du domaine de la logistique, etc. Cette thèse a comme but d'offrir aux scientifiques un outil capable de prédire les performances des applications *HPC* destinées à l'exécution sur un des systèmes dans la figure 6.3, avec cible prioritaire sur les architectures P2P. A part les systèmes *cloud* et ceux utilisant JNGI, tout autre type d'architecture *HPC* peut être modélisé avec dPerf. Les applications exécutées sur ces systèmes peuvent utiliser une topologie centralisée ou décentralisée. Une caractéristique inédite de dPerf est sa capacité de fournir des prédictions pour les applications utilisant une topologie décentralisée P2P. La figure 6.3 montre la présence d'un seul environnement décentralisé capable de gérer l'exécution d'une application HPC P2P : le P2Pdc[80, 44]. Cet environnement et le protocole de communication P2P-SAP sont développés par une même équipe afin de proposer une solution complète et optimisée pour le calcule HPC P2P. Notre outil modélise avec succès non seulement les application qui communiquent avec MPI centralisé ou décentralisé, mais aussi celles utilisant la topologie décentralisée, exécutées dans l'environnement P2Pdc

6.2 La méthodologie

Nous allons parcourir les différentes méthodes proposées dans cette thèse, que nous avons réunies dans un seul outil de prédiction de performance, c'est-à-dire dPerf. Nous allons présenter le rôle de chacun des éléments décrits dans le chapitre 5 à travers le processus de prédiction avec dPerf.

Dans la suite de cette section, nous présentons l'approche de dPerf pour obtenir des prédictions de performances précises. Le flux de travail (*work-flow*) de dPerf est visible dans la figure 6.4.

6.2.1 Réduction du *slowdown*

La réduction du *slowdown* implique la prise en compte du niveau d'optimisation du compilateur ainsi que la séparation des instructions de calcul de celles de communication.Des blocs d'instructions sont formés entre le début du code, les communications, et la fin du code.La durée de calcul des blocs d'instructions est obtenue.La durée de communication est obtenue dans la phase de simulation basée sur des fichiers de trace. Le résultat final de la prédiction est obtenu en faisant la somme des temps calculés à chaque étape de dPerf.

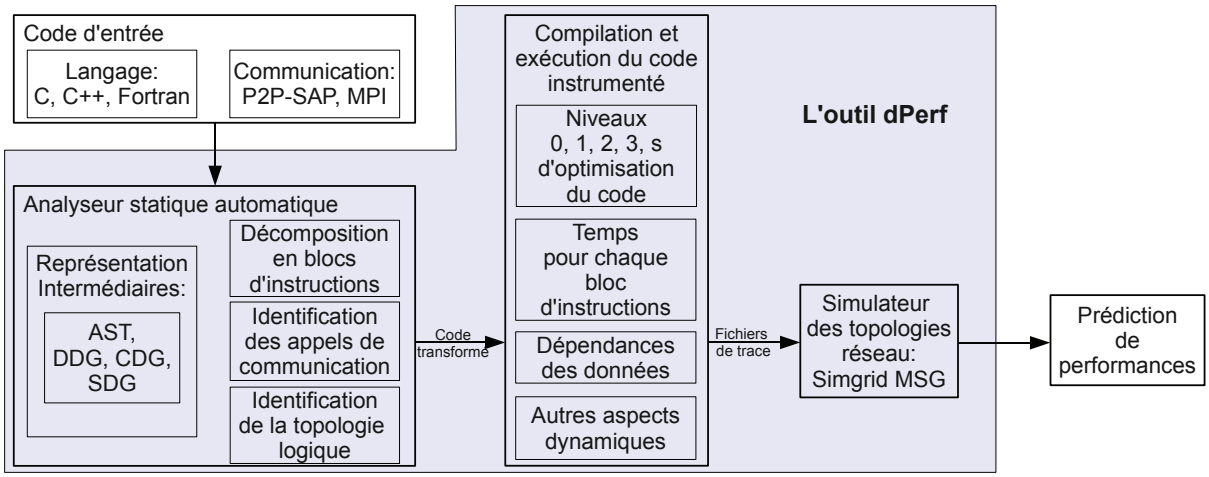


FIG. 6.4: Le flux de travail dans dPerf. Le code source d'une application est reçu en entrée, et une prédiction de performances est disponible à la sortie de dPerf.

6.2.1.1 Choisir le code d'entrée

Pour fournir une prédiction de performances, dPerf doit recevoir, en vue d'une analyse syntaxique, le code source d'une application *HPC* écrite en C, C++, ou Fortran, dont la communication se fait en MPI ou P2P-SAP. Les contraintes que nous venons de mentionner permettent à dPerf de couvrir une plage d'applications *HPC* beaucoup plus large que les autres outils de prédiction de performance. Ces contraintes de langage de programmation et de formalisme de communication sont imposées par les outils -ROSE et Simgrid. Plus précisément, dPerf a une contrainte pour les langages de programmation imposés par ROSE (voir le chapitre 5.4), l'outil dont les fonctions d'analyse et de transformation source-à-source se trouvent à la base de dPerf (voir figure 6.1). La contrainte relative au formalisme de communication est imposée par le module MSG de Simgrid (voir la description dans le chapitre 5.5). Le type du code d'entrée accepté par dPerf est montré dans la figure 6.4. Le choix de l'application à analyser représente la première étape dans le flux de travail de notre outil.

6.2.1.2 Les niveaux d'optimisation du compilateur

En collaboration avec J. Vienne[115] du laboratoire LIG de Grenoble, nous avons étudié l'effet des différents niveaux d'optimisation d'un code par les compilateurs G++ et ICPC. Cette étude était menée sur un programme d'exemple. Ce code était compilé en choisissant les niveaux "0", "1", "2", "3" d'optimisation (voir [59]pour G++, et [64] pour ICPC). Les particularités des différents niveaux d'optimisation sont présentées dans le tableau 6.1.

Nous avons expérimenté avec les deux compilateurs et les différents niveaux d'optimisation sur trois architectures différentes de calcul dans le but de voir à quel point les mesures d'une boucle dépendent de l'architecture, et s'il y a moyen de prendre des mesures d'une façon portable. La figure 6.5 présente les courbes du temps $t_{moyen_{iteration}}$ pour le compilateur ICPC, à tous les niveaux d'optimisation et sur trois architectures Intel :

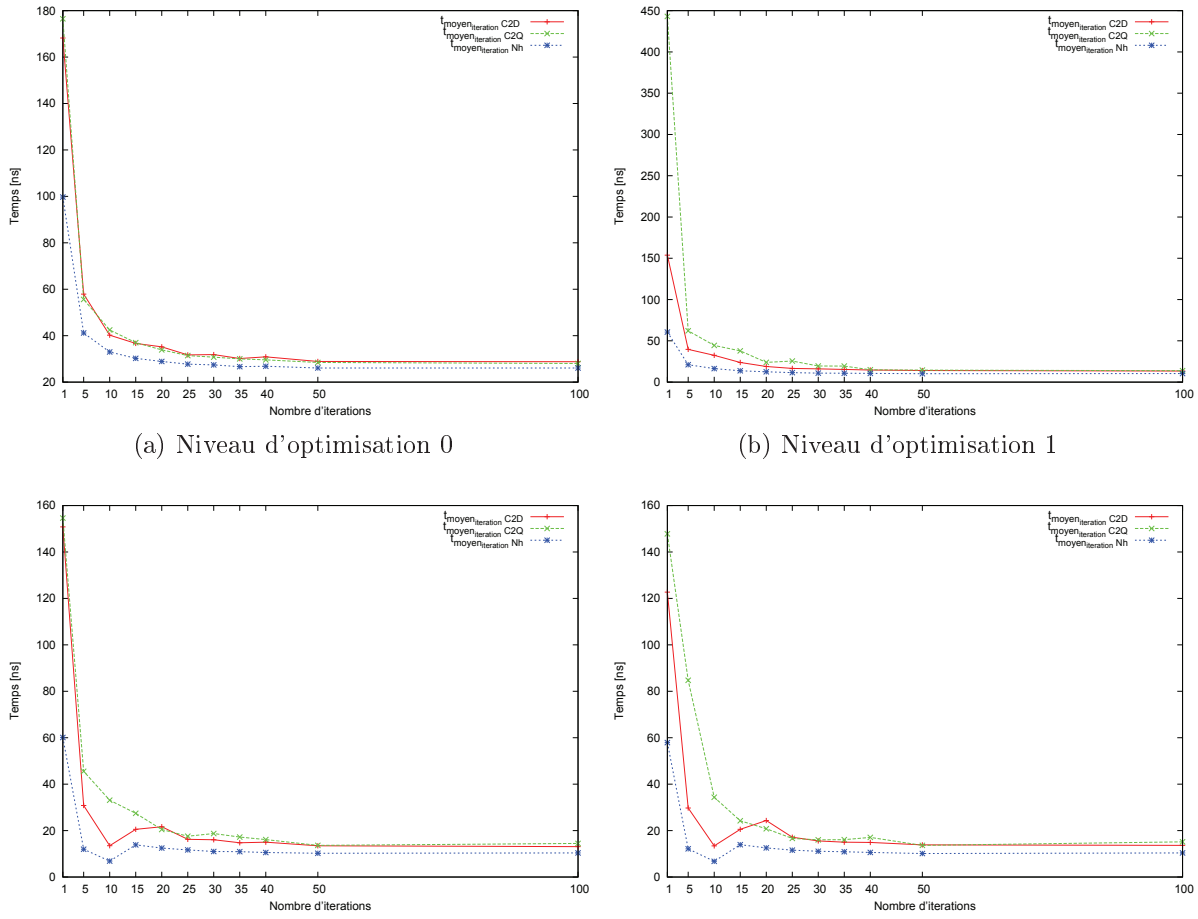
TAB. 6.1: Vue sur les principaux niveaux d'optimisation des compilateurs G++ et ICPC. [1] Le compilateur ;

Comp.[1]	Optimisation				
	-O0	-O1	-O2	-O3	-Os
G++	réduction du temps de compilation et debugging du code	option implicite ; réduction de la taille et du temps d'exécution ; temps de compilation non augmenté	toutes les optimisations sans compromis entre l'espace et la vitesse ; performances augmentées ; temps de compilation élevé	-O2 et optimisation des boucles	optimisation de la taille du binaire et toutes les optimisations du -O2 qui n'augmentent pas la taille du binaire
ICPC	sans optimisations (désactivées)	avec optimisations	avec optimisations (option implicite)	-O2 plus d'autres optimisations plus agressives au niveau des boucles ; ne s'applique pas pour toutes les applications	niveau non-disponible

Core 2 Duo (C2D), Core 2 Quad (C2Q), et Nehalem (Nh). Dans le cas de g++ (dont les résultats ne sont pas présentés ici), un cinquième niveau d'optimisation (-Os) est pris en compte par dPerf. Les résultats expérimentaux présentés dans la partie III de ce document, utilisent le compilateur g++. Les courbes montrent clairement qu'un *benchmark* des blocs d'instructions reste dépendant de l'architecture, et donc dPerf nécessitera au moins un nœud de calcul de chaque type qui fera partie de l'architecture de calcul cible.

Cette étude confirme que les valeurs des $t_{moyen_{iteration}}$ d'une boucle donnent des courbes identiques -pour les niveaux 0 et 1 d'optimisation- et proches -pour les niveaux d'optimisation 2 et 3. Ces courbes montrent que la technique optimisée de *benchmarking* par blocs d'instructions peut être appliquée de même façon sur un code source compilé avec G++ ou avec ICPC, pour n'importent quels niveaux d'optimisation. La réduction du *slowdown* tient compte du compilateur, et de l'impacte du niveau d'optimisation choisit au moment de la compilation, sur la prédiction de performances d'un code source.

Nous avons décidé de doter dPerf avec le support nécessaire aux prédictions de performances quelle que soit l'optimisation de compilation ciblée par le développeur ou l'utilisateur de l'application. Ainsi, dPerf compile le code transformé avec un niveau d'optimisation au choix. Les détails obtenus de cette manière, ainsi que le contenu des fichiers de traces seront en concordance avec le niveau d'optimisation ciblé.



(a) Niveau d'optimisation 0

(b) Niveau d'optimisation 1

(c) Niveau d'optimisation 2

(d) Niveau d'optimisation 3

FIG. 6.5: Résultats d'une étude sur l'impact des optimisations du compilateur au niveau du temps moyen par itération d'une boucle. Expériences faites avec un code simple contenant une structure répétitive avec réutilisation des variables en fin du programme. Compilateur utilisé : ICPC

Le flux de travail de dPerf est visible dans la figure 6.4. La réduction du *slowdown* est réalisée de façon entièrement automatique. dPerf utilise des fonctions existantes dans ROSE pour obtenir les représentations intermédiaires (*IR*) du code d'entrée. La plus simple *IR* générée dans cette phase est l'AST (*Abstract Syntax Tree*), un arbre syntaxique abstrait (voir la description dans le chapitre 5.4). À partir de l'AST, dPerf peut aussi obtenir, toujours à l'aide des méthodes définies dans ROSE, le DDG, le CDG, ou le SDG. Nous rappelons que le SDG est une représentation intermédiaire contenant toutes les dépendances de contrôle et de données, ainsi que les chemins de propagation des variables, dans le code source analysé.

Pour la prédiction faite avec dPerf, l'analyse de l'AST est prioritaire. Le code d'entrée est décomposé dans des blocs simples d'instructions. Ici, interviennent les modules C, C++, ou Fortran (voir figure 6.1), en fonction du langage utilisé. Chacun des blocs est vérifié pour voir si des appels de communication sont trouvés parmi les instructions.

Les trois modules que nous avons développés pour dPerf, le module C, le module C++, et le module Fortran visibles dans la figure 6.1, prennent en charge l'identification des lignes clés d'instructions. Dans chacun de modules, il y a une liste avec les événements, ou les noms des fonctions de communication pour chaque formalisme supporté. À l'état actuel de développement de dPerf, les appels de communication de la bibliothèque MPI et du formalisme P2P-SAP sont identifiés avec succès, d'autres pouvant facilement être définis. L'identification de ces instructions marque le passage d'un code séquentiel à une communication, soit l'interruption temporaire des instructions séquentielles, et le début d'un échange d'information avec un autre nœud impliqué dans le calcul *HPC* . C'est à ce moment de la réduction du *slowdown* que la bibliothèque PAPI est utilisée. La figure 6.6 montre les endroits dans le code où dPerf insérera les appels vers la bibliothèque PAPI. Cette bibliothèque est responsable de la prise des mesures à l'aide des compteurs hardware, après le processus automatique d'instrumentation. Les compteurs hardware accédés avec

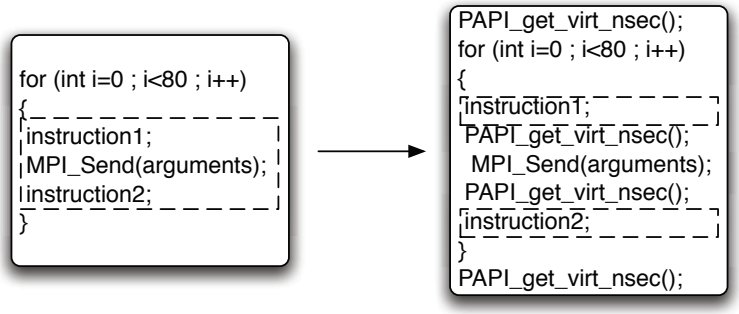


FIG. 6.6: La décomposition d'un bloc d'instructions (celui à gauche) en sous-blocs (voir l'image droite) : un bloc contient des instructions séquentielles, et l'autre contient uniquement l'appel de la communication. L'identification des communication P2P-SAP ou MPI se fait de la même façon, le deux étant gérés par dPerf

PAPI prennent des mesures très précises de la durée d'exécution des lignes d'instructions séquentielles.

Nous décrivons, par la suite, deux techniques de *benchmarking* qui font, d'un coté l'identification des éléments clés du code, et de l'autre coté les modifications au niveau de l'AST pour obtenir un nouveau code source instrumenté :

1. *benchmarking* simple par blocs d'instructions ;
2. *benchmarking* optimisé par blocs d'instructions.

Nous avons proposé ces techniques pour la première fois dans [43]. Nous mentionnons que la première technique peut être appliquée pour toute application acceptée par dPerf. Contrairement à celle-ci simple, la technique optimisée ne peut pas être employée pour certaines applications dont les structures répétitives (FOR, DO WHILE) ont des limites de répétitions qui ne sont pas résolubles statiquement.

6.2.1.3 Benchmarking simple par blocs d'instructions

Pour la technique simple de *benchmarking* par bloc d'instructions, nous nous sommes inspirés de la technique de *micro-benchmarking* de ChronosMix [38]. Dans le contexte

HPC d'aujourd'hui, une analyse basée sur *micro-benchmarking* d'instructions n'est plus adaptée. En conséquence, dPerf ne s'appuie pas sur la technique employée dans ChronosMix, mais il étend celle-ci en regroupant, par bloc d'instructions, toutes les lignes de code -séquentiel- situées entre deux communications. Dans le cas de ChronosMix, l'analyse syntactique était faite sur un AST généré par Sage++, le prédécesseur de ROSE. Après une étude réalisée en collaboration avec J. Vienne[2] sur le potentiel des représentations intermédiaires fournies par ROSE, nous nous sommes décidés de nous baser sur ROSE pour la création des modules C, C++ et Fortran dans dPerf (voir figure 6.1). Dans [42] (testingRoseExecutables.pdf), nous avons cherché des détails sur les différentes analyses syntactiques et les transformations que nous puissions faire avec ROSE.

Cette technique est la plus simple, d'où son nom, et elle implique l'identification des lignes d'instructions clés et l'insertion des appels PAPI sans modifier les lignes du code original. Ce principe que nous avons défini dans dPerf est visible dans la figure 6.6. Nous avons programmé dPerf afin d'identifier le début de l'application analysée pour insérer les appels PAPI indiquant le commencement de l'exécution. D'une manière similaire, dPerf trouve la dernière instruction de l'application et il insère l'appel PAPI qui obtiendra la dernière mesure des compteurs hardware. Le reste de la technique présentée ici consiste à parcourir l'AST en entier pour chercher les communications. Dès qu'une instruction de communication est trouvée, dPerf insère un appel PAPI avant la communication, indiquant ainsi qu'un bloc d'instructions vient de prendre fin. Ensuite, dPerf extrait de l'AST tous les paramètres de communication correspondant à la ligne de communication courante. Notre outil prépare une sortie des plus pertinents paramètres de communication. dPerf modifiera légèrement les noms des événements de communication P2P-SAP ou MPI, car nous avons la liberté de les changer afin que le simulateur réseau puisse les lire. Ensuite, une nouvelle ligne PAPI est insérée à la suite de la communication afin d'indiquer le début d'un nouveau bloc d'instructions. Plus tard dans le flux de travail, ces mesures prises avec PAPI, ainsi que les paramètres des communications, seront écrits dans un fichier de trace. Un extrait d'un tel fichier est présenté dans le listing 6.1.

Listing 6.1: Extrait d'un fichier de dPerf. Le *benchmarking* simple est utilisé. Le temps de calcul (*compute*, en nanosecondes) d'un bloc par le processus p0 est mesuré avec PAPI. Parmi les paramètres de communication nous avons : la source, l'action (Isend, Recv, etc.), la destination, et la taille (en octets).

```
1 p0 compute 5385
2 p0 Isend p0 8429748
3 p0 Recv p0
4 p0 compute 46447205
5
6 #compute en nanosecondes
7 #communication (Isend et Recv) en octets
```

Dans le listing ci-dessus, la ligne *p0 compute 5385* du fichier de trace signifie qu'un bloc d'instructions a pris 5385 nanosecondes pour être exécuté par le processus 0. La

[2] Au moment de l'étude, Jérôme Vienne était membre de l'équipe Mescal du laboratoire LIG (Laboratoire d'Informatique de Grenoble)[114]

ligne *p0 Isend p0 8429748* représente les détails d'une communication, sa durée n'étant pas comprise. Cette ligne signifie que le processus 0 fait une action de type Isend vers lui-même, et la taille des données transférées est de 8429748 octets. Dans le cas d'une application communiquant en P2P-SAP, seul le nom de l'action changera (voir Listing 6.2, c'est-à-dire "Isend" deviendra "Send", et "Recv" s'appellera "Receive", l'approche restant la même.

Listing 6.2: Extrait d'un fichier de trace écrit par dPerf. Le code analysé utilise P2P-SAP pour communiquer.

```
1 p0 compute 8677327
2 p0 Send p4 8200
3 p0 compute 1466
4 p0 Receive p4 8200
5
6 #compute en nanosecondes
7 #communication (Isend et Recv) en octets
```

6.2.1.4 Benchmarking optimisé par blocs d'instructions

Cette technique optimisée est aussi connue en tant que "règle de l'itération de *threshold*" [43]. Elle étend la méthode simple par la recherche et la modifications des nids de boucles. Le principe de cette règle (visible dans la figure 6.7) est de déterminer un nombre minimal d'itérations -ou de ré-exécutions- pour une boucle éligible à cette fonction. Ce nombre minimal d'itérations aura une valeur inférieure au nombre réel tout en calculant l'équivalent de la boucle entière, avec ou sans communications.

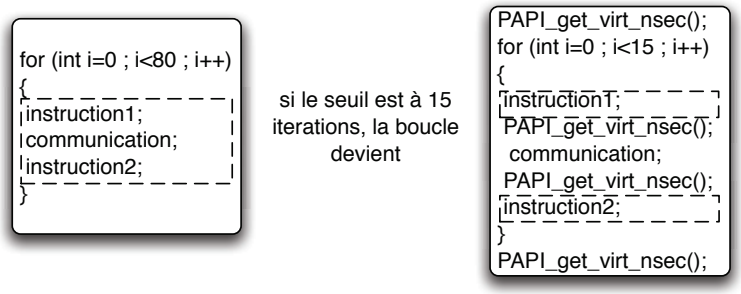


FIG. 6.7: La décomposition d'un bloc d'instructions (celui à gauche) en sous-blocs (voir l'image droite) avec la technique optimisée de *benchmarking* : un bloc contient des instructions séquentielles, et l'autre uniquement l'appel de la communication ; Une modification du nombre d'itérations a lieu. L'identification des communication P2P-SAP ou MPI se fait de la même façon, le deux étant gérés par dPerf

La technique optimisée de *benchmarking* que nous proposons et que nous avons intégrée dans dPerf n'est pas compatible avec certaines applications. Elle peut être appliquée quand les limites des boucles appartiennent aux cas suivants :

1. limites fixées, par exemple :

```
for (int i=0; i<12; i++){
//les instructions
}
```

ou avec la limite supérieure N connue, indépendante :

```
for (int i=0; i<N; i++){
//les instructions
}
```

2. limites particularisées (non-fixées) :
   A. la limite dépend d'autres instructions, et la propagation de sa valeur peut être suivie dans le SDG. Dans ce cas, une analyse du SDG suffira ;
   B. la limite dépend d'autres instructions, mais la nature convergente de l'application analysée ne permet pas de suivre la propagation de la variable avec le SDG. Dans cette situation, dPerf résoudra la convergence du code selon la règle de la convergence (voir section 6.2.1.4).

La décision d'appliquer ou non la technique optimisée est prise selon une étude de la propagation des variables à travers le code source, faite avec le SDG. En regardant dans le SDG les détails d'une boucle, à partir du plus haut niveau, dPerf découvre si des dépendances de données existent entre le corps de la boucle et le reste du code source. Si de telles dépendances sont trouvées, la règle de l'itération de *threshold* ne peut pas être appliquée, et dPerf cherche à gérer les boucles situées aux niveaux inférieurs selon la technique du *threshold* . Si toutes les sous-boucles dans un nid comportent des dépendances des données, la technique simple (voir 6.2.1.3) est appliquée.

Pour une boucle aléatoirement choisie, la différence entre le coût d'une itération et celui des dizaine ou centaines d'itérations est considérable. Nous avons observé qu'après un certain nombre d'itérations de la même boucle, le temps moyen par itération devient constant. Autrement dit, dans l'exécution d'une boucle, il existe un seuil après lequel le comportement de la boucle devient connu, ou prévisible. Nous avons défini notre méthode optimisée de *benchmarking* par blocs selon cette observation de l'itération de seuil.

La règle de l'itération de *threshold* s'est montrée efficace, car elle prend en compte l'effet du chargement des instructions dans la mémoire, ou simplement l'effet de chargement mémoire. Sans cet aspect, la technique n'aurait pas fonctionné.

Considérons th comme étant le seuil (*threshold*) de référence, soit un nombre d'itérations d'une boucle inférieure au nombre maximal. Dans ce cas, t_{th} est le temps -exprimé en nanosecondes- dans l'exécution de la boucle, où le seuil th est atteint. Si la limite d'itérations de la boucle est égale à 1, nous exprimons le temps moyen pour une itération de la boucle par :

$$t_{moyen_{iteration}} = \frac{t_{th}}{th} \tag{6.6}$$

Le $t_{moyen_{iteration}}$ est une valeur moyenne qui prend en compte le temps de chargement des instructions dans la mémoire. Le temps total estimé de la boucle devient :

$$t_{boucle_{estime}} = t_{moyen_{iteration}} \times 1 = \frac{t_{th}}{th} \tag{6.7}$$

Dans le cas d'une boucle avec n iterations, la formule 6.7 devient :

$$t_{boucle_{estime}} = t_{moyen_{iteration}} \times n = \frac{t_{th}}{th} \times n \tag{6.8}$$

En général, pour un nombre d'itérations supérieur à th, le $t_{moyen_{iteration}}$ doit rester dans un pourcentage d'erreur de $\pm\varepsilon$, valeur choisie par l'utilisateur. L'erreur ε peut donc être exprimée en fonction du seuil th et d'un nombre d'itérations x supérieur au seuil :

$$\varepsilon > |t_x - t_{th}|, \forall x > th \tag{6.9}$$

Le seuil idéal du nombre d'itérations

dPerf est capable de calculer le seuil idéal pour une estimation précise du temps total d'une boucle, sans une exécution complète.

Ainsi, dPerf prend en compte le pourcentage d'erreur ε accepté par l'utilisateur, et vérifie si la différence des derniers deux temps moyens par itérations reste dans un intervalle $\Delta \in [-\varepsilon; +\varepsilon]$.

$$th = \{x \,|\, \forall \Delta = t_n - t_{n-1},\, 0 \leq |\Delta| \leq \varepsilon,\, n-1 < n < x\} \tag{6.10}$$

où ε est fixé par l'utilisateur. Pour un code d'exemple, présenté dans le listing 6.3,

nous avons exposé dans la figure 6.8 plusieurs niveaux d'erreur. La courbe montre le temps moyen par itération qui suit une pente décroissante. Si dPerf choisit pour th une valeur proche de 0, donc faible, cette décision impliquera une erreur plus importante. dPerf calcule, donc, le th le plus faible qui garantit que toute itération au delà de ce seuil reste dans l'intervalle d'erreur accepté.

Listing 6.3: Code d'exemple utilisé pour montrer les niveaux d'erreur acceptés pour la technique du seuil.

```
1  #include <cstdio>
2
3  int main ()
4  {
5      int noExec=100, a = 12, b = 11, aux, sum, diff=0;
6      double e,f,g,h=0;
7      for (int i=0;i<noExec;i++){
8          sum = a+b;
9          diff = a-b;
10         e = a*b;
11         f = a/b;
12         g = a%b;
```

```
13              aux=a;
14              a=b*b;
15              b=aux;
16              sum += a+a;
17              diff =a-sum;
18              e=f*g;
19              g=e/sum;
20              a++;
21              b++;
22              sum = a+b+b+b+a;
23          }
24          cout<<sum<<" "<<diff<<" "<<e<<" "<<f<<" "<<g<<" "<<aux<<endl;
25          cout<<a<<" "<<b<<" "<<g<<" "<<endl;
26          return 0;
27      }
```

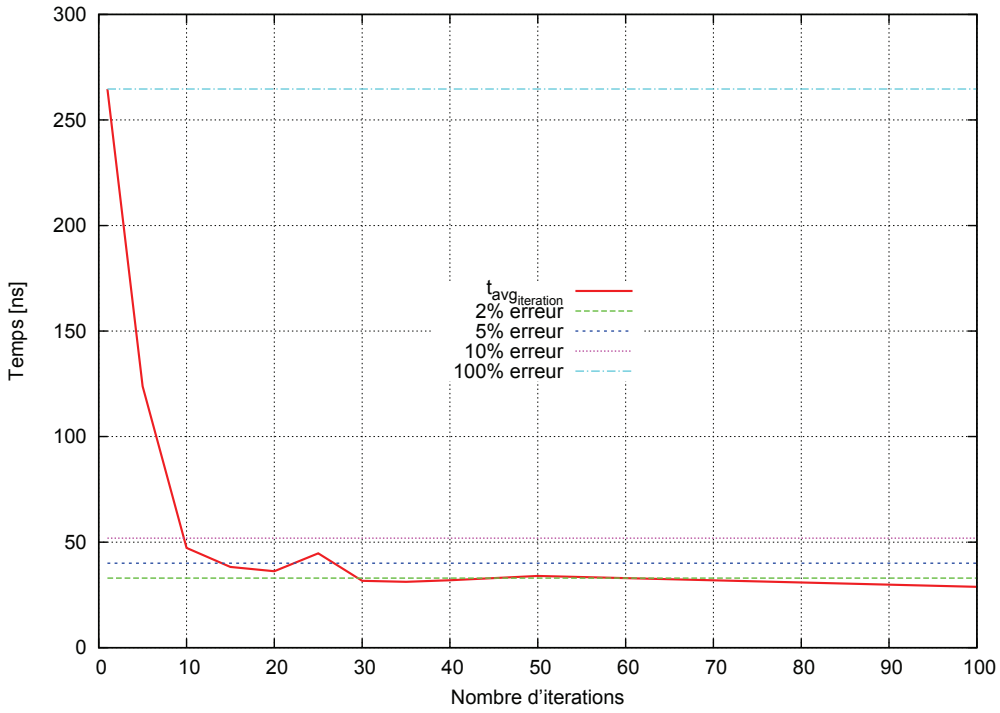


FIG. 6.8: La courbe du temps moyen par itération d'une boucle de test. Les différents niveaux d'erreur que l'utilisateur pourra imposer, peuvent influencer le *threshold* .

En appliquant le *benchmarking* optimisé, le code transformé -sorti de l'analyseur statique- prendra considérablement moins de temps pour être exécuté. Par exemple, pour

le *benchmark* IS de NAS [10], code écrit en C, utilisant MPI pour l'échange de messages, deux prédictions de performances sont faites. L'une d'elles est faite par dPerf utilisant la technique simple de *benchmarking* (voir $t_{prediction}$ dans la figure 6.9), et l'autre par dPerf avec la technique optimisée ($t_{prediction\,threshold}$). Le coût pour accéder à la prédiction -quand dPerf utilise la méthode du seuil- est considérablement inférieur au coût d'une prédiction basée sur le *benchmarking* simple.

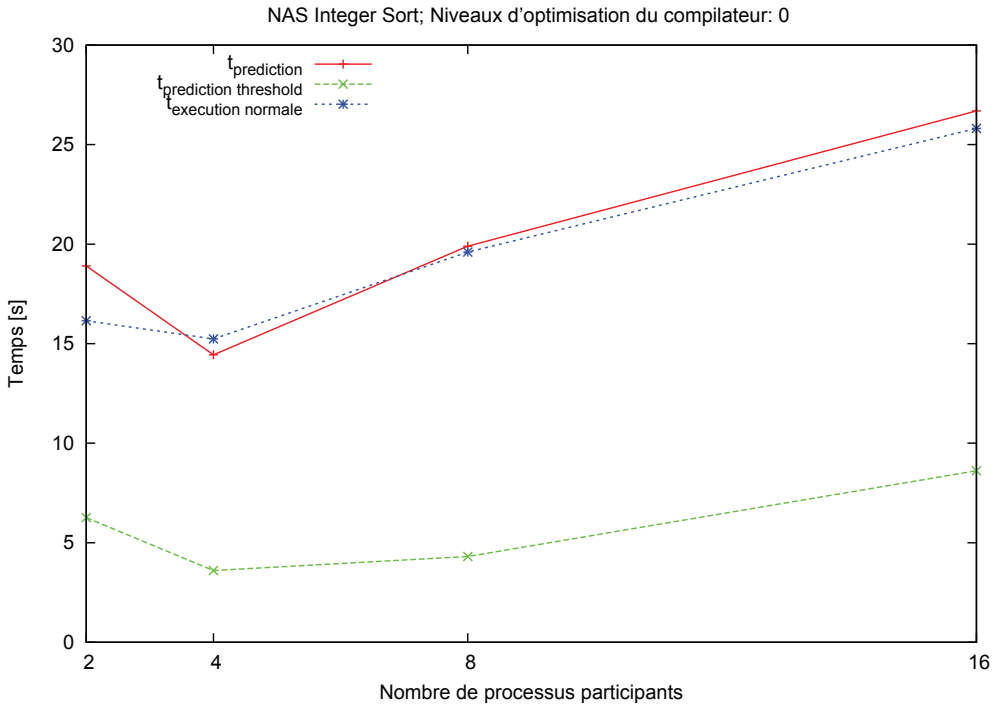


FIG. 6.9: Pour un *benchmark* de NAS, le coût de prédiction avec la technique simple de *benchmarking* est très proche du temps de l'exécution normale. Contrairement à ces temps rapprochés, le coût de prédiction, quand dPerf utilise la technique de *threshold* , est fortement réduit. Pour ces résultats, le compilateur n'utilise pas d'optimisations (le niveau choisi est 0).

L'impact de la modification des limites des boucles sur la prédiction

En collaboration avec J. Vienne[3], J.-M. Vincent et J.-F. Méhaut, du laboratoire LIG de Grenoble, nous avons réalisé une étude (voir [115], le chapitre 4.3) concernant l'effet des modifications du nombre d'itérations des boucles sur les performances d'un code. Cette étude nous a aidé a mieux définir la gestion des nombres d'itérations des boucles, ainsi que la prise en compte des différents niveaux d'optimisation des compilateurs.

[3]Au moment de l'étude, Jérôme Vienne était membre de l'équipe Mescal du laboratoire LIG (Laboratoire d'Informatique de Grenoble)[114]

Nous avons écrit un code expérimental afin d'observer le comportement des boucles du point de vue du chargement des instructions dans la mémoire. Ce code contient une seule structure répétitive, avec une réutilisation des données à la fin du programme. Il est généré et lancé à partir des scripts que nous[4]avons écrits (voir [42], le répertoire "scripts_iterations"). La plateforme de calcul utilise des processeurs Intel Nehalem et Core 2 Quad. La topologie réseau n'est pas prise en compte, le programme de test étant séquentiel. La compilation du code est faite avec la version 4.4.1 de G++ -*GNU C++ compiler*- et avec la version 11.1 d'ICPC-*Intel C++ Compiler*. La figure 6.10 montre les résultats des expériences menées avec le code contenant une seule boucle. Aucun niveau d'optimisation au moment de la compilation n'a pas été utilisée c'est-à-dire que le niveau d'optimisation est -O0. Les lignes horizontales représentent trois niveaux d'erreurs

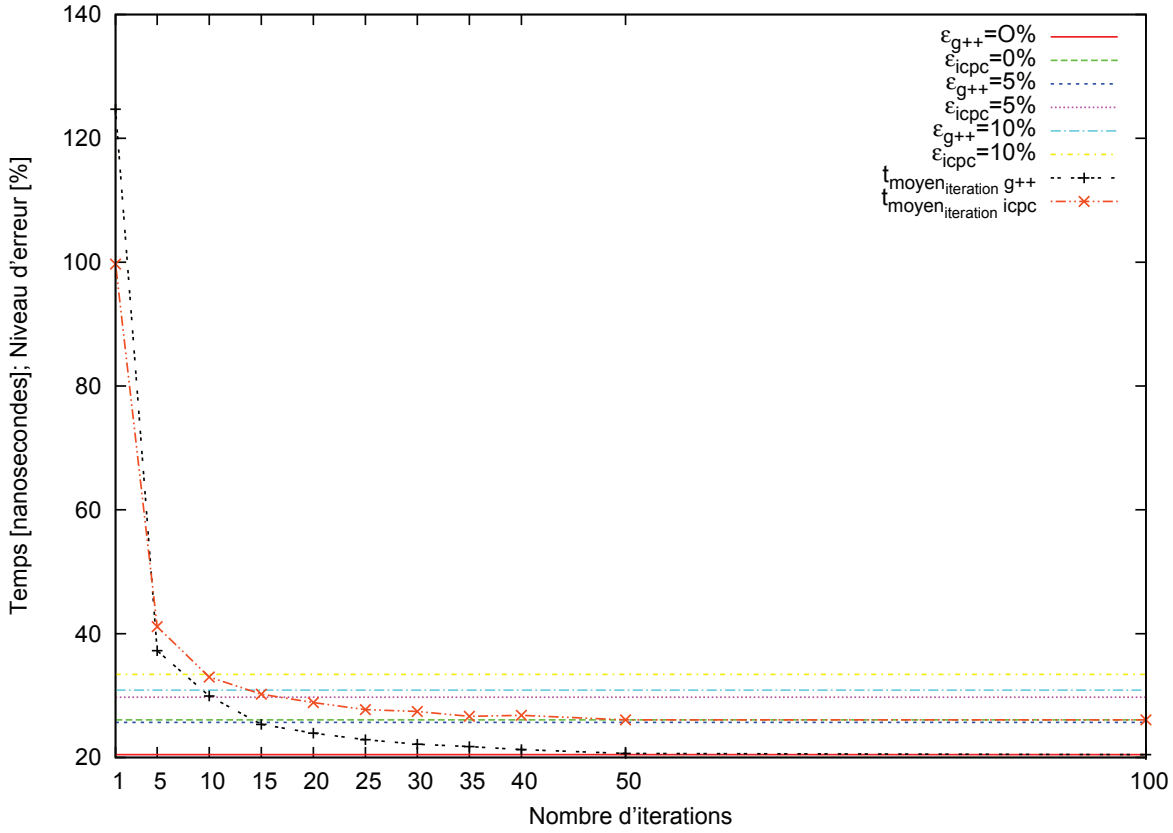


FIG. 6.10: Expériences avec un code d'exemple en vue d'étudier le temps moyen par itération d'une boucle. Code compilé avec G++ et puis ICPC. Les niveaux potentiels d'erreur de 0, 5 et 10% sont indiqués.

potentielles pour chacun des compilateurs. Si, par exemple, l'utilisateur choisit $\varepsilon = 10$ [%] avec g++, le seuil théorique qui nous permettra d'appliquer la technique optimisée de *benchmarking* par bloc, sera autour de 10 itérations, soit :

$$th = 10 \text{ itérations et } t_{th} = 30 \text{ ns, pour } \varepsilon = 10\ \%.$$

[4] Ce travail a été effectué en collaboration avec l'équipe Mescal du LIG

Cette expérience confirme qu'une modification du nombre d'itérations d'une boucle, dont les limites d'itérations sont constantes ou identifiables, est acceptable si un minimum de th itérations est effectué. Ce nombre th assure la prise en compte de l'effet de chargement des instructions dans la mémoire. Comme le montre les courbes $t_{moyen_{iteration}}$, au delà de th, l'erreur de prédiction du temps d'exécution de la boucle reste en dessous le ε choisi par l'utilisateur.

Nous avons proposé et implémenté la technique optimisée de *benchmarking* par bloc d'instructions dans dPerf, tout en tenant compte des expérimentations faites sur un programme simple, avec une seule structure répétitive. Le potentiel de notre technique était validé dans [43] -pour une application *HPC* écrite dans C/MPI- et dans [44] -pour une application *HPC* pair-à-pair décentralisée écrite dans C/P2P-SAP.

La règle de la convergence Une extension du *benchmarking* optimisé par bloc d'instructions est la règle de la convergence. Ceci s'applique dans le cas où les limites des nids de boucles d'un plus haut niveau -dans l'arborescence syntaxique- dépendent d'un point de convergence. Ce sont, en général, les structures DO WHILE qui appartiennent à cette catégorie. Cette règle ne fonctionne pas pour toutes les applications convergentes.

Nous avons défini la règle de la convergence dans dPerf comme étant une estimation du nombre d'itérations nécessaires à la convergence du calcul effectué à l'intérieur d'un boucle. Nous imposons un minimum de 4 itérations à effectuer. Ces 4 itérations permettent à dPerf de :

- calculer le $t_{moyen_{iterations}}$;
- faire une estimation sur le nombre d'itérations encore nécessaires au delà des 4 premières.

Nous avons effectué plusieurs essais avant de nous prononcer sur le comportement des boucles convergentes. Dans nos essais, les boucles suivent une courbe descendante. Plus le nombre d'itérations augmente, moins de temps est requis pour le calcul d'une itération. Le nombre d'itérations que dPerf estime ne se fait pas d'une façon statique, mais semi-statique. La règle de la convergence est introduite dans le code analysé -l'insertion se produisant dans la phase de la réduction du *slowdown* .

L'avantage de la technique optimisée de *benchmarking* par rapport à celle simple de *benchmarking* par blocs consiste dans la réduction en temps réel du nombre d'itérations d'une boucle convergente. Cette technique est appliquée d'une façon "temps réel" car elle dépend de résultats des calculs effectués entre chaque deux itérations.

6.2.1.5 Rendre le code transformé

Une des phase de la réduction du *slowdown* est de rendre un nouveau code source (voir "Code transformé" dans la figure 6.4) contenant toutes les modifications faites pendant la traversée de l'AST et du SDG. Le processus de conversion du nouveau AST - car toutes les modifications se font au niveau des nœuds de l'AST- dans un code source correct du point de vue syntaxique, est appelé *unparsing*. Le code obtenu est écrit dans le même langage que celui du code d'entrée.

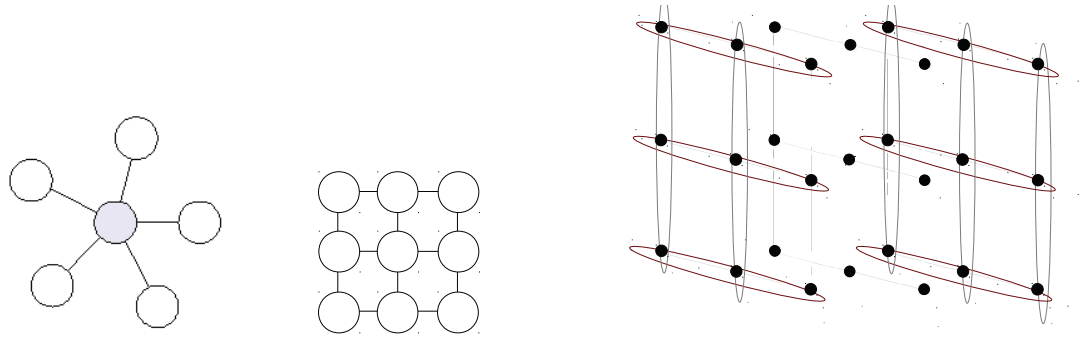
6.2.2 Extrapolation de performances

6.2.2.1 L'identification de la topologie logique du code d'entrée

Les caractéristiques de communication d'une application peuvent être identifiées pendant la réduction du *slowdown* . L'identification est faite au niveau de l'AST, tout comme l'identification des instructions de communication. dPerf peut identifier avec succès trois types de topologie logiques, ou de schéma de communications utilisés par l'application :

- le type maître-travailleur (*master-worker*) ;
- le type maille en 2 dimensions (*2d mesh*) ;
- le type tore en 3 dimensions (*3d torus*) ;

Les spécifications de chacun des types ci-dessus sont définies dans les modules de dPerf qui gèrent le C, le C++ et le Fortran. Ces modules sont écrits en C++ mais toute instruction à insérer dans l'AST correspond au langage analysé. Les trois types de communication sont présentés dans la figure 6.11.



(a) Maître-travailleur (étoile) (b) Maille 2d (c) Tore 3d

FIG. 6.11: Trois topologies logiques sont identifiables. La topologie maître-travailleur (le processus maître en gris) dans 6.11a ; une communication de type maille en 2 dimensions dans 6.11b ; le tore en 3 dimensions dans 6.11c.

dPerf réutilise l'AST crée au début de la réduction du *slowdown* . L'AST est parcouru à nouveau en cherchant les instructions de communication. Le but est, cette fois-ci, d'analyser les paramètres de communication pour valider la topologie logique de l'application. Si le code ne suit aucun des types présentés ci-dessus, la décision est prise et l'analyse statique prend fin. Au cas contraire, si dPerf arrive à la fin du parcours de l'AST sans générer d'exceptions, il affiche le résultat de l'identification et rend le code transformé.

L'identification de la topologie de communication utilisée dans le code analysé a une double utilité :

– proposer aux scientifiques des idées pour configurer ou choisir l'architecture de calcul adaptée au type de calcul envisagé par le développeur de l'application. Cette fonctionnalité a été validée dans [39] en tant que composante du système ADAPT pour l'identification automatique du paradigme de l'application. Cette identification aide à choisir d'une façon automatique l'architecture *cloud* qui fournira le résultat de calcul dans le plus court délai.

– prendre une décision par rapport au mécanisme de passage à l'échelle (*scaling-up*) de la prédiction de performances de l'application concernée que nous ferons dans la partie 6.2.2.2.2.

La contrainte à respecter par une application de type maître-travailleur est :

$$\forall k \in [1; ntaches]\ ,\ k \in\ \mathbb{N}\ ,\ \nexists\, v_j | v_i v_j \in E(G_i)\ avec\ j \neq 0 \tag{6.11}$$

où $G_k(V, E)$ est un nombre de k graphes orientés, $V(G_k)$ représente les tâches, $E(G_k)$ sont les communications d'un programme P, et $G_k(V, E)$ représente le schéma de communication de la tâche numéro k. Autrement dit, une application maître-travailleur doit communiquer dans une topologie logique d'étoile, le nœud maître se situant au centre de l'architecture de calcul. Dans le cas d'une application maître-travailleur, le message pour confirmer l'identification réussite est montré dans le tableau 6.2.

TAB. 6.2: Le résultat de l'identification faite par dPerf dans la phase statique. Ce résultat était obtenu pour un programme simple en MPI. [1] D'autres cas non gérés par dPerf ou non identifiables ; [(2)]Temps pour obtenir la décision ;

Code d'entrée	Maître-travailleur	Maille 2d	Tore 3d	Autre[1]	$t_{decision}$ [(2)] [ms]
code1.c	Oui	Non	Non	Non	0.243

Les application qui ne suivent pas le paradigme maître-travailleur sont en suite vérifiées si ils utilisent la topologie maille en 2d. Les contraintes à respecter par une application de ce type sont : $\forall k \in [1; n\,taches] \cap\ \mathbb{N}\ ,$

$$V_2(G_k) = \{v_j | (v_j v_{j+1}, v_j v_{j+c}), (v_j v_{j-1}, v_j v_{j+c}), (v_j v_{j-c}, v_j v_{j+1}), (v_j v_{j-1}, v_j v_{j-c}) \in E(G_j)\} \tag{6.12}$$

où $j \in \{1, c, (c \times (l-1)+1), (c \times l)\}$,

$$\begin{aligned} V_3(G_k) = \{v_j | &(v_j v_{j-1}, v_j v_{j+c}, v_j v_{j+1}), (v_j v_{j-1}, v_j v_{j-c}, v_j v_{j+1}), \\ &(v_j v_{j+1}, v_j v_{j-c}, v_j v_{j+c}), (v_j v_{j-1}, v_j v_{j-c}, v_j v_{j+c}) \in E(G_j)\} \end{aligned} \tag{6.13}$$

où $j \in (1; c) \cup ((c \times (l-1)+1); c \times l) \cap\ \mathbb{N}\ \cup \{p \times c + 1\} \cup \{(p+1) \times c\}, \forall l \geq 2$, et $p \in [1; l-2] \cap\ \mathbb{N}\ ;$

$$V_4(G_k) = \{v_j | (v_j v_{j-1}, v_j v_{j+1}, v_j v_{j-c}, v_j v_{j+c}) \in E(G_j)\} \tag{6.14}$$

où $\forall j \notin [1; c] \cup [(c \times (l-1)+1); c \times l] \cap\ \mathbb{N}\ \cup \{p \times c + 1\} \cup \{(p+1) \times c\}$
Dans les equations ci-dessus,

- $G_k(V, E)$ est un nombre de k graphes ;
- $V(G_k)$ représente les tâches. $V_i(G_k)$, où $i \in \{2, 3, 4\}$, contient tous les nœuds de la maille 2d qui communiquent avec un nombre i de nœuds voisins ;

- $E(G_k)$ sont les communications d'un programme P ;
- $G_k(V, E)$ représente le schéma de communication de la tâche numéro k ;
- c et l sont respectivement le nombre de colonnes et le nombre de lignes de la maille 2d.

La validation de ces contraintes suit le principe suivant. dPerf analyse le code source pour l'identification des appels de communication, puis notre outil extrait le rang du processus destinataire parmi les paramètres de la communication. Dans le cas des mailles 2d, cette variable se trouve dans une chaine des entiers. Habituellement, la valeur d'une telle variable n'est accessible que pendant l'exécution. À cause de ceci, dPerf identifie le destinataire de la communication avec une méthode inverse à la propagation des variables :

1. Après avoir identifié la fonction de communication, dPerf accède le SDG de l'application (voir la section 5.4.2.4).
2. Dans le SDG, dPerf remonte sur le chemin de propagation de la variable de destination jusqu'à sa définition.
3. Sachant qu'en général les partenaires de communications sont calculés par rapport au nombre de nœuds, aux colonnes c et aux lignes l, dPerf peut calculer si un voisin existe ou si sa valeur est "null".
4. Si un voisin (destination) existe, dPerf incrément le nombre de connexions d'un nœud (voir les formules 6.12, 6.13, 6.14).
5. Toutes valeurs identifiées ou calculées sont enregistrées dans une liste chaînée contenant pour pour chaque rang un nombre des partenaires de communication.

Pendant le processus d'identification de la topologie de communication, dPerf suppose que la numérotation des nœuds, soit leur rang, commence de gauche à droite et de haut en bas, à partir de 1 e jusqu'à la valeur $c \times l$ (c est le nombre de colonnes et l est le nombre de lignes). Toute application analysée dont les valeurs de $V_i(G_k)$ ne sont pas égales à celles montrées ci-dessous, n'utilise pas la topologie logique de la maille en 2d.

$$V_i(G_k) = \begin{cases} \phi, & \forall i \notin \{2,3,4\} \\ 4\,noeuds, & i = 2 \\ (c + l - 4) \times 2\,noeuds, & i = 3 \\ (c-2) \times (l-2)\,noeuds, & i = 4 \end{cases} \tag{6.15}$$

où i est le nombre de nœuds voisins de chacun des nœuds dans $V_i(G_k)$, c et l sont respectivement le nombre de colonnes et de lignes de la maille 2d.

Si l'application ne respecte pas les contraintes d'une maille en deux dimensions, dPerf vérifie la topologie tore en 3d. Le principe d'identification est identique à celui d'une topologie maille en 2d avec d'autres contraintes au niveau des voisins de communication :

$\forall k \in [1; n\,taches] \cap \mathbb{N}$,

$$V_5(G_k) = \{v_j| \begin{array}{c}(v_jv_{j-c}, v_jv_{j+1}, v_jv_{j+c}, v_jv_{j-1}, v_jv_{j+c\times l}),\\ (v_jv_{j-c}, v_jv_{j+1}, v_jv_{j+c}, v_jv_{j-1}, v_jv_{j-c\times l}),\\ (v_jv_{j-1}, v_jv_{j+c\times l}, v_jv_{j-c\times l}, v_jv_{j+c}, v_jv_{j-c}),\\ (v_jv_{j+1}, v_jv_{j+c\times l}, v_jv_{j-c\times l}, v_jv_{j+c}, v_jv_{j-c}),\\ (v_jv_{j-1}, v_jv_{j+c\times l}, v_jv_{j+1}, v_jv_{j-c\times l}, v_jv_{j+c}),\\ (v_jv_{j-1}, v_jv_{j+c\times l}, v_jv_{j+1}, v_jv_{j-c\times l}, v_jv_{j+c})\end{array} \in E(G_j)\} \tag{6.1}$$

où $j \in [c+1; c(l+1)] - \{(q+1)c\} - \{q\times c+1\} \cap \mathbb{N}$, $\forall q \in [1; l-2], et \forall l > 2, c > 2, p > 2$;

$$V_6(G_k) = \{v_j|(v_jv_{j+1}, v_jv_{j-1}, v_jv_{j+c}, v_jv_{j-c}, v_jv_{j+c\times l}, v_jv_{j-c\times l}) \in E(G_j)\} \tag{6.17}$$

où $j \in [1; c\times l\times p] - (q\times c+1; (q+1)c)$, $q \in [1; l-2] \cup [1+a\times c\times l; l-2+a\times c\times l]$, et $a \in [1; p] \cap \mathbb{N}$.

Dans les equations ci-dessus,

- $G_k(V, E)$ est un nombre de k graphes ;
- $V(G_k)$ représente les tâches. $V_i(G_k)$, où $i \in \{5, 6\}$, contient tous les nœuds de la maille 2d qui communiquent avec un nombre i de nœuds voisins ;
- $E(G_k)$ sont les communications d'un programme P ;
- $G_k(V, E)$ représente le schéma de communication de la tâche numéro k ;
- c, l, et p sont respectivement le nombre de colonnes, le nombre de lignes, et le nombre de plans du tore en 3d.

Toute application analysée, dont les valeurs de $V_i(G_k)$ ne sont pas égales à celles montrées ci-dessous, n'utilise pas la topologie logique du tore en 3d. Dans le cas d'un tore 3d avec les nœuds des extrémités inter-connectés verticalement et horizontalement, voici le $V_i(G_k)$

$$V_i(G_k) = \begin{cases}\phi, & \forall i \notin \{5, 6\}\\ 2[(c-2)(l+p-4) + (l-2)(p-2)]\,noeuds, & i = 5\\ 4(c+l+p-4) + (c-2)(l-2)(p-2)\,noeuds, & i = 6\end{cases} \tag{6.18}$$

où i est le nombre de nœuds voisins de chacun des nœuds dans $V_i(G_k)$. c, l et p sont respectivement le nombre de colonnes, de lignes, et de plans du tore 3d.

L'identification des types maille 2d et tore 3d nécessite (i) de connaître la ligne de commande qui sera utilisée pour lancer l'application, (ii) et le réglage des macros pour uniformiser le nom de la variable du rang et celui du vecteur des voisins. La ligne de commande permet a dPerf de connaître le nombre denœuds Nb_noeuds qui participent au calcul.

6.2.2.2 Simulation avec Simgrid MSG et les résultats de prédiction

Le fichiers de traces sont donnés en entrée du module MSG de Simgrid. Celui-ci est responsable de la simulation des échanges de données entre processus (participants), ainsi

que de l'addition des temps de calcul aux temps de communication. Tous les types d'événements doivent être définis dans le module MSG. Les événements de communication sont définis comme des échanges entre deux processus liés par un réseau défini en externe. Du point de vue de la simulation, le calcul est défini comme un temps d'attente égal à la valeur d'une ligne "compute" du fichier de trace.

La simulation est un des facteurs qui contribuent à la prédiction de performances. La simulation est basée sur le contenu des fichiers de trace, mais aussi sur la description de la plateforme *HPC* . La description d'une plateforme contient les détails de l'architecture de calcul, c'est-à-dire (i) les nœuds de calcul, et (ii) la topologie réseau. Avec ces deux éléments, nous pouvons définir tout type de système homogène ou hétérogène. Le résultat de la phase de simulation est le t_{predit}, la prédiction faite par dPerf. La précision de dPerf est donné par :

1. les techniques simples ou optimisées de *benchmarking* par blocs ;
2. l'utilisation des fonctions PAPI permettant de prendre des mesures à l'aide des compteurs matériels de performance ;
3. la polyvalence du module MSG de Simgrid, que nous avons adapté aux besoins de dPerf.

Le fait d'appuyer la phase finale du processus de prédiction sur MSG permet à dPerf de fournir deux types de résultats de prédiction (i) l'une quand l'utilisateur souhaite des prédiction pour des différentes topologies réseaux, mais avec un nombre fixe de nœuds, (ii) et l'autre pour un système de calcul où le nombre de nœuds est extrapolé.

6.2.2.2.1 Prédictions sur des multiples topologies réseau Le premier résultat fourni par dPerf est une prédiction pour un même nombre des nœuds que celui de l'architecture cible que sur celle utilisée pour exécuter dPerf, soit une prédiction 1 :1 des nœuds. Cette prédiction permet aux développeurs et scientifiques d'expérimenter le comportement des communications d'une application *HPC* .

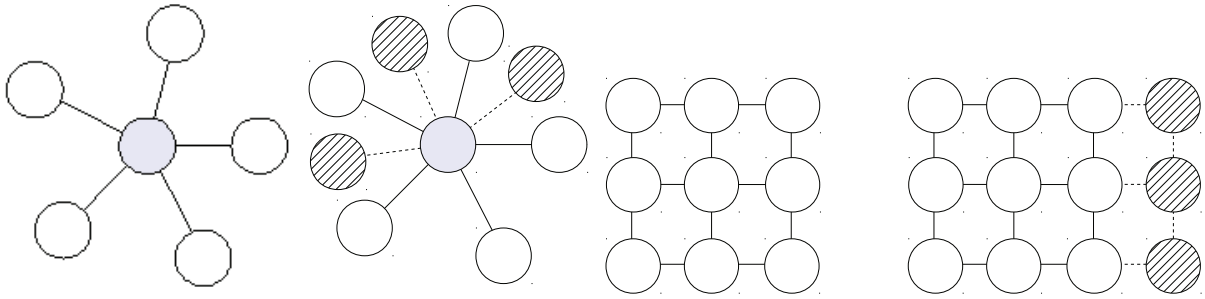
6.2.2.2.2 L'extrapolation du nombre de nœuds Grâce au *benchmarking* par blocs et au module MSG adapté pour dPerf, le deuxième type de résultat est une prédiction à une échelle différente de celle utilisée pour l'exécution de dPerf, soit 1 :N des nœuds. Ce cas permet de voir le comportement d'une application quand une augmentation de l'échelle est envisage pour l'architecture de calcul.

Soit N le nombre de nœuds, l'extrapolation de ce nombre de nœuds est directement liée à la topologie de communication utilisée par l'application. Pour cela, la prédiction de dPerf pour un nombre N'de nœuds, où $N < N'$, est précédée par l'identification de la topologie logique, soit $dPerf_{toplog}$. Les scénarios possible sont les suivants :

1. Si $dPerf_{toplog}$ est de type maître-travailleur (voir la figure 6.12a et 6.12b). L'extrapolation de N pour cette topologie logique est faite selon la formule

$$N' = \begin{cases} N + i & si\, N = j,\, \forall i > 0,\, j > 0,\, i, j \in \mathbb{N} \\ 2^{k+p} & si\, N = 2^k,\, \forall k > 0,\, p > 0,\, k, p \in \mathbb{N} \\ N + 2 \times k & si\, N = 2 \times p,\, \forall k > 0,\, p > 0\, k, p \in \mathbb{N} \end{cases} \tag{6.19}$$

en fonction des contraintes imposées par l'utilisateur. Si l'application peut être lancée pour $\forall N$, l'extrapolation suit la formule $N' = N + i$, mais si l'application nécessite un $N = 2^k$, alors $N' = 2^{k+p}$.



(a) (b) (c) (d)

FIG. 6.12: Extrapolation de N à N' pour une topologie maître-travailleur (a,b), et maille en 2d (c,d). (a,c) Les topologies identifiées par dPerf. (b,d) La topologie $dPerf_{toplog}$ d'un nombre augmenté de nœuds. Extrapolation faite par dPerf.

2. Si $dPerf_{toplog}$ est de type maille en 2d (voir la figure 6.12c et 6.12d), l'extrapolation nécessite une augmentation de N selon la formule 6.20 ou 6.21.

$$N' = N + c \times i, \forall i > 0, i \in \mathbb{N} \tag{6.20}$$

pour ajouter un nombre de i nœuds à la maille prédite, où c est le nombre de colonnes mais aussi le nombre de nœuds sur chaque ligne de la maille.

$$N' = N + l \times i, \forall i > 0, i \in \mathbb{N} \tag{6.21}$$

pour ajouter un nombre de i nœuds à la maille prédite, où l est le nombre de lignes mais aussi le nombre de nœuds sur chaque colonne de la maille.

3. $dPerf_{toplog}$ est de type tore en 3d. Ce cas est similaire au celui de la maille en 2d avec la mention que l'extrapolation se fait selon la formule :

$$\forall i > 0, i \in \mathbb{N}, N' = \begin{cases} N + c \times i, pour\ ajouter\ i\ lignes \\ N + l \times i, pour\ ajouter\ i\ colonnes \\ N + p \times i, pour\ ajouter\ i\ plans \end{cases} \tag{6.22}$$

où c, l, et p sont respectivement le nombre de colonnes, de lignes et de plans du tore en 3d.

6.2.2.2.3 La limitation du module MSG Pendant le développement et la préparation des études de prédiction de dPerf, une contrainte était imposée au niveau du module MSG de Simgrid. Celle-ci se réfère à la taille des messages échangés par les processus pendant la simulation basée sur les fichiers de traces.

Pour une communication d'un taille $w_{message}$, la limitation à la précision imposée par MSG est :

$$w_{message} \geq 100KB \tag{6.23}$$

quel que soit le sens de la communication. Cette restriction contribuera, ou plus précisément elle augmentera, le pourcentage d'erreur des résultats expérimentaux présentés dans le chapitre 7.

Clauss et al. proposent dans [41] une amélioration du module MSG. Ceci implique un nouveau mécanisme de simulation basée sur des fichiers de traces qui enlève la contrainte 6.23. Pour les prédictions de dPerf, l'amélioration proposée par Clauss et al. pourra contribuer à une précision augmentée de notre outil. Les résultats des études de précision de dPerf présentés dans ce manuscrit utilise l'ancienne version du module MSG, et donc la contrainte 6.23 s'applique.

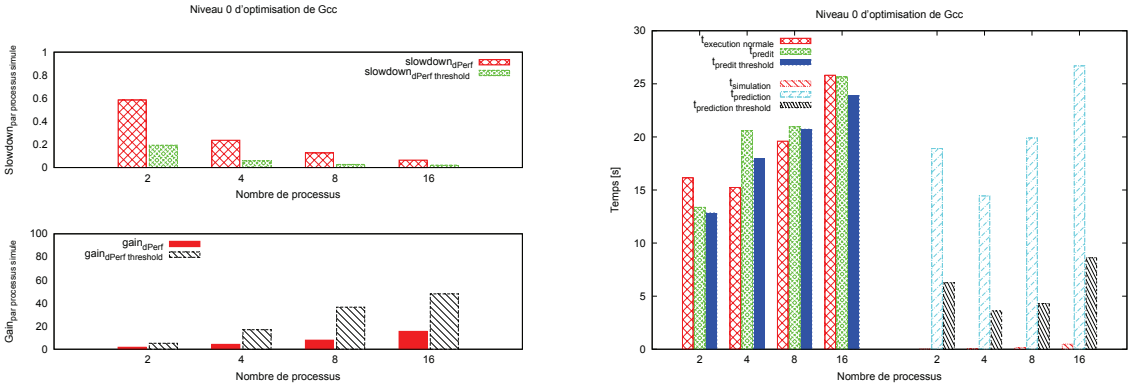
6.2.3 Le coût pour accéder à une prédiction faite avec dPerf

La prédiction de performances réalisée par dPerf a un faible coût, et ce, grâce à de nombreuses techniques. Dans un premier temps, la réduction du *slowdown* (contenant les deux techniques de *benchmarking*) réduit le *slowdown* (par processeur simulé) de dPerf à approximativement 1.1 -pour le *benchmarking* simple- et pouvant aller jusqu'à 0.2 -pour le *benchmarking* optimisé avec la modification des limites des boucles. Ces valeurs du *slowdown* par processeur simulé sont assez réduites pour notre outil, car il fait partie des outils hybrides où le *slowdown* par processus simulé de la majorité des outils reste assez important (voir le tableau 3.1). Nous pouvons ainsi parler d'un gain quand il s'agit de dPerf (voir figure 6.13a). Dans la partie III de ce document, nous présentons plus en détail des valeurs du *slowdown* et du gain obtenus pendant les expérimentations. Dans un deuxième temps, la simulation basée sur des fichiers de trace prend un temps très faible par rapport au temps d'exécution d'une application et même par rapport au temps de prédiction de dPerf (voir figure 6.13b).

6.3 L'implémentation

Nous présentons dans cette partie la façon dont nous avons implémenté les méthodes présentées dans la section 6.2, qui utilisent les fonctions des outils et des bibliothèques définies dans la section 5. Le code source de dPerf représentant le meilleur exemple de l'implémentation, dans cette section nous présentons uniquement les aspects les plus pertinents de dPerf.

dPerf est développé en C++, et il utilise des fonctions de ROSE, elles aussi écrites en C++. dPerf instrumente le code avec PAPI, dans la phase statique, pour générer des fichiers de trace compatibles avec le module MSG dans Simgrid. Les liaisons entre dPerf et PAPI, ou dPerf et Simgrid, sont les modules C, C++ et Fortran qui sont responsables avec l'insertion des instructions clés dans le code analysé, imposant ainsi le format des traces destinées à l'exécution dans Simgrid.



(a) Le *slowdown* et le gain par processus simulé.

(b) Le temps de simulation (avec Simgrid MSG) par rapport à $t_{execution\,normale}$, t_{predit} (simple et optimisé), et $t_{prediction}$ (simple et optimisé).

FIG. 6.13: Exemple de *slowdown* , gain et temps de simulation ($t_{simulation}$) obtenus avec dPerf pour l'application NAS IS sur un nombre de 2, 4, 8 et 16 processus, et pour un problème de classe A.

6.3.1 Réduction du *slowdown*

dPerf commence la phase de la réduction du *slowdown* avec le code source à analyser. L'extension du fichier d'entrée doit représenter parfaitement le langage utilisé pour son développement. En conséquence, l'extension ".c" pour un fichier C++ produira une exception qui arrêtera la réduction du *slowdown* . Une même situation intervient quand l'extension ".cpp" est utilisée pour un fichier C. L'utilisation des extensions de fichiers appropriés est imposée non pas par dPerf, ni par ROSE, mais par le compilateur EDG -intégré dans ROSE frontend- qui est chargé de créer l'AST d'un code source. Pour obtenir cet AST, dPerf appelle une première fonction de ROSE pour transformer le code d'entrée en arbre syntactique. et commence la traversée de l'AST. Des variables globales sont définies pour indiquer les caractéristiques de l'application et le type de l'analyse à faire. À partir de ces instructions, le module de dPerf correspondant au langage à analyser sera chargé de la suite. Prenons, par exemple, le cas d'un code d'entrée en C qui communique selon une topologie logique décentralisée P2Pdc[5]. Les variables globales actives seront *BLOCK_C_MPI*, *SIMGRID_ENABLED*, *P2PDC_ENABLED*, *NS_ENABLED*, ou *MAIN*. dPerf utilise la bibliothèque PAPI et d'autres structures de données pour l'enregistrement des traces de calcul et de communication. À cette effet, dPerf introduit dans l'entête du code analysé toutes les instructions nécessaires à la compilation avec succès du code transformé. Ces instructions varient en fonction du langage analysé. dPerf introduit des bibliothèques telles que *papi.h*, *iostream*, *fstream*, *sstream*, *list*, *math.h*, *stdlib.h*, ou *string.h*. Celles-si sont ajoutés à l'AST par une fonction que nous avons défini dans dPerf.

[5]P2Pdc est un environnement de calcul P2P décentralisé de haute performance développé au LAAS-CNRS à Toulouse en France

Notre outil a un corps de fonctions commun pour tous les modules. Leur définition correspondra à chacun des langages analysés. Une fonction existe dans dPerf pour permettre d'analyser l'un des parent d'un nœuds de l'AST. Tous les événements de communication que dPerf cherchera dans un AST sont définis dans une liste. Si dPerf doit "apprendre" à identifier et gérer un nouveau formalisme de communication, nous allons procéder à la mise à jour de cette liste. Si l'un des nœuds de l'AST est important et il doit être retenu, dPerf copie son adresse et passe à la transformation. Une vérification est faite pour s'assurer qu'un nœud de l'AST représentant une communication n'est pas traité plusieurs fois.

Tous les nœuds de l'AST sont du type *SgNode*, mais en fonction du rôle dans le code, un nœud peut élargir le *SgNode* pour avoir un autre type, par exemple *SgIfStmt* (pour une structure de type IF dérivé de *SgNode*).

Le module C est décrit dans la classe "Block C". Ici, plusieurs fonctions sont définies en vue de la réalisation des transformations dans l'AST. Ces transformations suivent :

- l'identification des communications qui appartiennent aux structures conditionnelles (IF) ou répétitives (FOR, DO WHILE), où la communication n'appartient pas à un bloc d'instructions ; Conforme à la figure 6.6, dPerf va "isoler" chaque communication en introduisant des appels PAPI avant et après la communication, or ceci n'est pas acceptable si la communication n'est pas comprise dans un bloc d'instructions délimité par "" et "". Ainsi, dPerf va transformer ce type d'instruction IF dans un bloc d'instructions avec une seule ligne de code ;
- l'insertion des fonctions PAPI indiquant le début d'une prise de mesure. Un tel point est indiqué par la variable de dPerf nommée "startPAPI". Celle-ci ne peut pas apparaître qu'au début du code et/ou après une communication. Une fonction que nous avons défini est chargée avec cette action ;
- l'insertion des fonctions PAPI indiquant la fin d'une prise de mesure et l'affichage du temps de computation en tant que trace. Un tel point est indiqué par la variable de dPerf nommée "stopPAPI". Celle-ci ne peut apparaître qu'à la fin du code et/ou avant une communication ;
- l'instrumentation des fonctions de communication de la bibliothèque MPI et du protocole P2P-SAP. Ce dernier est un protocole de communication disponible uniquement dans P2Pdc, l'environnement P2P décentralisé développé au LAAS-CNRS (Toulouse). dPerf identifie ces fonctions et extrait les arguments pertinents. Avec ces informations, dPerf met en place l'écriture d'une trace contenant les acteurs de chacune des communications ainsi que les paramètres pertinents pour la phase de la simulation ;
- la recherche d'un nœud particulier. Celle-ci est une action très fréquente dans la réduction du *slowdown* de dPerf car elle permet d'identifier des informations dans un AST en vue de l'identification d'un communication, de l'extraction des arguments de la communication, ou de l'identification du nœud avant lequel dPerf insérera des instructions d'instrumentation.

Les modules C++ et Fortran ont le même rôle que celui pour "C" sauf que les méthodes sont implémentées pour correspondre à la syntaxe du C et respectivement du For-

tran. De nouveaux modules pour plus de langages peuvent être ajoutés. Les contraintes pour élargir la plage des langages supportés sont :

- si la réduction du *slowdown* utilise les fonctions de ROSE, le nouveau langage doit être géré par ROSE. Sinon, un nouveau analyseur statique peut être développé avec des fonctions qui couvrent un segment différent de langages.
- un module sera défini pour ce nouveau langage, en définissant dans celui-ci des fonctions de transformation existantes aussi dans les modules C, C++ ou Fortran.
- le développeur doit s'assurer que les transformations faites par le module correspondant au nouveau langage sont 100 % correctes du point de vue syntactique, ou l'exécution du code transformé échouera.
- garantir la gestion des événements de communication du code analysé par le module MSG du simulateur Simgrid.

dPerf analyse et transforme un code d'entrée d'une manière qui garantit que le résultat de l'exécution de ce code seront des fichiers de traces qui suivront le modèle accepté par le simulateur. Ce modèle est aussi décrit dans le listing 6.1. Il y a deux catégories d'événements dans les fichiers de traces :

1. le calcul, représenté par le terme en anglais *compute* ;
2. la communication, représentée par le nom de celle-ci. Par example, une action où un message est envoyé pourra apparaitre comme *P2P_Send*, *Isend*, *Bcast*, etc. Une action de réception des messages pourra être du type *P2P_Receive*, *Recv*, etc.

L'action correspond obligatoirement à l'une des actions reconnues par Simgrid MSG. Le tableau 6.3 présent une liste d'événements de communication reconnue par dPerf et Simgrid MSG, par rapport à la bibliothèque MPI et au protocole P2P-SAP.

L'identification de la topologie logique utilisée dans une application analysée est faite en analysant les arguments des communications. Plus précisément, dPerf cherche les émetteurs et les récepteurs, et en fonction de ceux-ci il valide ou non l'un des trois paradigmes reconnus :

1. maître-travailleur
2. maille 2d
3. tore 3d

Le paradigme maître-travailleur impose une communication entre le maître et quel que soit le récepteur, et en sens inverse d'un émetteur vers le maître uniquement. Si ce n'est pas le cas, l'application n'est pas du type maître-travailleur. Ce type d'analyse est fait assez souvent en utilisant l'AST seulement. Pour les paradigmes maille et tore il est souvent nécessaire de consulter le SDG -pour les dépendances et la propagation des variables- pour identifier les rapports entre tous les processus participants. Ces rapports sont représentés par des listes chainées et la décision est prise une fois que toutes fonctions de communication soient analysées. L'implémentation de l'identificateur du paradigme implique au moins une traversée de l'AST ou SDG, et des comparaisons entre l'événement courant et un autre, déjà traité.

La réduction du *slowdown* prend fin avec l'*unparsing* de l'AST transformé dans un nouveau code, suivie par l'exécution de celui-ci.

Tab. 6.3: Les événements reconnus et gerés par dPerf.

Type d'action	Formalisme	Pré-défini dans Simgrid	Défini au LIFC pour dPerf
compute	Quel que soit	Oui	Non
comm_size	Quel que soit	Oui	Non
Send	P2P-SAP	Non	Oui
Receive	P2P-SAP	Non	Oui
Send	MPI	Oui	Non
Recv	MPI	Oui	Non
Isend	MPI	Oui	Non
Irecv	MPI	Oui	Non
wait	MPI	Oui	Non
barrier (avec sémaphores)	MPI	Oui	Non
bcast	MPI	Oui	Non
reduce	MPI	Oui	Non
allreduce	MPI	Oui	Non
sleep	MPI	Oui	Non
barrier (sans sémaphores, avec des retards)	MPI	Non	Oui
gather	MPI	Non	Oui
gatherV	MPI	Non	Oui
scatter	MPI	Non	Oui
Ssend_init	MPI	Non	Oui
Recv_init	MPI	Non	Oui
Start	MPI	Non	Oui

6.3.2 Simulation basée sur fichiers de traces

6.3.2.1 Obtenir les fichiers de trace

Le code transformé pendant la réduction du *slowdown* est compilé en utilisant l'un des niveaux d'optimisation du code (voir section 6.2.1.2). L'exécution de ceci génère des fichiers de traces. Il y a un fichier par processus employé. Les fichiers contiennent des temps de calcul de chaque bloc d'instructions, et les paramètres de communication les plus pertinents (voir un extrait de fichier de traces dans le listing 6.1). Le temps de calcul est mesurés avec les compteurs hardware et il est exprimé en nanosecondes. Ces fichiers serviront d'entrée pour la simulation de la topologie réseau ciblée. Ces fichiers prennent en compte le niveau d'optimisation ciblé par le développeur ou l'utilisateur de l'application. De cette façon, le résultat de la phase de simulation prendra en compte le choix d'optimisation utilisé par le compilateur.

6.3.2.2 Ré-jouer les traces

Après avoir obtenu les fichiers de trace, ceux-ci sont rè-joués dans Simgrid. Pour ce type de simulation, un minimum de trois fichiers sont requis : (i) la définition et le traitement des actions, (ii) la description de l'architecture de calcul, et (iii) le ou les fichiers de traces. Le principe est le suivant :

1. Tout événement -de computation ou communication- est défini dans le fichier "actions.c" du module MSG. Ensuite, toute action est simulée par Simgrid selon la définition existante dans le module MSG.
2. l'architecture de calcul est définie dans un fichier plateforme [57]. Celui-ci contient :
   - la puissance de calcul de chacun des nœuds. Pour dPerf, nous avons choisi de fixer la puissance des nœuds à 10^9 pour transformer le temps de computation de nanosecondes en secondes.
   - les différents types de liens utilisés dans l'architecture de calcul pair-à-pair. Un lien est caractérisé par identificateur, bande passante, et latence. La bande passante est exprimée en octets et la latence en seconde.
   - les liens entre chaque deux nœuds afin que l'échange des messages soit possible quel que soit la topologie logique utilisée dans l'application.
3. la simulation commence avec le lancement du module MSG de Simgrid, tout en donnant en entrée un fichier de trace. Le résultat représente le calcul de prédiction pour le code analysé.

6.4 La disponibilité publique de dPerf

L'outil dPerf sera disponible en libre téléchargement sur le site web du projet ANR-CIP [1]. Pour faciliter la distribution de notre outil dans le domaine de l'*HPC* , un guide de l'utilisateur accompagnera les sources de dPerf. Ce guide est généré avec Doxygen et contient une partie des commentaires existants dans le code source de dPerf, ainsi que ses classes et ses fonctions. Au moment de la rédaction de ce manuscrit, aucune interface graphique n'existait pour dPerf. Dans le chapitre "Conclusions et perspectives futures" nous présentons les travaux en cours par rapport à la disponibilité publique de notre outil, ainsi que des extensions qui permettent de faciliter le processus de prédiction de performances avec dPerf.

Troisième partie

Étude de cas

Chapitre 7

Étude de la fonctionnalité et de la précision

Introduction

Ce chapitre a pour but de tester la précision des prédictions faites par dPerf. Dans une première phase, nous testons l'efficacité et l'impact de la réduction du *slowdown* sur le processus de prédiction. Ceci est réalisé sur une implémentation Laplace séquentielle afin de valider la précision de la prédiction des calculs sans les communications. Puis nous analysons la précision des prédictions faites par dPerf pour deux applications distribuées qui sont complètement différentes. Les caractéristiques de celles-ci sont présentées dans le tableau 7.1. La première est issue de la suite de *benchmark* NAS, c'est l'application IS. Cette application est intéressante car elle utilise un ratio élevé temps de communications sur temps de calcul, elle permet donc de compléter les prédictions faites sur l'implémentation de Laplace en séquentiel. La dernière application est une implémentation du problème mathématique de l'obstacle faite dans le cadre du projet ANR CIP. Cette application permet de tester la précision globale de dPerf. dPerf utilise la réduction du *slowdown* pour prédire les performance de ces applications.

Grâce aux techniques simple et optimisée de *benchmarking* par bloc d'instructions, dPerf a un *slowdown* réduit par rapport aux outils de prédiction directement comparables au nôtre (voir la partie I, le chapitre 3). Du point de vue du langage accepté en entrée de dPerf, nous n'exploiterons pas tout le potentiel de notre outil, car les applications choisies sont toutes écrites en C. Nous avons choisi deux applications (voir le tableau 7.1) pour étudier la précision de la prédiction de dPerf. La topologie logique ainsi que le système de calcul pour chacune des applications placent celles-ci dans des environnements *HPC* opposés. Sur l'un d'entre eux, hétérogène, la communication se fait d'une façon centralisée, et sur l'autre, homogène, la topologie utilisé est celle décentralisée. Nous verrons que dPerf réussit à donner des prédictions avec un pourcentage d'erreur acceptable.

TAB. 7.1: Les applications distribuées utilisées pour étudier la précision des prédictions de performances faites par dPerf. (1) NAS Integer Sort ;

N°	Application	Bibliothèque de communication	Topologie de communication	Topologie logique	Type de système	Langage
1	Laplace	aucune	aucune (séquentielle)	séquentiel	-	C
2	NAS IS(1)	MPI	maître-travailleur	centralisée	hétérogène	C
3	Le problème de l'obstacle	P2P-SAP	maille 2d	P2Pdc (décentralisée)	homogène	C

7.1 Précision de la prédiction du temps de calcul

Nous avons choisi une implémentation séquentielle de la formule Laplace pour étudier la prédiction de performances d'un code ne contenant pas de communications. Cet aspect montre l'efficacité des techniques de *benchmarking* par bloc implémentées dans dPerf. Elle est appliquée pendant la réduction du *slowdown* .

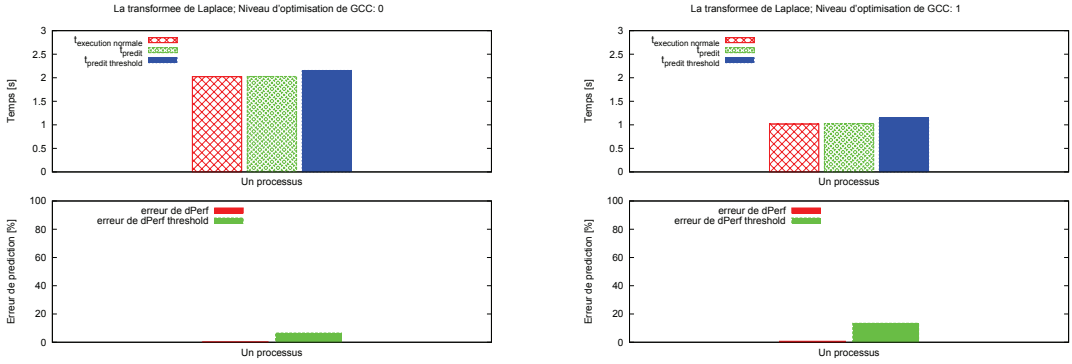
Pour ce faire, dPerf prend les mesures de référence, soit le temps d'une exécution réelle de l'application, sur un nœud Intel Core 2 Duo à 2,26 GHz, 3MB cache, sans utiliser les interfaces réseau. Les valeurs présentées par la suite sont obtenues à partir des temps moyens de 10 exécutions.

Dans la figure 7.1, le temps de référence $t_{execution\,normale}$ est comparé à deux temps prédits par dPerf, l'un utilisant la technique simple de *benchmarking* (t_{predit}), et l'autre appliquant celle de *benchmarking* optimisé ($t_{predit\,threshold}$). L'erreur de la prédiction est représentée par les deux dernières valeurs dans chacune des sous-figures. L'analyse prend en compte chaque niveau d'optimisation du compilateur GCC. Nous remarquons que les prédictions faites par dPerf sont très proches des temps d'exécution, fait confirmé par les pourcentages d'erreur compris entre 0,06 - 0,97%, et 6,53 - 23,21% respectivement pour dPerf avec *benchmarking* simple et optimisé par bloc d'instructions.

Les expérimentations faites avec dPerf et l'implémentation de la transformée de Laplace montrent que l'outil issu de la contribution de cette thèse donne des bons résultats de prédiction du temps de calcul, pour les deux techniques de *benchmarking* . En moyenne, l'erreur de la prédiction du temps de calcul reste acceptable, avec un niveau du *slowdown* égal à l'unité pour tout niveau d'optimisation, raison pour lequel celui-ci n'est pas représenté graphiquement.

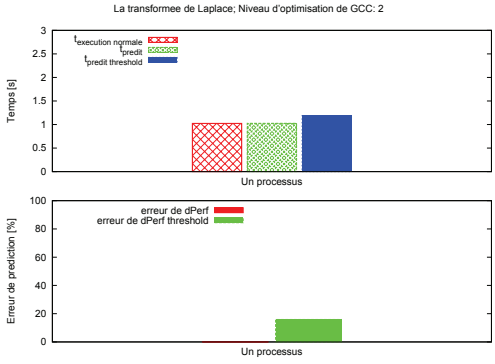
7.2 Prédiction de performance de l'application NAS Integer Sort

La deuxième application choisie est le *benchmarking* IS (*Integer Sort*)[10], un programme parallèle extrait de NAS-PB (*Numerical Aerodynamic Simulation - Parallel Bench-*

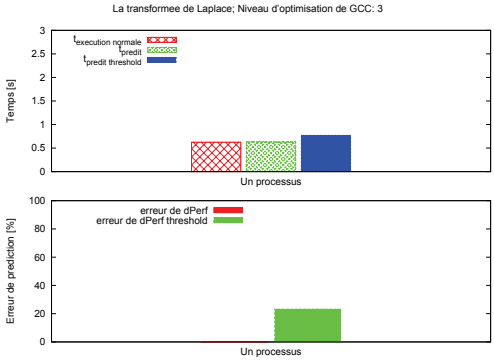


(a)

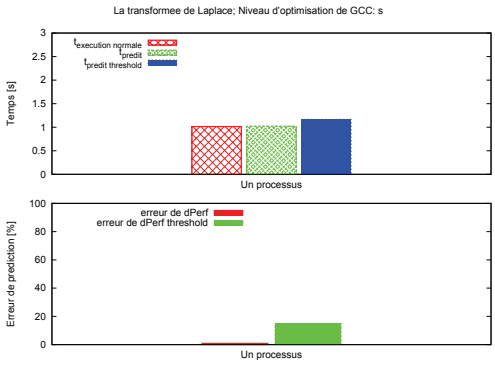
(b)



(c)



(d)



(e)

FIG. 7.1: Exécution, prédiction et erreur à la prédiction pour la transformée de Laplace, implémentation séquentielle.

mark). Cette application fait le tri parallèle d'entiers selon la méthode du tri par réceptacle (*barrel sort*). Ce programme est un test qui utilise intensivement le réseau, car la plupart de son temps d'exécution est représentée par la communication et non pas par le calcul. La figure 7.2 montre, pour les niveaux pertinents d'optimisation de GCC, le pourcentage de communication et de calcul d'une exécution normale de cette application.

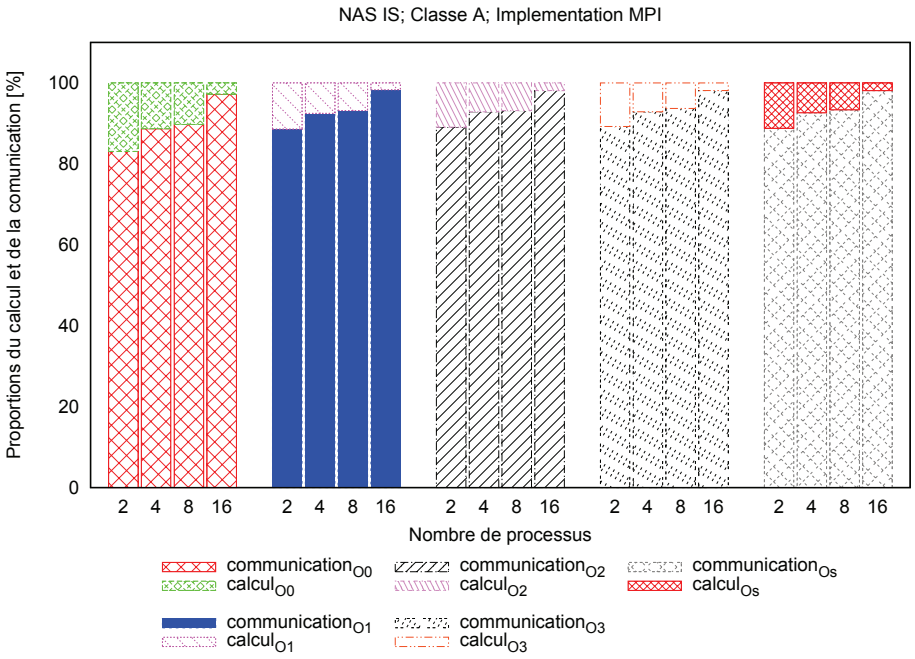


Fig. 7.2: Le pourcentage de calcul et de communication, en fonction du niveau d'optimisation de GCC, pour l'exécution de NAS IS.

Les communications utilisées par NAS IS sont du type point-à-point non-bloquantes, telles que le MPI_Send et le MPI_Irecv, ou collectives, telles que le MPI_Allreduce, le MPI_Alltoall, ou le MPI_Alltoallv.

Deux plate-formes de test ont été mises en place pour observer et étudier la précision de dPerf :

1. L'expérience *IS-Exp1* contenant deux grappes hétérogènes d'un total de 16 nœuds :
   (a) 8 nœuds Intel Pentium D à 2,8 GHz, 1MB cache, cartes réseau à 1 Gbps, connectés par un *switch*(srt_1) HP Procurve 2848 à 1 Gbps par port ;
   (b) 8 nœuds Intel Core 2 Duo à 2,33 GHz, 4MB cache, cartes réseau à 1 Gbps, connectés par un *switch*(srt_2) Cisco Catalyst 2900XL à 100 Mbps par port.
   (c) *Campus Network* est un réseau composé de plusieurs *switch* dans une topologie de type anneau (ou *ring topology*)ayant un *backbone* à 100 Mbps.

   La topologie réseau est présentée dans la figure 7.3.

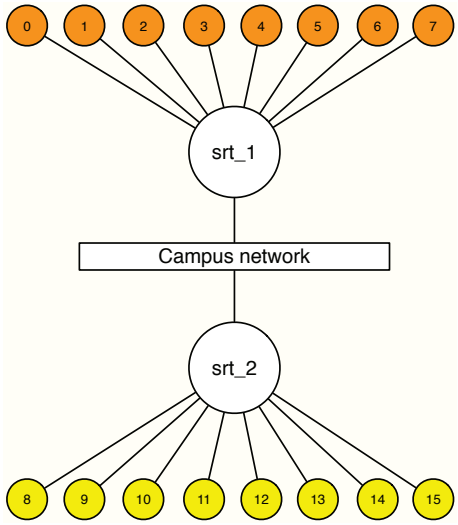


FIG. 7.3: La topologie réseau de la plate-forme *IS-Exp1*. Les nœuds identiques sont représentés avec une même couleur.

2. L'expérience *IS-Exp2* contenant 16 nœuds répartis sur quatre sites pour obtenir un degré d'hétérogénéité élevé par rapport à *IS-Exp1* :
   - nœud 0 (cyan dans la figure 7.4) : Intel Bi-Xenon à 2.8 GHz, 512 KB cache, 3 GB mémoire vivre, carte réseau à 1 Gbps ;
   - nœuds 1,6,13-15 (vert) : Intel Core 2 Duo à 3 GHz, 6 MB cache, 1 GB mémoire vivre, cartes réseau à 1 Gbps ;
   - nœuds 3,7 (violet) : Intel Pentium 4 à 3 GHz, 1 GB cache, 1 GB mémoire vivre, cartes réseau à 1 Gbps ;
   - nœuds 2,4,5,8-12 (orange) : Intel Core 2 Duo à 2.33 GHz, 4 MB cache, cartes réseau à 1 Gbps ;
   - *swrt* un routeur Linksys WRT54GL, avec des ports Ethernet à 100Mbps ;
   - *srt*, un *switch* FORE Systems ES 2810, avec des ports Ethernet à 100Mbps ;
   - *sat*, un *switch* Allied Telesyn AT-FS708LE, avec des ports Ethernet à 100Mbps ;
   - *Campus network*, un réseau connectant le nœud 0, le routeur et les *switch* ci-dessus dans une topologie de type anneau. Le *backbone* est de 100Mbps.

   La taille du problème choisie est celle la plus grande disponible pour un *benchmark* NAS : la classe A.

   Les prédictions de performances de dPerf pour NAS IS sont faites pour un nombre de 2^n noeuds, où $n \in \{1, 2, 3, 4\}$. L'ordre dans lequel sont employés les nœuds est :

$$\begin{cases} noeud0, noeud1 & pour\ n = 1 \\ noeud0 - noeud3 & pour\ n = 2 \\ noeud0 - noeud7 & pour\ n = 3 \\ noeud0 - noeud15 & pour\ n = 4 \end{cases}$$

   La topologie réseau est présentée dans la figure 7.4.

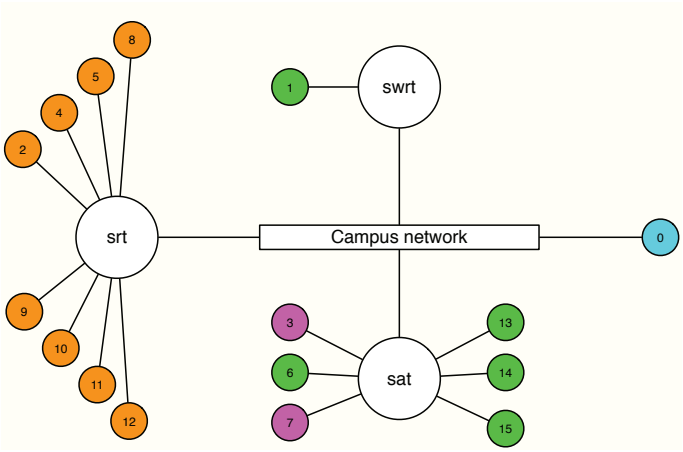


FIG. 7.4: La topologie réseau de la plate-forme *IS-Exp2*. Les nœuds identiques sont représentés avec une même couleur.

IS-Exp1

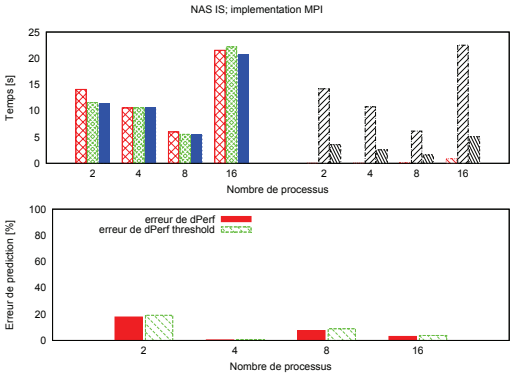
Les premiers résultats de prédiction pour une application distribuée sont montrés dans la figure 7.5. Trois groupes de résultats sont affichés. Le premier groupe, soit la première série de 2, 4, 8, et 16 processus, servent à comparer le temps de l'exécution avec la prédiction faite par dPerf. Dans ce cas, chacune des deux techniques de *benchmarking* par bloc d'instructions seront utilisées a leur tour. Le deuxième groupe, soit la série de 2 à 16 processus suivante, présent le pourcentage d'erreur à la prédiction. La troisième serie de 2 à 16 processus montre les valeurs suivantes :

1. le coût de la simulation ;
2. le coût total pour accéder le résultat de la prédiction faite par dPerf ;
3. le coût total pour accéder le résultat de la prédiction faite par dPerf en utilisant la technique optimisée de *benchmarking* par bloc d'instructions.

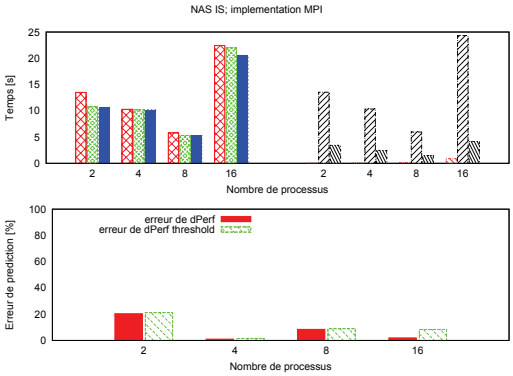
Pour cette application, dPerf à une précision qui augmente avec le nombre de nœuds participants. L'erreur d'approximative 23% obtenue pour 2 nœuds diminuera pour 4, 8 et 16 nœuds en dessous 11%. Cette amélioration de la précision est liée au caractère intensivement communiquant de NAS IS. Avec une augmentation du nombre de nœuds, le temps de calcul devient peu important par rapport au temps de communication.

Par rapport aux outils mentionnés dans les chapitres 1- 3, dPerf est caractérisé par une valeur réduite du *slowdown* par processus simulé. Si la plupart des prédictions sont obtenus pour un *slowdown* inférieur ou égal à 1, la méthode utilisée est caractérisée par un gain, et non pas par un *slowdown* . Dans la figure 7.6, nous présentons le *slowdown* et le gain du processus de prédiction avec dPerf pour le scénario *IS-Exp1*. La première série de résultats est obtenue pour la prédiction utilisant le *benchmarking* simple, quant pour la deuxième série, dPerf applique la technique optimisée de *benchmarking* par bloc.

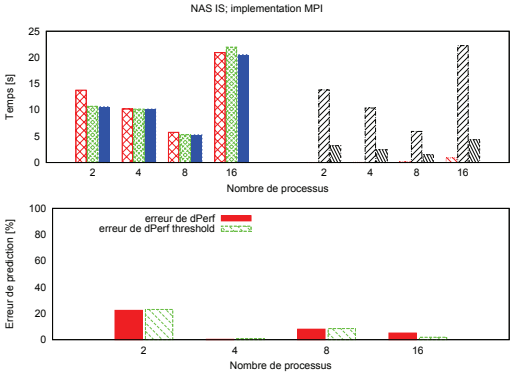
Les proportions du temps d'exécution de NAS IS qui sont attribuées à la communication et au calcul sont celles présentées dans la figure 7.2. Ce pourcentage est indépendant du système hôte. Dans le cas de NAS IS, la précision de dPerf n'est pas diminuée par la



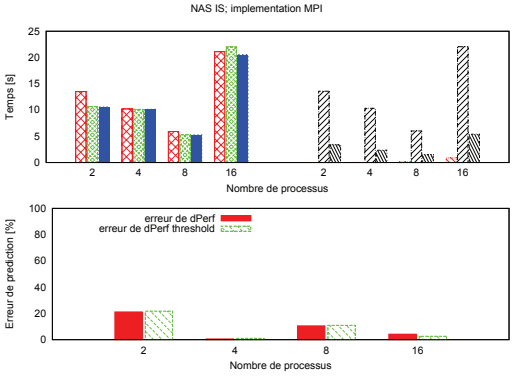
(a) Niveau 0 d'optimisation



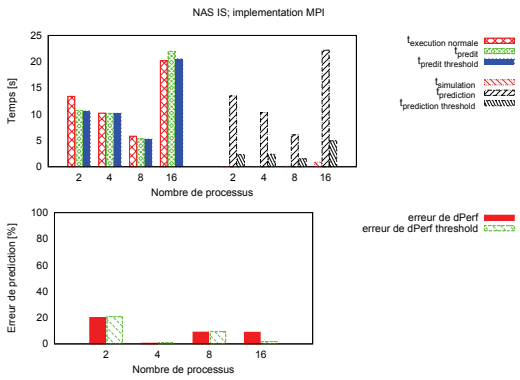
(b) Niveau 1 d'optimisation



(c) Niveau 2 d'optimisation

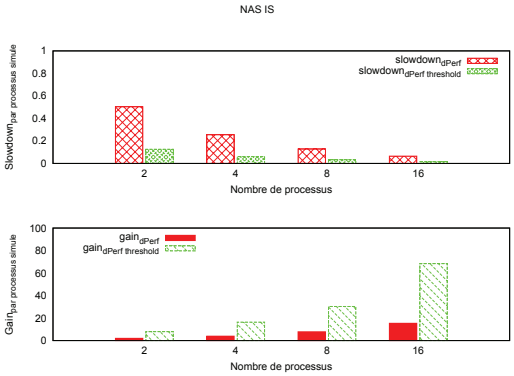


(d) Niveau 3 d'optimisation

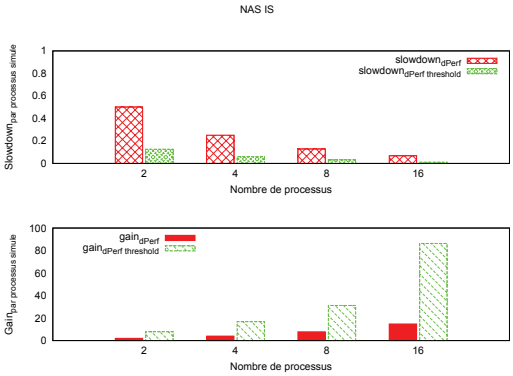


(e) Niveau s d'optimisation

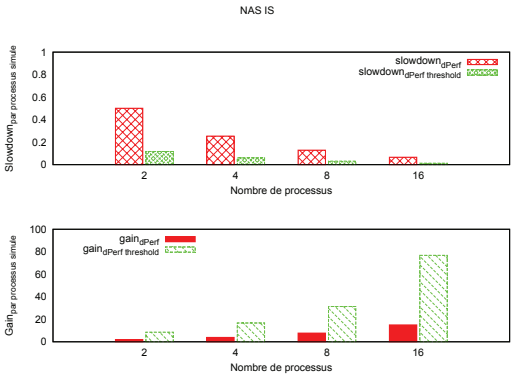
FIG. 7.5: Le scenario *IS-Exp1*. Le temps prédit est comparé au temps d'exécution. L'erreur montre la précision de dPerf par rapport à la réalité. Les résultats sont étudiés pour tous les niveaux d'optimisation pertinentes disponibles dans GCC.



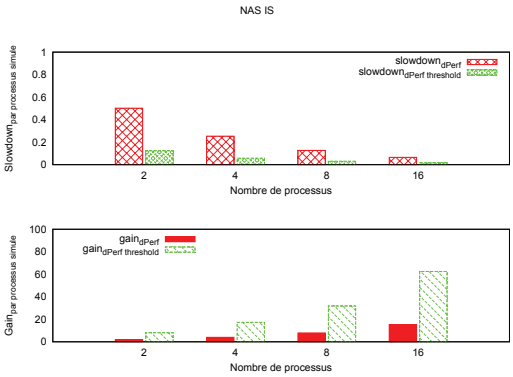
(a) Niveau 0 d'optimisation



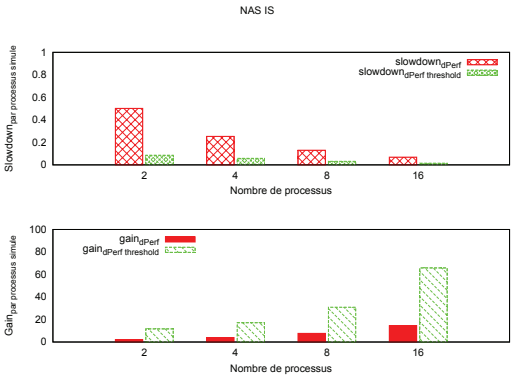
(b) Niveau 1 d'optimisation



(c) Niveau 2 d'optimisation



(d) Niveau 3 d'optimisation



(e) Niveau s d'optimisation

FIG. 7.6: Le *slowdown* et le gain pour le scénario *IS-Exp1*.

taille de messages, car celles-ci sont de l'ordre des mégaoctets. Dans ce cas, la contrainte de Simgrid MSG liée à la taille des messages envoyés (voir la section 6.2.2.2.3) ne s'applique pas.

IS-Exp2

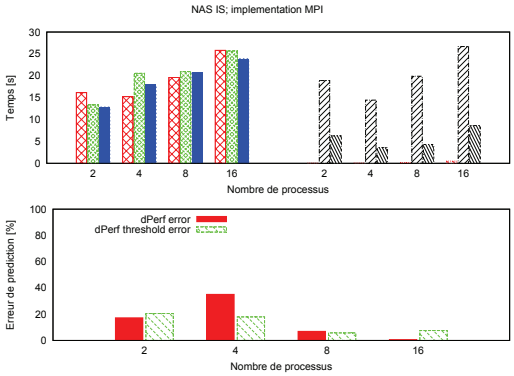
Les résultats pour le deuxième scénario sont montrés dans la figure 7.7. Les mêmes trois groupes de valeurs sont présentés, comme décrit en début du scénario *IS-Exp1*. La précision de dPerf augment avec le nombre de nœuds. Une erreur de 56% est obtenue pour 4 nœuds, mais celle-ci descend en dessous 23% pour 8 et 16 nœuds. Ce phénomène est causé par le nombre important de communications, le nombre réduit de blocs séquentiels, et le niveau élevé d'hétérogénéité du système.

Pour ce scénario, nous remarquons aussi le *slowdown* très réduit, et le gain qui caractérise d'avantage la prédiction faite par dPerf(voir la figure 7.8).

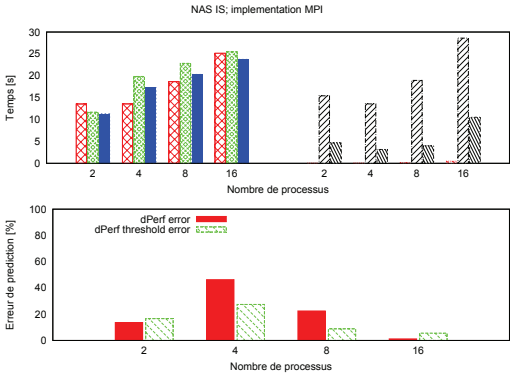
Les proportions du temps d'exécution de NAS IS qui sont affectée à la communication et au calcul sont celles présentées dans la figure 7.2. Comme mentionnée dans la section précédente, ce pourcentage est indépendant du système hôte. Dans le cas de NAS IS, la précision de dPerf n'est pas diminuée par la taille de messages, car celles-ci sont de l'ordre des mégaoctets. La contrainte de Simgrid MSG liée à la taille des messages envoyés (voir la section 6.2.2.2.3) ne s'applique pas.

La prédiction de performances de NAS IS nous a permis d'étudier deux caractéristiques de dPerf :

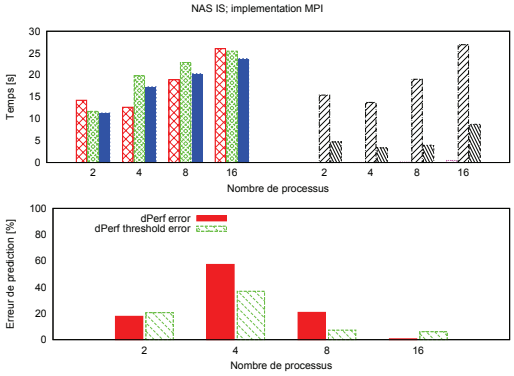
1. la précision à la prédiction de performances d'une application distribuée C/MPI. L'erreur de dPerf diminue avec l'augmentation du nombre de nœuds ;
2. la rapidité d'obtention du résultat de prédiction. Celle-ci se reflète dans les valeurs du *slowdown* et du gain.



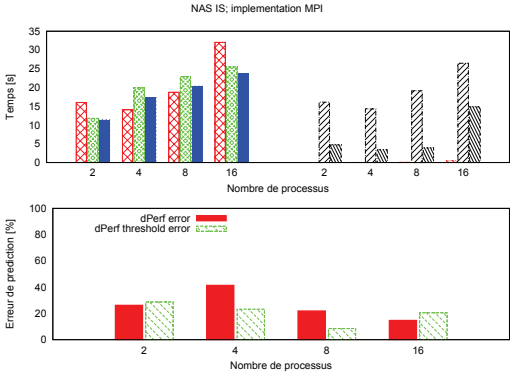
(a) Niveau 0 d'optimisation



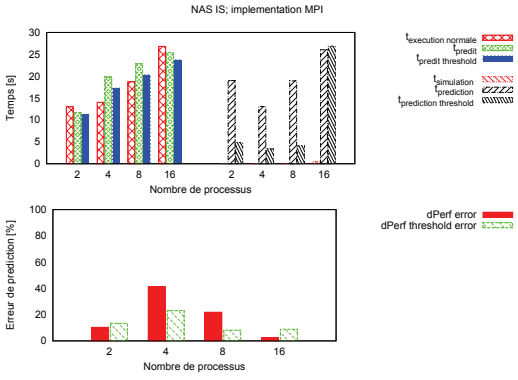
(b) Niveau 1 d'optimisation



(c) Niveau 2 d'optimisation

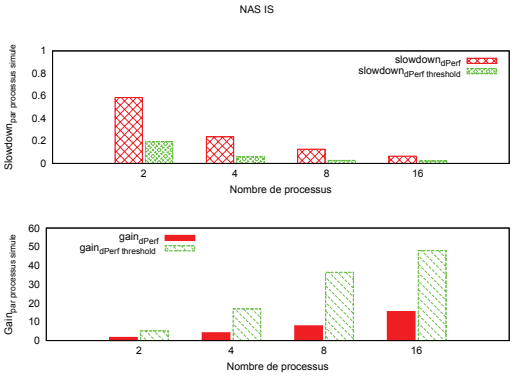


(d) Niveau 3 d'optimisation

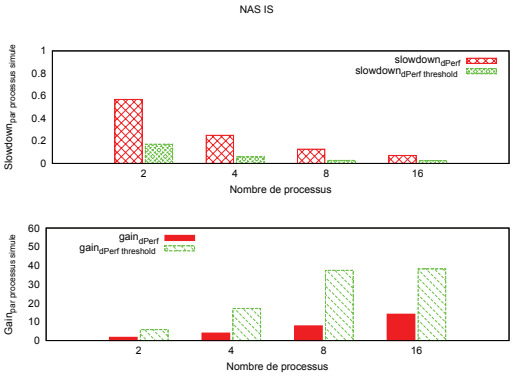


(e) Niveau s d'optimisation

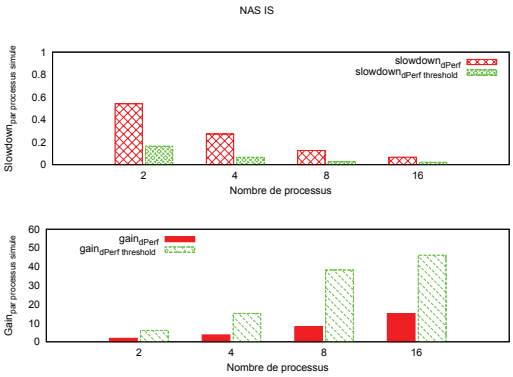
FIG. 7.7: Le scénario *IS-Exp2*. Le temps prédit est comparé au temps d'exécution. L'erreur montre la précision de dPerf par rapport à la réalité. Les résultats sont étudiés pour tous les niveaux d'optimisation pertinentes disponibles dans GCC.



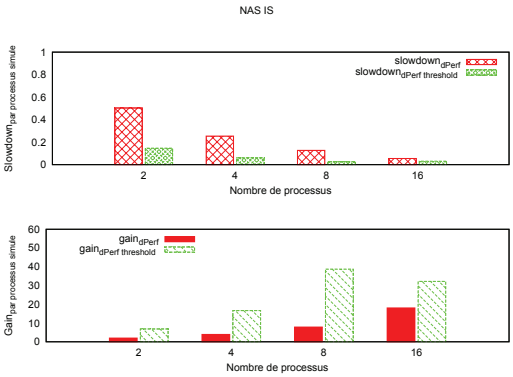
(a) Niveau 0 d'optimisation



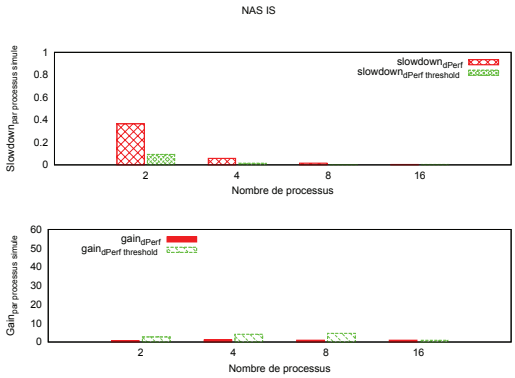
(b) Niveau 1 d'optimisation



(c) Niveau 2 d'optimisation



(d) Niveau 3 d'optimisation



(e) Niveau s d'optimisation

FIG. 7.8: Le *slowdown* et le gain pour le scénario *IS-Exp2*.

7.3 Prédiction de performance du problème de l'obstacle

La troisième application que nous avons choisie pour étudier la précision de dPerf est l'implémentation C du problème de l'obstacle, une application réelle appliquée en mathématique et en mécanique. Cette application est développée au laboratoire IRIT-ENSEEIHT[99] à Toulouse (en France). Une une implémentation C/P2P-SAP est faite au laboratoire LAAS-CNRS à Toulouse (France). Le but de cette implémentation est de déployer ce code dans un environnement de calcul intensif parallèle pair-à-pair. À cet effet, l'environnement P2Pdc à été proposé par Nguyen et al. dans [80]. Dans cet environnement, le code de l'obstacle communique en utilisant le protocole P2P-SAP[49]. L'implementation C/P2P-SAP est faite dans le cadre du projet ANR CIP (Calcul Intensif Pair à pair)[1] pour utiliser le protocole de communication P2P-SAP.

La figure 7.9 montre, pour chaque niveau d'optimisation de GCC, le pourcentages de communication et de calcul d'une exécution normale de l'application.

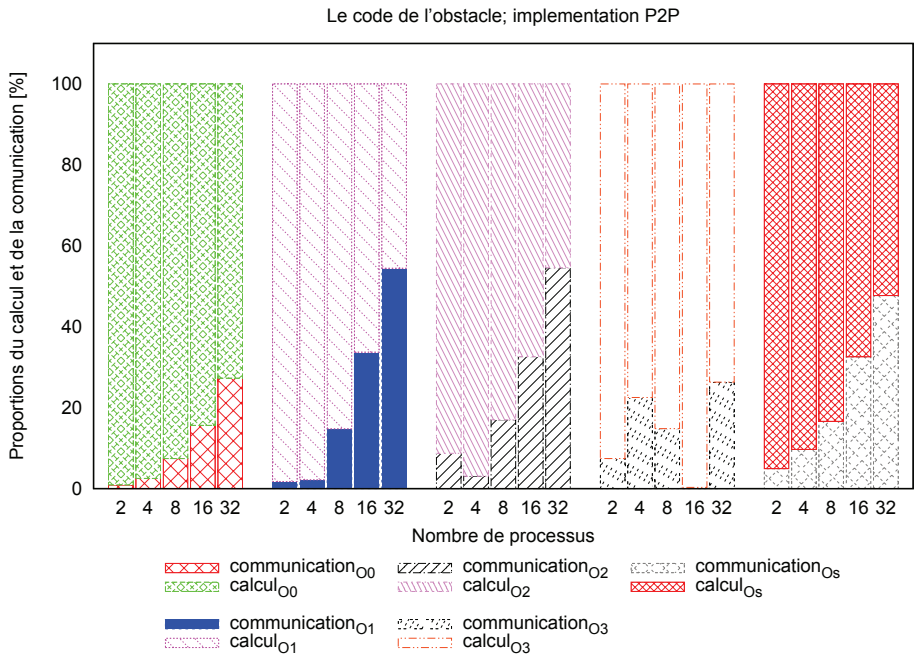


Fig. 7.9: Le pourcentage de calcul et de communication, en fonction du niveau d'optimisation de GCC, pour l'exécution du code de l'obstacle.

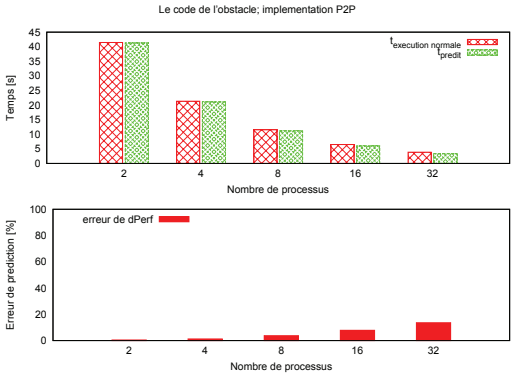
Une grappe de calcul homogène (d'un total de 32 nœuds) a été utilisée pour l'étude de dPerf avec l'application de l'obstacle. Cette grappe se nomme *Bordeplage* et elle fait partie du système Grid5000[4], une grille de calcul contenant environ 5000 nœuds repartis sur 9 sites à travers la France.

- 32 nœuds Intel Xeon EM64T à 3 GHz, 1 MB L2 cache, 2 GB mémoire vivre ;
- un cœur par processeur et un processeur par nœud est employé ;
- les cartes réseau des nœuds sont à 1 Gbps, avec une latence de 100 microsecondes ;
- le *backbone* de la grappe est à 10 Gbps, avec une latence de 100 microsecondes.

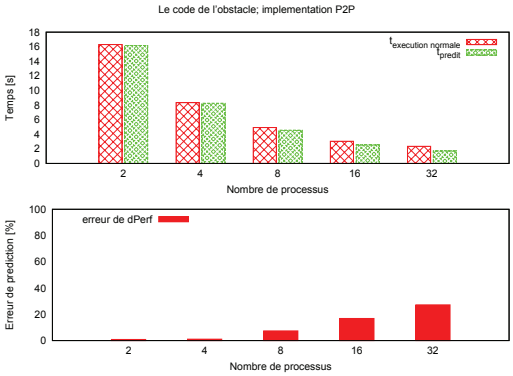
La figure 7.10 montre les résultats de la prédiction pour tous les niveaux pertinents d'optimisation du compilateur. Le temps prédit est comparé au temps d'exécution, et l'erreur de prédiction est exprimée en pourcentage.

Les pourcentage de temps que l'exécution du code de l'obstacle en P2P passe dans le calcul ou dans la communication sont celles présentées dans la figure 7.2. Ce pourcentage est indépendant du système hôte. Dans le cas du problème de l'obstacle, la précision de dPerf est affectée par la taille de messages, car celles-ci ne passent pas 10 Ko. Dans ce cas, la contrainte de Simgrid MSG liée à la taille des messages envoyés (voir la section 6.2.2.2.3) s'applique. L'augmentation du nombre de nœuds entraine une majoration du taux d'erreur, car les messages diminuent en taille et leur fréquence augmente.

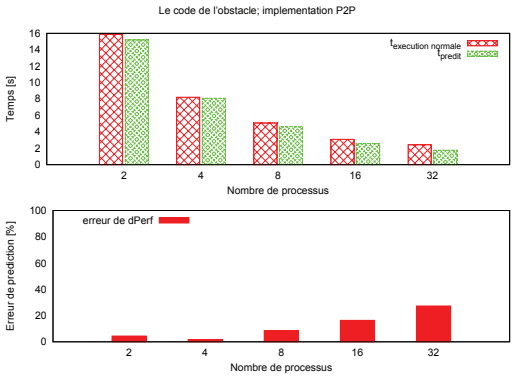
La prédiction de performances du code de l'obstacle nous a permis d'étudier la précision de dPerf à la prédiction de performances d'une application distribuée C. Cette application utilise la bibliothèque de communication P2P-SAP dans un environnement pair-à-pair décentralisé.



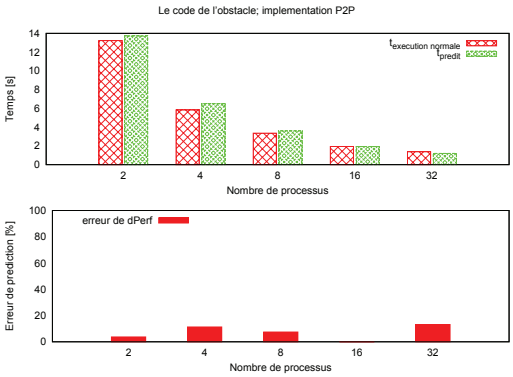
(a) Niveau 0 d'optimisation



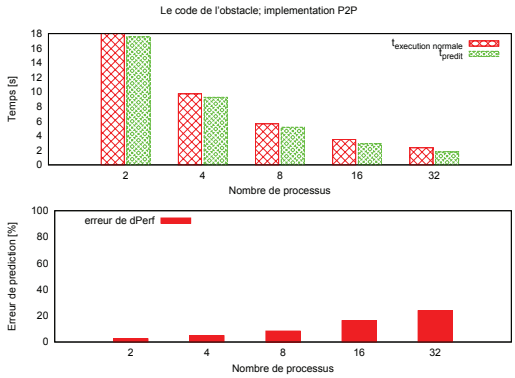
(b) Niveau 1 d'optimisation



(c) Niveau 2 d'optimisation



(d) Niveau 3 d'optimisation



(e) Niveau s d'optimisation

FIG. 7.10: Le temps d'exécution, la prédiction et l'erreur de dPerf pour le code de l'obstacle.

Chapitre 8

Étude d'une application : le problème de l'obstacle

Introduction

Le problème de l'obstacle est un problème classique de mécanique. L'implémentation étudiée dans ce chapitre à été présentée dans la section 7.3. Nous allons présenter l'identification de la topologie de communication utilisée dans l'application, ainsi que deux types de prédictions portées sur le réseau et sur le nombre de nœuds de l'architecture de calcul.

L'architecture de calcul hôte utilisée pour la prise des mesures de référence est une grappe de calcul homogène. Celle-ci fait partie du système Grid5000[4], elle se nomme *Bordeplage*, et ses principales caractéristiques sont :

- 32 nœuds Intel Xeon EM64T à 3 GHz, 1 MB L2 cache, 2 GB mémoire vivre ;
- un cœur par processeur et un processeur par nœud est employé ;
- les cartes réseau des nœuds sont à 1 Gbps, avec une latence de 100 microsecondes ;
- le *backbone* de la grappe est à 10 Gbps, avec une latence de 100 microsecondes.

La taille du problème de l'obstacle que nous avons choisi est de 64 avec des communications synchrones.

8.1 Identification de la topologie logique

Pendant la phase de l'extrapolation de performances, dPerf identifie dans le code de l'obstacle les lignes qui participent à la définition de la topologie logique. Plus précisément, dPerf identifie l'obtention du rang, le tableau des nœuds voisins, et les communication qui utilisent tous ces détails. La figure 8.1 est une capture d'écran d'une structure conditionnelle IF contenant un appel de fonction de communication. Celle-ci est obtenue a partir de l'AST du code d'entrée (voir 5.4.2.1) à l'aide des méthodes de ROSE. Les éléments pertinents identifiés, dPerf instrument l'application à l'aide de l'AST. De cette façon, pendant la phase d'extrapolation de performances, dPerf obtiendra le résultat de l'identification de la topologie, selon les conditions de la section 6.2.2.1. Le résultat de l'identification obtenu par dPerf indique que cette application communique en utilisant

une topologie maille en 2 dimensions. Cette information est utilisé par dPerf dans 8.3 pour l'extrapolation des résultats.

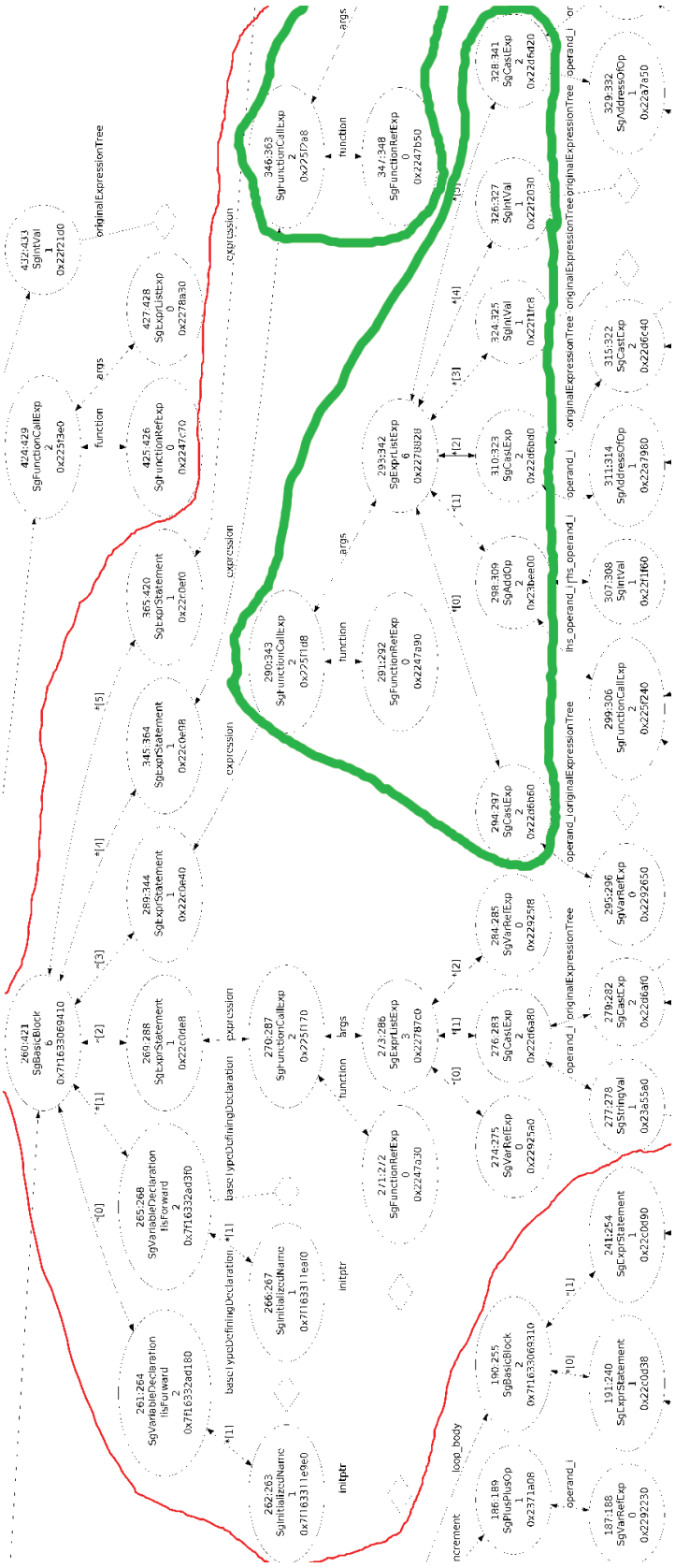


FIG. 8.1: Identification d'un appel de communication à l'aide de l'AST du code de l'obstacle.

8.2 Prédiction de performances sur un réseau cible

Après avoir étudié la précision de dPerf (voir la section 7.3), nous l'utiliserons pour prédire des performances similaire aux celles du système hôte sur un système cible. Les deux systèmes se différencient par rapport à la topologie réseau, les nœuds et leur nombre étant fixés.

Trois nouvelles topologies réseau sont testées :

1. Grid5000-Bordeplage (identique à celle réelle) : des cartes réseau à 1 Gbps pour chaque nœud connectées à un *backbone* à 10 Gbps, avec une latence de 100 micro-secondes ;
2. Daisy-xDSL [72, 51, 47] (voir la figure 8.2) :

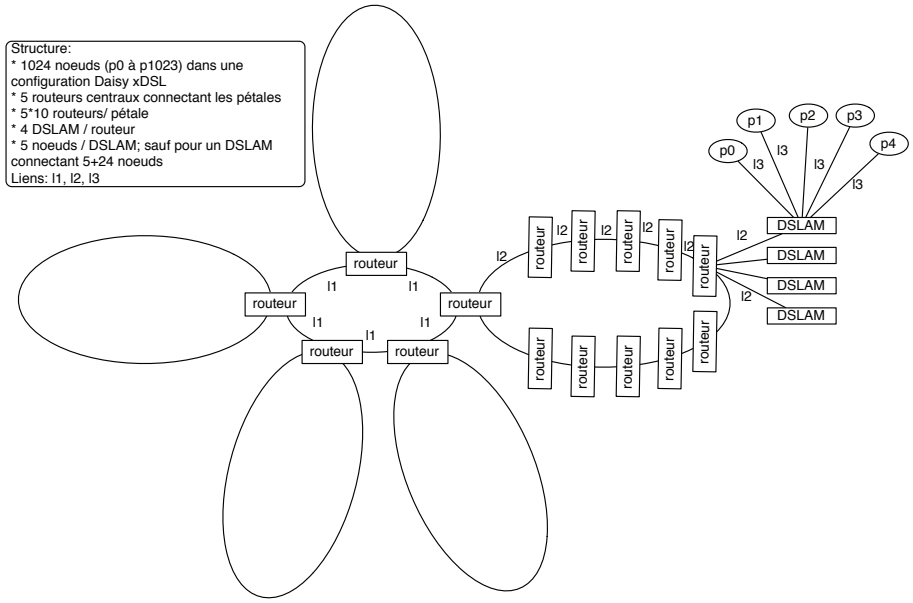


Fig. 8.2: La topologie réseau de la marguerite (*daisy*).

   - 5 routeurs centraux ;
   - 5 pétales, chacune contenant(connectant) 10 routeurs ;
   - 4 unités DSLAM ; Chaque routeurs situé sur les pétales connecte ces 4 unités ;
   - 5 nœuds seront connectés à chaque DSLAM, sauf ceux centraux ;
   - tous les liens connectant les routeurs ont une bande passante de 10 Gbps, sauf l'anneau central dont la bande passante à 100 Gbps ;
   - tous les liens connectant un DSLAM à un routeur ont une bande passante de 10Gpbs ;
   - les liens connectant un nœud à un DSLAM ont une bande passante entre 5 et 10 Mbps, valeur attribuée aléatoirement.
3. LAN : des cartes réseau à 100 Mbps pour chaque nœud connectées à un *backbone* à 1 Gbps ;

La prédiction de performance du système utilisant l'une des trois topologies réseau mentionnées ci-dessus est présentée dans la figure 8.3. Ce type de prédiction faite par

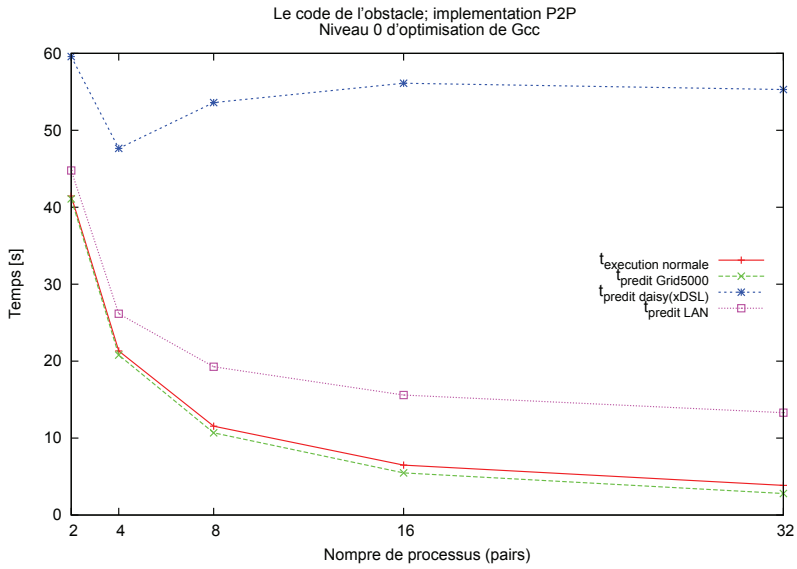


FIG. 8.3: Vue comparative des performances du code de l'obstacle pour des différentes topologies réseau testées avec dPerf.

dPerf nous permet d'étudier, pour une application analysée telle que le code de l'obstacle, quel est l'impact de la topologie réseau sur les performances du système. La prédiction de dPerf montre l'équivalent de puissance en fonction de la configuration du réseau. Un résumé de la figure ci-dessus est présenté dans le tableau 8.1

8.3 Extrapolation du nombre de nœuds

Un autre type de prédiction faite par dPerf implique la modification du nombre de nœuds pour un système cible. dPerf applique les principes de la section 6.2.2.2.2, en fonction de (i) la topologie logique identifiée dès la prédiction faite pour 2 nœuds, (ii) et

TAB. 8.1: Comparaison des puissances de calcul entre le système hôte et les prédictions faite par dPerf pour le code de l'obstacle.

Nœuds		Performance	Nœuds	
nombre	topologie		nombre	topologie
4	xDSL	inférieure par peu à	2	Grid5000
2	LAN	inférieure par peu à	2	Grid5000
4	LAN	inférieure par peu à	4	Grid5000
8	LAN	égale à	4	Grid5000
32	LAN	inférieure par peu à	8	Grid5000

des fichiers de trace issus de l'exécution du code transformé.

La topologie logique de l'application de test choisie a été identifiée dans la section 8.1 : une maille en 2d. À partir de cette information et des fichiers de trace obtenus pour 2 nœuds, dPerf prédit les performance du code de l'obstacle pour 2^n nœuds, où $n \in [2;8] \cap \mathbb{N}$. Le résultat de la prédiction est présenté dans la figure 8.4. L'extrapolation est comparée avec des exécutions réelles pour 2, 4, 8, 16 et 32 nœuds. Au delà de 32 nœuds, dPerf peut prédire les performances du code de l'obstacle à base de fichiers de traces et de la topologie réseau identifiée. L'erreur importante indiquée sur chaque sous-figure est due à la taille des messages échangés. Le simulateur MSG de Simgrid est optimisé pour des messages supérieures à 100 Ko. En dessous de cette taille, le coût de traitement d'une communication devient important et introduit des perturbations dans la simulation. À cause de ceci, plus le nombre de nœuds augmente, plus il y aura des perturbation importantes.

Ces expériences montrent que l'outil dPerf réussit a proposer des prédictions de performances des applications distribuées, quelle que soit la topologie logique - centralisée ou décentralisée- et pour différentes bibliothèques de communication.

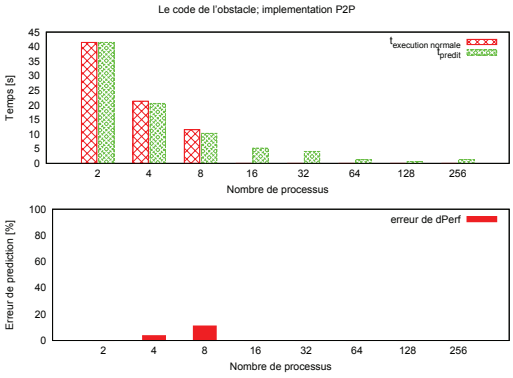
Conclusion

Dans ce chapitre, nous avons étudié la prédiction de performances du code de l'obstacle. Cette application ([79] p.58-62) appartient a une classe de problèmes de simulation très répandues.

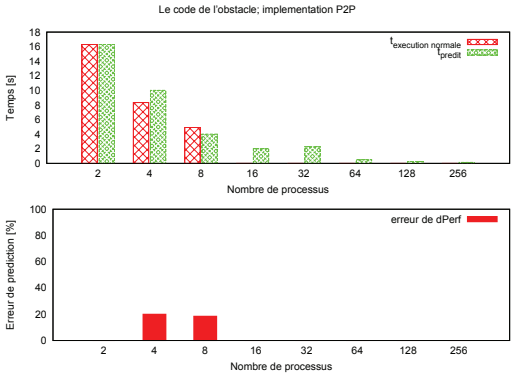
Le code de l'obstacle nous à été mis à disposition par le LAAS/CNRS, un de nos partenaires du projet ANR-CIP. Il est appliqué dans des domaines tels que la mécanique et la mathématique financière. Prédire les performances du code de l'obstacle avec notre méthodologie montre que nous proposons des solutions de prédictions de performances pour des applications dans des conditions réelles. Une particularité du code de l'obstacle que nous avons utilisé est son implémentation C/P2Pdc. Ceci signifie que le problème à été adapté à l'environnement de calcul décentralisé P2P nommé P2Pdc[80, 44][6]. L'environnement P2Pdc permet le lancement du code de l'obstacle sur un système P2P, communiquant de façon décentralisé. Les auteurs de P2Pdc proposent un environnement nouveau afin de faciliter le calcul HPC pour un coût réduit, tout en gardant une fiabilité du système. Nous avons donc proposé une méthodologie qui permet l'analyse et la prédiction des applications HPC habituelles, mais aussi la prédiction de performances des applications nouvelles pour le P2P décentralisé.

Ce chapitre montre une étude de notre méthodologie pour prédire les performances d'un problème exécuté dans de conditions réelles, avec des applications dans les domaines des mathématique et de la mécanique. De cette manière, la méthodologie que nous proposons permet de prédire des performances dans des conditions réelles, pour des applications répandues.

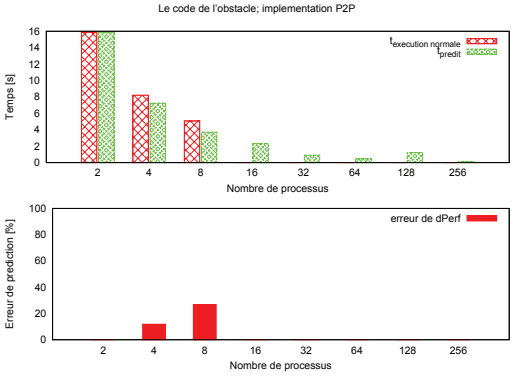
[6] P2Pdc est un environnement de calcul P2P décentralisé de haute performance développé au LAAS-CNRS à Toulouse en France



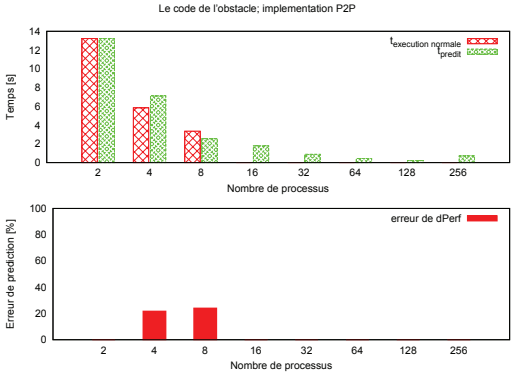
(a) Niveau 0 d'optimisation



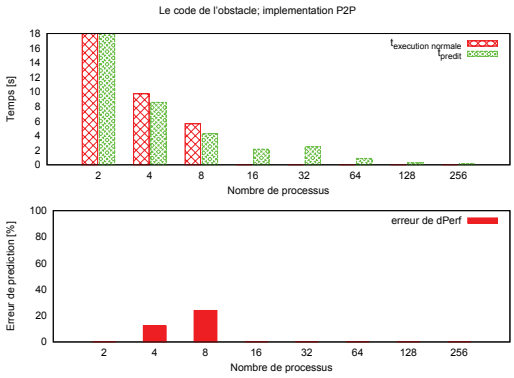
(b) Niveau 1 d'optimisation



(c) Niveau 2 d'optimisation



(d) Niveau 3 d'optimisation



(e) Niveau s d'optimisation

FIG. 8.4: Prédiction de performance du code de l'obstacle avec dPerf, à base des traces pour 2 nœuds

Conclusion

Le travail de recherche présenté souhaite contribuer au domaine du calcul de haute performance avec un outil de prédiction de performances. Cet outil nommé dPerf est une approche qui utilise la réduction du *slowdown* et l'extrapolation de performances. Par dPerf, nous proposons une méthode de prédiction de performances pour les codes MPI, mais surtout pour les applications distribuées exécutées sur un système pair-à-pair (P2P) communicant selon une topologie décentralisée.

Le domaine *HPC* demande que les ressources de calcul mises à la disposition des scientifique soient les plus compétitives possible. Les architectures de calcul suivent de plus en plus le modèle de la mondialisation économique par le fait que les nœuds de calcul sont de moins en moins concentrées sur un même site informatique. Cette répartition des ressources implique l'étude des performance des systèmes, ainsi que l'analyse des nouvelles applications qui seront optimisée pour ce type d'architecture de calcul. Notre outil trouve son application dans un cadre *HPC* ou les investissement dans des nouveaux ressources de calcul doit être attentivement calculé.

Une étude des méthodes existantes nous montre qu'il n'y a pas une solution générique au problème de la prédiction de performances en ce qui concerne les différentes langages de programmation, les multiples bibliothèques de communication, et la diversité des systèmes *HPC* . dPerf est une méthode qui propose l'analyse des performances des applications parallèles ou distribuées C, C++ ou Fortran. L'outil vient avec support pré-défini pour deux bibliothèques de communication, l'une très répandue - le MPI- et l'autre en cours de développement - le P2P-SAP- pour les systèmes *HPC* pair-à-pair. La modularité de dPerf permette aux développeurs de définir des nouveaux formalismes de communication si ils le souhaite, ainsi acceptant, en vue d'une analyse, une plage plus répandue des applications. L'approche issue de cette thèse contient deux technique de *benchmarking* par blocs d'instructions, contribuant au *slowdown* réduit et à un gain important.

À travers les expérimentations, nous avons montré la capacité de dPerf d'analyser rapidement des différentes applications. Ce type d'analyse permet d'identifier les aspect d'un code source liés à la topologie de communication, aux échanges de messages, et au calcul séquentiel. Dans la phase de réduction du *slowdown* , l'exécution du code transformé contribue à l'identification des paramètres clés des communications pour créer les trace d'exécution. Dans le cas de certains applications, cette étape obtient les information sur la dépendances entres données qui manquaient pendant l'analyse des représentations intermédiaires. La phase de la réduction du *slowdown* sert à préparer l'étape finale de la prédiction de dPerf, soit l'extrapolation de performances. Avec les informations acquises auparavant, à ce stade dPerf peut proposer deux types de prédiction : (i)d'une topologie réseau cible ; (ii) ou d'un système cible contenant un nombre extrapolé de nœuds de calcul. Les tests ont validé l'approche que nous proposons, l'erreur à la prédiction se trouvant, en général, dans un intervalle accepté.

Perspectives

Les études effectuées pendant ce travail de recherche ont ouvert des perspectives d'amélioration de la contribution et de développement de l'outil dPerf et de son applicabilité. Dans un première temps, nous souhaitons étudier la précision de dPerf sur une gamme plus importante de systèmes. Au même temps, nous envisageons l'analyse de plusieurs applications *HPC* distribuées P2P qui seront écrites dans C++ et Fortran, afin de mieux justifier le choix du *framework* ROSE pour la réduction du *slowdown* .

Nous analysons la possibilité d'ajouter à la liste des langages acceptés le Java qui n'est pas reconnu par ROSE. Pour cela, une intégration dans dPerf est envisagée pour l'outil de prédiction des performances des applications Java/JNGI, nommé P2PPerf[50, 51]. Avec ces fonctionnalités héritées, dPerf proposera des solutions pour une gamme d'applications encore plus répandue, notamment celles utilisant une topologie centralisée dans un environnement P2P JNGI.

Une étude est mené pour l'intégration de dPerf dans le *framework* Simgrid. Nous envisageons de proposer dPerf en tant que bibliothèque de prédiction de performances intégrée dans Simgrid. De cette manière, l'outil issue de cette thèse sera mis à la disposition des scientifiques à travers les sources du *framework* Simgrid, une importante communauté des chercheurs pouvant ainsi en bénéficier.

Du point de vue de la complexité des systèmes d'aujourd'hui, nous nous intéressons aux méthodes de prédiction du temps de calcul des applications sur des nœuds multi-cœur.

Les travaux futurs dans le domaine du *HPC* ne se limitent pas à la hétérogénéité des systèmes P2P, et pour cette raison, nous envisageons une applicabilité de dPerf dans la prédiction des performances pour les applications *Cloud*. Pour ceci, une collaboration est en cours entre LIFC/DISC (Université de France-Comté, France) et Maths&CS (Emory University, États-Unis) pour la recherche d'un mécanisme d'adaptation automatique des applications *HPC* et P2P pour le *Cloud*.

La précision de dPerf est une caractéristique fondamentale que nous devrons adapter aux applications et aux systèmes à modéliser. Pour ceci, une première étape des travaux futurs consiste à intégrer les dernières améliorations des outils sur lesquels est développé dPerf. Plus précisément, nous envisageons d'utiliser la version de Simgrid MSG proposée par Clauss et al.[41] qui aura un impact positif direct sur la précision de dPerf.

Publications

1 Des conférences internationales avec comité de lecture

Bogdan F. Cornea and Julien Bourgeois. Performance Prediction of Distributed Applications Using Block Benchmarking Methods. In PDP'11, 19-th Int. Euromicro Conf. on Parallel, Distributed and Network-Based Processing, Ayia Napa, Cyprus, pages 183–190, February 2011. IEEE Computer Society Press.

Bogdan F. Cornea, Julien Bourgeois, The Tung Nguyen, and Didier El-Baz. Performance Prediction in a Decentralized Environment for Peer-to-Peer Computing. In HotP2P'11, Int. Workshop on Hot Topics in Peer-to-Peer Systems, IPDPS Workshops, Anchorage, Alaska, United States, May 2011. IEEE Computer Society Press.

Julien Bourgeois, Vaidy Sunderam, Jaroslaw Slawinski, and Bogdan Cornea. Extending executability of applications on varied target platforms. In HPCC'11 : 13-th IEEE Int. Conf. on High Performance Computing and Communications, September 2011. IEEE Computer Society.

2 Des revues internationales avec comité de lecture

Bogdan F. Cornea and Julien Bourgeois. A Framework for Efficient Performance Prediction of Distributed Applications in Heterogeneous Systems. Journal of Supercomputing. July 2011. Note : soumis.

3 Des rapports techniques internes

Bogdan Cornea and Julien Bourgeois. Simulation of a P2P Parallel Computing Environment - Introducing dPerf, A Tool for Predicting the Performance of Parallel MPI or P2P-SAP Applications. Technical Report RT2010-04, LIFC - Laboratoire d'Informatique de l'Université de Franche Comté, March 2010.

Bibliographie

[1] ANR CIP project web page. http ://spiderman-2.laas.fr/CIS-CIP. 86, 111, 126

[2] CETUS project webpage. http ://cetus.ecn.purdue.edu. 53

[3] The Clusterix project. http ://www.clusterix.pcz.pl. 26

[4] Grid5000 platform. http ://www.grid5000.fr. 126, 129

[5] The Jumpshot tool webpage. http ://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm. 49

[6] The KOJAK tool. http ://www2.fz-juelich.de/jsc/kojak/. 50

[7] LLVM project webpage. http ://llvm.org. 53

[8] The message passing interface standard. http ://www-unix.mcs.anl.gov/mpi. 83

[9] monALISA project. http ://monalisa.caltech.edu. 26

[10] NAS parallel benchmark official website. http ://www.nas.nasa.gov/Resources/Software/npb.html. 97, 116

[11] The NS2 network simulator. http ://www.isi.edu/nsnam/ns. 38

[12] The OMPItrace tool. http ://www.bsc.es/media/1382.pdf. 50

[13] Open64 compiler. http ://www.open64.net/. 40, 53

[14] OSCAR project webpage. http ://www.kasahara.elec.waseda.ac.jp/intro.en.html. 53

[15] PAPI project website. http ://icl.cs.utk.edu/papi/. 65

[16] PAPI sc2008 handout (papi-2008.pdf). http ://icl.cs.utk.edu/graphics/posters/files/. 65

[17] The Paraver tool webpage. http ://www.bsc.es/plantillaA.php ?cat_id=485. 50

[18] Perfmon project webpage. http ://perfmon2.sourceforge.net/. 64

[19] PIPS project website. http ://http ://pips4u.org. 53

[20] PoCC project webpage. https ://www-roc.inria.fr/~pouchet/software/pocc/doc/htmldoc/htmldoc/main.html. 53

[21] Polaris compiler. http ://polaris.cs.uiuc.edu/polaris/polaris-old.html. 53

[22] ROSE Compiler user manual. http ://rosecompiler.org/ROSE_UserManual/ROSE-UserManual.pdf. 69

[23] ROSE Compiler web site. http ://rosecompiler.org/. 68

[24] Sage application - asci benchmarks. https ://asc.llnl.gov/computing_resources/purple/archive/benchmarks. 16

[25] The SGIgrid project. http ://www.wcss.wroc.pl/pb/sgigrid/en/index.php. 26

[26] SUIF project website. http ://suif.stanford.edu. 53

[27] Sami Achour, Meher Ammar, Boubaker Khmili, and Wahid Nasri. Mpi-perf-sim : Towards an automatic performance prediction tool of mpi programs on hierarchical clusters. In *Proceedings of the 2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing*, PDP '11, pages 207–211, Washington, DC, USA, 2011. IEEE Computer Society. 41

[28] Vikram S. Adve, Rajive Bagrodia, James C. Browne, Ewa Deelman, Aditya Dube, Elias N. Houstis, John R. Rice, Rizos Sakellariou, David J. Sundaram-Stukel, Patricia J. Teller, and Mary K. Vernon. POEMS : End-to-end performance design of large parallel adaptive computational systems. *IEEE Transactions on Software Engineering*, 26 :1027–1048, 2000. 36

[29] Anant Agarwal, Ricardo Bianchini, David Chaiken, David Kranz, John Kubiatowicz, Beng hong Lim, Kenneth Mackenzie, and Donald Yeung. The mit alewife machine : Architecture and performance. In *In Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 2–13, 1995. 15

[30] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. Loggp : Incorporating long messages into the logp model — one step closer towards a realistic model for parallel computation. Technical report, Santa Barbara, CA, USA, 1995. 14

[31] R.J. Allan. Survey of hpc performance modelling and prediction tools. 2009. 12

[32] Uwe Aßmann and Andreas Ludwig. Introducing connections into classes with static meta-programming. In *3rd Int. Conf. on Coordination, number 1594 in LNCS*. Springer, 1999. 38

[33] Rosa M. Badia, Francesc Escalé, Edgar Gabriel, Judit Gimenez, Rainer Keller, Jesús Labarta, and Matthias S. Müller. Performance prediction in a grid environment. In *Grid Computing*, volume 2970 of *Lecture Notes in Computer Science*, pages 257–264. Springer Berlin / Heidelberg, 2004. 33

[34] Rosa M. Badia, Jesus Labarta, Judit Giménez, and Francesc Escalé. Dimemas : Predicting mpi applications behaviour in grid environments, June 2003. 33

[35] Earth Science Division Berkeley Lab. Esd technology listing. http ://esd.lbl.gov/research/tech_transfer/. 15

[36] François Bodin, Peter Beckman, Dennis Gannon, Jacob Gotwals, Srinivas Narayana, Suresh Srinivas, and Beata Winnicka. Sage++ : An object-oriented toolkit and class library for building Fortran and C++ restructuring tools. In *OON-SKI '94 : The 2nd annual object-oriented numerics conference*, pages 122–136, 1994. 69

[37] Julien Bourgeois. Prédiction de performance statique et semi-statique dans les systèmes répartis hétérogènes, 2000. PhD thesis, LIFC computer science laboratory, University of Franche-Comté, France. 34

[38] Julien Bourgeois and François Spies. Performance prediction of an NAS benchmark program with ChronosMix environment. In *Euro-Par '00 : Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pages 208–216, London, UK, 2000. Springer-Verlag. 34, 69, 85, 91

[39] Julien Bourgeois, Vaidy Sunderam, Jaroslaw Slawinski, and Bogdan Cornea. Extending executability of applications on varied target platforms. In *HPCC'11, 13-th IEEE International Conference on High Performance Computing and Communications*. IEEE Computer Society, September 2011. 100

[40] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid : A generic framework for large-scale distributed experiments. In *UKSIM '08 : Proceedings of the 10th International Conference on Computer Modeling and Simulation*, pages 126–131, Washington, DC, USA, 2008. IEEE Computer Society. 42, 75

[41] Pierre-Nicolas Clauss, Mark Stillwell, Stéphane Genaud, Frédéric Suter, Henri Casanova, and Martin Quinson. Single Node On-Line Simulation of MPI Applications with SMPI. In *IPDPS'11 : International Parallel & Distributed Processing Symposium*, Anchorange (AK), United States, May 2011. IEEE. 42, 106, 140

[42] Bogdan Florin Cornea. Thesis : online resources. http ://bogdan.cornea.perso.neuf.fr/files/thesis. 75, 92, 98

[43] Bogdan Florin Cornea and Julien Bourgeois. Performance prediction of distributed applications using block benchmarking methods. In *PDP'11, 19-th Int. Euromicro Conf. on Parallel, Distributed and Network-Based Processing*, Ayia Napa, Cyprus, 2011. IEEE Computer Society. 80, 85, 91, 93, 99

[44] Bogdan Florin Cornea, Julien Bourgeois, The Tung Nguyen, and Didier El-Baz. Performance prediction in a decentralized environment for peer-to-peer computing. In *HotP2P'11, 8-th International Workshop on Hot Topics in Peer-to-Peer Systems, IEEE IPDPS Workshops*. IEEE Computer Society, May 2011. 86, 87, 99, 134

[45] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP : towards a realistic model of parallel computation. *Proceedings of the 4th ACM SIGPLAN symposium on Principles and practice of parallel programming*, 28(7) :1–12, 1993. 13, 14

[46] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. Logp : towards a realistic model of parallel computation. In *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPOPP'93, pages 1–12, New York, NY, USA, 1993. ACM. 36

[47] Eugen Dedu and Emmanuel Lochin. A study on the benefit of TCP packet prioritisation. In *PDP'09 : Proc. of the 17th Euromicro Conf. on Parallel, Distributed and Network-based Processing*, pages 161–166, Weimar, Germany, 2009. IEEE Computer Society. 132

[48] Luiz DeRose. The HPM Toolkit. http ://www.alphaworks.ibm.com/tech/hpmtoolkit. 49

[49] Didier El Baz and The Tung Nguyen. A self-adaptive communication protocol with application to high performance peer to peer distributed computing. In *PDP '10 : Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pages 327–333, Washington, DC, USA, 2010. IEEE Computer Society. 3, 83, 86, 126

[50] Jean-Baptiste Ernst-Desmulier, Julien Bourgeois, Minh Thanh Ngo, François Spies, and Jérome Verbeke. Simulating and optimizing a peer-to-peer computing framework. In *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, pages 376–376, Washington, DC, USA, 2006. IEEE Computer Society. 38, 83, 85, 140

[51] Jean-Baptiste Ernst-Desmulier, Julien Bourgeois, and François Spies. P2pperf : a framework for simulating and optimizing peer-to-peer-distributed computing applications. *Concurrency and Computation : Practice & Experience*, 20(6) :693–712, 2008. 38, 83, 85, 132, 140

[52] Marcio Faerman, Alan Su, Richard Wolski, and Francine Berman. Adaptive performance prediction for distributed data-intensive applications. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, SC'99, New York, NY, USA, 1999. ACM. 23

[53] Thomas Fahringer. On estimating the useful work distribution of parallel programs under the P3T : A static performance estimator. *Concurrency, Practice and Experience (Ed. Geoffrey Fox)*, 8 :28–2, 1996. 21

[54] Thomas Fahringer and Hans P. Zima. A static parameter based performance prediction tool for parallel programs. In *ICS'93 : Proceedings of the 7th international conference on Supercomputing*, pages 207–219. ACM, 1993. 21

[55] Steven A. Finney. Real-time data collection in Linux : A case study. *Behavior Research Methods, Instruments, and Computers*, 33 :167–173, 2001. 65

[56] Matthew I. Frank, Anant Agarwal, and Mary K. Vernon. Lopc : Modeling contention in parallel algorithms, 1997. 36

[57] Marc-Eduard Frincu, Martin Quinson, and Frédéric Suter. Handling Very Large Platforms with the New SimGrid Platform Description Formalism. Technical Report RT-0348, INRIA, 2008. 111

[58] Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. The scalasca performance toolset architecture. *Concurr. Comput. : Pract. Exper.*, 22 :702–719, April 2010. 51

[59] GNU. Gnu c and c++ compiler optimization options. http ://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html. 88

[60] Maxim Goldin. Profiling HPC applications. http ://msdn.microsoft.com/en-us/library/ff678493(v=VS.100).aspx, May 2010. 52

[61] Duncan Grove and Paul Coddington. Precise mpi performance measurement using mpibench. In *Proceedings of HPC Asia*, 2001. 24

[62] Duncan A. Grove. Performance modelling of message-passing parallel programs, 2003. PhD thesis, Department of Computer Science, University of Adelaide, Australia. 24

[63] S. D. Hammond, G. R. Mudalige, J. A. Smith, S. A. Jarvis, J. A. Herdman, and A. Vadgama. Warpp : a toolkit for simulating high-performance parallel scientific codes. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Simutools'09, pages 19 :1–19 :10, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 40

[64] Intel. Intel c++ compiler optimization options. http ://a-math.colorado.edu/computing/software/man/icc.html. 88

[65] Guillaume Jourjon and Didier El Baz. Some solutions for peer-to-peer global computing. In *PDP '05 : Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 49–58, Washington, DC, USA, 2005. IEEE Computer Society. 83

[66] D. J. Kerbyson, Al H. J., A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings. Predictive performance and scalability modeling of a large-scale application. In *SC'01 : Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '01, pages 37–37, New York, NY, USA, 2001. ACM. 16

[67] Rafiqul Zaman Khan, Abdul Quaiyum Ansari, and Kalim Qureshi. Performance prediction for parallel scientific applications. *Malaysian Journal of Computer Science*, 17(1) :65–73, 2004. 24

[68] Milind V. Kulkarni and Daniel J. Quinlan. Ast compression for large scale projects. https ://iscr.llnl.gov/annual_report/fy2005/summer/students/kulkarni.html, 2005. Institute for Scientific Computing Research Annual Report. 71

[69] Krzysztof Kurowski, Ariel Oleksiak, Jarek Nabrzyski, Agnieszka Kwiecien, Marcin Wojtkiewicz, Maciej Dyczkowski, Francesc Guim, Julita Corbalan, Jesus Labarta, and Poznan Supercomputing. Multi-criteria grid resource management using performance prediction techniques. In *Proceeding of the CoreGrid Integration Workshop*, 2005. 26

[70] Krzysztof Kurowski, Ariel Oleksiak, Jarek Nabrzyski, Agnieszka Kwiecien, Marcin Wojtkiewicz, Maciej Dyczkowski, Francesc Guim, Julita Corbalan, Jesus Labarta, and Poznan Supercomputing. Multi-criteria grid resource management using performance prediction techniques. In *Proceeding of the CoreGrid Integration Workshop*, 2005. 26

[71] Jesús Labarta, Sergi Girona, Vincent Pillet, Toni Cortes, and Luis Gregoris. Dip : A parallel program development environment. In *Euro-Par, Vol. II'96*, pages 665–674, 1996. 33

[72] Sebastien Linck, Eugen Dedu, and Francois Spies. Distance-dependent RED policy (DDRED). In *International Conf. on Networking (ICN)*, pages 1–6, Sainte-Luce, Martinique, 2007. IEEE. 132

[73] Niel K. Madsen. Divergence preserving discrete surface integral methods for maxwell's curl equations using non-orthogonal unstructured grids. *J. Comput. Phys.*, 119 :34–45, June 1995. 15

[74] John Mellor-Crummey. Hpctoolkit : Multi-platform tools for profile-based performance analysis. *APART'03 : 5th International Workshop on Automatic Performance Analysis*, November 2003. 48

[75] Barton P. Miller, Mark D. Callaghan, Jonathan M. Cargille, Jeffrey K. Hollingsworth, R. Bruce Irvin, Karen L. Karavanic, Krishna Kunchithapadam, and Tia Newhall. The Paradyn parallel performance measurement tool. *IEEE Computer*, 28 :37–46, November 1995. 47

[76] Tran Ngoc Minh and Lex Wolters. Using historical data to predict application runtimes on backfilling parallel systems. In *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, PDP'10, pages 246–252, Washington, DC, USA, 2010. IEEE Computer Society. 27

[77] Csaba Andras Moritz and Matthew I. Frank. Logpc : modeling network contention in message-passing programs. In *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '98/PERFORMANCE '98, pages 254–263, New York, NY, USA, 1998. ACM. 15

[78] Philip J. Mucci, Shirley Browne, Christine Deane, and George Ho. Papi : A portable interface to hardware performance counters. In *In Proceedings of the Department of Defense HPCMP Users Group Conference*, pages 7–10, 1999. 66

[79] The Tung Nguyen. *Un environnement pour le calcul intensif pair à pair*. PhD thesis, Institut National Polytechnique de Toulouse (INP Toulouse), 2011. 134

[80] The Tung Nguyen, D. El Baz, P. Spiteri, G. Jourjon, and Ming Chau. High performance peer-to-peer distributed computing with application to obstacle problem. In *IPDPSW '10 : IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pages 1 –8, april 2010. 83, 86, 87, 126, 134

[81] Michael Noeth, Jaydeep Marathe, Frank Mueller, Martin Schulz, and Bronis de Supinski. Scalable compression and replay of communication traces in massively parallel environments. In *SC'06 : Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 144. ACM, 2006. 51

[82] M.G. Norman, G. Chochia, P. Thanisch, and Emmanuel Issman. Predicting the performance of the diamond dag computation, 1992. 15

[83] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox. Pace–a toolset for the performance prediction of parallel and distributed systems. *Int. J. High Perform. Comput. Appl.*, 14 :228–251, August 2000. 35

[84] Technical University of Dresden. The Open Trace Format project. http ://www.tu-dresden.de/zih/otf. 34, 48, 49

[85] Technical University of Dresden. The Vampir project. http ://www.tu-dresden.de/zih/vampir. 48, 49

[86] Technical University of Dresden. The VampirTrace project. www.tu-dresden.de/zih/vampirtrace. 33, 49

[87] University of Wisconsin (USA). The DynInst api, part of the paradyn project. http ://www.paradyn.org/html/dyninst-history.html. 47

[88] S. O Olabiyisi, E. O Omidiora, E. F. M. Uzoka, M. Victor, and B. A Akinnuwesi. A Survey of Performance Evaluation Models for Distributed Software System Architecture. *WCECS 2010 : Proceedings of the World Congress on Engineering and Computer Science*, I(october), 2010. 12

[89] Mikael Pettersson. Perfctr project webpage. http ://user.it.uu.se/~mikpe/linux/perfctr/. 64

[90] Sabri Pllana, Ivona Brandic, and Siegfried Benkner. Performance modeling and prediction of parallel and distributed computing systems : A survey of the state of the art. In *Proceedings of the First International Conference on Complex, Intelligent and Software Intensive Systems*, pages 279–284, Washington, DC, USA, 2007. IEEE Computer Society. 12

[91] Sundeep Prakash and Rajive L. Bagrodia. MPI-SIM : using parallel simulation to evaluate mpi programs. In *WSC'98 : Proceedings of the 30th conference on Winter simulation*, pages 467–474, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press. 34

[92] Dan Quinlan, Richard Vuduc, Thomas Panas, Jochen Härdtlein, and Andreas Sæbjørnsen. Support for whole-program analysis and the verification of the one-definition rule in C++. In *Proceedings of the Static Analysis Summit*, volume 500-262 of *NIST Special Publication*, pages 27–35, Gaithersburg, MD, USA, 2006. 71

[93] Martin Quinson. Découverte automatique des caractéristiques et capacités d'une plate-forme de calcul distribué, 2003. PhD thesis, École normale supérieure de Lyon (ENS Lyon), France. 23

[94] Matei Ripeanu, Adriana Iamnitchi, and IanFoster. Cactus application : Performance predictions in grid environments. In *EuroPar'01 : Proceedings of European Conference on Parallel Computing*, 2001. 15, 16

[95] Rafael H. Saavedra and Alan J. Smith. Analysis of benchmark characteristics and benchmark performance prediction. *ACM Transactions on Computer Systems*, 14(4) :344–384, 1996. 22, 34

[96] Markus Schordan and Dan Quinlan. A source-to-source architecture for user-defined optimizations. In *Modular Programming Languages*, volume 2789 of *Lecture Notes in Computer Science*, pages 214–223. Springer Berlin / Heidelberg, 2003. 53, 68

[97] Sameer S. Shende and Allen D. Malony. The TAU parallel performance system. *The International Journal of High Performance Computing Applications*, 20 :287–331, 2006. 48

[98] Jens Simon and Jens michael Wierum. Accurate performance prediction for massively parallel systems and its applications. In *European Conference on Parallel Processing*, pages 675–688, 1996. 14

[99] Pierre Spitéri and Ming Chau. Parallel asynchronous richardson method for the solution of obstacle problem. In *Proceedings of the 16th Annual International Symposium on High Performance Computing Systems and Applications*, HPCS'02, pages 133–, Washington, DC, USA, 2002. IEEE Computer Society. 126

[100] Corina Stratan and Valentin Cristea. A framework for performance prediction in distributed systems. *U.P.B. Scientific Bulletin, Series C*, 71(3), 2009. 26

[101] David Sundaram-Stukel and Mary K. Vernon. Predictive analysis of a wavefront application using loggp. In *In Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 141–150, 1999. 36

[102] Andrew Sunderland and Andrew Porter. Profiling parallel performance using vampir and paraver. Technical report. 49, 50

[103] Ryutaro Susukita, Hisashige Ando, Mutsumi Aoyagi, Hiroaki Honda, Yuichi Inadomi, Koji Inoue, Shigeru Ishizuki, Yasunori Kimura, Hidemi Komatsu, Motoyoshi Kurokawa, Kazuaki J. Murakami, Hidetomo Shibamura, Shuji Yamamura, and Yunqing Yu. Performance prediction of large-scale parallell system and application using macro-level simulation. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC'08, pages 20 :1–20 :9, Piscataway, NJ, USA, 2008. IEEE Press. 38, 39

[104] Agnieszka Szymańska-Kwiecień, Jan Kwiatkowski, Marcin Pawlik, and Dariusz Konieczny. Performance Prediction Methods. *PIPS '06 : Proceedings of the International Multiconference on Computer Science and Information Technology*, pages 363 – 370, 2006. 12

[105] Nathan Tallent, John Mellor-Crummey, Laksono Adhianto, Mike Fagan, and Mark Krentel. Hpctoolkit : performance tools for scientific computing. *Journal of Physics : Conference Series*, 125(1), 2008. 48

[106] Nathan R. Tallent. Hpctoolkit : Top-down analysis of node performance. *MCS Divisional Seminars and Colloquia*, August 2003. 48

[107] Mustafa M. Tikir, Michael A. Laurenzano, Laura Carrington, and Allan Snavely. Psins : An open source event tracer and execution simulator for mpi applications. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, Euro-Par '09, pages 135–148, Berlin, Heidelberg, 2009. Springer-Verlag. 39

[108] Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean christophe Hugly, Eric Pouyoul, and Bill Yeager. Project jxta 2.0 super-peer virtual network. Technical report, Sun Microsystems Inc., May 2003. 38

[109] Tsinghua University. The SIM-MPI simulator. http ://www.hpctest.org.cn/resources/sim-mpi.tgz. 40

[110] Arjan J. C. van Gemund. Performance prediction of parallel processing systems : The pamela methodology. In *SC'93 : International Conference on Supercomputing*, pages 318–327, 1993. 17

[111] Arjan J. C. van Gemund. Symbolic performance modeling of parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(2) :154–165, 2003. 16

[112] Jerome Verbeke, Neelakanth Nadgir, Greg Ruetsch, and Ilya Sharapov. Framework for peer-to-peer distributed computing in a heterogeneous, decentralized environment. In *In Proceedings of the 3rd International Workshop on Grid Computing*, pages 1–12. Springer-Verlag, January 2002. 38

[113] Jeffrey S. Vetter and Michael O. McCracken. Statistical scalability analysis of communication operations in distributed applications. In *Proceedings of the 8th ACM SIGPLAN symposium on Principles and practices of parallel programming*, PPoPP'01, pages 123–132, New York, NY, USA, 2001. ACM. 50

[114] Jérôme Vienne. Page au Laboratoire d'Informatique de Grenoble. http ://mescal.imag.fr/membres/jerome.vienne/Webpage/Webpage.html. 92, 97

[115] Jérôme Vienne. *Prédiction de Performances d'Applications de Calcul Haute Performances sur Réseau Infiniband.* PhD thesis, Université Joseph Fourier de Grenoble, Laboratoire d'Informatique de Grenoble, 2010. http ://mescal.imag.fr/membres/jean-marc.vincent/JMV-homepage/PhD/Vienne.pdf. 88, 97

[116] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active messages : a mechanism for integrated communication and computation. *SIGARCH Comput. Archit. News*, 20 :256–266, April 1992. 15

[117] Terry L. Wilmarth, Gengbin Zheng, Eric J. Bohm, Yogesh Mehta, Nilesh Choudhury, Praveen Jagadishprasad, and Laxmikant V. Kale. Performance prediction using simulation of large-scale interconnection networks in pose. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, PADS'05, pages 109–118, Washington, DC, USA, 2005. IEEE Computer Society. 37

[118] Rich Wolski, Neil T. Spring, and Jim Hayes. The network weather service : A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 15 :757–768, 1999. 22

[119] Alvaro Wong, Dolores Rexachs, and Emilio Luque. Parallel application signature. *Cluster'09 : IEEE International Conference on Cluster Computing and Workshops, 2009*, pages 1–4, September 2009. poster. 27

[120] Alvaro Wong, Dolores Rexachs, and Emilio Luque. Extraction of parallel application signatures for performance prediction. In *HPCC'10 : 10th IEEE International Conference on High Performance Computing and Communications*, pages 223–230, 2010. 27

[121] Alvaro Wong, Dolores Rexachs, and Emilio Luque. Parallel application signature for performance prediction. In *PDPTA'10 : International Conference on Parallel and Distributed Processing Techniques and Applications*, volume 2. CSREA Press, 2010. 27

[122] Alvaro Wong, Dolores Rexachs, and Emilio Luque. PAS2P tool, parallel application signature for performance prediction. In *Para 2010 : State of the Art in Scientifc and Parallel Computing,*, 2010. 27

[123] Xing Wu, Frank Mueller, and Scott Pakin. Automatic generation of executable communication specifications from parallel applications. In *IPDPS'11 : 25th IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2011. 28

[124] Leo T. Yang, Xiaosong Ma, and Frank Mueller. Cross-platform performance prediction of parallel applications using partial execution. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC'05, pages 40–, Washington, DC, USA, 2005. IEEE Computer Society. 25

[125] Dmitrijs Zaparanuks, Milan Jovic, and Matthias Hauswirth. Accuracy of performance counter measurements. In *ISPASS 2009 : IEEE International Symposium on Performance Analysis of Systems and Software*, pages 23–32, Boston, MA, USA, 2009. 64, 67

[126] Jidong Zhai, Wenguang Chen, and Weimin 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. ACM, 2010. 40

[127] Jidong Zhai, Tianwei Sheng, Jiangzhou He, Wenguang Chen, and Weimin Zheng. Fact : fast communication trace collection for parallel applications through program slicing. In *SC'09 : Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, USA, 2009. ACM. 40, 41

[128] Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant V. Kalé. Bigsim : A parallel simulator for performance prediction of extremely large parallel machines. In *IPDPS'04*, 2004. 37