Contributions à la performance du calcul scientifique et embarqué

Fabien Coelho

Université Paris Diderot – Paris 7

Contributions à la performance du calcul scientifique et embarqué

Mémoire présenté en vue de l'obtention du diplôme
d'habilitation à diriger des recherches

par

Fabien Coelho

le vendredi 11 octobre 2013, devant le jury composé de :

- Carole Delporte, Prof. Université Paris Diderot (Paris 7), rapporteur
- Philippe Clauss, Prof. Université de Strasbourg, rapporteur
- Christian Lengauer, Prof. Universität Passau (Allemagne), rapporteur
- Cédric Bastoul, Prof. Université de Strasbourg, examinateur
- Albert Cohen, DR INRIA, examinateur et président du jury
- François Irigoin, DR MINES ParisTech, examinateur

Rapport A/560/CRI (référence **hal-01254385**)
Centre de recherche en informatique, MINES ParisTech, PSL Research University,
35, rue Saint-Honoré, 77305 Fontainebleau cedex, France.
Version 692 du 2016-01-13

Résumé

Ce document résume mes résultats de recherche après vingt années d'activité, y compris les travaux en collaboration avec trois étudiants ayant soutenu leur thèse de doctorat : Julien Zory, Youcef Bouchebaba et Mehdi Amini. Il fournit une vue générale des résultats organisés selon les principales motivations qui ont soutenu leur développement, à savoir la performance, l'élégance, les expériences et la diffusion des connaissances. Ces travaux couvrent l'analyse statique pour la compilation ou la détection de problèmes dans les programmes, la génération de communications avec des méthodes polyédriques, la génération de code pour des accélérateurs matériels, mais aussi des primitives cryptographiques et un algorithme distribué. Ce document ne donne cependant pas une présentation détaillée des résultats, pour laquelle nous orientons le lecteur vers les articles de journaux, de conférences ou de séminaires correspondants. Les quatre thèmes abordés sont : la **performance** – l'essentiel des travaux vise à optimiser les performances de codes sur diverses architectures, du super calculateur à mémoire distribuée, à la carte graphique (GPGPU), jusqu'au système embarqué spécialisé ; l'**élégance** – est un objectif des phases de conception, de même que trouver si possible des solutions optimales, tout en devant rester pratiques ; les **expériences** – la plupart des algorithmes présentés sont implémentés, en général dans des logiciels libres, par exemple intégrés à des gros projets comme le logiciel PIPS ou diffusés de manière indépendante, de manière à conduire des expériences qui montrent l'intérêt pratique des méthodes ; les **connaissances** – une large part de de mon activité est dédiée à la transmission des connaissances, pour des étudiants, des professionnels ou même le grand public. Le document se conclut par un projet de recherche présenté sous la forme d'une discussion et d'un ensemble de sujets de stage ou de thèse.

Abstract

This document summarizes my research achievements in twenty years of activity, including works with three students who have defended their PhD: Julien Zory, Youcef Bouchebaba and Mehdi Amini. It provides a comprehensive overview of the results organised along the main drivers behind their development, namely performance, elegance, experiments and knowledge. It covers static analyses for compiling or reporting programming issues, advanced code transformations, generation of communications based on polyhedral techniques, code generation for hardware accelerators, as well as cryptographic primitives and a distributed algorithm. However, it does not aim at providing a detailed presentation of the subjects covered, for which we refer to the corresponding journal, conference or workshop papers. The four themes of these documents are: **performance** – most of the work aims at optimizing the performance of codes on various architectures, from distributed memory super computers to general purpose graphical processing units (GPGPU) or to small special purpose embedded systems; **elegance** – care is taken about design issues and to find optimal solutions when possible, although the proposed methods must also be practical; **experiments** – most of the algorithms are implemented, usually in open source software, either integrated in large projects such as PIPS or distributed on their own, so as to conduct extensive experiments which demonstrate their practical usefulness. **knowledge** – a large part of my activity is dedicated to transmitting knowledge, from students to professionals or even the general public. The document is concluded by a research project presented as a discussion and a set of subjects for PhD thesis or internships.

Remerciements

Je tiens tout d'abord à remercier les membres du jury, en particulier les professeurs Carole Delporte, Philippe Clauss et Christian Lengauer, qui m'ont fait l'honneur d'accepter de rapporter sur ce mémoire, et ensuite les examinateurs de la soutenance, le professeur Cédric Bastoul, et les directeurs de recherche Albert Cohen et François Irigoin. Je tiens également à remercier vivement le professeur Olivier Carton qui a accueilli ma demande d'inscription en HDR auprès de l'UFR informatique de l'Université Paris Diderot.

Je tiens ensuite à remercier mes collègues actuels et passés du CRI à l'École des mines de Paris, qui m'ont toujours soutenus depuis des années, en particulier François Irigoin déjà cité, Corinne Ancourt, Pierre Jouvelot, Claire Medrala, Robert Mahl et Jacqueline Altimira. Je remercie également les étudiants dont j'ai participé ou participe à l'encadrement des thèses : par ordre chronologique Julien Zory, Youcef Bouchebaba, Mehdi Amini, Amira Mensi et Karel De Vogeleer.

Je remercie enfin ma famille qui me supporte à l'année : Vanina mon épouse, Justine ma fille, et mes parents.

Table des matières

Résumé iii

Abstract iii

Remerciements v

Table des matières viii

Table des figures ix

1 Introduction 1
- 1.1 Brève Chronologie . . . 1
  - 1.1.1 Stages 1992-1993 . . . 1
  - 1.1.2 Thèse 1993-1996 . . . 2
  - 1.1.3 Post-doc et thèse de Julien Zory 1996-1999 . . . 2
  - 1.1.4 Visite IMEC 1999, thèse Youcef Bouchebaba 1999-2002 . . . 2
  - 1.1.5 Enseignements et ingénierie 1998-2012 . . . 2
  - 1.1.6 Reprise d'une recherche plus active depuis 2008 . . . 3
- 1.2 Principes conducteurs . . . 3

2 Performance 5
- 2.1 Macro-parallélisme . . . 5
  - 2.1.1 Contexte historique . . . 6
  - 2.1.2 Présentation de HPF . . . 6
  - 2.1.3 Compilation de HPF . . . 9
  - 2.1.4 Modèle affine pour entrées-sorties et replacements . . . 11
  - 2.1.5 Optimisation des communications . . . 14
- 2.2 Micro parallélisme . . . 14
  - 2.2.1 Onde24, une application simple . . . 16
  - 2.2.2 Optimisations des expressions . . . 16
  - 2.2.3 Réduction énergétique . . . 23
  - 2.2.4 Traitement d'image embarqué . . . 25
- 2.3 Autres optimisations . . . 26
  - 2.3.1 Manipulations polyédriques . . . 26
  - 2.3.2 Preuve d'effort . . . 27
  - 2.3.3 Comparaison de relations à distance . . . 30
- 2.4 Conclusion . . . 30

3 Élégance 33
- 3.1 HPF grandeurs et décadences . . . 33
  - 3.1.1 Retours sur la conception de HPF . . . 34
  - 3.1.2 Analyse de l'échec de HPF . . . 36
- 3.2 Stratégies de compilation . . . 38
  - 3.2.1 Compilation FREIA . . . 38
  - 3.2.2 Discussion . . . 42

3.3 Pragmatisme . . . 43
3.3.1 Affronter une combinatoire . . . 43
3.3.2 Adapter l'application au code cible . . . 45
3.3.3 Co-conception langage, compilateur, exécutif et matériel . . . 46
3.3.4 Maximiser les défauts de cache . . . 47
3.3.5 Conseils relationnels . . . 48
3.4 Évidences . . . 48
3.4.1 Cadre général . . . 48
3.4.2 Preuves . . . 49
3.5 Conclusion . . . 51

4 Expériences **53**
4.1 Implémentations . . . 53
4.1.1 Newgen . . . 54
4.1.2 Linear/C3 . . . 55
4.1.3 PIPS . . . 56
4.2 Expérimentations . . . 57
4.2.1 Quelques exemples . . . 58
4.2.2 Conséquences . . . 58
4.3 Conclusion . . . 59

5 Connaissances **61**
5.1 Séminaires et tutoriels . . . 61
5.1.1 Séminaires de recherche . . . 62
5.1.2 Grand public . . . 62
5.1.3 Tutoriel logiciels libres . . . 62
5.2 Enseignement . . . 62
5.3 Outils pédagogiques . . . 63
5.4 Contributions aux logiciels libres . . . 64

6 Projet de Recherche **67**
6.1 Contexte et enjeux . . . 67
6.1.1 Compilation . . . 67
6.1.2 Contexte technologique . . . 68
6.1.3 Le matériel . . . 68
6.1.4 Enjeux . . . 70
6.1.5 Méthodologie . . . 70
6.2 Sujets de recherche . . . 71
6.2.1 Sujets de thèse . . . 71
6.2.2 Sujets de master recherche et ingénieur . . . 72
6.3 Conclusion . . . 75

Postface **77**

Bibliographie **84**

Table des figures

2.1 Boucles parallèles de HPF . . . 7
2.2 Modèle de placement des données en HPF . . . 8
2.3 Possibilités multiples de l'alignement . . . 8
2.4 Possibilités multiples de la distribution . . . 9
2.5 Programme `triangle` . . . 10
2.6 Système linéaire pour le programme `triangle` . . . 10
2.7 Code SPMD des nœuds pour `triangle` . . . 11
2.8 Code de l'hôte pour `triangle` . . . 11
2.9 Exemple de replacement HPF, illustré à la figure 2.10 . . . 12
2.10 Replacement du tableau `A` dans le code de la figure 2.9 . . . 13
2.11 Équations du replacement . . . 13
2.12 Traduction des replacements (à gauche) en simples copies (à droite) . . . 15
2.13 Onde24 – Propagation d'ondes 2D . . . 17
2.14 Onde24 – Exemple d'enregistrement par un sismographe . . . 17
2.15 Onde24 – Calculs et zones . . . 18
2.16 Onde24 – différence finie au cœur du domaine . . . 18
2.17 Onde24 – Choix des factorisations conduisant aux deux expressions de la figure 2.18 . . . 20
2.18 Onde24 – les deux factorisations possibles de par l'alternative de la figure 2.17 . . . 21
2.19 Optimisation incrémentale guidée par des heuristiques . . . 21
2.20 Onde24 – quelques choix de multiplications-additions et Mflop/s sur P2SC 595 . . . 22
2.21 Application Cavity . . . 24
2.22 Exemple avec deux tableaux... . . . 25
2.23 Allocation d'un tableau circulaire . . . 25
2.24 Données vivantes après un pavage . . . 25
2.25 ANR999 – une application utilisant l'API FREIA . . . 26
2.26 Preuve d'effort par défi et réponse . . . 27
2.27 Preuve d'effort par résolution et vérification . . . 28
2.28 Arbre de Merkle pour preuve d'effort . . . 30
2.29 Construction d'un arbre de hachage sur une relation . . . 30

3.1 Styles de directives interne et externe . . . 35
3.2 Onde24 – Placements HPF . . . 37
3.3 Stratégie de compilation de FREIA . . . 39
3.4 Extrait de ANR999 (figure 2.25) après pré-traitement . . . 40
3.5 DAG initial de l'application ANR999 . . . 40
3.6 DAG optimisé de l'application ANR999 . . . 40
3.7 Placement du DAG de l'application ANR999 sur SPoC . . . 41
3.8 Placement du DAG de l'application ANR999 sur Terapix . . . 41
3.9 Placement du DAG de l'application ANR999 sur OpenCL . . . 41

Chapitre 1

Introduction

Après quelques années d'une carrière d'enseignant-chercheur, il est d'usage de rédiger un document appelé *Habilitation à diriger des recherches* (HDR) :

- la ***recherche*** couvre l'exploration et éventuellement la découverte de vérités ou modes opératoires pour comprendre et résoudre de nouveaux problèmes, ou améliorer les techniques de résolutions existantes ;
- la ***direction*** sous-tend l'idée d'un choix discriminant des problèmes abordés, mais aussi d'une attitude exigeante dans leur traitement scientifique ; la direction peut aussi se comprendre dans le sens de *management*, à savoir la direction de la recherche d'étudiants en doctorat ou d'autres chercheurs ;
- l'***habilité*** ne concerne pas une qualité intrinsèque pourtant souhaitable de l'impétrant, mais plutôt une reconnaissance par ses pairs d'une aptitude à diriger de la recherche telle que définie ci-dessus.

C'est précisément ce document que vous tenez entre vos mains. Sa rédaction est l'occasion de prendre du recul, de s'interroger, de faire le point, pour continuer ou infléchir une trajectoire après des années d'activité. Je reconstitue tout d'abord et brièvement le chemin linéaire parcouru ces dernières années à la section 1.1, puis j'analyse les principes directeurs de mes activités scientifiques à la section 1.2. Ces principes constituent la base du plan pour organiser la présentation globale de mes différents travaux aux chapitres suivants.

1.1 Brève Chronologie

Collégien fasciné par la possibilité de déléguer à des machines des tâches plus ou moins intelligentes, j'abordai l'informatique en suivant la page programmation en BASIC du magazine *Science et Vie* au début des années 1980. J'achetai bientôt mon premier ordinateur en cassant ma tirelire pour un Sinclair ZX81 disposant de 1 Ko de mémoire vive puis une extension mémoire de 16 Ko : affichage sur la télévision familiale, sauvegarde très aléatoire des données sur lecteur de cassette, aucune application mais juste de la programmation. Lycéen, je participai au club informatique local et investissai dans un Amstrad CPC664 : moniteur monochrome bien vert, lecteur de disquette (oui !) 64 Ko de mémoire vive pour servir un processeur Z80. Je me passionnai ensuite pour la programmation Pascal enseignée en classes préparatoires, avant d'atterrir par le hasard des concours à l'École des mines de Paris. Je m'aperçu alors qu'il était possible de transformer cette passion en option impliquant cours spécifiques et stages, et de faire des choses concrètes et techniques dans l'océan de mou des cours qui m'étaient imposés. Les choses sérieuses commençaient.

1.1.1 Stages 1992-1993

L'été 1992 devait être consacré à un stage ingénieur se déroulant de préférence à l'étranger. Grâce aux contacts de Pierre Jouvelot, il était question que mon stage se déroule au MIT. Cette perspective me réjouissait grandement, jusqu'à ce qu'elle ne tombe à l'eau à quelques jours de l'échéance et que je me

retrouve pour l'été dans le bureau de François Irigoin au CRI, à Fontainebleau. Très libre, je m'occupai de choses et d'autres, et suivis en particulier les discussions qui s'ouvraient sur la standardisation d'un Fortran parallèle pour machine à mémoire distribuée, déjà appelé HPF. Ces travaux d'été aboutirent à un rapport de stage [16] mais aussi à la présentation des travaux du *Forum HPF* auprès de la communauté des chercheurs français du domaine dans le cadre du groupe de recherche Paradigme [17].

Je profitai également de la troisième année d'études aux mines pour m'inscrire en DEA Systèmes Informatiques à l'Université Paris 6, en parallèle des cours de mon école, avec la couverture de mon professeur d'option Robert Mahl et le soutien et l'accueil de son adjoint François Irigoin. J'appréciais particulièrement la partie travaux de groupes consistant à lire, analyser et présenter des articles de recherche. Mes travaux se poursuivirent naturellement au cours de mon stage d'option jusqu'en juin [19] et de mon stage de DEA jusqu'en septembre [18]. Ils aboutirent à une première publication à la conférence Euro-Par 1994 [20].

1.1.2 Thèse 1993-1996

À la suite de ces stages assez orientés recherche, la poursuite de ces travaux en thèse sur un sujet ouvert me paraissait naturelle et souhaitable. C'est d'ailleurs pour cela que j'avais entamé un DEA afin d'éviter de perdre une année en suivant cette voie. Dès octobre 1993 je m'inscris en thèse à l'École des mines de Paris, tout en effectuant mon service militaire en tant que scientifique du contingent, ce qui m'offrait l'occasion de faire de la recherche avec une coupe de cheveux très rase offerte par la nation, une solde permettant de couvrir la consommation mensuelle de cigarettes d'un bidasse moyen, et des réductions dans les transports ferroviaires pour rentrer voir sa famille. Mes travaux ont porté sur la compilation du langage HPF, notamment à base de modélisations polyédriques, avec une implémentation au sein du compilateur PIPS *(Paralléliseur Interprocédural de Programmes Scienfiques)*. J'ai soutenu ma thèse [25] début octobre 1996, après exactement trois ans de travaux. Au cours de cette période, j'ai également participé aux réflexions autour de la formalisation de certains travaux de la thèse de Béatrice Creusillet.

1.1.3 Post-doc et thèse de Julien Zory 1996-1999

À la suite de ma soutenance, Julien Zory fut recruté comme doctorant au centre et j'ai particulièrement suivi et encadré ses travaux dont le point de départ était certaines des conclusions ou des problématiques ouvertes par mes propres travaux. En plus de notre collaboration, Julien travailla également avec François Irigoin et Frédéric Desprez. Il soutint sa thèse fin 1999 [87]. Au cours de cette même période j'ai encadré des stages ingénieur et de DEA, en particulier François Didry en 1998 [78] qui a travaillé sur l'interface graphique de PIPS et Pham Dinh Son en 2000 [86] qui a poursuivi l'implémentation de certains travaux de Julien et ajouté une passe de typage explicite des expressions dans PIPS, analyse nécessaire à la manipulations d'expressions arithmétiques.

1.1.4 Visite IMEC 1999, thèse Youcef Bouchebaba 1999-2002

J'ai passé une moitié de l'année 1999 en visite à l'IMEC (Institut de micro-électronique) à Leuven en Belgique (région flamande) dans l'équipe de Francky Catthoor, où j'ai travaillé en particulier avec Thierry Omnès qui soutiendra sa thèse à l'École des mines deux ans plus tard [83]. Cette visite m'amènera aux problématiques de transformations de code pour l'informatique embarquée pour laquelle la puissance électrique disponible est limitée, qui seront abordées dans la thèse de Youcef Bouchebaba [10].

1.1.5 Enseignements et ingénierie 1998-2012

Les années suivantes ont été très lourdement consacrées à l'enseignement, avec en particulier le montage et la gestion quotidienne de plusieurs Mastères Spécialisés sur la période 1998-2010, ainsi que des pointes de charge d'enseignement choisies et acceptées montant néanmoins jusqu'à 450 heures de cours et TD par an. En parallèle, j'ai participé à l'ingénierie logiciel de PIPS, comme l'adaptation à différentes tâches, en particulier pour l'analyse de très gros codes industriels (plus de 100000 lignes) pour lesquels la gestion du volume des données résultant des analyses posait une problématique supplémentaire. J'ai également durant cette période apporté diverses améliorations concernant la fonction de récursion automatique sur les structures de données *Newgen* (outil de génie logiciel qui gère les structures de données

de la représentation intermédiaire de PIPS), ainsi que le passage de PIPS sous *subversion* pour proposer un environnement de développement centralisé et accessible à distance. Mes travaux dans PIPS m'ont également amené à paralléliser et automatiser les tests de non régression de manière à détecter au plus vite des changements liés aux modifications des développeurs. J'ai aussi accompli quelques travaux de recherche liés à mes domaines d'enseignement, comme les bases de données et la cryptographie.

Ces années ont été caractérisées par des collaborations nombreuses avec mes collègues, des doctorants, des élèves, des stagiaires... ce n'est pas un travail isolé qui est présenté ici mais les résultats de travaux souvent communs.

1.1.6 Reprise d'une recherche plus active depuis 2008

J'ai repris une activité plus intense de recherche avec la participation à l'encadrement des doctorants Amira Mensi, Mehdi Amini [2] et Karel De Vogeleer, et au projet ANR FREIA qui a impliqué deux laboratoires de l'École des mines de Paris (morphologie mathématique (CMM) et informatique (CRI)), Télécom Bretagne et l'industriel THALES pour construire un environnement de développement permettant de porter des applications de traitement d'image sur divers accélérateurs matériels.

1.2 Principes conducteurs

À regarder de loin ces différents travaux, il n'est pas évident de leur trouver une cohérence : quoi de commun en effet entre la compilation de HPF et une fonction cryptographique de preuve d'efforts ? Même si les différences, les évolutions sont nombreuses, et c'est heureux – je ne souhaite pas faire toujours la même chose – on retrouve dans tous ces travaux la permanence de quatre principes qui les guident :

Performance chercher à aller plus vite (ou moins vite !), optimiser une solution ;

Élégance trouver l'expression d'un problème plus simple et plus générale ;

Expériences mettre réellement en œuvre les solutions proposées ;

Connaissances partage des connaissances acquises, sous forme de conférences, de cours ou de logiciels.

Ces quatre principes vont servir de plan à cet exposé au cours des prochains chapitres qui nous séparent de la conclusion de ce document. Dans le chapitre 2 consacré à la performance, je présente ma thèse et celles de Julien Zory, Youcef Bouchebaba et Mehdi Amini, ainsi que d'autres travaux motivés par la recherche de solutions si possible optimales. Le chapitre 3, portant sur l'élégance, analyse l'échec relatif du langage HPF, puis les approches ambitieuses de stratégies de compilation globale, j'y poursuis les différentes ruptures de pensées que j'ai essayé de contribuer à mon modeste niveau au cours des travaux engagés avec mes doctorants, avant d'aborder la questions des preuves. Le chapitre 4 se focalise sur les expériences qui ont soutenu tous mes travaux. Ceux-ci impliquent en effet une partie implémentation souvent diffusée sous forme de logiciels libres et une partie étude expérimentale détaillée. Le chapitre 5 discute du partage de connaissances sous la forme de conférences et tutoriels, au cours de nombreux enseignements assurés auprès de divers publics, et dont l'aboutissement a été le développement de nouveaux outils pédagogiques. Enfin, le chapitre 6 décrit mon projet de recherche, essentiellement dans la continuité des travaux présentés ici, c'est à dire visant à tirer le meilleur parti des matériels disponibles pour des domaines d'application.

Je reprends dans ce document les principaux résultats de mes différents travaux, pour en donner à la fois une vue globale et aussi en analyser avec le recul la pertinence, en allant à l'essentiel. Je mets également en évidence les limites des méthodes proposées. Par contre, je ne répète pas dans le détail les résultats scientifiques déjà présentés par ailleurs, mais je cite les articles correspondants pour le lecteur qui souhaiterait plus d'informations.

Chapitre 2

Performance

Résultat optimal qu'une machine peut obtenir.
(dictionnaire Le Petit Robert*)*

Les travaux auxquels j'ai participé ont presque tous pour point commun l'optimisation de codes ou d'algorithmes, souvent dans un contexte de parallélisme matériel, en passant des plus grosses machines disponibles à des machines tenant dans la paume de la main. Ces travaux couvrent en particulier :

- la génération des communications dans le cadre de la compilation du langage HPF ;
- la réduction du temps d'exécution par la manipulation des expressions ;
- la réduction des transferts mémoires qui induisent la consommation des applications embarquées de type traitement du signal ;
- le placement d'applications de traitement d'image sur des accélérateurs matériels particuliers ;
- la *minimisation* des performances atteignables pour les fonctions cryptographiques de preuve d'effort ;
- la comparaison efficace du contenu de tables dans des bases de données distantes.

La section 2.1 discute le macro-parallélisme des super-calculateurs, et de la compilation du langage HPF qui leur est dédié. Ensuite, la section 2.2 discute le micro-parallélisme du processeur et des applications embarquées. La section 2.3 discute nos contributions plus exotiques dans le domaine de la cryptographie et des bases de données, mais là encore avec des aspects performances et parallélisme. La section 2.4 résume ces résultats et ouvre quelques perspectives.

2.1 Macro-parallélisme

Cette section présente des contributions visant l'exploitation du parallélisme gros grain sur machines à mémoire distribuée, faites durant ma thèse ainsi que celle de Julien Zory que j'ai encadrée et avec qui j'ai collaboré. Les résultats restent d'actualité, car les super-calculateurs du calcul scientifique des années 2010, soit 20 ans plus tard, sont toujours des machines à mémoire distribuée, et le même type d'architecture se retrouve dans les petits systèmes embarqués, par exemple le MPPA ManyCore de Kalray créé en 2012.

La section 2.1.1 introduit le contexte de la création du langage HPF au milieu des années 1990. Je présente ensuite le langage à la section 2.1.2, avec d'une part l'expression du parallélisme et d'autre part celui du placement explicite des données. La compilation du langage, en particulier ce qui concerne les communications, est abordée avec une approche affine à la section 2.1.3. Elle est raffinée dans le cas particulier des entrées-sorties et des replacements à la section 2.1.4. L'optimisation des communications, c'est à dire leur placement dans le programme et éventuellement la possibilité de les enlever est abordé à la section 2.1.5. J'analyserai plus spécifiquement les raisons de l'échec du langage dans le chapitre suivant, à la section 3.1.2.

2.1.1 Contexte historique

Dans le domaine du calcul scientifique, il a toujours été très difficile d'extraire des performances pratiques proches des performances théoriques pour lesquelles sont vendus les super-calculateurs. Le passage aux machines vectorielles dans les années 80 a nécessité de lourdes adaptations des codes, qui entravaient éventuellement leur fonctionnement sur stations de travail (RS6000 par exemple) puis sur les premières machines parallèles.

Le calcul haute performance au début des années 1990 est un marché de niche, morcelé entre de trop nombreux acteurs tels Thinking Machine Corporation (TMC), Cray Research, Digital Equipment Corporation (DEC), International Business Machines (IBM) ou Fujitsu.... À ce moment, le modèle d'architecture (parallélisme à grain fin ou gros grain, vectoriel, mémoire partagée ou distribuée) et de programmation associé (passage de message, parallélisme de données, threads) de l'avenir pour la haute performance n'est pas clair du tout[1]. Malgré la monté en performance exponentielle des puissances de calcul des processeurs, qui suit la loi de Moore depuis trois décennies, les besoins en gros calculs sont toujours aussi pressants. Comme moteur de cet appétit insatiable, on peut citer l'arrêt des essais nucléaires décidé par les grandes puissances à la suite du démantèlement du bloc soviétique, qui fait du coup porter le maintien de la dissuasion nucléaire sur les efforts de modélisation. Les super-calculateurs les plus puissants du monde en 2012 sont toujours associés à des centres de recherche nucléaire tels les laboratoires nationaux américains Oak Ridge, Lawrence Livermore et Argonne, ou le CEA français.

Dans ce contexte incertain, et face à la montée en puissance des architectures à mémoire distribuée, les utilisateurs de ces machines, développeurs des applications, hésitent à investir dans des architectures potentiellement non pérennes. La solution proposée est de définir un standard inspiré des différentes extensions proposées à Fortran, le langage de choix du calcul scientifique depuis les années 50, pour le calcul sur machines parallèles à mémoire distribuée. Un forum qui regroupe utilisateurs, fabriquants de machines, fournisseurs de logiciels se base donc sur Fortran D, Vienna Fortran et quelques autres pour décrire un nouveau langage, le High Performance Fortran (HPF).

2.1.2 Présentation de HPF

Le standard HPF est construit sur Fortran 90, et y ajoute essentiellement des directives sous forme de commentaires spéciaux. Je décris ci-après les manières d'exprimer le parallélisme en HPF, puis le placement des données, et enfin les autres aspects.

Parallélisme

Le premier objectif de HPF est de produire de la performance en exploitant le parallélisme de l'application. Le modèle principal est celui du parallélisme de données. Quelques extensions ont aussi été proposées pour le parallélisme de contrôle, mais n'ont pas été retenues.

Une première manière d'exprimer la parallélisme est l'utilisation de notations de tableaux déjà proposées par Fortran 90.

```
! A, B et C sont des matrices
      A = B + C
! V et W sont des vecteurs
      V(1:10) = 2.0*W(21:30)
```

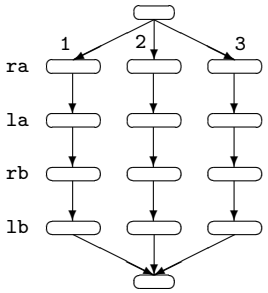
Ces notations ne permettant pas la transposition des dimensions, elles ont été complétées par une construction `FORALL`. Cette nouvelle construction a ensuite été intégrée telle quelle à Fortran 95.

```
! transpose B dans A
      FORALL(i=1:N, j=1:N)
        A(i,j) = B(j,i)
      END FORALL
```

Le parallélisme de données exprimé par ces notations de tableau n'est pas le parallélisme d'une boucle. En effet, la sémantique data-parallèle implique que la partie droite de l'assignation soit entièrement calculée avant d'effectuer l'affectation. On ajoute donc des dépendances nombreuses à l'intérieur même

[1] en fait, on prendra presque tout...

```
!hpf$ independent
      do i = 1, 3
        la(i) = ra(i)
        lb(i) = rb(i)
      end do
```




```
forall(i = 1:3)
  la(i) = ra(i)
  lb(i) = rb(i)
end forall
```

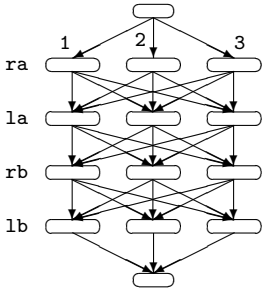


FIGURE 2.1 – Boucles parallèles de HPF

des expressions, revenant à des barrières entre chaque instructions qui induisent des synchronisations, comme montré à la figure 2.1 côté droit pour le FORALL.

Pour proposer une boucle simplement parallèle, où les itérations sont indépendantes les unes des autres comme suggéré à la figure 2.1 côté gauche, une directive INDEPENDENT a été introduite. Elle permet également de déclarer les variables privées aux itérations avec NEW et éventuellement les variables faisant l'objet d'une réduction associative-commutative avec REDUCTION :

```
      s = 0
!HPF$ INDEPENDENT, NEW(x), REDUCTION(s)
      DO I=1, N
         x = A(i)
         s = s + x*x
         A(i) = 1./x
      ENDDO
```

Certaines opérations intrinsèques de Fortran présentent une certaine forme de parallélisme, comme les réductions sur les tableaux exprimées avec les fonctions SUM, MAX...

```
! A est une matrice
      s = SUM(A)
```

Ces fonctions intrinsèques ont été complétées dans HPF avec MAXLOC, MINLOC et quelques autres, ces fonctions supplémentaires étant ensuite intégrées à Fortran 95.

Placement des données

Le second point capital de HPF est constitué des directives de placement des données, qui s'appuie sur un modèle conceptuel affine à deux niveaux représenté à la figure 2.2.

Les quatre directives impliquées sont TEMPLATE, PROCESSORS, ALIGN et DISTRIBUTE : les deux premières permettent de déclarer des tableaux virtuels utilisés comme gabarit pour le placement ou pour décrire les processeurs ; les deux autres précisent le placement lui-même.

Les données des tableaux de l'application sont d'abord *alignées* les unes relativement aux autres, ce qui permet de mettre les éléments en correspondance selon les dépendances des calculs qui seront opérés. Ce premier niveau permet de préciser des transpositions, duplications, écrasements et autres transformations géométriques régulières entre les éléments comme illustré à la figure 2.3. Le rôle du *template* intermédiaire peut être tenu par l'un des tableaux de l'application. Ce template est ensuite *distribué* sur un arrangement de processeurs virtuels, soit par blocs, soit par cycles, soit par cycles de blocs, comme illustré à la figure 2.4. Tous les éléments de tableaux alignés avec un élément de template distribué sur un processeur particulier se retrouvent alors sur ce processeur. Ce modèle génère un placement essentiellement régulier des données, qui convient bien à certains types de calculs comme les différences finies, mais pas à des calculs plus irréguliers comme ceux des méthodes par éléments finis.

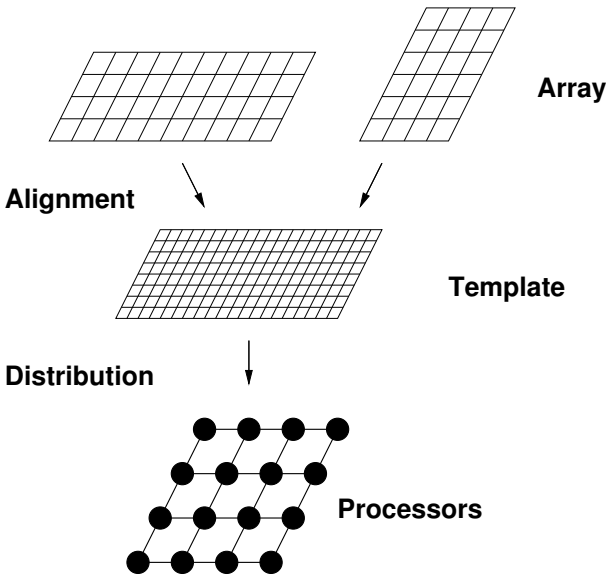


FIGURE 2.2 – Modèle de placement des données en HPF

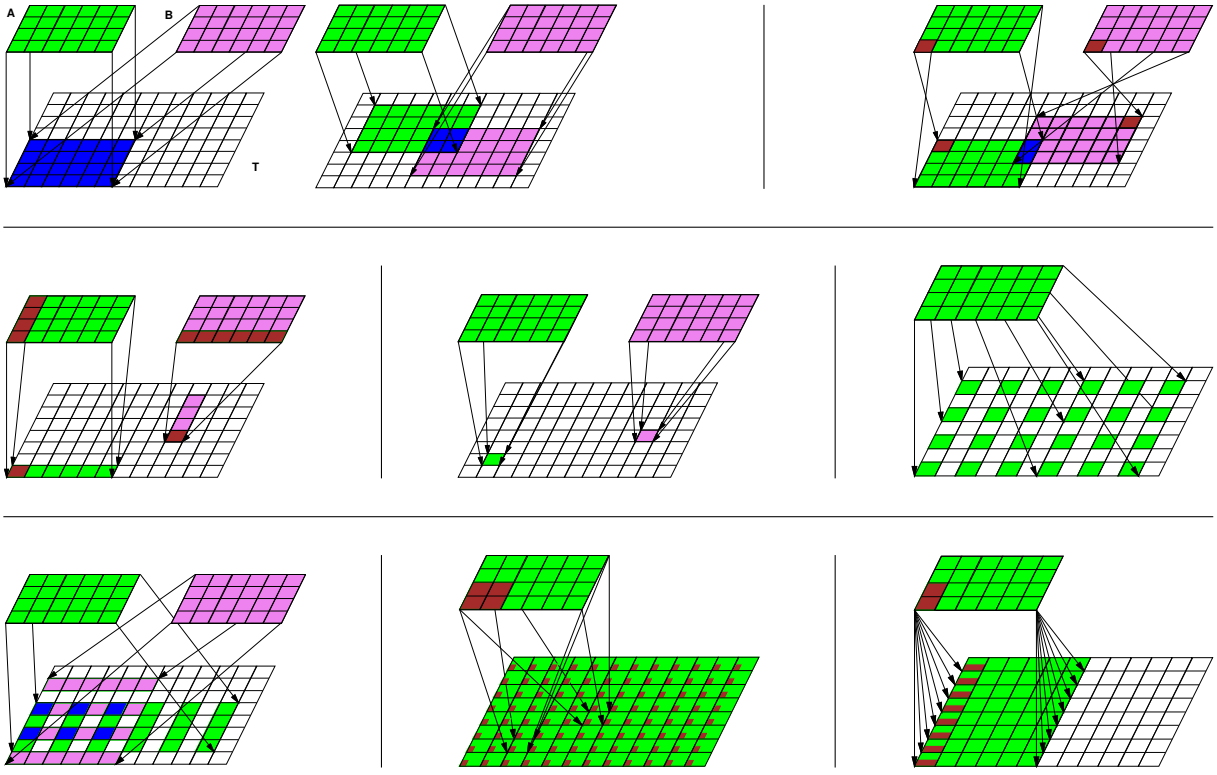


FIGURE 2.3 – Possibilités multiples de l'alignement

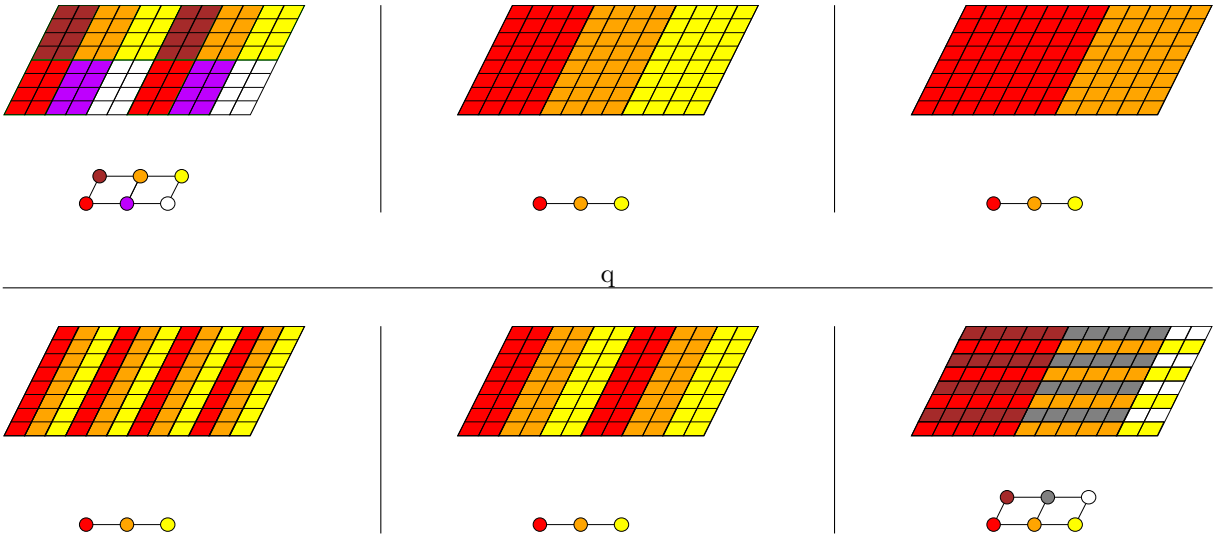


FIGURE 2.4 – Possibilités multiples de la distribution

Le placement peut éventuellement être modifié à l'exécution de l'application avec les directives de placement dynamique `REALIGN` et `REDISTRIBUTE`.

L'utilisateur a aussi la possibilité de déclarer le placement des tableaux en paramètre d'une sous-routine. Ce placement peut être soit descriptif (les arguments seront placés ainsi), soit prescriptif (il faut placer ainsi les arguments), soit hérité (compilateur, débrouille-toi !) avec la directive `INHERIT`. Cette dernière déclaration n'est évidemment d'aucune aide pour le compilateur, qui n'est donc pas en mesure de générer un code efficace dans ce cas, et doit s'appuyer entièrement sur l'exécutif.

Autres aspects de HPF

Le cœur du langage est décrit ci-dessus. En particulier, on notera que rien n'est prévu pour le traitement des entrées-sorties.

Sans précision particulière sur une fonction, il est éventuellement pertinent pour les performances que chaque processeur d'une machine parallèle exécute l'appel, afin d'éviter des communications. Cependant, ce n'est pas du tout une bonne idée pour l'affichage. Des directives supplémentaires ou des analyses seront nécessaires pour prendre de telles décisions.

Des entrées-sorties parallèles posent de plus la question de la vision du système de fichiers sous-jacent distribué sur les nœuds de calculs, et de la manière dont les données écrites pourront être effectivement relues de manière cohérente. Cette problématique étant très proche des matériels disponibles, un consensus a été difficile à trouver.

2.1.3 Compilation de HPF

La première difficulté est de compiler *correctement* ce gros langage dans un contexte d'exécution parallèle à mémoire distribuée : la spécification de Fortran 90 est déjà un long document auquel HPF ajoute quelques centaines de pages. La seconde difficulté est de le compiler *efficacement*, au moins pour une part significative des applications visées, en particulier les plus simples.

Cependant, au moment où la spécification paraît, le langage est au delà de l'état de l'art. De plus, sa richesse n'est pas suffisante pour bon nombre d'applications. Cette contradiction met en évidence toute la complexité attachée au développement de HPF dans les années qui vont suivre. Elle est illustrée à la section 3.1.2 par la difficulté qu'aura Julien Zory au début de ses travaux de thèse à porter en HPF une application Fortran 77 parallèle et simpliste pour en tirer des performances correctes.

Approche affine

Une modélisation sous forme d'équations affines [7, 8] intégrée au cadre polyédrique [79, 80] nous a permis de proposer un cadre général pour la compilation du langage HPF, qui porte sur tous les aspects

```
      program triangle
      real A(30,30)
!hpf$ template T(68,30)
!hpf$ align A(i,j) with T(2*i,j)
!hpf$ processors P(4,2)
!hpf$ distribute T(block,cyclic(5)) onto P
      read *, m
      do j=3,m
        do i=3, m-j+1
          A(i,j) = 1.0D0
        end do
      end do
      end program
```

FIGURE 2.5 – Programme `triangle`

explication	*équations*
tableau `A`	$1 \leq \alpha_1 \leq 30$, $1 \leq \alpha_2 \leq 30$
template `T`	$1 \leq \theta_1 \leq 68$, $1 \leq \theta_2 \leq 30$
processeurs `P`	$1 \leq \psi_1 \leq 4$, $1 \leq \psi_2 \leq 2$
alignement	$\theta_1 = 2\alpha_1$, $\theta_2 = \alpha_2$
distribution	$\theta_1 = 17\psi_1 + \delta_1 - 16$, $0 \leq \delta_1 \leq 16$, $\theta_2 = 10\gamma_2 + 5\psi_2 + \delta_2 - 4$, $0 \leq \delta_2 \leq 4$
région manipulée	$\alpha_1 + \alpha_2 \leq 1 + m$, $\alpha_1 \geq 3$, $\alpha_2 \geq 3$
adressage local	$2\lambda_1 = \delta_1 - \eta_1 + 2$, $0 \leq \eta_1 \leq 1$, $\lambda_2 = 5\gamma_2 + \delta_2 + 1$

FIGURE 2.6 – Système linéaire pour le programme `triangle`

calculs, communications et allocations. Elle peut se concrétiser sous une forme matricielle, ouvrant la porte à des manipulations qui exploitent les propriétés spéciales de ces matrices telles l'unimodularité, les pseudo-inverses ou les formes de Hermite. Ces équations peuvent être manipulées par des bibliothèques mathématiques spécialisées (Linear/C3, Polylib) qui implémentent des sur-approximations des opérateurs ensemblistes ou permettent la génération de code.

Le placement issu des directives de placement HPF se décrit naturellement sous formes d'équations affines entières reliant les indices de boucles, ceux des tableaux, templates et processeurs, voire ceux de tableaux locaux des nœuds de calculs selon le mode d'adressage local envisagé. L'écriture précise de ces systèmes pour un code particulier nécessite des analyses avancées des accès aux tableaux telles que celles données par les régions convexes de tableaux décrites dans la thèse de doctorat de Béatrice Creusillet (*Array Region Analyses and Applications*, École des mines de Paris, décembre 1996).

Cette modélisation s'interprète géométriquement comme l'intersection de polyèdres entiers issus des inégalités et de treillis entiers issus des égalités. Certaines variables peuvent être interprétées comme des paramètres inconnus disponibles seulement à l'exécution, pourvu qu'elles n'interviennent pas comme des multiplicateurs : ceci pose la question du nombre de processeurs éventuellement paramétrique, donc le caractère inconnu au moment de la compilation peut aboutir à une taille de bloc inconnue qui sort du cadre où les manipulations des systèmes d'équations affines sont aisées.

La figure 2.5 illustre cette modélisation avec une zone triangulaire paramétrique du tableau `A` initialisée à zéro. Pour être utile, ce code devrait bien sûr ensuite traiter ces données et sortir des résultats. À partir de ce code, on peut dériver les équations de la figure 2.6, avec les variables : (α_1, α_2) les dimensions du tableau, (θ_1, θ_2) les dimensions du template, (ψ_1, ψ_2) les dimensions de l'arrangement de processeurs, δ_1 l'index dans un bloc sur la première dimension, (δ_2, γ_2) l'index dans le bloc et le numéro de cycle de la seconde dimension, m le paramètre, (λ_1, λ_2) un mode d'adressage local aux nœuds et η_1 une variable auxiliaire.

Ces systèmes qui représentent des ensembles de données peuvent ensuite être manipulés de manière ensembliste pour déterminer les calculs à effectuer localement à un nœud, les données à transférer entre les processeurs, etc. Ils incluent alors les équations concernant chaque tableau impliqué en sus de la définition du nœud chargé d'effectuer le calcul. La génération de code de parcours de ces ensembles avec des techniques *ad hoc* permet ensuite de réaliser les opérations de calculs et de communications. Les figures 2.7 et 2.8 montrent un tel code généré pour chaque nœud de calcul avec un paramétrage par

```
      program trianglenode
! partie locale du tableau A
      real Aloc(9,15)
! identité du noeud dans P(4,2)
      integer psi1, psi2
      psi1 = ...
      psi2 = ...
! réception du paramètre
      m = receive(...)
! parcours de la zone locale pour l'initialisation
      if (m.geq.5.and.17*psi1.leq.2*m+12) then
        do gamma2 = (2-psi2)/2, (6-psi2)/2
          do alpha2 = max(10*gamma2+5*psi2-4, 3),
     $          10*gamma2+5*psi2
            do alpha1 = max((17*psi1-15)/2, 3),
     $           min(17*psi1/2, m+1-alpha2, 30)
              lambda1 = (18-17*psi1+2*alpha1)/2
              lambda2 = alpha2-5*psi2-5*gamma2+5
              Aloc(lambda1, lambda2) = 1.0D0
            end do
          end do
        end do
      end if
      end program
```

FIGURE 2.7 – Code SPMD des nœuds pour `triangle`

```
      program trianglehost
! définition du paramètre
      read *, m
! diffusion auprès des noeuds
      broadcast(m, ...)
      end program
```

FIGURE 2.8 – Code de l'hôte pour `triangle`

l'identité du processeur et pour l'hôte du programme `triangle`. Si l'ensemble des données décrit n'est pas exact, des communications supplémentaires peuvent être nécessaires dans certains cas pour assurer que les bonnes valeurs seront disponibles.

Le code de parcours de polyèdre généré pour la définition du tableau inclus de nombreuses opérations entières. Certaines sont potentiellement coûteuses, en particulier la division et les extréma qui nécessitent des branchements ou au moins des gardes. La simplification et l'optimisation pointue de ces codes constituent un facteur important pour que de tels codes s'exécutent efficacement.

2.1.4 Modèle affine pour entrées-sorties et replacements

Je n'ai pas implémenté totalement l'approche générale décrite ci-dessus au cours de ma thèse. Par contre, elle a été déclinée, complétée, implémentée et testée dans les cas particuliers des entrées-sorties [22] et des replacements (modification au cours de l'exécution du placement des données, appelé *redistributions* en anglais) [65]. Voici les principaux résultats obtenus, et les subtilités croisées au cours du chemin.

Entrées-sorties

Dans le cadre des entrées-sorties, j'ai opté pour un modèle de machine avec un hôte frontal assurant le lancement du programme et les interactions avec les périphériques et des nœuds dédiés aux calculs. Ce modèle reste général car le rôle de l'hôte peut très bien être tenu par un des processeurs de calcul.

La modélisation des communications induites par les entrées-sorties s'intègre parfaitement dans le modèle affine décrit à la section 2.1.3, avec la simplification qu'il n'y a qu'un seul destinataire ou source des données selon que l'opération sur le périphérique est une sortie ou une entrée.

Par contre, le code généré doit prendre garde à l'exactitude ou non de la référence analysée. Dans le cadre d'une entrée et lorsque la description n'est pas exacte, une phase de collecte préliminaire des nœuds vers l'hôte est nécessaire : elle assure que les éléments diffusés après l'opération contiennent bien les valeurs initiales s'ils n'ont pas été effectivement redéfinis par l'entrée. La zone effectivement transférée pour une entrée ou une sortie peut s'appuyer sur les régions IN-OUT (élements de tableau importés dans dans une zone et exportés pour être utilisés par la suite), de nature différente des régions READ-WRITE (éléments de tableaux effectivements lus ou écrits, mais éventuellement créés localements) car prenant en

```
      parameter (n=20)
      dimension A(n)
!hpf$ template Ts(2,n,2)
!hpf$ align A(i) with T(*,i,*)
!hpf$ processors Ps(2,2,2)
!hpf$ distribute T(block,block,block) onto Ps
      A = ...
!hpf$ processors Pt(5,2)
!hpf$ redistribute T(*,cyclic(2),block) onto Pt
      ... = A
```

FIGURE 2.9 – Exemple de replacement HPF, illustré à la figure 2.10

compte l'utilisation réelle des données, comme présenté au chapitre 5 de [87].

L'éventuelle duplication de certaines données sur plusieurs processeurs de calcul induite par l'alignement se traduit par des variables libres sur les dimensions des processeurs. Elle permet de générer des diffusions, où un même message préparé est envoyé à plusieurs destinataires, amortissant ainsi le coût de préparation et profitant éventuellement de modes de communication particuliers du matériel sous-jacent.

Ce travail a également débouché sur une réflexion concernant les entrées-sorties parallèles et leur adaptation à un contexte HPF. En effet, l'approche suivie peut facilement s'adapter à un cadre où certains des processeurs de calculs, mais pas tous, disposent d'accès à des périphériques, par exemple des disques de stockage. L'idée est alors de représenter ces processeurs particuliers sous forme d'un arrangement supplémentaire de processeurs, et d'ajouter une *distribution* au sens HPF des processeurs de calculs vers ces processeurs particuliers :

```
!hpf$ processors P(4,2)
!hpf$ io processors Pio(2)
!hpf$ distribute P(block,*) onto Pio
```

Il induit des équations affines supplémentaires qui s'intègrent très naturellement au cadre général.

Replacements

Les replacements sont les redéfinitions dynamiques de HPF qui modifient le placement d'un tableau pour l'adapter à une nouvelle phase de calcul. Ils s'adaptent très bien à la modélisation affine, et il n'y a normalement aucune incertitude sur la zone concernée par les opérations puisque toutes les données sont à transférer.

Par contre, dans un souci d'optimisation, on peut restreindre la communication aux données qui seront effectivement utilisées dans la suite, par exemple avec une analyse de régions *IN*.

Le replacement décrit par l'exemple de la figure 2.9 est illustré à la figure 2.10. L'arrangement cubique des processeurs sources est coupé en deux, chacun des processeurs d'une moitié possédant la moitié du tableau en local, soit une duplication d'un facteur quatre. L'arrangement rectangulaire des processeurs cibles possède également une duplication d'un facteur deux sur la petite dimension.

Les équations liées au replacement sont présentées et complétées par rapport à [65] à la figure 2.11, avec : α_1 la dimension du tableau, $(\theta_1, \theta_2, \theta_3)$ les dimensions du template, $(\psi_1, \psi_2, \psi_3, \psi'_1, \psi'_2)$ les dimensions des arrangements de processeurs source et cible, δ_2 l'index dans un bloc sur la seconde dimension dans la source, (δ'_1, γ'_1) l'index dans le bloc et le numéro de cycle de la première dimension dans la cible, et enfin (λ_1, λ'_1) un mode d'adressage local aux nœuds dans la source et la cible respectivement.

La particularité du travail sur les replacements est qu'il exploite les opportunités liées aux duplications des données sur les processeurs pour générer des diffusions et partager la charge entre les processeurs qui disposent des mêmes données. Dans l'exemple ci-dessus les deux duplications de part et d'autre du replacement s'expriment par l'absence de contraintes reliant les dimensions du template (variables θ_1 et θ_3) à certaines des dimensions des arrangements de processeurs (variables ψ_1, ψ_3 et ψ'_2). La diffusion s'exprime naturellement en laissant libre la variable ψ'_2. Par contre, l'équilibrage de la charge doit relier les données disponibles sur la source $\psi_1\psi_3$ avec les besoins dans la cible avec ψ'_1 avec une simple forme linéaire, par exemple :

$$\psi'_1 = 4c + 2\psi_3 + \psi_1 - 2$$

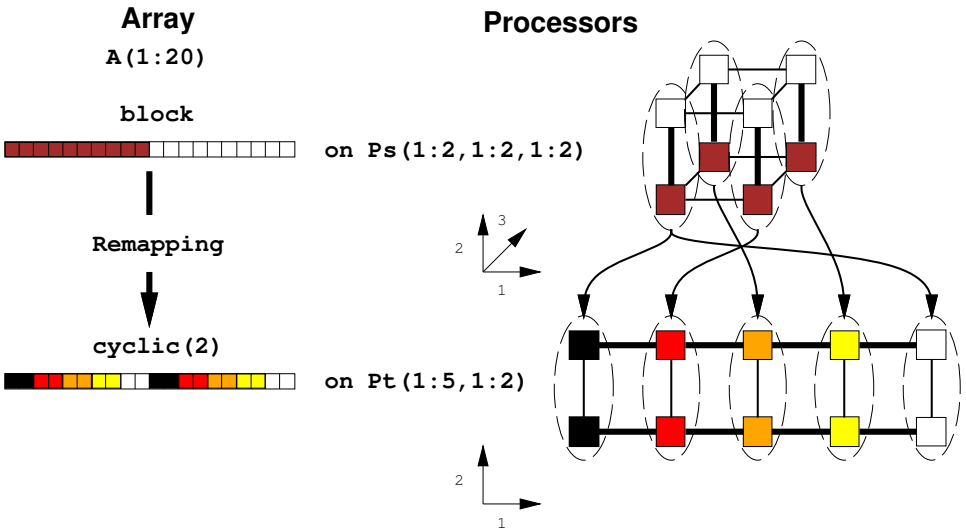


FIGURE 2.10 – Replacement du tableau `A` dans le code de la figure 2.9

description	*équations*
paramètre	n
tableau `A`	$1 \leq \alpha_1 \leq n$
template `T`	$1 \leq \theta_1 \leq 2$, $1 \leq \theta_2 \leq n$, $1 \leq \theta_3 \leq 2$
alignement	$\theta_2 = \alpha_1$
processeurs `Ps`	$1 \leq \psi_1 \leq 2$, $1 \leq \psi_2 \leq 2$, $1 \leq \psi_3 \leq 2$
distribution sur `Ps`	$0 \leq \delta_2 \leq 9$, $\theta_2 = 10\psi_2 + \delta_2 - 9$
adresse locale sur `Ps`	$\lambda_1 = \delta_2$
processeurs `Pt`	$1 \leq \psi'_1 \leq 5$, $1 \leq \psi'_2 \leq 2$
distribution sur `Pt`	$0 \leq \delta'_1 \leq 1$, $\theta_2 = 10\gamma'_1 + 2\psi'_2 + \delta'_1 - 1$
adresse locale sur `Pt`	$\lambda'_1 = 2\gamma'_1 + \delta'_1$

FIGURE 2.11 – Équations du replacement

La variable c incarne le cycle de partage sur les groupes de processeurs cibles. Elle ne préjuge cependant pas du placement effectif de ces processeurs au sens HPF sur des processeurs au sens matériel du terme. La prise en compte de ce niveau pourrait potentiellement induire des communications moindres en profitant des données déjà effectivement disponibles sur un processeur physique.

2.1.5 Optimisation des communications

Indépendemment de la génération de code de communication avec des méthodes affines discutée ci-dessus, nous avons aussi travaillé à l'optimisation des communications induites par un calcul parallèle.

Traduction des replacements HPF en copies de tableaux

La compilation des replacements HPF peut être transformée en simples copies de tableaux [31], en associant à chaque point du programme les placements valides d'un tableau, et en insérant des affectations quand une mise à jour dans un état particulier est nécessaire. Alliée à la compilation des communications présentée à la section précédente, ces deux techniques offrent une gestion pratique complète des replacements de HPF.

Le problème est placé dans le cadre très classique en compilation d'une propagation sur le graphe de contrôle, en constituant un sous-graphe du graphe de contrôle complet appelé le *Remapping graph*, ou graphe de replacements. Les nœuds du graphe sont les points où le placement évolue : en plus des directives explicites de replacement, on trouve aussi les appels de fonctions impliquant des tableaux distribués. À partir d'une construction initiale du graphe, et en prenant en compte les effets en lecture, en écriture, les redéfinitions et les replacements sur les tableaux, on précise, pour chaque référence, les versions du tableau qui doivent être utilisées, et celles qui sont vivantes. Cette propagation des informations sur le graphe permet des optimisations, en particulier d'éliminer les communications des replacements si les données sont encore disponibles dans la forme attendue.

Le résultat net de l'optimisation est un programme HPF à placement statique, équivalent au programme dynamique déclaré initialement. La figure 2.12 illustre sur un exemple simple de calcul d'un produit de matrices le résultat des opérations : deux tableaux supplémentaires sont déclarés, et des copies sont insérées dans le code pour exprimer le passage d'un placement à l'autre. Les allocations sont faites au plus près des besoins. Enfin, le tableau `B` est gardé malgré l'initialisation de la variante `B2` parce que le tableau va être encore utilisé pour la sortie finale. On évite ainsi un mouvement. Par contre, l'algorithme, qui travaille au niveau tableaux, ne détecte pas que la variante `B2` correspond à une réplication de `B` qui le contient, donc qu'une réaffectation à `B` n'occasionnerait en fait aucune communication, et donc la mémoire correspondant pourrait être libérée.

Une restriction importante pour que cette méthode soit applicable est que les placements doivent être statiquement connus. Cet aspect est discuté plus avant à la section 3.1.1 qui revient sur la conception du langage HPF et les contraintes souhaitables pour en faciliter à la fois l'utilisation et la compilation.

Placements des communications hôte-accélérateur

Une approche similaire a été proposée lors des travaux de thèse de Mehdi Amini pour la compilation des communications hôte-accélérateur de type GPU [3, 4, 5], où l'on gère un tableau dans deux états, sur l'hôte et sur l'accélérateur supposé unique, et où l'on cherche à éliminer des transferts coûteux entre les deux.

L'analogie consiste à identifier un placement sur l'hôte et un sur l'accélérateur. Contrairement à HPF, il n'y a pas de points de replacements explicites, mais ceux-ci sont dérivés des utilisations effectives des données placées par le compilateur soit du côté de l'accélérateur si un noyau de calcul (nid de boucles) compatible a été extrait, soit du côté de l'hôte. Le meilleur point de placement des communications nécessaires est ensuite décidé sur le graphe de flot de contrôle complet, et non pas sur un graphe réduit comme dans le cas HPF.

2.2 Micro parallélisme

Après des années consacrées à l'exploitation d'un macro-parallélisme de données sur des multi-processeurs à mémoire distribuée pour en tirer des performances qui restent souvent éloignées des performances crêtes,

```
      program remaps
      parameter (n=1000)
      real, dimension(n,n):: A, B, C
      allocatable A, B, C
!hpf$ align with A:: B, C
!hpf$ distribute A(block,block)
      allocate A, B, C
      read *, B
      C = B - 1
!hpf$ realign B(i,*) with A(i,*)
!hpf$ realign C(*,j) with A(*,j)
!hpf$ independent(j, i), new(s)
      do j=1, n
        do i=1, n
! réduction sur s
          s = 0
          do k=1, n
            s = s + B(i,k)*C(k,j)
          end do
          A(i,j) = s
        end do
      end do
!hpf$ realign with A:: B, C
      print *, A-B
      end program
```

```
      program remaps
      parameter (n=1000)
      real, dimension(n,n):: A, B, C, B2, C2
      allocatable A, B, C, B2, C2
!hpf$ align with A:: B, C
!hpf$ align B2(i,*) with A(i,*)
!hpf$ align C2(*,j) with A(*,j)
!hpf$ distribute A(block,block)
      allocate B
      read *, B
      allocate C
      C = B - 1
! copies...
      allocate B2
      B2 = B
! on garde B...
      allocate C2
      C2 = C
      deallocate C
      allocate A
!hpf$ independent(j, i), new(s)
      do j=1, n
        do i=1, n
! réduction sur s
          s = 0
          do k=1, n
            s = s + B2(i,k)*C2(k,j)
          end do
          A(i,j) = s
        end do
      end do
      deallocate B2, C2
      print *, A-B
      deallocate A, B
      end program
```

FIGURE 2.12 – Traduction des replacements (à gauche) en simples copies (à droite)

l'écart pouvant être attribué en partie aux coûts et à la complexité des communications avec des espoirs modérés d'améliorer significativement l'état de l'art, la problématique de l'exécution efficace d'un code sur un simple mono-processeur m'a paru encore plus importante.

Un premier élément de motivation a été de constater que le problème était (et est toujours) loin d'être résolu. Ajouter quelques parenthèses à une expression dans l'application Onde24 (voir la section 2.2.1) faisait varier les performances d'un facteur deux, du même ordre de grandeur que le gain atteint avec la version parallélisée sur réseau de stations de travail.

Un second élément était que la compilation de HPF, en particulier ce qui concerne les communications et l'adressage des données, aboutit à la génération automatique d'un code qu'en tant que programmeur je trouvais largement améliorable (code invariant dans des boucles, réduction de force à appliquer, expressions communes) mais dont les compilateurs n'arrivaient pas à extraire seuls de bonnes performances.

Ces deux problèmes ont en commun que l'amélioration des performances passe par des techniques mathématiques et de compilation largement connues et décrites depuis de nombreuses années, mais exploitant des propriétés spécifiques des opérateurs, comme l'associativité et la commutativité, la distributivité permettant la factorisation.

Un point commun avec les travaux précédents est que l'amélioration des performances s'appuie toujours sur le parallélisme, mais cette fois à l'intérieur du processeur, grâce à la disponibilité d'unités de calculs multiples et/ou pipelinées, ainsi que la combinaison de certains opérateurs implémentée directement par le matériel.

Mes travaux ont ensuite évolué vers la réduction de la consommation énergétique pour les applications embarquées, avec la thèse de Youcef Bouchebaba entamée à la suite de mon post-doc à l'IMEC, puis dans le cadre du projet FREIA focalisé sur les applications de traitement d'image tournant sur des petits accélérateurs matériels hybrides de type SoC *(System on Chip)*. Ces travaux sont présentés à la section 3.2. Ces dernières machines n'excluent cependant pas les performances au sens de la vitesse, ni le parallélisme dans de l'exécution des opérations.

2.2.1 Onde24, une application simple

L'évaluation des expressions du code initial et du code généré automatiquement a pour point de départ l'application Onde24 développé par l'IFP (Institut Français du Pétrole). Il s'agit d'un code de géophysique de moins de 1000 lignes de Fortran 77 qui calcule une propagation d'ondes 2D dans un milieu hétérogène avec une méthode de différences finies. Les conditions aux limites sont absorbantes sur les bords du terrain pour modéliser la propagation dans un espace plus grand que la grille représentée, réflexives sur la frontière du haut vers l'atmosphère.

Les figures 2.13 et 2.14 illustrent la propagation de l'onde dans un milieu hétérogène pour la première, et les mesures réalisées par un sismographe pour la seconde. On voit en particulier la réflexion de l'onde occasionnée par la discontinuité carrée du terrain dans le quart en bas à droite. La technique de calcul à base de différences finies aboutit naturellement à de simples tableaux bi-dimensionnels qui représentent directement l'espace du terrain. La figure 2.15 montre la situation géométrique des données et les différents motifs de calculs qui s'appliquent aux les différentes zones.

2.2.2 Optimisations des expressions

Les travaux de thèse de Julien Zory [87], que j'ai encadré et auxquels j'ai participé, ont principalement porté sur l'optimisation des expressions dans un cadre commutatif et associatif, et plus généralement de liberté de manipulations algébriques où un grand nombre de transformations sont explorées avec l'objectif de réduire les temps d'exécution des codes.

Expression de Onde24

La figure 2.16 montre l'expression la plus intensive de l'application Onde24 telle qu'elle apparait dans le code source. La pression (tableau `U`) pour le point de l'espace `(i,j)` au temps suivant `kp` est calculée en fonction des pressions aux deux temps précédents `km` et `kp` (espace réutilisé) au même point, des pressions voisines au temps précédent (décalages de 1 et 2), et de la vitesse caractéristique du milieu au point considéré (tableau `V` qui modélise le terrain), selon le schéma IN de la figure 2.15.

Cette expression est intéressante a plusieurs titres et a permis à Julien Zory de tester différentes techniques de compilation [88] d'expressions qui améliorent les performances :

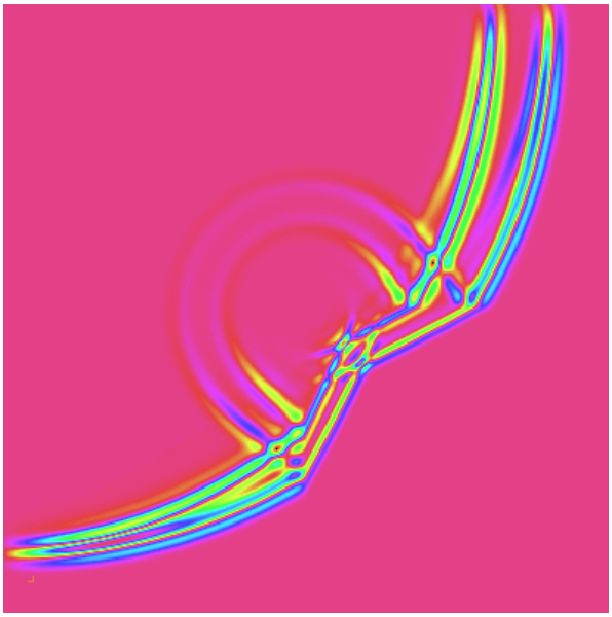


FIGURE 2.13 – Onde24 – Propagation d'ondes 2D

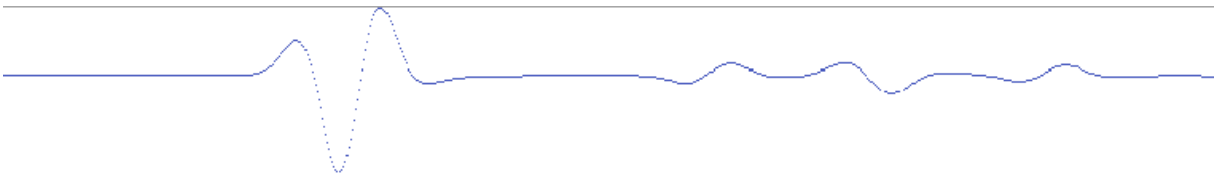


FIGURE 2.14 – Onde24 – Exemple d'enregistrement par un sismographe

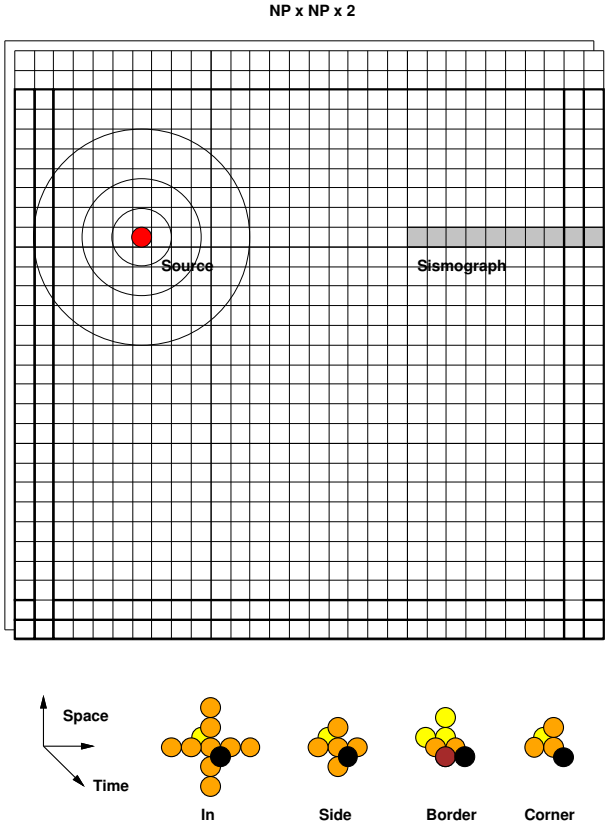


FIGURE 2.15 – Onde24 – Calculs et zones

```
 DO j = 3, NP-2
   DO i = 3, NP-2
     U(i,j,kp) =
$       2*U(i,j,km) - U(i,j,kp)
$     - V(i,j) * ( 60 * U(i,j,km)
$     - 16*(U(i+1,j,km) + U(i-1,j,km)
$         + U(i,j-1,km) + U(i,j+1,km))
$         + U(i+2,j,km) + U(i-2,j,km)
$         + U(i,j-2,km) + U(i,j+2,km))
    END DO
 END DO
```

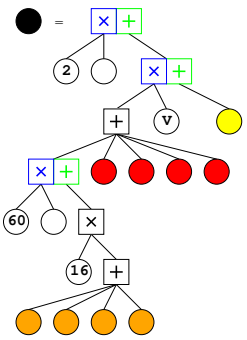


FIGURE 2.16 – Onde24 – différence finie au cœur du domaine

- elle est tout à fait représentative de ce que peut produire la discrétisation d'équations physiques dans un code de calcul ;
- il existe deux factorisations distinctes de l'expression, soit sur `V(i,j)` comme présenté dans la version initiale, soit sur `U(i,j,km)` après distribution ;
- les additions, dans un contexte associatif-commutatif, permettent d'équilibrer ou au contraire de déséquilibrer l'expression, modifiant le parallélisme interne et la pression sur les registres ;
- le résultat de multiplications est additionné, ce qui ouvre la voie à l'utilisation éventuelle d'opérations *multiply-add* combinées ;
- le programmeur a fait un choix explicite de forme de l'expression sans doute plus guidé par un critère de lisibilité que par une question de performance ou de stabilité numérique.

À partir de cette formule il existe **69300** expressions mathématiquement équivalentes, dérivées en exploitant les propriétés algébriques des opérateurs. Les performances de l'exécution effective de ces expressions, par exemple sur les processeurs IBM Power2, varient du simple au double.

Manipulations d'expressions

Il est possible de reprendre toutes les optimisations de code décrites dans le *Dragon Book* de Aho, Sethi et Ullman (*Compiler Principles, Techniques and Tools*, Addison-Wesley, 1986) et de les appliquer dans un contexte de liberté algébrique, voire d'en appliquer d'autres fondées sur les propriétés mathématiques des opérateurs. Voici un petit catalogue des transformations prises en compte :

substitution en avant *(forward substitution)* propagation d'une expression dans différents contextes où elle est utilisée, pour éventuellement ouvrir de nouvelles possibilités de factorisation ou de détection d'invariants. Cette transformation fait le contraire de l'*atomisation* (passage en expressions élémentaires de type code 3-adresses). Elle augmente le nombre d'opérations.

$$\begin{array}{l|l} t = a + b & \\ x = t + c & x = a + b + c \\ y = t + d & y = a + b + d \end{array}$$

élimination de code mort *(dead code elimination)*

$$\begin{array}{l|l} t = a + b & \textit{rien} \\ t \text{ n'est pas utilisé dans la suite} & \end{array}$$

distribution d'une opération sur une autre, là encore pour mettre à jour d'éventuelles possibilités d'optimisation. Cette transformation peut augmenter le nombre d'opérations.

$$x(y + z) + yz = xy + xz + yz$$

simplification d'une opération sur un élément neutre ou absorbant, ou de toute autre propriété algébrique, comme par exemple :

$$x - x + x = x$$
$$\sqrt{x^2} = |x|$$

réduction de force remplacer des opérations coûteuses par des opérations éventuellement moins coûteuses.

$$3x = x + x + x$$
$$4x = x << 2 \text{ – } \textit{pour } x \textit{ entier}$$

factorisation d'une expression (contraire de la distribution), éventuellement avec la commutativité :

$$yx + yz + xz = y(x + z) + xz$$
$$xy + xz + yz = y(x + z) + xz$$

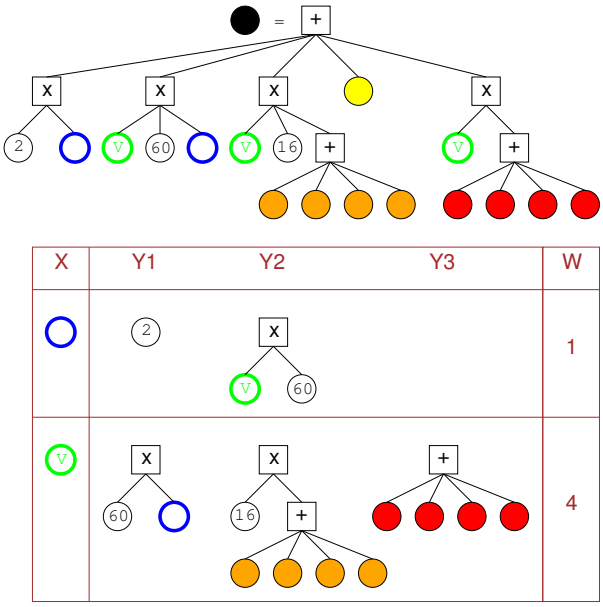


FIGURE 2.17 – Onde24 – Choix des factorisations conduisant aux deux expressions de la figure 2.18
Pour la première, l'élément commun est présent dans deux sous-expressions.
Pour la seconde, l'élément commun est dans trois sous-expressions

reconnaissance de FMA ***(Floating-point Multiply Add)*** et autres opérations combinées proposées par certains processeurs :

$$ax + b = \text{fma}(a, x, b)$$

équilibrage *(balancing)* ou déséquilibrage de l'arbre représentant une expression en jouant sur l'associativité et éventuellement la commutativité :

$$((a + b) + c) + d = (a + b) + (c + d)$$

déplacement de code invariant *(invariant code motion)* hors d'une boucle (peut être le contraire de la substitution en avant) :

`do` $i = 1, n$	$x = 2d$
$\quad x = 2d$	`do` $i = 1, n$
$\quad A(i) = x * i$	$\quad A(i) = x * i$

élimination de sous-expressions communes *(common subexpression elimination)* pour éviter de faire deux fois le calcul d'une opération (peut avoir l'effet inverse de la substitution en avant) :

$x = a + b + c$	$t = a + c$
$y = a + d + c$	$x = t + b$
	$y = t + d$

Il est important de noter que ce petit catalogue comporte des transformations et leurs contraires : la factorisation a l'effet inverse de la distribution, de même que la détection des invariants et des expressions communes vis à vis de la propagation en avant. Au cours des développements et des expériences, nous avons réalisé qu'une normalisation des expressions était nécessaire pour neutraliser les choix éventuellement opérés par le développeur afin d'en faire peut-être d'autres. Cette phase de normalisation aboutit à augmenter temporairement le nombre d'opérations nécessaires, pour découvrir de nouvelles opportunités

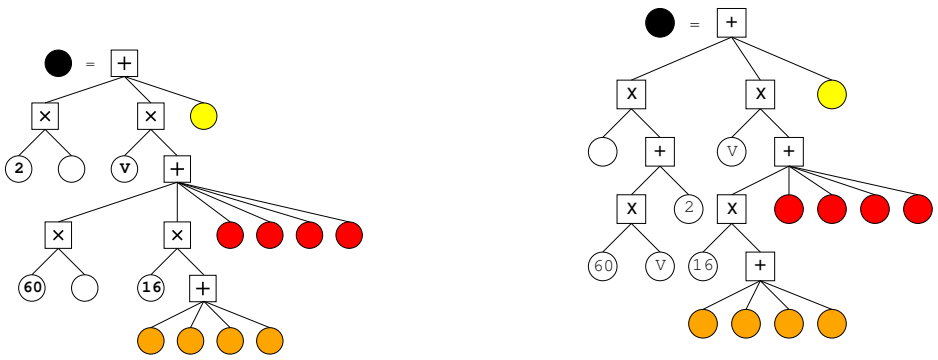


FIGURE 2.18 – Onde24 – les deux factorisations possibles de par l'alternative de la figure 2.17

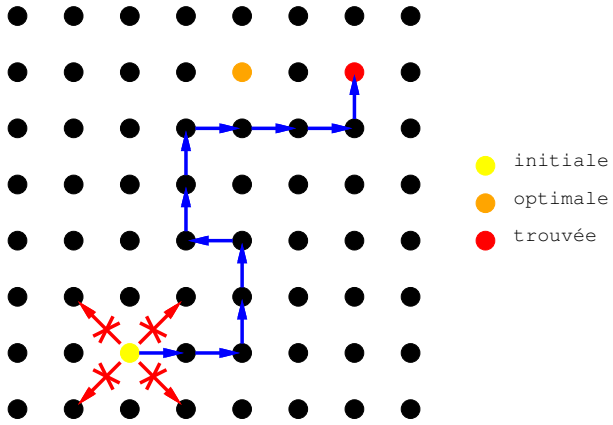


FIGURE 2.19 – Optimisation incrémentale guidée par des heuristiques

d'optimisation, ou, au pire, retrouver celles qui ont été défaites. C'est par exemple le cas avec les deux factorisations possibles de l'expression principale de Onde24 présentées aux figures 2.17 et 2.18.

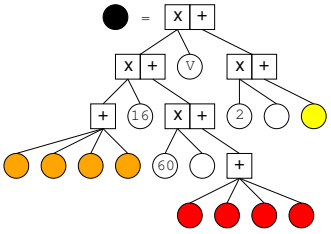
Un autre aspect est que des structures de données doivent être adaptées aux opérations que l'on envisage. Il est par exemple utile de représenter directement les opérations associatives et commutatives en utilisant des opérateurs n-aires plutôt que binaires afin d'éviter des manipulations d'arbres. Ces opérateurs sont représentés directement dans l'implémentation réalisée dans PIPS [74] par une liste d'arguments passés aux opérations arithmétiques.

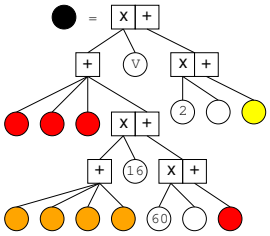
Approche heuristique

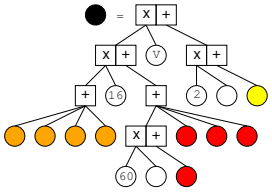
La démarche globale d'optimisation des expressions proposée au cours des travaux de Julien Zory est très combinatoire, et nombre des sous-problèmes abordés sont connus comme étant NP-complets. De plus, ces problèmes théoriques combinatoires ne sont qu'une approximation de l'objectif réel qui est de réduire le temps d'exécution du code : minimiser dans l'absolu le nombre d'opérations effectuées n'est de ce point de vue en rien la garantie de l'exécution la plus rapide. En conséquence, le résultat des travaux n'était pas une optimisation complète et totale, mais plutôt une série d'améliorations incrémentales à base d'itérations guidées par des heuristiques, illustrée par la figure 2.19

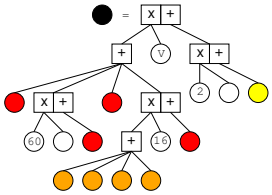
Le choix de ces heuristiques ne s'est cependant pas fait au hasard : pour profiter du parallélisme interne à un processeur il faut trouver des opérations indépendantes les unes des autres, ce qui se traduit naturellement par une recherche d'équilibrage de la forme globale des expressions produites exprimée par une simple mesure du centre gravité de l'expression (profondeur moyenne pondérée des opérations), en particulier pour la factorisation et la recherche des `fma`. D'autres fonctions de coût sont envisageables, comme le poids total pour un processeur sans parallélisme où il faut réduire le nombre d'opérations, ou la profondeur maximale pour un processeur à parallélisme large (VLIW).

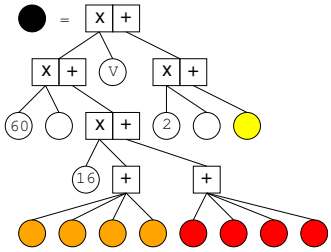
Un point important est que les différentes transformations décrites ci-dessus interagissent entre elles.

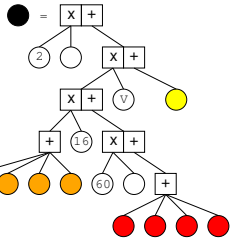

version 1. – 151 Mflop/s

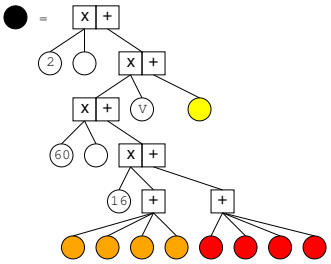

version 1a. – 141 Mflop/s

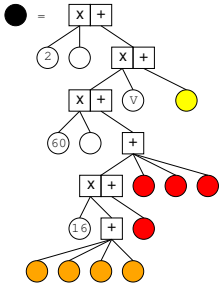

version 1b. – **164** Mflop/s

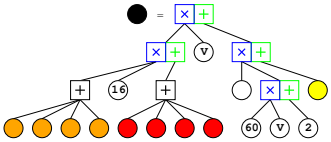

version 1c. – 144 Mflop/s


version 2. – 149 Mflop/s


version 3. –137 Mflop/s


version 4. – 130 Mflop/s


version 4a. – 126 Mflop/s


version 5. – **162** Mflop/s

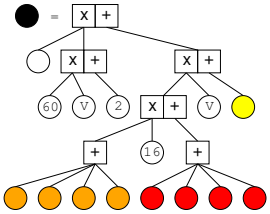

version 6. – 142 Mflop/s

FIGURE 2.20 – Onde24 – quelques choix de multiplications-additions et Mflop/s sur P2SC 595
La version initiale de l'expression atteignait 102 Mflop/s, domaine de taille 800×800 *– hors cache*

Certaines opérations en interdiront d'autres ; par exemple l'extraction d'une `fma` peut empêcher la détection d'une expression commune comme dans l'exemple suivant : $x = ab + c; y = ab + d$. Il a donc été nécessaire d'ordonner les différentes opérations d'optimisation de manière à préserver au mieux l'intérêt des transformations suivantes, en faisant des choix qui leurs sont favorables, et en prenant les décisions les plus importantes au plus tôt. À titre d'illustration, le gain potentiel du déplacement de code invariant hors d'une boucle est d'un ordre de grandeur (n itérations de la boucle) au dessus de l'extraction d'une expression commune qui ne serait pas invariante.

Les résultats obtenus ont été positifs à double titre :

- une implémentation rapide de ces différentes transformations a pu être réalisée, malgré les combinatoires sous-jacentes explosives, grâce à des choix judicieux d'outils (en particulier l'outil CAVEAT du CEA), d'algorithmes et d'heuristiques ;
- les améliorations de performances en temps d'exécution des codes traités ont été bonnes à très bonnes, quand les codes s'y prêtaient bien sûr.

Par contre, la stratégie d'implémentation en partie à l'extérieur de PIPS avec l'outil de réécriture CAVEAT du CEA ne nous a pas permis de diffuser ce module sous forme de logiciel libre, ni de le maintenir sur le long terme.

La figure 2.20 montre un petit sous-ensemble de la diversité des solutions de factorisation et de multiplication-addition combinées possibles avec l'expression principale de l'application Onde24, ainsi que les performances résultantes pour des processeurs Power2 IBM à l'époque.

Notre démarche est restée heuristique dans la mesure où nous ne disposons pas de modèle de performance précis et fiable pour les processeurs cibles, et où une approche exhaustive et expérimentale n'aurait pas sa place dans un compilateur à cause de son coût prohibitif et ses difficultés d'implémentation (par exemple le coût d'exécution d'une expression peut dépendre du comportement mémoire qui peut changer d'une exécution à l'autre si la taille du problème change).

2.2.3 Réduction énergétique

Le travail de thèse de Youcef Bouchebaba [10, 11, 13, 12, 84, 15, 14] ont été initié à la suite de mon post-doc à l'IMEC pour étendre et automatiser les résultats obtenus sur la compilation de calculs de type convolution à base de motif. Au cours de cette thèse que j'ai encadrée, Youcef a collaboré avec moi-même et François Irigoin. Il s'est focalisé sur la minimisation des transferts mémoire, qui sont une source importante de consommation de courant électrique, en prenant en compte les registres, la hiérarchie de cache et la mémoire principale. Les applications cibles sont algorithmiquement simples, dans le domaine du traitement du signal embarqué : téléphones portables, appareils photo et caméra numériques. Elles consistent en des parcours de tableaux avec des calculs à base de motifs réguliers. Les transformations mises en œuvre sont classiques, principalement :

fusion de boucles successives pour factoriser les espaces d'itérations et réutiliser les données au plus vite : la gestion des bords pose des difficultés particulières et induit des conditions supplémentaires dans les nouveaux nids de boucles construits ;

pavage multi-niveau pour gérer les différentes mémoires, en partant du ou des antémémoires et en allant jusqu'au niveau des registres ;

déroulage de boucles pour faire apparaître explicitement les intersections des motifs de calcul ;

ré-ordonnancement très important des instructions résultantes.

Les déroulage et réordonnancement de boucles se nomment parfois dans la littérature *unroll-and-jam*. Ces différentes transformations doivent bien sûr respecter les dépendances de données de l'application de manière à assurer la pertinence des calculs effectués.

L'accent est mis sur l'allocation et l'adressage des données temporaires utilisées pour stocker les résultats intermédiaires entre les tuiles de calcul. Des petits tableaux intermédiaires sont mis à jour à chaque niveau de boucle. Le dernier niveau mémoire correspond à une gestion explicite des registres, avec un glissement des valeurs entre motifs contigus.

```
! convolution
      INTEGER FUNCTION ma1(x,y,z)
      INTEGER x, y, z
      ma1 = (68*x + 99*y + 68*z) / (2*68+99)
      END FUNCTION ma1
! écart maximal
      INTEGER FUNCTION ma2(x,y,z,l,m,o,p,q,r)
      INTEGER x, y, z, l, m, o, p, q, r
      ma2 = MAX(ABS(x-m),ABS(y-m),ABS(z-m),ABS(l-m),
     $          ABS(o-m),ABS(p-m),ABS(q-m),ABS(r-m))
      END FUNCTION ma2
! inversion
      INTEGER FUNCTION ma3(x)
      INTEGER x
      ma3 = 255-x
      END FUNCTION ma3
! seuillage
      INTEGER FUNCTION ma4(x,y,z,l,m,o,p,q,r)
      INTEGER x, y, z, l, m, o, p, q, r
      ma4 = 255
      IF (x.GT.m.OR.y.GT.m.OR.z.GT.m.OR.l.GT.m.OR.
     $    o.GT.m.OR.p.GT.m.OR.q.GT.m.OR.r.GT.m) THEN
         ma4 = 0
      END IF
      END FUNCTION ma4
! chaîne de calcul
      SUBROUTINE traite(IN,OUT)
      PARAMETER(N=200,M=640)
      INTEGER N, M
      INTEGER IN(N,M), OUT(N,M), Tem(N,M), Gim(N,M), Cim(N,M), Te2(N,M)
      INTEGER i, j
      DO i=2, M-1
         DO j=1, N
            Tem(j,i) = ma1(IN(j,i-1),IN(j,i),IN(j,i+1))
         END DO
      END DO
      DO i=2, M-1
         DO j=2, N-2
            Gim(j,i) = ma1(Tem(j-1,i),Tem(j,i),Tem(j+1,i))
         END DO
      END DO
      DO i=3, M-2
         DO j=3, N-2
            Cim(j,i) = ma2(Gim(j-1,i-1),Gim(j,i-1),Gim(j+1,i-1),
     $                     Gim(j-1,i),  Gim(j,i),  Gim(j+1,i),
     $                     Gim(j-1,i+1),Gim(j,i+1),Gim(j+1,i+1))
         END DO
      END DO
      DO i=3, M-2
         DO j=3, N-2
            Te2(j,i) = ma3(Cim(j,i))
         END DO
      END DO
      DO i=4, M-3
         DO j=4, N-3
            OUT(j,i) = ma4(Te2(j-1,i-1),Te2(j,i-1),Te2(j+1,i-1),
     $                     Te2(j-1,i),  Te2(j,i),  Te2(j+1,i),
     $                     Te2(j-1,i+1),Te2(j,i+1),Te2(j+1,i+1))
         END DO
      END DO
      END SUBROUTINE traite
      PROGRAM cavity
      PARAMETER(N=200,M=640)
      INTEGER N, M
      INTEGER IN(N,M), OUT(N,M)
      CALL readimage(IN, 'in/200-640.pgm')
      CALL Traite(IN, OUT)
      CALL writeimage(OUT, 'out/200-640.pgm')
      END PROGRAM cavity
```

FIGURE 2.21 – Application CAVITY

```
DO i = ...
   DO j = ...
      A(j,i) = ...
      B(j,i) = f(A(j,i),A(j-2,i-2))
   END DO
END DO
```

FIGURE 2.22 – Exemple avec deux tableaux...

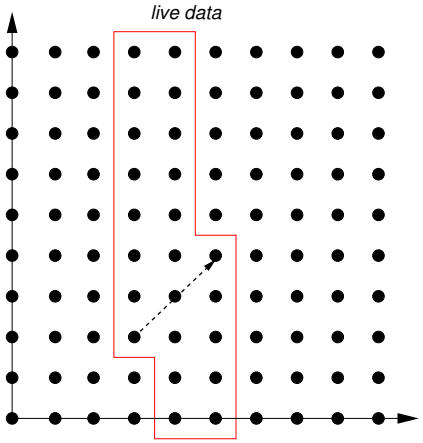


FIGURE 2.23 – Allocation d'un tableau circulaire

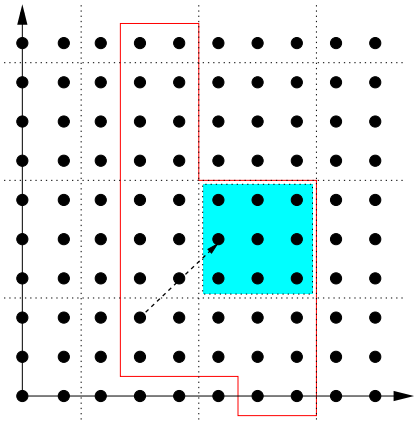


FIGURE 2.24 – Données vivantes après un pavage

Les tests effectués se sont appuyés entre autre sur une application de traitement d'image médicale pour détecter des cavités dans des images obtenues par scanner. Une version en Fortran en est présentée à la figure 2.21. La fusion de tous les nids de boucles permet d'obtenir un seul nid calculant un pixel de l'image finale à chaque itération, en combinant tous les calculs intermédiaires.

Les données intermédiaires de niveau cache peuvent être stockées dans un tampon circulaire unique ou bien dans une série de petites mémoires tampon selon les différentes dimensions du pavage. Ces deux solutions sont présentées pour l'exemple de la figure 2.22, où le tableau intermédiaire `A` est éliminable, selon les transformations opérées : dans la figure 2.23, le tableau est simplement remplacé par un tampon circulaire. Par contre, dans le cas d'un pavage, la taille ou la structure de ce tampon doit être adaptée comme le montre la figure 2.24.

2.2.4 Traitement d'image embarqué

Le projet ANR FREIA *Framework for Embedded Image Applications* a regroupé le CRI (Centre de recherche en informatique, MINES ParisTech et ARMINES), le CMM (Centre de morphologie mathématique, MINES ParisTech et ARMINES), l'ENST-Bretagne et THALES. L'objectif du projet était de fournir en environnement de développement d'applications de traitement d'image pour des accélérateurs matériels embarqués, allant du langage (une API en C) au générateur de code VLIW pour un processeur SIMD. Les résultats permettent de faciliter le portage d'applications de traitement d'image en temps réel sur des accélérateurs de calcul spécifiques, à base de traitement de flux (pipeline vectoriel) ou d'architecture SIMD, qui sont de plus dynamiquement reconfigurables.

Les enjeux de ces optimisations couvrent à la fois la vitesse des calculs, le coût économique d'une solution et la consommation énergétique. Elles ont pour but de permettre de profiter de ces architectures matérielles avec un effort de développement limité. Le projet a abordé trois aspects :

- la conception d'une API spécifique en C pour le traitement d'image afin d'assurer la pérénité et la portabilité des codes ; la figure 2.25 montre une application utilisant cette API, avec des initialisations, des entrées-sorties et des calculs utilisant à la fois des opérateur complexes et basiques sur des images.

- l'implémentation d'applications tests pour valider à la fois l'API en entrée et les performances en sortie ;

```
#include <freia.h>                          // use FREIA user API
int main(void) {
  int32_t min, vol;                         // declarations...
  freia_data2d *in = freia_create_data(...); // and similarly od, og
  freia_dataio fin, fout;                   // input & output descriptors
  freia_common_open_input(&fin, 0);         // idem fdout
  freia_common_rx_image(in, &fin);          // get input image
  freia_aipo_global_min(in, &min);          // some computations
  freia_aipo_global_vol(in, &vol);
  freia_cipo_dilate(od, in, 8, 10);       // dilate with 8 neighbors at depth 10
  freia_cipo_gradient(og, in, 8, 10);   // gradient with 8 neighbors at depth 10
  printf("input min=%d vol=%d\n", min, vol);  // show results
  freia_common_tx_image(od, &fout);         // idem og
  freia_common_destruct_data(in);           // idem od, og...
  freia_common_close_input(&fin);           // idem fdout
  return 0;
}
```

FIGURE 2.25 – ANR999 – une application utilisant l'API FREIA

- la conception d'un environnement de développement, une chaîne de compilation aussi complète que possible pour exécuter les applications sur trois types d'accélérateurs différents.

Ma contribution a concerné essentiellement la chaîne de compilation globale [69, 70, 9, 71, 72, 73] et est détaillée à la section 3.2. Je discute également à la section 3.3.3 des aspects co-conception langage, compilateur, exécutif et matériel.

2.3 Autres optimisations

D'autres travaux ont aussi visé l'amélioration des performances à prendre cette fois à un sens plus large que la réduction du temps d'exécution. Tous ces développements n'ont hélas pas toujours fait l'objet de publications, mais ont cependant été importants pour démontrer le côté pratique de certaines méthodes d'analyses ou de transformations proposées par ailleurs.

2.3.1 Manipulations polyédriques

La mise en pratique de méthodes de calculs polyédriques dans les phases d'analyses (*transformers*, préconditions, régions de tableaux, tests de dépendances) et de transformations (tuilage, génération de code de communication pour HPF) dans un prototype de compilateur interprocédural comme PIPS pose vite des problèmes de performance, et une grande attention doit être portée sur l'implémentation des structures de données et des opérateurs devant manipuler les polyèdres entiers. J'ai donc été amené à intervenir de manière fine dans l'implémentation de la bibliothèque linéaire **Linear/C3** et indirectement sur la **Polylib** utilisées par PIPS afin de réduire les temps d'exéction et d'améliorer la justesse des résultats.

Amélioration des calculs sur les polyèdres

Une série d'améliorations concerne les opérateurs de calculs sur les polyèdres eux-mêmes. Voici quelques exemples de ces travaux.

La premier travail effectué au cours de ma thèse concernait l'amélioration de l'implémentation de l'algorithme de Ancourt et Irigoin *row echelon* qui génère les boucles de parcours de points entiers dans un polyèdre. Le principe de l'algorithme est de projeter les variables d'itération pour trouver les bornes de boucles les plus externes, et d'éliminer les contraintes redondantes en prenant en compte les bornes des boucles externes en descendant les niveaux de boucles. Les expériences ont montré que le code généré pouvait parfois être relativement laid et lourd, en particulier parce que des contraintes coûteuses (contenant plus de variables et d'opérations) étaient gardées à la place de contraintes beaucoup plus simples. L'algorithme est en effet sensible à l'ordre dans lequel les contraintes sont éliminées. L'idée

naturelle et simple pour améliorer le résultat a été de trier les contraintes de manière à essayer d'éliminer les contraintes les plus coûteuses en priorité, et donc de générer un code plus simple.

Un second travail a été d'implémenter une simplification rapide des systèmes de contraintes pour certaines opérations, au sens de représenter un même ensemble avec des équations plus simples (moins de variables et d'opérations). Les différentes opérations se passent beaucoup mieux avec des égalités qu'avec des inégalités qui peuvent induire une explosion exponentielle du nombre de contraintes lors des projections, par exemple pour l'élimination de redondance ou bien pour calculer une représentation par système générateur des polyèdres. En conséquence, une recherche d'égalité, par exemple avec une délinéarisation :

$$an + m = 0 \wedge -a < m < a \Rightarrow n = 0 \wedge m = 0$$

où n et m sont des formes linéaires entières est toujours profitable.

Le calcul de l'enveloppe convexe de polyèdres est une opération nécessaire, fréquente et particulièrement coûteuse pour les différentes analyses de PIPS. Cette opération fait en effet appel à l'algorithme de Chernikova pour basculer le système de contraintes en système générateur et opérer l'union plus simplement sur ce système, puis faire l'opération inverse. De plus, les risques de débordements sont élevés au cours de ces opérations, et il est souhaitable de produire une approximation la moins mauvaise possible lorsque ceux-ci ont lieu. Avec Corinne Ancourt et François Irigoin, différentes approches ont été implémentées en décomposant les systèmes en sous systèmes orthogonaux. Ainsi on calcule l'enveloppe convexe (opérateur $\vee$) de deux systèmes de contraintes en extrayant la partie commune orthogonale P' des deux systèmes :

$$P_1 \vee P_2 = (P \cap X_1) \vee (P \cap X_2) = P' \cap ((P'' \cap X_1) \vee (P'' \cap X_2))$$

En cas de débordement au cours de ces calculs, le système P est une approximation de l'enveloppe convexe de P_1 et P_2.

2.3.2 Preuve d'effort

Un autre exemple de mon travail dans le domaine de l'optimisation des performances est le développement de protocoles cryptographiques de preuve d'effort. Le principe de ces protocoles, introduits par Dwork et Naor, est celui du « timbre » : pour obtenir un service sur le réseau, un client doit faire un calcul coûteux dont la vérification sera très rapide par le fournisseur du service. Il s'agit donc de concevoir des fonctions particulièrement dissymétriques dont les performances ne doivent pas pouvoir être optimisées.

La première contribution [40] porte sur des fonctions dont la vitesse est bornée par les performances des *accès mémoire* de la machine, et l'analyse des autres propositions effectuées dans la littérature, avec l'introduction de deux critères d'optimalité pour comparer ces fonctions. La seconde contribution [55] est un résultat plus théorique. Il s'agit d'une fonction de preuve d'effort sans interaction directe entre le demandeur et le fournisseur, pour laquelle la variance de l'effort demandé est quasi nulle. Le troisième point est le développement en OpenCL par Etienne Servais [85], élève du cycle ingénieur de l'École des mines, d'un générateur `hascash` qui lui a permis de prendre le record du plus gros timbre `hashcash`.

Principe

Le principe des preuves d'effort est qu'un serveur requiert de la part d'un client qui demande un service un calcul coûteux pour le client (la résolution d'un problème) mais dont la solution est facile à vérifier, soit que le serveur lui-même construit le problème (résolution interactive d'un défi par le client illustré

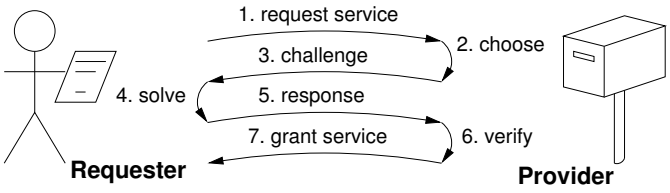


FIGURE 2.26 – Preuve d'effort par défi et réponse

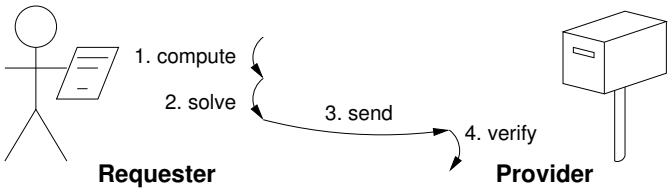


FIGURE 2.27 – Preuve d'effort par résolution et vérification

à la figure 2.26) ou que celui-ci a été construit par le client en fonction du service demandé (résolution-vérification illustré à la figure 2.27). L'objectif de cette preuve d'effort est de limiter la capacité d'un client éventuellement mal intentionné à faire des requêtes et donc de charger le serveur de demandes qui compromettent sa disponibilité. Un exemple d'application qui vient à l'esprit est la limitation des messages électroniques pour lutter contre le *spam*. Ce type d'algorithme est également utilisé par le système de monnaie numérique *Bitcoin* (cf présentation sur Wikipedia) pour valider les transactions financières.

Hashcash en OpenCL

Hashcash est un standard qui a été développé pour générer une preuve d'effort par inversion partielle d'une fonction de hash, en l'occurence SHA1. Le résultat de la recherche combinatoire est une chaîne de caractère qui décrit le service demandé, un message envoyé à telle date à telle adresse électronique, et dont le hash commence par un certain nombre de chiffres binaires à zéro annoncé à l'avance. Lors de ses travaux d'option, Etienne Servais a implémenté une version optimisée en OpenCL qui a ensuite tourné sur une carte graphique. Il a ainsi établi un nouveau record d'inversion partielle. La preuve d'effort suivante permet d'envoyer un message à Calvin le 19 janvier 2038.

```
1:52:380119:calvin@comics.net:::9B760005E92F0DAE
```

Son empreinte cryptographique avec SHA1 commence par 52 bits à zéro :

```
0000000000000756af69e2ffbdb930261873cd71
```

Ces travaux ont été présentés par l'élève à une conférence [85].

Preuve d'effort bornée par les performances mémoire

La première idée amusante et novatrice – qui n'est pas la mienne, mais qui a été un moteur de mon intérêt pour ces travaux – est que l'objectif était de *minimiser* les performances et non pas de les maximiser comme c'est le cas d'habitude. La seconde idée, qui là encore n'est pas de moi, est d'égaliser les capacités des clients en concevant une fonction dont le temps de calcul est borné par les accès mémoire plutôt que par la vitesse intrinsèque du processeur. En effet, tandis que cette dernière valeur évolue exponentiellement selon la loi de Moore depuis bientôt quarante ans, la première reste limitée par la propagation des signaux dans les circuits et évolue donc beaucoup moins dans le temps.

La technique que j'ai proposée, déclinée en différentes variantes, entrelace une combinatoire exponentielle à des accès à un tableau t représentant une fonction ou une permutation aléatoire. La version à base de défis est la suivante : le serveur choisit un point de départ x_0 dans un domaine entier de taille 2^n et effectue ℓ itérations en combinant les résultats d'un accès au tableau avec des entiers (v_i ou w_i) choisis aléatoirement selon un chemin binaire b dans l'espace des solution énumérées :

$$x_i = t(x_{i-1} \oplus (b_i ? v_i : w_i))$$

et calcule enfin une somme de contrôle peu onéreuse sur le chemin suivi : $s = h(x_0 \dots x_\ell)$. À partir de ce défi, le client doit retrouver connaissant le point de départ et les entiers perturbateurs v_i et w_i un chemin binaire qui donne la même somme de contrôle, en général le chemin initial si la somme de contrôle est de taille suffisante. Le coût de la recherche pour le client correspond au nombre de chemins explorés. Il est de l'ordre de 2^ℓ, alors que la vérification peut être faite en recalculant directement la somme de contrôle à partir du chemin binaire proposé par le client, avec un coût de l'ordre de ℓ. Le travail du client est donc exponentiel par rapport à celui du serveur, ce qui est optimal. Un point clef est que les accès de tableaux induisent des défauts de cache, ce qui repose sur le fait que le comportement des itérations

est suffisament imprévisible pour rendre inefficace des techniques de *pre-fetching* : la formule itérative ne doit pas permettre de savoir quel élément va être utile avant d'avoir effectué le calcul de l'itération précédente.

Parallélisation d'une preuve d'effort

Un autre aspect intéressant de ces travaux a été l'analyse des performances de la recherche de solutions pour n'importe quelle technique de preuve d'effort. Les travaux précédents se fondaient sur une limitation des performances par la mémoire : en regardant de plus près les argumentations développées, il apparaissait que, dans l'esprit des auteurs, il s'agissait d'une limitation par la latence, en s'appuyant sur le fait que les calculs itératifs utilisaient des données dépendantes les unes des autres via des accès de tableaux. La chaîne de ces dépendances devant être respectée, le temps de calcul est donc borné par la latence de la mémoire.

Le raisonnement est bien sûr correct, sauf sur un point : si une recherche pour un essai est en effet bornée inférieurement par la latence, rien n'empêche de faire d'autres essais de manière simultanée. En d'autres termes, il s'agit de paralléliser la recherche de solutions du client pour buter non pas sur la latence mémoire, mais sur la bande passante de celle-ci. Cette recherche simultanée est forcément possible car le principe de la preuve d'effort est d'avoir une dissymétrie entre la recherche et la vérification rapide.

Cet argument de parallélisation de la recherche diminue hélas l'intérêt de la limitation par la performance de la mémoire de ces fonctions de preuves d'effort. En effet, si la latence mémoire évolue peu d'une machine à l'autre, la bande passante évolue elle beaucoup plus. Des machines plus chères proposent des bus plus larges et permettent le transfert de volumes plus important par unité de temps.

Étude de la variance des preuves d'effort

Dans l'étude de ces protocoles et fonctions de preuve d'effort, un point particulier a éveillé ma curiosité : celui de la variance de l'effort de la recherche de solution du côté du demandeur, pour les protocoles de type résolution et vérification.

Dans les protocoles de type défi et réponse, il y a une interaction directe entre le fournisseur et le demandeur de service, ce qui permet au fournisseur de choisir un problème dans un ensemble fini, d'en calculer une propriété, et d'obliger le demandeur à retrouver cet élément particulier de l'ensemble en question. L'effort du demandeur est alors naturellement borné supérieurement par la taille de l'espace de recherche.

Par contre, les protocoles de type résolution et vérification ne supposent aucun échange entre le demandeur et le fournisseur. Le demandeur doit donc lui-même se poser un problème et en trouver une solution. La solution couramment adoptée consiste à lier le problème à la description textuelle du service requis, par exemple *envoyer un mail à Calvin le 20 mars 2013*, et à chercher une inversion partielle d'une fonction de hachage appliquée à cette description et un compteur i :

$$h(\text{description}{:}i) \bmod N = 0$$

La description et la valeur du compteur trouvée est envoyée au fournisseur qui pourra aisément vérifier la propriété. En supposant que la fonction de hachage h se comporte comme une fonction aléatoire, le nombre d'itération moyen de cette recherche est N, de même que sa variance, et la recherche n'est pas bornée. La répartition géométrique des succès de la recherche implique des recherches malchanceuses. Par exemple, 2% des recherches requièrent plus de quatre fois la moyenne ($e^{-4} \approx 0.02$).

La question que je me suis posée est donc de savoir si il était possible d'imaginer une fonction intéressante de recherche bornée dans ce type de protocoles de preuve d'effort. La littérature offre une seule fonction bornée qui consiste en une simple formule : le demandeur doit calculer une racine carrée, qui est vérifiée par le fournisseur en calculant le produit. L'effort induit par cette technique n'est cependant pas optimal, contrairement à la méthode précédente, mais reste intéressant.

La nouvelle technique de preuve d'effort introduite pour résoudre cette question est basée sur un arbre de Merkle, *i.e.* un binaire arbre de hachage. L'idée est d'obliger le demandeur du service à calculer *probablement* 90% de cet arbre, alors que le fournisseur n'a qu'une vérification très partielle à faire. Le demandeur sélectionne une petite partie des feuilles de l'arbre pour les envoyer, avec en plus des hash intermédiaires nécessaires à la vérification du calcul, comme illustré par la figure 2.28. Le coût de la vérification pour le receveur est de l'ordre de grandeur du nombre de feuilles sélectionnées. Un nombre

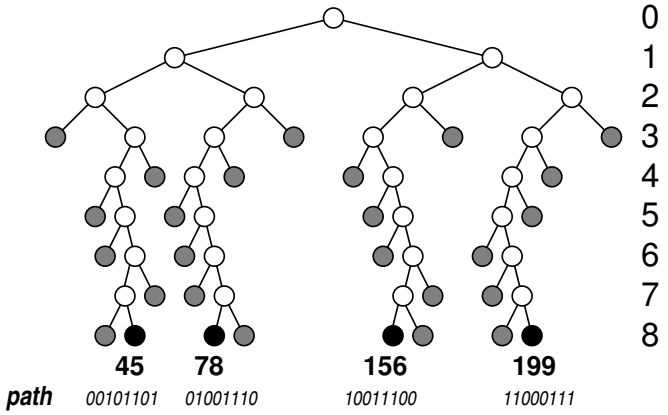


FIGURE 2.28 – Arbre de Merkle pour preuve d'effort

faible de feuilles sélectionnées pseudo aléatoirement de manière non prévisible par le demandeur assure que celui-ci doit avoir calculé l'essentiel de l'arbre pour passer la vérification. Pour que la sélection à opérer ne puisse être connue à l'avance – sinon il serait facile de produire un arbre partiel en calculant uniquement les feuilles demandées – celle-ci est basée sur une suite pseudo-aléatoire initialisée à partir du hash à la racine de l'arbre, de manière à dépendre du calcul complet de celui-ci.

2.3.3 Comparaison de relations à distance

Un dernier exemple d'optimisation parallèle concerne la conception et l'implémentation d'un algorithme de comparaison de tables dans une base de données relationnelle à distance, par exemple pour resynchroniser des données lors de la rupture d'une synchronisation [58, 62]. L'idée est de minimiser les communications en construisant un arbre résumant le hachage de la table entière *(hash-tree)* illustré à la figure 2.29. L'algorithme compare ensuite les sommes de contrôle à partir de la racine, et redescend les branches jusqu'à identifier les tuples différents. La contribution principale est l'utilisation d'un hachage additionnel pour identifier les clefs primaires et regrouper des tuples de manière uniforme et déterministe. L'implémentation disponible sous la forme du logiciel libre `pg_comparator` [36]. Elle permet de comparer et synchroniser les contenus de relations dans des bases de données PostgreSQL, MySQL et SQLite.

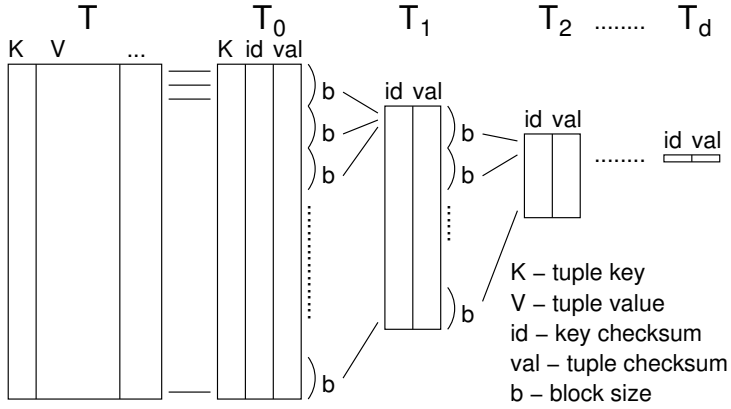


FIGURE 2.29 – Construction d'un arbre de hachage sur une relation

2.4 Conclusion

Ce chapitre a présenté nos contributions dans le domaine de la performance, c'est à dire de l'optimisation (minimisation ou parfois maximisation) d'un critère mesurable (en général le temps, éventuellement

l'énergie ou les communications) dans l'exécution d'un code de calcul intensif. Ces contributions font porter l'effort sur le compilateur par des analyses statiques et de la génération de code adaptée à la cible, mais sont aussi algorithmiques.

Les premiers résultats portent sur la compilation du langage HPF, qui visait à permettre de programmer les supercalculateurs à mémoire distribuée de manière simple et efficace en étendant un langage séquentiel avec des directives qui précisent le parallélisme et le placement des données. La principale contribution est l'application de méthodes polyédriques pour assurer la génération des communications. Malgré les méthodes de compilation proposées par les chercheurs, ce langage ne sera pas un succès, les performances apportées par les compilateurs n'étant pas à la hauteur des espérances. Les raisons de cet échec sont analysées au prochain chapitre.

La seconde partie concerne l'optimisation algébrique des expressions flottantes ou entières, avec la Thèse de Julien Zory. Elle a été motivée par nos résultats sur HPF, en particulier la nécessité d'améliorer les codes de communication, mais aussi les formules calculées par l'application. En effet, le coût des super-calculateurs étant très élevé, et leur programmation parallèle difficile, il est important d'exploiter au mieux les capacités de chaque processeur disponible. Nos résultats ont montré le potentiel de ces optimisations agressives. Les méthodes proposées fonctionnent en exploitant au mieux le parallélisme interne des processeurs et en limitant la pression sur les registres. Il est possible d'étendre ce type d'approche à d'autres domaines, par exemple celui des règles de filtrage des parefeu, ou encore les applications embarquées de traitement d'image discutées ci-après.

La troisième partie de nos contributions concerne le monde des applications embarquées, pour des applications de traitement du signal et de traitement d'image. Dans ce contexte particulier, le soucis de vitesse est modulé par la limitation de l'énergie utilisée, en particulier dans des contextes où l'appareil est alimenté par une batterie. Le travail de la thèse Youcef Bouchebaba porte sur la génération de code prenant en compte complètement la hiérarchie mémoire. Plus récemment mon travail au sein du projet FREIA s'est focalisé sur le placement d'applications de traitement d'image sur des accélérateurs spécialisés en manipulant directement les expressions de calcul. Une partie du projet de recherche présenté au dernier chapitre propose de poursuivre cette voie : il s'agit d'ajuster un processus de compilation à un domaine d'application particulier pour tirer un meilleur profit des cibles matérielles, sans pour autant devoir se résoudre à une programmation manuelle.

Les dernières contributions présentées sont plus diverses et opportunistiques, souvent liées à mes domaines d'enseignement, mais restent liées à l'optimisation des performances et à parallélisation. Il s'agit de simplification de systèmes linéaires utilisés par PIPS pour des analyses (transformers, préconditions, régions de tableau) et la génération de codes (communications HPF), afin de réduire les coûts des opérations. Mes contributions sur les fonctions cryptographiques de preuve d'effort visent à maximiser le temps d'éxécution, et discutent les possibilités de parallélisation de ces fonctions. L'algorithme de comparaison de données relationnelles réduit les communications en calculant des résumés hiérarchiques.

Chapitre 3

Élégance

Qualité esthétique qu'on reconnaît à certaines formes naturelles ou créées par l'homme dont la perfection est faite de grâce et de simplicité.
(dictionnaire Le Petit Robert*)*

Le second volet de cette présentation aborde une dimension plus esthétique. En plus d'aller vite, on souhaite que les solutions proposées ou leur preuves soient simples, générales, complètes, optimales, en un mot si possible élégantes. Évidemment, ce n'est pas parce qu'on vise un objectif que celui-ci est nécessairement atteint.

Nous discutons d'abord à la section 3.1 de la spécification du langage HPF et de nos contributions à celle-ci, puis des raisons qui ont conduit à l'échec du langage, en illustrant les difficultés à l'aide de l'application Onde24 déjà décrite au chapitre précédent. La section 3.2 discute du développement ambitieux de stratégies globales de compilation, à la fois dans le cadre du projet FREIA et dans celui des travaux des thèses que j'ai encadrées et auxquels j'ai participé. La section 3.3 décrit ensuite nos tentatives pragmatiques de changer l'approche usuelle de certains problèmes combinatoires ou simplement complexes d'un point de vue ingénierie et requérant néanmoins une réponse effective. Enfin, la section Évidences 3.4 discute la constructions de certaines preuves.

3.1 HPF grandeurs et décadences

Lorsque HPF paraît, la famille du calcul scientifique applaudit à grand cri ce nouveau langage qui va permettre de développer des codes portables et néanmoins performants sur des machines à mémoire distribuée. Cependant, comme nous l'avons déjà indiqué section 2.1.2, ce langage est un langage de comité où les négociations sont dures et les décisions prises à la majorité, quelles que soient les contradictions entre les différentes décisions.

Deux camps se sont affrontés. D'un côté les utilisateurs souhaitaient que toutes les applications rentrent dans un langage où le compilateur s'occupera de tout, en compilation séparée, et avec le maximum d'aspects définissables à l'exécution. De plus, dans un marché très hétérogène, ils souhaitent ne pas perdre leur investissements logiciels si ceux-ci ne sont pas portables d'une machine à l'autre. De l'autre les développeurs de machines et de compilateurs essayaient de restreindre le plus possible, avec parfois une certaine naïveté sur des points clefs inutilement contraints et un manque d'imagination, ce point étant illustré ci-après par la question des déclarations explicites des bords de tableau à allouer localement.

Le langage obtenu est un mauvais compromis, qui ne satisfera ni les uns ni les autres. Il est difficilement compilable car au delà de l'état de l'art, il ne fournit pas suffisamment d'informations statiques au compilateur pour faire un bon travail. Il ne convient malgré tout pas à des applications très simples. Il n'est pas sûr qu'un bon compromis existait et aurait permis d'aboutir à un langage à l'objectif très ambitieux : permettre la portabilité en performance sur différents super-calculateurs parallèles à mémoire distribuée avec un modèle de programmation à fil unique d'exécution *(single thread)* et une vision unifiée de la mémoire. Il est certain que l'état atteint n'était pas optimal, comme nous allons le montrer dans

les deux prochaines sections. Un article de Ken Kennedy, Charles Koelbel et Hans Zima (The Rise and Fall of High Performance Fortran: an Historical Object Lesson, 3rd ACM conference on History of Programming Languages, 2007) discute, vu de ces acteurs très impliqués, les cinq raisons de l'échec du langage : l'immaturité des compilateurs qui conduit à de mauvaises performances, le manque d'expressivité du placement des données, des implémentations incompatibles au moins au plan performance, le manque d'outils, l'impatience de la communauté. Pour tous ces différents problèmes, les auteurs suggèrent qu'un investissement fédéral massif aurait pu apporter des solutions. Je donne ici ma propre analyse de cet échec qui diffère en partie de cette vue.

3.1.1 Retours sur la conception de HPF

Dans cette empoignade, j'ai milité à distance en intervenant sur les listes de discussions, avec un succès limité, pour une définition statique de HPF qui permette au compilateur de profiter du maximum d'informations disponibles, sans pour autant contraindre inutilement les possibilités d'expression offertes au programmeur. Dans un secteur moins essentiel mais cependant esthétiquement important, j'ai essayé de contribuer à créer un langage homogène, orthogonal et complet. Cette vue est défendue dans un article [26]. HPF, s'il avait été un succès, aurait été un très délicat point d'équilibre entre expressivité pour les utilisateurs et implémentabilité par les fournisseurs.

Assume vs Inherit

Un premier exemple de complication vient des paramètres de fonctions lorsque ceux-ci sont distribués. Dans un souci de complétude, la définition de HPF comprends une directive `INHERIT` qui exprime que le placement d'un tableau est à prendre tel qu'il est : le compilateur ne dispose d'aucune information et doit se débrouiller pour générer un code correct à défaut d'un code performant.

Outre le fait que cette directive ne sert à rien pour l'aspect « Haute Performance » de HPF et qu'elle complique sérieusement la tâche du compilateur et de l'environnement d'exécution, il n'y a aucun moyen en HPF de donner au compilateur l'information qui permettrait par exemple de dériver des variantes efficaces pour différents contextes de placement des tableaux passés à la routine.

Une difficulté de la compilation de HPF est liée à ce placement hérité. C'est à la fonction appelée, qui n'a pas d'information sur ce qui lui arrive (inherit indique fondamentalement que l'environnement d'exécution doit gérer n'importe quel placement), de s'arranger avec le placement des paramètres, dans un contexte de compilation séparée. Une meilleure approche aurait au contraire été que cela soit du ressort de l'appelant, puisqu'il dispose à la fois de l'information sur le placement des arguments et sur le placement attendu par la fonction via une déclaration de prototype. Cependant, le langage HPF ne requérait pas de déclaration de placement dans l'interface des routines.

Pour renforcer les informations données au compilateur et potentiellement remplacer ce placement hérité au profit de placements prescriptifs ou descriptifs dans un contexte de compilation séparée, nous avons proposé [76] une directive `ASSUME` pour déclarer au compilateur les différents contextes de placements et lui permettre de générer du code optimisé spécifiquement pour ces contextes.

```
      subroutine matmul(A, B, C)
      real, dimension(:,:):: A, B, C
!hpf$ ASSUME
!hpf$   ALIGN WITH A:: B, C
!hpf$   DISTRIBUTE A(BLOCK, BLOCK)
!hpf$ OR
!hpf$   ALIGN WITH A:: B, C
!hpf$   DISTRIBUTE A(*, BLOCK)
!hpf$ OR
!hpf$   ALIGN WITH A:: B, C
!hpf$   DISTRIBUTE A(BLOCK, *)
!hpf$ OR
!hpf$   NOTHING
!hpf$ END ASSUME
      ...
      end subroutine
```

```
!hpf$ independent
      do i=1, n
!hpf$   new
        x = A(i)
        B(i) = x + x*x
!hpf$   reduction
        s = s + x
      end do
```

```
!hpf$ independent(i), new(x), reduction(s)
      do i=1, n
        x = A(i)
        B(i) = x + x*x
        s = s + x
      end do
```

FIGURE 3.1 – Styles de directives interne et externe

Dans l'exemple ci-dessus, une fonction peut avoir trois types de placements pour ses paramètres tableaux, ou aucun, décrivant ainsi des données potentiellement locales pour des tableaux non distribués.

En sus de cet ajout, nous avons décri une technique pour traduire simplement un programme utilisant cette directive en un programme à placement statique et unique équivalent. Elle consiste à cloner la routine en autant d'exemplaires que de placements particuliers déclarés. Malgré ses qualités pratiques, cette proposition n'a pas suscité l'adhésion du forum et il l'a rejetée. Pourtant, des approches qui offrent un maximum d'information statique au compilateur auraient pu aider HPF à atteindre les hautes performances visées.

Replacements statiques

Une autre erreur de conception qui a empêché l'utilisation de HPF pour certaines applications a été la décision de retrait en 1996 du cœur du langage HPF 2.0 des replacements, devenus de simples *extensions approuvées*, ce qui correspondait à un enterrement de première classe.

L'argument principal de ce retrait était la difficulté – réelle – de compiler des replacements dans leur définition générale. En effet, les directives de replacement pouvant être placée à n'importe quel point de l'exécution du programme, un tableau pouvait avoir en un point donné des placements différents selon le flot suivi par l'exécution, ce qui compliquait singulièrement la tâche du compilateur.

En analysant mieux le problème sous l'angle de la compilation d'une part, et sous l'angle des applications d'autre part, il apparaît [31] que ces cas correspondent plus à des bugs qu'à des exemples pertinents, et auraient dû être rapportés comme tels à l'utilisateur, sans même tenter une compilation de toute façon inefficace. En conséquence, la solution simple que nous avons proposée est plutôt d'ajouter la contrainte que le placement des tableaux doit être connu directement et statiquement dans le programme, à la fois au niveau des appels de routine et à l'intérieur d'une routine, au moyen d'une simple analyse structurelle du code. Sous cette condition, nous avons montré que les replacements peuvent être compilés efficacement en étant traduits sous une forme statique avec de simples copies entre des tableaux (section 2.1.5). Par ailleurs, notre méthode permet de détecter facilement des opportunités d'optimisation supplémentaires, comme par exemple conserver un tableau sous plusieurs formes distribuées s'il n'est utilisé qu'en lecture dans l'application.

Le résultat de cet épisode est qu'un aspect utile du langage, les replacements, pertinents par exemple pour une simple multiplication de matrice, et tout à fait compilable, a été sorti de HPF.

Déclaration des réductions

Un exemple personnel de contribution à la définition syntaxique du langage HPF a été l'homogénéisation du style des directives de déclaration de propriétés sur les variables scalaires d'une boucle. Mon analyse [26] n'a cependant pas suscité une adhésion totale.

HPF 1.0 a prévu une directive `NEW(...)` associée à la déclaration d'indépendance des itérations d'une boucle qui permet de préciser le traitement d'un scalaire à l'intérieur de cette boucle. Lors des discussions sur HPF 2.0, il était proposé une nouvelle directive `REDUCTION` permettant de déclarer une telle sémantique sur une opération dans une boucle, dans un style interne.

Cependant, une analyse plus précise montre que le traitement réellement souhaité est identique à celui des variables privées aux itérations, et qu'il n'y a pas de raison de différencier les syntaxes. De plus, on se rend compte que deux styles (figure 3.1) homogènes permettent de spécifier des propriétés sur des variables, soit en précisant l'instruction par un commentaire, soit en précisant les variables concernées pour une zone donnée.

Il s'avère que ces déclarations de propriétés sont utiles non seulement pour des boucles, mais également pour des séquences hors de toute boucle. Elles permettent par exemple de préciser le caractère privé de certains scalaires pour des calculs aux coins d'un domaine sur une méthode de différences finies. Il y a donc besoin d'une directive qui permette de définir la portée de ce genre de directive. Une telle directive a fini comme simple extension approuvée dans la définition de HPF 2.0 malgré son caractère indispensable à des codes réalistes.

Déclaration des bordures

Un dernier exemple des errements des discussions sur la définition du langage HPF est la proposition de déclaration explicite des ombrages *(shadow width)* ou bordures des tableaux. Ces zones permettent de stocker la copie des données des processeurs voisins lorsque celles-ci sont nécessaires à un calcul. Elles simplifient grandement l'adressage des éléments distants récupérés via des communications en les unifiant au stockage des éléments locaux.

Parce que les développeurs des compilateurs ne savaient pas comment calculer ces bordures dans le cadre d'une compilation séparée, il était proposé de rajouter une syntaxe permettant leur déclaration explicite. Afin de tenter de couper court à cette approche, j'avais décrit [27, 25] comment calculer cette information au lancement du programme. La technique s'appuie sur des fonctions simples de support générées à la compilation, et qui permettent de résoudre dynamiquement l'équation du calcul de cette bordure en propageant les informations dans l'arbre d'appel. Cette technique convient très bien à un cadre de compilation séparée, mais n'a cependant pas réussi à dissuader le comité d'inclure l'extension `SHADOW` dans la définition de HPF 2.0.

3.1.2 Analyse de l'échec de HPF

La fin de mes travaux de thèse a coïncidé avec la définition de HPF 2.0, dernière tentative du forum pour sauver HPF, en limitant les aspects du langage d'une part pour aider les implémenteurs, et en l'étendant d'autre part pour satisfaire les besoins pressants des utilisateurs, en particulier dans le domaine des calculs irréguliers comme ceux des méthodes à base d'éléments finis. Les compilateurs HPF ne délivraient pas les performances attendues. La seule façon d'obtenir des performances correctes sur les calculateurs à mémoire distribuée semble alors d'optimiser à la main les codes et les communications. Par ailleurs, la santé économique chancelante des différents acteurs allait bientôt se concrétiser par une série de faillites et de rachats. En bref, les choses n'allaient pas très bien.

L'alternative à HPF dans le domaine des machines parallèles à mémoire distribuée reste le passage de message programmé manuellement, avec la bibliothèque de gestion explicite des communications MPI *(Message Passing Interface)*. À défaut d'offrir une facilité de programmation, ces bibliothèques permettent d'obtenir de bonnes performances, tout en restant portable, même si la portabilité des performances est encore une autre question. Par ailleurs, au niveau des machines à mémoire partagée, où la problématique de l'adressage et du partage de la mémoire est à la charge du matériel et du système d'exploitation, il faut profiter du parallélisme interne de haut niveau des processeurs *(hyper threading, multi-core)*. Pour cela, le standard OpenMP *(Open Multi-Processing)* permet au développeur de contrôler le parallélisme de tâche ou de données et les synchronisations, là encore de manière standardisée.

Je commençai à ce moment à travailler avec Julien Zory [87], qui se consacrait entre autre au test en HPF de l'application Onde24 de l'IFP (Institut français du pétrole) déjà présentée à la section 2.2.1. Le portage et l'exécution de Onde24 était effectué sur plusieurs compilateurs et architectures : le compilateur HPF de IBM pour SP2, le compilateur Adaptor de Thomas Brandes (membre de mon jury de thèse, et avec qui Julien avait travaillé en DEA), et enfin mon propre prototype de compilateur HPF. Ce code est l'exemple même d'un petit code réaliste et néanmoins trivialement parallèle : il comprend quelques entrées-sorties et fait des opérations simples sur de larges matrices avec des accès aux voisinages. Aucune technique avancée n'est normalement nécessaire à sa compilation. Les efforts qu'a dû déployer Julien pour obtenir des temps d'exécution proches des performances attendues sur la machine étaient sans proportion par rapport à la taille et à la simplicité du code.

Les points d'achoppement comprenaient :

- l'utilisation de vecteurs temporaires pour les calculs sur les bords, qui doivent être multipliés en versions Ouest, Sud et Est et placés indépendemment, comme illustré à la figure 3.2 en orange ;

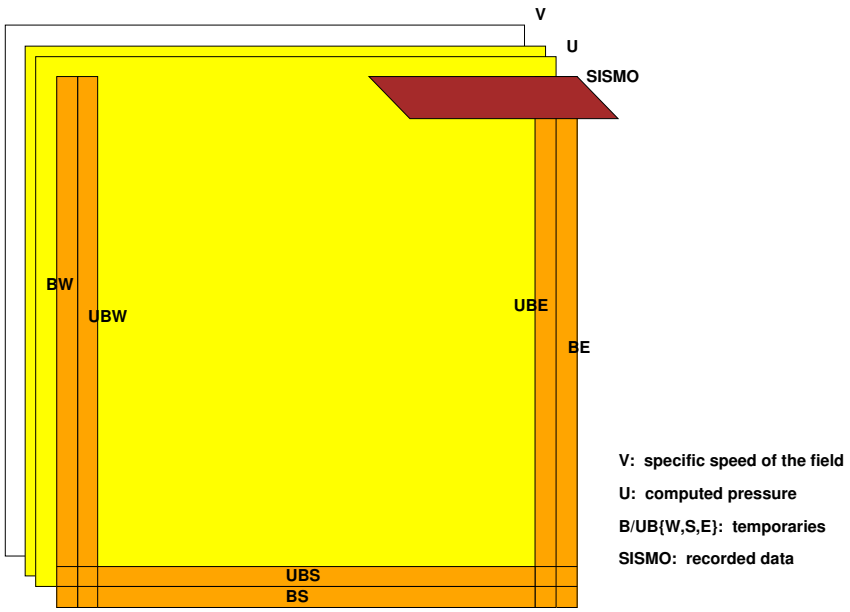


FIGURE 3.2 – Onde24 – Placements HPF

- les entrées-sorties réalisées à l'intérieur d'une boucle, qui nécessitent soit une analyse de région de tableau, soit une réécriture sous forme de boucle implicite ou de notation de tableau ;
- les calculs aux quatre coins du domaines qui impliquent des déclarations de variables privées hors de toute boucle ;
- la présence de réductions qui n'est pas supportées par tous les compilateurs.

Bref, beaucoup d'efforts pour des résultats somme toute médiocres, malgré des connaissances et un support avancé dans le domaine de la compilation, ce qui augurait mal de l'utilisation de HPF dans un milieu industriel moins pointu. Les compilateurs HPF ne tenait simplement pas les promesses du langage.

Le compromis essentiel déjà décrit entre expressivité et performance induit des conséquences importantes sur le développement des compilateurs. En effet, un compilateur se doit d'abord de générer un code correct qui respecte la sémantique du langage. Pour HPF, cela implique la compilation de Fortran 90 complet, sur lequel s'appuie la spécification, dans un contexte où les tableaux de l'application peuvent être distribués n'importe où sur une machine parallèle. Les performances ne viennent que dans un second temps. Pour un petit marché comme celui du calcul scientifique, avec quelques (grosses) machines, quelques développeurs, quelques chercheurs, et quelques applications clefs, le coût était excessif.

Les espoirs sous-jacents de l'approche étaient que le développement des compilateurs serait un succès. Un code scientifique initialement développé dans un cadre séquentiel serait adaptable facilement et efficacement à HPF au moyen de quelques directives ajoutées ça et là, qui conviendraient à toutes les implémentations, les compilateurs tous performants étant plus ou moins interchangeables.

L'approche réaliste aurait été de décider très rapidement qu'un code HPF était forcément un code spécial, qui vise des performances exceptionnelles sur un matériel exceptionnel, et qu'il demandera des efforts particuliers pour les développeurs de l'application. La performance était la seule raison d'être de ces machines et du langage. De plus, il fallait aider les implémenteurs des compilateurs du langage en leur donnat un objectif atteignable. Dans ce cadre, ce qui importait n'était pas qu'un code Fortran quelconque puisse être accepté par un compilateur HPF, mais plutôt qu'un code HPF développé exceptionnellement puisse *aussi* fonctionner dans un cadre séquentiel classique. Donc que HPF soit dans Fortran, et non pas que tout Fortran soit dans HPF.

En terme de stratégie de développement du langage, il aurait donc été pertinent de commencer modestement, et d'ajouter à l'ambition au fur et à mesure que les techniques de compilations s'amélioraient, en suivant de près la recherche, au lieu de poser des défis inaccessibles en définissant un langage limité en informations statiques disponibles pour la compilation, mais exigeant en résultat sur les performances.

Évidemment, il est facile de dire tout cela après coup. De plus, même si ces magnifiques conseils avaient été suivis, leur réalisme technique n'aurait en rien garanti le succès de HPF : ce possible décrit ci-dessus n'aurait sans doute pas été suffisant du point de vue des utilisateurs pour assurer la pérennité et la profitabilité des investissements nécessaires dans des applications développées spécifiquement.

3.2 Stratégies de compilation

Cette section illustre le développement d'une chaîne complète de compilation, en s'appuyant sur une stratégie globale qui enchaîne différentes techniques. L'originalité n'est pas dans les méthodes particulières utilisées, qui sont en général classiques et pas nécessairement optimales, mais au contraire dans la sélection de méthodes simples et leurs mises en œuvre ensembles de manière cohérente pour atteindre un objectif ambitieux. Il s'agit de trouver une solution simple et pratique, si possible élégante, à un problème de compilation global et complexe.

3.2.1 Compilation FREIA

L'environnement logiciel de portage et d'optimisation des applications du projet FREIA introduit à la section 2.2.4 a le double objectif d'obtenir de très bonnes performances mais aussi de développer le compilateur avec un investissement logiciel faible. La chaîne de compilation développée est décomposée en trois phases détaillées à la figure 3.3 :

préparation du code : à partir du source de l'application, les fonctions de haut niveau de l'API sont expansées, les constantes propagées, les boucles déroulées, le code simplifié, de manière à obtenir des blocs de base contenant des longues séquences d'opérations de base sur des images. La figure 3.4 montre le résultat de cette phase sur l'exemple de la figure 2.25 : après *inlining* et simplifications, il ne reste plus que des opérateurs de base (AIPO) dans une seule séquence.

optimisation des expressions : à partir des séquences d'opérations images, on construit et optimise les expressions de calculs sur images, sous la forme de DAG (graphes orientés acycliques). Le DAG extrait de l'application après pré-traitement est exposé à la figure 3.5. La version optimisée à la figure 3.6 montre que la dilatation répétée 10 fois (en bleu), initialement cachée par deux niveaux d'appels de fonctions, a été reconnue comme une expression commune et n'est plus calculée qu'une seule fois.

générateurs de code : les graphes sont ensuite traduits pour les matériels cibles par trois générateurs de codes, spécifiques à chaque cible.

Les générateurs de code s'appuient sur des heuristiques simples pour découper et ordonnancer les opérations image sur les matériels cibles très différents. Des heuristiques sont utilisées afin de résoudre de manière pratique les problèmes NP-complets sous-jacents :

SPoC : accélérateur fondé sur un pipeline vectoriel d'opérations image qui peut combiner plusieurs dizaines d'opérateurs en une seule passe, avec comme contrainte principale le fait que seules deux images peuvent être envoyées dans le pipeline et deux extraites à l'autre bout. La figure 3.7 présente le placement effectué pour l'application ANR999. Les opérations remplissent autant que possible le pipeline matériel. Deux appels à l'accélérateur sont nécessaire pour effectuer le calcul complet.

Terapix : accélérateur fondé sur un tableau SIMD de 128 processeurs synchronisés disposant d'une mémoire locale limitée. Comme la mémoire disponible est petite, il est nécessaire de découper les images avec un mécanisme de tuilage. La figure 3.8 montre le découpage des opérations de l'application ANR999 sur cette cible. Les coupes verticales permettent d'équilibrer les calculs et les communications tout en réduisant les calculs redondants qui doivent être effectués dans les zones inter-tuiles. Le compilateur choisit également l'allocation mémoire et l'ordonnancement des opérations – non visible sur la figure des tâches – qui sera exploité par l'environnement d'exécution pour effectuer les calculs.

OpenCL est une cible générique pour utiliser des cartes graphiques et des processeurs multi-cœurs avec une mémoire partagée. Il propose un modèle de parallélisme de tâches data-parallèles. La

Phase 1 application preprocessing – enlarge basic blocks

- **1.** simplification of safety checks
- **2.** inlining of FREIA high-level library functions
- **3.** constant propagation
- **4.** loop full unrolling
- **5.** only for SPoC : convergence while unrolling
- **6.** control simplification
- **7.** code flattening

Phase 2 DAG optimizations

- **1.** DAG construction per basic block
- **2.** operator normalization *(improve CSE stage)*
- **3.** algebraic optimizations : constant image detection and propagation
- **4.** common sub-expression elimination (CSE), with commutativity and reductions
- **5.** dead image operation removal
- **6.** forward and backward copy propagation
- **7.** extraction of remaining copies

Target-specific back-ends for SPoC, Terapix and OpenCL

Phase 3.1 SPoC configuration : map DAG onto hardware

- **1.** DAG splitting and scheduling of sub-DAGs
- **2.** instruction compaction and path selection
- **3.** pipeline overflow management
- **4.** unused image cleanup and image reuse

Phase 3.2 Terapix configuration : map DAG onto hardware

- **1.** DAG splitting along scalar dependencies and connected components
- **2.** scheduling and memory allocation, including double buffers
- **3.** instruction generation, best tile size selected at runtime
- **4.** unused image cleanup and image reuse

Phase 3.3 OpenCL code generation

- **1.** DAG splitting along scalar dependencies and connected components
- **2.** operator aggregation
- **3.** OpenCL kernel generation
- **4.** unused image cleanup and image reuse

FIGURE 3.3 – Stratégie de compilation de FREIA

```
freia_aipo_global_min(in, &min);     // some computations
freia_aipo_global_vol(in, &vol);
freia_aipo_dilate_8c(od, in, k8c);
freia_aipo_dilate_8c(od, od, k8c);
freia_aipo_dilate_8c(od, od, k8c);
freia_aipo_dilate_8c(od, od, k8c);
freia_aipo_dilate_8c(od, od, k8c);
freia_aipo_dilate_8c(od, od, k8c);
freia_aipo_dilate_8c(od, od, k8c);
freia_aipo_dilate_8c(od, od, k8c);
freia_aipo_dilate_8c(od, od, k8c);
freia_aipo_dilate_8c(od, od, k8c);
I_0 = 0;                             // scalar stuff...
tmp = freia_common_create_data(...); // image allocations...
freia_aipo_dilate_8c(tmp, in, k8c);
freia_aipo_dilate_8c(tmp, tmp, k8c);
freia_aipo_dilate_8c(tmp, tmp, k8c);
freia_aipo_dilate_8c(tmp, tmp, k8c);
freia_aipo_dilate_8c(tmp, tmp, k8c);
freia_aipo_dilate_8c(tmp, tmp, k8c);
freia_aipo_dilate_8c(tmp, tmp, k8c);
freia_aipo_dilate_8c(tmp, tmp, k8c);
freia_aipo_dilate_8c(tmp, tmp, k8c);
freia_aipo_dilate_8c(tmp, tmp, k8c);
freia_aipo_erode_8c(og, in, k8c);
freia_aipo_erode_8c(og, og, k8c);
freia_aipo_erode_8c(og, og, k8c);
freia_aipo_erode_8c(og, og, k8c);
freia_aipo_erode_8c(og, og, k8c);
freia_aipo_erode_8c(og, og, k8c);
freia_aipo_erode_8c(og, og, k8c);
freia_aipo_erode_8c(og, og, k8c);
freia_aipo_erode_8c(og, og, k8c);
freia_aipo_erode_8c(og, og, k8c);
freia_aipo_sub(og, tmp, og);
```

FIGURE 3.4 – Extrait de ANR999 (figure 2.25) après pré-traitement

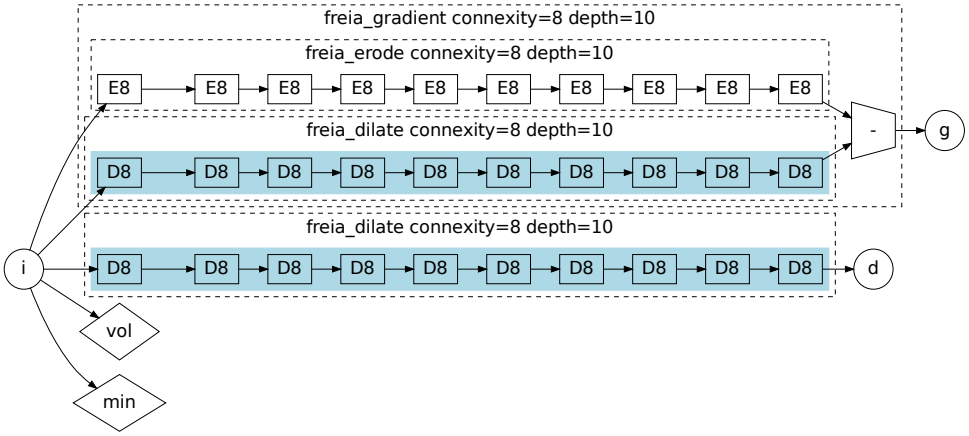


FIGURE 3.5 – DAG initial de l'application ANR999

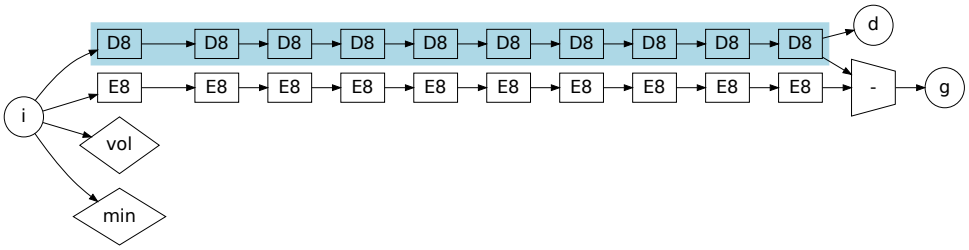


FIGURE 3.6 – DAG optimisé de l'application ANR999

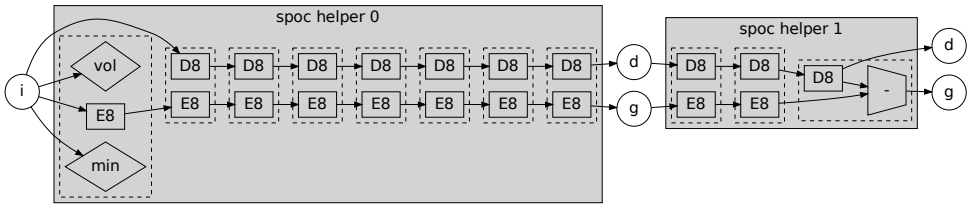


FIGURE 3.7 – Placement du DAG de l'application ANR999 sur SPoC

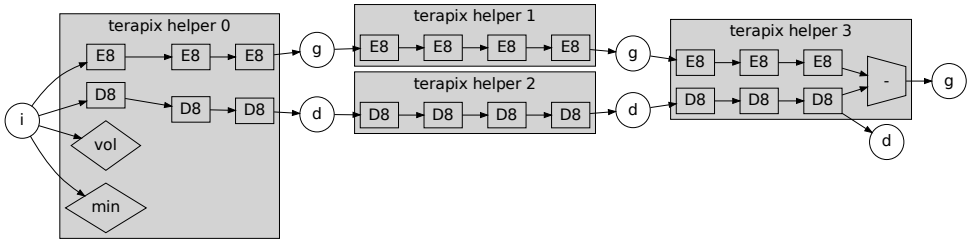


FIGURE 3.8 – Placement du DAG de l'application ANR999 sur Terapix

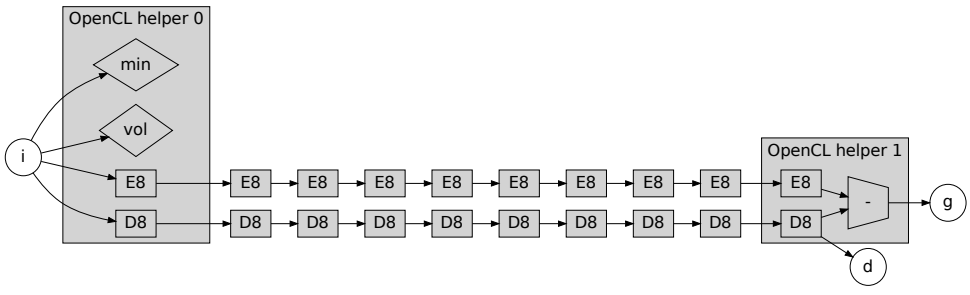


FIGURE 3.9 – Placement du DAG de l'application ANR999 sur OpenCL

figure 3.9 montre les tâches générées pour OpenCL. Quelques opérateurs sont combinés, et des versions spécialisées à la valeur du noyau de calcul sont générées pour les érosions (E8) et les dilatations (D8).

L'ordonnancement des tâches pour ces trois cibles, ainsi que la génération de l'allocation pour Terapix, s'appuie sur une même technique heuristique d'ordonnancement des instructions à partir d'une séquence d'instructions appelée *list scheduling* dans le *Dragon Book*. Il s'agit simplement de prendre les opérations dans l'ordre des dépendances, et de choisir à chaque étape l'opération à effectuer avec un tri. La seule « intelligence » de la méthode réside dans le tri qui peut localement favoriser la résolution de telle ou telle contrainte en fonction des spécificités de l'architecture. Par exemple pour SPoC, privilégier la consommation des images permet de limiter le nombre d'images vivantes et donc d'avoir moins d'interruptions du pipeline ; pour Terapix, l'accent est mis sur la meilleure réutilisation de la mémoire qui est en général le point critique. Dans tous les cas, ces méthodes prennent des décisions locales sans garantie d'optimalité globale, et obtiennent en pratique des résultats excellents, souvent optimaux.

3.2.2 Discussion

Au début du projet FREIA, l'idée était de se pencher séparément sur des aspects gros-grain (opérations) et moyen-grain (tuilage) automatiques, en s'appuyant sur les transformations déjà implémentées dans l'outil PIPS. D'autres phases de génération de code ont été étudiées par THALES pour SPEAR-DE (un environnement graphique spécifique) et par l'ENST-Bretagne pour générer un code assembleur spécifique de l'accélérateur SIMD Terapix.

La partie gros-grain consiste à ajuster automatiquement les expressions de tableaux de l'application de traitement d'image en fonction des objectifs d'optimisation et du matériel. L'inspiration initiale était de reprendre les techniques de manipulation à base de réécriture développées dans PIPS lors de la thèse de Julien Zory, en les adaptant à ce nouveau contexte où les données ne sont pas de simples scalaires mais des tableaux denses, impliquant en particulier des contraintes de stockage. On pense en particulier à des optimisations classiques de compilation comme l'élimination de code mort, la factorisation d'expressions communes ou l'extraction d'invariant de boucles. Cette phase implique donc une modélisation des matériels cibles pour guider les décisions de configuration et reconfiguration matérielles, ainsi que pour prendre en compte des transferts mémoire nécessaires, ou au moins d'en instruire l'environnement d'exécution. Cependant, en pratique, les opportunités d'appliquer de telles optimisations se sont avérées très limitées, comme nous nous y attendions vus les exemples d'applications disponibles. Par contre, la combinatoire réduite a permis de valider rapidement la stratégie globale de calcul, par exemple en décidant certaines fusions de boucles ou au contraire une scission des calculs en fonction des possibilités du matériel sous-jacent, en particulier si on emploie un matériel reconfigurable ou si les possibilités du silicium limitent les opérations qui peuvent être effectuées en une passe. Les résultats obtenus ont validé ces intuitions.

L'idée de la partie optimisation grain moyen était de réutiliser les techniques de compilation appliquées au traitement du High Performance Fortran durant ma thèse et celle de Youcef Bouchebaba au contexte du traitement d'image et à ces matériels, en particulier l'accélérateur SIMD Terapix. Les transformations envisagées incluaient le tuilage, la fusion de boucles et l'inclusion en ligne de procédures, avec pour objectif de générer les transferts mémoire. À la fin du projet, cette phase a complètement disparu : il est intéressant d'en analyser la raison. Le matériel impose des contraintes telles que le degré de liberté pour le tuilage est en pratique nul, ce qui a amené à déléguer celui-ci à l'environnement d'exécution, en le guidant juste pour l'allocation mémoire et l'ordonnancement des opérations locales sur les tuiles dans l'accélérateur Terapix. Ces deux dernières décisions peuvent être prises directement au niveau du DAG des calculs d'images à opérer, donc sur la même structure de données et le même niveau de détail que la phase gros grain décrite précédemment qui ciblait initialement l'accélérateur SPoC.

La génération pour la cible OpenCL partage du code en terme de découpage du graphe d'expressions avec la cible Terapix. Par contre les aggrégations de tâches effectuées est différente, de même que la nécessité de générer un noyau de calcul spécifique qui combine éventuellement plusieurs opérations images.

Suivre une telle approche globale, qui combine de manière plutôt pragmatique de nombreuses techniques pour compiler un domaine vers une architecture particulière, a des conséquences significatives pour le chercheur :

- Tout d'abord, les publications sont plus difficiles, car en général il y a peu ou pas d'améliorations

d'une transformation particulière utilisée, l'originalité étant dans la combinaison effective de techniques déjà connues, et la solution atteinte n'est pas nécessairement optimale ;

- L'effort d'implémentation nécessaire pour montrer l'efficacité de la méthode est très important, si on doit le comparer à un chercheur qui développe un algorithme dans un cadre ou pour un problème plus contraint ;
- En général, on cherche à compiler des applications complètes, par opposition à traiter optimalement un cas simple : lors des expériences on est confronté à de nombreux aspects (syntaxe, entrées-sorties...) rencontrés dans une application réelle qui compliquent fortement les expériences.

Malgré cela, on a, si cela fonctionne à la fin, la satisfaction d'avoir résolu de manière pratique et concrète un problème complet. Les travaux de manipulations d'expressions associatives-commutatives dans le cadre de la thèse de Julien Zory [87] et présentés à la section 2.2.2 sont un exemple d'approche globale qui combine de nombreuses techniques heuristiques simples pour résoudre un problème complexe. La thèse de Mehdi Amini [2] que j'ai co-encadrée est aussi une bonne illustration de la mise en œuvre d'une approche globale pour résoudre un problème complexe en utilisant tous les outils disponibles, pour lui déplacer autant de calculs que possible d'une application scientifique vers un GPGPU.

3.3 Pragmatisme

L'objectif même de la recherche est de remettre en cause les croyances, les compromis, les idées qui nous sont transmises, afin d'essayer d'en comprendre les limites, les simplifications abusives, et de créer son propre champ d'exploration, voire faire éventuellement des compromis différents pour résoudre les problèmes abordés. La première section 3.3.1 discute des aspects combinatoires abordés principalement lors des travaux de Julien Zory. La second 3.3.2 discute de la philosophie de renversement du paradigme de compilation avec les travaux de Youcef Bouchebaba : l'objectif de performance permet éventuellement de prendre quelques raccourcis avec la correction des transformations, par exemple en négligeant la justesse des calculs sur les bords de l'image. La troisième 3.3.3 discute de la co-conception de tout un éco-système lors du projet FREIA déjà présenté, avec des transferts de responsabilités entre langage, compilateur, exécutif et matériel. La quatrième et dernière section 3.3.4 aborde l'objectif paradoxal des fonctions de preuves d'effort bornées par les performances mémoire, qui cherchent à *maximiser* les défauts de cache.

3.3.1 Affronter une combinatoire

Un premier exemple d'une telle démarche est celle qui a conduit à toute la partie optimisation associative-commutative (AC) de la thèse de Julien Zory. En quoi cette approche remet-elle en cause l'état de l'art ?

Propriétés des opérations informatiques

Un frein particulier aux manipulations algébriques est que les types de données informatiques, entiers ou flottants, ont des capacités limitées par rapport à leurs homologues mathématiques. Les opérations implémentées sur ces types sont en général commutatives (addition, produit), par contre rarement associatives à cause des arrondis nécessaires, en particulier pour les flottants, comme le montre cet exemple trivial en `perl` :

```
print 1.0 + 0.000000000000000001 - 1.0;
# résultat 0

print 1.0 - 1.0 + 0.000000000000000001;
# résultat 1e-18
```

Un autre exemple est l'opérateur `fma` *(floating point multiply-add)* de certains processeurs. L'intégration des deux opérations peut se faire au bénéfice de la précision, parce que le résultat intermédiaire est éventuellement stocké sur un nombre de bits supérieur.

En conséquence les calculs numériques sont intrinsèquement approximatifs, éventuellement instables, et sensibles à l'ordre des opérations effectuées. C'est pourquoi les langages de programmation restreignent les possibilités d'interprétation des expressions.

Langages contraints

La spécification des langages de programmation précise donc, éventuellement de manière très directe, la manière d'interpréter une expression. L'objectif de ces restrictions est de préciser sans ambiguïté possible la sémantique du langage, et d'éviter toute surprise au programmeur.

C'est en particulier le cas du langage C dont l'associativité gauche des opérateurs est un élément essentiel de la sémantique, en particulier pour les expressions logiques :

```
int * pr = ...;
if (pr!=NULL && *pr!=0)
  ...
```

Il est cependant important de noter que ces transformations étaient autorisées au début de la définition de langage. Le langage Fortran est une exception importante dans ce domaine. Il autorise les manipulations associatives et commutatives limitées, dans la mesures où celles-ci respectent les parenthèses spécifiées par l'utilisateur, comme illustré ici :

```
! pas de réassociation possible
      x = (v + z) + (w + (v - z))
! réassociation libre par le compilateur
      x = v + z + w + v - z
```

Cependant, aucun compilateur ne semble en tenir compte en pratique.

Un cas particulier dans ce contexte est le cas d'un code généré par un compilateur, par exemple pour effectuer des communications dans HPF, par opposition au code donné par l'utilisateur. Autant le respect à la lettre du code du programmeur peut se comprendre, autant celui spécial, généré de manière automatique, peut offrir des opportunités d'amélioration et ne nécessite pas forcément de telles précautions.

Hypothèses nécessaires

Une barrière supplémentaire à prendre en compte est que certaines optimisations nécessitent des hypothèses complémentaires pour être appliquées.

Un première exemple d'une telle hypothèse est la correction du code, de manière à ce que la sémantique du langage soit connue et que l'impact des transformations effectuées soit maîtrisé. Par exemple, une optimisation sur un pointeur non initialisé peut faire la différence entre un code qui fonctionne (du point de vue de l'utilisateur) et un plantage.

Un second exemple est que certaines simplifications peuvent faire disparaître des conditions d'erreur. Par exemple pour l'opération suivante :

```
x = 1 / y - 1 / y
```

la substitution `x = 0` n'est pas valable si la variable `y` est nulle. Dans le cas d'un entier, on évite une division par zéro qui arrêterait le programme, et dans le cas flottant on évite un résultat `NaN` *(not a number)* qui sera propagé à tous les calculs suivants qui utilisent `x`. Les exceptions sont globallement incompatibles avec toute optimisation.

Un troisième exemple de contrainte est qu'une optimisation doit éventuellement améliorer l'exécution du code dans *tous* les cas. Considérons la boucle suivante :

```
do i = 1, n
   x = log(y)
   a(i) = a(i) + x
end do
```

L'extraction du calcul de `x` de la boucle ajoute une opération coûteuse lorsque la boucle est vide, et peut de plus provoquer une erreur si `y` est négatif et donc le logarithme non défini. La solution est de mettre une garde sur le calcul afin de vérifier si il doit bien être effectué ($n \geq 1$) et de le déplacer hors de la boucle `i`.

Ce type d'argument interdit très rapidement toute optimisation si celle-ci aboutit potentiellement à la moindre variation numérique d'un résultat, dans la mesure où cette différence peut toujours conduire, dans certains cas particuliers, à des résultats différents.

En d'autres termes, il est nécessaire d'être agressif et de prendre un risque vis à vis du code, que ce risque soit négatif (le résultat sera différent) ou positif. Cette sortie du cadre strict de la définition du langage peut être controllée par l'utilisateur via des options du compilateur qui peut l'autoriser ou l'interdire.

Représentation interne

Si l'on suit les prescriptions usuelles, les représentations intermédiaires des compilateurs s'appuient sur une forme particulière des opérations appelée *code 3 adresses* et contenue dans des séquences linéaires d'opérations appelées *basic blocks*. Cette approche décompose toute expression en opérations élémentaires impliquant deux opérandes et un résultat. Elle permet d'exprimer simplement et d'implémenter efficacement certaines optimisations classiques. Elle est particulièrement adaptée aux processeurs qui proposent en matériel ce type d'opération entre des registres contenant des données immédiatement disponibles.

Cette représentation a cependant un inconvénient majeur vis à vis des optimisations associatives-commutatives, voire qui cherchent à exploiter complètement les propriétés algébriques des différentes opérations utilisées : les opérations sont complètement « atomisées » et l'arbre des expressions concernées devient implicite.

Un autre exemple de cette restructuration destructrice d'information est le passage en *basic blocs* avec branchements des codes sources. Cette forme permet de représenter aussi bien des codes structurés (boucles, conditions) que des codes non structurés venant de langages peu exigeants (goto, branchements arbitraires), et plus proches de la machine. En particulier, la structure des boucles est implicite après ces transformations.

En conséquence, l'optimisation associative-commutative remet en cause profondément la structure interne de la compilation, dans la mesure où elle a besoin d'une forme plus proche du source, non atomisée et encore structurée, pour pouvoir repérer et exploiter facilement les opportunités d'optimisation.

Combinatoire et mesures

Le dernier frein à l'application de ce type d'optimisation est le caractère exponentiel des combinatoires impliquées. Les différents problèmes et sous-problèmes sont la plupart du temps au moins NP-complets, à quelques rares exceptions près. Les dénombrements des possibilités incluent des exponentielles et des factorielles.

Au delà du nombre important de solutions potentielles qui s'offrent à l'optimiseur, un second problème est que le critère dont nous souhaitons l'optimisation, à savoir le temps d'exécution, n'est pas directement estimable. Le temps d'exécution dépendra certes de ces choix, mais aussi de nombreux aspects invisibles de l'optimiseur comme la disponibilité des données dans le cache, de des choix qui seront faits par d'autres transformations comme le déroulage des boucles ou l'affectation des registres.

Cette combinaison de problématiques, avec une optimisation très combinatoire et un critère d'optimisation non directement estimable, s'annule paradoxalement et simplifie la procédure de résolution : on ne va pas chercher à optimiser, au sens de trouver *la* meilleure solution absolue, mais plutôt à s'en approcher à l'aide d'heuristiques qui utilisent des critères faciles à mesurer comme la profondeur moyenne de l'arbre d'expressions.

Un point intéressant de cette approche est qu'elle réutilise des algorithmes classiques comme l'algorithme d'équilibrage de Huffman, ou des algorithmes de compilation, dans un cadre plus large.

3.3.2 Adapter l'application au code cible

Un second exemple de remise en cause (relative) d'une approche est celle qui a conduit au début de la thèse de Youcef Bouchebaba. Plutôt que de partir d'un code dont on respecterait la sémantique, je souhaitais étudier les conditions nécessaires aux meilleures performances d'un code, en particulier en terme de gestion de la mémoire, de l'adressage, mais aussi des registres. Ces paramètres étaient en effet clefs pour la minimisation de la consommation d'énergie d'applications embarquées, qui sont très sensibles aux transferts mémoires.

Cette étude, qui part du résultat souhaité plutôt que du point de départ applicatif, permet de déterminer la nature du code que l'on souhaite produire, plutôt simple, de type traitement du signal. Elle permet ensuite de guider les différentes transformations qui seront effectuées sur les applications cibles pour se rapprocher de l'objectif et exprimer les contraintes nécessaires, comme par exemple :

- intégration de conditions à l'intérieur d'un nid de boucle ;
- découpage des données en morceaux restant dans le cache ;
- réordonnancement complet des calculs pour exploiter les tuiles ;
- utilisation maximale des registres en faisant glisser les motifs.

On aboutit alors à une série de transformations de programme qui combinent du classique (fusion de boucles, décalage d'itérations, distribution, tuilage, déroulement, etc.) vers une destination prédéfinie, avec en pratique de bons résultats.

3.3.3 Co-conception langage, compilateur, exécutif et matériel

Dans le cadre du projet FREIA, dont les principaux résultats ont été présentés à la section 2.2.4, le développement du compilateur a permis d'influencer la définition du langage (l'API en C à compiler), de l'exécutif et de certains points matériels des accélérateurs. Cette section discute les aspects de co-conception abordés dans le projet.

Signature des opérateurs

Les signatures des opérateurs image proposés ont été homogénéisées et complétées de manière à en simplifier la description dans le compilateur. Un *langage* plus orthogonal est plus facile à traduire et manipuler automatiquement.

Niveaux d'interface

Un concept considéré pourtant comme essentiel au début du projet, celui de la définition de deux niveaux d'interface, l'un composé d'opérateurs élémentaires et l'autre d'opérateurs complexes à expanser s'appuyant sur les premiers, a été fortement atténué en cours de route. En effet, selon le matériel cible envisagé les opérateurs simples ne sont pas nécessairement les mêmes, et la manière de les traduire pourrait évoluer au moins au niveau interne de la bibliothèque accessible au développeur.

Transfert du compilateur vers l'exécutif

Côté environnement d'exécution pour l'accélérateur SIMD, l'exécutif prend finalement en charge le tuilage (les tuiles de calcul sont nécessairement rectangulaires vu les contraintes de l'accélérateur en question), alors que c'était initialement attribué au compilateur. En effet, l'environnement d'exécution dispose de plus d'information, par exemple la taille réelle de l'image, alors que cette information n'est pas disponible lors de la compilation et conduirait à générer un code paramétrique assez complexe aussi bien écrit une fois pour toutes dans la bibliothèque. La compilation n'apporte pas de valeur ajoutée dans ce cas.

Transfert de l'exécutif vers le compilateur

Par contre, en sens inverse, l'allocation mémoire pour ce même accélérateur était initialement assurée par l'environnement d'exécution, au prix d'une diminution d'un facteur deux de la mémoire disponible. Les deux mémoires permettaient de simplifier l'adressage lors du recouvrement des calculs d'une tuile par le chargement des données de la suivante. Cette facilité conduit à gâcher typiquement un quart de la mémoire puisque la moitié des tuiles est utilisée uniquement en interne des calculs et ne fait pas l'objet de transferts mémoire.

En remontant cette phase au niveau du compilateur, on retire alors à l'environnement d'exécution la gestion de toute la mémoire, ce qui permet d'ajuster au strict nécessaire la quantité de mémoire utilisée pour les recouvrements, et donc d'augmenter la taille des tuiles. L'impact sur les performances générales est important car les pertes induites par les zones aux bords recalculées diminuent rapidement quand la taille des tuiles augmente.

Équivalence fonctionnelle

Un point qui a été la source de nombreux débats entre les intervenants *soft* et *hard* du projet était la non équivalence fonctionnelle des codes exécutés par la première version de l'accélérateur Terapix. En effet, les calculs effectués au bord des images étaient faux pour cet accélérateur car les valeurs par défaut nécessaires pour les pixels hors champs n'étaient pas considérées ; au contraire des valeurs plus ou moins aléatoires étaient utilisées par l'effet du voisinage torique interne au processeur.

Pire, tenter de générer des calculs corrects pour les bords n'aurait pu être possible qu'au prix d'une dégradation drastique des performances car les valeurs externes à prendre en compte dépendent de la sémantique des opérateurs de voisinage utilisés, et ne peuvent pas être imposées si des calculs successifs sont cumulés dans l'accélérateur (*i.e.*, les boucles sont fusionnées), alors même que cette fusion est essentielle pour obtenir de bonnes performances.

Le contraste était fort avec l'accélérateur vectoriel qui lui traite correctement ces bords et ne réduisait donc pas la taille de l'image résultat. À partir du moment où deux accélérateurs ne donnent pas les mêmes résultats pour une même application, toute comparaison objective devient difficile, même si d'un point de vue applicatif la perte d'une bordure est admise pour l'obtention de bonnes performances.

La décision a été prise de couper le rebouclage interne des processeurs et d'ajouter la circuiterie nécessaire à la génération des valeurs par défaut sémantiquement pertinentes. L'impact sur le processeur est faible, mais la simplification et la correction gagnées nous semblaient importantes.

Discussion

Ces différents exemples illustrent les subtiles interactions entre le compilateur, son amont et son aval, et l'impact de décisions prises assez tôt, voire trop tôt, et dont on s'aperçoit après coup qu'elles ont un impact négatif. Le dialogue entre disciplines a permis de les résoudre et de corriger les choix initiaux, mais mon impression est que ce type de décision est souvent pris de manière prématurée dans un projet avec des conséquences négatives.

3.3.4 Maximiser les défauts de cache

Après des années consacrées à optimiser des codes pour en *maximiser* les performances en exploitant au mieux la hiérarchie mémoire, c'est avec délectation que je découvrais l'article de Martín Abadi, Mike Burrows et Ted Wobbler (Moderately hard memory-bound functions, ACM Trans. Inter. Tech. 5 (2) : 299–327) qui cherchait au contraire à *minimiser* les performances d'un algorithme pour n'importe quelle implémentation...

Les conditions du problème sont évidemment très spéciales. Il s'agit de protocoles cryptographiques particuliers, présentés à la section 2.3.2, qui visent à prouver qu'un intervenant a fait un certain effort, exprimé par exemple en temps de calcul ou en nombre d'accès aléatoires en mémoire, et que cet effort est aisément vérifiable par celui qui le requiert, sans avoir bien sûr à suivre le même chemin et donc à fournir le même effort.

L'idée d'attaquer la problématique sur le terrain des accès à la mémoire vise à adapter la technique de preuve d'effort à un environnement très hétérogène d'acteurs : alors que les performances varient de manière très importante en allant des petites machines de poche aux gros serveurs, on ne peut pas en dire autant des performances des accès à la mémoire. En conséquence, si la fonction de preuve d'effort dépend structurellement de ces accès, la variation des performances de haut en bas du spectre des machines sera moindre.

Au delà de cette idée originale, amusante et utile, la proposition pratique de l'article ne m'a pas semblé très intéressante pour ce qui concerne les performances. L'écart entre la recherche de solution et la vérification était simplement quadratique, cette progression étant, qui plus est, due à l'exploitation de propriétés très particulières des fonctions aléatoires.

En étudiant très précisément la structure de la fonction proposée, je me suis rendu compte que l'objectif d'utiliser des accès mémoire pouvait être partiellement séparé de l'aspect d'écart combinatoire recherché pour agrandir l'écart entre la recherche de solution et la vérification de l'effort. Ceci m'a amené à la définition d'une nouvelle fonction.

Les relecteurs des différentes soumissions décrivant ces travaux m'ont permis d'avoir l'occasion de réfléchir en profondeur à mon approche, en particulier à la preuve d'optimalité de la fonction proposée, qui est toujours à l'état de rapport interne.

3.3.5 Conseils relationnels

Cette section ne traite pas de psychologie ni de relation humaine. Ce dernier exemple d'approche pragmatique, lié à ma passion pour le logiciel libre d'une part et à l'enseignement de la modélisation relationnelle d'autre part est le petit prototype de schéma *advisor* développé spécifiquement pour PostgreSQL [38]. J'ai constaté depuis quelques années que les modélisations et les réalisations faites par les étudiants, mais aussi par des professionnels, voire parfois par moi-même, contiennent souvent des erreurs plus ou moins grossières ou subtiles.

L'idée un peu saugrenue que j'ai eu est d'essayer de détecter certaines de ces erreurs directement à partir du schéma de la base déjà implémentée, en interrogeant au moyen de vues le schéma système pour découvrir les relations isolées (personne ne les référence, elles ne référencent personne), les erreurs de types sur les clefs étrangères, les index inutilisés, des droits incohérents, etc. Il s'agit donc d'intégrer sous forme d'un schéma spécial une expertise indépendante et éventuellement subjective qui teste le maximum d'aspects d'un schéma applicatif de base de donnée. Ce schéma implémente une analyse statique d'un schéma relationnel, et est exprimé lui-même dans un cadre relationnel.

Un prototype dénommé SALIX [1] (nom latin du saule pleureur) a été développé pour PostgreSQL et MySQL et appliqué à l'analyse des schémas relationnels de plus de 500 logiciels libres [63, 64]. Les résultats ont été validés statistiquement pour ne garder que les conclusions réellement significatives. Ils montrent en particulier une utilisation très anecdotique des contraintes d'intégrité référentielle dans ces projets et, pour les projets fondés sur MySQL, la dépendance quasi exclusive de ces schémas au moteur de stockage *MyISAM* qui n'implémente aucune notion transactionnelle. La qualité des modèles de données relationnels des logiciels libres est faible, et n'est clairement pas la préoccupation majeure de leurs développeurs.

3.4 Évidences

Le dernier point que je souhaite aborder dans ce chapitre consacré à l'esthétique de mon travail de recherche n'est pas le moindre. Il a trait aux évidences, dans le sens anglais du terme, c'est à dire des preuves mathématiques. Si proposer des algorithmes, des approches, des solutions, les implémenter, les tester, est très bien, y ajouter une preuve que ce que l'on fait est parfait, optimal, ou même seulement correct, est mieux. Enfin, l'idéal est que la preuve soit elle-même élégante, simple et incontournable. Je n'y suis sans doute pas toujours arrivé, mais cela a quand même été un objectif constant. Je note également que la rédaction pénible de ces preuves a toujours été l'occasion d'aller au cœur des méthodes proposées, donc de mieux les comprendre. Enfin, ces preuves ont très souvent été un moteur pour perfectionner les méthodes proposées, par exemple afin d'atteindre un optimum.

3.4.1 Cadre général

Une première satisfaction mathématique de mes travaux, principalement avec l'aide de François Irigoin et Corinne Ancourt, est d'avoir exprimé les travaux de compilation de HPF dans le cadre élégant d'une modélisation affine sur des points entiers [7, 22, 65, 8], que l'on peut interpréter géométriquement comme des polyèdres et des treillis. Cette modélisation, outre son élégance, ouvre la porte à de nombreuses manipulations, de nombreux algorithmes de transformations qui exploités convenablement amènent naturellement à des solutions des problèmes que l'on cherche à résoudre.

Un autre exemple de ces cadres généraux auxquels des travaux se raccrochent est l'expression et l'optimisation du problème des redistributions dans le cadre classique en compilation d'un problème de graphe et de propagation d'informations sur ce graphe [31], pour aboutir à la résolution assez satisfaisante du problème posé.

Un dernier exemple de contribution est la formalisation des travaux sur les régions de Béatrice Creusillet à la fin de ses travaux de thèse. L'essentiel était fait, implémenté, prouvé, testé ; seule manquait une présentation simple et formelle, donc intelligible, des facteurs multiples intervenant dans les calculs : la structure du code, les transformeurs qui expriment le fonctionnement du code, les préconditions, les différents types de régions, et d'y intégrer en plus la notion de sur ou sous-approximation, de préférence de manière syntaxiquement simple, par exemple avec des transformations élémentaires de nature algébrique.

En partant de la notion classique d'état *(store)* et d'exécution par transformation de cet état, on peut exprimer le comportement du programme en écrivant des équations pour chaque élément syntaxique

de construction d'un langage simple (boucle, affectation...), puis naturellement décrire la propagation des connaissances extraites comme les préconditions ou les régions. Comme l'implémentation réelle des opérateurs d'analyse et la représentation des ensembles sont approchées par rapport aux opérateurs idéaux, en sur ou sous approchant le résultat réel, les équations idéales se déclinent naturellement en sur et sous approximations des régions.

3.4.2 Preuves

Voici quelques exemples choisis de développements mathématiques auxquels j'ai participé concernant des preuves mathématiques d'algorithmes, d'optimalité, ou de dénombrements, éventuellement en collaboration avec des collègues ou des doctorants. Le but n'est pas de répéter ici les preuves, mais plutôt de les discuter.

Optimalité des replacements

Lors de mes travaux sur les replacements de HPF présentés à la section 2.1.4, une preuve d'optimalité a été développée pour les communications, qui couvre à la fois le nombre et le contenu des messages. Ce développement assez simple prend en compte l'exactitude de la description utilisée pour représenter les données à transférer, les aggrégations des messages, et le fait que l'implémentation ne transmet pas de messages vides. Ce dernier point n'était pas dans l'implémentation initiale pour laquelle des messages vides pouvaient être envoyés et reçus dans certains cas particuliers. C'est la preuve elle-même qui a été le moteur de cette amélioration. Un dernier aspect sur cette preuve est qu'elle ne prend pas en compte le placement particulier des processeurs au sens HPF sur les processeurs physique de la machine, parce que cette informations n'est pas disponible. La conséquence principale est qu'en prenant en compte cette information, on pourrait imaginer construire des communications un peu plus réduites dans ce cas.

Dénombrement et optimalité des expressions

Un premier point important dans les travaux de Julien Zory était de mesurer explicitement la combinatoire à laquelle conduisaient les manipulations d'expressions associatives-commutatives. Ces dénombrements d'arbres binaires, prenant en compte les opérations combinées type multiplication-addition, sont présentés à la section 7.3.1 de sa thèse [87]. Le problème des dénombrements n'est en général pas de compter, mais plutôt de comprendre ce que l'on compte et donc de compter ce que l'on souhaite dénombrer, et pas d'autres choses. Dans le cas des réassociations, la subtilité est que certains éléments peuvent être considérés comme interchangeables on non, selon le niveau auquel on se place : par exemple $(x_1 + x_2) + x_3$, $(x_2 + x_1) + x_3$ ou $x_3 + (x_2 + x_1)$ sont les mêmes arbres d'expression commutative d'un point de vue géométrique et numérique, mais ils sont numériquement distincts de $x_1 + (x_2 + x_3)$ qui leur est structurellement identique. Nous avons donc compté les arbres dans une classe d'équivalence à la commutativité près.

L'optimalité des associations et des factorisation associative-commutative proposée par Julien Zory fait l'objet de l'annexe A de sa thèse. La preuve prend en compte le rang des variables au sens de la profondeur de leur création dans un nid de boucles, de manière à privilégier les solutions qui favorisent l'extraction d'invariants. En conséquence, la complexité optimisée est un polynôme dont les termes d'ordre supérieur représentent les opérations les plus internes, et la minimisation est à prendre au sens lexicographique.

Optimalité de Hokkaido

Les démonstrations mathématiques dans le domaine de la cryptographie sont assez particulières. Une raison fondamentale est que ce que l'on veut souvent prouver, c'est la difficulté, au sens de la complexité, de certaines opérations qui sont forcément possibles. Par exemple, on souhaiterait prouver que des collisions de fonctions de hachages $h(m_1) = h(m_2)$ ne peuvent pas être *facilement* construites, ou que le seul moyen pratique de recouvrer un message chiffré est d'utiliser la clef utilisée pour son chiffrement $m = d(k, c)$. Cependant, prouver l'impossible ou le difficile n'est pas naturel, parce qu'il faut faire cette preuve indépendamment de toute méthode de résolution, connue ou inconnue. Pour les méthodes connues, il est facile de les analyser précisément et de prouver leur complexité. Mais qu'en est-il de ce qui n'a pas encore été imaginé ?

J'ai été confronté à cette problématique de preuve ou de justification pour ma fonction de preuve d'effort Hokkaido [40]. Le but est de montrer que la résolution d'un problème occasionne intrinsèquement de nombreux défauts de cache, quelle que soit la méthode de recherche utilisée. Sur les deux autres travaux proposés dans la littérature, les preuves apportées n'étaient pas complètement convaincantes :

- *Abadi et al.* n'offre pas de preuve au sens mathématique du terme, mais plutôt une longue suite d'arguments et de discussions pour montrer que les auteurs ont envisagé et paré beaucoup d'attaques potentielles, sans pour autant prétendre à l'exhaustivité ;
- *Dwork et Naor* offrent par contre une preuve assez lourde qui s'appuie sur une modélisation mathématique de la hiérarchie mémoire d'un ordinateur et du fonctionnement du cache ; cependant, le fait que résoudre le problème nécessite de passer par des accès de cache est l'objet d'une proposition non formellement prouvée (mais sans doute vraie).

En abordant la preuve de ma fonction, vivement encouragé en cela par l'exigences des relecteurs, je n'étais pas très motivé par suivre ces deux exemples. La preuve que j'ai proposée est intéressante à plusieurs titres, parce qu'elle répond à certaines des lacunes des méthodes précédentes, ou, à tout le moins, explicite les hypothèses nécessaires :

- tout d'abord, elle traite explicitement le cas de méthodes de résolution à énumération non déterministe, qui explorent l'espace des solutions en profitant de manière privilégiée des données disponibles dans le cache : il faut montrer que de telles méthodes ne gagnent pas en efficacité ;
- elle néglige par contre explicitement les méthodes qui sont susceptibles d'énumérer plusieurs fois une même solution : c'est une restriction très subtile, car pour ma méthode, et sans doute aussi pour les méthodes concurrentes, il est possible d'imaginer pour certains paramétrages des techniques de recherche de solution dont la complexité est certes catastrophique, mais qui n'utilisent pas ou peu la fonction tabulée à l'origine des défauts de cache souhaités ;
- enfin, elle évite de modéliser mathématiquement la hiérarchie mémoire d'un ordinateur et le comportement des accès en plaçant cette problématique à un niveau axiomatique : un accès mémoire ou un défaut de cache est simplement une unité de coût liée à l'accès à une fonction tabulée (résolution par énumération déterministe) ou à la détermination du fait qu'un élément appartient à un ensemble (résolution par énumération non déterministe qui évite les doublons).

Cette preuve n'est pas parfaite, puisqu'elle laisse de côté le cas des algorithmes qui peuvent explorer plusieurs fois la même solution. Cependant ces cas particuliers n'étaient pas non plus complètement abordés dans les arguments ou preuves des autres travaux. C'est donc plutôt un mérite d'expliciter cet aspect.

Enfin, l'écriture de cette preuve a été l'occasion pour moi de réaliser que la méthode que je proposais avait un ratio de travail entre la recherche et la vérification optimal, ce avec une démonstration particulièrement simple : le client cherche une solution à un problème soumis dans un espace de taille 2^ℓ, et doit ensuite la communiquer au serveur, ce qui nécessite forcément ℓ bits, qui est justement la complexité de la vérification de la solution proposée. Ici encore, l'écriture même de la preuve a été le moteur de certains des résultats obtenus.

Optimalités d'une preuve d'effort

Un point important des preuves est bien sûr la recherche d'une certaine optimalité, qui doit d'abord être définie, ce qui n'est pas toujours évident. Lors de mes travaux sur les preuves d'effort, j'ai introduit deux nouvelles notions d'optimalité qui permettent de comparer les différents protocoles et fonctions proposés dans la littérature :

optimalité des communications, si les volumes données communiquées entre le demandeur du service et le fournisseurs sont minimales, ce qui est facilement le cas pour les techniques à base d'énumération qui retournent la valeur du compteur pour la solution trouvée ;

optimalité du calcul, si la vérification de la preuve d'effort est proportionnelle aux volumes des données reçues, que celles-ci soient minimales ou non : ce critère est important parce qu'il permet d'éviter des attaques de déni de service sur les systèmes de preuve d'effort, en évitant que de fausses preuves d'effort nécessitent un travail important pour être écartées.

Ces deux notions ne sont cependant pas aussi évidentes que cela à mettre en pratique. Peu des fonctions de preuve d'effort de la littérature sont optimales selon les critères que j'ai proposé.

3.5 Conclusion

Ce chapitre a abordé des aspects esthétiques de mon travail de recherche, ayant trait à la conception du langage HPF, aux stratégies globales de compilation, au pragmatisme dans la résolution des problèmes, et aux preuves.

La première partie a discuté les problèmes de conception de HPF, tout à la fois dans son esthétique que dans la conception détaillée des caractéristiques du langage de manière à assurer que sa compilation pourra être efficace. Mon analyse des raisons de l'échec du langage est une contribution originale de cette partie : HPF était trop gros pour un petit marché, en avance sur l'état de l'art, avec un comité votant des ajouts ou des retraits sans préoccupation claire de leur utilité et de leur compilation. En d'autres mots, trop d'ambition, trop de promesses qui n'ont pas pu être tenues et ont conduit à la déception et l'échec du langage.

La second partie a présenté une stratégie globale de compilation développée dans le cadre du projet ANR FREIA, qui permet de compiler des applications de traitement d'image vers des accélérateurs matériels spécifiques. Un point clef de l'approche est de réutiliser des techniques de compilation classiques, usuellement appliquées à des données scalaires dans des registres, pour les appliquer à des opérations sur des images. Ceci a permis de développer très rapidement un compilateur et des générateurs de code spécifique au sein du logiciel PIPS, et de décliner des expériences précises qui montrent l'intérêt de cette approche globale, comme cela sera présenté au chapitre suivant.

La troisième partie discute le pragmatisme qui guide les approches développées, par exemple la remise en cause des expressions ou le développement d'analyses statiques dans le cadre des bases de données relationnelles. Un aspect particulier abordé est la co-conception, avec un langage qui exprime la sémantique, du compilateur qui assure des transformations, et enfin de l'exécutif et du matériel qui implémentent in fine les fonctionnalités.

Enfin, la dernière partie discute enfin des preuves dont la recherche est, sans surprise, un moteur pour améliorer les méthodes proposées.

Chapitre 4

Expériences

Le fait de provoquer un phénomène dans l'intention de l'étudier (de le confirmer, de l'infirmer, ou d'obtenir des connaissances nouvelles s'y rapportant).
(dictionnaire Le Petit Robert*)*

Le monde de l'informatique est a priori celui du déterminisme total, du prévisible intrinsèque, du parfaitement connu : les circuits du processeur n'appliquent qu'une logique programmée et prévue, malgré leurs fondements quantiques, et sont donc toujours modélisables et simulables. Nous sommes donc loin des sciences expérimentales au sens classique du terme, telle la biologie, dont les vérités dérivent du vivant variable et mal connu. L'ordinateur vient au contraire de la logique mathématique. Il est donc paradoxal de parler d'expériences dans ce domaine : on ne voit pas Alan Turing faire des expériences sur sa machine ; il prouve plutôt des théorèmes...

Cependant, les effets et les causes du déroulement d'un programme conduisent très vite à une grande combinatoire. La machine a une complexité remarquable. Les entrées-sorties basées sur les réseaux de transferts d'une machine parallèle, les accès disques de la mémoire virtuelle, les interactions entre des processus en compétition pour l'utilisation d'une mémoire cache, tout cela ajoute une incertitude physique, et dissimule le déterminisme sous-jacent. Enfin, nous nous sommes souvent intéressés à des problèmes difficiles au sens théorique du terme, aux limites de ce qui est pratiquement calculable. Nous avons proposé des méthodes heuristiques pour résoudre des classes de problèmes. En conséquence, il nous a toujours été nécessaire de montrer le côté pratique et efficace de méthodes proposées sur le papier. Pour cela, il a fallu implémenter (section 4.1) nos algorithmes, et faire de multiples expérimentations 4.2.

4.1 Implémentations

L'implémentation effective des algorithmes d'analyse et de transformation développés au cours des recherches auxquelles j'ai été associé est un travail d'ingénierie, donc d'ingénieur, qui ne fait pas directement partie de la recherche (en tous cas pas celle directement valorisable par des publications), mais qui est essentiel pour justifier au delà des mots et des mathématiques la réalité des résultats obtenus, en particulier dans le domaine de l'informatique.

Un aspect important de l'implémentation d'une nouvelle technique est que celle-ci s'appuie souvent sur d'anciennes ou d'autres analyses, qui ne sont pas nécessairement originales mais qui sont par contre indispensables pour la faire fonctionner. Cela implique aussi un travail d'ingénierie lourd qui n'est pas facile à valoriser en nombre de publications. Je me suis donc très tôt lancé dans l'implémentation de méthodes de compilation comme le traitement des communications aux voisinages pour la compilation des boucles parallèles [16, 19, 18], et j'ai toujours encouragé mes doctorants dans cette voie.

4.1.1 Newgen

Les structures de données de PIPS sont déclarées avec l'outil de génie logiciel Newgen principalement développé par Pierre Jouvelot et Rémi Triolet. À partir de la déclaration de domaines assez simples, l'outil génère des fonctions et des macros destinées à différents langages cibles qui permettent de manipuler de manière homogène les structures de données correspondantes : allocation et libération de la mémoire, accès aux champs, tests des unions, sérialisation des données dans un fichier, etc. J'ai apporté plusieurs contributions à Newgen.

Typage

L'implémentation initiale pour le langage C était fondée sur des macros générées par le compilateur Newgen. Le résultat était efficace, mais par contre il n'y avait aucun contrôle du typage, ce qui occasionnait des difficultés dans le développement de nouvelles fonctionnalités dans PIPS parce que des erreurs de type pouvaient se propager assez loin avant d'être détectées. De plus, le typage des structures était lui même faible et géré dynamiquement par l'outil.

J'ai donc remplacé les macros générées par des structures C classiques et des fonctions. Les structures reprennent et explicitent l'organisation dynamique précédente. Les fonctions assurent la fourniture au compilateur de toutes les informations de typage nécessaires. J'ai également complété Newgen pour pouvoir utiliser de manière explicite des structures de données implémentant des fonctions à base de table de hachage (fonctions et ensembles).

Optimisation de la récursion automatique

Une seconde contribution à l'outil de génie logiciel Newgen a été l'implémentation optimisée d'une fonction de récursion automatique sur des structures de données arbitraires de Newgen. À partir d'un point de départ, la structure de données est parcourue de manière à visiter tous les domaines (types) souhaités : à la descente dans le graphe une fonction de décision permet de décider de continuer ou non la récursion, et à la remontée une fonction de réécriture permet de modifier le nœud. Chacun des nœuds est parcouru une seule fois en mémorisant au fur et à mesure ceux qui ont déjà été visités.

Les principales optimisations, simplifications et extensions ont consisté à permettre de spécifier un ensemble de types à visiter avec pour chacun une fonction de filtrage et de réécriture ; à passer un contexte (pointeur vers une structure de donnée arbitraire choisie par l'appelant) qui sera transmis aux fonctions de contrôle de la récursion ; à ne suivre *que* les arcs du graphe susceptibles d'aboutir aux nœuds de type souhaité, en effectuant une fermeture transitive sur le graphe de type.

Le résultat ultime de ces travaux est la fonction à nombre variable d'arguments dont le prototype est donné ci-dessous :

```
void gen_context_multi_recurse
  (void * start_node,
   void * context,
   [int domain_to_visit,
    bool (*domain_filter)(domain, context),
    void (*domain_rewrite)(domain, context)]*
   NULL);
```

Elle a été utilisée pour l'implémentation de nombreuses analyses et transformations de code dans PIPS [82, 74, 75].

Stockage des résultats d'analyses de PIPS

L'application d'analyses interprocédurales fondées sur des méthodes polyédriques sur de très gros codes industriels, par exemple un code de EDF plus de 1000 routines et 100 000 lignes de code, aboutit à des volumes de données très importants, typiquement de l'ordre de 500 Mo, obtenus au prix de nombreuses heures de calculs. Dans le paralléliseur et analyseur PIPS, ces données sont stockées en mémoire puis dans des fichiers au format texte qui stockent la sérialisation des structures de données spécifiques gérées par l'outil de génie logiciel Newgen.

L'implémentation initiale du stockage en mémoire était fondée sur des listes associant, pour chaque module et analyse, les données calculées. Pour quelques dizaines d'analyses et milliers de modules, la

taille de cette liste est évidement prohibitive. Elle a donc été remplacée par une cascade de tables de hachage : à un module correspond une association qui lie les analyses aux données calculées.

Un second aspect a été de compresser le stockage texte sur disque des données de manière à diminuer l'espace de stockage mais aussi d'accélérer l'analyse syntaxique pour la récupération des structures. Ce stockage reste cependant au format texte de manière à faciliter le debug éventuel des structures en pouvant consulter facilement le contenu fichier, même si le format sérialisé d'une structure de donnée arbitraire et récursive n'est bien sûr pas très digeste.

Une troisième amélioration importante a été de virtualiser les numéros globaux de structures utilisés par la sauvegarde de manière à pouvoir ouvrir à nouveau d'anciennes données sauvées, même si la structure de celles-ci a été modifiée entre temps, dans la mesure où ces modifications sont mineures. Avant cette modification, l'ajout d'un nouveau champ dans une structure Newgen même sans rapport changeait ces numéros globaux et rendait obsolètes et inutilisables des résultats d'analyses calculés à grand peine.

Conclusion sur Newgen

Newgen était très novateur dès sa conception à la fin des années 1980, et il apporte toujours aujourd'hui une valeur ajoutée significative pour la manipulations des structures de données au sein de PIPS, en particulier parce qu'il uniformise les méthodes d'accès à ces structures et simplifie beaucoup la modification ou l'ajout de nouvelles structures. Il rend le langage C utilisé dans PIPS supportable.

4.1.2 Linear/C3

Il est très vite apparu que, sur des codes réels, l'implémentation à base d'entiers 32 bits de la bibliothèque linéaire ne suffisait pas à représenter les systèmes de contraintes issus des analyses, malgré des simplifications permanentes des systèmes linéaires créés lors des analyses. En effet, des coefficients apparaissent ayant pour valeur le PPCM (plus petit commun multiple) des différents coefficients, en particulier lors de la linéarisation des systèmes pour les calculs des dépendances. Nous avons donc porté la bibliothèque pour utiliser les entiers 64 bits, et pour détecter les débordements de capacité lors des calculs entiers. Une autre contribution, décrite à la section 2.3.1, est l'optimisation de l'algorithme de génération de code de parcours de polyèdre et d'autres opérateurs polyédriques.

Passage en 64 bits

Une première étape a été de porter la bibliothèque linéaire en s'appuyant sur des entiers 64 bits. Une implémentation générique fondée sur un type paramétrable `Value` a donc été développée, occasionnant des transformations très importantes des codes sources de la bibliothèque linéaire et de PIPS (mais moins que si on avait pris du multiprécision qui induit la nécessité très coûteuse de gérer la mémoire pour les valeurs). Cependant le portage en entier 64 bits ne suffit pas pour représenter les polyèdres de certains codes, et les résultats des analyses étaient potentiellement faux. En effet, les entiers des processeurs implémentent une arithmétique modulo : les dépassements de capacité lors d'une addition ou d'une multiplication ne sont donc pas détectés.

Gestion des débordements

La seconde étape a donc été de détecter les dépassements de capacité des différentes opérations effectuées sur les entiers et de générer des exceptions. Ceci pose plusieurs difficultés :

1. le langage C qui sert de base à la bibliothèque ne possède pas de mécanisme naturel d'exception intégré au langage, contrairement à d'autres langages comme C++ ou Java ; il n'était pas question de changer le langage pour ne pas forcer l'utilisation de la librairie avec C++ uniquement ;

2. le C ne dispose pas d'une gestion de mémoire automatique ; lorsqu'une exception est levée, les structures allouées doivent pouvoir être libérées manuellement ou bien la mémoire est perdue ;

3. la détection manuelle des débordements $\frac{xy}{y} = x$ fait naturellement appel à une division entière, qui est un opérateur particulièrement coûteux sur les processeurs ; certaines versions préliminaires des processeurs SPARC de Sun déléguaient même le calcul au système d'exploitation via une exception.

L'implémentation de sauts arbitraires s'appuie donc sur des *long jump* fournis par une bibliothèque spécifique du système. Elle fournit des fonctions de très bas niveau qui permettent de remonter arbitrairement dans la pile d'appels pour se replacer dans un contexte antérieur. Un jeu de macros CPP (`TRY`, `CATCH`) permet d'écrire un code ressemblant à du C++ qui cache en effet les appels aux fonctions de saut susmentionnées. D'autres macros cachent les vérifications liées aux débordements et l'appel éventuel à la levée d'exception qui précise son type. Il permet également d'attraper et de renvoyer certaines exceptions de manière à assurer à la main la libération des structures de données temporaires allouées par les algorithmes.

La précision et la confiance dans les résultats des analyses ont été grandement améliorées par cette nouvelle implémentation. Néanmoins il a été nécessaire d'adapter la sémantique des différents opérateurs en fonction des besoins lorsqu'une erreur est détectée, de manière à renvoyer un résultat compatible avec l'attente de l'algorithme utilisant ces opérateurs : création d'une sur ou sous approximation du résultat réel, basculement sur un autre algorithme, etc. Ces adaptations fines ont été faites au cas par cas, principalement par Corinne Ancourt et Béatrice Creusillet.

Enfin, notre jeu de macros a été repris pour l'implémentation 64 bits de la **Polylib**, qui avait les mêmes besoins de précision et de détection des risques de débordements.

4.1.3 PIPS

Les principaux efforts d'implémentation concernent cependant le logiciel PIPS lui-même, qui utilise l'outil de génie logiciel et la bibliothèque linéaire cités précédemment. Parmi les implémentations d'analyses et de transformations d'intérêt général que j'ai développées dans PIPS en fonction des besoins, éventuellement en collaboration avec des collègues, des doctorants, des élèves ou des stagiaires, je peux citer entre autre : une détection de réductions simples, une « atomisation » paramétrique, une phase de clonage, une interface graphique fondée sur Java avec des menus construits dynamiquement [78], l'ajout d'une phase de typage explicite d'un programme Fortran [86], et la détection d'invariants de boucles [6]. Un point commun des différents compilateurs ou phases de transformations fondés sur PIPS, décrits ci-après, est leur caractère source-à-source, même lorsqu'il s'agit de compiler pour un accélérateur matériel. Ce choix simplifie beaucoup le développement et le déboguage des phases, en laissant en général les tâches de bas niveau comme la sélection des instructions ou l'allocation de registre au compilateur suivant qui est vu comme un simple générateur de code binaire.

HPFC

Le prototype de compilateur développé pendant mes stages d'option et de DEA, ma thèse, et amélioré après avec, entre autres, des contributions de Julien Zory, représente 20500 lignes de C pour le compilateur, 5900 lignes de Fortran pour la partie exécutif qui utilise PVM ou MPI (en fait la bibliothèque de support est beaucoup plus importante, mais l'essentiel des fonctions sont générées automatiquement à partir de modèles), et enfin 1200 lignes de scripts divers (shell, sed).

Eole

Les différentes techniques suggérées pour les manipulations d'expressions associatives-commutatives ont été implémentées au sein du prototype EOLE développé pour l'essentiel par Julien Zory. Une partie des transformations, en particulier l'équilibrage, et la détection d'expressions communes, sont directement dans PIPS et ont été développées en C. La partie proprement associative-commutative s'appuie sur l'outil CAVEAT du CEA, initialement un simplificateur d'expressions, et la bibliothèque de comparaison de motifs associatifs-commutatifs STORM des universités d'Orléans et de Stony Brook. Elle a été développée en C++. Les transferts de données entre les deux parties utilisent les structure de données sérialisées de Newgen. Les résultats obtenus ont été très bons, les temps de compilation restant très raisonnables malgré la combinatoire des problèmes sous-jacents. Cet aspect était une condition essentielle pour montrer le caractère pertinent du problème abordé et des solutions proposées.

FREIA

Le compilateur pour FREIA, qui traduit l'API vers trois accélérateurs comme décrit à la section 2.2.4, a été implémenté dans PIPS. La première phase s'appuie sur une série de transformations classiques

qui étaient déjà disponibles ou ont été développées ou améliorées pour l'occasion : *inlining*, évaluation partielle, déroulage de boucle, suppression de code mort, etc. La réutilisation de l'existant est maximale. La seconde phase, qui construit et optimise le graphe d'opérations image, et la troisième phase qui implémente les générateurs de code spécifiques à chaque cible ont été développés dans une série de passes. Ces passes représentent moins de 10000 lignes de code en C ajoutées au compilateur PIPS et utilisant sa représentation intermédiaire et l'outil Newgen.

Infrastructure

Un travail important effectué dans PIPS a été de mettre à jour l'infrastructure de compilation et de gestion des sources de manière à permettre un mode de développement à distance et l'existence de branches indépendantes. Ce travail a été motivé par l'utilisation de l'outil à distance par d'anciens du centre, en particulier Ronan Keryell, Nga Nguyen, Youcef Bouchebaba et Alain Muller.

Le modèle initial de développement des trois logiciels interdépendants Newgen, Linear/C3 et PIPS était un modèle centralisé local sur une machine Sparc SUN, à base de fichiers partagés et d'un historique facultatif utilisant les outils SCCS ou RCS. L'éditeur de fichier permettait d'assurer l'exclusivité des accès à un fichier particulier. Les `Makefile`s de chaque répertoire étaient générés automatiquement par un script. Deux versions existaient en parallèle, une version de développement et une version de production, avec la possibilité de mettre à jour la version de production à partie d'une partie de la version de développement. Ce modèle avait l'inconvénient de nécessiter la connexion physique à la machine hébergeant le logiciel, ou au moins le partage des fichiers de données par NFS.

Le nouveau modèle de développement, commun aux trois logiciels, est maintenant un modèle accessible à distance [45]. Des dépôts SVN accessibles sur Internet via le protocole HTTPS hébergent les sources des trois logiciels. Chacun peut librement obtenir une copie de travail de ce dépôt et compiler en local. Les `Makefiles` sont partagés et non plus générés automatiquement. La distinction entre développement et production existe toujours, mais est cette fois fondée sur des branches personnelles attribuées à chaque développeur, et créées en fonction des besoins.

Il faut noter que le passage du mode fichiers directement partagés à un mode où chaque développeur dispose d'une copie locale, et gère éventuellement ses propres branches, nécessite une révolution de ses pratiques : mise à jour et maintenance de sa propre copie de travail, validation personnelle des modifications, décision de fusionner les branches à prendre, nécessité de propager ses modifications pour que celles-ci soient sauvegardées, en particulier dans le cas d'un ordinateur portable.

Tests de non régression

Pour un logiciel aussi gros que PIPS, les tests de non régression permettent de s'assurer de la stabilité des résultats obtenus par les différentes passes d'analyses et de transformations, malgré les modifications apportées par les développeurs qui interviennent sur le projet. Les développeurs ont parfois tendance à ne pas faire tourner ces tests, ou bien rarement, parce que ceux-ci sont longs et chargent beaucoup la machine du développeur, en particulier quand il s'agit d'un ordinateur portable. En conséquence, on peut détecter des changements de manière tardive et avoir du mal à trouver la modification qui les ont occasionnés.

De manière à améliorer cette situation, j'ai développé un système d'exécution automatique lors des modification et périodique la nuit sur une machine dédiée avec 16 cœurs et 16 Go de mémoire. La parallélisation de la compilation et de l'exécution des tests, simplement fondée sur *GNU make* qui gère à la fois les dépendances et une limite de charge, permet d'obtenir les résultats en quelques minutes qui sont ensuite envoyés sur une liste de diffusion. Le fait que le développeur qui « casse » certaines passes règle les problèmes qu'il a créés n'est bien sûr pas garanti, mais le système permet au moins d'identifier l'auteur et les modifications qui créent un changement, ce qui est une amélioration significative par rapport à la situation précédente.

4.2 Expérimentations

Au cours de mes travaux, j'ai souvent été confronté à des besoins de mesures précises, d'analyses des résultats obtenus, de modélisations pour comprendre les différents effets des causes possibles. Ces travaux

ont souvent ouvert des questionnements moteurs des progrès accomplis ensuite. Voici quelques exemples de ces travaux expérimentaux, suivis des enseignements que j'en ai tirés.

4.2.1 Quelques exemples

Je fais de la recherche depuis 1992, et j'ai eu l'occasion d'expérimenter avec de nombreux matériels au cours de ces 20 ans : mes premières expériences avec mon prototype de compilateur HPF ont commencé sur le réseau de stations de travail Sun disponible à l'École des mines de Paris [19, 18, 20].

J'ai ensuite eu accès à différentes machines comme la CM5 du SEH, la SP2 de l'École des mines de Paris, la ferme de DEC Alpha du LIFL, le CRAY T3D de l'IDRISS... sur lesquelles j'ai pu faire de nombreuses mesures qui ont alimenté ma réflexion et mes travaux. En particulier, la ferme d'Alpha a été une bonne source de comparaison pour ma technique de compilation des communications des replacements HPF [65].

Le SP2 local a permis à Julien Zory [88, 87] de très nombreuses expériences avec HPF et les manipulations d'expressions comme celles rapportées à la figure 2.20. Les expériences nous ont donné des résultats surprenants par rapport à ce que nous attendions : en effet, alors que sur P2SC l'équilibrage des expressions apportaient de meilleures performances, c'était exactement le cas inverse sur MIPS R10000. L'interprétation de ce résultat contre-intuitif a induit une étude poussée du compilateur MIPS pour découvrir que celui-ci déroulait systématiquement les boucles sans prendre en compte la pression sur les registres. Pendant toute la thèse de Julien, les expériences ont été un moteur pour détecter rapidement des problèmes concernant les applications, les compilateurs et les matériels, et pour les résoudre, plus lentement.

L'utilisation de stations de travail ou de simulateurs a aussi été le support des mesures effectuées pour les travaux sur l'exploitation fine de la hiérarchie mémoire avec Youcef Bouchebaba [13, 12, 10, 11, 15, 14]. Une partie des expériences pour les accélérateurs cible du projet FREIA ont été fondées sur des simulateurs fonctionnels instrumentés, qui ont permis à la fois d'exécuter les codes de manière assez performante sur une station de travail ou un ordinateur portable, tout en donnant aussi des résultats en terme de temps d'exécution.

Divers GP-GPUs ont été utilisés pour des expérimentations : lors des travaux FREIA pour expérimenter la sortie OpenCL du compilateur, lors des expériences sur l'implémentation parallèle de `hashcash` par Etienne Servais, et également les nombreuses expériences effectuées par Mehdi Amini pour sa thèse.

Mes travaux sur les fonctions cryptographiques de preuve d'effort [40] bornées par la mémoire ont permis quelques expériences amusantes : l'objectif de ces fonctions est que leur performance doit faiblement varier avec l'âge de la machine. Pour mesurer cet effet, il a falu donc expérimenter avec de vieilles machines sorties du placard, un peu poussiéreuses, trouver des pièces de rechange, installer un système récent dans un espace limité, et faire fonctionner le tout assez longtemps pour obtenir des mesures.

Un dernier exemple concerne l'algorithme de rapprochement de deux relations de base de données afin de découvrir les différences en un minimum de communications [47]. Mesurer le temps d'exécution ne pose pas de problème ; par contre un point important pour le propos de l'algorithme était de simuler une communication de faible débit, ce qui a été fait en ajoutant du contrôle de débit artificiel sur des interfaces locales de la machine cliente, avec quelques manipulations des communications initiées par le noyau Linux.

4.2.2 Conséquences

L'activité expérimentale pousse à l'humilité. Les résultats sont souvent délicats à obtenir, instables car dépendant de nombreux facteurs pas tous maîtrisés, comme la charge de la machine ou du réseau, les interactions avec les autres processus, etc. Il faut donc être tour à tour attentif, soupçonneux, patient : si l'on sait quand la campagne de mesure commence, la date de fin est toujours très incertaine.

Ensuite, la mesure de la performance nécessite certes des outils, mais aussi une compréhension fine de ce qui est réellement mesuré, et des différentes contraintes qui bornent la vitesse des calculs. Quand je fais le rapprochement de deux relations à distance pour différents paramètres, suis-je en train de mesurer la performance des disques des machines hébergeant la base, ou bien la performance du réseau qui m'y relie en latence ou en débit, la vitesse de l'interpréteur d'expressions du système de base de données, ou encore la vitesse de l'interpréteur perl qui exécute mon algorithme sur le client ? Il est bien sûr nécessaire de répondre à ce type de question pour pouvoir dire que l'on comprend les mesures obtenues.

Malgré ces difficultés, la construction d'une campagne de mesure est passionnante en elle-même : à partir d'une hypothèse fondée sur une analyse de la complexité de l'algorithme exécuté et éventuellement un modèle qui permet de relier cette complexité aux différents facteurs composant l'environnement de l'expérience, on fait des mesures, et on valide ces mesures par rapport au prévision du modèle. Je ne pense pas qu'il me soit arrivé un jour de faire une seule itération pour arriver à un résultat satisfaisant et conforme. Il y a toujours eu un imprévu, une subtilité qui avait été négligée à tort et qui devenait évidente après l'expérience, ou un résultat étrange qui nécessitait de reprendre le modèle pour l'affiner ou bien en comprendre les limites ou les hypothèses implicites.

Les mesures de performance des transformations de code effectuées avec Youcef Bouchebaba en sont une illustration : même si on attendait des mesures linéaires avec des points bien alignés sur une droite fonction de la taille du domaine de calcul et avec une pente prévisible, les décrochements éventuels à l'occasion du passage d'un niveau de cache à un autre par exemple étaient éventuellement eux plus inattendus.

Un dernier aspect des expériences concerne leur présentation intelligible dans un article ou au cours d'une présentation. Il s'agit de fait d'une démarche pédagogique, qui doit permettre de mettre en exergue ce qu'on veut montrer dans un espace souvent limité.

4.3 Conclusion

Les implémentations et les expériences constituent des investissements en temps considérables pour valider les propositions faites. Elles paraissent indispensables à ce domaine de recherche et à la manière dont je l'envisage. Par contre, leur impact sur les publications, l'aune de la recherche, semble globallement négatif : les semaines passées à assurer que les débordements sont détectés dans les algorithmes de manipulation de polyèdres sont irrémédiablement perdues.

Chapitre 5

Connaissances

Avoir présent à l'esprit un objet réel ou vrai.
(dictionnaire Le Petit Robert*)*

J'ai eu l'occasion de partager mes connaissances dans différents contextes : second cycle d'écoles d'ingénieur, présentations de recherches lors de colloques ou de séminaires, et enfin la diffusion de logiciels libres. Pour la partie enseignement, la diversité des publics, en âge, en niveau et en objectifs, et celle des sujets abordés ont toujours été pour moi une source d'enrichissement. Mes principales activités dans ce domaine restent quand même dans le domaine des technologies de l'information, en allant des mathématiques qui en sont un fondement à l'utilisation quotidienne pratique d'un ordinateur.

Mes principaux publics d'élèves sont ceux des écoles d'ingénieur, soit dans le cycle ingénieur lui-même (ingénieur civil, corps des mines et ISIA à l'École des mines de Paris, ESIGETEL), soit dans des cycles de formations plus professionnalisantes comme des Mastères Spécialisés (Ingénierie des Applications Réseaux-Multimedia – IAR2M, Management des Systèmes d'Informations et des Technologies – MSIT), et enfin dans le cadre de la formation continue données directement ou via d'autres écoles d'ingénieurs comme l'ENST Bretagne. Le public de la formation continue aborde un sujet souvent pratique lors de stages courts et denses, et est évidemment très différent du public des écoles : plus centré sur ses problèmes, plus intéressé par des solutions pratiques et opérationnelles, et par contre moins intéressés par les questions théoriques ou abstraites.

La diffusion des connaissances, théoriques ou pratiques, devant une audience plus ou moins attentive, est une épreuve qui peut tout autant servir de tremplin pour étendre ses propres connaissances en creusant un sujet : j'ai constaté comme tous mes collègues qu'entre connaître un sujet et être capable de le faire comprendre à un élève ou un collègue, il y a un pas significatif dans le niveau de compréhension que l'on doit en avoir. Ces présentations ont donc été pour moi l'occasion d'approfondir des sujets, et d'approfondir ma connaissance et ma technique du domaine de la pédagogie, jusqu'à faire des propositions pratiques pour un outil [43, 44, 42]. Enfin, il est aussi possible de diffuser des connaissances ou une expérience sous forme de contributions à des logiciels libres comme Apache (serveur web), PostgreSQL (serveur de base de données relationnelle) ou Subversion (outil de gestion de révisions).

5.1 Séminaires et tutoriels

Les séminaires de recherche et les conférences grand public constituent un format que j'apprécie : avec typiquement une heure devant soi, il est possible de se donner un temps suffisant pour expliquer des aspects fins des problématiques présentées. C'est un bon intermédiaire entre la session dans une conférence, où il faut parler vite pour donner ses résultats, et le cours étalé sur plusieurs jours : l'unité de lieu et de temps ne bloque pas la profondeur de l'exposé. De plus, les sujets sur lesquels j'ai donné des séminaires sont de plus en rapport avec mes thèmes de recherche, ce qui est une motivation supplémentaire.

5.1.1 Séminaires de recherche

Dès le début de mes travaux j'ai été amené à parler à de nombreuses reprises du *High Performance Fortran*, la première fois pour en présenter les particularités à la communauté du data-parallélisme en France à l'occasion d'une réunion du groupe Paradigme [17]. De telles présentations ont été renouvelées à l'IDRIS [21], au CEA [28], à un atelier d'automne PRISM [29] lors d'un tutoriel HPF à Renpar'7 [23] en français et à Europar'99 [33] en anglais.

Au delà de la simple présentation du langage HPF, j'ai participé à la présentation des techniques de compilation de HPF [67] lors de l'école de printemps PRS en 1996 avec Cécile Germain et Jean-Louis Pazat. La compilation des replacements a été présenté à une journée PRS [66]. La compilation des communications a également fait l'objet d'un séminaire Rumeur en 1996 [24]. Enfin, j'ai pu exprimer mes vues sur le langage lors d'un séminaire de recherche de l'INRIA [30].

J'ai présenté les travaux commun avec Julien Zory au séminaire Dagstuhl 99161 [34]. Les travaux FREIA on fait l'objet de plusieurs séminaires [69, 71].

5.1.2 Grand public

Je dois être un bon client des spammeurs puisque je reçois typiquement plus de 1000 messages non souhaités par jour. C'est donc naturellement que je me suis intéressé à ce sujet et que j'ai proposé des contributions présentées à la section 2.3.2.

L'étude de ce phénomène touche l'économie, les sociétés civile et mafieuse, la mondialisation et la misère mondiale, la psychologie, mais aussi des aspects plus techniques comme les protocoles de messagerie et le fonctionnement général d'Internet. Les solutions proposées couvrent elles aussi ces aspects : protocoles cryptographiques divers, lois et poursuites judiciaires, filtres statistiques, systèmes collaboratifs pour identifier les messages, outils d'analyses, etc.

La DSI (Direction des systèmes d'information) de RFF (Réseau ferré de France, établissement public gérant les voies ferrées en France) m'a invité à présenter la problématique du spam au cours de leur séminaire annuel en juin 2006 [48]. Une version améliorée de cette présentation a ensuite été donnée à une réunion mensuelle de l'ANDSI (Association nationale des DSI) [49] en novembre 2006. Enfin, une présentation grand public a eu lieu à l'occasion du cycle bellifontain de conférences « le goût du savoir » en février 2007 [53]. Un nouvelle formation mise à jour lors d'une journée de la DSI de l'INSERM en juin 2010 [61].

La DSI de RFF m'a à nouveau invité en juin 2007 pour son séminaire annuel, cette fois pour deux conférences :

Cryptographie et 'Pataphysique présente [51] les différentes problématiques de la cryptographie sous l'angle des attaques qui fonctionnent : recouvrements de mots de passe, crackage de clefs secrètes, mé-compréhension des mécanismes de signature.

Quelques idées sur la sécurité des réseaux discute [52] les grands principes de la sécurité des réseaux en se fondant sur des exemples d'affaires récentes ou plus anciennes.

La conférence sur la cryptographie a également eu lieu dans le cadre bellifontain [56, 57].

Enfin, dans le cadre des conférences organisée par FedISA, j'ai fait une intervention sur le *Cloud Computing* [60].

5.1.3 Tutoriel logiciels libres

J'ai eu l'occasion d'être invité par Télécom Bretagne aux journées *Autour du Libre* en 2000 pour y présenter les possibilités du langage de script `perl` [35]. Ce tutoriel perl a servi de base à une offre de stage de formation continue ajouté au catalogue de l'ENST-Bretagne et est régulièrement demandée.

5.2 Enseignement

J'ai la chance rare d'enseigner dans des conditions exceptionnelles : peu d'élèves, de haut niveau car très sélectionnés, certains d'entre eux passionnés par les matières qui leurs sont proposées, et auxquels je dispense un enseignement plus proche du tutorat que de l'enseignement de masse.

Le domaine de l'informatique n'est pas limité à un simple *savoir* scientifique, même si la connaissance de résultats théoriques ou d'algorithmes est très utile. Il s'étend à un *savoir-faire* qui ne peut être démontré qu'en *faisant* soi-même pour vérifier si les méthodes proposées fonctionnent réellement dans la pratique. Une partie des domaines touche parfois à la *'pataphysique*, la science humoristique et paradoxale des exceptions qui n'en est pas une puisqu'elle étudie ce qui ne fonctionne pas, n'est pas reproductible. De plus, la carrière évoluant, les problématiques du *faire-savoir* pour valoriser ce qui est réalisé et du *faire-faire* managérial se posent également. En conséquence, la transmission passe beaucoup par des séances pratiques, des projets, des stages en entreprise.

L'enseignement a donc représenté une part importante de mon activité ces vingt dernières années, tant par choix que par opportunité. Voici à titre d'exemple les matières enseignées au cours de l'année 2012-2013. J'ai par ailleurs des responsabilités organisationnelles pour plusieurs cycles de l'École des mines de Paris :

Cycle ingénieur civil, tronc commun introduction à l'informatique.

Cycle ingénieur civil, enseignements spécialisés systèmes d'information, introduction aux réseaux.

Cycle ingénieur civil, Option MSI langage de script, cryptographie, gestion de sources, sécurité des réseaux, compilation.

MS MSIT exécutif modèles d'affaire.

Cycle du corps des mines, PESTO SI modèles d'affaire.

À ces formes d'enseignement classiques s'ajoute le suivi de projets et de stages en entreprise pour des optionnaires et des mastériens.

Il est intéressant de noter que les travaux d'étudiants, même dans le cadre de formation professionnalisante, peuvent donner lieu à des publications [81]. Le développement de formations avec des partenaires est également l'occasion de répondre à des appels d'offre sur des sujets de recherche plus orientés gestion, comme ce fut le cas concernant les modèles d'affaire, et là encore de faire quelques publications [77].

Globalement, les enseignements que j'ai assurés jusqu'à présent se focalisent sur l'ingénierie logicielle, les bases de données, le système, les réseaux et un peu de théorie, principalement à un niveau second cycle informatique. Une partie de mes supports de cours est diffusée [50] dans le cadre de l'opération *ParisTech Libres Savoirs* qui vise à donner accès gratuitement à des supports correspondant à des cours actuels.

5.3 Outils pédagogiques

À force d'organiser des séances pratiques avec des groupes d'élèves, on se trouve confronté aux vérités élémentaires de la pédagogie et de la psychologie : un élève a besoin de retours rapides pour valider ses tentatives de réponses aux questions posées. En l'absence de retour, immédiat ou au moins rapide, l'élève suppose soit que sa réponse est bonne sans y passer plus de temps, soit il se démotive. Un autre point important est que l'étudiant ne retourne pratiquement jamais sur ses réalisations passées : les corrections proposées ne sont pratiquement pas regardées, ce que j'ai vérifié en les mettant en ligne et en comptant les accès des élèves curieux... l'étudiant est passé à autre chose ; il a tourné la page. Il existe bien sûr des contre-exemples, mais ces exceptions restent largement minoritaires.

Un autre problème concerne l'enseignement du langage SQL utilisé pour interroger une base de données relationnelle : l'algèbre relationnelle induit de nombreuses requêtes équivalentes, et il est difficile de comparer une requête référence à une requête produite par l'étudiant. Pour répondre à cette problématique, j'ai développé un outil web [42, 44, 43] de validation automatique des questions d'un TP. L'outil *Corrector* permet à l'élève de valider ses réponses au fur et à mesure avec des techniques de corrections adaptées aux différents types de questions : les requêtes SQL sont exécutées afin de comparer la relation résultante ; la réponse de l'élève issue de l'interaction avec un serveur est vérifiée par une méthode cryptographique ; des mots clefs sont cherchés dans la réponse de l'élève...

L'utilisation de cet outil, en plus d'offrir un retour immédiat à l'élève et d'ajouter ainsi une motivation, permet de gérer de manière beaucoup plus facile des groupes d'élèves plus grands, où l'enseignant n'intervient plus que pour assister les étudiants bloqués plutôt que pour intervenir auprès de tous les étudiants à fin de validation.

Par contre, il n'est pas un outil universel et définitif. Tout d'abord, la mise au point de la correction automatique n'est pas toujours facile, en particulier pour des questions ouvertes pour lesquelles la correction est fondée sur une série de mots clefs dont le choix varie d'un étudiant à l'autre. Ensuite, certains élèves se focalisent sur l'obtention des points au détriment de l'acquisition des connaissances et des savoir-faire : tentative de copier-coller d'un dictionnaire entier pour trouver à coup sûr les mots clefs, recopie des réponses d'un autre étudiant sans se rendre compte qu'elle contient une identification de cet étudiant qui permet de détecter la supercherie, etc. La vigilance de l'enseignant est donc toujours requise.

Un autre exemple d'outil réalisé à une fin pédagogique est un comparateur de projets qui permet de mesurer la distance entre différents codes sources pour détecter des inspirations un peu directes dans les projets d'élèves. Les mesures de distance sont indépendantes des choix particuliers d'identificateurs ou de commentaires qui peuvent facilement être changés, et se focalisent sur des éléments de structure dont une signature caractéristique est construite sous forme d'un vecteur. Ces vecteurs sont ensuite comparés avec plusieurs mesures qui se focalisent sur l'inclusion de code, la reprise de code, etc. Un tri manuel est ensuite effectué sur les alertes soulevées par l'analyse automatique.

5.4 Contributions aux logiciels libres

Une dernière manière de diffuser ses connaissances vers un large public est de contribuer à ou d'écrire des logiciels libres, à la fois gratuits et disponibles sous forme de sources. C'est le cas des contributions dans Newgen, Linear/C3 et PIPS présentées à la section 4.1. J'ai aussi abordé et complété différents logiciels selon mes besoins souvent professionnels et quelquefois personnels. Dans ces réalisations comme dans mes travaux de recherche j'ai été guidé par des soucis de performance, d'élégance ou de connaissance...On peut rapidement citer pour les contributions à des logiciels libres largement utilisés :

Apache httpd *(logiciel serveur web)* le module macro pour faciliter l'écriture des fichiers de configurations en autorisant la définition de macros [32], en proposant une syntaxe qui s'intègre dans celle du fichier de configuration lui-même (pour éviter des outils extérieurs comme `cpp` ou `m4`). Ce module est fourni sous forme de package avec les principales distributions Linux, et sera directement intégré dans le serveur à partir de la version 2.4.6 sortie le 22 juillet 2013.

Subversion (SVN, outil de gestion de révisions) Cet outil est utilisé pour l'administration système, pour le logiciel PIPS mais aussi comme sujet d'enseignement en tant que tel :

- La principale contribution est la francisation [68] de la documentation interactive accessible à partir de la ligne de commande, en collaboration avec Marcel Gosselin et d'autres.
- Lors du transfert de PIPS sous subversion, j'ai développé un script de reprise de l'historique à partir de RCS `rcs2svn` [41] ainsi qu'un script de création de branches de développement `svn_branch.sh`.
- Lors du passage des fichiers systèmes du CRI sous subversion, j'ai eu besoin d'un outil de fusion de dépôts `svn-merge-repos` [46].
- vérifications automatiques paramétrables avant commit pour SVN `svn-pre-commit` [59] développé pour les dépôts de PIPS.
- utilisation de SVN conjointement à d'autres outils, en particulier `make`, `rsync` et `ssh`, pour partager, sauvegarder et déployer des configurations systèmes [54].

PostgreSQL *(serveur de base de données)* Diverses contributions directes ou indirectes, ainsi que d'outils autour de ce logiciel que j'utilise abondamment dans le cadre de différents enseignements, parmi lesquelles je peux citer :

- adaptation de l'infrastructure de compilation à l'ajout de nouvelles extensions sans avoir à repartir des sources de la base de données : `pgxs` [37].
- un autre exemple est l'ajout d'un petit mot clef `ALSO` dans des déclarations de façon à éclaircir leur comportement par défaut.
- affichage explicite de la localisation des erreurs de syntaxe, très utile pour les élèves.
- ajout d'agrégations booléennes et binaires (AND, OR).

- diverses corrections dans le type date.
- extension `tuplock` [39] pour verrouiller des tuples dans un table, *i.e.* empêcher la modification de leur contenu.
- comparaison à distance de relations en minimisant les communications, par exemple pour détecter les tuples différents lors d'une reprise après l'interruption d'une synchronisation de deux bases, avec `pg_comparator` [36, 47, 58, 62].
- contrôle d'un schéma de données à la recherche d'erreurs éventuelles avec `pg-advisor` [38] puis Salix Babylonica [1, 63, 64].

Chapitre 6

Projet de Recherche

Mon projet de recherche prolonge l'axe principal de mes travaux précédents : *concevoir des langages, éventuellement spécifiques à des domaines d'application, et les compiler pour la performance, de manière pragmatique, sans préjuger de leur forme (syntaxe, directives, API, scripts), des cibles matérielles (du processeur embarqué aux super-calculateurs), des objectifs (temps, consommation, ...) des problématiques spécifiques (ordonnancement, hiérarchie mémoire, communications, ...) ni des outils particuliers (compilation classique, outils polyédriques, JIT...).* Je montre la pertinence toujours actuelle de cet axe. J'illustre les idées proposées par des exemples issus de mes travaux passés ou récents, ou de possibles travaux futurs présentés sous forme de sujets de stage ingénieur, de stage de master recherche ou de thèse.

6.1 Contexte et enjeux

Cette première partie discute la problématique de la compilation, décrit quelques architectures matérielles récentes qui sont des cibles actuelles et futures, aborde les enjeux et présente les méthodes envisagées pour résoudre les problèmes ouverts.

6.1.1 Compilation

La compilation suppose de définir de manière pertinente deux frontières : l'une du côté du langage donné au développeur applicatif ou même à un autre compilateur de plus haut niveau, qui doivent pouvoir exprimer leurs besoins, et l'autre du côté de la cible, souvent mais pas nécessairement matérielle, impliquant éventuellement un environnement d'exécution riche, et dont le compilateur doit profiter au mieux.

En amont, il s'agit de savoir quelle est la bonne définition du langage pour permettre à l'utilisateur d'exprimer ce qu'il veut dans son domaine d'application, et lui permettre d'aider le compilateur dans des tâches qui lui sont particulièrement difficiles, et ce, sans contraindre inutilement la liberté de l'utilisateur. Ce langage peut être complètement spécifique, ou s'appuyer au contraire sur un langage largement utilisé comme C, mais restreint par exemple à une API particulière. En aval, il s'agit de définir ce que doivent permettre l'environnement d'exécution et le matériel, sur lesquels le code généré va s'appuyer. Un effet de bord positif de bonnes décisions prises en amont et en aval est que le compilateur peut devenir un outil moins onéreux à développer, et rapidement disponible quand une nouvelle cible arrive.

Ces aspects ont été abordés dans le cadre du projet ANR FREIA auquel j'ai participé et dont certains résultats ont été présentés et discutés dans les chapitres précédentes (sections 2.2.4, 3.2, 3.3.3, 4.2). Ce projet visait à fournir un environnement de développement complet pour des applications de traitement d'image, incluant leur compilation vers différentes classes d'accélérateurs matériels. L'un des enjeux est le dialogue permanent nécessaire avec les spécialistes du domaine applicatif pour l'aspect langage (quels opérateurs, quelles fonctions, quelle syntaxe) et les concepteurs du matériel (quelles opérations, quelles performances cibles, quel équilibre entre les différentes composantes de l'architecture matérielle). Ce dialogue n'est bien sûr pas uniquement limité aux interactions avec le compilateur : les concepteurs du matériel doivent aussi s'inspirer du domaine applicatif pour bien comprendre le fonctionnement des applications pour déterminer les caractéristiques souhaitables du matériel. Par contre, la conversation est

nécessaire pour définir une bonne répartition entre matériel, environnement d'exécution, compilateur et langage de manière à servir aux mieux les applications et leurs développeurs.

6.1.2 Contexte technologique

La progression des performances des microprocesseurs (calcul, mémoire) a suivi la loi de Moore de doublement périodique sur la fréquence et la densité d'intégration depuis plusieurs décennies. La progression des fréquences a cessé vers 2002 à cause du mur thermique, les processeurs ne pouvant plus dissiper l'énergie dégagée. La densité des circuits continue néanmoins de progresser, par exemple avec une finesse de gravure en 2012 de 28 nm pour TSMC et 22 nm pour Intel, mais cette progression va également s'arrêter sur les effets quantiques des composants, sans doute dans la décennie à venir, à moins d'une rupture technologique majeure non encore entrevue. Par contre, la vitesse de la lumière, qui induit un temps minimal de propagation des signaux, a fait sentir son effet depuis longtemps sur la latence des accès aux données. Un signal parcourt typiquement quelques centimètres entre deux tics d'un processeur moderne : dès lors que le signal ne peut parcourir qu'une distance faible entre deux instructions, les données doivent être très proches pour être utilisées immédiatement. L'objectif de nombreuses transformations de programme opérées par les compilateurs est donc de diminuer les communications nécessaires en augmentant la localité des calculs ou en masquant la latence des accès mémoire pour maintenir le processeur en activité.

Si on prend une perspective plus historique sur cette progression, force est de constater que les objectifs, les difficultés à lever, restent les mêmes au cours du temps : rapprocher les applications qui ont besoin de performance en vitesse, prix et consommation de ressources des architectures matérielles complexes capables de les fournir...

Pour ce qui concerne mes travaux passés, j'ai abordé les points suivants au cours de ma thèse et des trois thèses que j'ai co-encadrées :

- le développement du langage HPF pour favoriser l'usage des supercalculateurs parallèles à mémoire distribuée, avec la génération explicite des communications et de l'adressage de données réparties sur toute la machine ;
- les transformations d'expressions élémentaires pour tirer parti du parallélisme interne proposé par les processeurs (super scalaire, VLIW), en prenant en compte la pression de registre par exemple ;
- l'application de transformations de programme très agressives sur des applications simples de traitement du signal pour augmenter la durée d'utilisation sur batterie d'applications embarquées ;
- la compilation pour des accélérateurs de type GPGPU ;
- une chaîne de compilation pour des applications de traitement d'image embarquées exécutés avec des accélérateurs matériels spécifiques.

Dans tous les cas, ces travaux portent sur les analyses et transformations automatiques de programme pour les adapter à ces matériels et à des objectifs multiples de performance en vitesse, énergie, surface... pour un coût de développement aussi faible que possible. Il est intéressant de noter que la préoccupation environnementale se focalise maintenant aussi sur les supercalculateurs, qui sont non seulement classés par leur puissance de calcul depuis 1993 (`www.top500.org` novembre 2012 : Cray XK7, Linpack 17,6 Tflop/s) mais aussi par leur rendement énergétique depuis 2007 (`www.green500.org` novembre 2012 : Appro Int. Inc. Xtreme-X, Linpack 2500 Mflop/s/W).

Mes travaux présents et futurs continuent dans cette voie, avec pour cible des accélérateurs de calcul spécialisés, par exemple pour le domaine du traitement image ou du traitement du signal.

6.1.3 Le matériel

Voici quelques exemples de matériels cibles actuels.

Processeurs de flux (SPoC – CMM/MINES ParisTech)

Le processeur SPoC est spécifique au traitement morphologique d'images. Il a été développé par Christophe Clienti dans le cadre de sa thèse de doctorat (*Architectures flot de données dédiées au traitement d'images par Morphologie Mathématique*, École des mines de Paris, 2009). Des fonctions de traitement de haut niveau sont directement incluses sur le silicium, et un bus interne permet le transfert et l'accumulation des données entre ces traitements de manière continue et efficace, sous forme d'un long pipeline vectoriel. Un point clef de cet accélérateur est qu'il incorpore le traitement algorithmique des bords des images de manière transparente.

Processeur SIMD (Terapix – THALES)

Il s'agit d'un accélérateur matériel pour le calcul embarqué développé par l'industriel THALES. Un vecteur de 128 processeurs élémentaires permet des calculs en mode SIMD strict, qui permet d'exploiter un parallélisme de données et des opérations aux voisinages. La mémoire interne est limitée à quelques milliers de mots par processeur élémentaire et doit être gérée explicitement par le compilateur ou l'environnement d'exécution. Ici encore, les possibilités d'intégrer directement au matériel et à l'exécutif le traitement des bords est un facteur important de simplification pour la compilation.

Processeurs reconfigurables

Les matériels fondés sur des FPGA *(Field-Programmable Gate Array)* peuvent être dynamiquement reconfigurables : on peut les adapter dynamiquement aux besoins des applications, en fonction par exemple d'un mode d'utilisation ou d'une phase particulière de calcul. Cette possibilité ouvre des perspectives nouvelles pour la compilation et l'optimisation d'applications.

L'inconvénient de ces matériels, comparé à des circuit intégrés spécifiques, est la consommation supérieure de surface de silicium, la consommation énergétique plus importante et la vitesse moindre obtenue in fine. Pour des usages particuliers, un circuit intégré spécifique est par contre plus long à concevoir et industrialiser, et plus onéreux pour des faibles ou moyens volumes, ce qui laisse un espace à ces processeurs entre les ASIC et les processeurs généralistes programmables.

Cartes graphiques (GPGPU)

Les cartes graphiques récentes, ou leurs évolutions, permettent de faire des calculs généraux en simple voire double précision (GPGPU : *General Purpose Graphical Processing Unit*). L'unité Tesla S2050 de NVIDIA (Q2 2010) propose par exemple une puissance crête de 2 Tflop/s en double précision. Le fait même que ces cartes implémentent des calculs flottants doubles, qui ne sont d'aucun usage pour une carte graphique destinée à l'affichage, montre le positionnement de NVIDIA sur le marché du calcul haute performance. Certains supercalculateurs actuel, comme le Tianhe-1A en Chine, sont fondés sur ce type de cartes.

L'architecture de ces cartes est pour l'essentiel du SIMD *multi-threadé*. La programmation ou la compilation vers ces machines doit prendre en compte les nombreux processeurs élémentaires disponibles, les contraintes de gestion mémoire fortes qui impliquent souvent une allocation et une gestion manuelles des données, ainsi qu'une attention soutenue aux mouvements de données entre la machine hôte et la carte, mais aussi au sein de la carte elle-même avec des contraintes d'alignement.

Processeurs généralistes multi-cœurs (multi/many-core)

Les processeurs généralistes intègrent des capacités de calcul parallèle afin de tirer partie de l'intégration toujours plus poussée des transistors, soit avec une logique homogène (Intel dual-core, quad-core, hexa-core, octo-core), soit avec une logique de coprocesseurs de calcul hétérogènes (IBM CELL). Les processeurs de laboratoires montent à plusieurs dizaines de cœurs (Intel, 48 cœurs en 2010) voire plusieurs centaines de cœurs (Kalray MPPA ManyCore avec 256 cœurs en 2012). Le langage OpenCL permet de développer des codes devant tourner sur des GPGPU ou des multi-cœurs à mémoire partagée ; par contre il semble éventuellement plus adapté au premier type de matériel qu'au second.

Processeurs multi-core hétérogènes

L'arrivée des téléphone intelligents *(smartphones)*, par exemple la série des Galaxy S de Samsung, a créé un marché pour des processeurs à la fois rapides et très peu gourmands en puissance. L'approche *big.LITTLE* de ARM consiste à intégrer plusieurs processeurs, l'un faible consommation pour les tâches de fond, l'autre beaucoup plus puissant, éventuellement multi-cœur, qui n'est déclanché que quand la charge le requiert.

6.1.4 Enjeux

Les problèmes à résoudre pour obtenir des performances de ces nouveaux matériels sont très similaires à ceux des supercalculateurs disponibles il y a vingt ans : nombre de processeurs, latence des communications et nécessité d'une grande localité des données pour que les processeurs évitent la famine et puissent être actifs la plupart du temps. Le problème est pratiquement resté le même. La pression économique est cependant plus forte parce que ces matériels, qui étaient l'exception, sont maintenant la norme. Ce ne sont plus seulement des applications de calcul scientifique qui doivent profiter de ces matériels mais l'écosystème complet des applications informatiques, tant du côté des machines individuelles que des serveurs. Les fabriquants semblent craindre de ne pouvoir vendre leurs matériels s'ils n'affichent pas des performances supérieures à celles des générations précédentes, dans un marché habitué aux progressions de performance exponentielles.

Il faut de garder à l'esprit que la communauté scientifique n'a pas trouvé de bonnes solutions pour une programmation efficace et facile des machines à mémoire distribuée, qui reste un problème difficile. Les performances sont en général atteintes avec une programmation manuelle des applications par passage de message. La démocratisation de ces machines n'a pas rendu les problèmes plus simples, et nous ne sommes pas devenus plus intelligents pour les résoudre. Il est donc important d'être réaliste et de viser des objectifs atteignables pour éviter de futures déconvenues, comme l'échec du langage HPF.

La mise en place de solutions réalistes a besoin de s'appuyer sur des hypothèses restrictives (domaine d'application, processeurs particuliers) sur lesquelles s'appuyer pour se déployer. Quand un bon équilibre est trouvé entre une architecture et un domaine d'application, il peut être intéressant de reproduire sur une autre architecture le fonctionnement de l'architecture performante. Par exemple, dans la mesure où de bons résultats sont obtenus pour des applications de traitement d'image avec une approche vectorielle en flux, il est intéressant d'étudier les possibilités de programmation d'une machine multi-cœur dans ce mode particulier, c'est à dire viser l'exploitation d'un parallélisme de tâches plutôt qu'un parallélisme de données plus classique.

6.1.5 Méthodologie

Mes travaux passés et présents s'appuient sur des applications représentatives qui sont étudiées de près pour sélectionner les transformations et techniques qui s'avèreront les plus profitables. Le fait de s'appuyer sur des applications réelles plutôt que des cas d'école ajoute bien sûr une difficulté aux développements. Cette difficulté accrue est aussi une chance, parce qu'elle permet de se rendre compte des problèmes réels et spécifiques qui offrent des opportunités d'approfondir les techniques proposées et de les adapter. Pour illustrer cet aspect dans ma recherche passée, on peut citer le travail sur le traitement des entrées-sorties du langage HPF : peu d'articles de recherche se sont portés sur ces aspect particuliers, alors qu'il est indispensable de sauvegarder les résultats rapidement pour ne pas perdre le gain de temps de calcul.

Un objectif commun à mes travaux est de démontrer la validité des techniques sous forme de prototypes logiciels solides, le cas échéant distribués librement comme c'est le cas du logiciel PIPS. Cet objectif est ambitieux et coûteux : il est plus facile de faire une transformation à la main pour démontrer son intérêt que de l'implémenter dans le cas général pour des langages complexes comme Fortran ou C. De même, le développement d'un logiciel de production industriel plutôt qu'un démonstrateur est compliqué par l'interaction entre les passes développées par les autres utilisateurs de la plateforme.

C'est par exemple le cas du logiciel PIPS utilisé pour mes travaux et ceux de mes doctorants, qui a été utilisé comme base par une jeune entreprise, *HPC Project*, pour proposer des extensions et des services autour du calcul haute performance. Il n'est pas possible de modifier brutalement le logiciel sans des tests extensifs, qui n'auraient pas d'objet si j'en étais le seul utilisateur. Par contre, la démonstration implémentée ouvre la possibilité de construire de nouvelles analyses ou transformations qui réutilisent ces techniques et permettent de traiter des cas beaucoup plus réalistes, voire des codes industriels réels pour

ce qui concerne ma thèse, celle de Julien Zory et dans une moindre mesure celle de Youcef Bouchebaba qui ciblait un domaine d'application plus restreint.

6.2 Sujets de recherche

On trouvera ci-après une série de sujets de thèses, de stages ingénieur ou de master recherche, qui illustrent le projet général discuté ci-dessus. Certains de ces sujets sont très exploratoires. Ils proposent plus une idée de point de départ pour un développement dont le contenu dépendra de ce que cette phase rapporte. Les sujets plus recherche impliquent une phase préliminaire d'étude bibliographique. D'autres sujets complètent et étendent des travaux déjà effectués ou amorcés. Les derniers sujets sont plus conçus pour l'ingénieur. Ils illustrent le type d'investissement nécessaire pour développer une collaboration industrielle.

6.2.1 Sujets de thèse

Les deux premiers sujets correspondent à des travaux en cours.

Analyse de pointeurs

Thèse soutenue en juin 2013, sujet traité par Amira Mensi, co-encadrée avec François Irigoin et Corinne Ancourt.

Les différentes analyses statiques développées pour Fortran 77 dans PIPS (transformers, préconditions, régions, dépendances...) ont dû être adaptées pour C dont le support syntaxique a été ajouté. Ces analyses fonctionnent bien actuellement lorsque les programmes C ont une sémantique compatible avec Fortran, mais doivent être modifiées pour prendre en compte l'usage des pointeurs. Par exemple, les calculs de dépendances doivent considérer les possibilités d'*aliasing* ou non entre structures de données. Cela nécessite le développement de nouvelles analyses interprocédurales spécifiques, qui supposent de faire des choix sur les hypothèses éventuellement restrictives (par exemple le respect ou non du système de type), la représentation des informations, l'implémentation des opérateurs, avec des impacts à la fois sur la précision des résultats et le coût global de l'analyse.

Compilation orientée énergie pour systèmes logiciels communicants

Thèse en cours, traitée par Karel de Vogeleer, co-encadrée avec Gérard Memmi et Pierre Jouvelot.

La gestion de l'énergie structure la société. Dans le domaine des interactions humaines, la diffusion rapide de moyens informatiques ubiquitaires et portatifs (smartphones, netbooks, RFID) impose de concevoir des dispositifs logiciels et matériels qui intègrent dès l'origine les aspects énergétiques, de façon à offrir autonomie et légèreté dans une approche respectueuse de l'environnement. À partir d'une modélisation énergétique des dispositifs matériels et de leurs coûts d'utilisation (instruction, mémoire, transfert de données), il s'agit de concevoir des algorithmes de compilation qui minimisent la consommation énergétique tout en préservant les performances attendues en temps de calcul et occupation mémoire.

Macro-compilation d'applications de traitement du signal

Sujet de thèse.

Les applications de traitement du signal peuvent être vues comme des fonctions appliquées à des séries spatiales et temporelles vues sous forme de vecteurs, matrices ou tenseurs. Elles peuvent être typiquement développées dans un environnement de type Mathlab ou Scilab, s'appuyer sur des API spécifiques (par exemple VSIPL), ou sur un langage spécifique à un domaine comme FAUST dédié à la musique. Leur forme induit intrinsèquement de fortes opportunités de faire des calculs en parallèle, en exploitant le parallélisme de données et/ou de tâches.

L'objectif est de partir d'une application décrite à travers une vue fonctionnelle de haut niveau, d'effectuer à ce niveau des optimisations globales sur l'application (élimination de code mort, détection de calculs redondants ou d'invariants) puis de générer un code optimisé s'appuyant sur un exécutif portable sur GPGPU, par exemple fondé sur OpenCL. Ce travail est une forme de généralisation du projet ANR FREIA décrit précédemment à un domaine d'application plus large.

Optimisation énergétique d'applications informatiques

Sujet de thèse, exploratoire.

Le domaine de l'informatique embarquée (ordinateurs portables, téléphones intelligents ou non, véhicules électriques, objets divers...) est très sensible à la consommation énergétique, en particulier quand la source d'énergie est limitée. D'un autre côté, certaines applications ont des contraintes de temps-réel qui imposent des performances minimales pour que l'application soit utile.

De nombreux facteurs influent sur cette consommation pour un système particulier, comme la technologie du matériel lui-même, la fréquence de fonctionnement qui peut être ajustée, la possibilité de désactiver certains composants inutilisés. La manière dont l'application elle-même fonctionne, en particulier son trafic mémoire, mais aussi les instructions assembleur particulières sélectionnées, ont aussi une influence sur la consommation.

L'objectif est de développer des analyses automatiques ou des instrumentations pour évaluer la consommation énergétique d'applications informatiques fonctionnant sur des matériels particuliers (processeurs multicœur, GPGPU, accélérateurs spécialisés, SoC, processeurs embarqués type ARM Cortex-A9), puis de proposer des transformations de programme et éventuellement un environnement d'exécution associé afin réduire cette consommation pour des classes d'applications et de matériel.

Compilation optimisées de règles de parefeu

Sujet exploratoire, master recherche, potentiel de thèse.

Les pare-feux constituent un élément important de sécurité des réseaux informatiques. Le filtrage des paquets est en général défini par des listes de règles composées d'une conjonction de conditions booléennes et d'une action engagée si les conditions sont toutes vérifiées. Par exemple, l'outil `iptables` permet de configurer le pare-feu intégré au système d'exploitation Linux. Ces listes peuvent être de taille importante. Elles sont appliquées sur les paquets qui transitent sur un nœud du réseau, et occasionnant une charge CPU proportionnelle au flux. La vitesse des traitement détermine la vitesse des connexions fibre optique haut débit par exemple.

L'objectif est ici d'étudier les possibilités d'optimisation un système de règles de pare-feu de manière à limiter la charge du routeur filtrant. Le réarrangement des conditions des tests revient à supposer la commutativité et l'associativité de l'opérateur *et* logique, et de réarranger l'arbre d'évaluation et de décision pour optimiser un critère de performance. Cette optimisation peut prendre en compte des informations statiques, comme le fait qu'une condition soit vraie si une autre l'est déjà (par exemple un test d'appartenance de la destination d'un paquet à un sous-réseau qui peut être inclus dans un sous-réseau déjà testé), comme des informations dynamiques, par exemple un profil statistique des paquets soumis, avec la probabilité que les différentes conditions testées soient vraies. La cible par défaut d'un tel compilateur serait la génération de règles réordonnées et hiérarchisées, via la création de nouvelles chaînes et de branchements entre ces chaînes, par exemple avec `iptables`.

Un point complémentaire qui pourrait aussi être étudié est la possibilité d'intégrer un tel compilateur directement dans le système de gestion de règles du système. Un autre point qu'il peut être nécessaire de prendre en compte est la taille du code généré selon les méthodes d'optimisation envisagées. Une vision dynamique de l'optimisation est aussi envisageable, quand des indications sur le profil statistique des conditions sont connues à l'exécution, pour réarranger dynamiquement les règles et s'adapter aux variations du flux. Enfin, on peut imaginer profiter des manipulations du système de règles pour vérifier leur cohérence, ou au moins détecter des incohérences comme des règles jamais déclanchée par exemple.

6.2.2 Sujets de master recherche et ingénieur

Voici une série de sujet pour des masters recherche.

Génération d'instructions Terapix spécifiques à une application

Sujet de stage master recherche.

Le projet ANR FREIA développe une chaîne de compilation prenant en entrée une API de manipulation d'images utilisant entre autre des opérateurs morphologiques, et produit du code ciblant des accélérateurs matériels (accélérateur vectoriel SPoC, accélérateur SIMD Terapix, environnement d'exécution OpenCL pour GPGPU).

Une phase de compilation dite gros grain gère l'optimisation globale des opérations image d'une application, en particulier vers l'accélérateur SIMD Terapix. Une seconde phase bas niveau génère les routines en assembleur Terapix afin de traiter des calculs réguliers typiques des opérateurs image.

L'objectif est de combiner ces deux phases de niveaux différents en sous-traitant la génération de code assembleur pour les combinaisons d'opérateurs trouvées dans l'application, de manière à utiliser des routines optimisées plutôt que d'appeler les versions génériques. Du point de vue du haut niveau, il s'agit de demander la génération d'instructions spécifiques aux besoin de l'application. Les transformations souhaitables incluent la propagation de valeurs de tableaux constants et le déroulage de boucles internes.

Environnement MPPA ManyCore pour FREIA

Sujet de stage ingénieur ou master recherche.

Le projet ANR FREIA est introduit au premier sujet. Une nouvelle cible est le processeur MPPA (Multi-Purpose Processor Array) Manycore de Kalray, qui propose un architecture de clusters à mémoire distribués, chaque cluster multi-cœur partageant un segment de mémoire. La première version de la puce, MPPA-256, intègre 256 cœurs de calcul. Elle propose deux modèles de programmation, l'un à base de fils communiquants, et l'autre basé sur un flot de tâches.

L'objectif de ce projet est de développer un exécutif MPPA proposant de très bonnes performances. Cet environnement devra en particulier optimiser les transferts de données entre les clusters.

Environnement d'exécution OpenMP pour FREIA

Sujet exploratoire, stage master recherche.

Le projet ANR FREIA est introduit au premier sujet. Le parallélisme multi-cœur des processeurs Intel ou AMD modernes pourrait être exploiter.

L'objectif est de développer un environnement d'exécution fondé sur OpenMP pour FREIA, qui permette d'exploiter les parallélismes de données et de tâches sur les processeurs multi-cœurs.

Fonctions de preuve d'effort

Sujet de stage master recherche.

La preuve d'effort *(proof of work)* est une mesure de prévention des abus d'usage d'un protocole réseau, qui consiste à demander à l'utilisateur d'un service une preuve de son désir en requérant un calcul long à réaliser mais rapide à vérifier.

Ce type de mesure dissuasive de nature économique a été proposé en 1992 par Dwork et Naor. Depuis, différents types de fonctions ont été proposés, dont les performances sont bornées par les calculs ou les accès mémoire. Des critères d'optimalité ont été introduits.

Un exemple simple d'une telle fonction est `hashcash`. Il s'agit de trouver une inversion partielle (nombres de bits en tête d'un hash à zéro) pour une fonction de hachage cryptographique appliquée à une chaîne de caractères composée d'une description du service proposé (la date et le destinataire d'un message) et d'une partie variable.

L'objectif du stage est d'étudier les mécanismes de preuve mathématiques de ces fonctions proposés dans la littérature, et d'en proposer de nouveaux pour des fonctions existantes ou nouvelles. Sur le plan pratique, l'implémentation de certaines fonctions, optimisée pour divers matériels, pourra être aussi envisagée.

Détection avancée d'anomalies dans les programmes

Sujet de stage master recherche, exploratoire.

Le compilateur source à source PIPS procède à des analyses statiques avancées comme les calculs de préconditions ou celle des accès aux tableaux. Ces analyses permettent déjà de détecter certaines anomalies dans les applications analysées, par exemple d'utilisation de données de tableaux non initialisées. L'objectif est d'explorer plus avant l'utilisation de ces analyses pour détecter des anomalies probables ou certaines dans des codes Fortran ou C.

Parallélisation sur GPGPU de fonctions de preuve d'effort

Sujet de stage ingénieur ou master recherche.

Les protocoles cryptographique de preuve d'effort permettent à un serveur de requérir un calcul coûteux de la part d'un client qui demande un service, de manière à dissuader ou limiter les opportunités d'attaques de déni de service qui consisteraient à faire de nombreuses requêtes coûteuses auprès du serveur.

L'objectif est d'étudier les possibilités d'accélération sur GPGPU des différentes fonctions proposées dans la littérature, et de les implémenter pour mesurer les performances obtenues en pratique. Dans un second temps, on cherchera à caractériser les fonctions les plus résistantes à ce portage, et proposer de nouvelles fonctions plus résistantes, selon les points identifiés.

Pilote de compilation FREIA

Sujet de stage ingénieur.

Le projet ANR FREIA est introduit avec le premier sujet. L'objectif est de développer un script pilote *(driver)* pour le compilateur gros grain de FREIA. Cette nouvelle commande doit permettre à l'utilisateur d'appliquer les différentes transformations, optimisations, compilations et éditions de lien de manière complètement automatique et intégrée sur son code, en sélectionnant simplement le matériel cible et le source de son application. Ce pilote peut être écrit en python, langage de script déjà interfacé nativement avec PIPS. Le détail des transformations à appliquer dépend du contenu de l'application elle-même. Il est de plus nécessaire de décider automatiquement le niveau (la fonction) sur laquelle appliquer le processus de compilation ; le pilote peut donc être amené a effectuer quelques analyses limitées sur le code pour prendre ses décisions.

Régénération de C dans PIPS

Sujet de stage ingénieur.

La représentation interne du compilateur source à source PIPS est fondée sur un arbre de syntaxe abstrait proche du source initial. Certaines constructions sont cependant perdues dans la phase de normalisation, comme les `switch`, les sorties de boucles `break` transformées en `goto`, gestion des `return` multiples, etc.

L'objectif est d'améliorer cette phase de regénération du source, pour resortir les constructions perdues, mais aussi de générer des commentaires de documentation (par exemple pour doxygen) issus des résultats des analyses de code faites par PIPS.

Développement d'analyses de schémas relationnels

Sujet de stage ingénieur.

Le logiciel libre *Salix*, développé avec des étudiants du cycle ingénieur civil de MINES ParisTech, permet d'analyser automatiquement des schémas de bases de données relationnelles (MySQL ou PostgreSQL) au travers de requêtes SQL sur les méta données disponibles dans le *information schema* standard de la base, ou des méta-données spécifiques à la base.

L'objectif est de poursuivre le développement de ce logiciel sur les axes suivants :

- en ajoutant de nouvelles analyses dans le domaine des performances (suggestions d'indexes par exemple), des permissions des différents objets de la base, de la version de la base installée et d'autres vérifications à déterminer ;
- en ajoutant des mesures de complexité intrinsèques proposées pour les schémas relationnels ;
- en ajoutant au logiciel une génération de rapport plus complète et conviviale que l'interface actuelle, et qui identifie en particulier de manière précise la source des problèmes détectés ;
- étendre l'outil pour qu'il s'interface avec d'autres bases de données, par exemple Oracle ou MS SQL Server.

6.3 Conclusion

J'ai discuté ci-dessus les enjeux du domaine de la compilation, le type de problèmes envisagés et les outils et approches qui me semblent a priori pertinents pour les résoudre.

Les sujets proposés abordent la compilation (analyse et transformations) pour une large gamme de cibles : multi-core, many-core, GPU, processeurs embarqués SIMD, routeurs. Ils ouvrent également sur des problématiques d'ingénierie comme un script pour piloter une chaîne de compilation ou un re-générateur de code source plus performant. Ils poursuivent différents objectifs : la réduction des temps d'exécution ou de la consommation énergétique, mais aussi la qualité des logiciels.

Je souhaite aborder les problèmes de la compilation en visant des domaines industriels et avec des cibles spécifiques, parce que c'est là qu'il est possible d'apporter une valeur ajoutée sûre et significative, avec l'objectif d'obtenir automatiquement des performances comparable à ou meilleures que les développements manuels, et ce pour un coût bien moindre.

De plus, je compte poursuivre, lorsque l'opportunité se présentera, les travaux de recherche et de développements liés à mes thèmes d'enseignement et à mon travail d'ingénieur système occasionnel. Un dernier aspect important pour moi est le devoir de vulgarisation, de l'informatique mais aussi de sa recherche, auprès de différents publics professionnels ou profanes.

Postface

J'apprécie dans mon travail de recherche le contact avec de nombreuses personnes de haut niveau scientifique et intellectuel, créatives et rigoureuses, auprès desquelles ma pratique s'est enrichie et mon exigence envers moi-même a augmenté, afin de suivre leur exemple. L'enseignement m'est aussi apparu comme une école de rigueur et de communication : le niveau de maîtrise d'un sujet se trouve largement amélioré lorsque l'on doit l'expliquer à d'autres ; de plus, il est nécessaire de trouver des explications, une présentation, qui attirent et intéressent l'étudiant, sans pour autant céder à la facilité du spectacle agréable au contenu pédagogique faible.

En terme d'évolution, la progression, le glissement d'une préoccupation à l'autre s'est toujours faite de manière continue : la compilation du HPF avec ses communications, sa gestion de l'adressage et ses codes locaux a ouvert des problématiques spécifiques d'optimisation associative-commutative du code généré qui était peu efficace ; les mêmes transformations de code s'appliquent aussi bien à l'amélioration de la vitesse que de la consommation des applications ; l'optimisation peut aussi bien porter sur la maximisation des performances que sur leur minimisation.

Soumettre ses écrits au jugement de ses pairs s'est avéré une épreuve, souvent difficile, mais toujours motivante, au moins quand les relecteurs ont réellement travaillé sur l'article soumis et l'ont disséqué, analysé et critiqué, ce qui a été souvent le cas.

Bibliographie

[1] Alexandre Aillos, Samuel Pilot, Shamil Valeev, and Fabien Coelho. *Salix Babylonica* – advices about relational database schemas. `http://www.coelho.net/salix/`, August 2008. Open source software, version 1.0.0 in January 2012;.

[2] Mehdi Amini. *Transformations de programme automatiques et source-à-source pour accélérateurs matériels de type GPU.* – Source-to-Source Automatic Program Transformations for GPU-like Hardware Accelerators. PhD thesis, MINES ParisTech, December 2012. Thèse co-encadrée par Fabien Coelho, Ronan Keryell et François Irigoin.

[3] Mehdi Amini, Fabien Coelho, François Irigoin, and Ronan Keryell. Compilation et optimisation statique des communications hôte-accélérateur. In *Rencontres francophones du Parallélisme (RenPar'20)*, Saint-Malo, France, May 2011.

[4] Mehdi Amini, Fabien Coelho, François Irigoin, and Ronan Keryell. Static compilation analysis for host-accelerator communication optimization. In *24th Int. Workshop on Languages and Compilers for Parallel Computing (LCPC)*, Fort Collins, Colorado, USA, September 2011. Also Technical Report MINES ParisTech A/476/CRI.

[5] Mehdi Amini, Fabien Coelho, François Irigoin, and Ronan Keryell. Compilation et optimisation statique des communications hôte-accélérateur. *Technique et Science Informatiques (TSI)*, 31(8-9-10):1205–1232, December 2012. Numéro spécial RenPar'20.

[6] Corinne Ancourt, Fabien Coelho, and François Irigoin. A Modular Static Analysis Approach to Affine Loop Invariants Detection. In *NSAD: Numerical and Symbolic Abstract Domains*, volume 267 of *ENTCS*, pages 3–16, Perpignan, France, September 2010. Elsivier. 2nd International Workshop.

[7] Corinne Ancourt, Fabien Coelho, François Irigoin, and Ronan Keryell. A Linear Algebra Framework for Static HPF Code Distribution. In *Workshop on Compilers for Parallel Computers, Delft*, December 1993.

[8] Corinne Ancourt, Fabien Coelho, François Irigoin, and Ronan Keryell. A linear algebra framework for static HPF code distribution. *Scientific Programming*, 6(1):3–27, Spring 1997.

[9] Michel Bilodeau, Christophe Clienti, Fabien Coelho, Serge Guelton, François Irigoin, Ronan Keryell, and Fabrice Lemonnier. Présentation du projet FREIA. Congrès ANR à Lyon, January 2012.

[10] Youcef Bouchebaba. *Optimisation des transferts de données pour le traitement du signal : pavage, fusion et réallocation des tableaux.* PhD thesis, École des mines de Paris, November 2002. Thèse co-encadrée par Fabien Coelho et François Irigoin.

[11] Youcef Bouchebaba. Réutilisation de la mémoire pour le pavage. In *14ème rencontres francophones du parallélisme (RENPAR)*, Hammamet, Tunisie, April 2002.

[12] Youcef Bouchebaba and Fabien Coelho. Pavage pour une séquence de nids de boucles. *Technique et Science Informatiques (TSI)*, 21(5):579–603, 2002.

[13] Youcef Bouchebaba and Fabien Coelho. Tiling and memory reuse for sequences of nested loops. In Burkhard Monien and Rainer Feldmann, editors, *8th Int. Euro-Par Conference*, volume 2400 of *Lecture Notes in Computer Science*, pages 255–264, Paderborn, Germany, August 2002. Springer.

[14] Youcef Bouchebaba, Bruno Girodias, Fabien Coelho, Gabriela Nicolescu, and Aboulhamid El Mostapha. Buffer and register allocation for memory space optimization. *The Journal of VLSI Signal Processing*, pages 128–138, May 2007. Springer Netherlands.

[15] Youcef Bouchebaba, Gabriela Nicolescu, El Mostapha Aboulhamid, and Fabien Coelho. Buffer and register allocation for memory space optimization. In *Application-Specific Systems, Architecture and Processors (ASAP)*, pages 283–290. IEEE Computer Society, September 2006. 11-13 September 2006, Steamboat Springs, Colorado, USA.

[16] Fabien Coelho. Étude de la compilation du *High Performance Fortran*. Rapport de stage ingénieur, École des mines de Paris CRI/E/170, CRI, École des mines de Paris, September 1992.

[17] Fabien Coelho. Présentation du *High Performance Fortran*. In *Journée Paradigme sur HPF*. École des mines de Paris, April 1993.

[18] Fabien Coelho. Étude de la compilation du *High Performance Fortran*. Master's thesis, Université Paris VI, September 1993. Rapport de DEA Systèmes Informatiques. Technical report École des mines de Paris CRI/E/178.

[19] Fabien Coelho. Étude et réalisation d'un compilateur pour le *High Performance Fortran*. Rapport de stage d'option École des mines de Paris CRI/A/238, CRI, École des mines de Paris, June 1993.

[20] Fabien Coelho. Experiments with HPF Compilation for a Network of Workstations. In *High-Performance Computing and Networking, Springer-Verlag LNCS 797*, pages 423–428, April 1994.

[21] Fabien Coelho. Présentation du langage HPF. Invitation séminaire de l'IDRIS, June 1994.

[22] Fabien Coelho. Compilation of I/O Communications for HPF. In *5th Symposium on the Frontiers of Massively Parallel Computation*, pages 102–109, February 1995.

[23] Fabien Coelho. Présentation du *High Performance Fortran*. Tutoriel HPF – Transparents, Renpar'7, Mons, Belgique, May 1995.

[24] Fabien Coelho. Compilation des communications HPF. Invitation séminaire Rumeur, June 1996.

[25] Fabien Coelho. *Contributions to High Performance Fortran Compilation*. PhD thesis, École des mines de Paris, October 1996.

[26] Fabien Coelho. Discussing HPF design issues. In *Euro-Par'96, Lyon, France*, pages I.571–I.578, August 1996. LNCS 1123.

[27] Fabien Coelho. Init-time Shadow Width Computation through Compile-time Conventions. TR A 285, CRI, École des mines de Paris, March 1996.

[28] Fabien Coelho. Présentation de HPF. Invitation journée du CEA, February 1996.

[29] Fabien Coelho. Présentation de HPF. Invitation Ateliers d'automne PRISM, December 1996.

[30] Fabien Coelho. Présentation de HPF, problèmes et perspectives. Invitation Séminaire A3 de l'INRIA à Rocquencourt, November 1996.

[31] Fabien Coelho. Compiling Dynamic Mappings with Array Copies. In *Symposium on Principles and Practice of Parallel Programming (PPoPP)*, volume 32-7 of *ACM Sigplan*, pages 168–179, July 1997.

[32] Fabien Coelho. Macro module for apache. `http://httpd.apache.org/`, December 1998. For Apache HTTP server 1.3, 2.0, 2.2 and 2.4. Latest version is 1.2.1 in February 2012. Integrated since version 2.4.6.

[33] Fabien Coelho. HPF Tutorial. Invited to Europar in Toulouse, France. In English, August 1999.

[34] Fabien Coelho. Optimizing Expression Evaluation for ILP, April 1999. Dagstuhl Seminar 99161, *Instruction-Level Parallelism and Parallelizing Compilation*, organized by D. Arvind (Edinburgh), K. Ebcioglu (IBM Yorktown Heights), Ch. Lengauer (Passau), K. Pingali (Ithaca), R. Schreiber (HP, Palo Alto).

[35] Fabien Coelho. Tutoriels Perl, bases et avancé. Conférence Autour du libre, invité par Télécom Bretagne, February 2000.

[36] Fabien Coelho. `pg_comparator`: network and time efficient PostgreSQL, MySQL and SQLite table content comparison. `http://pgfoundry.org/projects/pg-comparator/`, August 2004. Open source software, version 2.2.0 in March 2013.

[37] Fabien Coelho. PGXS: Extension framework for PostgreSQL, July 2004. Included in PostgreSQL 8.0.

[38] Fabien Coelho. `pg-advisor` – proof of concept relational schema analyser. Mail to PostgreSQL development list, March 2004.

[39] Fabien Coelho. `tuplock`: Lock Tuple PostgreSQL Extension. `http://people.apache.org/~fabien/tuplock/`, 2004. Open source software, version 1.2.1 in December 2012.

[40] Fabien Coelho. Exponential memory-bound functions for proof of work protocols. Technical Report CRI/A/370, École des mines de Paris, September 2005. Version 3 in December 2006. Also IACR report 2005/356.

[41] Fabien Coelho. RCS to SVN repository conversion. `http://www.coelho.net/rcs2svn.html`, March 2005.

[42] Fabien Coelho. Correction automatique des TP avec Corrector. Technical Report CRI/E/289, École des mines de Paris, October 2006. Poster présenté à TICE'2006.

[43] Fabien Coelho. Corrector, a web interface for practice session with immediate feedback. Technical Report CRI/A/377, CRI, École des mines de Paris, April 2006.

[44] Fabien Coelho. Corrector, a web interface for practice sessions. In *TICE – Technologies de l'Information et de la Communication dans l'Enseignement supérieur et l'entreprise*, Toulouse, October 2006.

[45] Fabien Coelho. Infrastructure pour newgen, linear/c3 et pips. `http://svn.cri.ensmp.fr/svn/pips/trunk`, January 2006.

[46] Fabien Coelho. Merge svn repositories into one, in date order. `http://www.coelho.net/svn-merge-repos.html`, September 2006. Open source software developed for merging CRI system repositories.

[47] Fabien Coelho. Remote comparison of database tables. Technical Report CRI/A/375, CRI, École des mines de Paris, France, February 2006.

[48] Fabien Coelho. Spam ! spam ! spam ! Séminaire DSI de Réseau ferré de France (RFF), June 2006.

[49] Fabien Coelho. Spam ! spam ! spam ! Diffusion interne aux membres, November 2006. Séminaire de l'association nationale des directeurs de systèmes d'information (ANDSI). Revu et augmenté.

[50] Fabien Coelho. Diffusion de cours sur paristech libres savoirs. `http://www.paristech.fr/`, November 2006-2012. 3 cours diffusés : systèmes d'information, introduction aux réseaux, gestion de configurations logicielles.

[51] Fabien Coelho. Cryptographie et 'pataphysique. Séminaire DSI de Réseau ferré de France (RFF), June 2007.

[52] Fabien Coelho. Quelques idées sur la sécurité des réseaux. Conférence de vulgarisation. DSI Réseau ferré de France (RFF), June 2007.

[53] Fabien Coelho. Spam ! spam ! spam ! Séminaire « le goût du savoir », Fontainebleau, February 2007.

[54] Fabien Coelho. SVN for the System Administrator: Sharing and Deploying Configurations. Technical report CRI/E/295, École des mines de Paris, December 2007.

[55] Fabien Coelho. An (Almost) Constant-Effort Solution-Verification Proof-of-Work Protocol based on Merkle Trees. In Serge Vaudenay, editor, *Africa Crypt 2008*, number 5023 in LNCS, pages 80–93. Springer Verlag, June 2008. Cryptology eprint Archive 2007/433.

[56] Fabien Coelho. La cryptographie est-elle une sous-discipline de la 'pataphysique ? Séminaire « le goût du savoir », Fontainebleau, June 2008.

[57] Fabien Coelho. La cryptographie est-elle une sous-discipline de la 'pataphysique ? *La Lettre numéro 135*, July 2008. Résumé de la conférence du 19 juin, page 11. `http://www.ensmp.fr/Fr/Actualites/Lettre/`. ISSN: 1284-3709.

[58] Fabien Coelho. Comparaison de tables à distance. `http://wiki.postgresql.org/wiki/PGDay.EU%2C_Paris_2009`, November 2009. présentation à PG Day EU 2009, Télécom ParisTech.

[59] Fabien Coelho. `svn-pre-commit`: Configuration Pre-commit Hook for SVN. `http://www.coelho.net/svn-pre-commit.html`, December 2009.

[60] Fabien Coelho. Le *Cloud*. Conférence FedISA (Fédération de l'ILM, du stockage et de l'archivage), June 2010.

[61] Fabien Coelho. Spam ! spam ! spam ! Séminaire DSI de l'Institut national de la santé et de la recherche médicale (INSERM), June 2010.

[62] Fabien Coelho. Remote Comparison of Database Tables. In IARIA, editor, *3rd Int. Conf. on Advances in Databases, Knowledge, and Data Applications*, DBKDA, pages 23–28, St Marteen, The Netherlands Antilles, January 2011.

[63] Fabien Coelho, Alexandre Aillos, Samuel Pilot, and Shamil Valeev. A Field Analysis of Relational Database Schemas in Open-source Software. In IARIA, editor, *3rd Int. Conf. on Advances in Databases, Knowledge, and Data Applications*, DBKDA, pages 9–15, St Marteen, The Netherlands Antilles, January 2011. One of five *best paper* awards.

[64] Fabien Coelho, Alexandre Aillos, Samuel Pilot, and Shamil Valeev. On the Quality of Relational Database Schemas in Open-Source Software. *International Journal on Advances in Software*, 2011-4(3&4):378–388, May 2012.

[65] Fabien Coelho and Corinne Ancourt. Optimal Compilation of HPF Remappings. *Journal of Parallel and Distributed Computing*, 38(2):229–236, November 1996.

[66] Fabien Coelho and Corinne Ancourt. Optimal Compilation of HPF Remappings. Talk at PRS day at LIP, in Lyon, France, January 1996.

[67] Fabien Coelho, Cécile Germain, and Jean-Louis Pazat. State of the Art in Compiling HPF. In Guy-René Perrin and Alain Darte, editors, *The Data Parallel Programming Model*, volume 1132 of *LNCS*, pages 104–133. Springer Verlag, first edition, September 1996. Proceedings of the PRS Spring School, Les Ménuires, March 1996.

[68] Fabien Coelho and Marcel Gosselin. Traduction des messages de subversion en français. Contribution à Subversion, avec d'autres traducteurs, 2005-.

[69] Fabien Coelho and François Irigoin. Compiling for a heterogeneous vector image processor. `http://pips4u.org/doc/meetings/2010-pips-developer-day-25-october-2010`, October 2010. PIPS Developer Day 2010, MINES ParisTech; *Troisièmes Journées Nationales du GDR Génie de la Programmation et du Logiciel* (GDR GPL 2011) June 2011, Lille, France.

[70] Fabien Coelho and François Irigoin. Compiling for a heterogeneous vector image processor. In *Workhop on Optimizations for DSP and Embedded Systems (ODES'9)*, Chamonix, France, April 2011.

[71] Fabien Coelho and François Irigoin. Compilation for Image Hardware Accelerators. Séminaire du LIP6, February 2012. Also updated for Thales (Palaiseau, France) seminar, June 2012.

[72] Fabien Coelho and François Irigoin. API Compilation for Image Hardware Accelerators. *ACM Trans. on Architecture and Code Optimization*, 9(4):49:1–49:25, January 2013. Article 49.

[73] Fabien Coelho and François Irigoin. API Compilation for Image Hardware Accelerators. In *HIPEAC – 8th Int. Conf. on High Performance and Embedded Architectures and Compilers*, page 25 pages, Berlin, January 2013.

[74] Fabien Coelho, Pierre Jouvelot, Corinne Ancourt, and François Irigoin. Data and Process Abstraction in PIPS Internal Representation. In *Workshop on Internal Representations (WIR)*, Chamonix, France, April 2011.

[75] Fabien Coelho, Pierre Jouvelot, François Irigoin, and Corinne Ancourt. Data and Process Abstraction in PIPS Internal Representation. Troisième rencontres de la communauté française de compilation, Dinard, France, April 2011.

[76] Fabien Coelho and Henry Zongaro. `ASSUME` directive proposal. TR A 287, CRI, École des mines de Paris, April 1996.

[77] Marie-Hélène Delmond, Alain Keravel, Fabien Coelho, and Robert Mahl. Les modèles d'affaire, coproduction de valeur et systèmes d'information. In *Les Essentiels du Programme International de Recherche ISD*, Les Essentiels, pages 33–40. Fondation CIGREF, September 2012. Pre-published as `http://www.fondation-cigref.org/modeles-daffaires-coproduction-de-valeur-et-systemes-dinformation/`, April 2012.

[78] François Didry, Fabien Coelho, and Corinne Ancourt. Interface graphique java pour PIPS. `http://svn.cri.ensmp.fr/svn/pips/src/Scripts/jpips/`, 1998.

[79] François Irigoin, Mehdi Amini, Corinne Ancourt, Fabien Coelho, Béatrice Creusillet, and Ronan Keryell. Polyèdres et compilation. In *Rencontres francophones du Parallélisme (RenPar'20)*, Saint-Malo, France, May 2011.

[80] François Irigoin, Mehdi Amini, Corinne Ancourt, Fabien Coelho, Béatrice Creusillet, and Ronan Keryell. Polyèdres et compilation. *Technique et Science Informatiques (TSI)*, 31(8-9-10):987–1019, December 2012. Version étendue de l'article Renpar'20.

[81] Frédéric Laura, Fabien Coelho, and Marie-Hélène Delmond. Gestion durable des données : point sur les enjeux et proposition d'une démarche de pilotage de la performance appuyée sur un *balanced scorecard* thématique. In *Colloque de l'Association Information & Management*, number 15 in Colloque AIM. AIM, May 2010.

[82] Thi Viet Nga Nguyen, François Irigoin, Corinne Ancourt, and Fabien Coelho. Automatic detection of uninitialized variables. In Görel Hedin, editor, *12th Int. Conf. on Compiler Construction (CC)*, volume 2622 of *Lecture Notes in Computer Science*, pages 217–231, Warsaw, Poland, April 2003. Springer. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003.

[83] Thierry J.-F. Omnès. *Acropolis - un précompilateur de spécification pour l'exploration du transfert et du stockage des données en conception de systèmes embarqués à Haut Débit.* PhD thesis, École des mines de Paris, May 2001. (Fabien Coelho examinateur).

[84] Thierry J.-F. Omnès, Youcef Bouchebaba, Chidamber Kulkarni, and Fabien Coelho. *Low-Power Electronics Design*, chapter 37 – Recent Advances in Low-Power Design and Functional Coverification Automation from the Earliest System-Level Design Stages. Computer Engineering. CRC Press, November 2004. Christian Piguet, Editor.

[85] Etienne Servais. Hashcash parallelization on GPGPU using OpenCL. In *49th Int. Scientific Student Conference (ISSC)*, Novosibirsk, Russia, April 2011. Novosibirsk State University. *Best presentation award.* Largest hashcash stamp found – 48 then 52 bits. Travail d'option encadré par Fabien Coelho.

[86] Pham Dinh Son and Fabien Coelho. Typage et élimination des sous-expressions communes (contribution à l'optimisation de programmes scientifiques). Technical Report CRI/E/238, CRI, École des mines de Paris, September 2000.

[87] Julien Zory. *Contributions à l'optimisation de programmes scientifiques*. PhD thesis, École des mines de Paris, December 1999. Thèse co-encadrée par Fabien Coelho et François Irigoin.

[88] Julien Zory and Fabien Coelho. Using algebraic transformations to optimize expression evaluation in scientific codes. In *Proceeding of International Conference on Parallel Architectures and Compiler Techniques (IEEE PACT)*, pages 376–384, October 1998.