Sujet de thèse

Parallélisation et optimisation automatiques pour les systèmes many-cœurs

Cédric Bastoul

April 2, 2012

Abstract

Les limites technologiques de l'industrie des semiconducteurs ont imposé le passage vers les architectures multi-cœurs qui disposent de nombreuses ressources de calcul, mais qui imposent la réécriture des applications pour en bénéficier. Ces architectures offrent souvent plusieurs niveaux de parallélisme (par exemple, vecteurs/intrinsics, threads/OpenMP et tâches/MPI), des hiérarchies mémoires profondes, dans un contexte hétérogène (accès à des accélérateurs matériels tels les GPGPUs). Sur de telles architectures, la parallélisation et l'optimisation des programmes de calcul intensif peuvent améliorer les performances de plusieurs ordres de grandeur. Cependant, la programmation efficace de ces systèmes est encore réservée à des experts maîtrisant les techniques d'optimisation des codes et familiers avec plusieurs langages et architectures. Les outils de parallélisation et d'optimisation automatiques actuels ne visent typiquement qu'un seul niveau de parallélisme et se révèlent médiocres lorsqu'ils ne peuvent analyser statiquement les programmes avec précision. Le but de cette thèse est d'apporter des solutions novatrices pour résoudre ces défauts. Il s'agira de se baser sur les propriétés d'analyse et de composition des outils actuels pour proposer, étudier et développer un système de parallélisation et d'optimisation hiérarchiques capable de résoudre le défaut d'analyse statique durant l'exécution même du programme.

1 Thématique

Ce projet a pour objectif d'apporter des réponses théoriques et des solutions techniques au développement d'applications parallèles performantes sur les systèmes de calcul intensif à architecture multi et many-cœurs par des non-experts, sans sacrifice de productivité.

2 Encadrement

Cédric Bastoul (75%), Maître de Conférences à l'université Paris-Sud 11, laboratoire LRI UMR8623

Université Paris-Sud 11	Tel. :	01 72 92 59 65
LRI CNRS UMR 8623	Fax :	01 60 19 66 08
Campus Scientifique d'Orsay	Email :	cedric.bastoul@u-psud.fr
91405 Orsay	Web :	http://www.lri.fr/~bastoul

Christine Eisenbeis (25%), Directrice de Recherche à l'INRIA Saclay Île-de-France

INRIA Saclay Île-de-France	Tel. :	01 72 92 59 39
Parc Club Orsay Université	Fax :	01 60 19 66 08
91893 Orsay Cedex	Email :	christine.eisenbeis@inria.fr

3 Description des travaux

Le temps où la technologie permettait l'amélioration des performances des programmes mécaniquement, au travers de l'augmentation en fréquence des processeurs, est révolu depuis plusieurs années. La technologie actuelle des transistors atteint ses limites physiques, rendant plus difficile la miniaturisation d'un transistor sans fuite de courant électrique et par conséquent sans instabilité du circuit. Les conséquences de ces difficultés sont un ralentissement de l'augmentation en fréquence et en densité des transistors sur les processeurs. La réponse des architectes à cette crise technologique a été de placer plusieurs *cœurs* (des processeurs partageant certaines ressources telles qu'un niveau de mémoire cache ou un bus mémoire) sur une même puce. On parle d'architectures *multi-cœurs* dans le cas d'au plus quelques dizaines de cœurs (cas des processeurs généralistes actuels) et de *many-cœurs* quand ce nombre atteint plusieurs centaines (cas des accélérateurs matériels tels que les GPUs modernes). Cette solution permet à la fois de ne pas nécessiter l'augmentation de la fréquence d'horloge et de mettre plus de transistors sur une même puce. Mais le prix à payer pour tirer parti de ces architectures est la réécriture des applications. Et ce prix est aujourd'hui exorbitant.

Fondamentalement, les architectures multi ou many-cœurs n'apportent pas de problèmes inconnus en terme de parallélisation des programmes. Cependant, leur avènement pour le grand public (jusque dans les systèmes embarqués) change radicalement la portée de ces problèmes. Alors que la parallélisation était une affaire d'experts, tout programmeur développant une nouvelle application devrait aujourd'hui tenir compte de cette nouvelle dimension. Alors que les applications de calcul intensif, souvent très régulières, étaient les cibles privilégiées de la parallélisation, tous les logiciels sont aujourd'hui concernés. Alors qu'une machine parallèle était souvent dédiée à l'exécution d'un programme jusqu'à sa terminaison, des dizaines de programmes parallèles doivent aujourd'hui cohabiter sur un même système.

Malgré ces besoins, plusieurs années après l'arrivée des architectures multi-cœurs nous devons constater que la mutation des programmes est pour le moins timide. Si les systèmes semblent effectivement plus réactifs, c'est qu'au lieu de donner simplement l'illusion à l'utilisateur d'un fonctionnement multitâche (les différentes applications utilisant tour à tour le processeur), les programmes peuvent réellement s'exécuter en parallèle les uns par rapport aux autres. Mais en dehors de certains navigateurs web qui ont opéré leur mutation (par exemple Google Chrome qui utilise un processus par onglet), la plupart des programmes sont restés eux-mêmes séquentiels. La raison tient à la difficulté de la programmation parallèle qui est restée hors de portée du non-expert. Malgré les nombreuses tentatives, aucune proposition de nouveau langage ou modèle de programmation n'est encore parvenue à s'imposer en composant productivité et performance. Si des solutions satisfaisantes ne sont pas apportées, il est probable que l'augmentation du nombre de cœurs sur les puces ne dure pas et qu'en l'absence d'alternative aux limites de la technologie actuelle des transistors, nous observions une stagnation des performances sur les stations de travail personnelles. Nous proposons dans ce projet de travailler à des solutions d'aide à la parallélisation et à l'optimisation de programmes à la compilation pour permettre au non-expert de tirer parti des architectures many-cœurs actuelles et à venir tout en conservant un haut niveau de productivité.

3.1 Concepts, outils, défis et enjeux, pistes bibliographiques

Les architectures haute performance, tout comme les ordinateurs grand public modernes et certains systèmes embarqués, possèdent plusieurs niveaux de parallélisme. Un cas d'école de nos jours se compose d'au moins trois niveaux. Le premier correspond aux unités vectorielles (ou SIMD), capables de réaliser une même instruction sur un petit ensemble de données différentes. Le second correspond aux différents cœurs disponibles sur un même processeur, capables d'exécuter plusieurs threads ou processus en parallèle avec la propriété de partager une part de la hiérarchie mémoire. Le troisième correspond aux architectures multi-processeurs capables de réaliser des tâches en parallèle en utilisant une mémoire distribuée.

Du point de vue de l'algorithmique parallèle, ces niveaux de parallélisme sont équivalents (il s'agit du même modèle de calcul). Pourtant, il y a de grandes différences du point de vue de la compilation. La première tient à la programmation. Les unités vectorielles s'exploitent typiquement *via* l'utilisation d'instructions dédiées appelées *intrinsics*. Au niveau des cœurs on préfère classiquement OpenMP. Enfin on utilise généralement MPI pour le niveau supérieur. La seconde différence tient aux propriétés des performances. Les optimisations du premier niveau sont extrêmement fragiles: une différence en apparence mineure dans le code source peut avoir un impact très important sur les performances [1]. Le second niveau est plus robuste mais les interactions complexes entre les différents éléments matériels et le travail du compilateur rendent encore le comportement en performance difficilement prédictible. Le troisième niveau est quant à lui relativement plus stable, cependant les compromis à décider entre parallélisme et communications sont encore des sujets de recherche particulièrement ouverts pour les systèmes automatiques. Les défis sont donc nombreux et dérivent d'une part de la difficulté de modéliser les interactions entre les différentes optimisations et d'autre part de la nécessité de gérer efficacement l'ordonnancement et les communications au niveau parallélisme de tâche.

En partant d'un programme source naïf, les compilateurs actuels ont de grandes difficultés à appliquer les transformations nécessaires pour atteindre de hauts niveaux de performance. Le *modèle polyédrique* est une approche de recherche efficace, très adaptée à certaines classes de programmes à base de boucles et d'accès à des tableaux. Dans ces programmes, la sémantique peut être capturée par des fonctions affines et des systèmes de contraintes affines. Cependant les techniques les plus récentes d'optimisation polyédriques ne visent aujourd'hui de manière directe qu'un seul niveau de parallélisme. Le plus souvent il s'agit du niveau intermédiaire (exploitation des différents cœurs) [1, 2]. Nous nous proposons, dans le cadre de ce projet, de définir des techniques d'optimisation hiérarchiques tenant compte des spécificités de chaque niveau. Nous avons défini une approche de transformation dans le modèle polyhédrique qui permet la composition sans limite de transformations et qui par conséquent rend cette approche possible [3]. Nous avons de plus eu l'occasion d'étudier ces dernières années différentes stratégies d'optimisation (*itérative* [1], par *machine learning* [4] et par *modèle* [5, 6]). Chacune d'entre elles répond aux particularités d'un des niveaux de parallélisme. Le dernier niveau, qui supportera des communications explicites, devrait s'inspirer de techniques d'ordonnancement basées sur le graphe de dépendances et non sur des *barrières* [7]. De même, les résultats connus de génération de communications vers les accélérateurs matériels tels que les GPUs devront servir de base à ces travaux [8]. La stratégie définie pour chaque niveau de parallélisme dépendra ainsi des propriétés intrinsèques au niveau correspondant et à son modèle de programmation, pour fournir un système automatique d'optimisation complet ayant pour but ultime d'offrir de meilleures performances que les bibliothèques optimisées existantes.

3.2 Positionnement par rapport à l'état de l'art et à la concurrence nationale et internationale

En dehors des outils de profilage permettant de découvrir les parties critiques d'un programme ou d'observer son comportement, l'arsenal du programmeur désirant optimiser ses programmes sur les architectures modernes se décompose en trois principales familles d'« outils » : les langages parallèles, les bibliothèques et les compilateurs. Elles correspondent à autant d'approches complémentaires possédant leurs propres forces et limitations :

- Les langages parallèles peuvent mettre à disposition des abstractions de haut niveau (objets mathématiques, arbres, graphes etc.), des concepts propres à la programmation parallèle (boucles parallèles, réductions, tâches etc.) voire des moyens de contrôler finement l'exécution du programme (distribution des données, alignement etc.). De très nombreuses propositions ont été faites (parmis lesquelles Cilk, Chapel, C^n, CUDA, Fortress, HPF, UPX, X10 etc.), mais aucune ne s'est réellement imposée en face des trois modèles de programmation principalement utilisés aujourd'hui,

et pourtant reconnus comme insuffisants, que sont les threads, OpenMP et MPI. Les raisons sont doubles. D'une part il est préoccupant de développer une application dont la viabilité du support à long terme n'est pas encore certaine, et bien sûr les habitudes de programmation ont la vie dure. Mais d'autre part ces langages laissent encore une large part du travail au programmeur. En plus d'avoir à apprendre un nouveau langage, celui-ci doit en effet découvrir et exprimer lui-même le parallélisme de son application, parfois jusque dans les détails de la disposition des données.

- Les bibliothèques optimisées (telles LAPACK, MKL, IPP etc.) mettent à disposition du programmeur des fonctions très performantes prêtes à l'emploi (jeu de routines BLAS, FFT etc.). Certaines bibliothèques avancées sont capables de s'auto-optimiser sur certaines variations d'une architecture cible (ATLAS, FFTW, SPIRAL...). Si elles sont souvent très utiles localement, le spectre des fonctions disponibles et des architectures cibles est par essence limité. De plus il n'est pas possible de bénéficier d'optimisations globales. Si par exemple deux multiplications matricielles mettent en jeu la même matrice, deux appels successifs seront nécessaires alors qu'une optimisation d'ensemble aurait probablement été bien plus efficace.

- La dernière famille d'outils incontournables est celle des compilateurs (tels GCC, XL ou ICC). Ceux-ci peuvent mettre à disposition des extensions des langages (via des pragmas, tel OpenMP) ou offrir des capacités d'optimisation et de parallélisation automatiques (vectorisation, transformation de boucles). Les compilateurs optimisant peuvent offrir la meilleure productivité car ils ne nécessitent pas l'apprentissage d'un nouveau langage, avec la meilleure portabilité puisqu'ils se chargent d'optimiser pour une architecture donnée. Cependant si les récentes avancées (parmi lesquelles s'inscrit notre travail) les ont rendu relativement performants dans le cas de boucles de calcul intensif et d'architectures à un niveau de parallélisme, ils ne sont pas encore adaptés aux optimisations multi-niveaux nécessaires aux architectures modernes.

Notre projet se situe donc dans le cadre des outils de compilation, par rapport aux deux autres principales approches. Nous souhaitons repousser l'état de l'art de deux manières. Tout d'abord en visant des programmes non-entièrement réguliers où une partie de l'analyse devra être réalisée au moment de l'exécution. Ensuite, nous souhaitons viser des optimisations globales impliquant de multiples niveaux de parallélisme aux propriétés diverses avec comme cible première les architectures embarquées. Notre équipe est reconnue internationalement pour ses travaux en compilation. Certains de nos outils (en particulier le générateur de code CLooG) sont des standards de fait au niveau mondial. Nos principaux concurrents (mais également parfois collaborateurs) sont tout d'abord le groupe du Pr. Sadayappan à The Ohio State University qui oriente ses efforts vers des transformations plus générales (paramétriques) mais visant toujours essentiellement le multi-cœurs. Ensuite, le groupe du Pr. Hall à The University of Utah ouvre les techniques d'optimisation des compilateurs de haut niveau à l'utilisateur alors que nous souhaitons au contraire l'assister automatiquement. Enfin les entreprises IBM (Watson Research Center, USA) et Reservoir Labs Inc. (New York, USA) développent des compilateurs basés entre autre sur nos techniques et sur leurs propres extensions. Il est difficile de connaître leur état d'avancement mais leur succès est une démonstration de l'attrait des industriels pour nos technologies.

3.3 Résultats attendus et critères proposés pour juger du succès du projet

Ce projet a pour but la réalisation tant théorique que technique d'un système de parallélisation et d'optimisation multi-niveaux pour des codes réguliers et non réguliers. Le premier vecteur de diffusion des résultats sera les publications scientifiques dans les principales conférences et revues du domaines. Notre équipe a de plus une profonde culture de réalisation d'outils de niveau industriel. L'outil créé viendra compléter notre écosystème logiciel dédié à l'optimisation des programmes. Sa distribution sera le second vecteur de diffusion de nos résultats. La qualité de la réalisation sera évaluée par la comparaison

avec les compilateurs de production et les bibliothèques optimisées les plus récentes. L'intérêt scientifique sera mesuré par la qualité des publications associées alors que l'intérêt industriel sera indiqué par la distribution logicielle de l'outil réalisé.

4 Contexte du stage

Le centre de recherche INRIA Saclay – Île-de-France comprend 450 membres dont 390 scientifiques. Il regroupe 26 équipes de recherche réparties sur 8 sites et fonctionne en partenariat étroit avec l'université Paris-Sud, le CNRS (5 UMRs), l'École Polytechnique, l'École Normale Supérieure de Cachan ainsi que le pôle de compétitivité Systemtic et le réseau thématique de recherche avancée Digiteo.

GRAND LARGE est une équipe commune INRIA-LRI (CNRS et Université Paris-Sud) spécialisée dans la recherche et le développement de solutions pour le calcul haute performance sur les systèmes distribués à grande échelle (architectures massivement parallèles et grilles). Elle s'intéresse particulièrement à la conception, au développement, à l'expérimentation et la preuve d'environnements de programmation, de middlewares, d'algorithmes scientifiques, de bibliothèques et de compilateurs pour les applications de calcul haute performance. Dirigée par Bigitte Rozoy, elle compte 9 chercheurs et enseignants-chercheurs, 15 doctorants et post-doctorants, et 2 ingénieurs.

Le sous-groupe ARCHI est spécialisé dans les problématiques de compilation et de manipulation des programmes pour la parallélisation et l'optimisation automatiques. Leurs membres sont fortement impliqués, notamment au travers du réseau d'excellence HiPEAC, dans des collaborations européennes et internationales sur des projets scientifiques, des projets de transfert technologique et des plateformes logicielles libres, incluant le compilateur GCC ou le compilateur de haut niveau PoCC.

Cette thèse se déroulera dans le cadre et le contexte international du projet européen MANY, dédié à la création d'outils de programmation pour les architectures many-cœurs embarquées et rassemblant des partenaires espagnols, français, koréens et suédois du monde industriel comme académique.

References

[1] Louis-Noël Pouchet, Cédric Bastoul, Albert Cohen, and Nicolas Vasilache. Iterative optimization in the polyhedral model: Part I, one-dimensional time. In *ACM International Conference on Code Generation and Optimization (CGO'07)*, pages 144–156. San Jose, California, March 2007.

[2] Uday Bondhugula, Albert Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral program optimization system. In *ACM SIGPLAN Conference on Programming Language Design and Implementation PLDI'08*. Tucson, Arizona, June 2008.

[3] Cédric Bastoul. Code generation in the polyhedral model is easier than you think. In *PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques*, pages 7–16. Juan-les-Pins, France, September 2004.

[4] Louis-Noël Pouchet, Cédric Bastoul, John Cavazos, and Albert Cohen. A note on the performance distribution of affine schedules. In *2nd Workshop on Statistical and Machine learning approaches to ARchitectures and compilaTion (SMART'08)*. Göteborg, Sweden, January 2008.

[5] Cédric Bastoul and Paul Feautrier. Improving data locality by chunking. In *CC'12 International Conference on Compiler Construction, LNCS 2622*, pages 320–335. Warsaw, Poland, April 2003.

[6] Louis-Noël Pouchet, Uday Bondhugula, Cédric Bastoul, Albert Cohen, J. Ramanujam, and P. Sadayappan. Combined iterative and model-driven optimization in an automatic parallelization framework. In *Conference on Supercomputing (SC'10)*. New Orleans, LA, November 2010.

[7] Muthu Baskaran, Uday Bondhugula, Sriram Krishnamoorthy, J. Ramanujam, Atanas Rountev, and P. Sadayappan. A compiler framework for optimization of affine loop nests for GPGPUs. In *ICS'08 Proceedings of the 22nd annual international conference on Supercomputing*, pages 225–234. New York, NY, USA, 2008.

[8] Muthu Baskaran et al. Automatic data movement and computation mapping for multi-level parallel architectures with explicitly managed memories. In *PPoPP'13*, pages 1–10. New York, NY, USA, 2008.