Compilation d'applications de traitement du signal sur accélérateurs matériels à haute efficacité énergétique
Rapport d'avancement 1re année

Pierre Guillou
Directeur de thèse : François Irigoin
Maître de thèse : Fabien Coelho
Centre de recherche en informatique
MINES ParisTech

19 mai 2014

Introduction

Au cours des dernières années, la recherche de la haute performance dans les calculs numériques s'est accompagnée d'un effort global de réduction de la consommation énergétique. Dans l'impossibilité d'accroître plus encore la fréquence d'horloge, les concepteurs de microprocesseurs se sont orienté vers une augmentation du nombre de cœurs de calcul. Ainsi, les processeurs sont plus performant pour une consommation d'énergie moindre, au prix d'une complexité accrue dans leur utilisation.

Suite à mon projet de fin d'études, réalisé au **Centre de recherche en informatique** MINES ParisTech (CRI) à Fontainebleau, j'ai été invité à continuer mes travaux au travers d'une thèse consacrée à la compilation sur architectures à haute efficacité énergétique.

Le présent rapport rend compte des 8 premiers mois de ce doctorat, débuté en octobre 2013. Dans une première partie, je délimiterai le cadre et le sujet de cette thèse. Par la suite, au travers d'un exemple, j'expliciterai quelques problématiques rencontrées durant mes travaux.

Dans une troisième partie, je présenterai les activités annexes auxquelles j'ai participé durant ces premiers mois de thèse, à savoir les conférences, les séminaires. Je détaillerai également le portefeuille de compétences acquises durant ce semestre.

1 Contexte

La puissance de calcul des ordinateurs modernes ne cesse d'augmenter, au détriment de la facilité qu'ont les développeurs d'applications à la maîtriser.

1.1 Enjeux

Depuis près de cinquante ans, l'intégration toujours plus poussée des composants électroniques a provoqué une augmentation exponentielle de la performance des calculs numériques, phénomène connu sous le nom de *loi de Moore*. La puissance de calcul des téléphones portables actuels dépasse ainsi celle des calculateurs de la NASA qui ont envoyé l'homme sur la lune en 1969.

De nombreuses applications tentent de tirer parti de cette puissance de calcul, accessible notamment au travers de superordinateurs de plus en plus perfectionnés. Des simulations permettant de prédire le réchauffement climatique aux analyses de collisions du LHC en passant par les études de résistance des matériaux, la plupart de ces applications font appel à des connaissances pointues dans des domaines scientifiques variés.

D'autre part, l'avènement des processeurs multi-cœurs a entrainé l'apparition d'une multitude d'architectures innovantes, très variées et prometteuses — comme le coprocesseur Intel Xeon Phi qui compte plus de 50 cœurs de calcul, ou bien le *System-on-Chip* Tegra de NVIDIA qui fusionne CPU basse consommation et GPU haute performance dans la même puce.

Cette thèse propose de se placer comme intermédiaire afin de faciliter l'utilisation d'architectures non-conventionnelles par des développeurs d'applications spécifiques.

1.2 Problématiques

L'émergence de ces architectures exotiques offre de nouvelles possibilités en termes de performances et de consommation d'énergie. Ce dernier facteur a pris une place de plus en plus prépondérante à mesure que les *datacenters* et *supercomputers* se multiplient. Ceux-ci nécessitent en effet une énergie de plus en plus grande, à la fois pour alimenter leurs composants, mais aussi pour les refroidir.

À l'autre extrémité du spectre, les téléphones de type *smartphone* et les tablettes électroniques exposent aujourd'hui une puissance de calcul proche des stations de travail classiques, sous contrainte d'autonomie.

Cependant, tirer parti de ces architectures massivement parallèles est complexe. Un lourd travail d'ingénierie est nécessaire afin d'adapter un logiciel existant sur une architecture précise. Les coûts de développement sont très élevés et nécessitent des connaissances pointues concernant l'architecture cible.

Des bibliothèques logicielles facilitant le portage de logiciels vers des architectures massivement parallèles permettent d'exploiter ces architectures. Les plus connues sont *pthread*, *OpenMP* et *MPI*. Néanmoins, l'utilisation de ces bibliothèques tend d'une part à brider les performances des applications du fait de leur généricité, et d'autre part nécessite des compétences en programmation parallèle.

L'utilisation de compilateurs combinés à des langages spécifiques à un domaine dissimulant la complexité des architectures cibles permet d'offrir un environnement de programmation confortable aux développeurs logiciels, ainsi qu'une assurance quant aux performances à l'exécution.

Cette thèse se focalise sur cette dernière approche.

1.3 Sujet de thèse

La thèse que je réalise porte sur la compilation vers des architectures à haute efficacité énergétique. Elle s'intitule :

« La compilation face aux défis énergétiques du XXI^{e} siècle »

L'objectif de cette thèse est de développer un compilateur permettant de tirer parti d'architectures parallèles et peu consommatrices afin d'optimiser les performances et la dépense énergétique. Ce compilateur sera associé à un langage spécialisé à un domaine (DSL) afin de faciliter l'écriture d'applications. On se restreindra au domaine du traitement du signal, puisque les applications de ce domaine présentent généralement un parallélisme implicite, utilisent des opérateurs réguliers, et répondent à un besoin industriel car consommatrices de calculs.

1.4 Encadrement

Je suis basé au Centre de recherche en informatique, MINES ParisTech. Ce centre de recherche se concentre principalement sur la notion de langage, à travers l'analyse, la compilation et la preuve de langages de programmation. Le centre est basé à Fontainebleau, en Seine et Marne.

Mon directeur de thèse est François Irigoin, directeur du Centre. Je suis en outre supervisé par Fabien Coelho, enseignant-chercheur, qui est mon maître de thèse. En outre, j'appartiens à l'école doctorale Sciences et Métiers de l'Ingénieur (SMI) commune à MINES ParisTech et à Arts et Métiers ParisTech.

2 Un exemple : compilation vers une architecture massivement parallèle

Au cours des premiers mois de ma thèse, je me suis concentré sur une architecture massivement parallèle sur laquelle j'ai compilé des applications d'analyse d'images.

L'analyse d'images permet la reconnaissance d'éléments contenus dans une image. Une des branches de l'analyse d'images, la morphologie mathématique, fait l'objet de recherches dans le Centre de morphologie mathématique de MINES ParisTech.

Un exemple d'application d'analyse d'images permettant l'extraction automatique de plaques d'immatriculations est représentée Figure 1.

L'article de Coelho et Irigoin [11] présentait déjà une méthodologie pour porter des applications d'analyse d'image à l'aide d'un compilateur sur une variété d'architectures différentes à partir d'une interface de programmation commune appelée FREIA (FRamework for Embedded Image Applications) ayant fait l'objet d'un projet ANR entre 2008 et 2012. Les architectures cibles consistaient en

— deux accélérateurs graphiques implémentés sur FPGA ;
— deux CPUs et
— quatre GPUs via le langage OpenCL.

(a) Image d'entrée (b) Image de sortie

FIGURE 1 – Extraction de plaques d'immatriculation

L'interface FREIA permet de développer relativement facilement des applications de morphologie mathématique. Elle propose une interface séquentielle spécifique à l'analyse d'images et ne privilégie pas d'architecture cible. On peut donc l'assimiler à un langage spécifique au domaine de la morphologie mathématique.

Par extension de l'approche proposée dans l'article précédemment cité, j'ai réalisé un compilateur à partir de cette interface vers une architecture massivement parallèle composée de 256 cœurs, que j'ai testé sur une dizaine d'applications réelles. Ce travail s'inscrit donc directement dans mon projet de thèse.

Pour effectuer ce travail de compilation, il a fallu étudier au préalable l'architecture cible afin de déterminer la stratégie à mettre en œuvre pour réaliser cette chaîne de compilation.

2.1 Le processeur Kalray MPPA-256

L'architecture à haute efficacité énergétique sur laquelle j'ai travaillé est le processeur manycore de Kalray, appelé MPPA-256, MPPA signifiant « Multi-Purpose Processor Array ». Ce processeur est caractérisé par ses 256 cœurs de calcul, ainsi que sa consommation énergétique moyenne de 5 Watts.

2.1.1 Caractéristiques

En comparaison, les processeurs grand public Intel Haswell de dernière génération possèdent moins de 8 cœurs pour une consommation énergétique de plus de 30 Watts. Les co-processeurs Intel Xeon Phi, dédiés au calcul hautes-performances et présents dans Tianhe-2, l'ordinateur le plus puissant au monde [1], disposent quant à eux de 64 cœurs pour une consommation estimée à plus de 100 Watts.

Le MPPA-256 est plutôt destiné aux systèmes embarqués de part sa faible consommation énergétique, bien que Kalray envisage également d'en faire un coprocesseur destiné au calcul

1. TOP500, novembre 2013

haute-perfomance.

2.1.2 Architecture

Les 256 cœurs de calcul du MPPA-256 sont répartis en 16 clusters. Ces clusters communiquent par un sous-sytème réseau communément appelé *Network on Chip*.

Le processeur comporte également quatre clusters d'entrée/sortie répartis sur les quatres bords de la puce. Ceux-ci, comportant chacun quatre cœurs, gèrent les communications de la puce avec le système hôte, de la mémoire vive ou bien une autre puce via des interfaces DDR3, PCI Express et Ethernet, pour ne citer que les plus connues.

Une représentation de la puce est donnée Figure 2.

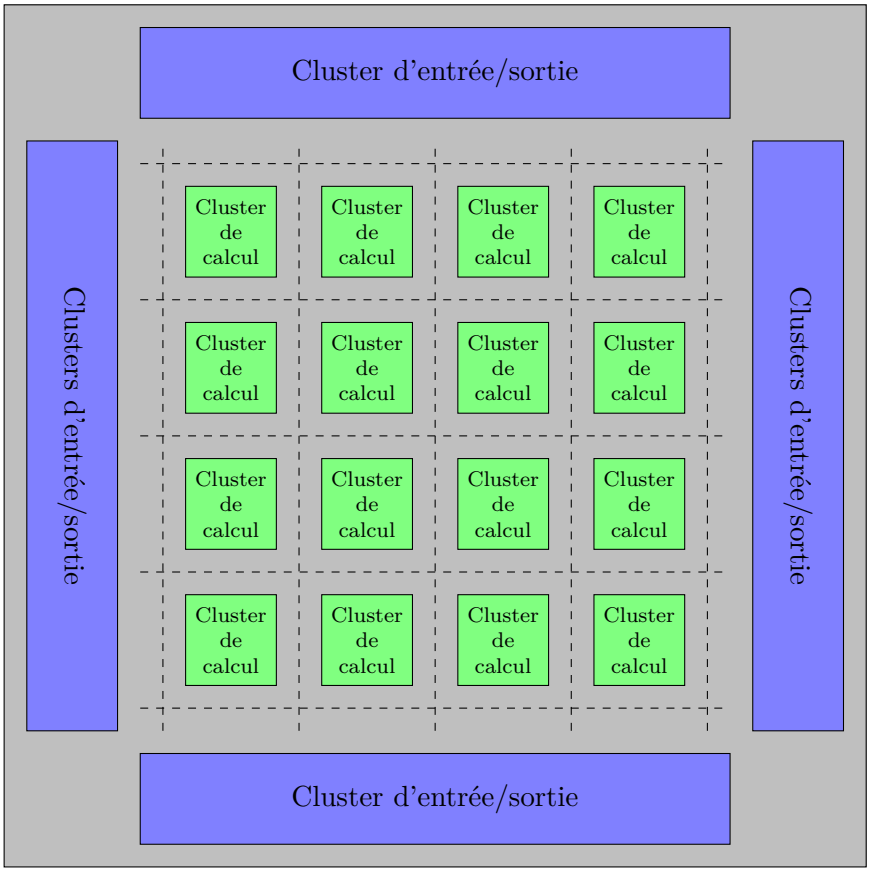


FIGURE 2 – La puce MPPA-256

Les cœurs qui composent le processeur de Kalray ont été conçu de façon à minimiser leur consommation énergétique. Ces cœurs sont basés sur l'architecture *VLIW* (*Very Long Instruction Word*), qui permet d'assurer un parallélisme d'instructions. Ils ne peuvent pas exécuter plus d'un thread simultanément. Ils sont assemblés en clusters.

Chaque cluster de calcul, représenté en Figure 3, est composé de 16 cœurs de calcul, gérés par un 17^{e} cœur. Ce dernier est équipé du système d'exploitation minimaliste NodeOS, qui va s'occuper de lancer le programme sur les autres cœurs. Un cluster dispose de 2 Mégaoctets de mémoire, celle-ci étant partagée entre le système d'exploitation (qui en occupe environ 500 ko)

et les données de calcul.

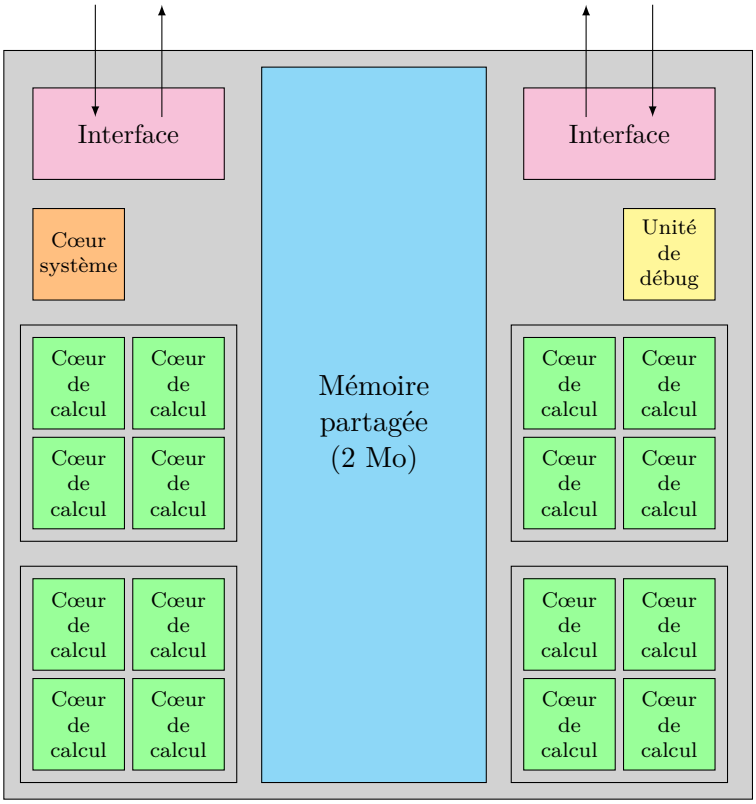


FIGURE 3 – Un cluster de calcul

Cette puce s'utilise principalement en tant qu'accélérateur matériel, en complément d'un processeur standard sur une station de travail. Les calculs et les données y sont déportés afin d'exploiter ses caractéristiques.

Cette architecture est complexe à exploiter : les 2 Mo disponibles par cluster de calcul s'avèrent rapidement limitants. Une deuxième mémoire de type DDR peut être accédée par les clusters d'entrée-sortie, afin de stocker les données, mais la gestion des transferts avec la mémoire de la machine hôte est à charge du programmeur. De plus, pour tirer entièrement parti des performances, il faut utiliser les 256 cœurs de calcul en même temps, et donc utiliser des algorithmes parallélisables.

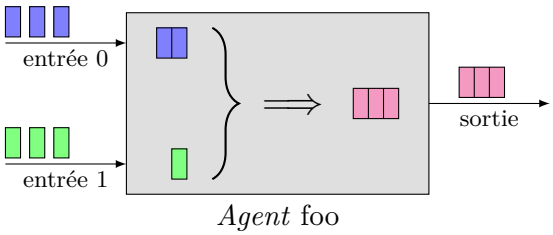
2.1.3 Modèles de programmation

Différents modèles de programmations sont proposés pour développer sur ce processeur. Le MPPA-256 est compatible avec les bibliothèques POSIX-Threads et OpenMP, et propose un support initial d'OpenCL.

Cependant, Kalray a développé en collaboration avec le CEA-LIST un langage de programmation *Streaming Dataflow* dédié aux architectures manycores et appelé *Sigma-C* (aussi orthographié ΣC) [15].

Il s'agit d'une surcouche de C89 basé sur les notions d'*agent*, boîte noire qui consomme et produit des données, et de *subgraph*, graphe orienté d'agents connectés.

Un agent est en général exécuté par un unique cœur du MPPA-256, mais peut être placé sur un cluster d'entrée/sortie ou bien sur l'hôte. En effet, pour transférer des données de l'hôte aux cœurs de calcul, il faut passer par les clusters d'entrée/sortie.



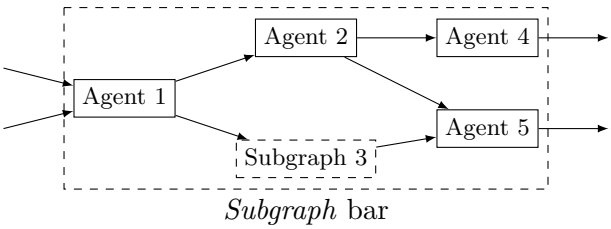
(a) Représentation

```
agent foo() {
  // describe agent interface
  interface {
    in<int> input0;
    in<int> input1;
    out<int> output;
    // declare the state machine
    spec{input0[2], input1, output[3]};
  }
  // loop over the state
  void start()
    exchange (input0 inp0[2],
              input1 inp1,
              output outp[3]) {
    outp[0] = inp0[0];
    outp[1] = inp1;
    outp[2] = inp0[1];
  }
}
```

(b) Code

FIGURE 4 – Agent *Sigma-C*

Un agent *Sigma-C* est représenté en Figure 4. Celui-ci dispose de deux entrées, consomme deux données sur l'une et une donnée sur l'autre. Il produit trois données sur sa sortie.



(a) Représentation

```
subgraph bar() {
  // describe subgraph interface
  interface { ... }
  map {
    // instanciate agents
    agent a1 = new Agent1();
    ...
    agent a3 = new Subgraph3();
    // connect agents
    //to subgraph interfaces
    connect (input0, a1.input0);
    ...
    connect (a5.output, output1);
    // connect agents
    connect (a1.output0, a2.input);
    ...
    connect (a3.output, a5.input1);
  }
}
```

(b) Code

FIGURE 5 – Subgraph *Sigma-C*

Ces agents peuvent se connecter les uns aux autres de façon à former un *subgraph*, comme représenté en Figure 5. On construit ainsi bloc par bloc un circuit de données qui vont s'échanger d'agent en agent.

Ce langage de programmation, de plus haut niveau que les solutions précédemment citées, abstrait la gestion des communications entre les différents cœurs, clusters, et même entre la puce et la machine hôte. C'est pour ces raisons que j'ai employé ce langage dans la suite de mes

travaux.

Néanmoins, ce langage reste limité. D'une part, l'abstraction des communications restreint le contrôle du matériel. De plus, le langage est assez rigide et n'autorise pas l'utilisation de pointeurs : la taille des données échangées doit être déterminée à la compilation, et ne peut dépendre des entrées du programme.

2.2 Génération de code *Sigma-C* pour l'analyse d'images

Grâce à un compilateur source-à-source, j'ai développé un générateur de subgraphs *Sigma-C* à partir d'applications de morphologie mathématique écrites avec l'API FREIA.

2.2.1 Implémentation

Afin de compiler des applications d'analyse d'images sur le MPPA-256, j'ai opéré en deux étapes :

environnement d'exécution j'ai tout d'abord implémenté des opérateurs de base d'analyse d'images en langage *Sigma-C*, afin d'avoir une bibliothèque minimaliste à disposition,

générateur de code j'ai utilisé le compilateur source-à-source PIPS, développé au Centre, pour générer automatiquement des applications *Sigma-C* avec des appels à ma bibliothèque.

La rigidité du langage *Sigma-C* impose des contraintes au niveau de l'environnenment d'exécution. Celui-ci doit par exemple connaître à l'avance la dimension des images traitées. De plus, la faible quantité de mémoire cache par cluster de calcul ne permettent pas de traiter l'image entière : les calculs sont réalisés ligne à ligne, et l'image est stockée dans la mémoire vive accessible par l'interface DDR des clusters d'entrée/sortie.

Le langage *Sigma-C* a aussi posé des difficultés pour la génération de code : les nombres de canaux d'entrée et de sortie des agents étant fixes, il faut le cas échéant insérer des agents qui vont répliquer les données entre des agents de calcul.

Le principe de cette génération de code est résumé Figure 6 :

1. le code applicatif FREIA est analysé et optimisé par PIPS et découpé en chaînes de calcul,
2. PIPS génère du code *Sigma-C* prenant appui sur ma bibliothèque *Sigma-C* d'analyse d'images, ainsi que du code FREIA pour l'environnement d'exécution,
3. ces applications sont ensuite compilées.

Le compilateur PIPS effectue deux séries de passes d'optimisation :

1. des optimisations génériques, qui visent à éliminer les opérations inutiles, ou à fusionner des chaînes de calcul redondantes,
2. des optimisations spécifiques au matériel cible.

FREIA dispose d'une implémentation logicielle appelée Fulguro. Les applications générées prennent appui sur cette implémentation côté hôte pour le chargement et la sauvegarde des images sur disque. Les communications avec l'implémentation *Sigma-C* sont réalisées par des *tubes nommés*.

L'exécution des applications générée suit plusieurs étapes, comme représenté en Figure 7.

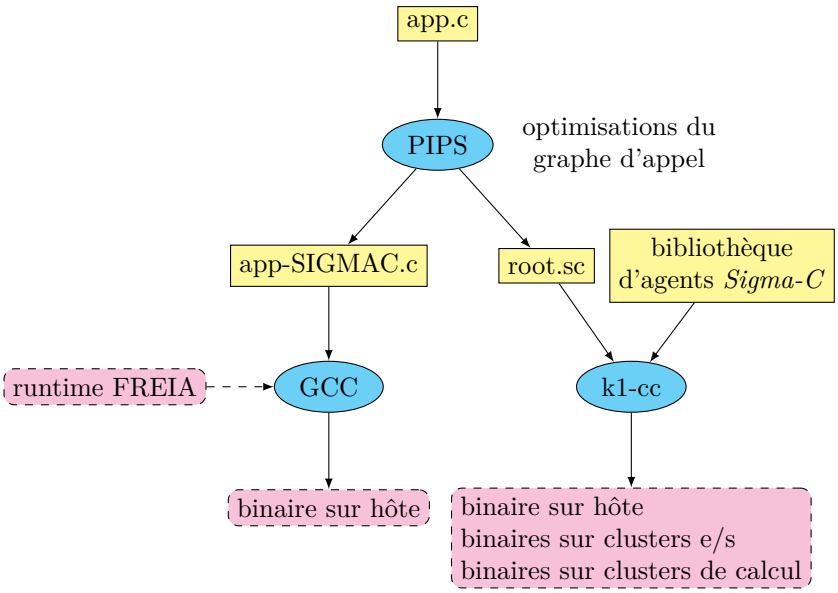


FIGURE 6 – Génération de code *Sigma-C* avec PIPS

1. les images d'entrée sont chargées depuis le disque dur de la machine hôte par Fulguro,
2. à travers un subgraph *Sigma-C* dédié et un tube nommé, on réserve la mémoire pour les images sur la carte d'extension de la puce,
3. les images sont transférées à la place réservée,
4. on lance les chaînes de calcul sur les images d'entrée. Les images sont traitées ligne par ligne par les clusters de calcul. Le résultat est stocké directement sur la mémoire de l'accélérateur.
5. Une fois les traitements achevés, les images résultantes sont transmises depuis la mémoire vive de l'accélérateur vers l'hôte, afin d'être sauvegardées sur le disque dur,
6. enfin, on libère la mémoire précédemment allouée.

Je tiens à souligner le soutien qu'a fourni Kalray pour ce projet. L'entreprise a en effet fourni d'une part une version optimisée en langage assembleur des opérateurs morphologiques, qui en effet s'avéraient limitant dans la chaîne de traitement. D'autre part, Kalray a également conçu l'environnement d'exécution *Sigma-C* visant à transférer les images entre l'hôte et l'accélérateur.

J'ai testé ma chaîne de compilation sur une dizaine de cas réels. Chaque cas ayant ses spécificités, cela a demandé beaucoup de temps afin de les faire fonctionner correctement.

2.2.2 Comparaison avec les autres cibles matérielles

Une fois les applications correctement portées vers le MPPA-256, j'ai comparé les temps d'exécution et la consommation d'énergie par application aux autres cibles matérielles décrites dans l'article de Coelho et Irigoin [11].

Kalray fournit un utilitaire logiciel mesurant la consommation énergétique d'un binaire exécuté par le MPPA-256. Grâce à celui-ci, j'ai pu obtenir la consommation énergétique moyenne des applications testées. Connaissant les temps d'exécution de ces applications sur les autres cibles, ainsi que leur puissance consommée, on en déduit la consommation énergétique.

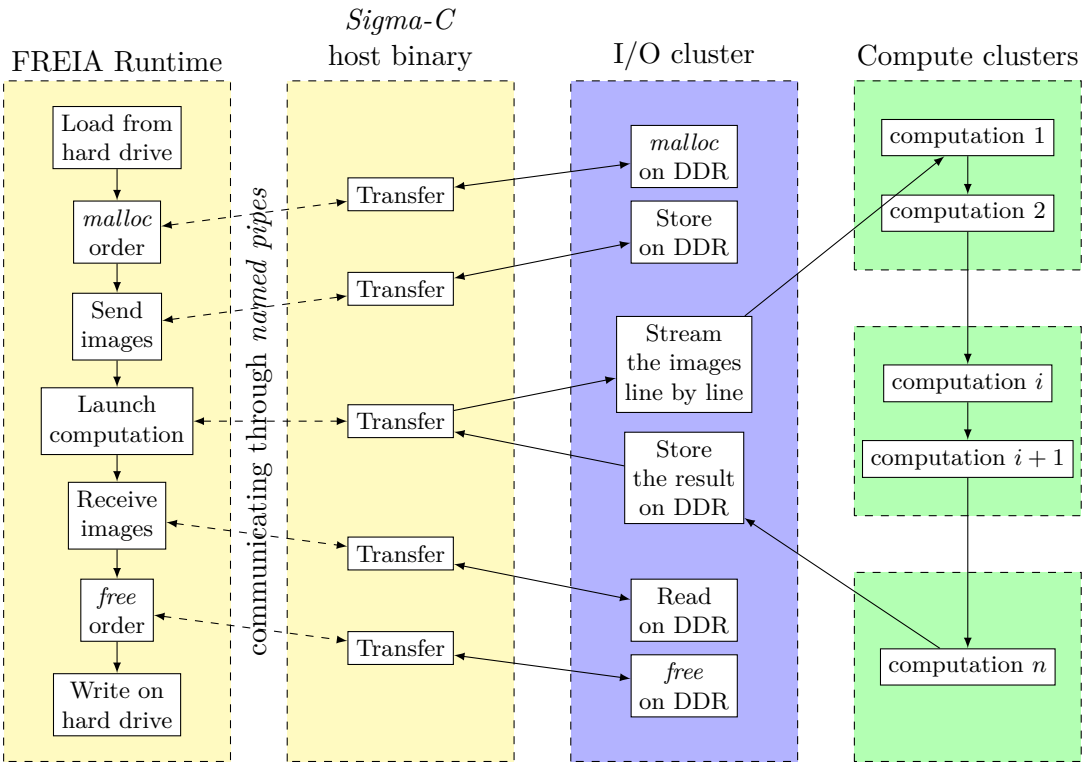


FIGURE 7 – Environnement d'exécution

J'ai représenté en Figure 8 les temps d'exécution des différentes cibles matérielles sur les différentes applications testées, relativement au MPPA-256, et en échelle logarithmique.

La Figure 9 représente les consommations énergétiques des différentes cibles matérielles, sur les mêmes applications.

Notre chaîne de compilation montre des temps d'exécutions honorables et des consommations énergétiques intéressants dans la moitié des applications (*anr999*, *burner*, *licensePlate*, *oop*, *toggle*). Les relativement pauvres performances du MPPA-256 pour les applications *antibio*, *deblocking* et *retina* sont causées par des boucles `while` non déroulées faisant intervenir des opérations de réduction dont les résultats sont communiqués à l'hôte.

De plus, et contrairement aux autres cibles, notre chaîne de compilation ne réalise pas encore d'optimisations spécifiques au MPPA-256. Les chaînes de calcul sont projetées telles quelles sur la cible, à raison d'un opérateur par cœur, formant ainsi un pipeline. La prochaine évolution consistera à faire intervenir du parallélisme de données pour faire en sorte d'utiliser le maximum de cœurs du MPPA-256.

Une estimation de l'énergie dépensée dans le cas d'occupation maximale des cœurs a été effectuée. Elle est représentée Figure 9 en colonne « MPPA best estimate ». Cette estimation, probablement optimiste, est une représentation de l'ordre de grandeur des performances maximales pouvant être atteinte grâce à cette puce.

Ces résultats ont fait l'objet de plusieurs présentations, dont une à un séminaire sur la compilation, et un poster à une conférence scientifique sur la parallélisation.

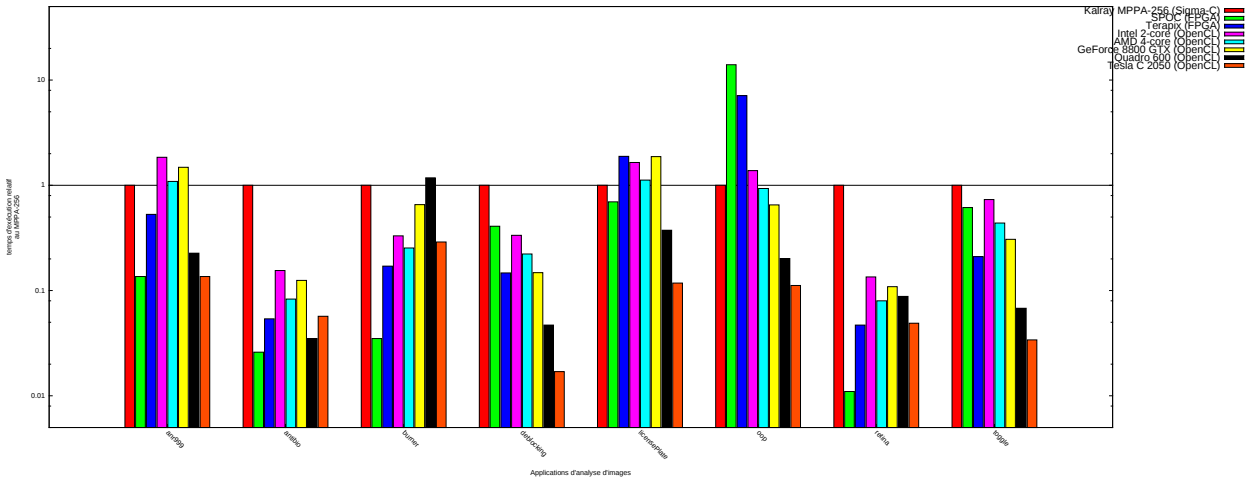


FIGURE 8 – Comparaison des temps d'exécution

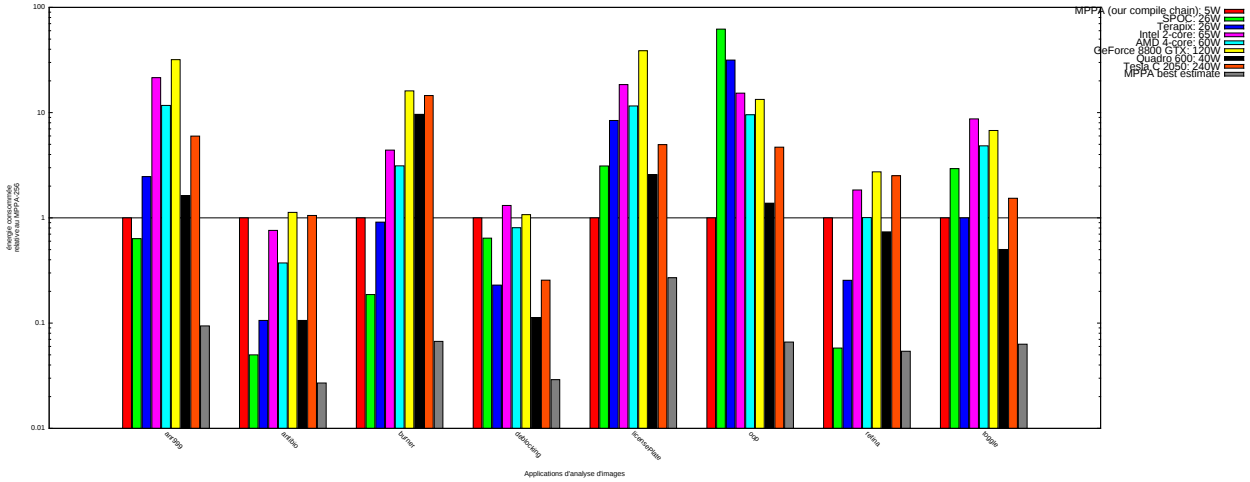


FIGURE 9 – Comparaison des énergies consommées

2.2.3 Travaux futurs

Les résultats évoqués précédemment montrent une marge de progression possible. À court terme, je vais donc me concentrer sur l'optimisation de cette chaîne de compilation, et notamment sur les éléments suivants :

déroulement de boucles `while` Quelques boucles `while` ne sont pas déroulée par notre compilateur, car dépendantes de caractéristiques des images traitées. Ceci génère des communications inutiles avec la machine hôte, que l'on peu réduire en déroulant ces boucles d'un facteur que l'on fera dépendre du nombre de cœurs de calculs non occupés ;

fusion d'opérateurs arithmétiques Dans le contexte *dataflow* du langage *Sigma-C*, ce sont les opérateurs les plus coûteux qui ralentissent les programmes. Les opérateurs arithmétiques connexes, particulièrement légers, peuvent être fusionnées afin de gagner en puissance de calcul, tant que le résultat n'est pas limitant ;

gestion des opérateurs de réduction Dans l'environnement d'exécution actuel, les opérateurs de réduction renvoient directement le résultat à la machine hôte. Si ce résultat est utilisé immédiatement, il convient de le diriger immédiatement vers les opérateurs concernés.

occupation d'un maximum de cœurs Notre chaîne de compilation génère des graphes *Sigma-C* tels quels : les images sont entièrement traités par une chaîne d'opérateurs qui sont chacun exécutés par un unique cœur de calcul. Le cas échéant, dupliquer les chaînes d'opérateurs pour les faire agir sur des portions d'images pourrait impacter positivement les performances, puisque les cœurs non occupés sont également alimentés.

J'espère pouvoir valoriser bientôt ces optimisations au travers d'une soumission à une conférence dédiée à la compilation.

À moyen terme, je m'intéresserai davantage aux applications de traitement du signal, pour déterminer comment adapter l'approche décrite ci-dessus. J'ai prévu d'étudier l'interface de programmation *VSIPL* (Vector Signal and Image Processing Library) décrite par Janka [20] à cette fin. Cette API décrit une bibliothèque logicielle en C et C++ dédiée au traitement du signal normalisée. L'étude consisterait à déterminer de quelle manière les opérateurs décrits dans cette API peuvent être décomposés en opérateurs de base afin de de les adapter plus efficacement à l'architecture cible, suivant l'approche décrite plus haut.

Évidemment, il s'agira de tester et de valider ces transformations sur des applications réelles. Pour ce faire, des contacts devront être noués avec des partenaires industriels (Thales ?) afin qu'ils puissent nous procurer leurs applications.

2.3 Étude de l'état de l'art

J'ai rassemblé un large échantillon d'articles et de rapports afin d'identifier l'état de l'art dans le domaine de la compilation de langages spécifiques à un domaine.

Techniques de compilation Citons tout d'abord le fameux *dragonbook* d'Aho et al [1], recueil de techniques de base pour construire un compilateur, ainsi que le livre de Zima [24], qui rassemble des transformations de code utiles pour les architectures parallèles. Ces livres sont indispensables pour celui qui s'intéresse de près à la compilation.

Compilation vers accélérateurs matériels La compilation vers des accélérateurs de type GPU a été étudié par Amini dans sa thèse [2], qui met en avant les transformations spécifiques à ce type d'architecture. La thèse de Bourgoin [9] introduit des langages de haut niveau dédiés à la programmation sur ces architectures en abstrayant la gestion de la mémoire. La généralisation des GPU dans les stations de travail et les superordinateurs modernes en font une cible de choix pour l'accélération des applications. Enfin, Coelho et Irigoin [11] proposent une méthodologie pour accélérer des applications d'analyse d'image sur différentes architectures grâce à un compilateur source-à-source. J'ai déjà explicité plus haut la mise en pratique de cette approche.

Transformations de code Concernant le thème des transformations de code, la thèse de Zory [25] explicite plusieurs transformations source-à-source pour optimiser les programmes

scientifiques. Guelton [16] avance une méthodologie et des outils afin de créer et manipuler facilement un compilateur source-à-source vers des architectures hétérogènes. L'article de Bandishti et al. [4] expose une transformation de pavage pour accélérer les opérations aux voisinages dont font partie les opérateurs de morphologie mathématique. L'étude des transformations de code sera essentielle dans mon doctorat afin d'adapter au mieux les applications au matériel cible via des optimisations spécifiques.

Gestion des mémoires – localité Les transferts de données sont souvent limitant dans les applications s'exécutant sur des architectures à mémoire distribuée. L'optimisation de ces transferts est souvent critique pour obtenir de bonnes performances. Cette problématique a notamment été étudiée par Bouchebaba durant son doctorat [8], qui s'intéresse particulièrement aux applications de traitement du signal, classe d'applications que je prévois d'étudier.

Traitement d'images Dans une approche parallèle à la nôtre, l'article de Ragan-Kelley et al. [23] propose un langage spécifique à une sous-classe particulière de traitement d'images ainsi qu'un compilateur dédié afin de gagner en confort de programmation ainsi qu'en performances, à l'aide d'optimisations à la compilation, destinée à une cible GPU.

3 Activités annexes

Le doctorat ne se compose pas uniquement de travaux de recherche au sein du centre. Le doctorant se doit de présenter ses travaux dans des conférences, sous la forme d'articles ou de posters. Il est également tenu à participer à des modules de formation scientifiques ou professionnalisants, ainsi qu'à donner des heures d'enseignement.

Dans ce chapitre, je détaillerai l'ensemble de ces activités annexes auxquelles j'ai participé durant les premiers mois de ce doctorat.

3.1 Conférences et séminaires

Au cours des derniers mois, j'ai présenté l'état de mes travaux au cours des conférences

7es Journées de la Compilation les 4, 5 et 6 décembre 2013 et

ComPAS'2014 du 22 au 25 avril 2014.

Ces conférences francophones ont rassemblé un grand nombre de chercheurs et doctorants spécialistes du domaine informatique.

Les 7es Journées de la Compilation sont un séminaire organisé par notre Centre, tenu à Dammarie-les-Lys (Seine et Marne). Ce séminaire rassemblait pendant trois jours une soixantaine de chercheurs œuvrant sur le thème de la compilation.

J'y ai présenté mes travaux sous la forme d'une présentation intitulée « Portage et optimisation d'applications d'analyse d'images sur architecture Kalray MPPA-Manycore » [2].

2. disponible en ligne à en suivant ce lien.

La conférence ComPAS'2014, d'envergure internationale, réunissait les communautés scientifiques francophones en Parallélisme, Architecture et Systèmes. Elle s'est déroulée à Neuchâtel en Suisse, dans les locaux de l'Université de Neuchâtel et a rassemblé environ 150 participants.

J'ai participé à la session posters de cette conférence en y exposant mes résultats.

De façon plus anecdotique, j'ai également présenté durant un séminaire interne au Centre quelques solutions logicielles pour augmenter son efficacité professionnelle [3].

Outre ces deux conférences, j'ai également assisté à plusieurs séminaires et soutenances de thèses :

— le 11 octobre 2013, soutenance d'HDR de Fabien Coelho,
— le 27 novembre 2013, soutenance de thèse de Dounia Khaldi, doctorante du centre,
— le 11 décembre 2013, soutenance de thèse de Mathias Bourgoin, doctorant au LIP6 à l'UPMC,
— le 27 février 2014, séminaire d'Alain Darte à l'ENSTA (Palaiseau, 91) sur le modèle polyédrique,
— le 14 avril 2014, séminaire sur les langages de programmation *streaming* et *dataflow* (St Germain au Mont d'Or, 69).

Je prévois d'assister aux 8es Journées de la Compilation qui se dérouleront du 1er au 3 juillet prochain à Nice, ainsi que d'effectuer des revues d'articles pour la conférence HPCC 2014 (High Performance Computing and Communications) qui se tiendra du 20 au 22 août prochain à Paris.

3.2 Portefeuille de compétences

Le doctorant est tenu de suivre au cours de son doctorat un minimum de 120 heures de formation, réparties en 60 heures de formation professionnalisante, et 60 heures de formation scientifique.

L'École propose un grand nombre de modules de formations professionnalisantes. Les doctorants peuvent s'y inscrire via le site ADUM [4].

Au cours de ces 8 derniers mois, j'ai assisté aux deux modules de formation suivants :

Intelligence Économique et Stratégique les 11, 12 et 13 mars 2014 et

Lecture Rapide les 24, 25 et 31 mars 2014.

J'ai ainsi accumulé 42h de formation sur les 60 heures demandées.

Je n'ai pas suivi de formation scientifique durant cette année, mes travaux ayant occupé la majorité de mon temps.

J'ai passé le test du TOEIC en mai 2013, et ai obtenu un score de 950 sur 990.

3.3 Activités d'enseignement

Le Centre dispense plusieurs cours aux élèves du cycle Ingénieurs Civils. Dans ce cadre, j'ai assisté Fabien Coelho durant quelques séances de travaux pratiques dans les cours qu'il enseigne, à savoir les Enseignements Spécialisés

3. cette présentation est disponible à cette adresse.
4. http://www.adum.fr/index.pl

Applications Réparties cours de 3e année sur la programmation réseau et

Systèmes d'Information cours de 2e année sur les bases de données.

J'ai participé à deux séances de deux heures pour l'ES Applications Réparties, les 24 février et 10 mars derniers, ainsi qu'à trois séances de deux heures l'ES Systèmes d'Information les 9, 12 et 16 mai derniers.

Conclusion

Cela fait bientôt huit mois que j'ai débuté mon doctorat au Centre de recherche en informatique de l'École des mines de Paris. Le sujet de thèse associé consiste en la compilation de langages spécifiques à un domaine vers une variété d'architectures matérielles sous contrainte de performance et de consommation d'énergie.

Durant cette période, je me suis principalement concentré sur un exemple particulier d'architecture massivement parallèle basse consommation, sur laquelle j'ai compilé à partir d'une interface spécifique des applications d'analyse d'images.

Avec une dizaine d'applications fonctionnelles, j'ai pu présenter mes résultats à deux reprises, lors d'un séminaire sur la compilation d'une part, et d'autre part sous forme de poster lors d'une conférence sur les architectures parallèles. Je prévois également d'en tirer un article pour soumission à court terme.

Ce travail m'aura permis d'identifier les difficultés auxquelles je risque d'être confronté durant les prochaines années. Le but étant de s'intéresser à d'autres domaines applicatifs et d'autres types d'architectures matérielles, en commençant par le traitement du signal.

En parallèle, j'ai étoffé mon portefeuille de compétences en participant à deux modules de formation professionalisants proposés par l'École. J'ai mené des activités d'enseignement auprès des élèves du Cycle ingénieurs civils.

Ainsi, j'espère par ce document avoir démontré mon implication dans ce sujet de thèse durant ces premiers mois de mon doctorat.

Références

[1] Alfred V Aho, Monica Lam, Ravi Sethi, and Jeffrey Ullman. *Compilateurs : principes, techniques et outils*. Pearson Education, Paris, 2007.

[2] Medhi Amini. *Source-to-Source Automatic Program Transformations for GPU-like Hardware Accelerators*. PhD thesis, MINES ParisTech, December 2012.

[3] Corinne Ancourt. *Génération automatique de codes de transfert pour multiprocesseurs à mémoires locales*. PhD thesis, Paris VI, March 1991.

[4] Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula. Tiling stencil computations to maximize parallelism. November 2012.

[5] Cédric Bastoul. *Improving Data Locality in Static Control Programs*. PhD thesis, UPMC, December 2004.

[6] Cédric Bastoul. *Contributions à l'optimisation des programmes de haut niveau*. PhD thesis, Paris Sud, December 2012.

[7] Bruno Bodin. *Analyse d'applications flot de données pour la compilation multiprocesseur.* PhD thesis, UPMC, December 2013.

[8] Youcef Bouchebaba. *Optimisation des transferts de données pour le traitement du signal : pavage, fusion et réallocation des tableaux.* PhD thesis, MINES ParisTech, November 2002.

[9] Mathias Bourgoin. *Abstractions Performantes pour Cartes Graphiques.* PhD thesis, UPMC, December 2013.

[10] Fabien Coelho. *Contributions à la compilation du High Performance Fortran.* PhD thesis, MINES ParisTech, March 1996.

[11] Fabien Coelho and François Irigoin. API compilation for image hardware accelerators. *ACM Transactions on Architecture and Code Optimization*, 9(4) :1–25, January 2013.

[12] Béatrice Creusillet. *Analyses de régions de tableaux et applications.* PhD thesis, MINES ParisTech, May 1996.

[13] Alain Darte and Frédéric Vivien. On the optimality of allen and kennedy's algorithm for parallelism extraction in nested loops. *Parallel Algorithms and Applications*, 12(1-3) :83–112, January 1997.

[14] Benoit Dupont de Dinechin, Duco van Amstel, Marc Poulhiès, and Guillaume Lager. Time-critical computing on a single-chip massively parallel processor. January 2014.

[15] Thierry Goubier, Renaud Sirdey, Stéphane Louise, and Vincent David. ΣC : a programming model and language for embedded manycores. 2011.

[16] Serge Guelton. *Building Source-to-Source Compilers for Heterogenous Targets.* PhD thesis, Telecom Bretagne, October 2001.

[17] F. Irigoin and R. Triolet. Supernode partitioning. pages 319–329. ACM Press, 1988.

[18] François Irigoin. *Partitionnement des boucles imbriquées - Une technique d'optimisation pour les programmes scientifiques.* PhD thesis, MINES ParisTech, June 1987.

[19] Laurent Jacques, Laurent Duval, Caroline Chaux, and Gabriel Peyré. A panorama on multiscale geometric representations, intertwining spatial, directional and frequency selectivity. *Signal Processing*, 91(12) :2699–2730, December 2011.

[20] Randall Janka and Randall Judd. Vsipl : An object-based open standard api for vector, signal, and image, 2000.

[21] Gilles Kahn. The semantics of a simple language for parallel programming. page 5, 1974.

[22] Dounia Khaldi. *Automatic Resource-Constrained Static Task Parallelization – A Generic Approach –.* PhD thesis, MINES ParisTech, November 2013.

[23] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide : A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *PLDI 2013*, page 12, June 2013.

[24] Hans Zima. *Supercompilers for parallel and vector computers.* ACM Press frontier series. ACM Press ; Addison-Wesley, New York, N.Y. : Wokingham, England ; Reading, Mass, 1990.

[25] Julien Zory. *Contribution à l'optimisation de programmes scientifiques.* PhD thesis, MINES ParisTech, December 1999.