Preuves en Coq de Programmes Multi-ML

Frédéric Gava et Julien Tesson
frederic.gava@univ-paris-est.fr
julien.tesson@lacl.fr

LACL (Université de Paris-Est)

17 novembre 2016

Thématiques : Coq, Vérification, Extraction, Certification, Parallélisme, Programmation fonctionnelle, OCaml, calcul haute-performance, multi-ML, programmation distribuée

Le stage se déroulera au :
Laboratoire d'Algorithme, Complexité, Logique (LACL),
http://www.lacl.fr
Université de Paris-Est Créteil, Val-de-Marne,
61 avenue du Général de Gaulle, 94010 Créteil cedex, France

Équipe : Vérification et Spécification des Systèmes
Directeur du laboratoire : Régine Laleau (laleau@u-pec.fr)

1 Introduction

1.1 Généralités

Certains problèmes, comme la simulation de phénomènes physiques et biologiques ou l'analyse d'une grande masse de données, nécessitent des performances que seules les machines **massivement parallèles** peuvent offrir. Leur programmation demeure néanmoins plus difficile que celle des machines séquentielles.

Actuellement, les données sont distribués sur des centaines (ou plus) de machines, chacune ayant des dizaines (ou plus) de processeurs et de coeurs. Ce type de machines est appelé **hybride** car on y trouve de la mémoire partagée (pour les coeurs) et de la mémoire distribuée (entre les machines). Pouvoir facilement programmer ce type de machines **massivement parallèles** est bien évidement un sujet de recherche actif.

La conception de langages adaptés, le développement et l'implantation d'algorithmes dédiés et de bibliothèques pour le calcul haute performance sont des sujets de recherches actives. De plus, la **vérification** de codes pour machines hybride n'est qu'à ses balbutiements comparé à la vérification de code séquentiels.

1.2 Le modèle multi-BSP

Le **parallélisme de données** est un paradigme de programmation parallèle dans lequel un programme décrit une séquence d'actions sur des valeurs parallèles. Le modèle **BSP** [2, 6] (Bulk-Synchronous Parallelism) vise à maximiser la portabilité des performances en ajoutant une notion de **processus explicites** au parallélisme de données.

Un programme BSP est écrit avec un paramètre indéfini qu'est le nombre de processeurs, et s'exécute en fonction du nombre de processeurs de l'architecture sur laquelle il a été lancé. Le modèle d'exécution BSP sépare synchronisation et communication et oblige les deux à être des **opérations collectives**. Il propose un **modèle de coût** [1] fiable et simple permettant de **prévoir les performances** de façon réaliste et portable sur un grand nombre d'architectures allant de la simple grappe de PC jusqu'aux multi-coeurs. Ainsi, il est possible d'**optimiser** les programmes à l'aide du bon choix algorithmique et suivant les paramètres BSP de la machine.

1. On appelle "coût" la complexité algorithmique d'un programme parallèle.

Le modèle multi-BSP [8] est une récente extension du modèle BSP en rajoutant de manière cohérente une hiérarchie de mémoire. Ainsi, une machine multi-BSP et vue comme un “arbre” de machines BSP : chaque machine BSP se décompose elle même en sous-machines BSP et ainsi de suite jusqu’aux coeurs de calcul.

Le modèle multi-BSP permet de mieux rendre compte des architectures **hybrides** car on voit par exemple une grappe de multi-coeurs comme un arbre a 2 niveau : un niveau (une machine BSP) pour les machines et un niveau pour les coeurs (une machine BSP pour chaque multi-coeurs).

1.3 Programmation fonctionnelle parallèle : BSML et multi-ML

Bulk Synchronous Parallel ML (**BSML**) est un langage **fonctionnel** pour la programmation parallèle BSP. Il est issu de recherches menées au **LIFO** (Orléans), dans le projet "Parallélisme, Réalité virtuelle et Vérification de systèmes" (PRV) et au **LACL** (UPEC), dans l’équipe "Vérification et Spécification des Systèmes".

Il existe une implantation partielle de ce langage sous forme d’une bibliothèque [2] pour le langage **Objective Caml** (OCaml). Cette bibliothèque permet d’écrire des programmes fonctionnels parallèles BSP portables sur une grande variété d’architectures. L’utilisation d’un langage fonctionnel (et donc très expressif) comme OCaml permet des implantations **efficaces** (en terme de performance) et **génériques** (**polymorphisme**, modules, **abstraction** des données, *etc.*) des algorithmes BSP. Pour BSML, il existe un **toplevel** permettant de tester et debugguer ses programmes sur n’importe quel PC. Il existe aussi un compilateur pour générer du code utilisant la bibliothèque **MPI**, standard de la programmation distribuée et donc du calcul haute-performance.

Dans le cadre d’une thèse en cours, nous développons une **extension** de BSML pour la programmation multi-BSP : **multi-ML** [1]. Il s’agit de structurer une **imbrication** d’expressions BSML, chacune ayant pour rôle de gérer son propre niveau BSP. L’imbrication se fait à l’aide d’appels récursifs spécifiques permettant de traverser la hiérarchie des mémoires jusqu’aux feuilles (les coeurs). Comme pour BSML, il y a un toplevel et un compilateur pour MPI. A la différence de BSML, un **système de types** est en cours d’implantation, permettant d’éviter une mauvaise utilisation du parallélisme (deadlocks notament).

1.4 Vérification et extraction de code BSML en Coq

Des premiers travaux de notre équipe et de celle du LIFO ont porté sur la vérification de programmes BSML en Coq [4, 3, 5, 7].

En effet, BSML est une extension parallèle **purement fonctionnelle** de OCaml. Les **primitives** peuvent être “**axiomatisées**” en Coq et ensuite utilisées en Coq comme des fonctionnalités normales. Ensuite, on peut prouver (comme d’“habitude” en Coq) des programmes puis en **extraire** des programmes BSML (OCaml+primitives) qui peuvent être exécutés sur le simulo séquentiel ainsi que sur des machines massivement parallèles [7].

2 But et déroulement du stage

Comme décrit précèdement, BSML (basé sur le modèle BSP) n’est plus adapté (et suffisament efficace) pour les machines modernes hybrides. Nous avons conçus multi-ML en se sens. Malheureusement, nous avons perdu les possibilités d’extraction de programmes parallèles certifiés.

A la différence de BSML, Multi-ML n’utilise pas que des primitives mais il est aussi basé sur une **extension syntaxique** de OCaml, par la définition de “multi-fonctions”. Se “contenter” d’axiomatiser des primitives ne semble plus suffisant. Nous avons commencé à regarder plusieurs solutions possibles :

1. Utiliser les possibilités d’extensions syntaxiques de Coq
2. Utiliser un petit outil externe qui prend en entrée des codes Coq (juste la partie ML) étendues pour multi-ML et qui en sortie génère des codes Coq simulant le code multi-ML (sequentialisation)
3. Définir de nouvelles primitives multi-BSP plus simples que le langage multi-ML lui-même et qui seraient axiomatisables en Coq (et suffisament expressives en pratique)

le stage porte donc sur l’étude des ces pistes pour pouvoir prouver en Coq des programmes multi-ML et pouvoir extraire depuis Coq des programmes multi-ML certifiés. Nous pourrions ensuite comparer l’**efficacité** des programmes extraits avec ceux écrits à la main.

2. `http://bsmllib.free.fr`

L'accent sera mis sur la correction de l'approche, sur son **expressivité** et surtout sur la **facilité de prise en main** pour des utilisateurs externes : pour nos partenaires du LIFO (Orléans), est-ce qu'il est facile ou non de pouvoir prouver en Coq des programmes multi-ML ? Nous avons déjà plusieurs programmes BSML certifiés et nous souhaitons aussi pouvoir comparer et "mesurer" la difficulté des preuves de correction entre des programmes BSML et des programmes multi-ML.

Des premiers benchmarks des implantations seront effectués tout d'abord sur un **simulateur séquentiel** puis sur les **grappes de PC** du LIFO (une grappe de 12 nœuds chacun de 8-cœurs).
Le stage (5/6 mois) pourra être organisé de la manière suivante :

1. (mois 1) Apprentissage (ou remise à niveau) de la programmation OCaml et de l'algorithmique parallèle ;
2. (mois 1) Apprentissage de Coq si besoin
3. (mois 1) Apprentissage de la programmation multi-ML (avec le doctorant Victor Allombert, principal développeur de multi-ML) ;
4. (mois 2-3) Comparaison des différentes méthodes
5. (mois 3-5) Preuves de petits exemples (programmes multi-ML simples)
6. (mois 6) Rédaction du mémoire et préparation de la soutenance.

Des connaissances en programmation fonctionnelle (OCaml ou SML ou Haskell ou Lisp ou Scheme) et en algorithmique parallèle et dans les algorithmes de graphes seraient les bienvenues mais, ne sont pas obligatoires (un goût pour la programmation efficace est, en revanche, conseillé). Une publication de ces résultats dans une conférence nationale (ou internationale) est envisageable. Possibilité de continuer, par la suite, par une thèse dans l'équipe "Verif-Specif" au LACL si une allocation de thèse est obtenue. Comme tout stage de M2, une indemnité mensuelle correspondant au 1/3 du SMIC (environ 500 euros) sera accordée par le LACL.

Références

[1] Victor Allombert, Frédéric Gava, and Julien Tesson. Multi-ML : Programming Multi-BSP Algorithms in ML. *Journal of Parallel Programming*, page 20, 2015. to appear.

[2] R. H. Bisseling. *Parallel Scientific Computation. A structured approach using BSP and MPI.* Oxford University Press, 2004.

[3] Wadoud Bousdira, Frédéric Loulergue, and Julien Tesson. A verified library of algorithmic skeletons on evenly distributed arrays. In Yang Xiang, Ivan Stojmenovic, Bernady O. Apduhan, Guojun Wang, Koji Nakano, and Albert Y. Zomaya, editors, *Algorithms and Architectures for Parallel Processing (ICA3PP)*, volume 7439 of *LNCS*, pages 218–232. Springer, 2012.

[4] F. Gava. Formal Proofs of Functional BSP Programs. *Parallel Processing Letters*, 13(3) :365–376, 2003.

[5] Louis Gesbert, Zhenjiang Hu, Frédéric Loulergue, Kiminori Matsuzaki, and Julien Tesson. Systematic Development of Correct Bulk Synchronous Parallel Programs. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 334–340. IEEE, 2010.

[6] D. B. Skillicorn, J. M. D. Hill, and W. F. McColl. Questions and Answers about BSP. *Scientific Programming*, 6(3) :249–274, 1997.

[7] Julien Tesson. *Environnement pour le développement et la preuve de correction systématiques de programmes parallèles fonctionnels*. PhD thesis, LIFO, University of Orléans, November 2011.

[8] L. G. Valiant. A bridging model for multi-core computing. *J. Comput. Syst. Sci.*, 77(1) :154–166, 2011.