Conception d'algorithmes multi-BSP sur les graphes et leur implantation à l'aide d'une extension de OCaml dédiée à la programmation distribuée multi-BSP

Frédéric Gava, Sovanna Tan et Julien Tesson
frederic.gava@univ-paris-est.fr
sovanna.tan@univ-paris-est.fr
julien.tesson@lacl.fr

LACL (Université de Paris-Est)

17 novembre 2016

Thématiques : Algorithmique sur les graphes, parallélisme, programmation fonctionnelle, OCaml, calcul haute-performance, multi-ML, Big-Data, 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.

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.

Récemment, avec la problématique du "**Big-Data**" (no-SQL) et de l'analyse de grandes masses de données (provenant par exemple, mais pas que, des réseaux sociaux), le développement et l'implantation efficace d'algorithmes traitant de très **grands graphes distribués** a regagné en intérêt. Ces graphes 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.

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** [3, 14] (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 synchroni-

sation 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.

Le modèle multi-BSP [16] 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** [2]. 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).

2 Algorithmes sur les graphes

2.1 OCamlGraph

De nombreux problèmes se résument en fait à un problème sur les graphes. Il existe de nombreuses bibliothèques et leur utilité n'est plus à démontrer. Par exemple, la plus connue pour le langage C++ est Leda [12]. On trouve aussi une bibliothèque pour Java [1], *etc.*

Il existe aussi des bibliothèques dédiées à la programmation d'algorithmes de graphes distribués. Généralement, ces langages se basent sur un modèle d'exécution à la BSP : chaque noeud du graphe, à partir de données provenant de ces voisins, calcul une nouvelle valeur ainsi que des données à renvoyer à ses voisins. On itère ce processus jusqu'à obtenir un point fixe. Malheureusement, ces langages, bien que très aisés à utiliser pour manipuler de grands graphes, ne sont pas extensibles (on est bloqué sur certaines manipulations) et ne tiennent pas forcement compte de la hiérarchie des mémoires d'une machine hybride (pour les performances). On trouve bien sûr **MapReduce** [6] et une version pour Matrice/Graphes en Java [13], Google's **Pregel** [11] pour (et sa version libre **Giraph**), GraphPar [7], *etc.*. Pregel par exemple a été quelques temps utilisé pour le calcul du PageRank et les chemins les plus courts, chez Google.

Malheureusement, les bibliothèques (séquentielles) sur les graphes ne sont pas aussi modulaires et génériques que **OCamlgraph** [3]. La bibliothèque OCamlGraph présente trois aspects différents :

1. Elle fournit un **module** regroupant une représentation abstraite des graphes et de nombreux algorithmes et opérations sur ceux-ci ;

1. On appelle "coût" la complexité algorithmique d'un programme parallèle.
2. `http://bsmllib.free.fr`
3. Téléchargeable à `http://ocamlgraph.lri.fr/index.fr.html`

2. Ensuite, Ocamlgraph fournit plusieurs structures de données pour les graphes. Certaines sont **persistantes** (purement fonctionnelles) et d'autres **impératives**. Certaines correspondent à des graphes orientés et d'autres à des non-orientés. Certaines étiquettent les noeuds, d'autres les arêtes, d'autres les deux. *etc.* Ces implantations sont écrites sous la forme de **foncteurs**[4] : on spécifie le type des sommets, des arêtes, de leurs labels, etc. et on récupère alors une structure de données ;
3. Enfin, Ocamlgraph fournit plusieurs opérations et **algorithmes classiques** sur les graphes. Ils sont également écrits comme des foncteurs, c'est-à-dire indépendamment de la structure de données utilisée pour représenter les graphes. En conséquence, on peut définir sa propre structure de données et réutiliser tous les algorithmes de cette bibliothèque — il suffit de fournir quelques opérations telles que l'itération sur tous les sommets, sur tous les successeurs d'un noeud, *etc.*.

2.2 Algorithmes BSP et multi-BSP sur les graphes

Il existe de nombreux **algorithmes BSP** effectuant des opérations sur les graphes (plus court chemin, flot maximum, décomposition, *etc.*) dans la littérature [15, 4, 10, 9, 8] (pour ne citer qu'eux). Malheureusement, ils sont rarement implantés et on trouve encore plus rarement une bibliothèque complète.

Par exemple, il existe une bibliothèque CGM (CGM est un modèle très proche de BSP) de graphes [4]. Néanmoins, elle est écrite en C++ pour des programmes C++. *Quid* donc du polymorphisme et de la sûreté d'exécution des programmes ? Les possibilités de composition des programmes s'en trouvent aussi limitées. Aussi, les graphes sont une structure très utile dans les algorithmes de vérification de programmes. Les données étant très diverses, combiner le polymorphisme de ML (ocamlgraph) et l'efficacité du parallélisme seraient un plus.

Pire, à notre connaissance, il n'existe pas d'algorithme multi-BSP pour les graphes. Ils sont tous à adapter de BSP (au mieux) ou a re-inventer.

Tout cela est problématique car dans le cadre du Big-data, beaucoup de programmeurs "réinventent la roue" en implantant des méthodes qui auraient pu être génériques et efficaces dans une vrai bibliothèque de graphes.

3 But et déroulement du stage

Le sujet du stage est la **conception** et l'**implantation** de divers **algorithmes multi-BSP sur les graphes en multi-ML** et à l'avenir (dans le cadre d'une thèse ?) du développement d'une **bibliothèque applicative générique**. Nous pouvons prendre, comme base, pour commencer, les algorithmes décrits dans [15, 4, 10, 9, 8]. Mais rechercher d'autres travaux existants sera aussi une première partie du stage. La bibliothèque envisagée doit au maximum rendre « transparent » le parallélisme à l'utilisateur. Les possibilités de réduction de listes, d'arbres, les VList (ou les Skip-List [5]) seront aussi à étudier si besoin est.

L'accent sera mis sur la conception d'algorithmes efficaces et sur des implantations les plus génériques (c'est-à-dire indépendantes de ce que contiennent les graphes) possibles : utilisation du polymorphisme de OCaml. Les premiers **tests** et benchmarks des implantations seront effectués tout d'abord sur un **simulateur séquentiel** puis sur les **grappes de PC** du LIFO (grappe de 12 nœuds chacun de 8-cœurs). Des comparaisons entre les performances théoriques (d'après les formules de coûts multi-BSP et les paramètres multi-BSP desmachines) et réelles seront aussi effectuées afin de valider expérimentalement l'approche et les implantations. Des tests sur de plus grosses machines seront possibles dans le cas du succès des précédents tests.

Le stage (5/6 mois) sera organisé de la manière suivante :

1. (mois 1) Apprentissage (et mise à niveau) de la programmation OCaml et de l'algorithmique parallèle ;
2. (mois 1) Apprentissage de la programmation multi-ML (avec le doctorant Victor Allombert, principal développeur de multi-ML) ;
3. (mois 1) Petit travail bibliographique sur l'algorithmique des graphes ;
4. (mois 2-3) Conception des éléments de base de la bibliothèque (constructeurs parallèles, itérateurs, réductions *etc.*) ;

4. Un foncteur est un module prenant en paramètre un autre module comme par exemple un foncteur des ensembles qui prendrait en paramètre un module contenant un type de données et une fonction de comparaison de ces données

5. (mois 3-5)Conception puis implantation d'algorithmes multi-BSP de graphes [5] et conception de la bibliothèque ;
6. (mois 5) Petit travail bibliographique pour la recherche d'applications possibles ;
7. (mois 5) Test des algorithmes sur différentes machines parallèles et vérification des performances par rapport aux formules de coûts ;
8. (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] The Data Structures Library in Java. `http://www.cs.brown.edu/cgc/jdsl/`.

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

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

[4] A. Chan, F. Dehne, and R. Taylor. Implementing and Testing CGM Graph Algorithms on PC Clusters and Shared Memory Machines. *Journal of High Performance Computing Applications*, 2005. to appear.

[5] P. Corbineau. Skip lists and probabilistic binary search trees. In *Journées Francophones des Langages Applicatifs*, 2005.

[6] J. Dean and S. Ghemawat. MapReduce : a Flexible Data Processing Tool. *Commun. ACM*, 53(1) :72–77, 2010.

[7] Michael DeLorimier. *GRAph parallel actor language : a programming language for parallel graph algorithms*. PhD thesis, California Institute of Technology, 2013.

[8] A. Ferreira, I. Guérin-Lassous, K. Marcus, and A. Rau-Chauplin. Parallel computation on interval graphs : algorithms and experiments. *Concurrency and Computation : Practice and Experience*, 14(11) :885–910, 2002.

[9] A. V. Gerbessiotis, C. J. Siniolakis, and A. Tiskin. Parallel Priority Queue and List Contraction : The BSP approach. *Computing and Informatics*, 21 :59–90, 2002.

[10] I. Gu'erin-Lassous and J. Gustedt. Portable List Ranking : an Experimental Study. *ACM Journal of Experiments Algorithms*, 7(7) :1–18, 2002.

[11] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, and N. Leiser. Pregel : A System for Large-scale Graph Processing. In *Management of data*, pages 135–146. ACM, 2010.

[12] Kurt Mehlhorn and Stefan Näher. Leda : a platform for combinatorial and geometric computing. *Commun. ACM*, 38(1) :96–102, 1995.

[13] Sangwon Seo, Edward J. Yoon, Jae-Hong Kim, Seongwook Jin, Jin-Soo Kim, and Seungryoul Maeng. Hama : An Efficient Matrix Computation with the MapReduce Framework. In *Cloud Computing (CloudCom)*, pages 721–726. IEEE, 2010.

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

[15] A. Tiskin. *The Design and Analysis of Bulk-Synchronous Parallel Algorithms*. PhD thesis, Oxford University Computing Laboratory, 1998.

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

5. L'implantation de tous les algorithmes multi-BSP ne sera pas demandée mais bien entendu, plus il y en aura, mieux ce sera...