Une bibliothèque parallèle sur les graphes ; implantation générique et expérimentations

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)

14 novembre 2014

Thématiques : Algorithmique sur les graphes, parallélisme, programmation fonctionnelle, OCaml, calcul haute-performance

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

1 Introduction

1.1 Généralités

Certains problèmes, comme la simulation de phénomènes physiques, biologiques ou chimiques de grande taille, 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, la conception et l'implantation d'algorithmes et de bibliothèques parallèles sont des sujets de recherches actives.

Récemment, avec la problématique du "Big-Data" (no-SQL) et de l'analyse des données des réseaux sociaux, l'algorithme et l'implantation de très grands graphes distribués a regagné en intérêt.

1.2 Parallélisme et modèle 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, 9] (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 machines massivement parallèles.

Ainsi, il est possible d'**optimiser** les programmes à l'aide du bon choix algorithmique et suivant les paramètres BSP de la machine.

1.3 Programmation fonctionnelle parallèle : BSML

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

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

systèmes" (PRV) et au **LACL** (Paris XII), dans l'équipe "Vérification des Systèmes".

Il existe une implantation partielle de ce langage sous forme d'une bibliothèque (la **BSMLlib** [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.

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. La plus connue pour C++ est Leda [8] (on trouve aussi une bibliothèque pour Java [1]). Malheureusement, celles-ci ne sont pas aussi modulaires et génériques que OCamlgraph.

Ocamlgraph [3] est une bibliothèque pour OCaml. Elle présente trois aspects différents :

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

Il existe de nombreux algorithmes BSP effectuant des opérations sur les graphes (plus court chemin, composants fortement connexes, *etc.*) dans la littérature [10, 3, 7, 6, 5]. 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 [3]. 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 algos de vérification de programmes. Les données étant très diverses, le polymorphisme de ML (ocamlgraph) et l'efficacité seraient un plus.

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 dans une bibliothèque de graphes.

3 But et déroulement du stage

Le sujet du stage est la conception et l'**implantation** de divers algorithmes BSP sur les graphes en BSML sous la forme d'une **bibliothèque applicative générique**. Nous prendrons comme base, pour commencer, les algorithmes décrits dans [10, 3, 7, 6, 5]. Cette bibliothèque doit au maximum rendre « transparent » le parallélisme à l'utilisateur. Les possibilités de réduction de listes, d'arbres, les VList (ou les Skip-List [4]) seront aussi à étudier.

2. `http://bsmllib.free.fr`
3. Téléchargeable à `http://ocamlgraph.lri.fr/index.fr.html`
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

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

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

1. Apprentissage (ou remise à niveau) de la programmation OCaml et de l'algorithmique parallèle ;
2. Apprentissage de la programmation BSML ;
3. Petit travail bibliographique sur l'algorithmique des graphes ;
4. Conception des éléments de base de la bibliothèque (constructeurs parallèles, itérateurs, réductions *etc.*) ;
5. Implantation d'algorithmes BSP de graphes [5] et conception de la bibliothèque ;
6. Test des algorithmes sur différentes machines parallèles et vérification des performances par rapport aux formules de coûts ;
7. Comparaisons de performances avec la bibliothèque C++ [3] ;
8. Petit travail bibliographique pour la recherche d'applications possibles ;
9. Test de l'application sur différentes machines parallèles ;
10. 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 matriciels 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 400 euros) sera accordée par le LACL.

Références

[1] The Data Structures Library in Java. `http://www.cs.brown.edu/cgc/jdsl/`.

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

[3] 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.

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

[5] 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.

[6] 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.

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

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

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

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

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