Flexibilité et Portabilité pour Embarrassingly Parallel Search

Samvel Balassanian Dersarkissian, Arnaud Malapert

Flexibilité et Portabilité pour Embarrassingly Parallel Search

Samvel Balassanian Dersarkissian, Arnaud Malapert

Université Côte d'Azur, CNRS, I3S, France
{samvel.dersarkissian,arnaud.malapert}@univ-cotedazur.fr

Mots-clés : *Programmation par contraintes, Solveur de contraintes, Parallélisme*

1 Introduction

La programmation par contraintes a pour but la résolution de problèmes combinatoires. Toutefois, la résolution de problèmes complexes peut être longue. Une façon d'accélérer cette résolution est la parallélisation des calculs [1]. Elle permet de tirer pleinement profit des processeurs multicœurs et des architectures des machines modernes. La méthode Embarrassingly Parallel Search (EPS) [2] est basée sur la décomposition d'un problème initial en un très grand nombre de sous-problèmes disjoints. Ces sous-problèmes sont ensuite résolus indépendamment en parallèle, par des unités de calcul nécessitant peu, voire aucune communication entre elles.

EPS est une méthode générique, cependant ses implémentations actuelles dépendent des architectures des machines pour lesquelles elles ont été programmées. Ceci limite fortement leur portabilité en pratique. Nous proposons de revoir le fonctionnement des communications au sein d'EPS de sorte à la rendre utilisable sur un ensemble de machines de différentes architectures. Nous exposons ensuite l'implémentation d'un prototype combinant des approches par partitionnement et par *portfolio*.

2 Embarrassingly Parallel Search

La méthode EPS se divise en deux étapes principales : décomposer un problème initial en *sous-problèmes*, puis faire résoudre ces sous-problèmes indépendamment les uns des autres, et en parallèle, par des *workers* coordonnés par un *master*. Lors de la décomposition, le master découpe le problème initial en sous-problèmes qui seront ensuite résolus par les workers.

Étant donné l'indépendance des sous-problèmes générés et des résolutions, on suppose que la somme de leurs temps résolution sera comparable au temps de résolution du problème initial. Un grand nombre de sous-problèmes permet d'équilibrer la charge de travail attribuée à chaque worker et d'exploiter pleinement les mécanismes d'ordonnancement.

La résolution des sous-problèmes n'est pas faite directement par EPS : les workers font appel à un solveur externe et transmettent ses résultats au master. Cette approche est intéressante, car elle permet de tirer profit des différents solveurs modernes.

Les techniques de portfolio – telles que celles employées en SAT – consistent en l'utilisation de plusieurs solveurs (ou configurations de solveurs) différents en même temps, pour résoudre un même problème. Généralement, on récupère ensuite la meilleure solution selon un critère particulier (temps de résolution, meilleure approximation, etc.). EPS implémente le portfolio en proposant le même problème (ou sous-problème) à plusieurs workers configurés différemment.

3 Des limites pratiques

EPS est principalement limitée par ses implémentations actuelles qui sont fortement dépendantes des machines pour lesquelles elles ont été développées. Actuellement, EPS nécessite un portage ad hoc pour chaque infrastructure ou plateforme matérielle.

En effet, les langages de programmations et librairies logicielles utilisées traditionnellement en programmation par contraintes et pour le calcul hautes performances (C++, MPI, OpenMP, etc.) rendent difficile la production d'un code générique et portable. Elles nécessitent souvent d'adapter ou recompiler le code pour chaque architecture de machine.

D'autres technologies de parallélisme de plus haut niveau (Threads en Java ou C++ par exemple) sont également utilisées fréquemment, cependant elles nécessitent toujours l'écriture d'un code complexe.

De plus, le master a un certain nombre de rôles non triviaux au sein d'EPS : il doit définir les sous-problèmes, les transmettre aux workers, récupérer les résultats, gérer les modifications de données (objectifs) et les éventuelles terminaisons (time-out). Le nombre et la complexité de ces différents rôles sont un frein à une implémentation générique d'EPS.

4 Un prototype plus générique

Pour se soustraire aux limitations induites par le master, nous avons révisé l'architecture de communication d'EPS. Nous avons apporté une granularité plus fine à son fonctionnement en introduisant de nouveaux acteurs et en précisant leurs rôles ainsi que les messages qu'ils transmettent. De plus, nous considérons dorénavant le worker comme une ressource de calcul quelconque et l'utilisons également pour la division du problème initial en sous-problèmes.

Nous implémentons cette architecture de communication au sein d'un prototype logiciel généraliste, indépendant des spécificités de bas niveau, et permettant de mettre en œuvre différents paradigmes de programmation (concurrence, parallélisme, distribution). Cette preuve de concept utilise le langage Java dont le bytecode est aisément portable. Nous utilisons également la libraire Scala *Akka* [4] qui permet un grand niveau d'abstraction et fournit implicitement plusieurs paradigmes de programmation parallèle et distribuée.

Ce logiciel est capable de tirer parti des architectures multicœurs. Il implémente également la possibilité d'utiliser des techniques de portfolio. Nous montrons la viabilité de cette preuve de concept logiciel en l'associant à un solveur (Choco [3]).

Dans un futur proche, nous voulons associer ce prototype à d'autres solveurs, puis le faire passer à l'échelle pour une utilisation au sein d'un cluster de machines. Finalement, nous souhaitons employer des techniques de sélection d'algorithme pour le choix d'un solveur adapté à la résolution d'un sous-problème donné.

5 Remerciements

Ce travail a reçu le soutien de l'Agence Nationnale de la Recherche à travers le projet MULTIMOD référencé ANR-17-CE22-0016.

Références

[1] Youssef Hamadi and Lakhdar Sais. *Handbook of Parallel Constraint Reasoning.* Springer, 2018.

[2] Arnaud Malapert, Jean-Charles Régin, and Mohamed Rezgui. *Embarrassingly Parallel Search in Constraint Programming.* Journal of Artificial Intelligence Research (JAIR), volume 57, 421–464, 2016.

[3] Charles Prud'homme and Jean-Guillaume Fages and Xavier Lorca. *Choco Documentation.* TASC - LS2N CNRS UMR 6241, COSLING S.A.S, 2017. http ://www.choco-solver.org

[4] Akka. *Lightbend, Inc.* http ://akka.io/