Embarrassingly parallel search pour la programmation par contraintes

Arnaud Malapert et Jean-Charles Régin

Université Côte d'Azur, CNRS, I3S, France
firstname.lastname@unice.fr

Mots-clés : *programmation par contraintes, parallélisme, recherche arborescente, portfolio.*

Parallélisme et concurrence Le principe de base du parallélisme est d'utiliser plusieurs unités de calcul concurremment pour accroître la puissance de calcul. L'objectif du parallélisme est non seulement de résoudre les problèmes plus rapidement, mais aussi de résoudre des problèmes de plus grande taille. Depuis les débuts de l'informatique, la plupart des programmes ont été définis pour s'exécuter sur une seule unité de calcul (généralement un processeur avec un seul cœur). De tels programmes sont dits séquentiels. Il y a 20 ans, les machines parallèles (possédant plusieurs processeurs) et les centres de calcul étaient très coûteux et réservés à l'usage de grandes entreprises ou de centres de recherche. Depuis une dizaine d'années, les ordinateurs intègrent des processeurs possédant plusieurs cœurs (processeur multi-cœur) et la puissance des centres de calcul est littéralement à portée de clics.

Paralléliser les programmes ou les méthodes de résolution est devenu nécessaire pour exploiter la puissance des architectures matérielles actuelles. Les difficultés pour paralléliser un programme sont induites par la gestion de la concurrence, c'est-à-dire les accès simultanés à une ressource partagée, ainsi que la communication et la synchronisation qui entraînent une perte de temps.

***Embarrassingly parallel search* pour la programmation par contraintes** Dans cet exposé, nous étudions la parallélisation de la procédure de recherche de solution(s) d'un problème en Programmation Par Contraintes (PPC). Nous présentons la méthode nommée *Embarrassingly Parallel Search* (EPS) [1, 2]. Cette méthode est basée sur la décomposition statique d'un problème en un très grand nombre de sous-problèmes disjoints qui sont ensuite résolus en parallèle par des unités de calcul avec très peu, voire aucune communication ou synchronisation. Dans une décomposition statique, toutes les tâches sont générées au début de la résolution et plus aucune subdivision n'est possible ensuite. Le principe d'EPS est d'arriver statistiquement à un équilibrage des temps de résolution de chaque unité de calcul afin d'obtenir une bonne répartition de la charge de travail. Cet équilibrage statistique ne signifie pas que les temps de résolution des sous-problèmes sont homogènes, mais plutôt que l'ordonnancement en temps réel de ces tâches offre une bonne accélération. La décomposition doit donc éviter les anomalies sans chercher à tout prix un partage en parts égales de l'espace de recherche.

EPS s'appuie sur la propriété suivante : la somme des temps de résolution de chacun des sous-problèmes est comparable au temps de résolution du problème en entier. Pour que cette propriété soit vérifiée, il ne faut pas que la décomposition génère de sous-problèmes triviaux, par exemple des sous-problèmes qui n'auraient même pas été considérés par une recherche séquentielle. Cette propriété est souvent vérifiée en PPC, ce qui nous permet de disposer d'une méthode simple et efficace en pratique.

Expérimentations EPS a été implémenté dans plusieurs solveurs de contraintes : OR-tools, Gecode et Choco2. Ceci a été possible car l'implémentation d'EPS est relativement aisée. En effet, ces implémentations s'appuient sur leurs solveurs de contraintes sous-jacents, mais elles n'ont pas requis de les modifier en profondeur.

Dans nos expérimentations, nous nous intéressons à la recherche de toutes les solutions d'un problème de satisfaction de contraintes, à prouver qu'un problème n'a pas de solution, et à la recherche d'une solution optimale d'un problème d'optimisation sous contraintes. Les résultats montrent que la décomposition doit générer au moins une trentaine de sous-problèmes par unité de calcul pour obtenir des charges de travail équivalentes entre les unités de calcul. Nous évaluons notre approche sur différentes architectures (machine multi-cœur, centre de calcul, et *cloud computing*) et montrons qu'elle obtient souvent un gain linéaire en fonction du nombre d'unités de calcul. Une comparaison avec d'autres méthodes telles que le *work stealing* ou le *portfolio* montre qu'EPS obtient de meilleurs résultats.

Développement en cours Des efforts ont été entrepris pour développer une nouvelle bibliothèque Java pour la parallélisation de solveurs de contraintes existants. La bibliothèque définit une architecture de communication et de synchronisation basée sur la norme *Message Passing Interface* (MPI) définissant une bibliothèque de fonctions pour exploiter des ordinateurs distants ou multiprocesseur par passage de messages. Cette architecture permet de construire des solveurs parallèles utilisant la méthode EPS, des portfolios de solveurs, ou des méthodes hybrides portfolios/EPS. Le travail du développeur est simplifié, car il ne gère plus la concurrence et il n'écrit que du code s'exécutant en séquentiel. Le solveur de contraintes Choco écrit en Java est une cible prioritaire, mais d'autres solveurs seront intégrés ensuite. Ces différents solveurs pourront collaborer.

Références

[1] Mohamed Rezgui. *Parallelism in constraint programming.* Thèse, Université Nice Sophia Antipolis, July 2015.

[2] Arnaud Malapert, Jean-Charles Régin, and Mohamed Rezgui. Embarrassingly Parallel Search in Constraint Programming. *J. Artif. Intell. Res. (JAIR)*, 57 :421 – 464, nov 2016.