Compilation itérative pour l'exécution de programmes chimiques sur une chaîne de compilation flot de données

Loïc Cudennec, Thierry Goubier

Compilation itérative pour l'exécution de programmes chimiques sur une chaîne de compilation flot de données.

Loïc Cudennec, Thierry Goubier

CEA, LIST
Saclay, France
{prénom.nom}@cea.fr

Mots-clés : *Programmation chimique, langage flot de données, compilation itérative.*

1 Introduction

Le paradigme de la programmation chimique [2] a été introduit à la fin des années 1980 comme une manière élégante de définir mathématiquement des programmes répartis. Le principe repose sur l'analogie des réactions chimiques, dans lequel un ensemble de molécules réagissent pour en former de nouvelles. La programmation chimique consiste à déclarer des données initiales typées ainsi que des opérateurs. L'ordre des réactions - la manière dont sont appliqués les opérateurs sur les données - est résolu à l'exécution, sans indications de la part du développeur. Les programmes chimiques sont par nature parallèles et non déterministes. Ce paradigme a été utilisé pour les grappes et grilles de calculateurs et il reste pertinent pour les processeurs many-coeurs. Une grande part de la complexité des programmes chimiques réside dans le logiciel système qui a la charge d'orchestrer les réactions. L'implémentation de ce logiciel fait resurgir les problèmes classiques d'exécution en environnement réparti, ce qui explique qu'il n'existe pas à notre connaissance d'implémentation efficace de langage chimique. L'objectif de ce travail est de fournir un support d'exécution efficace en se basant sur le paradigme de programmation flot de données.

2 Programmation chimique et programmation flot de données

Le paradigme de programmation flot de données structure les applications sous forme de graphe de tâches communicantes (des agents). Ce paradigme est largement utilisé dans les systèmes embarqués, pour des applications de traitement du signal notamment. Une implémentation efficace de ce paradigme est proposée par le langage ΣC [3], à l'origine conçu pour programmer la puce 256 coeurs MPPA [1]. Sur plusieurs aspects, le calcul chimique peut être comparé à un flot de données, dans lequel les réactions sont représentées par les agents et les molécules par les liens de communications. Réciproquement, une application flot de données peut décrire un ensemble de réactions chimiques appliquées sur des molécules, dans un ordre partiel. Dans cette analogie, écrire un programme chimique en ΣC consiste à déclarer un état initial composé d'un ensemble d'agents représentant les molécules (un agent avec un unique port de communication sur lequel est émis la valeur de la molécule) et un ensemble d'agents représentant les réactions (un agent avec des ports d'entrée et de sortie). Les connexions entre agents ne sont cependant pas décrites dans le programme. L'une des particularité de l'approche est de choisir un ordre partiel d'exécution des réactions chimiques lors de la phase de compilation. C'est lors de cette phase que les connexions sont établies afin de former un graphe d'instanciation de l'application. Ce graphe est ensuite exécuté sur la cible, produisant un ensemble de résultats assimilés aux molécules obtenues. Ces résultats peuvent alors être utilisés comme données initiales d'un nouveau programme chimique ΣC. Cette compilation rebouclée peut s'alimenter jusqu'à l'obtention d'un état stable.

3 Décider d'un ordre partiel des réactions chimiques

Un point dur de cette approche réside dans la construction d'un ordre partiel des réactions chimiques. D'un point de vue algorithmique, le problème consiste à construire un graphe de flot de données à partir des agents présents dans l'état initial. Les agents correspondant aux molécules initiales ne sont utilisés qu'une seule fois, alors que les agents correspondant aux réactions peuvent être répliqués autant que nécessaire. Il n'est pas obligatoire d'utiliser tous les agents dans la solution. Si dans le modèle chimique les réactions s'opèrent de manière dynamique et contextuelle, la construction statique d'une solution à la compilation permet de prendre en compte divers objectifs et contraintes. Parmi ces contraintes, nous pouvons citer le débit du flot de données, le degré de parallélisme, la profondeur du graphe ou encore le nombre d'instances et de canaux de communication. Ces contraintes peuvent servir des objectifs de performance d'exécution ou, plus pragmatiquement, permettre à l'application d'être déployée dans de bonnes conditions sur une cible embarquée. Dans ce papier nous ne proposons pas d'algorithmes employés pour calculer une solution. Cependant, nous pouvons discuter de la complexité d'exécution liée à l'intégration dans une boucle de compilation itérative. La construction du graphe doit s'effectuer de manière suffisamment rapide pour que l'enchaînement des phases de compilation et d'exécution ne soit pas pénalisée. Le modèle de calcul implique donc un équilibre matériel entre la machine de compilation et l'accélérateur many-coeurs. Les programmes chimiques peuvent de plus être conséquents en terme du nombre d'agents. Par exemple une application de calcul d'un maximum peut être décrite en autant d'agents que de valeurs à comparer, auxquels il faut ajouter un agent effectuant une sélection sur deux valeurs. Ce programme peut mener à un nombre considérable de solutions à évaluer, certaines étant équivalentes par symétrie. Par ailleurs, pour ce genre de programme, il est possible de paralléliser les phases de compilation et d'exécution en compilant une partie de l'application pendant que l'autre partie s'exécute sur la cible.

4 Conclusion

Le paradigme chimique offre un grand pouvoir d'abstraction pour la programmation d'architectures massivement parallèles. Dans ce papier nous proposons d'implémenter le paradigme chimique sur celui du flot de données, qui est aujourd'hui efficacement supporté. La solution proposée réduit l'aspect dynamique de l'exécution mais permet de prendre en compte des contraintes d'exécution plus fortes en optimisant l'ordre partiel des réactions chimiques.

Références

[1] Pascal Aubry, Pierre-Edouard Beaucamps, Frédéric Blanc, Bruno Bodin, Sergiu Carpov, Loïc Cudennec, Vincent David, Philippe Doré, Paul Dubrulle, Benoît Dupont De Dinechin, François Galea, Thierry Goubier, Michel Harrand, Samuel Jones, Jean-Denis Lesage, Stéphane Louise, Nicolas Morey Chaisemartin, Thanh Hai Nguyen, Xavier Raynaud, and Renaud Sirdey. Extended Cyclostatic Dataflow Program Compilation and Execution for an Integrated Manycore Processor. In *Alchemy 2013 - Architecture, Languages, Compilation and Hardware support for Emerging ManYcore systems*, volume 18, pages 1624–1633, Barcelona, Espagne, June 2013.

[2] Jean-Pierre Banâtre and Daniel Le Métayer. The gamma model and its discipline of programming. *Science of Computer Programming*, 15(1) :55 – 77, 1990.

[3] Thierry Goubier, Renaud Sirdey, Stéphane Louise, and Vincent David. Sigma-C : A programming model and language for embedded manycores. In Yang Xiang, Alfredo Cuzzocrea, Michael Hobbs, and Wanlei Zhou, editors, *Algorithms and Architectures for Parallel Processing*, volume 7016 of *Lecture Notes in Computer Science*, pages 385–394. Springer Berlin / Heidelberg, 2011.