Compilation et optimisation de langage parallèle

Nelson LOSSING – MINES ParisTech – Centre de Recherche en Informatique

Contexte

Avec le développement croissant de machines parallèles, les programmes doivent être repensés pour pouvoir pleinement tirer parti de la puissance de ces machines. De nombreux travaux concernant leur parallélisation existent. Deux approches sont classiquement utilisées : le parallélisme de données consiste à distribuer les données sur plusieurs processeurs qui effectuent le même calcul ; le parallélisme de tâches permet d'exécuter différents groupes d'instructions simultanément.

L'écriture de ces programmes parallèles est souvent difficile pour un non-expert du domaine. Bien plus difficile encore est la garantie que ce programme est correct et déterministe.

Aussi mes travaux de doctorat consistent à faciliter cette génération de code parallèle.

Objectif

L'objectif principal de ma thèse est de générer automatiquement du code parallèle à partir d'un code séquentiel, en utilisant une méthode innovante.

Cette génération doit, en effet, se faire étape par étape par une suite de transformations suffisamment simples pour que l'on puisse garantir leur correction.

Travaux réalisés

Je m'intéresse plus particulièrement au parallélisme de tâches, plus récent et moins souvent étudié dans la parallélisation automatique que le parallélisme de données. Pour cela, j'ai pu reprendre une partie des travaux réalisés par Dounia Khaldi [1] qui a travaillé sur la détection automatique de tâches pouvant être parallélisées dans un programme séquentiel.

L'architecture mémoire visée est une architecture à mémoire distribuée. Il faut donc assurer la cohérence des données, c'est-à-dire s'assurer que chaque donnée présente sur un processeur est à jour.

J'ai défini différentes étapes nécessaires à la génération automatique du code parallèle. Ces étapes peuvent être regroupées en phases qui structurent nos transformations. Plusieurs solutions peuvent être envisagées pour obtenir le résultat d'une phase.

Mes différentes phases de transformation sont :

1. la *détection* et le *placement* des tâches sur les différents processeurs disponibles ;
2. la *préparation* du code afin de pouvoir générer du code parallèle distribué ;
3. les *optimisations locales* au sein d'une tâche dans le but d'optimiser les instructions exécutées par un processeur et d'éliminer les communications redondantes ;
4. les *optimisations globales* au programme permettant de minimiser les communications ;
5. la *génération* du code parallèle distribué en convertissant le code par spécialisation sur une architecture cible ;
6. les *optimisations sur le code parallèle généré* afin d'optimiser les moyens de communications.

Plusieurs phases ont déjà été implémentées et des expérimentations ont montré la faisabilité de cette méthodologie.

Ainsi, j'arrive à générer automatiquement du code parallèle distribué.

Perspectives

Un article, concernant l'élimination de code mort en utilisant une analyse qui décrit les régions convexes de tableaux utilisées en lecture ou écriture, va être soumis à publication.

Dans la suite de ma thèse, je compte réaliser les preuves formelles de la correction des différentes étapes de ma génération de code.

Enfin, pour le moment, aucune optimisation sur le code généré n'est réalisée, en phase 6. Je vais poursuivre les études menées pour soit générer directement un meilleur code à la phase 5, soit optimiser le code généré avec la phase 6. Cela a pour but d'améliorer les performances, notamment la vitesse d'exécution, du code parallèle distribué.

Référence

[1] Dounia Khaldi, Parallélisation automatique et statique de tâches sous contraintes de ressources, thèse à l'École Nationale Supérieure des Mines de Paris, 2013.