Parallélisation de méthode d'optimisation entière sur GPU

Didier El Baz[1], Loic Dumas[1,2], Vincent Boyer[1], Moussa Elkihel[1], Jean-Michel Enjalbert[1]

[1] CNRS ; LAAS ; 7, avenue du Colonel Roche, F-31077 Toulouse, France. Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse France
{elbaz,ldumas,vboyer,elkihel,enjalbert}@laas.fr

[2] INSA de Toulouse ; 135 Avenue de Rangueil, F-31077 Toulouse, France Université de Toulouse ; UPS, INSA, INP, ISAE
ldumas@etud.insa-toulouse.fr

Mots-Clés : *programmation entière, calcul parallèle, GPGPU.*

1 Introduction

L'informatique a connu une évolution importante ces dernières années avec la généralisation du concept de parallélisme. Le parallélisme s'est imposé au niveau des architectures de processeurs, avec notamment les processeurs multi cœurs, comme au niveau des architectures de machines avec les clusters, supercalculateurs ou les réseaux de calcul pair à pair. Les fondeurs de cartes graphiques ont aussi pensé à utiliser leurs réalisations à des fins de calcul généraliste ; on parle alors de ¨General Purpose Graphic Processing Unit¨ ou GPGPU. De nouveaux outils comme CUDA ou Compute Unified Device Architecture ont été conçus pour utiliser le potentiel des cartes graphiques NVIDIA à des fins de calcul (cf [1]). Dans cet article nous présentons notre contribution à la parallélisation sur GPU de méthodes comme le Branch and Bound pour la résolution de problèmes de programmation entière. A notre connaissance aucune étude n'a été faite sur ce sujet. Un des défis à relever ici réside dans la structure de données irrégulière qui résulte de la nature des problèmes de programmation entière considérés comme par exemple les problèmes de sac à dos et des méthodes de résolution utilisées comme le Branch and Bound. Un autre intérêt de cette étude réside dans la spécificité du ¨GPU computing¨ qui s'eloigne des paradigmes classiques de programmation parallèle que l'on retrouve par exemple dans [2], [3] et [4].

2 Algorithme parallèle sur GPU

Dans cette première étude des méthodes de Branch and Bound parallèles ont été mises en œuvre sur GPU. Nous avons utilisé une carte GeForce GTX 260 (la carte GeForce GTX 260 comporte 192 cœurs).

Dans CUDA les threads du GPU sont regroupés en blocs. Le principe de l'algorithme parallèle consiste à faire séparer un nœud par chaque thread du GPU après classement par le CPU des nœuds

par borne supérieure décroissante et transfert des n meilleurs nœuds vers le GPU. Chaque thread transfère ensuite les nœuds fils créés vers le CPU.

Nous nous sommes concentrés sur le réglage du nombre de blocs et de threads afin de minimiser le temps de calcul parallèle. Une des difficultés soulevées par ce type d'algorithme parallèle réside dans la structure de donnée irrégulière. En effet, le nombre de nœuds n'est pas constant lors du déroulement de l'application, de sorte que les outils fournis par CUDA tels que le CUDA Occupancy calculator ne donnent pas d'information significative sur le choix du meilleur nombre de blocs.

Les premiers résultats sur GPU obtenus pour un problème de sac à dos fortement corrélé avec 96 variables sont présentés dans les tableaux ci-dessous (des expérimentations ont aussi été effectuées pour des problèmes fortement corrélés avec un nombre de variables plus important). Ces tableaux mettent en évidence qu'il existe une configuration optimale en terme de temps de calcul.

Blocs	1	4	8	16	24	32
Threads par bloc	16	16	16	16	16	16
Sommets envoyés à la GPU	16	16	16	16	16	16
Temps de calcul (s.)	71,88	7,34	1,33	1,16	0,97	1,92

TAB. 1 – temps moyen de calcul pour 30 instances fortement corrélées et 96 variables (cas avec 16 threads).

Blocs	24	24	24	24	24
Threads par bloc	16	32	64	128	256
Sommets envoyés à la GPU	16	16	16	16	16
Temps de calcul (s.)	0,97	1,07	1,07	1,42	2,48

TAB. 2 – temps moyen de calcul pour 30 instances fortement corrélées et 96 variables (cas avec 24 blocs).

Nous comptons poursuivre cette étude pour divers types d'instances et étudier la parallélisation sur GPU d'autres méthodes de résolution comme la programmation dynamique.

Références

[1] NVIDIA TESLA GPU Computing Solutions Solve the World's most Important Computing Challenges. CD NVIDIA, NVIDIA Corporation, Mai 2009.

[2] M. Elkihel, F. Viader. A parallel best-first branch and bound algorithm for 0-1 problems Single pool and multiple pool implementations. *Rapport LAAS No96103*. Parallel Optimization Colloquium (POC'96), Versailles (France), 25-27 Mars 1996.

[3] D. El Baz, M. Elkihel. Load balancing methods and parallel dynamic programming algorithms using dominance techniques applied to the 0-1 knapsack problem. *Journal of Parallel and Distributed Computing*, 65:74–84, 2005.

[4] M. Mezmaz, N. Melab, E-G. Talbi. An efficient load balancing strategy for Grid-based branch and bound algorithm. *Parallel Computing*, 33, No.4:302-313, 2007.