Programmation dynamique dense sur GPU

Vincent Boyer, Didier El Baz, Moussa Elkihel

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

Mots-Clés : *programmation dynamique dense, knapsack problem, calcul parallèle, GPGPU.*

1 Introduction

Depuis quelques années, les fondeurs de cartes graphiques développent des outils permettant d'utiliser leurs réalisations à des fins de calcul généraliste ; on parle alors de ¨General Purpose Graphic Processing Unit¨ ou GPGPU.

Dans cet article nous présentons une parallélisation de la programmation dynamique dense (cf. [1], [2], [3], [4]) sur GPU NVIDIA. Cette approche est appliquée à la résolution du problème du sac à dos en variables 0-1 qui est un problème d'optimisation combinatoire NP-complet (cf. [5]). Nous avons utilisé l'environnement de programmation fourni par NVIDIA, CUDA ou Compute Unified Device Architecture, qui a été conçu pour utiliser leurs cartes graphiques à des fins de calcul (cf [6]).

2 Programmation dynamique dense pour le problème du sac à dos

Le problème du sac à dos, ou knapsack problem (KP), est défini comme suit :

$$max\{\sum_{i=1}^{n} p_i.x_i \mid \sum_{i=1}^{n} w_i.x_i \leq c \; et \; x_i \in \{0,1\}, \; \forall i \in \{1,...,n\}\},$$

où n représente le nombre de variables du problème, c la capacité du sac à dos, et x_i, $p_i > 0$ et $w_i > 0$, $i \in \{1,...,n\}$, sont, respectivement, la variable de décision, le profit et le poids associés à l'objet i. Toutes les données du problème sont des entiers et nous nous plaçons dans le cas non-trivial tel que $w_i \leq c$, $\forall i \in \{1,...,n\}$ et $\sum_{i=1}^{n} w_i > c$.

La programmation dynamique dense est une méthode efficace pour la résolution de problèmes difficiles. Elle est, en effet, insensible au fait que les données du problèmes soient corrélées ou non. C'est une méthode récursive de complexité temporelle $\mathcal{O}(n.c)$ où à chaque étape $k \in \{1,...,n\}$ on

construit un tableau T^k de taille $c+1$ tel que :

$$Pour\ j \in \{0,...,c\} : T_j^k = \begin{cases} max\{T_j^{(k-1)}, T_{j-w_k}^{(k-1)} + p_k\},\ si\ j - w_k \geq 0, \\ T_j^{(k-1)},\ sinon. \end{cases}$$
$$avec\ T^0 = \{0,0,...,0\}.$$

Ce type d'algorithme est adapté à la programmation sur GPU NVIDIA du fait de la connexité des adresses des données manipulées. Nous avons donc dans cette première étude associé à un thread le remplissage d'une case du tableau T^k. Au total, $c+1$ threads seront excécutés. Sous CUDA, les threads sont regroupés dans des blocs qui seront distribués sur les multiprocesseurs (1 multiprocesseur ne peut prendre en charge qu'un bloc à la fois). Les $c+1$ threads ont donc été classés par groupe de 512, ce qui correspond à la taille maximale d'un bloc.

3 Résultats

L'algorithme parallèle que nous avons développé a été testé sur une carte GTX260 (24 Multiprocesseurs soit 192 coeurs, 1,4GHz) et les temps de calculs obtenus ont été comparés à ceux obtenus par l'algorithme séquentiel sur CPU (Intel Xeon 3,0GHz). L'ensemble des résultats obtenus est présenté dans le tableau 1.

Nombre de variables	10000	20000	30000	40000	50000
GPU (parallèle)	3.06	11.97	26.57	47.43	105.93
CPU (séquentiel)	57.83	230.54	536.14	1225.52	2752.81
Accélération	18.90	19.26	20.18	25.83	25.98

TABLE 1 – Comparaison des temps moyens de calcul pour 10 instances fortement corrélées (en seconde).

On constate que les temps de calcul en parallèle sur GPU sont nettement inférieurs à ceux obtenus en séquentiel sur CPU (accélération d'environ 22). Ces résultats sont d'autant plus encourageants que la fréquence d'horloge du CPU est deux fois plus grande que celle du GPU. Nous envisageons de continuer cette étude en essayant d'exploiter au mieux les possibilités offertes par CUDA, avec notamment une meilleure gestion de la mémoire.

Références

[1] R. Bellman. Dynamic Programming. *Princeton University Press*. 1957.

[2] J. Casti, M. Richardson, R. Larson. Dynamic Programming and Parallel Computers. *J. Optimization Theory and Applications*, 12(4):423–438, 1973.

[3] G.B. Dantzig. Discrete Variable Extremum Problem. *Operations Research*, 5:266–277, 1957.

[4] 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.

[5] S. Martello, P. Toth. Knapsack Problems. *John Wiley & Sons*, 1990.

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