Parcours de grands graphes sur architecture hybride CPU/GPU

J. Loiseau, F. Alin, C. Jaillet, et M. Krajecki
julien.loiseau@etudiant.univ-reims.fr

Résumé. Le benchmark du GRAPH500 est un classement mondial des supercalculateurs selon le parcours BFS d'un très grand graphe. Actuellement les machines de type IBM BlueGene obtiennent les meilleurs résultats, avec un algorithme exploitant exclusivement des CPU. Les solutions utilisant uniquement des GPU ne se hissent qu'à la 26ème place du classement. Le but de cette étude est de proposer une méthode de résolution hybride utilisant conjointement le meilleur des deux mondes du CPU et du GPU.

1 Introduction

En se référant à l'algorithme proposé par Checconi et al. (2012), implémenté sur les machines IBM BlueGene/Q, et les méthodes exclusivement GPU proposées par Merrill et al. (2015), nous développons une méthode hybride de résolution. Notre approche vise à affecter aux CPU des tâches de calcul mais également de communication entre les nœuds de calcul et, pour profiter de leur architecture massivement parallèle, à dédier les GPU exclusivement à la résolution de tâches de calcul.

2 Méthode de résolution

L'un des principaux écueils rencontrés dans la manipulation de très grands graphes réside dans la difficulté de leur représentation en mémoire et de leur stockage. Les problèmes actuels du GRAPH500 correspondent à des graphes de 2^{42} sommets, soit environ 1125 To de données. La masse des données en jeu ne peut donc pas être accessible de façon partagée par tous les serveurs d'un cluster ou d'une grande machine de calcul. La méthode proposée pour l'architecture BlueGene/Q (Checconi et al., 2012) répartit le travail sur les différents nœuds de calcul, chacun des nœuds ne dispose que d'une partie du graphe ; on obtient alors une empreinte mémoire bien plus adaptée aux caractéristiques du matériel actuellement disponible.

Dans notre cas, une machine de calcul représentera un couple CPU-GPU. En notant m le nombre d'unités de calcul avec $m = l \times l$ et $l = 2^k$ ($k \in \mathbb{N}$), la matrice d'adjacence du graphe est découpée en m blocs de même taille, répartis sur lesdites machines ; chaque machine gère donc un sous-ensemble de sommets en entrée et en sortie. Ce découpage permet de concentrer les communications en ligne ou en colonne pour les machines concernées.

Après génération, le graphe est réparti sur l'ensemble de l'architecture de calcul. L'algorithme BFS se décompose en 5 étapes, répétées en boucle : l'expansion de la frontière actuelle,

le partage sur la ligne de processus, la recherche des prédécesseurs de chaque sommet exploré, le stockage en mémoire de l'arbre de parcours BFS, et enfin l'échange de la file de sortie générée par la ligne i avec la file d'entrée de la colonne j.

Les GPU interviennent dans les phases d'exploration et de recherche des prédécesseurs des sommets de chaque nœud considéré. Pour l'heure la répartition de charge CPU-GPU, nécessaire pour exploiter au mieux toute la puissance du calculateur, est réalisée de façon statique.

3 Expériences

Les expériences sont réalisées sur le supercalculateur ROMEO de l'Université de Reims Champagne-Ardenne. Il est constitué de 130 nœuds équipés de deux CPU E5-2650 avec 8 cœurs cadencés à 2,6GHz, disposant de 32Go de mémoire, ainsi que de deux GPU NVIDIA K20Xm avec 6Go de mémoire. Nous utilisons les couples CPU-GPU comme des machines indépendantes ; le réseau d'interconnexion est un FatTree sur InfiniBand.

La version actuelle propose une implémentation complète de la génération, de la répartition et du BFS. Nous avons mené des tests sur le cluster ROMEO en utilisant jusqu'à 64 machines pour la résolution.

4 Conclusion

À ce jour nous disposons d'un code hybride fonctionnel qui nous a permis de valider les points essentiels de la stratégie algorithmique retenue :

— Sur ROMEO, nous avons pu vérifier que les temps de transfert inter-nœuds restent très modérés grâce à l'architecture InfiniBand, et ne sont donc pas un facteur de limitation des performances ;

— Malgré le coût supplémentaire d'échange de données entre les mémoires du CPU (*host*) et du GPU (*device*), la représentation bitmap des données du problème (matrice d'adjacence par exemple), associée à l'architecture massivement parallèle des GPU, autorise des stratégies très efficaces de parcours du graphe. Par ailleurs, en dimensionnant intelligemment les threads GPU, on peut masquer intégralement les échanges *host*$\Leftrightarrow$ *device*.

La solution actuelle nous a également permis d'identifier de nombreuses pistes d'optimisation, nous donnant bon espoir d'obtenir les performances escomptées à savoir de l'ordre de 10-20 GTEPS (milliards d'arêtes traversées par seconde).

Références

Checconi, F., F. Petrini, J. Willcock, A. Lumsdaine, A. R. Choudhury, et Y. Sabharwal (2012). Breaking the speed and scalability barriers for graph exploration on distributed-memory machines. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–12. IEEE.

Merrill, D., M. Garland, et A. Grimshaw (2015). High-performance and scalable gpu graph traversal. *ACM Transactions on Parallel Computing 1*(2), 14.