Visualisation radiale : approche parallèle entre CPU et GPU

Tianyang Liu*, Fatma Bouali**,*, Gilles Venturini*

* Université François-Rabelais de Tours, Laboratoire d'Informatique
64 avenue Jean Portalis, 37200 Tours, France, venturini@univ-tours.fr
** Université de Lille2, IUT, Dpt STID
25-27 Rue du Maréchal Foch, 59100 Roubaix, France, fatma.bouali@univ-lille2.fr

Résumé. Dans cet article, nous proposons une parallélisation sur CPU et GPU d'une méthode de visualisation radiale à base de points d'intérêt. Nous montrons que cette approche peut visualiser avec des temps très courts des millions de données sur des dizaines de dimensions, et nous étudions l'efficacité de la parallélisation dans différentes configurations.

1 Introduction

Traiter des données de plus en plus volumineuses fait partie des défis que se donne la communauté des chercheurs en fouille de données. Une des manières de répondre à ce défi est de définir des méthodes de fouille de données mettant en oeuvre le parallélisme, sur une ou plusieurs machines, et en utilisant le processeur (CPU) et/ou le calculateur graphique (GPU). Un certain nombre d'algorithmes ont déjà donné lieu à parallélisation sur GPU à la fois en fouille de données (voir un aperçu dans (Jian et al., 2011)), en visualisation scientifique (Weiskopf, 2006) ou en visualisation d'informations et plus spécifiquement en affichage de graphes (Frishman et Tal, 2007).

Dans le domaine de la fouille visuelle de données, les approches pouvant afficher le plus de données relèvent soit des approches orientées pixels et assimilées (voir (Keim et al., 1995) où 530.000 valeurs sont visualisées), soit des approches utilisant des densités (voir (Fua et al., 1999) où les coordonnées parallèles peuvent afficher jusqu'à 200.000 données). Cependant on peut constater que les interactions sont le point d'achoppement de ces méthodes visuelles pour le traitement de grands volumes de données (voir discussion dans (Florek, 2006)). Lorsque l'utilisateur formule une requête graphique, le système doit bien souvent faire des calculs supplémentaires et réafficher les données, partiellement ou entièrement, ce qui peut ralentir l'utilisateur dans son processus d'exploration (et rendre la méthode inutilisable).

Des approches de visualisation ont été étudiées pour remédier à ces inconvénients, comme pour le MDS (Ingram et al., 2009) en abaissant à la fois la complexité algorithmique et en parallélisant la méthode sur GPU. Cependant cette complexité est encore importante si l'on veut traiter des millions de données et il s'agit d'une approche peu interactive (sans redéfinition de l'espace de visualisation). Dans (Florek, 2006), l'objectif a été cette fois d'obtenir des temps d'affichage permettant les interactions. Le facteur d'accélération entre l'implémentation sur CPU (non parallélisée) et l'implémentation sur GPU va jusqu'à 60 (150.000 données en dimension 4 sont affichées en 0.3s sur GPU au lieu de 20s sur CPU).

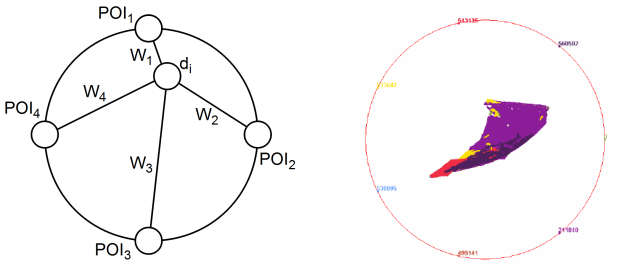

FIG. 1 – *Principe de la méthode d'affichage radial à base de points d'intérêt (à gauche) : chaque donnée est positionnée en fonction de sa similarité avec les points d'intérêt qui sont disposés autour du cercle. A droite nous donnons un exemple d'affichage sur la base ForestCovertype (581012 données et 54 attributs) issu de (Da Costa et Venturini, 2006) où les classes sont indiquées par des couleurs.*

Etape	Fréquence	Complexité séq.	Calcul sur	Parallélisme
E1 Lecture données	1	$\Theta(n * m)$	CPU	Oui
E2 Définition POIs	1	$\Theta(1)$	CPU	Non
E3 Calc. simil. données-POIs	1+$\Theta(I)$	$\Theta(n * m * k)$	GPU	Oui
E4 Affichage données	1+$O(I)$	$\Theta(n)$	CPU	Oui
E5 Interactions	$O(I)$	Retour E3 ou E4		

TAB. 1 – *Complexité et parallélisation choisie pour chacune des étapes de notre méthode en fonction du nombre de données n, de la dimension des données m, du nombre de POIs k et du nombre d'interactions réalisées I au cours d'une session utilisateur.*

Notre objectif consiste donc à proposer une approche de fouille visuelle de données qui 1) puisse visualiser des volumes de données multidimensionnelles dépassant les limites actuelles des approches existantes, 2) permette à l'utilisateur d'avoir des interactions graphiques les plus rapides possible. Ces objectifs passent par le choix d'une méthode visuelle ayant une complexité faible et pouvant se paralléliser sur CPU et GPU. Nous avons donc choisi comme visualisation candidate une méthode répondant le mieux possible à ces critères (Da Costa et Venturini, 2006). Elle fait partie des méthodes radiales comme RadViz (Hoffman et al., 1999). Ces approches considèrent que des points d'intérêt (POIs, appelés aussi "ancres dimensionnelles"), sont disposés sur un cercle par exemple, et que les données viennent se positionner à l'intérieur du cercle en fonction de leur ressemblance avec les POIs (voir figure 1). Ainsi, les POIs peuvent représenter des cas particuliers importants parmi les données, ou des hypothèses à tester. Chaque choix ou configuration des POIs vient donner une disposition particulière des données. L'utilisateur dispose de plusieurs interactions afin de modifier cette disposition et faire apparaître de nouvelles informations.

2 Parallélisation d'une méthode radiale : entre CPU et GPU

Si l'on note $d_1, ..., d_n$ les n données multidimensionnelles décrites selon m attributs, et $POI_1, ..., POI_k$ les k points d'intérêt choisis, une session utilisateur se déroule avec notre visualisation selon 5 étapes décrites dans la première colonne de la table 1. La parallélisation des différentes étapes de cet algorithme est résumée dans ce même tableau 1. La lecture des données (étape E1), effectuée une seule fois, est traitée par le CPU avec plusieurs tâches de lecture en parallèle.

Le choix des POIs (étape E2) est effectué de manière automatique pour la première visualisation, ensuite c'est l'utilisateur qui ajuste les points d'intérêt grâce aux interactions. La méthode automatique est appelée une seule fois, et son exécution reste peu coûteuse car elle porte sur un échantillon de données de taille fixe. Cet algorithme est donc exécuté sur le CPU, sans parallélisation particulière.

Le calcul des similarités entre les données et les POIs (étape E3) est l'opération la plus coûteuse puisque l'on doit calculer $\Theta(n * k)$ distances en dimension m. Cette multitude de calculs numériques se prête bien à une parallélisation sur le GPU. La tâche générique traitée par un coeur du GPU porte sur plusieurs données, définies comme un sous-ensemble de $\{d_1, ..., d_n\}$, dont on va calculer la similarité avec les POIs ainsi que les coordonnées d'affichage. Chaque paquet est traité par une tâche concurrente. En CUDA, les tâches sont regroupées en plusieurs blocs qui forment une grille. Le résultat du traitement d'un bloc est stocké temporairement dans la mémoire du GPU. Une fois tous les blocs traités, le résultat final (coordonnées d'affichage des données) est recopié dans la mémoire principale et peut ensuite être utilisé par les autres étapes de notre méthode sur le CPU. L'utilisation des différents types de mémoire est un point clé d'un programme CUDA, car ces mémoires ont des rôles et des performances très différents. Principalement, notre programme a été optimisé afin de ne lire qu'une seule fois les données, et nous avons choisi de stocker les informations nécessaires aux calculs (POIs, leurs coordonnées, etc) dans la mémoire constante du GPU qui est très rapide (mais limitée à 64ko).

Dans l'objectif d'effectuer des comparaisons, cette étape E3 a également été parallélisée sur le CPU. Pour cela nous avons utilisé le même principe global : les données sont découpées en paquet, et chaque paquet est traité par des tâches concurrentes sur le CPU (par exemple, de 1 à 12 tâches).

L'affichage des données (étape E4) a un coût linéaire en n, et dans notre implémentation actuelle, nous avons réalisé cette étape de manière concurrente sur le CPU. En ce qui concerne les interactions, un pré-traitement a souvent lieu avant de renvoyer l'exécution vers une des étapes précédentes de l'algorithme (E3 ou E4). Le seul pré-traitement ayant un coût de calcul est lié à la sélection, puisqu'il faut détecter quelles données appartiennent au cadre dessiné par l'utilisateur. Ce calcul est principalement proportionnel au nombre de pixels sélectionnés, et reste dans un temps acceptable sur le CPU.

3 Résultats

3.1 Bases, matériel, paramétrage

Les bases utilisées pour nos tests sont représentées dans la table 2. Nous avons conçu un générateur de données qui génère n données en dimensions m selon 5 classes de même effectif.

Bases	n	d	Taille fichier	Origine
1M10	1 10^6	10	99,6 Mo	Artificielle
7M10	7 10^6	10	698,5 Mo	Artificielle
14M10	14 10^6	10	1395,7 Mo	Artificielle
1M130	1 10^6	130	1265,1 Mo	Artificielle
Forest Cover Type (FCT)	581012	54	71,6 Mo	UCI
Pocker Hand (PH)	1 10^6	10	23,4 Mo	UCI
YearPredictionMSD (YPMSD)	515345	90	438,1Mo	UCI

TAB. 2 – *Un résumé des différentes bases testées. La notation "nMm" indique que le jeu de données artificielles correspondant possède n millions de données en dimension m. "UCI" signifie que la base vient du Machine Learning Repository.*

De cette manière, nous avons pu tester l'évolution de la complexité des opérations en fonction de n, m et k. Le matériel utilisé dans ces tests est courant : un portable Asus avec un i7 cadencé à 2,2GHz, un GPU Nvidia GTX560M (192 coeurs). Les résultats présentés dans la suite ont été calculés en moyenne sur 10 essais. Les temps sont exprimés en ms.

Pour les étapes utilisant le CPU (E1 et E4, mais aussi une version CPU de l'étape E3 utilisée dans nos comparaisons CPU-GPU), nous avons déterminé le meilleur nombre de tâches concurrentes à utiliser en utlisant de nombreux jeux de données (y compris ceux de la table 2) et en faisant croître le nombre de tâches concurrentes de 1 à 12. Nous avons mesuré le temps mis par le CPU pour accomplir les travaux demandés. Par manque de place, nous ne donnons que les conclusions obtenues : à l'issue de ces tests, nous avons conclu que les meilleurs paramètres pour les étapes ayant lieu sur CPU sont 4 tâches pour E1, 8 pour E3 et 4 pour E4 (paramètres utilisés dans le reste de l'article).

Pour le GPU, le paramétrage consiste à déterminer deux valeurs : le nombre de blocs B et le nombre de tâches par bloc T. L'efficacité de ce découpage dépend du matériel. Nous avons sélectionné plusieurs bases de données et nous avons fait varier ces deux paramètres. Dans la suite, nous avons sélectionné les valeurs $B = 256$ et $T = 128$ pour les résultats de l'approche GPU.

3.2 Performances de la méthode et apports du parallélisme

Nous avons mesuré les temps mis par notre méthode et ses variantes pour traiter, étape par étape, différents jeux de données (voir tableau 3). Les deux premières colonnes du tableau montrent que le temps de lecture des fichiers au format CSV est globalement divisé par 2 grâce à la parallélisation. La colonne E2-CPU1 donne les temps nécessaires pour le choix initial des POIs. Ici il s'agit d'un algorithme heuristique qui échantillonne des points au hasard et choisit le groupe de points qui maximise une mesure donnée. Le nombre d'itérations de cet algorithme est suffisant, d'après nos expériences précédentes, pour obtenir un résultat satisfaisant. Cette étape E2 s'exécute donc en un temps négligeable par rapport aux autres étapes. C'est la raison pour laquelle nous n'avons pas réalisé sa parallélisation (qui serait possible cependant).

Les colonnes E3-CPU1, E3-CPU8 et E3-GPU permettent de comparer entre elles des approches utilisant un parallélisme d'intensité croissante pour le calcul des similarités. Les résul-

Bases	E1-CPU1	E1-CPU4	E2-CPU1	E3-CPU1	E3-CPU8	E3-GPU	E4-CPU1	E4-CPU4
1M10	4229 [42]	2539 [24]	25 [0]	2136 [35]	560 [11]	54 [0]	483 [9]	342 [3]
7M10	30121 [412]	17465 [126]	25 [0]	14468 [65]	3532 [47]	324 [2]	4098 [39]	3226 [14]
14M10	60573 [918]	34807 [476]	25 [0]	29262 [151]	7832 [122]	738 [14]	8335 [230]	6165 [157]
1M130	39593 [247]	19308 [172]	70 [0]	11913 [62]	5040 [15]	321 [8]	403 [11]	357 [4]
FCT	6750 [52]	3664 [21]	45 [0]	5766 [35]	1865 [12]	135 [0]	233 [7]	168 [3]
PH	2842 [15]	1808 [13]	25 [0]	2055 [12]	520 [5]	85 [0]	436 [10]	374 [19]
YPMSD	15504 [111]	7042 [31]	64 [0]	7266 [175]	1806 [8]	123 [0]	165 [1]	223 [6]

TAB. 3 – *Temps de calcul de différentes approches, étape par étape, et pour différentes bases. Les valeurs en ms sont calculées en moyenne sur 10 essais avec écarts-types entre crochets. CPU1 signifie par exemple que le CPU est utilisé avec une seule tâche.*

tats montrent tout d'abord que le gain entre l'approche séquentielle CPU1 et parallèle CPU8 est déjà important (de 2 à 4). Ensuite, nous remarquons que le gain entre CPU8 et GPU est compris entre 10 et 12. Le gain entre une approche totalement séquentielle (CPU1) et une approche parallèle à grain fin (GPU) est compris entre 24 et 60.

Les deux dernières colonnes (E4-CPU1 et E4-CPU4) permettent de quantifier l'efficacité de la parallélisation de l'affichage sur CPU. La différence entre les colonnes permet de gagner quelques secondes sur les plus grosses bases, cependant, elle n'est pas aussi importante que prévue et doit pouvoir être améliorée.

4 Conclusions

Nous avons présenté dans cet article une parallélisation d'une méthode visuelle radiale afin de rendre possible le traitement de millions de données, notamment du point de vue de la vitesse d'affichage et du temps nécessaire pour les interactions. Cette parallélisation s'est appuyée à la fois sur le CPU et le GPU, en fonction des complexités a priori des opérations traitées.

Le principal résultat de l'article est de montrer que cette approche permet des affichages et des interactions dont les temps vont, pour des millions de données, de 0,2 à 10 secondes, tout en utilisant du matériel classique. Ce résultat montre que les approches radiales représentent une catégorie de méthodes se prêtant bien au passage à l'échelle à condition d'utiliser des algorithmes parallèles. Nos résultats permettent aussi d'augmenter de manière significa-

tive le volume de données multidimensionnelles manipulables par des approches visuelles et interactives.

Références

Da Costa, D. et G. Venturini (2006). An interactive visualization environment for data exploration using points of interest. In X. Li, O. R. Zaïane, et Z. Li (Eds.), *ADMA*, Volume 4093 of *Lecture Notes in Computer Science*, pp. 416–423. Springer.

Florek, M. (2006). Using modern hardware for interactive information visualization of large data. *Master's thesis, Comenius University, Bratislava*.

Frishman, Y. et A. Tal (2007). Multi-level graph layout on the gpu. *IEEE Transactions on Visualization and Computer Graphics 13*(6), 1310–1319.

Fua, Y.-H., M. O. Ward, et E. A. Rundensteiner (1999). Hierarchical parallel coordinates for exploration of large datasets. In *Proceedings of the conference on Visualization '99 : celebrating ten years*, VIS '99, pp. 43–50. IEEE Computer Society Press.

Hoffman, P., G. G. Grinstein, et D. Pinkney (1999). Dimensional anchors : A graphic primitive for multidimensional multivariate information visualizations. In *Workshop on New Paradigms in Information Visualization and Manipulation'99*, pp. 9–16.

Ingram, S., T. Munzner, et M. Olano (2009). Glimmer : Multilevel mds on the gpu. *IEEE Transactions on Visualization and Computer Graphics 15*, 249–261.

Jian, L., C. Wang, Y. Liu, S. Liang, W. Yi, et Y. Shi (2011). Parallel data mining techniques on graphics processing unit with compute unified device architecture (cuda). *The Journal of Supercomputing*, 1–26. 10.1007/s11227-011-0672-7.

Keim, D. A., M. Ankerst, et H. P. Kriegel (1995). Recursive Pattern : A Technique for Visualizing Very Large Amounts of Data. *Proceedings of the 6th conference on Visualization'95*, 279–286.

Weiskopf, D. (2006). *GPU-Based Interactive Visualization Techniques (Mathematics and Visualization)*. Secaucus, NJ, USA : Springer-Verlag New York, Inc.

Summary

In this paper, we propose to parallelize, on CPU and GPU, a radial-based visualization method that uses points of interests. We show that this approach may visualize in a few seconds millions of data with tens of dimensions, and we study the efficiency of the parallel approach in different configurations.