Ecole Polytechnique

Rapport Final

INF560 - Calcul Parallèle et Distribuée

COELHO LECHNER, Carlos
LOBATO GIMENES, Tiago

Palaiseau, France
Hiver 2014-2015

Table des matières

Table des figures

1 Introduction

Ce rapport provient de nos essais de étudier le problème k-NN(de l'anglais : "*k- nearest neighbors*", en français "*k-plus proches voisins*") dans le cadre du calcul parallèle. Pour cela on a utilisé le langage de programmation CUDA pour les cartes graphiques *NVidia*. Ce problème a plusieurs approches possibles, on en a essayé quelques unes pertinentes au contexte du parallélisme. On montre ici notre approche particulière au problème et on espère que vous trouvez ce problème et les algorithmes utilisés aussi intéressants et amusants que nous les avons trouvé.

2 Préliminaires

La méthode des k plus proches voisins est une méthode d'*apprentissage supervisé*.

Dans ce cadre, on dispose d'une base de données d'apprentissage constituée de N couples «entrée-sortie». Pour estimer la sortie associée à une nouvelle entrée x, la méthode consiste à prendre en compte (de façon identique) les k échantillons d'apprentissage dont l'entrée est la plus proche de la nouvelle entrée x, selon une certaine distance.

Par exemple, dans un problème de classification, on retiendra la classe la plus représentée parmi les k sorties associées aux k entrées les plus proches de la nouvelle entrée x.

Ici on regarde le pas d'identification des des k-plus proches voisins d'un point donné et comme on peut optimiser cette étape en utilisant le parallélisme.

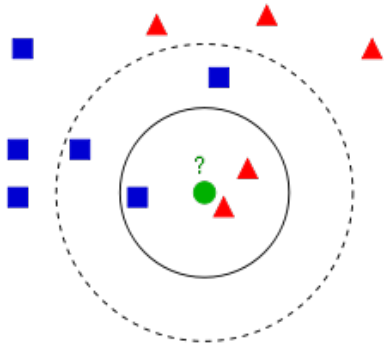

FIGURE 2.1 – Représentation du 3-KNN.

3 Première Solution : Solution CPU - Quicksort

À titre de comparaison on a fait une solution simple dans la CPU. La solution utilise le sort `std::sort` de la bibliothèque `<algorithm>` de C++. Il s'agit d'un tri bien optimisé dont le temps d'exécution attendu(en moyenne) est $O(n.log(n))$. Même si l'implémentation

n'est pas spécifiée et varie avec la distribution on peut imaginer pour ce rapport qu'il s'agit d'un quicksort(en français, *tri rapide*).

4 Deuxième Solution : Parallélisme Naïf

Notre idée ici était d'utiliser un algorithme parallèle non-optimal tout simplement pour faire une comparaison avec la méthode CPU et pouvoir voir le gain immédiat obtenu par l'utilisation de la carte graphique. Notre première idée était d'implémenter le tri pair-impair mais vu que nous l'avons déjà fait en cours(en Java) nous avons choisi un algorithme différent, le *bitonic sort*(en français, *tri bitonique*). Cet algorithme a un temps d'exécution de $O(n.log^2(n))$ mais avec un délai de $O(log^2(n))$ (temps parallèle).

L'astuce du tri bitonique se trouve sur la première figure, extraite de [2].

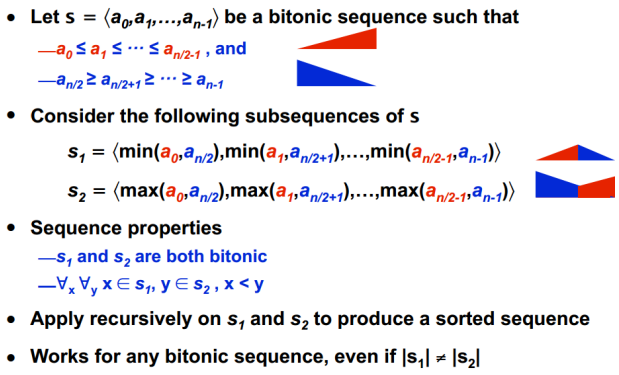

FIGURE 4.2 – Représentation de la logique derrière le tri bitonique.

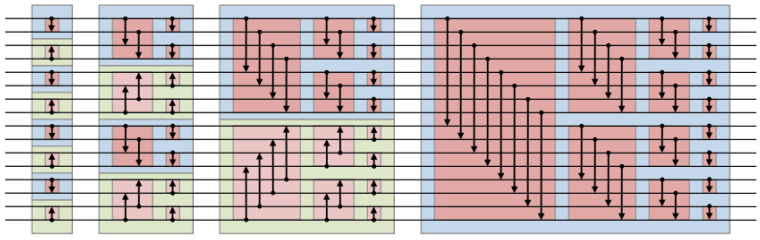

FIGURE 4.3 – Réseau de tri du tri bitonique.

5 Troisième Solution : Parallélisme Efficace

Pour faire une comparaison plus sérieuse on utilise un algorithme bien optimisé pour la GPU : le sort de la bibliothèque `thrust` de CUDA. On a observé que le gain de performance a été vraiment surprenant dans ce cas.

Pour voir la comparaison entre les 3 méthodes on a fait le graphique suivant :

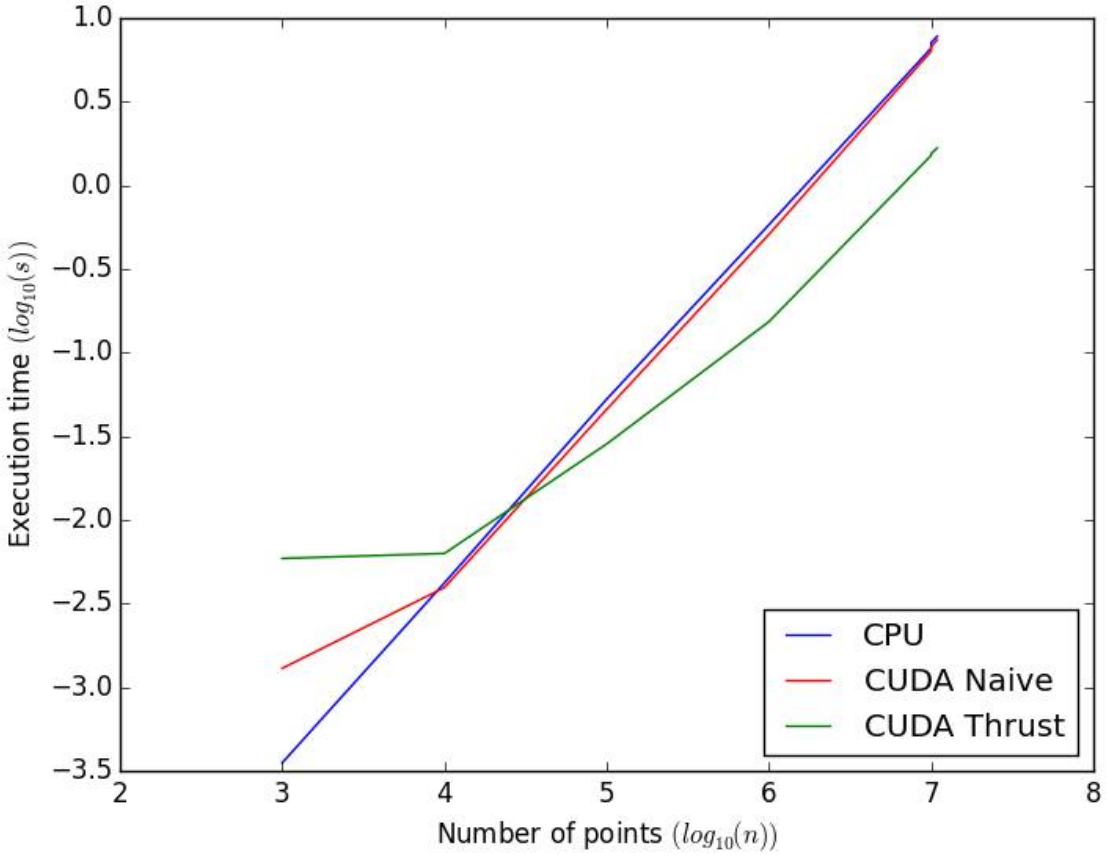

FIGURE 5.4 – Comparaison de temps d'execution sur un *Core i7 4700MQ - GTX850*.

6 Conclusion

En analysant les résultats, on peut voir que le but initial de solidifier les connaissances du calcul parallèle a été atteint. On a pu avoir le goût du pouvoir de la programmation en GPU.

C'était, surtout, une leçon importante pour nous, aspirants à devenir des ingénieurs logiciels, d'avoir acquis un outil de résolution de problèmes très puissant et très présent dans

le monde de l'informatique d'aujourd'hui comme celui de la programmation parallèle. Cela nous offre une nouvelle façon de penser et, donc, ça nous ouvre le champ des possibilités pour l'avenir.

7 Code

Les codes complets de nos programmes peuvent être trouvés sur le repositoire GitHub ci-dessous :

`https://github.com/tlgimenes/CUDAkNN`

8 Bibliographic References

Références

[1] Michalis Vazirgiannis, *Notes de cours de "INF582 - BigData : Data Mining* 2015 : École Polytechnique.

[2] Sesh Venugopal, *Course Notes : Bitonic Sort* 2012 : Rutgers University `http://www.cs.rutgers.edu/~venugopa/parallel_summer2012/bitonic_overview.html`

[3] H.W. Lang, *Course Notes : Bitonic Sort and Sorting Networks* 2010 : Fachhochschule Flensburg. `http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonicen.htm` `http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonicen.htm`

[4] E. Goubault, S. Putot, *INF560 - Calcul Parallèle et Distribué* 2015 : École Polytechnique. `http://www.enseignement.polytechnique.fr/profs/informatique/Sylvie.Putot/INF560/index.html`