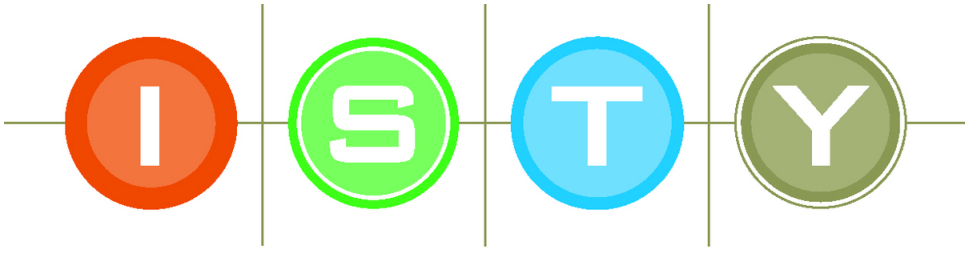
Tri parallèle d'un tableau avec OpenMP

Pierre AYOUB – Océane FLAMANT

10 novembre 2018



INSTITUT DES SCIENCES ET TECHNIQUES DES YVELINES

Table des matières

1 Introduction

Durant la réalisation de ce projet, nous avions comme objectifs, dans un premier temps, d'implémenter en langage C un algorithme de tri parallèle destiné à trier une grande base de données. Dans un second temps, il nous fallait effectuer des mesures du temps d'exécution de notre programme en fonction de plusieurs paramètres, tels que le nombre de threads utilisés ou encore la taille des données.

Plusieurs contraintes nous étaient imposées :

— Le squelette de l'algorithme à suivre était donné dans l'énoncé ;

— Nous avions à utiliser la bibliothèque de programmation parallèle OpenMP pour paralléliser notre algorithme ;

— Nous avions trois fonctions principales à implémenter de la manière suivante :

generator() : remplit un tableau de taille K avec des nombres réels aléatoires ;

tri() : tri un tableau de taille K ;

tri-merge() : à partir de deux tableaux chacun de taille K et trié, renvoie deux tableaux triés vérifiant que toutes les valeurs du premier tableau sont inférieures à celles du deuxième.

2 Développement du programme

Dans cette partie nous allons vous expliquer les choix que nous avons fait pour implémenter l'algorithme demandé ainsi que les modifications qui ont dû être effectuées.

2.1 Structure et choix

La structure globale suit l'algorithme défini dans l'énoncé mais nous allons vous détailler, dans cette partie, certaines partie du programme.

Avant d'implémenter les fonctions, nous avons de créer des macro-fonction qui permettent de calculer le minimum et le maximum dans le but d'optimiser et de faciliter la compréhension du code. Nous avons aussi mis les tailles des données en static car elles ne doivent pas être modifiées au cours de l'exécution du programme.

Pour l'algorithme de tri nous avons choisi le tri à bulles car même si ce n'est pas le plus rapide que l'on connaît aujourd'hui, il est simple à comprendre et à implémenter et il n'a pas besoin de tableau annexe pour que le tri s'effectue. De plus nous avons choisi de nous concentrer sur le parallélisme du programme et la différence de son temps d'exécution suivant les différents critères, non pas sur le temps d'exécution global.

2.2 Modification de l'algorithme

Nous avons rencontré quelques déconvenues avec l'algorithme donné.

En effet une fois nos fonctions réalisées et vérifiées nous avons lancé le programme et le résultat obtenu n'était pas le bon. Les valeurs contenues dans les tableaux étaient triés mais pour les tableaux entre cela n'était pas le cas. Nous avons donc dû effectuer de léger changement.

Tout d'abord, chaque boucle for commence à 0 et s'arrête à N. Ensuite nous avons supprimé les +1 des variables k, b1 et b2 car les boucles commencent à 0.Et enfin pour b2, le K est devenu k.

En résumé, k=(j%2) ; b1=1+(k+2*i)%N -> b1=(k+2*i)%N ; b2=1+(K+2*i+1)%N -> b2=(k+2*i+1)%N.

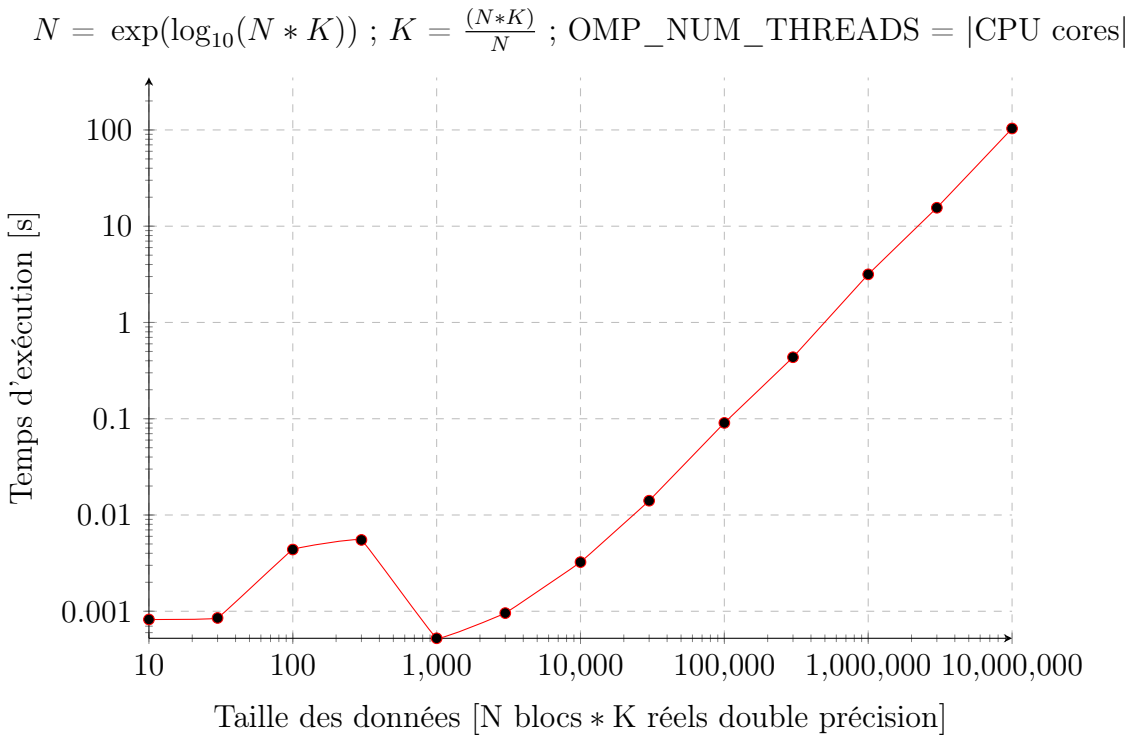


FIGURE 1 – Temps d'exécution en fonction de la taille des données

3 Performances

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

3.1 Mesures

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

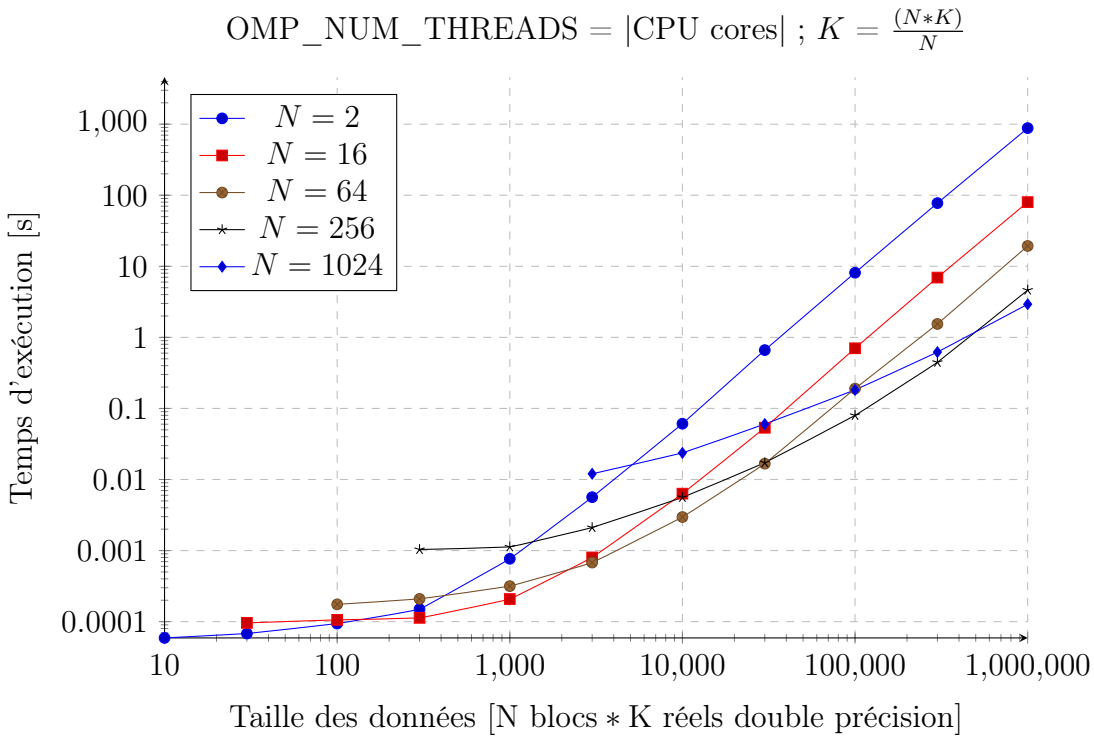


FIGURE 2 – Temps d'exécution en fonction de la taille des données et du nombre de blocs

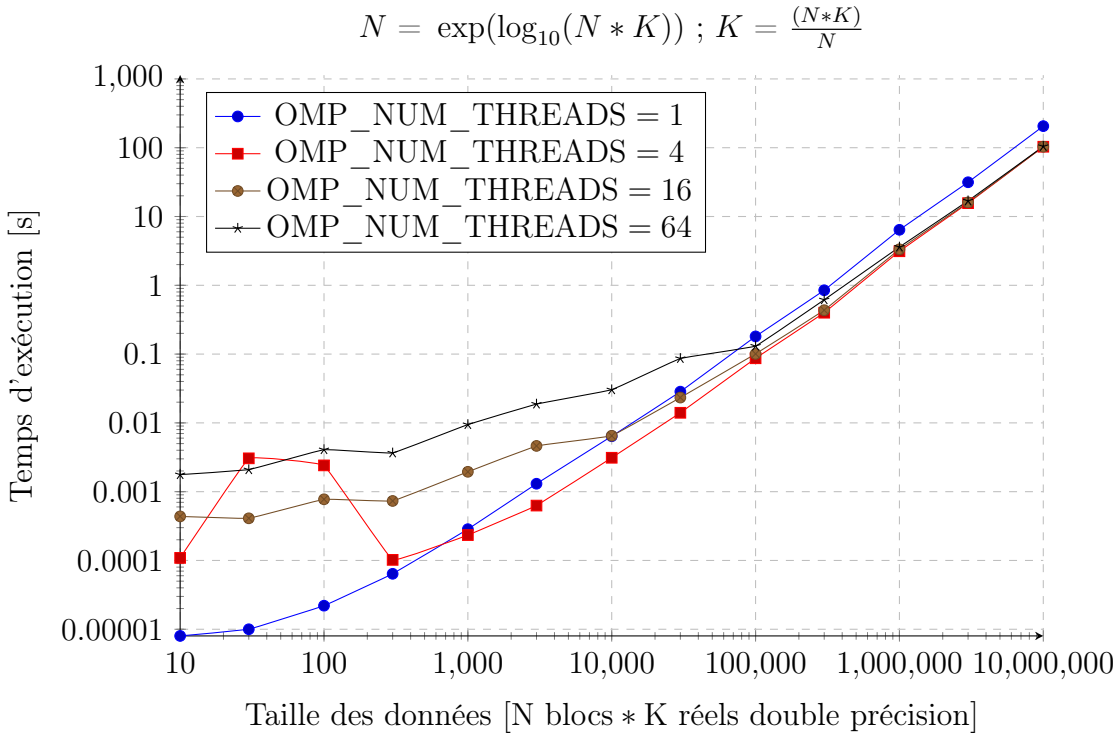


FIGURE 3 – Temps d'exécution en fonction de la taille des données et du nombre de threads

3.2 Perspectives d'amélioration

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

4 Conclusion

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.