Parallélisation de l'algorithme Bucket Sort en MPI

February 17, 2025

1 Introduction

L'objectif de ce travail est d'implémenter la parallélisation de l'algorithme de tri Bucket Sort en utilisant deux versions distinctes avec MPI. Les deux versions suivent la méthodologie établie :

- Le processus 0 génère un ensemble de nombres aléatoires.
- Les données sont distribuées aux autres processus.
- Chaque processus trie ses données en parallèle.
- Enfin, les données triées sont rassemblées par le processus 0.

L'objectif est de comparer les deux implémentations et de déterminer laquelle offre les meilleures performances et un meilleur équilibrage de charge.

2 Description des Implémentations

2.1 Implémentation avec des Intervalles Équidistants

Cette version divise l'intervalle des données en sous-intervalles équidistants entre la valeur minimale et maximale globale. Tout d'abord, chaque processus détermine la valeur minimale et maximale dans ses données locales. Ensuite, ces valeurs sont combinées via MPI_Allreduc pour obtenir les valeurs minimales et maximales globales. Avec ces informations, les limites des buckets sont générées de manière équidistante en fonction du nombre de processus utilisés.

Chaque processus classe ses données locales dans les buckets correspondants et les distribue aux autres processus en utilisant MPI_Alltoallv. Enfin, chaque processus trie les données reçues et le processus 0 les assemble en un tableau trié unique.

Le principal inconvénient de cette méthode est que si les données ne sont pas uniformément réparties, certains processus peuvent recevoir une quantité de données beaucoup plus importante que d'autres, ce qui entraîne un déséquilibre de charge.

2.2 Implémentation avec des Quantiles

Dans cette seconde implémentation, les limites des buckets sont calculées en utilisant des quantiles au lieu d'intervalles équidistants. Tout d'abord, toutes les données sont collectées par le processus 0 via MPI_Gather. Ensuite, ce processus calcule les quantiles en utilisant la fonction numpy.quantile, garantissant que chaque bucket contient approximativement la même quantité de données.

Une fois les quantiles calculés, ils sont diffusés à tous les processus via MPI_Bcast. Chaque processus classe alors ses données locales dans les buckets selon les quantiles et les distribue avec MPI_Alltoallv. Comme dans l'implémentation précédente, chaque processus trie ses données avant de les envoyer au processus 0 pour la consolidation finale.

Cette méthode permet une meilleure répartition de la charge dans les scénarios où les données ne sont pas uniformes, en évitant qu'un seul processus ne se retrouve avec une charge disproportionnée.

3 Résultats et Analyse des Performances

Pour évaluer les performances des deux implémentations, des tests ont été réalisés avec un total de 10 000 000 de données en utilisant quatre processus. Les temps d'exécution de chaque version sont présentés ci-dessous.

3.1 Temps d'Exécution

Les temps d'exécution mesurés pour chaque processus dans la version avec des intervalles équidistants sont les suivants :

Processus	**Temps d'exécution (s)**
Rank 0	22.154230
Rank 1	23.183552
Rank 2	23.377218
Rank 3	22.141528

Table 1: Temps d'exécution pour la version avec intervalles équidistants

Les temps d'exécution mesurés pour la version avec quantiles sont les suivants :

Processus	**Temps d'exécution (s)**
Rank 0	19.900762
Rank 1	19.870299
Rank 2	19.824403
Rank 3	19.870936

Table 2: Temps d'exécution pour la version avec quantiles

3.2 Analyse des Performances

En comparant les deux implémentations, on observe que la version basée sur les quantiles présente un temps d'exécution inférieur à la version avec intervalles équidistants. De plus, les temps d'exécution de chaque processus sont plus homogènes dans la version avec quantiles, ce qui indique un meilleur équilibrage de charge entre les processus.

Dans la version avec intervalles équidistants, on note une différence de plus de 1 seconde entre le processus le plus rapide et le plus lent, ce qui suggère que certains processus reçoivent plus de données que d'autres. Cela est dû au fait que la distribution des données n'est pas parfaitement uniforme, ce qui entraîne un regroupement inégal dans certains buckets.

En revanche, dans la version avec quantiles, tous les processus terminent en environ le même temps, avec une différence maximale de seulement 0.08 seconde. Cela confirme que la division des données en fonction des quantiles permet une répartition plus équilibrée de la charge de travail, réduisant ainsi les temps d'exécution.

4 Conclusion

Les résultats montrent que la version basée sur les quantiles est plus efficace et équilibrée par rapport à la version avec intervalles équidistants. Bien que la seconde version implique une surcharge initiale due au calcul des quantiles, cette charge est compensée par une meilleure répartition des données, évitant qu'un seul processus ne soit surchargé.

En conclusion, dans des applications où les données peuvent être biaisées ou suivre une distribution inconnue, l'utilisation des quantiles est une meilleure approche pour optimiser l'efficacité du traitement parallèle.