Sorbonne Université

UE MU4IN903 - « Calcul Haute Performance »
Master Informatique - M1 SFPN

Compte Rendu de Projet

Calle Viera Andersson
El Dahrawi Hichem

Enseignant
BOUILLAGUET Charles

Table des matières

Introduction

1 MPI

Dans cette section nous allons présenter nos choix d'implantation de CG avec l'utilisation de MPI.

Fonctionnement

Un élément important à prendre en compte en parallélisation est la répartition du travail à effectuer pour chaque processus. Dans un premier temps, nous devons décider la manière avec laquelle la matrice A est envoyée aux processus. Pour cela, deux choix s'offrent à nous ; envoyer la matrice entièrement à tous les processus ou alors envoyer seulement la partie nécessaire de A à chacun d'eux.

Le premier semble plus simple à programmer certes mais la quantité de mémoire utilisée est conséquente surtout si la dimension de la matrice est très grande ; d'autant plus que la transmission de celle-ci à tous les processus prendrait du temps et nous voudrions plutôt en gagner !

Le second, que nous avons choisi, permet de ne transmettre seulement qu'une partie de la matrice à chaque processus, ce qui pâlie les inconvénients cités précédemment mais qui augmentent par contre le nombre de communications.

La distribution de la matrice

Nous appelons `ROOT`, le processus de rang zéro. C'est lui qui calcule la taille des blocs et les envoie aux processus. P_i correspond au processus i.

Équilibrage de charge

La version que nous proposons divise la matrice A par ligne. Ainsi si *rows* est le nombre de ligne de A et *nproc* le nombre de processus, chacun d'eux reçoit $\frac{rows}{nproc}$ lignes de A. Le cas où *nproc* ne divise pas *rows* est géré en donnant au processus de rang le plus élevé, le reste de la division euclidienne de *rows* par *nproc*.

L'opération précédente est effectuée dans notre code par la fonction *calcul_repartition* : `ROOT` calcule donc $local\_rows = \frac{rows}{nproc}$ et $local\_rest\_rows = rows - (nproc - 1) * local\_rows$. Il initialise ensuite deux tableaux *ap_nb* et *ap_where* de taille *nproc*. Le premier (respectivement le second) contient dans sa case d'indice i la taille ($local\_rows$ ou $local\_rest\_rows$ si $i = nproc - 1$) (respectivement l'indice de la première ligne (rang*ap_nb[i])) du bloc assigné au processus P_i.

Les deux tableaux sont ensuite envoyés à tous les processus via la directive *MPI_Cast*.

Un meilleur équilibrage ?

Une optimisation possible serait de prendre en compte le caractère creux de la matrice. En effet, la distribution précédente divise A par ligne ; mais si par exemple le bloc 1 ne possède qu'un nombre très petit de coefficients non nuls par rapport au bloc 2, alors P_1 exécute beaucoup plus vite sa tâche et doit attendre P_2. Ainsi nous avons tenté de répartir équitablement le nombre de coefficients non nuls *nz* de A aux processus. Une/Des erreur(s) de programmation ne nous ont pas permis de la réaliser et donc de comparer les deux versions.

La distributions des blocs aux processus

La distribution de la matrice A est réalisée dans la fonction *distribution_matrice* après *calcul_repartition*. Avant l'appel à la fonction, les processus allouent l'espace nécessaire pour recevoir une matrice *A_local*.

De nouveau, deux tableaux *ax_nb* et *ax_where* de taille *nproc* sont initialisés, à partir du champ *Ap* de A, par `ROOT` : *ax_nb[i]* contient le nombre de coefficients non nul *nz_local* du bloc de A assigné à P_i et *ax_where[i]* contient la position du premier élément non nul du bloc de A assigné à P_i.

Ensuite `ROOT` procède à l'envoie de ces informations au processus via *MPI_Scatter* pour les *nz_local* et *MPI_Scatterv* pour les champs *Ap_local, Aj_local et Ax_local*.

Finalement, les processus reçoivent les données et initialisent leur matrice locale.

La dernière étape avant l'exécution de *cg_solve* est l'envoie du vecteur b aux processus via la directive *MPI_Scatterv* puisque seulement la partie utile au processus pour la résolution lui est envoyée.

Après cette étape de distribution chaque processus va exécuter *cg_solve*.
Matrice-vecteur
Puisque chaque processus possède seulement une matrice *A_local* nous ne modifions pas le code de *sp_gemv*. Le produit s'effectuera plus vite avec une matrice plus petite

Vecteur-vecteur
Les processus possèdent chacun une partie des deux termes du produit. Ainsi il est nécessaire que le résultat du produit réalisé par chaque processus soit communiqué à tous afin que le résultat final soit formé. Pour cela, nous utilisons la directive *MPI_Allreduce* avec la réduction *MPI_SUM* qui additionne les résultats des produits partiels et envoie cette somme à tous les processus. L'un des désavantages de cette méthode est l'espace alloué pour rien et les opérations redondantes qui en découlent. Une solution alternative serait par exemple d'utiliser *MPI_Gather*.

2 Hybride OpenMP-MPI

Conclusion

Même si nous n'avons pas pu allé très loin dans le projet nous avons quand même remarqué l'importance et l'efficacité de l'équilibrage de charge dans la parallélisation du code séquentiel, tandis que l'utilisation des fonctions MPI nous a permis de voir que différentes implémentations de la même idée peuvent avoir des résultats très différents sur l'efficacité.