École de technologie supérieure
Département de génie logiciel et des TI

Introduction au traitement parallèle avec MPI

N° du laboratoire	01
N° d'équipe	01
Étudiants	Alexandre Paquet Ara Sivaneswaran
Codes permanents	AN94010 SIVA19029209
Cours	LOG645
Session	Hiver 2017
Groupe	01
Chargé de cours	Lévis Thériault
Chargé de laboratoire	Kevin Lachance-Coulombe
Date	Jeudi 2 février 2017

Introduction

Le laboratoire nous a permis de mieux comprendre la base de la programmation parallèle. Nous avons utilisé MPI qui est un environnement de programmation parallèle. De plus, il nous a permi d'utiliser plusieurs outils comme le SSH, Unix, C et Makefile. Le travail a effectué durant le laboratoire était assez simple. Il y avait 2 problèmes assez simple à résoudre. La première étape consistait à le résoudre de façon séquentielle et par la suite, il fallait faire résoudre les même problèmes de façon parallèle. Finalement, il fallait afficher les matrices résultantes ainsi que temps d'exécution.

Analyse

Puisque tous les valeurs de la matrice résultante dépend des valeurs précédents, nous avons pas d'autre choix que de faire des boucles imbriquées pour instancier les valeurs du départ. Pour le problème 1, la dépendance est [k-1], il faut aller chercher la valeur dans l'altération ultérieure. Pour ce qui est du problème 2, c'est un peu compliqué. La première colonne a une dépendance identique au problème 1. Cependant pour les autres colonnes, les dépendances sont [k-1] et [j-1]. Il faut aller chercher la valeur de l'altération précédente et additionner cette valeur à la valeur de la rangée précédente dans l'altération précédente. La dépendance [k-1] nous empêche de diviser les calculs en 64 fois le nombre d'itérations, puisque les itérations sont dépendantes. On ne peut donc que diviser les calculs inter-itération. Cependant, la dépendance [j-1] nous empêche elle-aussi de diviser les calculs à l'intérieur d'une même ligne, car ce calcul nécessite le calcul de la case précédente pour le résultat. Ainsi, il ne nous reste que la division sou forme le ligne afin de paralléliser le code.

Voici les réponses aux questions demandées:

1. *Pour une exécution du programme donnée (« prog 1 5 8 »), combien de fois la fonction de calcul est-elle exécutée? Donner la moyenne par processeurs.*

   Il est exécuté 8 fois et en moyenne une fois par processeur.

2. *Quels sont les vecteurs de dépendance des cellules (excluant celles aux bordures de la matrice qui ont des dépendances différentes)?*

   Les vecteurs de dépendances sont [j-1] et [k-1] comme il a été énoncé plus haut.

3. *Quels sont les impacts de ces dépendances sur la communication et comment cela affecte-t-il votre agglomération? Utiliser un schéma si nécessaire.*

Comme dit précédemment, ces dépendances nous obligent à diviser les données sous forme de lignes, et d'envoyer ces lignes aux différents processeurs. Cela a pour effet de diminuer les communications, puisque si l'on pouvait diviser en unité plus affine encore,

on devrait augmenter les communications. Ainsi, en envoyant huit lignes, on a besoin de huit MPI_Send et huit MPI_Recv.

4. *Combien de messages les processeurs vont-ils échanger au total (pour chaque problème)?*

Au total, chaque processeur esclave envoie un message au processeur maître, ce qui fait un total de huit MPI_Send et huit MPI_Recv.

5. *Quelle est l'efficacité de votre programme parallèle et expliquez pourquoi elle n'est pas de 100%?*

L'efficacité de notre programme est d'environ 90%. Cela est dû principalement aux communications entre les processeurs esclaves et le processeur maître et à la synchronisation des messages du processeur maître, afin d'obtenir la bonne matrice résultante.

Conception

En ce qui concerne la conception des algorithmes séquentielles, cela était très simple. Pour le problème 1 et pour le problème 2, il suffit de deux boucles imbriquées parcourant les 64 indices de la matrice en calculant les valeurs de retour.

Afin de concevoir l'algorithme utilisant MPI pour paralléliser les deux problèmes énoncés ci-haut, il nous a fallu réfléchir aux dépendances des données que nous avions. Dans le cas du problème 1, cela était simple, car les données ne dépendent que d'elles-mêmes. Dans le cas du problème 2, cela était un peu différent, puisque nous avions une dépendances entre les colonnes. Pour ce faire, nous avons décidés de regrouper les valeurs de la même ligne et donc, d'envoyer une ligne à chaque processeur. En fait, au lieu d'envoyer une ligne à chaque processeur, étant donné que chaque ligne de la matrice est identique au début, nous initialisons les lignes dans chaque processeur pour ensuite faire le calcul. En ayant regroupé en lignes, nous brisons ainsi les dépendances et on permet aux huit processeurs esclaves de faire leur calcul. Une fois terminée, chaque processeur envoie sa ligne au processeur maître, qui lui va replacer les lignes dans le bon ordre afin d'obtenir le résultat escompté.

Discussion

Calcul de l'accélération pour problème 1

$S = t_1 / t_p$

$S = 0.658 / 0.082$

$S = 8.024$

Calcul de l'accélération pour problème 2

$S = t_1 / t_p$

$S = 0.658 / 0.081$

$S = 8.123$

Calcul de l'efficacité pour problème 1

$E = S/P$

$E = 8.024/9$

$E = 0.892$

Calcul de l'efficacité pour problème 2

$E = S/P$

$E = 8.123/9$

$E = 0.903$

En ce qui a trait à de possibles optimisations ou à de solutions alternatives, il serait possible de diviser la tâche en 64 cellules individuelles, et donc en 64 processeurs esclaves, qui recevrait par MPI_Send du processeur maître une cellule, ainsi que la cellule de gauche dans le cas où j est différent de 0. Cela permettrait de réduire le temps de calcul passé dans chaque processeur. Cependant, le temps de communication engendré par les MPI_Send et MPI_Recv serait plus important.

Conclusion

Le laboratoire nous a permis de se familiariser avec l'environnement de programmation en parallèle et les différents outils disponible. Le laboratoire se divisait en deux parties. La première partie devait être codé de façon séquentielle, tandis que la deuxième partie devait être en parallèle. Durant la partie parallèle, nous avons pu apprendre d'avantage sur la façon de coder en utilisant l'environnement MPI.