Master 2 Calcul Scientifique et Applications 2017-2018
Programmation parallèle et sur GPU

Introduction à `MPI`

1. **Ping-pong** : écrivez un programme qui échange un message de N nombres réels entre deux processus M fois, où N et M sont fournis par l'utilisateur. Le processus 0 envoie le message au processus 1, qui le renvoie au processus 0. Mesurez le temps pour cet échange et estimez le bandwidth. Variez la taille N du message. Présentez les résultats sous forme graphique.

2. **Boucle** : écrivez un programme qui envoie un message de N nombres réels dans une boucle constituée de plusieurs processus. La valeur de N ainsi que le nombre de tours que le message va effectuer dans la boucle doivent être fournis par l'utilisateur au début de l'exécution. Le processus 0 initialise le message et l'envoie au processus 1. Chaque processus doit imprimer les premiers éléments du message reçu ainsi que le rang du processus expéditeur, et ensuite envoyer le message au processus suivant. Le dernier processus envoie le message au processus 0 pour terminer le tour. Mesurez le temps pour réaliser plusieurs tours, pour des tailles de message différentes, et pour un nombre de processus différents. Présentez les résultats sous forme graphique.

3. **Recherche de nombres premiers** : écrivez un programme qui compte le nombre de nombres premiers entre 1 et un entier fourni par l'utilisateur. Utilisez l'algorithme naïf : pour tester si un entier N est premier ou non, vérifiez s'il est divisible exactement par des entiers plus petits. Le programme doit afficher comme résultat le nombre de nombres premiers trouvés : il n'est pas nécessaire d'afficher les nombres premiers eux-mêmes. Modifiez le programme pour l'exécuter en parallèle avec MPI. Distribuer le travail à faire (i) par bloc et (ii) de façon cyclique. Mesurez le temps écoulé pour des valuers de N différentes, et pour un nombre de processus différents. Présentez les résultats sous forme graphique. Quelle approche donne le meilleur gain de performance ? Pourquoi ?

4. **Master-slave** : écrivez un programme `master` qui envoie un nombre réel aléatoire entre 0 et 1 aux autres processus sur demande. Le programme `master` doit d'abord lire le nombre total N de valeurs à envoyer. Ecrivez le programme associé `slave` qui demande au `master` de lui envoyer un nombre aléatoire et qui affiche la valeur reçue. Lorsque les N valeurs ont été envoyées, le `master` envoie un nombre négatif à tous les processus `slave`. Avant de terminer, chaque `slave` affiche le nombre de valeurs reçues ainsi que leur somme (sans compter la valeur négative envoyée pour terminer l'exécution).

PTO

5. **Multiplication matrice-vecteur** : écrivez un programme qui effectue la multiplication d'un vecteur de N nombres réels aléatoires par une matrice de réels aléatoires de dimension (M x N) distribuée sur plusieurs processus. Chaque processus doit stocker un bloc de lignes de la matrice, qui doit être construite par le processus lui-même : il ne faut surtout pas construire toute la matrice sur un processus et ensuite la distribuer sur les autres. Le vecteur par contre doit être construit sur un processus et ensuite transmis aux autres. Le résultat doit être rassemblé sur le processus 0, qui doit l'afficher. Vous pouvez supposer que le nombre M de lignes est un multiple entier du nombre de processus : le programme doit vérifier que la valeur fourni par l'utilisateur respecte cette condition.

6. Téléchargez le programme `Heat2D.cpp` et les données d'éntrée `input.dat` du site du cours.

   Le programme fourni met en oeuvre **la résolution de l'équation de la chaleur en 2D**

   $$\frac{\partial}{\partial t}u(x,y,t) = \alpha\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)u(x,y,t)$$

   par la méthode des différences finies. $u(x,y,t)$ est un champ de température dans un domaine fini de l'espace à deux dimensions, défini par $x \in [x_{\text{lo}}, x_{\text{up}}]$ et $y \in [y_{\text{lo}}, y_{\text{up}}]$. Les coordonnées spatiales x et y sont discrétisées à l'aide d'une grille uniforme de points

   $$x_i = x_{\text{lo}} + i\Delta x, \qquad 0 \le i < N_x \qquad\qquad y_j = y_{\text{lo}} + j\Delta y, \qquad 0 \le j < N_y,$$

   tandis que le temps t est discrétisé selon $t = n\Delta t$, avec $n > 0$. La valeur de u à un point de la grille (x_i, y_j) à l'instant t_n est stockée dans u^n_{ij}. Les conditions limites imposées sur la solution recherchée sont $u(x_{\text{lo}}, y, t) = T_0$ et $u(x_{\text{up}}, y, t) = u(x, y_{\text{lo}}, t) = u(x, y_{\text{up}}, t) = 0$.

   Modifiez ce programme pour l'exécuter en parallèle avec MPI. Les lignes des matrices `u`, `du_x` et `du_y` doivent être distribuées sur des processus différents.

   Conseil : lors de l'évaluation du schéma de résolution pour la première/dernière ligne de la matrice locale, un processus particulier a besoin des valeurs $u^{(n)}_{ij}$ des dernières/premières lignes des matrices locales évaluées et stockées sur les processus voisins. Comment allez-vous gérer efficacement l'échange des données nécessaires ?

PTO

NB : Ces exercices peuvent être réalisés en utilisant uniquement les fonctions MPI suivantes :

`MPI_Bcast, MPI_Comm_rank, MPI_Comm_size, MPI_Finalize, MPI_Gather, MPI_Init, MPI_Recv, MPI_Reduce, MPI_Scatter, MPI_Send, MPI_Sendrecv`

Afin de mesurer le temps d'exécution, il faut utiliser `MPI_Barrier` and `MPI_Wtime`. Les tokens `MPI_ANY_SOURCE, MPI_ANY_TAG` peuvent également être utilisés.

Pour compiler prog.c :

```
mpicc prog.c -o prog.exe
```

Pour exécuter `prog.exe` avec n processus :

```
mpirun -np n -hostfile hosts prog.exe
```

où hostfile contient une liste de machines sur lesquelles le programme sera exécuté :

```
me052131> more hosts
148.60.41.1  slots=4
148.60.41.2  slots=4
148.60.41.3  slots=4
```

Cette liste doit comprendre la machine sur laquelle vous êtes connecté(e), et toutes les machines doivent être évidemment accessibles.

Pour exécuter `master.exe` sur un processeur ainsi que `slave.exe` sur 4 autres :

```
mpirun -np 1 master.exe : -np 4 slave.exe
```

Il faut pouvoir se connecter sur toutes les machines par ssh, sans devoir fournir explicitement un mot de passe. Pour le faire :

```
> cd $HOME/.ssh
> ssh-keygen -t rsa
  ....
> cp id_rsa.pub authorized_keys
```