Travaux pratiques MPI

François Willot
École des Mines de Paris
email: francois.willot@ensmp.fr

October 17, 2016

Le rapport est à rendre pour le vendredi 28/10 à 16h. Le rapport **court et concis** devra impérativement contenir les codes sources dans une version exploitable et compilable et être accompagné d'une notice expliquant: (i) le but du travail effectué et la démarche adoptée ; (ii) les difficultés et points délicats rencontrés lors de la parallélisation ; (iii) les solutions apportées.

1 Configuration

Configurer ssh pour ne pas avoir a taper votre mot de passe:

```
http://www.esiee.fr/~perrotol/ssh-guide.html
```

Ajoutez MPI au path :

```
echo "source /user/pvm/mpich2-install/cshrc.mpi" >> ~/.cshrc
```

2 Calcul d'une intégrale

Écrire un programme parallèle en C qui calcul π par la formule

$$\frac{\pi}{4} = \int_0^1 \frac{\mathrm{d}x}{1+x^2}.$$

Pour cela, utiliser la méthode des rectangles en divisant l'intervalle en n parties. Le nombre n est fourni au processus 0 qui le transmet aux autres nœuds. Chaque processus calcul l'intégrale sur un sous-domaine et les résultats ajoutés par une méthode de réduction. La somme est affichée à l'écran.

On peut utiliser les fonctions `MPI_BCAST, MPI_REDUCE` .

Utiliser des réels en double précision. Afficher l'erreur commise avec $\pi \approx 3.14159265358979323846264 3$.

3 Définition d'une structure de données en parallèle

On souhaite diviser une grille régulière en tranches réparties sur chaque processeur. Ainsi un tableau de données $2D$:

```
double x[maxn][maxn];
```

est stocké localement sur chaque nœud sous la forme :

```
double xlocal[maxn][maxn/nprocs];
```

où `nprocs` est le nombre de nœuds (pour simplifier on suppose que `nprocs` est un diviseur de `maxn`, par exemple `maxn` =12 et `nprocs` =4). Pour les calculs que nous devons réaliser, il est nécessaire de connaître les pixels adjacents. Ainsi pour mettre à jour la valeur de `x[i][j]` on doit avoir accès aux premiers voisins :

```
x[i][j+1]
x[i][j-1]
x[i+1][j]
x[i-1][j]
```

Pour les deux derniers, ceci nécessite de stocker localement deux lignes supplémentaires de recouvrement en haut et en bas, comme indiqué sur la figure 3.

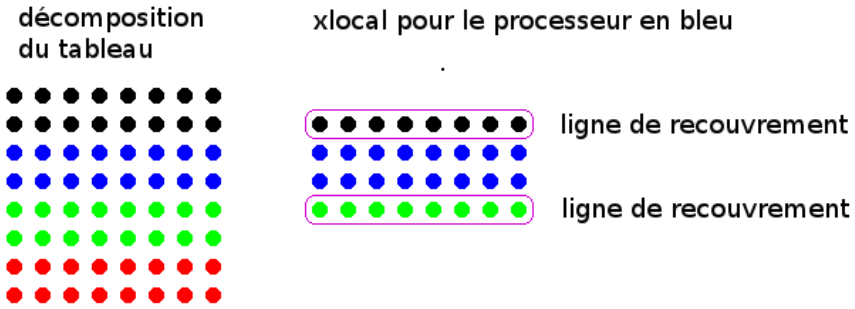

Figure 1: Répartition d'un tableau de données sur chaque nœud, une couleur par processeur. En plus de cela, deux lignes supplémentaires de recouvrement sont stockées sur chaque processeur.

Écrire un programme parallèle dans lequel chaque processus alloue un sous-tableau, l'initialise et envoie les lignes de recouvrement aux processeurs voisins. Les données du tableaux sont initialisées avec le rang du processeur et -1 pour la première et dernière ligne. On suppose que le milieu n'est pas périodique, c'est-à-dire que le processus du haut (resp. du bas) n'envoie et ne reçoit de données

que de celui-du dessous (resp. du dessus). Vérifier les valeurs du tableau en l'affichant à l'écran, sous la forme d'un tableau.

On pourra utiliser les fonctions `MPI_SEND, MPI_RECV, MPI_BARRIER`.

4 Résolution de l'équation de Laplace en parallèle

On souhaite implémenter l'algorithme suivant sur la structure définit au dessus :

```
while (not converged) {
  for (i,j)
    fnew[i][j] = 0.25*(f[i+1][j] + f[i-1][j]
                     + f[i][j+1] + f[i][j-1]);
  for (i,j)
    f[i][j] = fnew[i][j];
  }
```

Cette opération s'applique uniquement aux pixels intérieurs (voir figure 4). On utilisera comme critère de convergence :

```
err = 0;
for (i,j)
  err += (fnew[i][j] - f[i][j]) * (fnew[i][j] - f[i][j]);
err = sqrt(err);
```

et les valeurs du bord comme indiqué dans l'exercice précédent soit :

−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3
2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1

On pourra utiliser les fonctions `MPI_SEND, MPI_RECV`.

Ecrire le résultat dans un fichier texte, en respectant l'ordre des éléments.

5 Décomposition 2D

Récrire l'algorithme précédente avec une "décomposition 2D". Pour simplifiern, on suppose que le nombre de processeurs est un carré. Implémenter l'algorithme.

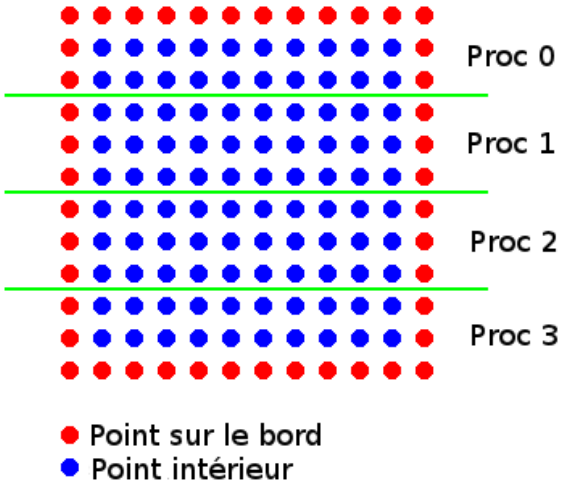

Figure 2: Pixels situés à l'intérieur et à l'extérieur du domaine, et répartition sur chaque processeur.

6 Benchmark (facultatif)

Calculer les temps d'exécution du programme pour divers tailles de système (i.e. taille du tableau) et un nombre de processeurs variable. Calculer en particulier le temps de communication et le temps de calcul. Calculer le nombre de processeur optimal pour une taille de système donnée, pour les deux décompositions (ou celle qui est implémentée).