

Travaux pratiques MPI – Liste des exercices

1	T.P. MPI – Exercice 1 : Environnement MPI	2
2	T.P. MPI – Exercice 2 : Ping-pong	3
3	T.P. MPI – Exercice 3 : Communications collectives et réductions	5
4	T.P. MPI – Exercice 4 : Transposée d'une matrice	6
5	T.P. MPI – Exercice 5 : Produit réparti de matrices	7
6	T.P. MPI – Exercice 6 : Communicateurs	13
7	T.P. MPI – Exercice 7 : Lecture d'un fichier en mode parallèle.....	14
8	T.P. MPI – Exercice 8 : Équation de Poisson	15

Travaux pratiques MPI – Exercice 1 : Environnement MPI

- Gestion de l'environnement de MPI : faire afficher un message par chacun des processus, mais **différent** selon qu'ils sont de rang **pair** ou **impair**
 - Soit par exemple pour les processus de rang **pair** un message du genre :

Je suis le processus **pair** de rang **M**

- Et pour les processus de rang **impair** un message du genre :

Je suis le processus **impair** de rang **N**

- Remarque : la fonction intrinsèque Fortran à utiliser pour tester la parité est **mod** : `mod(nombre1, nombre2)`

Travaux pratiques MPI – Exercice 2 : Ping-pong

- Communications point à point : *ping-pong* entre deux processus
 - ① Dans le premier sous-exercice, on fera uniquement un *ping* (envoi d'un message (*balle*) du processus 0 au processus 1)
 - ② Dans le deuxième sous-exercice, on enchaînera le *pong* après le *ping* (le processus 1 renvoyant le message reçu du processus 0)
 - ③ Dans le troisième sous-exercice, on effectuera une répétition du *ping-pong* en faisant varier à chaque fois la taille du message à échanger
- Soit :
 - ① Envoyer un message contenant 1000 nombres réels du processus 0 vers le processus 1 (il s'agit alors seulement d'un *ping*)
 - ② Faire une version *ping-pong* où le processus 1 renvoie le message reçu au processus 0 et mesurer le temps de communication à l'aide de la fonction `MPI_WTIME()`
 - ③ Faire une version où l'on fait varier la taille du message dans une boucle et mesurer les temps de communication respectifs ainsi que les débits

Remarques :

- La génération de nombres réels pseudo-aléatoires uniformément répartis dans l'intervalle $[0., 1.]$ se fait en Fortran par un appel au sous-programme

`random_number` :

```
call random_number(variable)
```

`variable` pouvant être un scalaire ou un tableau

- Les mesures de temps peuvent s'effectuer de la façon suivante :

```
.....  
temps_debut=MPI_WTIME()  
.....  
temps_fin=MPI_WTIME()  
print ('(... en",f8.6," secondes.")'),temps_fin-temps_debut  
.....
```

T.P. MPI – Exercice 3 : Communications collectives et réductions

- Il s'agit de calculer π par intégration numérique $\pi = \int_0^1 \frac{4}{1+x^2} dx$.
- On utilise la méthode des rectangles (point milieu).
- La fonction à intégrer est $f(x) = \frac{4}{1+x^2}$.
- *nbbloc* est le nombre de points.
- *largeur* = $\frac{1}{nbbloc}$ est le pas de discrétisation et la largeur de chaque rectangle.
- La version séquentielle est disponible dans le fichier `pi.f90`.
- Il vous faut écrire la version parallélisée avec MPI dans ce fichier.

Travaux pratiques MPI – Exercice 4 : Transposée d'une matrice

- Dans cet exercice, on se propose de se familiariser avec les types dérivés
- On se donne une matrice A de 5 lignes et 4 colonnes sur le processus 0
- Il s'agit pour le processus 0 d'envoyer au processus 1 cette matrice mais d'en faire automatiquement la transposition au cours de l'envoi

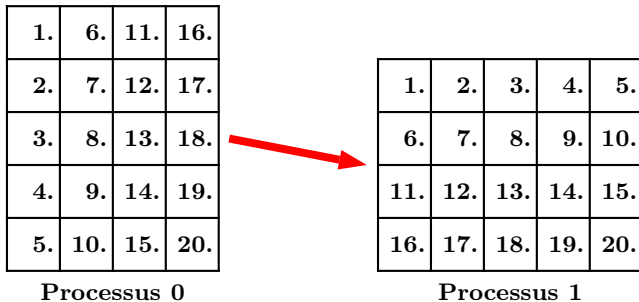


FIGURE 1 – Transposée d'une matrice

- Pour ce faire, on va devoir se construire deux types dérivés, un type `type_ligne` et un type `type_transpose`

Travaux pratiques MPI – Exercice 5 : Produit réparti de matrices

- Communications collectives et réductions : produit de matrices $C = A \times B$
 - On se limite au cas de matrices carrées dont l'ordre est un multiple du nombre de processus
 - Les matrices A et B sont sur le processus 0. Celui-ci distribue une tranche horizontale de la matrice A et une tranche verticale de la matrice B à chacun des processus. Chacun calcule alors un bloc diagonal de la matrice résultante C .
 - Pour calculer les blocs non diagonaux, chaque processus doit envoyer aux autres processus la tranche de A qu'il possède (voir la figure 2)
 - Après quoi le processus 0 peut collecter les résultats et vérifier les résultats

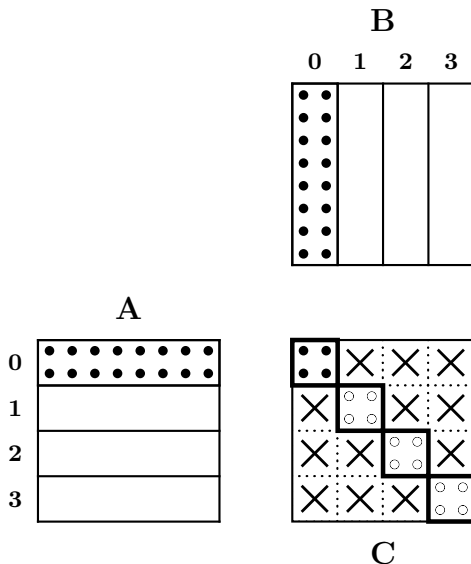


FIGURE 2 – Produit parallèle de matrices

- Toutefois, l'algorithme qui peut sembler le plus immédiat, et qui est le plus simple à programmer, consistant à faire envoyer par chaque processus sa tranche de la matrice A à chacun des autres, n'est pas performant parce que le schéma de communication n'est pas du tout équilibré. C'est très facile à voir en faisant des mesures de performances et en représentant graphiquement les traces collectées. Voir les fichiers `produit_matrices_v1_n3200_p4.slog2`, `produit_matrices_v1_n6400_p8.slog2` et `produit_matrices_v1_n6400_p16.slog2`, à utiliser via l'outil `jumpshot` de MPE (*MPI Parallel Environment*).

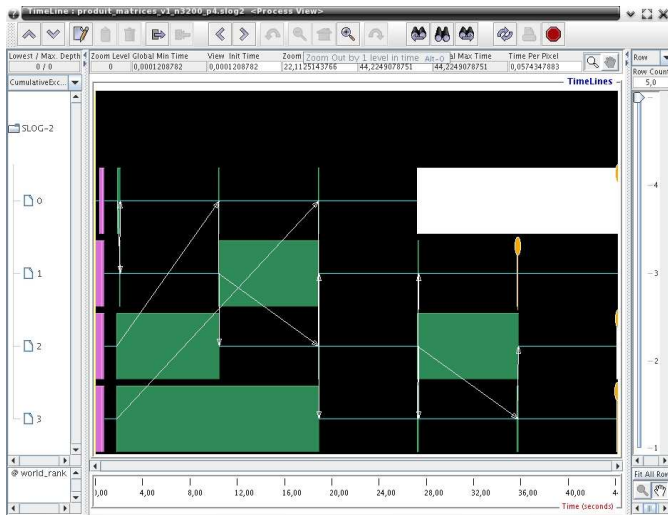


FIGURE 3 – Produit parallèle de matrices sur 4 processus, pour une taille de matrice de 3200 (premier algorithme)

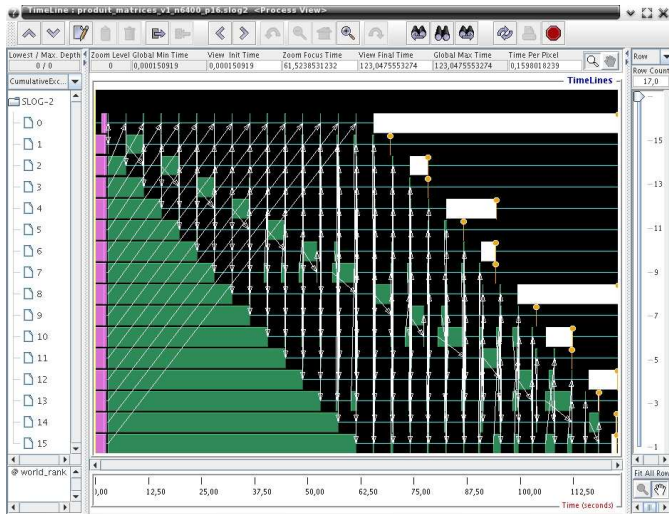


FIGURE 4 – Produit parallèle de matrices sur 16 processus, pour une taille de matrice de 6400 (premier algorithme)

- Mais en changeant l'algorithme pour faire *glisser* le contenu des tranches de processus à processus, on peut obtenir un équilibre parfait des calculs et des communications, et gagner ainsi un facteur 2. Voir la représentation produite par le fichier `produit_matrices_v2_n6400_p16.slog2`.

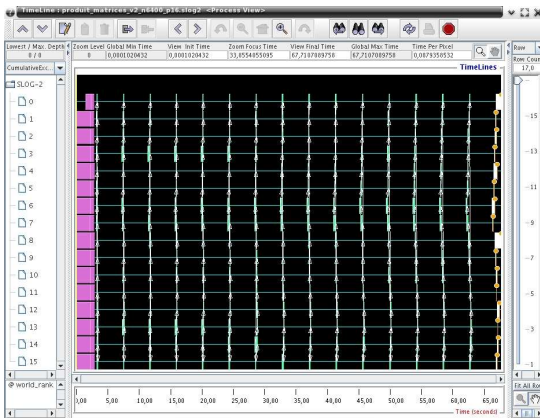


FIGURE 5 – Produit parallèle de matrices sur 16 processus, pour une taille de matrice de 6400 (second algorithme)

Travaux pratiques MPI – Exercice 6 : Communicateurs

- En partant de la topologie cartésienne définie ci-dessous, subdiviser en 2 communicateurs suivant les lignes via `MPI_COMM_SPLIT()`

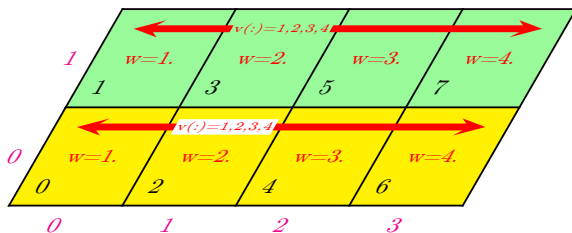


FIGURE 6 – Subdivision d’une topologie 2D et communication suivant la topologie 1D obtenue

T.P. MPI – Exercice 7 : Lecture d'un fichier en mode parallèle

- On dispose du fichier binaire `donnees.dat`, constitué d'une suite de 484 valeurs entières
- En considérant un programme parallèle mettant en œuvre 4 processus, il s'agit de lire les 121 premières valeurs sur le processus 0, les 121 suivantes sur le processus 1, etc. et d'écrire celles-ci dans quatre fois quatre fichiers appelés `fichier_XXX0.dat` ... `fichier_XXX3.dat`
- On emploiera pour ce faire 4 méthodes différentes, parmi celles présentées :
 - lecture via des déplacements explicites, en mode individuel ;
 - lecture via les pointeurs partagés, en mode collectif ;
 - lecture via les pointeurs individuels, en mode individuel ;
 - lecture via les pointeurs partagés, en mode individuel.
- Pour compiler et exécuter le code, utilisez la commande `make` et pour vérifier les résultats utilisez la commande `make verification` qui exécute un programme de visualisation graphique correspondant aux quatre cas à traiter.

Travaux pratiques MPI – Exercice 8 : Équation de Poisson

On considère l'équation de Poisson suivante :

$$\begin{cases} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= f(x, y) \quad \text{dans } [0, 1] \times [0, 1] \\ u(x, y) &= 0. \quad \text{sur les frontières} \\ f(x, y) &= 2. (x^2 - x + y^2 - y) \end{cases}$$

On va résoudre cette équation avec une méthode de décomposition de domaine :

- L'équation est discrétisée sur le domaine via la méthode des différences finies.
- Le système obtenu est résolu avec un solveur suivant la méthode de Jacobi.
- Le domaine global est découpé en sous domaines.

La solution exacte est connue et est $u_{exacte}(x, y) = xy(x - 1)(y - 1)$

Pour discrétiser l'équation, on définit une grille constituée d'un ensemble de points (x_i, y_j)

$$x_i = i h_x \quad \text{pour } i = 0, \dots, ntx + 1$$

$$y_j = j h_y \quad \text{pour } j = 0, \dots, nty + 1$$

$$h_x = \frac{1}{(ntx + 1)}$$

$$h_y = \frac{1}{(nty + 1)}$$

h_x : pas suivant x

h_y : pas suivant y

ntx : nombre de points intérieurs suivant x

nty : nombre de points intérieurs suivant y

Il y a au total $ntx+2$ points suivant x et $nty+2$ points suivant y

- Soit u_{ij} l'estimation de la solution à la position $x_i = ih_x$ et $x_j = jh_y$.
- La méthode de Jacobi consiste à calculer :

$$u_{ij}^{n+1} = c_0(c_1(u_{i+1j}^n + u_{i-1j}^n) + c_2(u_{ij+1}^n + u_{ij-1}^n) - f_{ij})$$

$$\text{avec : } \begin{aligned} c_0 &= \frac{1}{2} \frac{h_x^2 h_y^2}{h_x^2 + h_y^2} \\ c_1 &= \frac{1}{h_x^2} \\ c_2 &= \frac{1}{h_y^2} \end{aligned}$$

- En parallèle, les valeurs aux interfaces des sous-domaines doivent être échangées entre les voisins.
- On utilise des cellules fantômes, ces cellules servent de buffer de réception pour les échanges entre voisins.

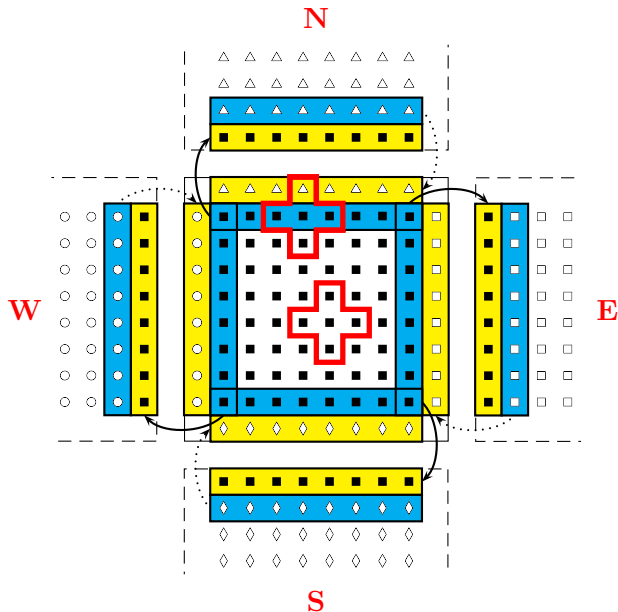


FIGURE 7 – Échange de points aux interfaces

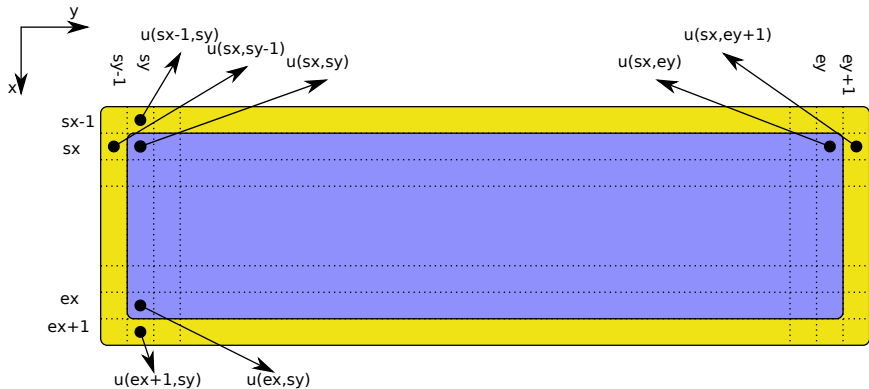


FIGURE 8 – Numérotation des points dans les différents sous-domaines

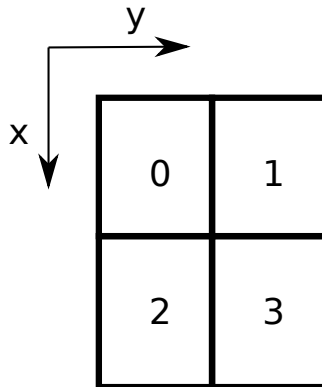


FIGURE 9 – Rang correspondant aux différents sous-domaines

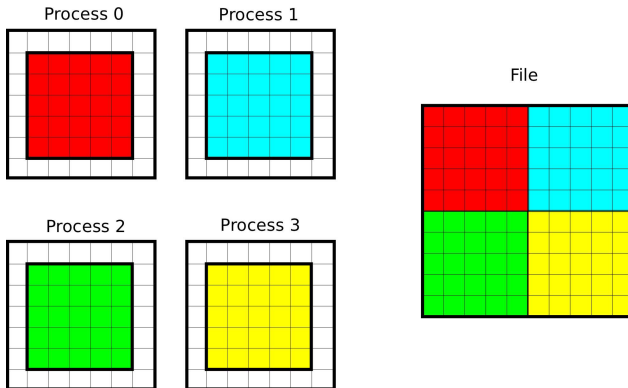


FIGURE 10 – Ecriture de la matrice u globale dans un fichier

Il s'agit de définir :

- Une vue, pour ne voir dans le fichier que la partie de la matrice u globale que l'on possède ;
- Un type afin d'écrire la matrice u locale (sans les cellules fantômes) ;
- Appliquer la vue au fichier ;
- Faire l'écriture en une fois.

- initialiser l'environnement MPI ;
- créer la topologie cartésienne 2D ;
- déterminer les indices de tableau pour chaque sous-domaine ;
- déterminer les 4 processus voisins d'un processus traitant un sous-domaine donné ;
- créer deux types dérivés *type_ligne* et *type_colonne* ;
- échanger les valeurs aux interfaces avec les autres sous-domaines ;
- calculer l'erreur globale. Lorsque l'erreur globale sera inférieure à une valeur donnée (précision machine par exemple), alors on considérera qu'on a atteint la solution ;
- reformer la matrice u globale (identique à celle obtenue avec la version monoprocesseur) dans un fichier **donnees.dat**.

- Un squelette de la version parallèle est proposé : il s'agit d'un programme principal (`poisson.f90`) et de plusieurs sous-programmes. Les modifications sont à effectuer dans le fichier `module_parallel_mpi.f90`.
- Pour compiler et exécuter le code, utilisez la commande `make` et pour vérifier les résultats. Utilisez la commande `make verification` qui exécute un programme de relecture du fichier `donnees.dat` puis le compare avec la version monoprocasseur.