Exercice 1 : environnement MPI

☞ Gestion de l'environnement de MPI : affichage d'un message par chacun des processus, mais **différent** selon qu'ils sont de rang **pair** ou **impair**

Exercice 2 : ping-pong

☞ Communications point à point : *ping-pong* entre deux processus

① Envoyer un message contenant 1000 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

☞ Les mesures de temps peuvent se faire de la façon suivante :

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

Exercice 3 : communications collectives et réductions

☞ En tirant à pile ou face sur chacun des processus, boucler jusqu'à ce que tous les processus fassent le même choix ou bien jusqu'à ce qu'on atteigne un nombre, fixé a priori, d'essais

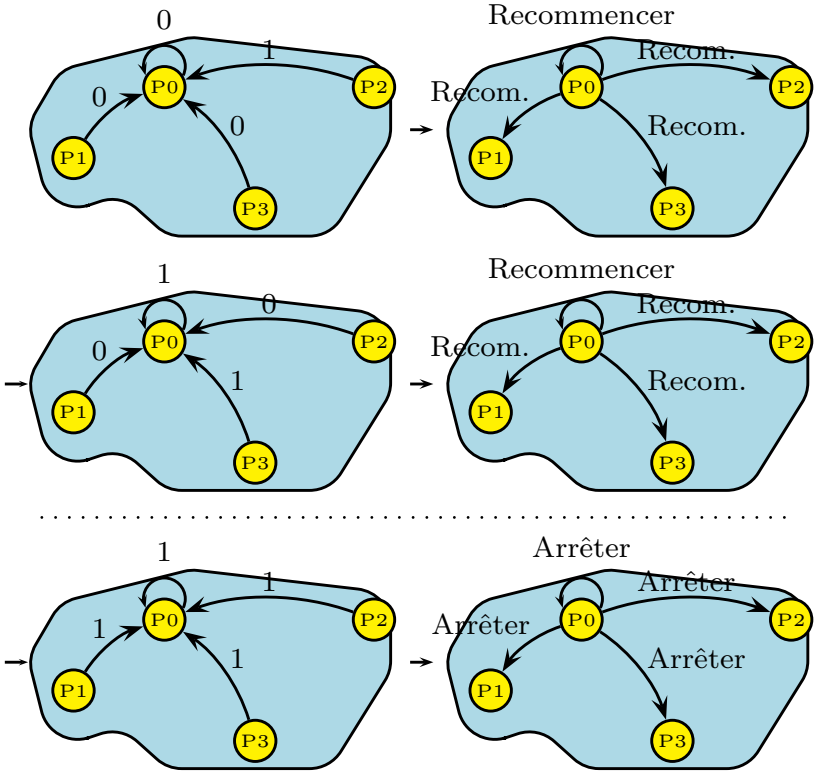

FIG. 1 – Tirage à pile ou face jusqu'à l'unanimité de tous les processus

Exercice 4 : produit 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

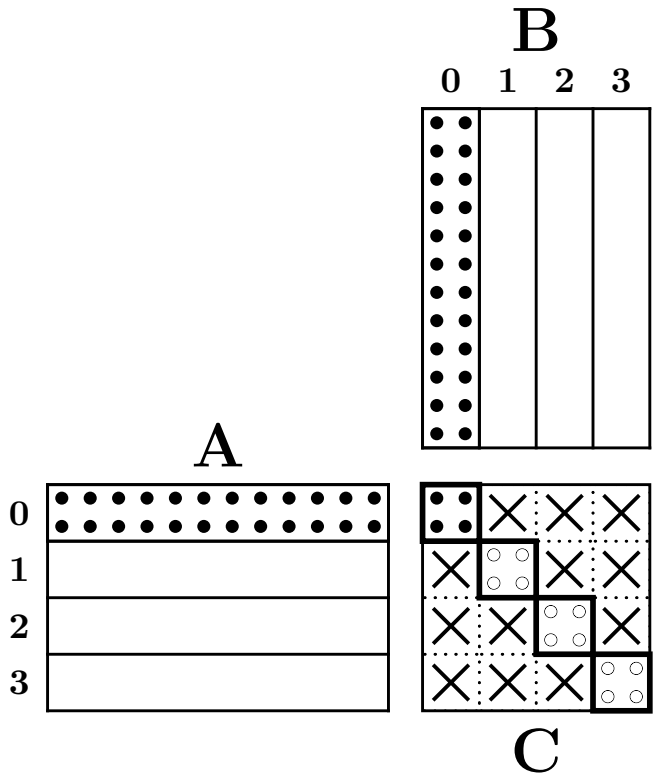

FIG. 2 – Produit réparti de matrices

Exercice 5 : transfert de lignes de matrice

- Dans cet exercice, on se propose de se familiariser avec les types dérivés
- On se donne une matrice A initialisée sur chacun des processus
- Il s'agit pour le processus 0 d'envoyer les *seconde* et *troisième* lignes de sa matrice au processeur 1 et pour celui-ci de les recevoir dans ses *avant-dernière* et *dernière* lignes

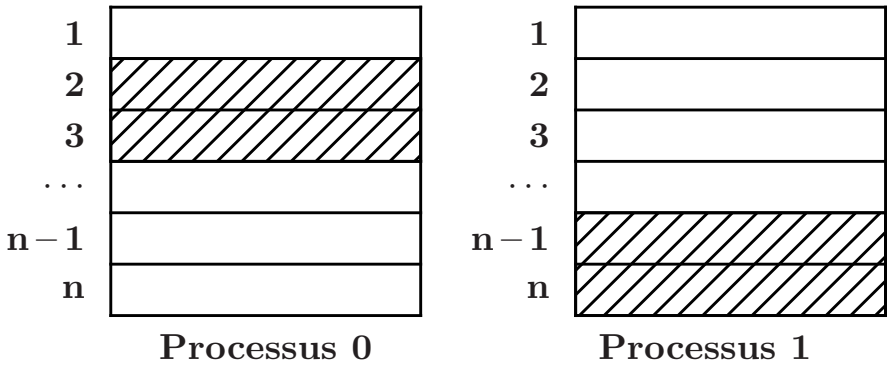

FIG. 3 – Transfert de deux lignes de matrice

- Pour ce faire, on va devoir se construire un type `type_ligne` qui décrira deux lignes de matrice

Exercice 6 : communicateurs

- En partant de l'exemple du paragraphe *Subdiviser une topologie cartésienne* du chapitre 8 *Communicateurs*, remplacez le sous-programme `MPI_CART_SUB()` par `MPI_COMM_SPLIT()` tout en conservant la même subdivision `Comm1D` de la topologie cartésienne `Comm2D`

Exercice 7 : équation de Poisson

☞ Résolution de l'équation de Poisson sur le domaine [0,1]x[0,1] par une méthode aux différences finies avec un solveur Jacobi

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

☞ Pour trouver une solution approchée à ce problème, on se définit une grille constituée d'un ensemble de points (x_i, y_j)

$$\begin{aligned} x_i &= i\ h_{\text{x}} \quad \text{pour } i = 0, \ldots, ntx+1 \\ y_j &= j\ h_{\text{y}} \quad \text{pour } j = 0, \ldots, nty+1 \end{aligned}$$

$$\begin{aligned} h_{\text{x}} &= \frac{1}{(ntx+1)} \\ h_{\text{y}} &= \frac{1}{(nty+1)} \end{aligned}$$

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

☞ La solution u à l'instant $n+1$ est fonction de la solution u à l'instant n

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

avec :

$$c_0 = \frac{1}{2}\frac{h_x^2 h_y^2}{h_x^2 + h_y^2}$$

$$c_1 = \frac{1}{h_x^2}$$

$$c_2 = \frac{1}{h_y^2}$$

☞ Pour simplifier la résolution, on se donne le nombre de points intérieurs en x et en y par sous-domaine, respectivement nx et ny. Ces valeurs seront lues dans un fichier d'entrée `poisson.data`

☞ Schéma du programme :

➪ décomposer le domaine en plusieurs sous-domaines, avec autant de sous-domaines que de processus ;

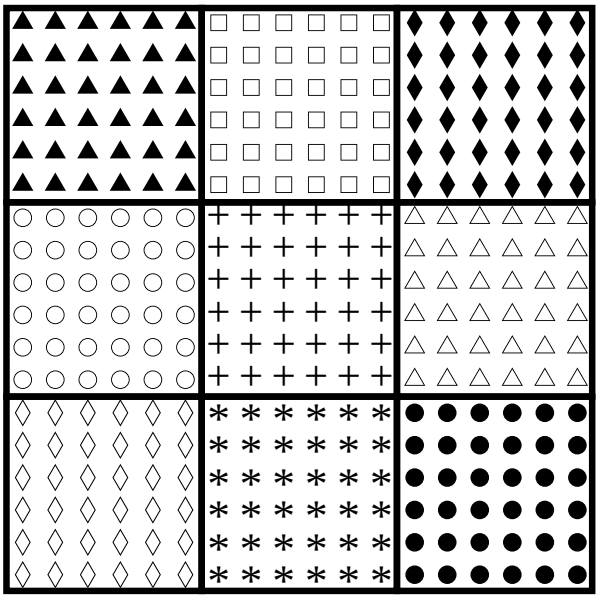

FIG. 4 – Découpage en sous-domaines

3	7	11	15
2	6	10	14
1	5	9	13
0	4	8	12

FIG. 5 – Numérotation des processus correspondant aux différents sous-domaines

- déterminer les 4 processus voisins d'un processus traitant un sous-domaine donné ;
- initialiser les valeurs de u, u_exacte et f ;
- échanger les valeurs aux interfaces avec les autres sous-domaines.

  Par exemple, si le domaine *idom* doit envoyer ses points $u(1, ny), ..., u(nx, ny)$ à son voisin *Nord*, comme indiqué sur la figure 7, ce dernier les recevra donc de ce qui est pour lui son voisin *Sud* et les stockera aux positions $u(1, 0), ..., u(nx, 0)$.

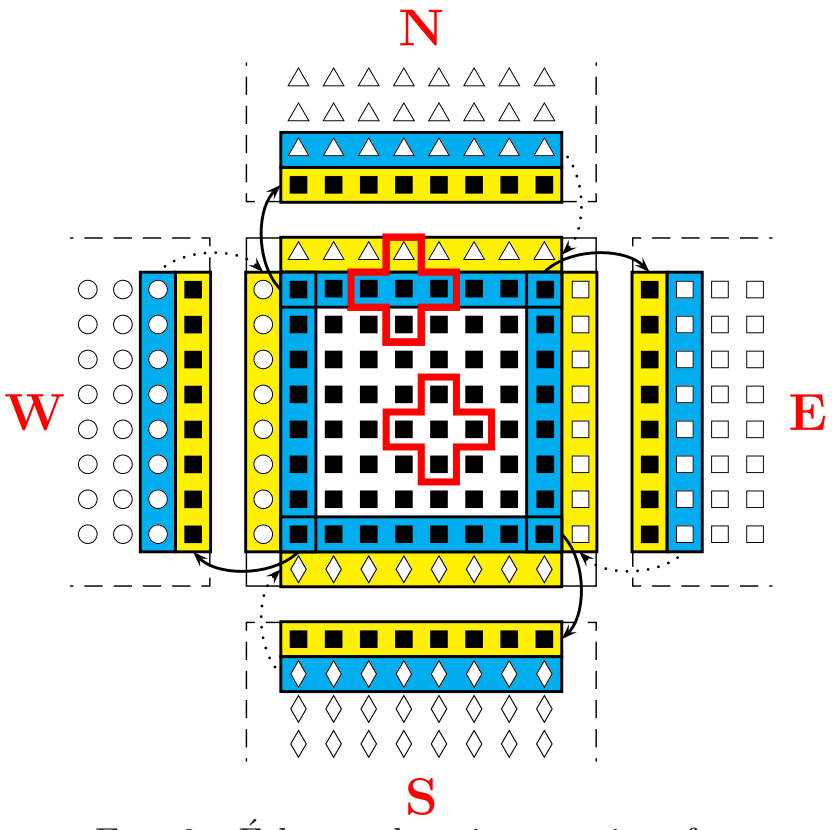

FIG. 6 – Échange de points aux interfaces

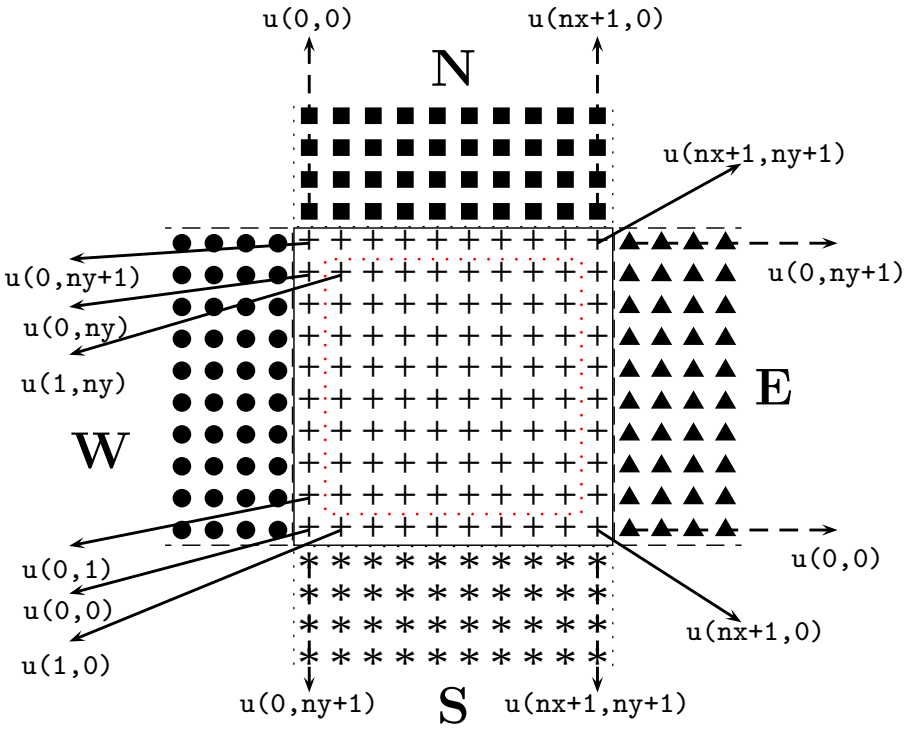

FIG. 7 – Numérotation des points dans les différents sous-domaines

- ▻▸ calculer ;
- ▻▸ calculer l'erreur sur un sous-domaine (maximum de la différence entre la valeur de u à l'instant $n+1$ et la valeur de u à l'instant n) ;
- ▻▸ comparer ce maximum avec tous les autres maximums des autres processus. Lorsque le maximum global sera supérieur à une valeur donnée (précision machine par exemple), alors on considèrera qu'on a atteint la solution.

☞ Répertoire : `tp7/poisson`

☞ Un squelette de la version parallèle est proposé : il s'agit d'un programme principal (`poisson.f90`) et de plusieurs sous-programmes