Communicateurs et produit matriciel en MPI

Résumé: Dans ce TD, nous allons mettre en œuvre un produit de matrice en MPI basé l'algorithme de double diffusion. Les diffusions se faisant sur certain processeurs uniquement, nous allons créer de nouveaux groupes de processus pour pouvoir utiliser la fonction `MPI_Bcast`.

1 Communicateurs MPI

Nous avons déjà utilisé le communicateur `MPI_COMM_WORLD` qui regroupe l'intégralité des processus lancés. Il est possible de créer d'autres communicateurs, c'est à dire d'autres groupes de processus, notamment en utilisant la fonction `MPI_Comm_split`.

```
int MPI_Comm_split ( MPI_Comm comm, int color, int key, MPI_Comm *newcomm )
```

- `comm` est le communicateur que l'on souhaite scinder
- `color` est un entier positif qui détermine à quel ensemble on appartiendra, le nouveau communicateur regroupant les processus de même couleur.
- `key` est un entier qui permet de déterminer le rang du processus au sein du nouveau communicateur. Deux processeurs de même couleur seront donc dans le même communicateur et leur rang sera déterminé en fonction de leur valeur respectives de `key`
- `newcomm` est le nouveau communicateur.

2 Produit de matrice parallèle

2.1 Rappel sur l'algorithme par double diffusion

On souhaite effectuer le calcul $C_{ij} = (A.B)_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$ pour tout i, j dans $[\![1, n]\!]$. L'Algorithme 1 décrit une façon naturelle d'effectuer ce calcul mais pas forcément adaptée à une parallélisation directe.

MatMult(A, B, C)
1: **Pour** $i = 1$ **à** n **:**
2: **Pour** $j = 1$ **à** n **:**
3: **Pour** $k = 1$ **à** n **:**
4: $C_{ij} \leftarrow C_{ij} + A_{ik}B_{kj}$

Algo. 1: Algorithme général séquentiel du produit de matrice

Les boucles 1 et 2 de cet algorithme sont parallèles. La boucle de la ligne 3 correspond à une opération de réduction (qui peut s'effectuer par exemple à l'aide d'un arbre) et réduit le parallélisme. On séquentialise généralement une partie de ces boucles afin d'ordonner et de régulariser les calculs (éviter des migrations de données excessives et désordonnées), de simplifier le contrôle et surtout d'adapter la granularité de la machine cible.

MatMult(A, B, C)
1: **Pour** $k = 1$ **à** n **:**
2: **Pour tout** i **: en parallèle**
3: **Pour tout** j **: en parallèle**
4: $C_{ij} \leftarrow C_{ij} + A_{ik}B_{kj}$

Algo. 2: Algorithme parallèle du produit de matrice

Il est donc préférable de permuter les boucles pour arriver à l'Algorithme 2 et d'effectuer alors la boucle externe (ligne 1) séquentiellement et les boucles internes (lignes 2 et 3) en parallèle. Si chaque C_{ij} est alloué à une unité de calcul fixe, alors les A_{ik} et les B_{kj} doivent circuler entre chaque étape de calcul, mais cela ne nuit pas au parallélisme et permet un recouvrement potentiel du calcul et des communications (même si le premier TD a démontré la difficulté de ce recouvrement avec MPI).

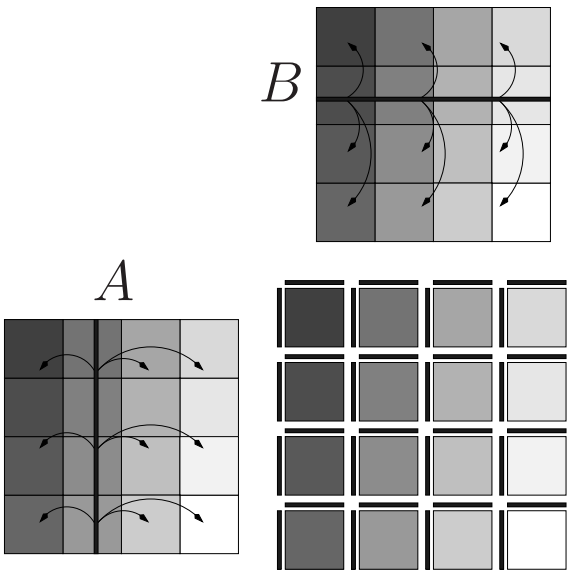

Fig. 1 – Distribution homogène contiguë pour une grille 4×4

Dans le cas où l'on dispose de $p.q$ processeurs identiques interconnectés en grille (voir Figure 1) : chaque processeur est responsable de sous-matrices de taille $\frac{n}{p} \times \frac{n}{q}$ de A, B et C. L'Algorithme 2 se déroule donc en n étapes et à l'étape k, la $k^{ème}$ colonne de A et la $k^{ème}$ ligne de B sont diffusées horizontalement (pour la colonne) et verticalement (pour la ligne). Chaque processeur reçoit donc un fragment de colonne de A et un fragment de ligne de B de tailles respectivement n/p et n/q et met à jour la partie de C dont il est responsable (voir Figure 1).

2.2 Mise en œuvre en MPI de l'algorithme par double diffusion

Vous trouverez dans http://www.ens-lyon.fr/~yrobert/algopar/MPI/src/ un canevas pour le programme de produit matriciel (`matmult_template.c`). Comme d'habitude, le programme est commenté et il n'y a qu'à remplir les trous. Une fois que le programme fonctionne, augmentez la granularité des communications. Comment faire pour évitez des copies inutiles lors des envois des fragment de lignes et de colonnes ?

Une solution est disponible à la même adresse dans `matmult.c`.