

TP MPI N°3 : Tri parallèle – Communication locale et collective

Un calcul parallèle est souvent composé d'un certain nombre d'étapes. Des échanges de données entre les processus sont généralement nécessaires au début et/ou à la fin de chaque étape. Cette pratique sera exercée à travers l'algorithme parallèle « tri pair/impair ».

1. Tri séquentiel

Le tri à bulles est un algorithme de tri d'un tableau qui consiste à faire remonter progressivement les plus petits éléments d'un tableau, comme les bulles d'air qui remontent à la surface d'un liquide.

```
Procédure Tri_a_bulle(T, n):  
E/S: T tableau de n éléments  
E : n taille du tableau  
  
for i=n-2 to 0 step -1  
    for j=0 to i step 1  
        compare-exchange(T[j], T[j+1])
```

Or cet algorithme est intrinsèquement séquentiel à cause d'une dépendance dans les comparaisons : T[j] est comparé à T[j+1] qui est lui-même comparé à T[j+2]. Une variante de ce tri est le tri pair-impair. La boucle interne est alors divisée en deux phases : une phase pour comparer les éléments d'indices pairs du tableau avec les suivants et l'autre phase pour comparer les éléments d'indices impairs du tableau avec les suivants. Ce qui fait que les comparaisons d'une phase sont indépendantes et peuvent être exécutées simultanément.

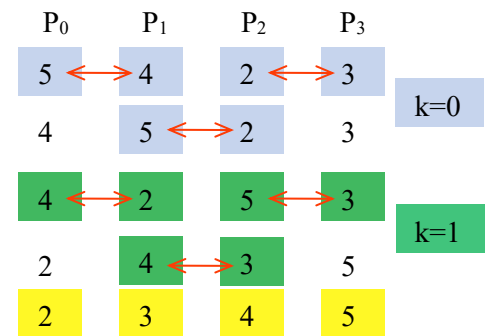
```
Procédure Tri_pair_impair(T, n):  
E/S: T tableau de n éléments  
E : n taille du tableau  
  
for i=0 to n-1 step 1  
    if i%2==0 // i pair  
        for j=0 to n-2 step 2  
            compare-exchange(T[j], T[j+1])  
    else // i impair  
        for j=1 to n-2 step 2  
            compare-exchange(T[j], T[j+1])
```

2. Tri parallèle pair-impair

Le tri parallèle pair-impair consiste à réaliser le tri d'un tableau de n éléments par p processus. Pour ce faire, les éléments du tableau sont d'abord distribués aux p processus à partir du processus `root`, puis l'algorithme parallèle échange les éléments entre les processus voisins, les trie, fait en sorte que les plus petits éléments remontent vers les processus du rang inférieur. A la fin de cet algorithme, tous les processus remontent ses éléments vers le processus `root` et forme une liste triée.

a. Tri pair/impair d'un tableau de p éléments sur p processus

Nous allons commencer par le cas où $n=p$, i.e. chaque processus a un seul élément du tableau à trier en mémoire. Cet élément sera généré de manière aléatoire dans chaque processus pour ce TP. La partie tri de l'algorithme parallèle est décrit comme suit :

Procédure Tri_pair_impair(T, n):*E/S : T tableau de n éléments non trié**E : n taille du tableau***for** k=0 **to** n/2-1 (if n even) **step** 1 P_i de i pair compare-exchange avec P_{i+1} P_i de i impair compare-exchange avec P_{i+1} 

Une trace de cet algorithme avec $n=p=4$ est illustré ci-dessus. On remarque qu'une itération k est composée de 2 phases. Dans la 1^{ère} phase, chaque processus ayant un rang pair communique avec son voisin du rang+1 : ex. P_0 communique avec P_1 , P_2 communique avec P_3 ; dans la 2^{ème} phase, chaque processus ayant un rang impair communique avec son voisin du rang+1 : ex. P_1 communique avec P_2 . A la fin des $n/2-1$ itérations, le plus petit élément du tableau se trouve dans le processus P_0 , le 2^{ème} plus petit élément est dans le P_1 , ..., et le plus grand élément est dans P_{p-1} .

Implémenter l'algorithme précédent, tester et valider votre programme.

Attention : la numérotation de tableau et de processus a été adaptée pour C et MPI, et commence à partir de 0.

b. Tri pair/impair d'un tableau de n éléments p processus, avec $n = m \cdot p$

Dans le cas où $n=m \cdot p$, chaque processus a m éléments en mémoire après l'étape de distribution. La partie tri de l'algorithme parallèle pair/impair est le suivant :

Procédure Tri_pair_impair:

chaque processus trie ses éléments

for k=0 **to** p/2-1 **step** 1 P_i de i **pair** travaille avec P_{i+1} :

- échange de leur sous-listes

- tri-fusion des 2 sous-listes

- P_i garde la 1^{ère} moitié de la liste- P_{i+1} garde la 2^{ème} moitié de la liste P_i de i **impair** travaille avec P_{i+1}

- échange de leur sous-listes

- tri-fusion des 2 sous-listes

- P_i garde la 1^{ère} moitié de la liste- P_{i+1} garde la 2^{ème} moitié de la liste

On demande de programmer cet algorithme en C/MPI. Tester et valider votre programme avec plusieurs jeux de tests générés de manière aléatoire et en faisant varier les valeurs de n et de p .

