Algorithmique et Programmation Parallèles
TD 3 –
Communications collectives bloquantes MPI

Exercice I: Produit scalaire

Soit la fonction :

```
double produit_scalaire( int N, double *a, double *b )
{
        double res = 0 ;
        for( int i = 0 ; i < N ; i++ )
                res += a[i] * b[i] ;
        return res ;
}
```

qui calcule le produit scalaire de deux vecteurs a et b de dimension N en séquentiel.

Question : En complétant le fichier `prod_scal/exo_prod_scal.c`, paralléliser la fonction produit_scalaire dans le cas où a et b sont des vecteurs distribués sur tous les processus MPI (N devient alors le nombre d'éléments associés au processus MPI appelant produit_scalaire).

Exercice II : Réduction

Question : Ecrire la fonction :

```
double reduction_somme( double in ) ;
```

qui retourne la somme de tous les in de tous les processus MPI :

- en utilisant des communications point à point bloquantes ;
- en utilisant des communications collectives autres que MPI_Reduce et MPI_Allreduce.

Tester cette fonction en reprenant l'exercice I.

Exercice III : Pièges sur les collectives

Dans le répertoire `pieges_coll/pieges/`, les fichiers suivants comportent des erreurs :

a) `piege_barriere.c`
b) `piege_scatter.c`
c) `piege_coll.c`

Expliquez les erreurs, apportez les corrections.

Exercice III : Implémentation d'un *broadcast*

Le programme `algo_bcast/exercice/mpi_bcast.c` prend en argument un entier n qui représente la taille en octets d'un tableau.
Seul le processus de rang 0 remplit ce tableau et le diffuse 100 fois aux autres processus en utilisant la fonction `MPI_Bcast`.

Question 1 : mesurer le temps pris par ce programme initial avec n=1000 pour des nombres respectifs de processus de 2, 4, 8, 16, 32, 48.

A présent, on désire implémenter nous-même notre propre communication collective mais en utilisant uniquement les communications point-a-point `MPI_Send` et `MPI_Recv`.

Question 2 : Implémenter l'**algorithme linéaire** (voir figure 1). Remplacer l'appel à `mpi_bcast` par une fonction **`linear_bcast`**. Faire les mesures de temps et les comparer au programme initial.

Question 3 : Implémenter le **premier algorithme en arbre binaire** (voir figure 2). Remplacer l'appel à `mpi_bcast` par une fonction **`btreev1_bcast`**. Faire les mesures de temps et les comparer au programme initial.

Question 4 : Implémenter le **deuxième algorithme en arbre binaire** (voir figure 3). Remplacer l'appel à `mpi_bcast` par une fonction **`btreev2_bcast`**. Faire les mesures de temps et les comparer au programme initial.

Question 5 : Implémenter un algorithme qui prenne en compte la topologie du cluster. Faire les mesures de temps et les comparer aux programme précédents. Pour ce faire :

a) **Créer des communicateurs** pour les processus qui appartiennent aux mêmes nœuds (utiliser la fonction **`MPI_Comm_split_type`** et la variable **`MPI_COMM_TYPE_SHARED`**).

b) Designer un processus maître par nœud et créer le communicateur de tous les processus maîtres

c) En s'appuyant sur le deuxième algorithme en arbre binaire, implémenter le *broadcast* en diffusant d'abord le tableau entre les nœuds puis en le diffusant à l'intérieur des nœuds.

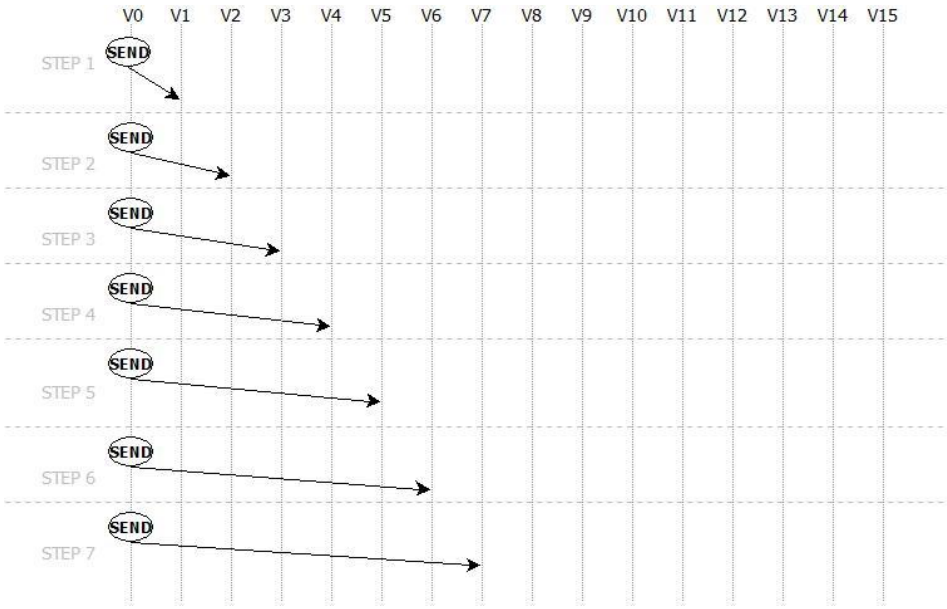

Fig 1 – Algorithme linéaire : For(1; i<p; i++) if(rank==0) send(data, i)

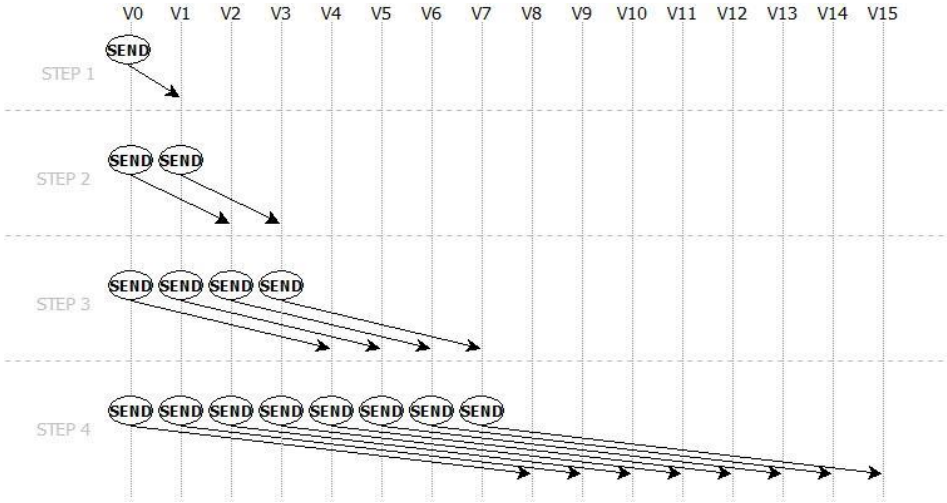

Fig 2 – Binary tree V1 : For(0; i<log(p); i++) if(rank<2^i) send(data, rank+2^i)

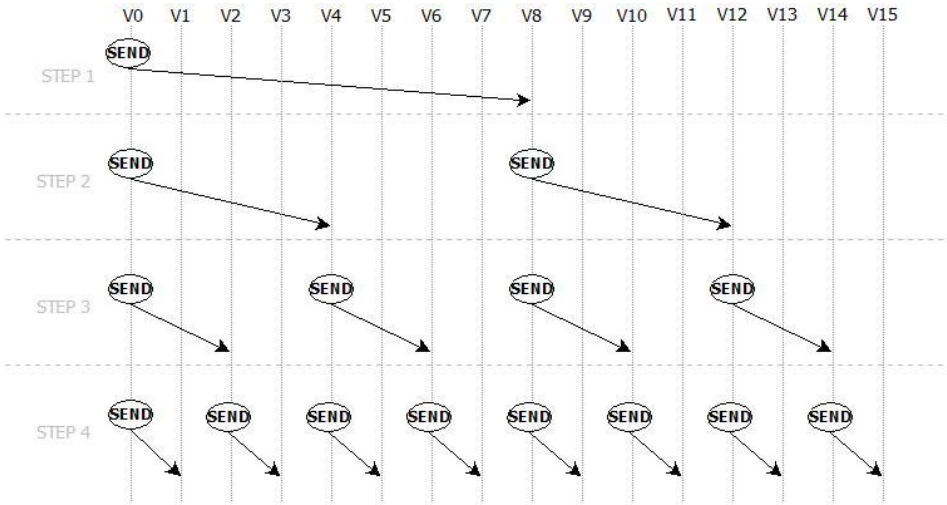

Fig 3 – Binary tree V2 : For(log(p); i>0; i--) if(rank%2^i ==0) send(data, rank+2^{i-1})