

1 Exercice : Comparaison d'implémentation de la mise-à-jour des mailles fantômes à l'aide de communications point-à-point et one-sided

Dans cette exercice, nous allons nous focaliser sur la mise-à-jour des mailles fantômes dans le code du jeu de la vie. Nous avons vu lors du TD précédent le principe de fonctionnement et implémenté un mécanisme de protection reprise. Pour rappel, **le domaine est découpé par bandes, ce qui fait que chaque processus a simplement un voisin au-dessus et un en-dessous**. Nous allons comparer trois implémentations :

- Utilisation classique des communications point-à-point
- Utilisation de communications one-sided avec des synchronisations collectives (MPI_Win_fence)
- Utilisation de communications one-sided avec le mécanisme minimal (post-start-complete-wait)

Les deux modes de synchronisations comparés dans ce TD correspondent à des communications de type "Active Target". En effet, le processus cible (qui détient la fenêtre où seront écrites les données) est actif dans la phase de communication : il appartient au groupe de processus dans l'appel collectif à MPI_Win_fence d'une part, et il expose explicitement la fenêtre (MPI_Win_post et MPI_Win_wait) d'autre part.

Informations diverses :

- Il est conseillé de charger un module MPI récent : `> module avail mpi, > module load mpi/...`
- La commande `> make` vous permet de compiler le code.
- Le code se lance de la manière suivante :
`> mpirun -n <nb_processes> mlife/mlife-mpiio -x <dim_x> -y <dim_y> -i <nb_iter>`
- exemple d'essai fonctionnel avec les communication point-à-point :
`> mpirun -n 4 ./mlife -x 50 -y 50 -i 20`
- exemple d'essai fonctionnel avec les communications one-sided fence :
`> mpirun -n 4 ./mlife-fence -x 50 -y 50 -i 20`
- exemple d'essai fonctionnel avec les communication one-sided pscw :
`> mpirun -n 4 ./mlife-pscw -x 50 -y 50 -i 20`

Question1 ()

[mlife-pt2pt.c] Nous commençons ici avec L'implémentation classique à base de communications point-à-point.

Compléter les TODO dans le fichier `mlife-pt2pt.c`.

Question2 ()

[mlife-fence.c] Nous allons maintenant utiliser les communications one-sided avec le mécanisme de synchronisation à l'aide de clôtures (fence).

Compléter les TODO dans le fichier `mlife-fence.c`. Expliquer les options passées à l'ouverture et la fermeture de "l'époque" avec les clôtures (fence)

Question3 ()

[mlife-pscw.c] Nous allons maintenant utiliser les communications one-sided avec le mécanisme de synchronisation minimal pscw.

Clompéter les TODO dans le fichier mlife-pscw.c. Expliquer la différence avec la méthode précédente.

2 Exercices : Révision des différents mécanismes MPI abordés dans ce cours

Correction du test.

Question1 ()

Topologie de processus.

(a) Creation du communicateur correspondant au graphe

```
1      int nnodes = 5;
2      int index[5] = {2,5,7,8,9};
3      int edges[index[nnodes-1]] = {1,2,0,2,4,0,3,2,3};
4
5      MPI_Graphe_create (comm_old,nnodes, index, edges,false,&
                        comm_graph);
```

(b) Poids des liens entre processus :

Un appel MPI_Neighbor_allgather sur le communicateur graphe avec envoie du rang permet à chaque processus de récupérer les rangs des voisins dont il doit recevoir des messages d'après le graphe.

Question2 ()

IO parallèles

```
1  Int buffer[8] = {rank, rank, rank, rank, rank, rank, rank, rank};
2  char * filename = "nom_de_fichier";
3  MPI_File * file_ptr;
4  MPI_Datatype datatmp, filetype;
5  MPI_Offset disp;
6  MPI_Aint lower_bound;
7  MPI_Aint extent;
8
9  if (rank==0)
10 {
11     MPI_Type_contiguous(1, MPI_INT, &datatmp);
12     lower_bound=0;
13     extent = 6*sizeof(int)
14     Int count = 2;
15     disp = 0;
16 }
17 elif (rank == 1)
18 {
19     MPI_Type_contiguous(2, MPI_INT, &datatmp);
20     lower_bound=-1*sizeof(int);
21     extent = 7*sizeof(int)
22     Int count = 8;
23     disp = 0;
24 }
25 elif (rank == 2)
26 {
27     MPI_Type_contiguous(2, MPI_INT, &datatmp);
28     lower_bound=-3*sizeof(int);
29     extent = 8*sizeof(int)
30     Int count = 6;
31     disp = 0;
32 }
33 elif (rank==3)
34 {
35     MPI_Type_contiguous(1, MPI_INT, &datatmp);
36     lower_bound=-6*sizeof(int);
37     extent = 7*sizeof(int)
38     Int count = 3;
39     disp = sizeof(int);
40 }
41
42 MPI_Type_create_resized(datatmp, lower_bound, extent, &filetype);
43 MPI_Type_commit(&filetype);
44
45 MPI_File_open(MPI_COMM_WORLD, filename, MPI_MODE_WRONLY |
46               MPI_MODE_CREATE | MPI_MODE_EXCL | MPI_MODE_UNIQUE_OPEN,
47               MPI_INFO_NULL, file_ptr);
48 MPI_File_set_view(file_ptr, disp, etype, filetype, "native",
49                   MPI_INFO_NULL);
50 MPI_File_write(file_ptr, &buffer, count, MPI_INT, MPI_STATUS_IGNORE);
51 MPI_File_close(file_ptr);
```