


| | | |
|---|---|--|
|  | EX - Examen n°1 | |
| Ndeye Arame Diago - Juan Angel Lorenzo | Architecture et programmation parallèle et réparation | |
| ING2-GSI-MI | Année 2018-2019 | |

Modalités

- Durée : 2 heures.
- Vous devez rédiger votre copie à l'aide d'un stylo à encre exclusivement.
- Toutes vos affaires (sacs, vestes, trousse, etc.) doivent être placées à l'avant de la salle.
- Aucun document n'est autorisé sauf la feuille des fonctions MPI fournie avec ce sujet.
- Aucune question ne peut être posée aux enseignants, posez des hypothèses en cas de doute.
- Aucune machine électronique ne doit se trouver sur vous ou à proximité, même éteinte.
- Aucun déplacement n'est autorisé.
- Aucun échange, de quelque nature que ce soit, n'est possible.

Questions de cours (7 points)

- 1 J'ai un code en OpenMP dans lequel chaque thread travaille sur des éléments différents d'un tableau de taille assez petite. Nous avons remarqué que le temps d'exécution est plus élevé pour une exécution en parallèle qu'en séquentiel. Quelle est la raison la plus probable pour ce comportement ? (1 point) ☐
- 2 Lorsque le code OpenMP suivant est exécuté, les identifiants des threads et les résultats montrés ne sont pas justes. Expliquez (1) pourquoi et (2) ce qu'il faut ajouter pour que le code fonctionne correctement (2 points) : ☐

```
1  #include <omp.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main (int argc, char *argv[]) {
6      int nthreads, i, tid;
7      float total;
8
9      #pragma omp parallel
10     {
11         tid = omp_get_thread_num();
12
13         if (tid == 0) {
14             nthreads = omp_get_num_threads();
15             printf("Number of threads = %d\n", nthreads);
16         }
17         #pragma omp barrier
18
19         total = 0.0;
20         #pragma omp for schedule(dynamic,10)
21         for (i=0; i<1000000; i++)
22             total = total + i*1.0;
23
24         printf ("Thread %d is done! Total= %e\n",tid,total);
25
26     } /* ** End of parallel region ** */
27 }
```

- 3 Expliquez la différence principale entre la fonction `MPI_Send()` classique et la fonction `MPI_Bsend()`. Dans quel cas l'utilisation de la deuxième fonction pourrait être plus avantageuse que la première? (2 points). ☐
- 4 Énumérez et expliquez le fonctionnement d'une fonction MPI qui permette de vérifier si un message est arrivé dans le récepteur sans besoin de le lire (2 points). ☐

Exercice 1 : Évaluation de la performance (3 points)

- 5 Supposons qu'un ordinateur passe la totalité de son temps à gérer un type particulier de calcul séquentiel. Après quelques efforts, nous arrivons à paralléliser ce calcul à 70%.
- a) Si le programme prenait à l'origine 100 secondes pour s'exécuter, combien de temps prendra la version parallèle dans un cluster de 4 nœuds? Considérez que chaque nœud est identique à l'ordinateur qui exécutait le code séquentiel.
 - b) Quelle est l'accélération (*Speedup*) obtenue par le code en passant de l'ancien au nouveau système (le cluster de 4 nœuds)? Et pour un cluster de 8 nœuds?
 - c) Quelle sera l'efficacité (*Efficiency*) du code dans le cluster de 4 nœuds? Et pour un cluster de 8 nœuds? Est-ce que c'est une bonne idée, économiquement parlant, d'exécuter le code dans le cluster de 8 nœuds?
- ☐

Exercice 2 : OpenMP (5 points)

- 6 Supposons un tableau **a** d'entiers de taille N qui sera partagé entre les différents threads d'un programme OpenMP avec $NUM_THREADS$ threads. Chaque thread devra calculer **la différence** par rapport à N des éléments des itérations qui leur sont affectées. Par exemple, le thread 0 calculera $N - a[0] - a[1] - a[2] \dots$. Ensuite, le master thread calculera le produit des éléments de ce tableau. Finalement, chaque thread montrera un message avec (1) la différence qu'il a calculée et (2) la valeur du produit calculée par le master. Écrivez un code OpenMP pour implémenter cette fonctionnalité. ☐

Exercice 3 : MPI (5 points)

7

Nous avons une matrice de données divisée en blocs carrés de taille identique et répartis entre 16 processeurs, comme montre l'image ci-dessous. Nous voulons que tous les processeurs de chaque ligne de blocs travaillent ensemble de la façon suivante : chaque processeur cherchera la valeur maximale de son bloc de données. Ensuite, ils enverront cette valeur au processeur de la première colonne de leur propre ligne (P_0 , P_4 , P_8 et P_{12} selon MPI_COMM_WORLD dans l'image), qui recevra et stockera les valeurs maximales (la sienne aussi) dans un tableau. Utilisez **des opérations collectives** et **des communicateurs** pour implémenter cette fonctionnalité en MPI.

| | X0 | X1 | X2 | X3 |
|----|----------|----------|----------|----------|
| Y0 | P0 0 | P1 0 | P2 0 | P3 0 |
| Y1 | P4 1 | P5 1 | P6 1 | P7 1 |
| Y2 | P8 2 | P9 2 | P10 2 | P11 2 |
| Y3 | P12 3 | P13 3 | P14 3 | P15 3 |

□



MPI Quick Reference in C

```
#include <mpi.h>
```

Environmental Management:

```
int MPI_Init(int *argc, char **argv[])
int MPI_Finalize(void)
int MPI_Initialized(int *flag)
int MPI_Finalized(int *flag)
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
int MPI_Abort(MPI_Comm comm, int errorcode)
double MPI_Wtime(void)
double MPI_Wtick(void)
```

Blocking Point-to-Point-Communication:

```
int MPI_Send (void* buf, int count,
MPI_Datatype datatype, int dest, int tag,
MPI_Comm comm)
```

Related: **MPI_Bsend**, **MPI_Ssend**, **MPI_Rsend**

```
int MPI_Recv (void* buf, int count,
MPI_Datatype datatype, int source, int
tag, MPI_Comm comm, MPI_Status *status)
```

```
int MPI_Probe (int source, int tag, MPI_Comm
comm, MPI_Status *status)
```

```
int MPI_Get_count (MPI_Status *status,
MPI_Datatype datatype, int *count)
```

Related: **MPI_Get_elements**

```
int MPI_Sendrecv (void *sendbuf, int
sendcount, MPI_Datatype sendtype, int
dest, int sendtag, void *recvbuf, int
recvcount, MPI_Datatype recvtype, int
source, int recvtag, MPI_Comm comm,
MPI_Status *status)
```

```
int MPI_Sendrecv_replace (void *buf, int
count, MPI_Datatype datatype, int dest,
int sendtag, int source, int recvtag,
MPI_Comm comm, MPI_Status *status)
```

```
int MPI_Buffer_attach (void *buffer, int
```

```
size)
```

```
int MPI_Buffer_detach (void *bufferptr, int
*size)
```

Non-Blocking Point-to-Point-Communication:

```
int MPI_Isend (void* buf, int count,
MPI_Datatype datatype, int dest, int tag,
MPI_Comm comm, MPI_Request *request)
```

Related: **MPI_Ibsend**, **MPI_Issend**, **MPI_Irsend**

```
int MPI_Irecv (void* buf, int count,
MPI_Datatype datatype, int source, int
tag, MPI_Comm comm, MPI_Request *request)
```

```
int MPI_Iprobe (int source, int tag, MPI_Comm
comm, int *flag, MPI_Status *status)
```

```
int MPI_Wait (MPI_Request *request,
MPI_Status *status)
```

```
int MPI_Test (MPI_Request *request, int
*flag, MPI_Status *status)
```

```
int MPI_Waitall (int count, MPI_Request
request_array[], MPI_Status
status_array[])
```

Related: **MPI_Testall**

```
int MPI_Waitany (int count, MPI_Request
request_array[], int *index, MPI_Status
*status)
```

Related: **MPI_Testany**

```
int MPI_Waitsome (int incount, MPI_Request
request_array[], int *outcount, int
index_array[], MPI_Status status_array[])
```

Related: **MPI_Testsome**,

```
int MPI_Request_free (MPI_Request *request)
```

Related: **MPI_Cancel**

```
int MPI_Test_cancelled (MPI_Status *status,
int *flag)
```

Collective Communication:

```
int MPI_Barrier (MPI_Comm comm)
```

```
int MPI_Bcast (void *buffer, int count,
```

```
MPI_Datatype datatype, int root, MPI_Comm
comm)
```

```
int MPI_Gather (void *sendbuf, int sendcount,
MPI_Datatype sendtype, void *recvbuf, int
recvcount, MPI_Datatype recvtype, int
root, MPI_Comm comm)
```

```
int MPI_Gatherv (void *sendbuf, int
sendcount, MPI_Datatype sendtype, void
*recvbuf, int recvcount array[], int
displ_array[], MPI_Datatype recvtype, int
root, MPI_Comm comm)
```

```
int MPI_Scatter (void *sendbuf, int
sendcount, MPI_Datatype sendtype, void
*recvbuf, int recvcount, MPI_Datatype
recvtype, int root, MPI_Comm comm)
```

```
int MPI_Scatterv (void *sendbuf, int
sendcount_array[], int displ_array[]
MPI_Datatype sendtype, void *recvbuf, int
recvcount, MPI_Datatype recvtype, int
root, MPI_Comm comm)
```

```
int MPI_Allgather (void *sendbuf, int
sendcount, MPI_Datatype sendtype, void
*recvbuf, int recvcount, MPI_Datatype
recvtype, MPI_Comm comm)
```

Related: **MPI_Alltoall**

```
int MPI_Allgatherv (void *sendbuf, int
sendcount, MPI_Datatype sendtype, void
*recvbuf, int recvcount_array[], int
displ_array[], MPI_Datatype recvtype,
MPI_Comm comm)
```

Related: **MPI_Alltoallv**

```
int MPI_Reduce (void *sendbuf, void *recvbuf,
int count, MPI_Datatype datatype, MPI_Op
op, int root, MPI_Comm comm)
```

```
int MPI_Allreduce (void *sendbuf, void
*recvbuf, int count, MPI_Datatype
datatype, MPI_Op op, MPI_Comm comm)
```

Related: **MPI_Scan**, **MPI_Exscan**

```
int MPI_Reduce_scatter (void *sendbuf, void
*recvbuf, int recvcount_array[],
MPI_Datatype datatype, MPI_Op op,
MPI_Comm comm)
```

```
int MPI_Op_create (MPI_User_function *func,
int commute, MPI_Op *op)
int MPI_Op_free (MPI_Op *op)
```

Derived Datatypes:

```
int MPI_Type_commit (MPI_Datatype *datatype)
int MPI_Type_free (MPI_Datatype *datatype)
int MPI_Type_contiguous (int count,
MPI_Datatype oldtype, MPI_Datatype
```

```

*newtype)

int MPI_Type_vector (int count, int
    blocklength, int stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)

int MPI_Type_indexed (int count, int
    blocklength_array[], int displ_array[],
    MPI_Datatype oldtype, MPI_Datatype
    *newtype)

int MPI_Type_create_struct (int count, int
    blocklength_array[], MPI_Aint
    displ_array[], MPI_Datatype
    oldtype_array[], MPI_Datatype *newtype)

int MPI_Type_create_subarray (int ndims, int
    size_array[], int subsize_array[], int
    start_array[], int order, MPI_Datatype
    oldtype, MPI_Datatype *newtype)

int MPI_Get_address (void *location, MPI_Aint
    *address)

int MPI_Type_size (MPI_Datatype *datatype,
    int *size)

int MPI_Type_get_extent (MPI_Datatype
    datatype, MPI_Aint *lb, MPI_Aint *extent)

int MPI_Pack (void *inbuf, int incout,
    MPI_Datatype datatype, void *outbuf, int
    outcount, int *position, MPI_Comm comm)

int MPI_Unpack (void *inbuf, int insize, int
    *position, void *outbuf, int outcount,
    MPI_Datatype datatype, MPI_Comm comm)

int MPI_Pack_size (int incout, MPI_Datatype
    datatype, MPI_Comm comm, int *size)

Related: MPI_Type_create_hvector,
    MPI_Type_create_indexed_block,
    MPI_Type_create_darray,
    MPI_Type_create_resized,
    MPI_Type_get_true_extent, MPI_Type_dup,
    MPI_Pack_external, MPI_Unpack_external,
    MPI_Pack_external_size

```

Groups and Communicators:

```

int MPI_Group_size (MPI_Group group, int
    *size)

int MPI_Group_rank (MPI_Group group, int
    *rank)

int MPI_Comm_group (MPI_Comm comm, MPI_Group
    *group)

```

```

int MPI_Group_translate_ranks (MPI_Group
    group1, int n, int rank1_array[],
    MPI_Group group2, int rank2_array[])

int MPI_Group_compare (MPI_Group group1,
    MPI_Group group2, int *result)

MPI_IDENT, MPI_CONGRUENT, MPI_SIMILAR,
MPI_UNEQUAL

int MPI_Group_union (MPI_Group group1,
    MPI_Group group2, MPI_Group *newgroup)

Related: MPI_Group_intersection,
    MPI_Group_difference

int MPI_Group_incl (MPI_Group group, int n,
    int rank_array[], MPI_Group *newgroup)

Related: MPI_Group_excl

int MPI_Comm_create (MPI_Comm comm, MPI_Group
    group, MPI_Comm *newcomm)

int MPI_Comm_compare (MPI_Comm comm1,
    MPI_Comm comm2, int *result)

MPI_IDENT, MPI_CONGRUENT, MPI_SIMILAR,
MPI_UNEQUAL

int MPI_Comm_dup (MPI_Comm comm, MPI_Comm
    *newcomm)

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

int MPI_Comm_free (MPI_Comm *comm)

```

Topologies:

```

int MPI_Dims_create (int nnodes, int ndims,
    int *dims)

int MPI_Cart_create (MPI_Comm comm_old, int
    ndims, int dims_array[], int
    periods_array[], int reorder, MPI_Comm
    *comm_cart)

int MPI_Cart_shift (MPI_Comm comm, int
    direction, int disp, int *rank_source,
    int *rank_dest)

int MPI_Cartdim_get (MPI_Comm comm, int
    *ndim)

int MPI_Cart_get (MPI_Comm comm, int naxdim,
    int *dims, int *periods, int *coords)

int MPI_Cart_rank (MPI_Comm comm, int
    coords_array[], int *rank)

int MPI_Cart_coords (MPI_Comm comm, int rank,
    int maxdims, int *coords)

```

```

int MPI_Cart_sub (MPI_Comm comm_old, int
    remain_dims_array[], MPI_Comm *comm_new)

int MPI_Cart_map (MPI_Comm comm_old, int
    ndims, int dims_array[], int
    periods_array[], int *new_rank)

int MPI_Graph_create (MPI_Comm comm_old, int
    nnodes, int index_array[], int
    edges_array[], int reorder, MPI_Comm
    *comm_graph)

int MPI_Graph_neighbors_count (MPI_Comm comm,
    int rank, int *nneighbors)

int MPI_Graph_neighbors (MPI_Comm comm, int
    rank, int maxneighbors, int *neighbors)

int MPI_Graphdims_get (MPI_Comm comm, int
    *nnodes, int *nedges)

int MPI_Graph_get (MPI_Comm comm, int
    maxindex, int maxedges, int *index, int
    *edges)

int MPI_Graph_map (MPI_Comm comm_old, int
    nnodes, int index_array[], int
    edges_array[], int *new_rank)

int MPI_Topo_test (MPI_Comm comm, int
    *topo_type)

```

Wildcards:

```

MPI_ANY_TAG, MPI_ANY_SOURCE

```

Basic Datatypes:

```

MPI_CHAR, MPI_SHORT, MPI_INT, MPI_LONG,
MPI_UNSIGNED_CHAR, MPI_UNSIGNED_SHORT,
MPI_UNSIGNED, MPI_UNSIGNED_LONG, MPI_FLOAT,
MPI_DOUBLE, MPI_LONG_DOUBLE, MPI_BYTE,
MPI_PACKED

```

Predefined Groups and Communicators:

```

MPI_GROUP_EMPTY, MPI_GROUP_NULL,
MPI_COMM_WORLD, MPI_COMM_SELF, MPI_COMM_NULL

```

Reduction Operations:

```

MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD,
MPI_BAND, MPI_BOR, MPI_BXOR, MPI_LAND,
MPI_LOR, MPI_LXOR

```

Status Object:

```

status.MPI_SOURCE, status.MPI_TAG,
status.MPI_ERROR

```