

	EX - Examen n°1	
Rachid Chelouah - Juan Angel Lorenzo	Programmation Système, Parallèle, Réseau et Distribuée.	
ING2-GSI	Année 2016-2017	

Modalités

- Durée : 3 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.
- Aucune sortie n'est autorisée avant une durée incompressible d'une heure.
- Aucun déplacement n'est autorisé.
- Aucun échange, de quelque nature que ce soit, n'est possible.

Questions courtes (5 points)

- ① | Quel est le problème avec le programme suivant ? Modifiez le code pour qu'il fonctionne correctement. ☐

```
1  # include <omp.h>
2  # include <stdlib.h>
3  # include <time.h>
4  # include <iostream>
5  using namespace std;
6
7  int main () {
8      const int n = 4;
9      float b[] = {1.0,2.0,3.0,4.0};
10     float a[] = {0,0,0,0};
11
12     srand (time(NULL));
13     omp_set_num_threads(4);
14
15     #pragma omp parallel
16     for (int i=0;i<n;i++)
17         a[i] = b[i] * rand();
18
19     return 0;
20 }
```

- ② | Expliquez la différence entre les tubes anonymes et nommés (*unnamed* et *named pipes*). ☐

- ③ | Quel est l'inconvénient d'utiliser une variable unique (verrou ou *lock*) pour résoudre le problème de l'exclusion mutuelle ? ☐

- ④ | De quelle façon un processus peut savoir si un message dans une file de messages est destiné à lui ? ☐

- ⑤ | Expliquez la stratégie de partage du travail `#pragma omp for schedule(dynamic [, chunk])`. Dans quelles situations peut-il être avantageux de prendre cette stratégie ? ☐

Exercice 1 : Héritage d'un tube ordinaire (3 points)

Écrivez un programme qui crée des tubes ordinaires (unammed pipes) puis un fils. Successivement, le père et le fils s'échangent des messages ("*Salut PAPA*" et "*Salut Fils*").

Exercice 2 : MPI (3 points)

Écrivez un programme MPI qui utilise des opérations point-à-point pour calculer le produit matrice-vecteur $A \times x = y$. La matrice A et le vecteur x doivent être initialisés par le processus master et envoyés aux autres processus. Utilisez une stratégie de découpage de A par tranches. À la fin du produit, chaque processus devra envoyer sa partie du résultat au master, qui rassemblera tout dans le vecteur y .

Exercice 3 : OpenMP (3 points)

Supposons un tableau *tab* d'entiers de taille n rempli aléatoirement et une pile remplie avec des valeurs entiers. Nous avons à notre disposition une fonction `depile()` qui depile et retourne la première valeur disponible dans la pile ou -1 si la pile est vide. Supposons un programme qui, en parcourant le tableau, extrait une valeur x de la pile et met à jour chaque case du tableau avec le produit de la valeur de la case et x . On répète cette procédure jusqu'à ce que la pile est vide. Écrivez un code OpenMP avec P threads, tel que $P \leq n$, qui parallélise ce traitement.

Exercice 4 : Carré des éléments d'un tableau (6 points)

Supposons un programme MPI qui calcule le carré de chaque élément d'un tableau d'entiers de la façon suivante :

- Une fois crée et rempli, le processus *master* partage le tableau entre tous les processus (cela inclut lui-même). Le nombre d'éléments du sous-tableau envoyé à chaque processus sera différent pour chacun et généré aléatoirement.
- Ensuite, chaque processus remplace chaque élément du sous-tableau reçu par sa valeur carrée.
- Finalement, le *master* calculera la somme de toutes les valeurs carrées de chaque processus.

Écrivez le code MPI qui implémente ce traitement. Utilisez des opérations collectives lorsque cela sera possible.



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_hindexed,
    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_COMGRUENT, 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_COMGRUENT, 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

```