EISTI Ecole Internationale des Sciences du Traitement de l'Information	**EXAMEN DE RATTRAPAGE**

Rachid CHELOUAH – Esma TALHI – Juan Angel LORENZO	Architecture et programmation parallèle et répartie
ING2-GSI-MI	Année : 2017 – 2018

Modalités

- Durée : 2 heures.
- Vous devez rédiger votre copie à l'aide d'un stylo à encre exclusivement.
- Toutes vos affaires (sacs, vestes, trousses, etc.) doivent être placées à l'avant de la salle.
- Aucun document n'est autorisé sauf la feuille des fonctions MPI fournies avec le sujet.
- Tous les codes peuvent être rédigés en C ou C++.
- 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 (5,5 points)

1. Quels sont les différentes politiques d'ordonnancement des itérations en OpenMP ? Expliquez le principe de chacune d'entre elles
2. Expliquez la différence entre :
   `#pragma omp single`
   `#pragma omp critical`

3. Expliquez la différence entre les fonctions collectives `MPI_scatter` et `MPI_Scatterv`.

4. En exécutant le programme ci-dessous avec la commande suivante :
   `mpirun -np 4 ./prog` ce dernier renvoie une erreur. Selon vous quel est le problème et que faut-il corriger pour que le programme fonctionne correctement

```
1   #include "mpi.h"
2   #include <stdio.h>
3   #include <stdlib.h>
4
5   int main (int argc, char *argv[])
6   {
7     int numtasks, rank, tag=1, alpha, i;
8     float beta;
9
10    MPI_Status stats;
11
12    MPI_Init(&argc,&argv);
13    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
14    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
15
16    if (rank == 0) {
17     if (numtasks > 2)
18      printf("Numtasks=%d. Only 2 needed.Ignoring extra...\n",numtasks);
19      for (i=0; i<10; i++) {
20        alpha = i*10;
21        MPI_send(&alpha, 1, MPI_INT, 1, tag, MPI_COMM_WORLD);
22        printf("Task %d sent = %d\n",rank,alpha);
23      }
24    }
25
26    if (rank == 1) {
27     for (i=0; i<10; i++) {
28      MPI_recv(&beta, 1, MPI_BYTE, 0, tag, MPI_COMM_WORLD, &stats);
29      printf("Task %d received = %f\n",rank,beta);
30     }
31    }
32
33  MPI_Finalize();
34  }
```

Exercice 1 : Le Ping Pong en MPI (4 points)

Ecrire le programme MPI qui permet à deux processus d'échanger un nombre, qu'ils incrémentent avant chaque envoi, jusqu'à atteindre une valeur donnée.

Exercice 2 : La trace d'une matrice en OpenMP (4 points)

Soit A une matrice carrée n x n. La trace de A est la somme des éléments diagonaux de A. Ecrire le programme OpenMP qui permet de calculer la trace d'une matrice.

Exercice 3 : Nombres premiers (6,5 points)

Le crible d'Eratosthène est une méthode de recherche des nombres premiers plus petits qu'un entier naturel N donné. L'algorithme procède par élimination : il s'agit de supprimer d'une table d'entiers de 2 à N tous les multiples d'un entier. En supprimant tous les multiples, à la fin il ne restera que les entiers qui ne sont multiples d'aucun entier, et qui sont donc les nombres premiers. Nous souhaitons compter combien y-a-t-il de nombres premiers 2 à N.

1. Donnez le code séquentiel.
2. Parallélisez ce code en utilisant l'API OpenMP. Expliquez la stratégie suivie.
3. Parallélisez le code séquentiel en utilisant MPI. Expliquez la stratégie suivie ainsi que le choix de modes de communications : point à point ou collectives.

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

Related: `MPI_Get_elements`

```
int MPI_Sendrecv (const 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 *buffer_addr, int
    *size)
```

Non-Blocking Point-to-Point-Communication:

```
int MPI_Isend (const 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 (const 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 (const void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
int MPI_Gatherv (const void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, const int recvcount_array[],
    const int displ_array[], MPI_Datatype
    recvtype, int root, MPI_Comm comm)
int MPI_Scatter (const void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
int MPI_Scatterv (const void *sendbuf, const
    int sendcount_array[], const int
    displ_array[], MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
int MPI_Allgather (const void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, MPI_Comm comm)
```

Related: `MPI_Alltoall`

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

Related: `MPI_Alltoallv`

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

Related: `MPI_Scan, MPI_Exscan`

```
int MPI_Reduce_scatter (const void *sendbuf,
    void *recvbuf, const 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, const int blocklength_array[], const int displ_array[], MPI_Datatype oldtype, MPI_Datatype *newtype)

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

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

int **MPI_Get_address** (const 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** (const void *inbuf, int incount, MPI_Datatype datatype, void *outbuf, int outcount, int *position, MPI_Comm comm)

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

int **MPI_Pack_size** (int incount, 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, const int ranks1[], MPI_Group group2, const int ranks2[])

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, const int ranks[], 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, const int dims[], const int periods[], 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 maxdims, int dims[], int periods[], int coords[])

int **MPI_Cart_rank** (MPI_Comm comm, const int coords[], int *rank)

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

int **MPI_Cart_sub** (MPI_Comm comm_old, const int remain_dims[], MPI_Comm *comm_new)

int **MPI_Cart_map** (MPI_Comm comm_old, int ndims, const int dims[], const int periods[], int *new_rank)

int **MPI_Graph_create** (MPI_Comm comm_old, int nnodes, const int index[], const int edges[], 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, const int index[], const int edges[], int *new_rank)

int **MPI_Topo_test** (MPI_Comm comm, int *status)

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