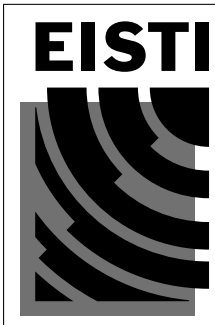	**EX - Examen n°1**
Rachid Chelouah - Juan Angel Lorenzo	Architecture et programmation parallèle et répartie
ING2-MI	Année 2016–2017

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.

— 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 (4 points)

(1) On a récupéré le code `mpicode.cpp` ci-dessous qui envoie certains éléments d'un tableau d'un émetteur à plusieurs récepteurs. Si on exécute le code avec la commande '`mpirun -np 4 mpicode`' le programme donne une erreur. Pourquoi? Qu'est-ce qu'il faut corriger et quelle sera la sortie une fois le code corrigé? □

```
1  # include <mpi.h>
2  # include <iostream>
3  # define SIZE 4
4
5  using namespace std;
6
7  int main(int argc , char *argv []) {
8     int numProc;
9     int tag = 30;
10    // en C MPI_Init(&argc, &argv);
11    MPI::Init(argc ,argv);
12    // en C MPI_Comm_rank(MPI_COMM_WORLD, &numProc)
13    numProc = MPI::COMM_WORLD.Get_rank();
14
15    if ( numProc == 0) {
16       int values[SIZE] = {1,2,3,4};
17       for (int i=0;i<SIZE;i++){
18             cout <<"proc "<< numProc << " values["<<i <<"] = "<< values[i] << endl;
19             // en C  MPI_Send(values, i+1,MPI_INT, i+1, tag)
20             MPI::COMM_WORLD.Send(values ,i+1, MPI::INT ,i+1, tag);
21       }
22    } else {
23       int rxval[SIZE];
24       // en C MPI_Receive(rxval ,numProc, MPI_INT,0, MPI_ANY_TAG, MPI_COMM_WORLD, &status)
25       MPI::COMM_WORLD.Recv(rxval ,numProc, MPI::INT,0, MPI::ANY_TAG);
26
27       cout <<"proc "<< numProc <<" a recu le vecteur ";
28       for(size_t j = numProc; j--;) cout << rxval[j] << " ";
29       cout << endl;
30    }
31    // en C MPI_Finalize()
32    MPI::Finalize();
33    return 0;
34 }
```

(2) Expliquez la différence entre '`#pragma omp master`' et '`#pragma omp single`'. □

(3) Dans le code OpenMP suivant, qu'est ce qu'il faut ajouter pour que le programme fonctionne correctement ? Quel sera le contenu correct du vecteur **a** à la fin de la fonction `addArray` ? □

```
1  # include <omp.h>
2  # include <iostream>
3  using namespace std;
4
5  void addArray (int n, int *a, int *b)
6  {
7      int i;
8      #pragma omp parallel
9      {
10         #pragma omp for
11         for (i=0; i<n-1; i++)
12             a[i] = b[i] + b[i+1];
13     }
14     a[i]=b[i];
15 }
16
17 int main(int argc , char *argv []) {
18     const int n = 4;
19     int b[n] = {1,2,3,4};
20     int a[n] = {0,0,0,0};
21
22     addArray(n,a,b);
23 }
```

(4) Dans une opération MPI de partage des données, qui doit appeler la fonction `MPI_Gather()` ? □

Exercice 1 : OpenMP (4 points)

Écrivez un programme OpenMP qui calcule le produit scalaire de deux tableaux a et b de taille n. Chaque thread devra calculer la somme partielle de ses éléments. À la fin, le master thread devra montrer le résultat final du produit scalaire.

Exercice 2 : MPI (4 points)

Supposons N processus, numérotés de 0 à $N-1$ et configurés en anneau. C'est-à-dire, le noeud x reçoit des données du noeud $x-1$ et envoie des données au noeud $x+1$. À la fin de l'anneau, le noeud $N-1$ reçoit des données du $N-2$ et envoie des données au noeud 0.

Écrivez un programme MPI qui passe un message dans l'anneau de la façon suivante : tout au début du programme, le processus master générera un message (un entier positif). Le message devra passer pour chaque processus de l'anneau. Chaque fois que le message retourne au processus master, la valeur du message sera décrémentée. Lorsque la valeur du message reçue par un processus soit 0, le processus enverra cette valeur au processus suivant et terminera sa exécution.

Note : *Ce n'est pas nécessaire d'écrire en toute exactitude les paramètres des commandes MPI utilisées, tant que la syntaxe soit assez claire pour expliquer les données envoyées et/ou reçues, l'émetteur et le destinataire.*

Exercice 3 : Produit de matrices (8 points)

Nous voulons calculer le produit de deux matrices carrées $A_n \times B_n$.

— Donnez le code séquentiel.

— Parallélisez ce code en utilisant l'API OpenMP. **Expliquez clairement** la stratégie suivie pour paralléliser le code.

— Paralléliser le code séquentiel en utilisant MPI. **Expliquez clairement** la stratégie suivie pour paralléliser le code ainsi que le choix des opérations MPI point à point et/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** (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 incount, 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 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, 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