EISTI Ecole Internationale des Sciences du Traitement de l'Information	**ING2 GSI – MI : EXAMEN DE RATTRAPAGE** **ARCHITECTURE ET PROGRAMMATION PARALLÈLE**
Rédigé par : équipe pédagogique du cours d'Architecture et Programmation Parallèle	*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.

Exercice 1 : Questions de cours (7 points)

1. Expliquer la différence entre les modèles de programmation parallèle Openmp et MPI (1 point)

2. Les directives **critical** et **atomic** peuvent être utilisées pour protéger la mise à jour d'une variable partagée. Expliquer la différence principale entre ces deux directives et donner un exemple (2 points).

3. Expliquer les différences entre les clauses **private, firstprivate et lastprivate** (2 points).

4. Expliquez la différence entre les communications collectives MPI **Radiation (Broadcast)** et la **Réduction** (2 points).

Exercice 2 : (5 points)

Dans cet exercice, nous allons utiliser OpenMP pour paralléliser un programme calculant π. π peut être calculé simplement par intégration :

$$\pi = \int_0^1 f(x)dx \text{ avec } f(x) = \frac{4}{1+x^2}$$

Une approximation de cette relation est :

$$\pi = h\sum_{i=1}^{n} f(x_{i-1/2}) \text{ avec } h = \frac{1}{n} \text{ et } x_{i-1/2} = \frac{i-1/2}{n}$$

Écrire le programme parallèle du calcul de π. Chaque thread calcule une partie de la somme qu'il stocke dans un tableau partagé à l'indice correspondant à son rang. La somme totale est calculée par le thread maître après la région parallèle.

Exercice 3 : (4 points)

Il s'agit d'un anneau de communication. Écrire un programme MPI où parmi n processus, le processus de rang r reçoit la valeur 1000 + (r − 1) du processus de rang r − 1, 1 ≤ r ≤ n − 1, et où le processus de rang 0 reçoit la valeur 1000 + (n − 1) du processus de rang n − 1.

Exercice 4 : (4 points)

Soit A une matrice carrée réelle de taille (n, n) diagonale par blocs. Écrire un programme MPI où on affecte à chaque processus de rang r un bloc diagonal de taille p dont tous les coefficients ont pour valeur r, chaque processus de rang r calcule sa trace locale. Le processus de rang 0 récupère toutes les traces locales pour calculer la trace globale de la matrice. Soit *np* le nombre de processus.
NB : La trace d'une matrice A est la somme des éléments diagonaux de A.

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