38/40

CS 5334/4390 Spring 2017
Shirley Moore, Instructor
Exam 2 Part 2 Retake 40 points

Name Sheikh Tanveer Hossain

You may use notes for this exam. You may not use any books, computers, PDAs, cellphones, or other electronic aids. Please show all work and explain your answers fully. Each problem is worth 20 points. See page 3 for MPI syntax.

1. Assume you have 16 processes in MPI_COMM_WORLD and arrays A and B each containing 64 double precision floating point numbers.
   a) Write the MPI statements that create a 2D Cartesian communicator comm2d for the 16 processes.

int ndims, reorder; MPI_Comm Comm2d; int dims[2] period[2];
ndims=2; reorder=0; dims[0]=4; dims[1]=4; Period[0]=0; Period[1]=0;
MPI_Cart_Create(MPI_Comm_World, ndims, dims, periods, reorder
Comm 2d,);

   b) Write the MPI statements to create 1D row and column subcommunicators named rowcomm and colcomm, respectively.

MPI_Comm rowcomm, Colcomm;
int remain_dims [2]; remain_dims[0]=0; remain_dims[1]=1;
MPI_Cart_sub (Comm 2d, remain_dims; rowcomm);
remain_dims[0]=1, remain_dims[1]=0;
MPI_Cart_sub (Comm 2d, remain_dims; colcomm);

   c) Assume that A and B have been distributed in a 2D block distribution to the processes in comm2d. Write the MPI collective communication statements that will distribute the blocks of A in each row of comm2d to all the processes in that row and the blocks of B in each column of comm2d to all the processes in that column.

* Consider that the processors kept their respective value of A and B in double precision variable named a and b respectively after distribution.

⇒ Double ~~*recv~~ *recvA, * recvB;
recv = malloc(recv, 4* sizeof(double));
MPI_Allgather(a, 1, MPI_Double, recvA, 1, MPI_Double,
rowcomm)

-2 Each process has a 2x2 block of A and a 2x2 block of B, so the MPI_Allgathers need to send data of length 4, not 1.

(P.T.O)

MPI_Allgather(b, 1, MPI_Double, recvB, 1, MPI_Double, col comm)

2. Both NVIDIA GPUs and Intel Xeon Phi processors can be thought of as accelerators or co-processors. Compare and contrast the two in terms of the programming model, execution model, and memory system. and contrast

The compereison is given below:

~~Co-processors~~ ~~NVIDIA GPUs~~

~~Relatively 1 single core~~ of

Xeon Phi processors	NVIDIA GPUs
(1) One single core of xeon phi processors are more powerfull then GPU	(1) A single GPU thread is relatively less power full.
(2) The number of cores is around 72	(2) Thousands of cores available.
(3) Good for Multiple data Multiple instruction.	(3) Good for Multiple data single instruction.
(4) Have acess to 16 GB of high-bandwidth memory.	(4) No such option is available
(5) Can store relatively large private memory	(5) Relatively low private memory.
(6) 2D mesh interconnect system helps to do foster communication.	(6) No such option Available.
(7) Vectorization system available	(7) Vectorization system is not available.
(8) ~~For so doin~~ Good for doing Moderate complex calculation in large number	(8) Good for doing simple operation in large number.

(P.T.O)

9. The code is not needed to be~~e~~ changed very much.

10. The total code ~~is~~ can be run on Xeon-phi processor.

11. No seperate librery is needed.

12. We can run the MPI, open-MPI codes here

9. The part we want to run in GPU is needed to be edited rigorously.

10. Only ~~the~~ ~~GPU~~ a part of the code is ren on GPU.

11. We use CUDA librery to use the GPU

12. The code must be written with CUDA commands No MPI, open MPI allowed.

MPI function C syntax:

```
int MPI_Comm_size(MPI_Comm comm, int *size)

int MPI_Comm_rank(MPI_Comm comm, int *rank)

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

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

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

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

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_Wait(MPI_Request *request, MPI_Status *status)

int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype,
    int root, MPI_Comm comm)

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)

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_Scatter(void *sendbuf, int sendcount, 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)

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_sub(MPI_Comm comm, const int remain_dims[], MPI_Comm *comm_new)

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

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