TEXAS STATE UNIVERSITY

The rising STAR *of Texas*

CS 7389D: Scalable Supercomputing
Lab 3

Objective

Program distributed memory machines using MPI collectives.

Submission Instructions

Please create a folder called "`Lab3`" and commit the following deliverables in the repo.

Deliverable: `compute_stddev.c`

Information

MPI_Reduce

MPI_Reduce takes an array of input elements on each process and returns an array of output elements to the root process. The output elements contain the reduced result. Figure 1 pictorially explains how calling MPI_Reduce by all processes results in the application of element-wise MPI_Op operation in an array. The exact function prototype for MPI_Reduce looks like this:

```
MPI_Reduce(
void* send_data,
void* recv_data,
int count,
MPI_Datatype datatype,
MPI_Op op,
int root,
MPI_Comm communicator)
```

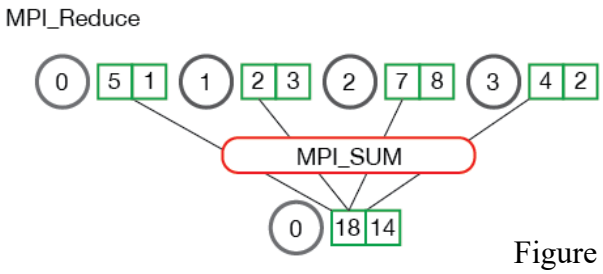

Figure 1

The reduction operations defined by MPI include:

MPI_MAX - Returns the maximum element.
MPI_MIN - Returns the minimum element.
MPI_SUM - Sums the elements.
MPI_PROD - Multiplies all elements.
MPI_LAND - Performs a logical and across the elements.
MPI_LOR - Performs a logical or across the elements.
MPI_BAND - Performs a bitwise and across the bits of the elements.
MPI_BOR - Performs a bitwise or across the bits of the elements.
MPI_MAXLOC - Returns the maximum value and the rank of the process that owns it.
MPI_MINLOC - Returns the minimum value and the rank of the process that owns it.

DEPARTMENT OF COMPUTER SCIENCE
601 University Drive | Building and Suite | San Marcos, Texas 78666-XXXX
phone: 512.245.XXXX | *fax:* 512.245.XXXX | WWW.TXSTATE.EDU

MPI_Allreduce

Many parallel applications will require accessing the reduced results across all processes rather than the root process. MPI_Allreduce will apply MPI_Op operator on every element of an array across all processes and distribute the results to all processes (compared to only process 0 in MPI_Reduce). The exact function prototype is the following:

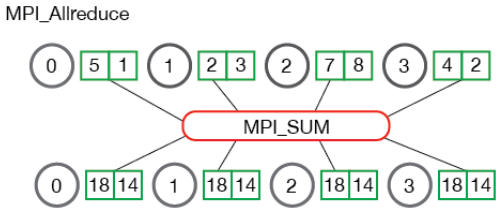

Figure 2

```
MPI_Allreduce(
   void* send_data,
   void* recv_data,
   int count,
   MPI_Datatype datatype,
   MPI_Op op,
   MPI_Comm communicator)
```

Note that, MPI_Allreduce is identical to MPI_Reduce with the exception that it does not need a root process id (since the results are distributed to all processes). Figure 2 illustrates the communication pattern of MPI_Allreduce.

Task: Finding standard deviation of a distributed set of numbers

This lab provides a template called compute_stddev.c that takes as input an integer *N* – the number of elements per process. Each MPI process then generates an array of *N* random numbers.

Modify compute_stdev.c to compute the average and the standard deviation of all numbers across all processes. Standard deviation is a measure of the dispersion of numbers from their mean. A lower standard deviation means that the numbers are closer together and vice versa for higher standard deviations. To find the standard deviation, one must first compute the average of all numbers. After the average is computed, the sums of the squared differences from the average are computed. The square root of the average of the sums is the final result. Following is a sample command of how to run your program:

```
Lab3$ mpirun -np 4 ./compute_stddev 10
```

Most commonly used functions

Function	Description
int MPI_Init(int *argc, char **argv)	Initialize MPI
int MPI_Finalize()	Exit MPI
int MPI_Comm_size(MPI_Comm comm, int *size)	Determine number of processes within a comm
int MPI_Comm_rank(MPI_Comm comm, int *rank)	Determine process rank
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)	Send a blocking message
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int src, int tag, MPI_Comm comm, MPI_Status *status)	Receive message
MPI_Isend (void *buf, int count, MPI_Datatype dtype, int dest, int tag, MPI_Comm comm, MPI_Request *req)	Non-blocking send message
MPI_Recv (void *buf, int count, MPI_Datatype dtype, int source, int tag, MPI_Comm comm, MPI_Request *req)	Non-blocking receive message
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)	Root sends data to all other processes in the same

	communicator. Must be called by all processes
int MPI_Gather(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm)	One process gathers data from all other processes in the same communicator. Must be called by all processes
int MPI_Allgather(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int recvcnt, MPI_Datatype recvtype, MPI_Comm comm)	All the processes collect data from all other processes in the same communicator. Must be called by all processes
int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)	One process collects all the data and performs an operation (MPI_SUM, MPI_MIN, MPI_MAX, MPI_PROD, logical AND, OR, XOR etc.)
int MPI_Allreduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)	All the processes collect data from all other processes and perform an operation
MPI_Barrier(MPI_Comm comm)	Synchronize all processes in the same communicator at this point

References:

1. **https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce/**
2. https://www.tacc.utexas.edu/systems/user-services