CS 484 FA17
Midterm 2, November 9th, 2017
Total time: 75 minutes No outside materials or devices allowed

Name: ____________________________

NetID: ____________________________

<table>
<thead>
<tr>
<th>Points</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>T/F, Multiple Choice</td>
<td>/ 10</td>
</tr>
<tr>
<td>Short Answer</td>
<td>/ 20</td>
</tr>
<tr>
<td>Code Comprehension 1</td>
<td>/ 10</td>
</tr>
<tr>
<td>Code Comprehension 2</td>
<td>/ 10</td>
</tr>
<tr>
<td>Communication Cost</td>
<td>/ 20</td>
</tr>
<tr>
<td>Code Writing</td>
<td>/ 15</td>
</tr>
<tr>
<td>Isoefficiency</td>
<td>/ 10</td>
</tr>
<tr>
<td>Total</td>
<td>/ 95</td>
</tr>
</tbody>
</table>
1) True or False / Multiple Choice (10 pts)

a) MPI processes are spawned when MPI_Init is called. (True / False)

b) If an MPI program is executed on a single processor machine using multiple processes, the processes can communicate with each other using global variables. (True / False)

c) The major benefit of message passing parallel programming model (MPI) over shared memory parallel programming model (OpenMP) is that the programs written using the former in general have better scalability. (True / False)

d) Which of the following MPI function is analogous to omp_get_thread_num()?
   i) MPI_Cart_create()
   ii) MPI_Comm_rank()
   iii) MPI_Comm_size()
   iv) MPI_Comm_split()

e) In MPI, a single process may belong to multiple communicators. (True / False)

f) In part 2 of MP3, load balancing has been added to the program. What did Charm++ load balancer migrate in this assignment?
   i) Particles in overloaded chares to underloaded chares
   ii) Particles in overloaded processors to underloaded processor
   iii) Chares in overloaded processors to underloaded processor
   iv) Chares in overloaded particles to underloaded particles

g) In Charm++, a chare that is an element of a chare array must remain on the same processor where it was initially placed by the runtime system throughout the execution of the program. (True / False)

h) Charm++ interface file must contain nothing but legal C++ code. (True / False)

i) Charm++ Structured Dagger can be used in any C++ function (True / False)

j) Create your own non-trivial true or false question that relates to our course material covered after the Midterm1 and provide an answer for it.
2) Short Answers (20 pts)

2.1) What do isoefficiency functions inform us about a given computation? Let $W$ be the work and $P$ be the number of processors. Is it better for $W$ to grow rapidly with respect to $P$ (example $W = P^3$) or slowly ($W = \log P$)?

2.2) How is MPI_Send different from MPI_Isend? Provide a brief explanation on what each guarantees when each call returns and also describe the usage with matching receiver.
2.3) Provide a short description of each MPI collectives: MPI_Reduce, MPI_Bcast, MPI_Gather and MPI_Scatter.

2.4) Briefly describe what PUP framework is for Charm++ and give an explanation on what it does.
3) Code comprehension 1 (10 pts)

Consider the following MPI code.

```c
int nprocs, myRank, len, buffer, root, count;
char hostname[MPI_MAX_PROCESSOR_NAME];

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myRank);

root = 0;
buffer = 0;

if(!myRank)
   buffer = 15;

count = nprocs;

MPI_Bcast(&buffer, count, MPI_INT, root, MPI_COMM_WORLD);
printf("My Rank = %d Result = %d\n", myRank, buffer);

MPI_Finalize();
```

The above program is meant to broadcast a number from rank 0 to everyone; however, the program is not written correctly. Locate the bug(s) in the above program and propose a fix to it.
4) Code comprehension 2 (10 pts)

The following MPI program exchanges certain buffers between partner processes (Each process has exactly one matching process to communicate).

```c
int nprocs, myRank, toRank;
int*(sendBuffer);
int*recvBuffer;
int bufferCount = /* Some constant */
int programIter = /* Some constant */

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myRank);

for(int iter = 0; iter < programIter; iter++) {
    toRank = /* Calculate its neighbor's rank */
    sendBuffer = /* set elements */

    MPI_Ssend(sendBuffer, bufferCount, MPI_INT, toRank, 0, MPI_COMM_WORLD);
    MPI_Recv(recvBuffer, bufferCount, MPI_INT, toRank, 0, MPI_COMM_WORLD);

    // Apply the received buffer to my matrix.

    // Does some calculation
}

MPI_Finalize();
```

Identify the potential (non-syntax) problem with the above MPI code and propose a fix to that problem. Explain how your fix will resolve the problem.
5) Communication cost (20 pts)

Consider a vector of size N located at process 0 (root), in a system with P processors. A scatter algorithm distributes the vector in chunks of size $\frac{N}{P}$ to each process in the group in the order of their ranks. The chunk at distance $i \times \text{chunk}$ from the start of the array, should go to the process with rank $i$, where $i = 0, ..., P - 1$. Assume $N \% P == 0$

a) Consider an MPI Scatter implementation that scatters the vector, using point-to-point messages from the root to every other process in the group. What is the communication cost of your implementation? You can use a communication model where the cost of sending / receiving a message is $\alpha + \beta \times m$ where $\alpha$ is the latency and $\beta$ is the cost of transmitting a byte. (10 points)
b) How would the communication cost change if we used a binary tree arrangement of processes, rooted at process 0? The root of each subtree extracts the data meant for itself, sends the data meant for processes in the left sub-tree to its left child, and sends the data meant for processes in the right sub-tree to its right child. The recursion is terminated when the size of the tree is 1. Derive an expression for completion time of this algorithm. 

(10 points)
6) Code writing (15 pts)

Write an MPI program segment that computes the *trace* of a square matrix. A *trace* is the sum of the elements in the main diagonal of the matrix. Assume there are $m^2$ processors which each process has a $n$ by $n$ submatrix. The matrix follows a checkerboard block decomposition with an $m$ by $m$ arrangement of blocks. The first rank starts in the top-left corner and ranks continue in row-major order so that the last rank ends in the bottom-right corner. The final value of the trace is to be made available in the variable `TRACE` of ALL processors associated with the blocks along the main diagonal. The other processors will have zero in their `TRACE` variable. Keep in mind MPI_Allreduce() is an expensive call. Efficient solution is important, in particular, limit the communication only to the ranks that need to communicate.
7) **Isoefficiency (10 pts)**

Assume a parallel algorithm with problem size $W$ requires $N^3$ operations. The amount of computation on each processor is equal to $\frac{N^3}{P}$. The amount of communication is given by the equation $4*N$. Compute the isoefficiency of the algorithm.
**MPI Reference Sheet**

MPI_Init(int *argc, char ***argv)
MPI_Finalize()

MPI_Comm_rank(MPI_Comm comm, int *rank)
MPI_Comm_size(MPI_Comm comm, int *size)

MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
MPI_issend(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)
MPI_Ssend(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
MPI_Rsend(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

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

MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
MPI_Allreduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
MPI_Allgather(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)
MPI_Altoall(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)

**MPI Data Types**

MPI_CHAR
MPI_INT
MPI_FLOAT
MPI_DOUBLE
MPI_INTEGER
MPI_CHARACTER