CISC 372: **Exam 1**

Instructions

This exam consists of 13 problems.

You may use one 8.5×11" sheet of paper with anything written on both sides. No other notes, references, or devices may be used.

No points will be subtracted for an incorrect answer, so it is to your advantage to answer all questions.

You may either print this out and write you answers in this exam, or read the exam on your monitor and write your answers on your own paper.

If you print this out: write your name in the space on the following page. There should be plenty of room to work out the problems, but you can also write on your own paper.

If you are not printing this out: write your solutions on your own paper. Put your name on each page and number the pages. Make it clear which problem you are answering.

In any case: scan everything you have written and email it to your proctor.

Problem	Points	Max
1		5
2		5
3		10
4		5
5		10
6		5
7		5
8		5
9		5
10		5
11		30
12		5
13		5
Total		100

Name: ______________________________________

MPI Reference

Types.

```
MPI_Comm   MPI_Datatype   MPI_Status
```

Constants.

```
MPI_COMM_WORLD   MPI_INT   MPI_FLOAT   MPI_DOUBLE   MPI_CHAR   MPI_SHORT   MPI_BYTE   MPI_LONG
MPI_ANY_SOURCE   MPI_ANY_TAG    MPI_PROC_NULL   MPI_STATUS_IGNORE
MPI_STATUSES_IGNORE   MPI_SUM   MPI_MAX   MPI_MIN   MPI_PROD   MPI_LAND   MPI_LOR   MPI_LXOR
MPI_BAND   MPI_BOR   MPI_BXOR
```

Functions.

```
int MPI_Init(int *argc, char ***argv)
int MPI_Finalize(void)
int MPI_Get_processor_name(char *name, int *resultlen)
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
int MPI_Send(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_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_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count)
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 *recvcounts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
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)
int MPI_Allgather(void *sendbuf, int sendcount, MPI_Datatype sendtype,
             void *recvbuf, int recvcount, MPI_Datatype recvtype, 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 *sendcounts, int *displs, MPI_Datatype sendtype,
             void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
int MPI_Alltoall(void *sendbuf, int sendcount, MPI_Datatype sendtype,
             void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)
int MPI_Alltoallv(void *sendbuf, int *sendcounts, int *sdispls, MPI_Datatype sendtype,
             void *recvbuf, int *recvcounts, int *rdispls, MPI_Datatype recvtype, MPI_Comm comm)
double MPI_Wtime(void)
```

Part 1: Write the code to do this

Write code to do what is requested. Do not do more than what is asked. For example, you do not need to write a complete program, or call `MPI_Init`. You only need to write statements to do what is described—and nothing more. Use the communicator `MPI_COMM_WORLD` unless instructed otherwise.

Example: Assume `a` is an array of 10 `doubles`. Send (all of) `a` to process 5 using tag 99.

Answer: `MPI_Send(a, 10, MPI_DOUBLE, 5, 99, MPI_COMM_WORLD)`

1. (5 points) Declare `a` to be an array of 100 `doubles`.

2. (5 points) Assume `a` and `b` are arrays of 100 `ints`. Send (all of) `a` to process 7 while receiving into `b` from process 3, using tag 0 in both cases. The receive status is not needed.

3. (10 points) Assume `x` has type `float`. Receive into `x` from any source using any tag, then divide `x` by 2.0, then send `x` back to whichever process sent you `x` using the same tag that process used in its send. If you need to use any variables other than `x`, be sure to declare them before you use them.

4. (5 points) Assume variables `x` and `s` of type `double` have been declared on every process. Get the maximum over all processes of the `x` variables into the variable `s` on process 32. The result is needed on process 32 only.

5. (10 points) A string of length `nprocs` is distributed across all `nprocs` processes, so that for each i ($0 \leq i <$ `nprocs`), process i has character i of the string stored in a variable `c` of type `char`. Using a collective operation, write code to get the entire string into variable `s` on every process. First declare `s` and make sure `s` is big enough to hold the string. Don't forget that in C, strings are null-terminated.

Example: `nprocs` = 3, the value of `c` on process 0 is '`c`', the value of `c` on process 1 is '`a`', and the value of `c` on process 2 is '`t`'. After executing your code, the value of `s` on all processes will be "`cat`".

Section 2: Point-to-point semantics

In each of the following, you are shown a code snippet that is to be executed by all processes. If it helps, you can imagine that the code is inserted in the body of the `main` function, with the usual MPI boiler-plate added:

```
#include ...
int main() {
  int nprocs, rank;
  MPI_Init(NULL, NULL);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    ... code snippet goes here ...
  MPI_Finalize();
}
```

Assume that MPI, `nprocs`, and `rank` have all been initialized properly, and that the program is run with at least 2 processes (`nprocs` ≥ 2).

6. (5 points) Can the following code deadlock? (Yes or no).

```
int x = 0, y;
MPI_Send(&x, 1, MPI_INT, (rank+1)%nprocs, 0, MPI_COMM_WORLD);
MPI_Recv(&y, 1, MPI_INT, (rank+nprocs-1)%nprocs, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

7. (5 points) Can the following code deadlock? (Yes or no).

```
int x = 0, y;
if (rank == 0) {
  MPI_Send(&x, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
  MPI_Recv(&y, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
} else if (rank == 1) {
  MPI_Send(&x, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
  MPI_Recv(&y, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```

8. (5 points) Can the following code deadlock? (Yes or no).

```
int x = 0, y;
if (rank%2 == 0) {
  MPI_Send(&x, 1, MPI_INT, (rank+1)%nprocs, 0, MPI_COMM_WORLD);
  MPI_Recv(&y, 1, MPI_INT, (rank+nprocs-1)%nprocs, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
} else {
  MPI_Recv(&y, 1, MPI_INT, (rank+nprocs-1)%nprocs, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  MPI_Send(&x, 1, MPI_INT, (rank+1)%nprocs, 0, MPI_COMM_WORLD);
}
```

9. (5 points) Can the following code deadlock? (Yes or no).

```
int x = 0;
if (rank > 0)
  MPI_Send(&x, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
else if (rank == 0) {
  for (int i=1; i<nprocs; i++)
    MPI_Recv(&x, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
if (rank == 1)
  MPI_Send(&x, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
else if (rank == 0)
  MPI_Recv(&x, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

10. (5 points) Assume `nprocs` ≥ 3. Write all possible outputs from the following code:

```
int x;
if (rank == 0) {
  MPI_Recv(&x, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  printf("%d", x);
  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Recv(&x, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  printf("%d", x);
} else if (rank == 1) {
  MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
  MPI_Barrier(MPI_COMM_WORLD);
} else if (rank == 2) {
  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
}
```

11. (30 points) Consider this sequential program:

```
#include ...
int main(int argc, char * argv[]) {
  int n = atoi(argv[1]);
  double a[n];
  for (int i=0; i<n; i++)
    a[i] = arccosh(i*i); // arccosh is some function from double to double
  double s = 0.0;
  for (int i=0; i<n; i++)
    s+=a[i];
  printf("%lf\n", s);
}
```

Write a parallel, MPI version in which the array `a` is block distributed using the Standard Block Distribution scheme. The distributed array should be initialized in the same way as the sequential program. The output should be the same as the sequential, except possibly for some round-off error. A skeleton MPI code is given below, you just need to write the missing section.

```
#include ...
int main(int argc, char * argv[]) {
  int nprocs, rank;
  MPI_Init(&argc, &argv);
  int n=atoi(argv[1]);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  ... write the code to be inserted here ...
  MPI_Finalize();
}
```

Section 4: Performance

12. (5 points) You measure the performance of a sequential program. The program executes in two phases: the first phase takes 4 seconds, and the second phase takes 24 seconds. You have determined that the first phase is inherently sequential — no parallelization is possible — but the second phase can be parallelized very effectively.

According to Amdahl's Law, what is the maximum possible speedup that can be obtained on this program?

13. (5 points) A sequential program takes one hour to execute. You have 1200 cores at your disposal. What is the minimum efficiency required of a parallel version to get the runtime equal to or below 10 seconds?