

CS 475 Fall 2010 Midterm II

Nov 18 2010

Name _____

Please read these instructions completely before proceeding. Do not turn this page until you are asked to. This is a open book/notes exam. Please answer all questions. There are three questions, and/or two of them have extra credit points. Please do not attempt them until you finish the rest of the exam. Use the points (total points=75, the scores will be scaled and eventually normalized) to guide your time management.

Prob. No.	Max score	Your Score
1	20 +5	
2	20	
3	35 + 10	
Total	75 + 15	

Problem 1. Collective Communications in MPI:

[20+5 pts]

In an MPI program, every processor has int array variables A[], B[], C[], D[], and E[] (they have been allocated with the appropriate size so that none of the buffers referenced below will cause seg faults). The program is run on 4 processors (numbered 0...3 as usual), and at a certain time, four of the variables A, B, C and D have the same values on a given processor, but different across the processors. The values are (leftmost is the smallest address):

A=B=C=D=[0 2 4 6 8 10 12 14] in P0

A=B=C=D=[1 3 5 7 9 11 13 15] in P1

A=B=C=D=[3 6 9 12 15 18 21 24] in P2

A=B=C=D=[35 30 25 20 15 10 5 0] in P3

What will be the state of the variable after each of the following function calls is executed? For each question, assume that the initial state is the one given above, not the one from the execution of the previous question. Show only the variables that have changed (signatures of common collectives are on the last page).

MPI_Reduce(A, B, 5, MPI_INT, MPI_SUM, 3, MPI_COMM_WORLD); [4 pts]

MPI_AllReduce(A, C, 2, MPI_INT, MPI_SUM, MPI_COMM_WORLD); [4 pts]

`MPI_Scatter(A, 2, MPI_INT, C, 2, MPI_INT, 0, MPI_COMM_WORLD);` [4 pts]

`MPI_AllGather(A, 2, MPI_INT, D, 2, MPI_INT, MPI_COMM_WORLD);` [4 pts]

The MPI Library has no function called `MPI_AllScatter`. You are asked to propose such a function. Give the most logical definition of its functionality and illustrate what will happen on the above data with the following call

`MPI_AllScatter(A, 2, MPI_INT, E, 2, MPI_INT, 0, MPI_COMM_WORLD);` [4 pts].

Why has the MPI Library has not included such a function so far. [5 pts ec]

Problem 2. Knapsack:

[20 pts]

Remember that in the divide-and-conquer knapsack, it is possible to do a larger “base-case,” by switching to the memory-inefficient but twice-as-fast full-table method the base whenever the size of the problem is `BKSIZE` or smaller. We want to analyze the performance gains of this strategy. Assume that we are solving a problem with 4000 items and a knapsack of capacity 10M (10^7).

What is the work done by the D&C algorithm if the recursion goes all the way down to 1 (*without* the early exit)? Approximate this to the next higher power of two. [5 pts]

Assume that at each level the two subproblems are of exactly equal size. What is the work done by the D&C algorithm if the switch to a full-table happens at a problem of size 2^{30} (4GB) [10 pts]? Therefore, what is the percentage savings provided by the larger base case [5 pts]? [15 pts]

Problem 3. MPI Collectives Programming Exercise: [35 pts]

The `MPI_Gather` function has a bug on our machine, and we have to re-implement it using plain `MPI_send` and `MPI_recv`. We will only implement the special case, where the root is always processor 0, and the number of processors is a power of two.

First let's work out an example. Every processor in an 8-processor machine has an array of 4 integers. We are going to gather this data to processor 0. First draw the task-channel graph, i.e., a complete binary tree describing *all* the events [4 pts]. Also show the agglomeration (clustering) that produces the binomial tree of a standard divide-and-conquer algorithm from this [4 pts]. [8 pts]

Next, show in the table below, the sequence of point-wise communications that take place on processors 0–7. Line up each send with the corresponding receive on the destination processor. [7 pts]

Proc 0	Proc 1	Proc 2	Proc 3	Proc 4	Proc 5	Proc 6	Proc 7

Now, if there are 64 processors (the initial data size is still 4), write the sequence of events that occur on processors 0, 2, 24, 44, and 61. [10 pts].

Now for the hard part. Write a single *generic* snippet of code that achieves the above computation. The communications will be in a loop whose iterations are parameterized by some function of `myid` and `numprocs` (assume that these have been properly initialized). The destinations and sources of the `send` and/or `recv` calls in each iteration will also be a function of these and the iteration counter. Do not worry about the exact data values (addresses and/or buffer sizes) [10 pts + 10 ec]

It may be helpful for you to think of the answers to the following questions.

How many rounds does processor `myid` participate in (expressed as a function of `myid` and `numprocs`)?

What is the set of source processors of the data that it receives in each round?

What is the set of successive targets that it sends to in each round?

Signatures of common MPI collective communication routines

```
int MPI_Recv( void *buf, int count, MPI_Datatype datatype, int source, int
tag, MPI_Comm comm, MPI_Status *status )
int MPI_Gather ( void *sendbuf, int sendcnt, 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 )
int MPI_Scatter ( void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf,
int recvcnt, MPI_Datatype recvtype, 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_Reduce(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype,
MPI_Op op, int root, MPI_Comm comm )
```