

Message passing and MPI

Assignments and problems

Graham Fagg

CS-594 Spring 2006

Rules

- Two weeks to complete assignments and problems. Hand in by midday 1st Feb 2006. I will go over the answers on Feb 8th.
- If you have problems and do not ask for help (until 10 minutes before the deadline) beware!
- Hand-in written work either on paper or Email to me fagg@cs.utk.edu
- Code is to be tarred with makefiles, output etc and a MD5 signature sent to me.
 - md5sum mywork.tar |& mail fagg@cs.utk.edu
 - Code is to run on boba machines. If I cannot verify it by remaking it and running it then I will assume it does not!
 - Broken code with comments gets more points than non working non commented code.
 - A short description of design is always needed.
- Test code with BOTH Open MPI and FT-MPI (2.0rc1)

Part A

Correctness and buffering?

- | | |
|---------------------------------|---------------------------------|
| • Proc 0 | Proc 1 |
| • MPI_Send (data,size.. 1) | MPI_Send (data,size.. 0..) |
| • MPI_Recv (indata,insize..1..) | MPI_Recv (indata, insize.. 0..) |

Above is a head to head send. This might or might not work depending on the system, MPI implementation or other factors such as timing.

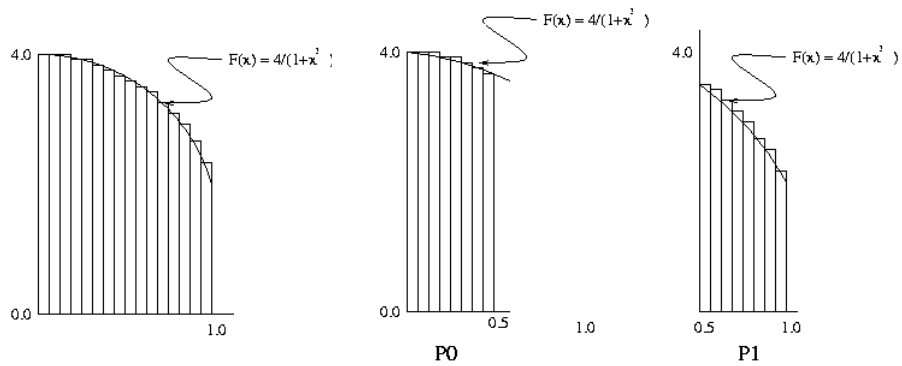
1. Write a paragraph on why the above is an incorrect [non-deterministic] MPI code.
2. Re-write the above code in 3 different ways to make it work (hint, there are 4 simple ways (lookup MPI_Sendrecv))
3. Write a simple test to see when the above deadlocks. If it deadlocks.

Part B

Collecting with collectives

1. Write a SPMD code with 1 master and 3 slaves that calculates pi using code handed out in class.
 - Use p2p calls.
 - Master sends out slices
 - Slaves calculate integral slices
 - Master sums to get pi.
2. Change the above to use collective calls
 - Master sends out slices all at once
 - Master receives all slices at once
 - Master sums to get pi.
3. Change 2 so that the master does not have to sum the data explicitly but the collective for receiving the partial results does this.

Part B



Part C

Benchmarking part I

1. Write a simple benchmark to test the performance of a data Exchange between two processes (what you were doing in Part 1c)
2. Benchmark your three implementations
3. Write not more than a page on why the three versions might execute differently and what you expected.
4. Repeat the exercise for a SWAP. I.e. the two buffers used for sending and receiving are the same. Comment on performance, memory requirements and *correctness* of your implementations.

Part D

Benchmarking part II

1. Write a simple benchmark that tests MPI_Alltoall for four messages sizes (128bytes, 4K, 64K and 256K*) (use MPI_Wtime). Display min, avg (mean), max and std_dev.
2. Using the MPI profiling interface write 2 new versions of alltoall:
 1. Post all sends and all recvs and then do a wait all
 2. Synchronous version where only one node sends each 'step'. (Do you need a token to control who sends?)For each method, measure and try and explain the results. Compare to the Alltoall in each MPI implementation.

*(data size at each node)