

CS442/ECE432: Homework 2

February 12, 2008

General

- This homework is about material covered in Chapter 2 of the text book.
- The answers to this homework are **due February 21, 2008**.
- Submit via e-mail to `riesen@cs.unm.edu` (Mail it before class on the 21st.) Your subject line must say: “Homework 2 Submission”.
- Keep your answers (and programs) succinct.
- Obey the University rules on plagiarism. In particular, do use libraries and the web to find information you need to answer the questions, but do not copy whole answers or programs. Reference your sources. The work you turn in must be *your* work.
- You are allowed to use the example programs on the class web site as a starting point.

Grading

With so many students and programs, we want to make the testing and grading as automated as possible. It is therefore very important that you follow the rules below exactly. If we have to manually inspect your program and modify it to work with our scripts, we will deduct 10 points.

One File Per Program

For each of the exercises below, create a single file name `e_xx.c`, where the `xx` is 01 for the first exercise, 02 for the second, etc.: “`ex_01.c`”, “`ex_02.c`”, ...

For C++ files use “`ex_01.cc`”, “`ex_02.cc`”, ...

Clean, Readable Code

Your code must be clean and readable. Be consistent with you indentation, use descriptive names for your variables and constants, etc.

Your program must compile without warnings (we use the `-Wall` flag for `gcc`). Your program must be valid MPI; e.g., there must be no pending messages when `MPI_Finalize()` is called.

Some of this is subjective, but we will subtract up to 10 points for code we cannot read or understand, even if it works.

Languages

Write your programs in C or C++. They must compile with the GNU C compiler and link against an MPI-2 library. Even when using C++, you must use the C bindings for MPI; i.e., use `MPI_Send()` not `MPI::Comm::Send()`.

Tar File

Put your source code, no executables or object files, into a single tar file and compress it. Name your archive “Hwk2.tgz” It must untar into the current directory; i.e., no sub directories. You can use zip instead of tar. In that case name your file “Hwk2.zip”

The archive must contain a file named “AUTHOR” which contains your name. Thus, a complete archive for this homework would contain the following files **only**: “ex_01.c”, . . . , “ex_05.c”, “AUTHOR”.

Documentation

Any documentation and solutions that are not programs must be in pdf or plain ASCII text.

Assignments

Exercise 1: Global Sum

Write a global sum function. It should have the prototype shown in Listing 1. At the root node, it should return the sum of all summands submitted by the participating nodes. On all other nodes, it should return 0.

Use only point-to-point MPI functions to implement `gsum()`, and use a fixed amount of storage on each node, independent of how many nodes participate. That is, no node should receive much more data than any other node. For example, collecting all summands on the root node and performing the additions is not acceptable.

```
gsum(int summand, int root, MPI_Comm comm);
```

Listing 1: Function Prototype for Exercise 1

20 points For a working function

Exercise 2: Barrier Synchronization

We have seen that performing a reduction and then a broadcast of the result forces a synchronization point, just as if we had called an `MPI_Barrier()`. Could we achieve the same synchronization doing the broadcast first, and then the reduction? Explain why.

10 points For a correct answer

Exercise 3: Two-node Synchronization

Write a subroutine that synchronizes exactly two nodes using only point-to-point operations. Name your subroutine `twosync()`. It takes a single integer as an argument which denotes the other node you wish to synchronize with.

20 points For a working function

Exercise 4: User-Defined Reduction Function

Write a user-defined reduction function that checks whether the contributions from each process are sorted in rank order with the smallest value residing at rank 0. You may assume the values submitted range from 0 to `MAX_INT`. The result stored on rank 0 after the reduction should be 0, if the list is sorted, or some positive value if the list is not sorted.

Each node will contribute a single integer to the reduction.

20 points For a working function

Exercise 5: Check a Sorted List

Each rank has a sorted list of values such that the smallest value on rank n is greater or equal to the largest value on rank $n - 1$. Write a program that checks the local lists of values and then makes sure that the individual lists are in rank order and do not overlap.

20 points For a working program