

# CS442/ECE432: Homework 3

February 26, 2008

## General

- The answers to this homework are **due March 4, 2008**.
- Submit via e-mail to `riesen@cs.unm.edu` (Mail it before class on the 4th.) Your subject line must say: “Homework 3 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. Ask me for help if you are stuck.

The whole homework will receive **zero points**, if I detect cheating. Any offenses after that will result in dismissal from class and a **report** to the University.

- 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 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 “Hwk3.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 “Hwk3.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\_02.c”, “ex\_03.c”, “ex\_03.pdf”, and “AUTHOR”.

## Documentation

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

## Assignments

### Exercise 1: Selective Sum

Write a user-defined reduce operation that can be used with `MPIReduce()` and the other collective operations that accept a user defined function. Your function must accept the following value/index pairs: `MPI_2INT`, `MPI_FLOAT_INT`, and `MPI_DOUBLE_INT`. The first entry in each of these type signatures is a value, and the second entry is an index which we will use as a flag.

Your function must have the exact prototype shown in Listing 1. Submit a file named `ex_01.c` (or `ex_01.cc`) that contains only your function. The file must compile on its own, so make sure all necessary `.h` files are included.

```
void selective_sum(void *invec, void *inoutvec, int *len, MPI_Datatype *dt);
```

Listing 1: Function Prototype for Exercise 1

Your function must work on vectors of length `len`. Add the values at each offset  $i$  of the vector using the appropriate C function: integer, float, or double add. If the flag at index  $i$  is 0, ignore value  $i$  (in that vector) and do not add it to the total.

**20 points** For a working function (`ex_01.c` or `ex_01.cc`)

### Exercise 2: ID Numbers

An administrative process requires identification numbers to be assigned to each transaction. (E.g., Order numbers, customer IDs, etc.) There are some constraints for an ID number to be “acceptable”:

- ID numbers are seven-digit combinations of the numerals 0–9.

- Two consecutive digits may not be the same. Leading consecutive zeros are OK. ID 0000000 is OK by this rule.
- The sum of the digits may not be 7, 11, 13, 21, or 33.
- The seven digits cannot all be even or odd. Again, ignore leading zeros.

Write a parallel program that calculates how many ID numbers are available, given the above constraints.

Your program should run on arbitrary number of nodes (within reason), and the work performed on one node should be about the same as the amount of work on any other node. At the end of the run, each rank should print out how many IDs it has processed. Like this:

```
[000] Processed x IDs
[001] Processed x IDs
[002] Processed x IDs
...
```

Where  $x$  is the number of IDs checked by that rank.

The outputs do not need to be in rank order. If I define `DEBUG`, your program must print why it is rejecting an ID. Your code, for the case where the digit sum is 13, should look like this:

```
1 #if defined(DEBUG)
2     printf("[%3d] Rejecting ID %7d because digit sum is 13\n", my_rank, current_id);
3 #endif
```

Your program must output how many total valid combinations it found. With the above rules, that number should be 4870849. Make sure your program produces the correct result independent of the number of nodes it runs on.

**20 points** For a working program (`ex_02.c` or `ex_02.cc`)

### Exercise 3: Row versus Column Access

Write a (serial) program that takes an integer as a command line argument like this:

```
./ex_03 -n 1000
```

If the user forgets to provide `-n num` your program should print an error message and exit.

Use the command line argument provided to allocate a matrix of integers of size  $n \times n$ , where  $n$  is the command line value. Fill the matrix, starting at `A[0][0]`, with the values  $0, 1, \dots, n^2 - 1$

Now, start a timer. You can use `MPI_Wtime()` or the function I'll provide that uses `gettimeofday()`.

Do the following 20 times: add all the individual values in the matrix in row-major order.

Stop the timer. Divide the timer value by 20 and print it.

Repeat the above procedure, but do the addition in column-major order.

When you access a matrix in row-major order, you are doing this:

$$\sum_{j=0}^n \sum_{i=0}^n a_{ij}$$

I.e., you are adding all the values in row  $i$ , before adding the values in row  $i + 1$ .

When you access the matrix in column major order, you are doing this:

$$\sum_{i=0}^n \sum_{j=0}^n a_{ij}$$

I.e., you are adding all the values in column  $j$ , before adding the values in column  $j + 1$ .

Run your program on `hammer.hpc.unm.edu` and on one other machine. Use several different values for  $n$ . Write a brief description of what you have observed. Provide a plot with  $n$  on the x-axis, and the row and column access times on the y-axis. Do that for both machines. You may combine all results in a single plot, as long as it remains readable. Make sure you label both axes and the two (or four, if you provide a single plot) plot lines. Specify the CPU and how much main memory the second machine has.

**10 points** For a working program (`ex_03.c` or `ex_03.cc`)

**20 points** For a concise description of what you have found (`ex_03.pdf`)

**20 points** For a clear, complete with labels, plot of your measurements (`ex_03.pdf`)