ECE 473 --Assignment 03
Due Date provided on BlackBoard
Global Sum

Task: Write a global sum operation with MPI

You are to write a function with the following specification:

```
void global_sum(
double* result, int rank, int size, double my_value);
```

This function must be executed by all tasks in MPI_COMM_WORLD and computes the sum of all the my_value's and returns the sum of all the tasks individual 'my_value's via the result pointer. (All processors must know the global sum after the call returns)
Note, this is similar to what MPI_Allreduce with addition does, except I only expect you to deal with scalar local my_values (i.e. not the reductions of arrays of values).
Note, for convenience, you may assume that the number of processors you run this on is a 'power of 2', i.e. 2, 4, 8, 16, etc. You must put in some error checking code to see ensure your code only executes if the number of processors matches this assumption and if not, immediately exits.

As mentioned above, this is essentially equivalent to an MPI_Allreduce (with the SUM operator), but you are to write it only using MPI_Send and MPI_Recv (or MPI_Sendrecv). You are not to use any collective communication routines. The general discussion of binomial trees that might be useful for this project is located in Section 3.5 of the Quinn textbook. The project will be broken down into the following files (must contain these):

HW04_first_last.c
// this is where main() is and also where each process picks a number as his “my_value”, and then they all invoke global_sum() and then report the global sum to stdout.
functions.c
// this is where you have global_sum() defined, among others
functions.h
// this is where you have your global_sum() prototype, among others
Makefile
// this is a typical Unix makefile that will compile your main program as well as functions.c file to objects, and will then link them together into a final executable.

Therefore, I should be able to use my own Makefile, and HW04.c file, and compile in your
code and still work. You may want to pick a value for each processes' my_value that will make it easy for you to determine if it is working or not. (like their ranks)
Develop your programs first on your local machine, however you MUST also execute it on Palmetto on 2, 4, 8, 16, and 32 processors via the PBS queueing system.
Submission into BlackBoard:

I expect that your project will reside in a single directory, and that will have all necessary files in that directory. That directory should be named first_last_ass4, with the files mentioned above inside. From there, you will tar and gzip that directory, and submit that to BlackBoard. It will contain all your files, as well as the PBS scripts, and the output files from Palmetto. That can be done from the command line as follows with a tar command piped to gzip.

```
$ pwd
/home/wjones/classes/473
$ tar cf - ./HW04_will_jones/ | gzip > HOW04_will_jones.tar.gz
$ ls
HW01 HW02 HW03 HW04_will_jones
HW04_will_jones.tar.gz
```

For future reference, you can untar and unzip a tar.gz file like this:

```
$ tar xvfz will_jones_ass4.tar.gz
./will_jones_ass4/
./will_jones_ass4/functions.o
./will_jones_ass4/functions.c
./will_jones_ass4/will_jones_ass4_gsum.c
./will_jones_ass4/Makefile
./will_jones_ass4/runmain
./will_jones_ass4/functions.h
./will_jones_ass4/will_jones_ass4_gsum.o
```

It would be a good idea to test to make sure your tarring and zipping worked by copying it to a temp directory, and unzipping it to make sure it is all there.
From there, upload to BlackBoard you .tar.gz file. Also upload any output files you obtained from Palmetto.

Grading Rubric:

1) you will get minimum points for a completely working solution where the number of communication phases is O(2log n) or O(log n) “C”
2) you will get additional points if your solution uses a bitmask in controlling the stages of data exchange. “B”
2) you will get additional points for a solution that does NOT do a reduce followed by a broadcast. (i.e. O(log n) instead of O(2log n)) “A”

Appendix:

Unix Makefiles are very powerful tools, not just for coding. There are some very advanced features out there; however here is a simple example of my Makefile for this assignment. It might be a good idea to look over a Makefile tutorial as well. There are many out there. Here is a link to one a friend of mine wrote:

http://www.parl.clemson.edu/~plouis/summerStudents/make_cvs_vi.pdf
Note that below the indentation is created by using a single TAB, not spaces. This is important.

```
CFLAGS = -g -Wall -Wstrict-prototypes
PROGS = runmain
OBJECTS = HW04_will_jones.o functions.o
LDFLAGS = -lm
CC = gcc
MCC = mpicc

all: $(PROGS)

runmain: $(OBJECTS)
	$(MCC) $(LDFLAGS) -o runmain $(OBJECTS)

HW04_will_jones.o: HW04_will_jones.c
	$(MCC) $(CFLAGS) -c HW04_will_jones.c

functions.o: functions.c functions.h
	$(MCC) $(CFLAGS) -c functions.c

clean:
	rm -f $(PROGS) *.o core*
```