CSCI 473

Intro to Parallel Systems - SP25
Assignment 6 - MPI Global Sum

Read through the documentation in its entirety. Create the source code necessary to solve the problem, run experiments, and write a report or presentation about your process and findings.

In this assignment, create a function that will generate a global sum of values. You are to write this only using MPI point to point communication primitives: `MPI_Ssend` and `MPI_Recv`. Do not use any collective communication routines.

1. Write your own function calls with the following prototypes:

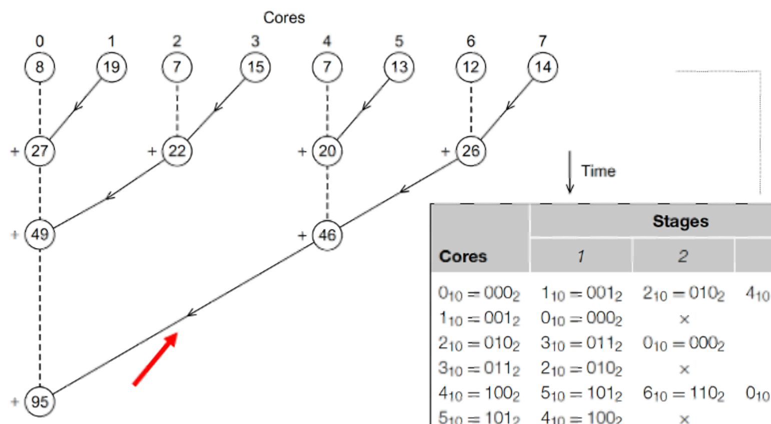
```
void global_sumA(double* result, int rank, int size, double
my_value);
void global_sumB(double* result, int rank, int size, double
my_value);
```

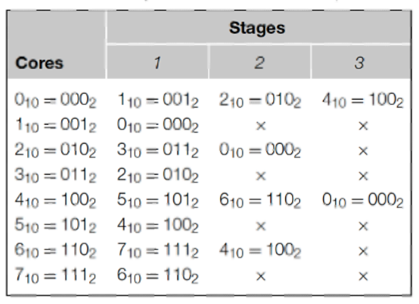
2. This function must be executed by all tasks in `MPI_COMM_WORLD` and computes the sum of all the `my_value`'s and returns the sum of all the tasks individual `my_value`'s via the result pointer. All processors must know the global sum after the call returns.

3. For convenience, assume that the number of processors you run this on is a power of 2 (i.e. 2, 4, 8, 16, etc.). You must include error checking to ensure your code only executes if the number of processors requested AND received matches this assumption. If not, immediately exit.

4. `global_sumA:` Write a version of this code that is straightforward, using the SPMD type process as discussed in class. Once process 0 knows the global sum, send the answer to all other processes. Include timing information.

5. `global_sumB:` Write a second version of this code that uses a variation of the tree-based strategy discussed in previous sections. Note that this image shows a situation in which only process 0 knows the global sum. Recall we want all processes to have the global sum after it is calculated. Include timing information.





Cores	Stages 1	Stages 2	Stages 3
$0_{10} = 000_2$	$1_{10} = 001_2$	$2_{10} = 010_2$	$4_{10} = 100_2$
$1_{10} = 001_2$	$0_{10} = 000_2$	×	×
$2_{10} = 010_2$	$3_{10} = 011_2$	$0_{10} = 000_2$	×
$3_{10} = 011_2$	$2_{10} = 010_2$	×	×
$4_{10} = 100_2$	$5_{10} = 101_2$	$6_{10} = 110_2$	$0_{10} = 000_2$
$5_{10} = 101_2$	$4_{10} = 100_2$	×	×
$6_{10} = 110_2$	$7_{10} = 111_2$	$4_{10} = 100_2$	×
$7_{10} = 111_2$	$6_{10} = 110_2$	×	×

FIGURE 1.1
Multiple cores forming a global sum

6. Example execution is found in the class recording from 03/21/2025. Your output must match this exactly (excluding the binary representation of the string).

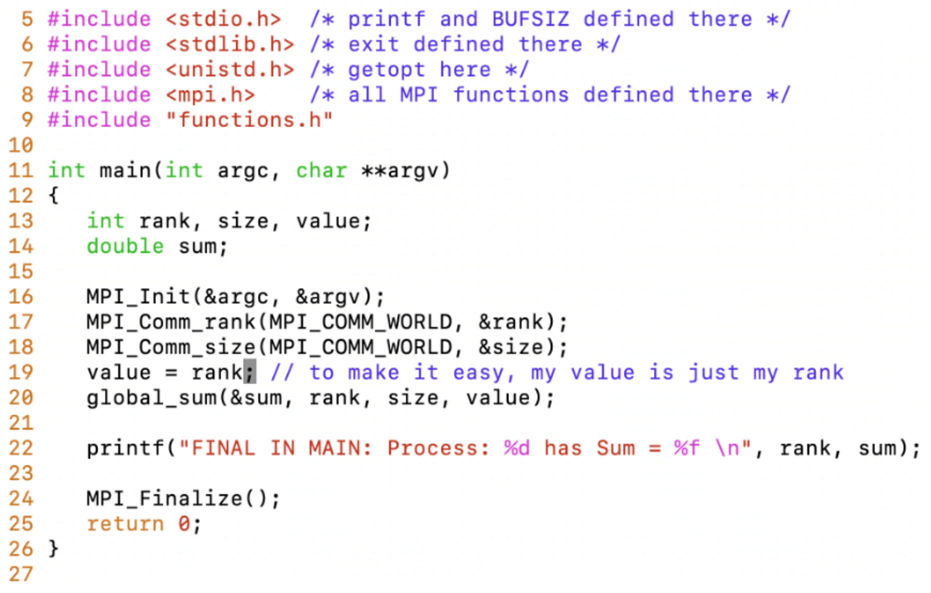
7. Create either a report or a presentation that shows the program being compiled, executed, and discussed. At minimum, test with number of processes = 2, 4, 8, and 16. Take plenty of screenshots and detail everything. Compare and contrast the two versions of the code. Come up with the best way of doing this.

8. Submission: Use the following organizational folders and ZIP the final version. Do not forget to `make clean` before zipping:

./code	// source codes and scripts
./code/gsum.c	// where the main() is. Each processes “my_value” should be initialized to its rank. All processes should invoke global_sum() & print the final answer.
./code/driver	// the executable created by compiling gsum.c
./code/functions.c	// global_sumA() and global_sumB() are defined
./code/functions.h	// global_sum() prototypes, among any other helper functions
./code/Makefile	// typical Makefile (given in Appendix of this assignment)
./code/sbatch.bash	// bash script used to run experiments
./presentation	// PPT presentation with large graphs/plots and PDF export
./report	// PDF of LaTex report and also the whole project downloaded too
./data	// some collected example data, etc
./README.txt	// text file with description about how to run your codes

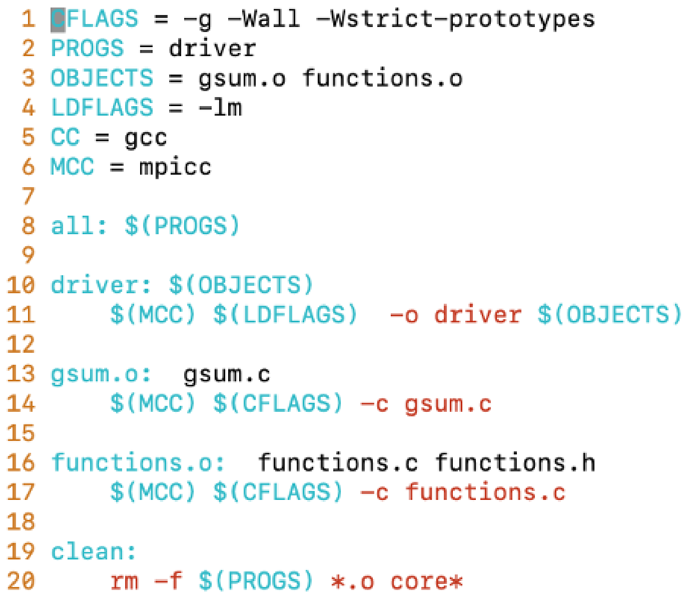
Appendix

Example gsum.c



```c
#include <stdio.h>  /* printf and BUFSIZ defined there */
#include <stdlib.h> /* exit defined there */
#include <unistd.h> /* getopt here */
#include <mpi.h>    /* all MPI functions defined there */
#include "functions.h"

int main(int argc, char **argv)
{
    int rank, size, value;
    double sum;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    value = rank; // to make it easy, my value is just my rank
    global_sum(&sum, rank, size, value);

    printf("FINAL IN MAIN: Process: %d has Sum = %f \n", rank, sum);

    MPI_Finalize();
    return 0;
}

```

Example Makefile



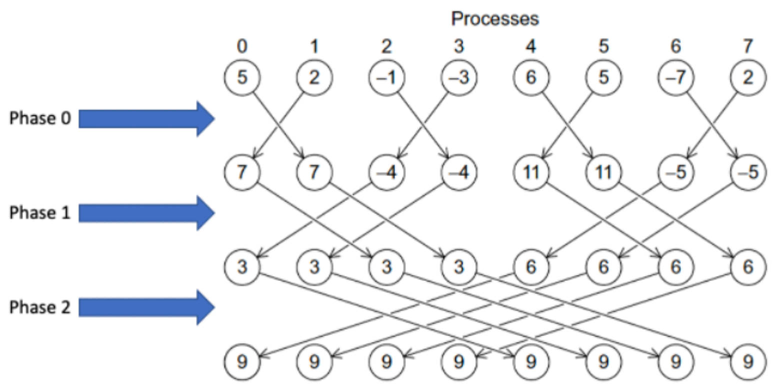
```makefile
CFLAGS = -g -Wall -Wstrict-prototypes
PROGS = driver
OBJECTS = gsum.o functions.o
LDFLAGS = -lm
CC = gcc
MCC = mpicc

all: $(PROGS)

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

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

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

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

Hint for global sum being generated for every process

Look at the example output. Each "phase" is one set of communication that is taking place essentially concurrently. So since np = 8 in my example, there were log(8) = 3 phases. Your code should execute in these phases too. This idea is represented with this image:

FIGURE 3.8

A global sum followed by distribution of the result



Rubic

- Correct code structure and Makefile
- Only uses MPI_Ssend() and MPI_Recv()
- Uses correct prototypes
- Version A with SPMD reduction
- Version B with tree-based reduction
- Checks power of two
- Code prints correctly
- Report/Presentation that includes: compile, execute, and run on 2,4,8,16 processes