CSCI-UA.0480-003
Parallel Computing
Lab Assignment 1

In this lab you will write MPI code to find numbers divisible by *any* of the given numbers a, b, and c in the range between 2 and N (inclusive) and test scalability and performance. For example: find numbers divisible by 2, 3, and 5 in the range between 2 and 10000,

General notes:

- The name of the source code file is: netID_code.c where netID is your NetID.
- You compile with *mpicc -std=c99 -Wall -o checkdiv netID_code.c*
- To execute it I type: *./checkdiv N a b c*
  Where N is a positive number bigger than 2 and less than or equal to 100,000,000;
  and a, b, and c and positive numbers between 2 and 100 (inclusive).
- The output of your program is a text file *N*.txt (N is the number entered as argument to your program).
  For example, if I type: ./checkdiv 10 2 3 5
  The output must be a text file with the name 10.txt and that file contains: 2 4 5 6 8 9 10
  You need to put a single space between each two numbers. **The numbers must be written in the file in ascending order and one number per line** like this example above.
- You program must also print on screen:
  times of part1 = num1 s part2 = num1 s part3 = num1 s
  num1, num2, and num3 are floating points representing the time taken in seconds.
  To know what are part1, etc, continue reading. They are defined next page.
- You can assume that we will not do any tricks with the input (i.e. We will not deliberately test your program with wrong values of N, negative, float, non-numeric, etc).

The sequential code:

The sequential code is easy. You will just have a loop that goes from 2 to N (inclusive). For each number test whether it is divisible by any of the given numbers a, b, and c. It it is divisible by at least one of them, then it must be included in the output file.

The parallel code:

Assume you have p processes.

- Process 0 reads the command line and extract N, a, b, and c.
- The range 2 to N (i.e. N-2+1 numbers) will be divided among the processes with the last process (in terms of rank) taking slightly extra/less work if the range is not divisible by the range. So, if N = 10 and we have two processes, process 0 will work on range 2 to 6 and process 1 will work on the range 7 to 10.
- Then, each process works on its range and generating its own list.
- Each process sends its list to process 0.
- Finally, process zero creates the file N.txt (where N is the upper bound of the range) and writes all the numbers there.

How to measure the performance and scalability of your code?

To see how efficient your implementation is, you need to compare against a sequential version. Therefore, we will use several methods.

1. The program is divided into several parts: part 1 where process 0 reads the inputs and sends them to other processes, part 2 where each process generates its list and sends it to process 0, and part 3 where process 0 write the data to the disk.

2. We say in class how to use clock() and MPI_Wtime(). We will use both them as follows (Note: The following is a pseudo-code. You need to write proper one with correct declarations and header files):

```
start1 = clock();
  part1
end1 = clock();
start2 = MPI_Wtime();
  part2
send2 = MPI_Wtime();
reduction operation, MAX to get largest end2-start2
start3 = clock();
  part3
end3 = clock();
```

3. Parts 1 and 3 are executed sequentially by process 0 while part 2 is executed by several processes. So, you need to calculate the following:

- Time for part 1: end1 – start1
- Time for part 2: reduction operation, the MAX one, among (end2 – start2) calculated by each process.
- Time for part 3: end3 – start3

Note on using clock()

- You need to #include <time.h>
- define variables start1, start2, … of type clock_t
- After you subtract the two numbers, divide them by CLOCKS_PER_SEC to get the number in seconds. That is: (end1 - start1)/ CLOCKS_PER_SEC
- Important: You don't need to do that with MPI_Wtime();

4. When executing your full program, use the Linux time command. For example

time ./checdiv 1000000 2 5 7

Your code must show some speedup, relative to running with only one process, in part2. Finally, you will report these numbers in the report as discussed below.

The report

For that report, to generate the graphs, assume a = 3 b = 5 and c = 11.
We may test your program with different numbers though.

Write a report that contains the following graphs.

Graph1:

- A bar with N values (100, 1000, 10000, 100000, 1 million, 10 millions, and 100 millions) as x-axis.
- The speedup (y-axis) (part 2 with 1 process / part 2 with p process).
- For each number in the x-axis, draw four bars: speedup with one process, with two processes, with three processes, with four processes, and with 8 processes.

Graph 2:

- Same N values in x-axis and same bars (1, 2, 4, and 8 processes) but the y-axis is the overall speedup generated by the *time* command in step 4 above. That command generates three numbers: user, system, and real. We want the *real* one. The y-axis will then be the real time for one process divided by the real time of p process where p varies for each bar.

What to submit:

A single zip file. The file name is your **netID_lab1.zip** where netID is your, well, NetID.
Inside that zip file you need to have:

- **netID_code.c**
- pdf file containing the two graphs the file name must be **netID_report**

Submit the zip file through NYU classes.

You will get -1 for each mistake in the naming conventions above.

Enjoy!