Assignment 2

1 Synopsis

Please read the document, ***General Requirements Programming Assignments***, posted on the course website and make sure you follow them when you do this assignment. This assignment is to write your first MPI program, a simple program designed as a warm-up exercise in order for you to overcome the technical hurdles that you might have in writing and running MPI programs. Your task is to design a parallel algorithm to compute the *natural logarithm* function by numerical approximation. This is the method used in the Chapter 4 Lecture Notes for estimating the value of π.

2 Overview

You will design a parallel algorithm to compute the natural logarithm function by ***numerical approximation***. The natural logarithm of x, often written $ln(x)$, is the number y such that $e^y = x$, where e is ***Euler's constant***, the mathematical constant whose first twenty decimal digits are 2.7182818284590452353. For example, $ln(10) \approx 2.3025$. Computing this function by numerical approximation means that its value is computed by finding successively closer estimates of its value. The technique used to compute π in the Chapter 4 lecture notes by numerical approximation took advantage of the ***Second Fundamental Theorem of Calculus:***

If a function f is differentiable over an interval $[a, b]$ and we know what the first derivative of $f(x)$ is, then a consequence of the second fundamental theorem of calculus is that

$$\int_a^b f'(t)dt = f(b) - f(a) \tag{1}$$

Another way to say this is that the area under the curve of $f'(x)$ on the interval $[a, b]$ is $f(b) - f(a)$. We can use ***numerical integration*** to compute this area. The simplest numerical integration technique is the one we used to compute π, the ***rectangle rule***. The rectangle rule divides the area under the curve into equal width rectangles whose heights are the values of the function at the midpoints of the rectangles' bases. The area is approximately the sum of these areas. As the number of rectangles increases, the sum of the areas converges to the true area. Your program must use the rectangle rule to compute the area under the curve.

3 Program Invocation, Input, and Output

The MPI program expects two command line arguments: the number whose logarithm is to be determined, followed by the number of intervals used in the approximation. For example, the program, when run under MPI, with arguments 10 and 4096, would be called like this:

```
mpirun [MPI options] natlog  10.0  4096
```

The program should print four numbers separated by tab characters as its output:

input_number<tab>*computed_value*<tab>*absolute_error*<tab>*elapsed_time seconds*

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.

The input value should be printed with as many decimal digits as it needs, but no more than 16. The computed value and the error should be printed with 16 decimal digits of precision. The elapsed time in seconds should be displayed with 6 digits of decimal precision. For the above run, the output would be the following, where the time depends on the actual processors and process count:

```
10   2.3015967217706401   0.0009883712234058    0.000416 seconds
```

4 Program Implementation Requirements

- The program must use the rectangle rule.
- The program must work correctly regardless of how many tasks are run.
- The process whose rank is zero should handle all input and output; no other process may perform I/O.
- The program must detect when there are insufficient command line arguments and report this as a usage error. It must also do input validation: if either argument is non-numeric, or if either argument is not a positive number greater than or equal to 1, it must report this as a usage error.
- In order to compute the absolute error, the program must use the math library's `log()` function. The absolute error should be the absolute value of the difference between the computed value and the value returned by `log()`.
- The elapsed time should be computed following the model program in Chapter 4, using the `MPI_Wtime()` function.
- The program must be documented and written to comply with the requirements stated in the ***General Requirements Programming Assignments*** referred to above.

5 Program Grading Rubric

The program will be graded based on the following rubric out of 80 points:

- The program must compile and run on any `cslab` host. If it does not compile and link on any `cslab` host, it loses 60 points.
- **Correctness** (40 points)
  - The program should do exactly what is described above. Incorrect output, incorrectly formatted output, missing output, or output containing other characters are all errors.
  - The program should produce the same output no matter how many processes it is given, except for the elapsed time.
  - It should handle errors correctly.
- **Performance** (16 points)

  When run with successively greater numbers of processes, the elapsed time should decrease, except that if the number gets too large (larger than the available processors generally), it will start to increase again. Check that this behavior occurs on average. On any one run, it may not be exactly like this, but over many runs it should behave like this.
- **Design and clarity** (8 points)

  Is the program organized clearly? Are functions and variables designed in an appropriate way? Are the tasks divided appropriately? If not, the program will lose points.
- **Compliance with the Programming Rules** (16 points)

  Are all of the rules stated in that document observed? Programs that violate them will lose points accordingly.

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.

6 Submitting the Homework

This is a short project, due on a non-class meeting day. It is due by the end of the day (i.e. 11:59PM, EST) on October 14, 2019. Follow these instructions precisely to submit it.

Your program should be a single source code file, written in C. It must have a `.c` suffix. It does not matter what base name you give it because the `submithwk` program will change its base name anyway to the form `username_hwk2.c` (or more accurately, `username_hwk2.`*XXX*, where *XXX* is the suffix you gave it.)

Assuming this file is in your current working directory and is named `myhwk2.c` you submit by entering the command

```
submithwk_cs49365 -t  2  myhwk2.c
```

The `-t` tells the command it is a plain text file. The program will copy your file into the `hwk2` subdirectory

```
/data/biocs/b/student.accounts/cs493.65/hwks/hwk2/
```

and if it is successful, it will display the message, "`File username_hwk2.c successfully submitted.`"

You will not be able to read this file, nor will anyone else except for me. But you can double-check that the command succeeded by typing the command

```
ls -l /data/biocs/b/student.accounts/cs493.65/hwks/hwk2/
```

and making sure you see a non-empty file there.

If you put a solution there and then decide to change it before the deadline, just replace it by the new version. Once the deadline has passed, you cannot do this. I will grade whatever version is there at the end of the day on the due date. You cannot resubmit the program after the due date.

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.