



Assignment 2

1 Synopsis

Please read the document, *Programming Requirements and Guidelines*, posted on the course website and make sure you follow them when you do this assignment. For this assignment, you'll write your first MPI program, a simple program serving as a warm-up exercise so that you learn how to write and run MPI programs. You have to design and implement 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'll 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 it. 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 the first derivative of $f(x)$, 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 closed 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 actual area is approximately the sum of these areas. As the number of rectangles increases and their widths decrease, the sum of their areas converges to the true area. Your program must use the rectangle rule to compute the area under the curve. To compute the natural log, you'll need the area under a different curve. It's your job to determine which curve you'll need.

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
```

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 (digits to the right of the decimal point.) 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
```

Because we are having problems with the Open MPI installation in our lab, you'll have to limit the run of the program to the lost host only. The only MPI option to use should be `--use-hwthread-cpus` and you should discard the error messages by redirecting them to `/dev/null`. For example the above command for running it would be:

```
mpirun --use-hwthread-cpus natlog 10.0 4096 2> /dev/null
```

To save your output to a file, if you want, you'd enter:

```
mpirun --use-hwthread-cpus natlog 10.0 4096 2> /dev/null > myfile
```

The error message that will appear multiple times is:

```
Authorization required, but no authorization protocol specified
```

Just ignore it. No one as yet know what is wrong.

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 nonnumerical, 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 *Programming Requirements and Guidelines* 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** (60 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** (10 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** (10 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** (20 points)

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

Instructions for Submitting the Assignment

1. Use the `submithwk_cs49365` program to submit your program. It can be run on any `cslab` host, so login to a `cslab` host when you're ready to submit. To submit, if your file is named `myhwk.c`, you'd enter

```
$ cd ~  
$ submithwk_cs49365 -t 2 myhwk.c
```

The program will try to copy `myhwk.c` into the directory

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

and if it is successful, it will display the message, "File `hwk2_username.c` successfully submitted."

where `username` is your username. You will not be able to read this file, nor will anyone else except for me. But you can doublecheck that the command succeeded by entering the command

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

and making sure you see a nonempty file named `hwk2_username.c` where `username` is your user name and whose date of last modification is the time at which you ran the command.

2. *You can do step 1 as many times as you want. Newer versions of the file will overwrite older ones.*



Deadline

You have one week to do this assignment. It must be submitted before its *deadline*, which is **Friday, March 7, at 7:00 PM. After that, you will not receive credit for completing it.**

Grading Rubric

This assignment is **10%** of your final grade. The output must be correct and the program must conform to the rules written in the *Programming Rules* document on the class's webpage. It must be thoroughly documented.