Program #1: Summing n numbers

CptS 411, Fall 2018

Due: September 6, 2018, by 11:59pm on WSU Blackboard dropbox

General Guidelines

Read carefully!

- This is an individual assignment. This means, there should *not* be any collaboration or discussion. You are expected to work on your own and the resulting work should be solely yours. Referring to external sources (including the web) for potential solutions or source code, is strictly *not* allowed and will be considered plagiarism. You are allowed/encouraged to use any materials or ideas provided during the lecture (in class). You are allowed to lookup any C function help if you want to use (e.g., gettimeofday for timing and such basic utility functions).
- You are expected to C/C++. Do not use any built-in libraries or functions to implement your code. The main logic implemented in your code should be entirely yours and written from scratch.
- Submission should contain the following elements: i) source code (in a separate folder); and ii) report (PDF). Please zip the contents under a folder with your last name (in the format `lastname_proj1.zip`) and submit.
- If something is not clear in the question, you can post that question (either as public or as private) on Piazza. Obviously desist from posting any of your code or solutions on Piazza in public mode. That will be viewed as a violation of Piazza use.
- You have to test your program on your own laptop (preferred) or using any of the EECS servers that you have access to, which has a C/C++ compiler. There is no need for any parallel computer for this project.
- Grading will be based on a combination of factors: correctness, design and implementation efficiency, coding style, exception handling, test drivers, source code readability, and the written report.

1 Problem

In this programming project you will write a program that implements and tests two functions:

i) a serial function to add n numbers, and

ii) a function that simulates the parallel addition of n numbers in $\lg p$ time steps.

For the serial function, your API should be:
`int serialSum(int A[], int n)`, where A is the input array with n elements.

For the second function for simulating the parallel addition, your API should be:
`int parallelSumSimulator(int A[],int n, int p)`, where A is the input array with n elements, and p denotes the number of processes.

In both cases, the functions should return the global sum.

For this entire assignment, assume $n = p$ in both your design and for testing purposes. You can also enforce the assumption that p is a power of 2.

You are encouraged to implement both functions as part of the same program, so that you can reuse any utility functions (see Section 2) you may want to implement alongside the main two functions.

Your program executable should take as arguments n and p as input. Since $n = p$ for this assignment, these two arguments should always have the same value. In addition, the value should be a power of 2. Otherwise the program should error out (gracefully!).

Approach: For a given input value of n (same as p), the code should first call a function called `generateInput` which uses n and returns an integer array A of size n. The function should use a random generator function (e.g., `rand` or `drand48()` or some similar variant) to populate each of the n entries in A. This will be your input array. If you want an example to refer to on how to generate random numbers in `C`, refer to the following example: `https://www.tutorialspoint.com/c_standard_library/c_function_rand.htm`.

Once you have generated A, you are going to call both the `serialSum` and `parallelSumSimulator`

functions. The algorithms underlying both these sum functions were described in detail in class on August 28, 2018. The lecture notes are available and posted on the course website: `https://www.eecs.wsu.edu/~ananth/CptS411/Lectures/Scribes/iScribe_ParallelSum.pdf` (pages 1 and 2 show the serial and parallel algorithms respectively). Further notes specific to each of the two implementations are provided below.

1.1 The serial program

It should be pretty clear and simple what this code should do.

1.2 The parallel simulation program

For this part, you are going to basically write a serial code that "simulates" the $\mathcal{O}(\lg p)$ tree-based parallel algorithm that we discussed in class (page 2 of the above lecture notes). The output should be made available in "process" p_0 and it can do the output finally. There is no need for a broadcast.

I call this simulation because you are still going to write a serial code and inside the main body of your summing procedure, you are going to add a `for` loop that iterates from process 0 through process $p-1$.

Algorithm 1 *Pseudocode for Parallel Sum Simulator*

parallelSumSimulator(...)

`for t:  0 to` $\lg(p-1)$ //for each timestep

`for process id i:  0 to p-1.` //for each process

//Implement here what each process id p_i is supposed to do during the t^{th} timestep.

//This part of the code is the heart of your logic. So please think carefully prior to writing the code.

`Output sum.` //You can assume that this output is happening from process p_0.

2 Utility functions

As part of your implementation, you will have to write a few utility functions. These functions can help you time parts of the code, generate an input automatically for a given size, and implement a test driver.

Generating the input: We already described this function `generateInput` in the previous section.

Timing: As part of your code, you should measure runtime and add it in your report. For measuring runtime, you can use functions such as `gettimeofday` or `clock`. But make sure you are timing and reporting only the main body of the code that computes the sum—i.e., don't include time taken for generating input or doing any other Input/Output.

If you are not familiar on how to use these timing functions, you can do a "man" for help on those functions in a Unix environment, or refer to these simple example codes online:

- `gettimeofday`: `http://www.cs.loyola.edu/~jglenn/702/S2008/Projects/P3/time.html`
- `clock`: `https://www.tutorialspoint.com/c_standard_library/c_function_clock.htm`

Test driver: You are expected to write a test driver to adequately test your code for a range of values of n (same as p). Testing should ensure the answers are correct, and also it should give you the necessary timing data so that you can plot in your project report.

3 Project Report- (cannot exceed 2 pages)

Along with your source code, also submit a one-page design document (as a PDF - no other formats allowed).

This report should contain two sections.

Experimental platform: Here you describe the basic architectural features of the machine you used for all your testing. Just use one machine for testing. The features to include are CPU architecture, clock speed, memory size, cache size.

Performance evaluation: For this part, you present the timing results for both functions (serialSum and parallelSumSimulator) as a function of p (same as n). Make sure you don't dump your output results. Instead use a helpful chart so that I can understand the execution behavior of your code. The results should be presentable as in a technical report.

If you observed a variance in the execution times for the same input size, then run your code a bunch of times (say, 10) and provide the mean and standard deviation. Use the mean for the plot.

In addition to the chart, add a short justification in text to say whether what you observe is what you were expecting (or not). You are allowed to speculate for potential reasons in any deviation observed.

4 Deliverables

1. Source code in a folder.
2. Project report as PDF.
3. Any other excel or tables you may have generated (auxiliary files).