

Project 1

MATH 471/571 - Parallel Scientific Computing

Michal A. Kopera

Due date: Feb 19th, 2020

Please complete the tasks outlined below and submit a PDF write-up along with the code and other files (Makefile, run script) you have used to obtain the results. You submit your solution by putting it in a folder named Project 1 in your Google Drive personal folder.

I will grade your project and provide you with some feedback. You will have about a week to correct any issues. Please refer to the Syllabus regarding how the project points and mastery points affect your final grade. You can earn one project point for each complete task, and one mastery point for completing the mastery part. There is no partial credit, so you have to complete all the steps to earn a point.

The projects are your individual work. I am happy to clarify some muddy points, but I would like to see your individual attempt to solve the project. If you got help from somebody, or used a resource outside class (i.e. website) please provide appropriate acknowledgement and/or reference. **You can** discuss your ideas on Slack, but **you cannot** share your codes.

In this project we will continue working with the standard deviation example, and will explore whether different algorithms, which lead to the same result, have different parallel scalability.

Task 1 - The “naive” standard deviation algorithm

For this task, you will need the parallel standard deviation code we have discussed in class. If you have completed recent homework assignment, you will have that program almost ready to go. If you haven't, please follow the steps I have shown you in Week 5 lecture to write your own MPI standard deviation code.

The one change from the code we worked on in class is the data set. Instead of reading data from a file and scattering it, create a data array using a random number generator function (make sure you have included `sodlib.h`):

```
double random_number(){
```

```
    return (double)rand() / (double)RAND_MAX ;
}
```

Make sure the problem size N is the input parameter to the code, rather than is read from the file. The random function above generates a random number between 0 and 1 by sampling a uniform distribution. The standard deviation of such distribution is

$\sigma_{exact} = \frac{1}{\sqrt{12}}$. This should help you to validate whether your code works correctly.

Complete the following steps:

1. Run a strong scaling experiment using $nproc=1,2,4,8,16,32,64$ and problem size $N = 2^{28}$. In your write-up, show the speedup and efficiency plots. Comment on what is happening. How does the parallel efficiency change with the number of processes?
2. Run a weak scaling experiment using the same number of processors, starting with $nproc=1$ and $N = 2^{22}$ and doubling the problem size each time you double the number of processes. In your write-up show the weak scaling efficiency plot and comment on it.

Task 2 - Welford's algorithm

The Welford's algorithm computes the standard deviation, but does not require two passes through the data - one to compute the mean, the other to compute the standard deviation. Rather, it completes one pass through the data performing the following operations:

$$1) \bar{x}_0 = x_0$$

For k increasing from 1 to $N-1$, repeat:

$$2) \bar{x}_k = \bar{x}_{k-1} + \frac{x_k - \bar{x}_{k-1}}{k}$$

$$3) M_k = M_{k-1} + (x_k - \bar{x}_k)(x_k - \bar{x}_{k-1})$$

then compute the standard deviation by:

$$4) \sigma = \sqrt{\frac{M_{N-1}}{N}}$$

To complete this task, write a serial program implementing this algorithm and show that it gives the same result as the one in Task 1.

Task 3 - Parallel implementation of Welford's algorithm

Welford's algorithm can be modified to be performed in parallel. Suppose we have two sets of data points: x_A and x_B , with N_A and N_B points, respectively, means \bar{x}_A and \bar{x}_B and squared sums M_A and M_B . (you have computed \bar{x} and M in Task 2). Let's say that two sets together form a set x with $N = N_A + N_B$ elements.

Then the mean of the set x is:

$$\bar{x} = \bar{x}_A + (\bar{x}_A - \bar{x}_B) \frac{N_A}{N}$$

and

$$M = M_A + M_B + (\bar{x}_A - \bar{x}_B)^2 \frac{N_A N_B}{N}$$

which finally gives

$$\sigma = \sqrt{\frac{M}{N}}.$$

Complete the following steps:

1. Use the idea above to create a parallel version of Welford's algorithm. Present the algorithm as an ordered list of steps or make a graphic representation similar to what we used in class. Identify which collective communication operations need to be used.
2. Implement the parallel algorithm as a program and show that it produces correct result.
3. Create strong scaling experiment using $nproc=1,2,4,8,16,32,64$ and problem size $N = 2^{28}$. In your write-up, show the speedup and efficiency plots and comment on how they compare to strong scaling in Task 1.
4. Run a weak scaling experiment using the same number of processors, starting with $nproc=1$ and $N = 2^{22}$ and doubling the problem size each time you double the number of processes. In your write-up show the weak scaling efficiency plot and comment on how it compares to weak scaling in Task 1.

Mastery

Time the computation and communication time for both algorithms separately. Compare them in one plot. What conclusions can you draw from the plot?

If the scaling plots in Task 3 do not look much different from Task 1, try a significantly smaller (or significantly larger) problem size to see whether the problem size affects the strong and weak parallel efficiency.

(optional) You can also look into timings for individual MPI operations to investigate how they affect the total communication time. There are better and worse ways of implementing the algorithm in Task 3, so timing of individual routines can help you figure out which are the most time-consuming operations.