Homework #1

ME 471/571

All of your work for this assignment should be turned in using a Jupyter notebook. Turn in a single notebook (the `.ipynb` file) for all problems. Be sure all of your results are visible, because I will not re-run your notebook. You may also turn in a PDF version of the notebook, but these don't usually look as nice as the notebook itself.

1. (**Welford's Algorithm (serial)**). Assume you have an array of numbers $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Formulas for the mean, sample variance and standard deviation are given by

$$\bar{x}_n \equiv \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \text{(Mean)}$$

$$S^2 \equiv \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x}_n)^2, \qquad \text{(Sample variance)}$$

$$S \equiv \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x}_n)^2}, \qquad \text{(Standard deviation)}$$

The goal of this problem is to develop a serial, *single pass* algorithm for computing the above statistics.

(a) Show analytically that the mean can be incrementally updated using the formula

$$\bar{x}_n = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$

(b) Define

$$M_n \equiv \sum_{i=1}^{n}(x_i - \bar{x}_n)^2$$

Show analytically that M_n can be incrementally updated using

$$M_n = M_{n-1} + (x_n - \bar{x}_n)(x_n - \bar{x}_{n-1})$$

(c) Use the above ideas to write a serial program that computes the mean, sample variance and standard deviation in a single-pass. The resulting algorithm is essentially the serial *Welford's Algorithm* for computing these statistics.

(d) How close to the value $1/\sqrt{12}$ can you get for the standard deviation of a set of uniformly distributed numbers? Show, either in a table or a plot, the standard deviation of your array $\mathbf{x}$ verses size of the array.

Turn in either hand-written work (or LaTeX) for the first two questions. For the serial algorithm, turn in a Jupyter notebook. Verify that you are getting the correct results by comparing your statistics to the NumPy results using `np.mean(x)`, `np.var(x)` and `np.std(x)`. Show that your statistics compare to the NumPy's results to essentially machine precision. Generate your data set using the random number generator used in class.

2. (**Parallel Welford's Algorithm**). Suppose we have two sets of data X_A and X_B with means $\bar{X}_A$ and $\bar{X}_B$, squared sums M_A, M_B and counts N_A and N_B. These quantities can be combined to form aggregate statistics $\bar{X}_X$ on the combined data set X with count N_X.

(a) Show that

$$\bar{X}_X = \bar{X}_A + (\bar{X}_B - \bar{X}_A)\frac{N_B}{N_X}$$

(b) Use Problem 2a to show that the related expression

$$\bar{X}_X = \bar{X}_B + (\bar{X}_A - \bar{X}_B)\frac{N_A}{N_X}$$

also holds.

(c) Show that

$$M_X = M_A + M_B + (\bar{X}_B - \bar{X}_A)^2 \frac{N_A N_B}{N_X}$$

(d) Combine the above ideas to write a single pass, parallel version of Welford's Algorithm for computing the mean, sample variance and standard deviation.

Turn in either hand-written work (or LaTeX) for the first two questions. For the parallel algorithm, turn in a Jupyter notebook for two different systems (your laptop plus one other system). For each notebook, you should also

- Verify that you are getting the correct results by comparing your statistics to the NumPy results.
- Use 'timeit' to get timing results
- Display your timing results using Pandas
- Create a plot showing strong scaling results
- Create a plots showing efficiency

Generate your data set using the random number generator used in class.

3. (**Approximate** π) Use a Monte-Carlo method to approximate the value of π. The algorithm is as follows.

- Generate a sequence of N random pairs of numbers $(x_i, y_i) \in [-1, 1] \times [-1, 1]$. Your N should be as large as possible, $N \sim 10^6$.
- Count the number of values K in the sequence which are in the circle of radius 1, centered at the origin.
- The ratio of the number of pairs in the circle to N should approach the ratio of the area of the circle to the area of the enclosing box. Use this to approximate π.

Run this on a single processor for a sequence of N values and report the value and timing results. Then, run this using N points on p processors, gathering the results and reporting the timing and your approximation. You should be able to show that by using more processors, you can get a better approximation for essentially the same total time. Create a *weak scaling* plot.

There are two ways to do this problem. You can either

(a) Create a single large array in the parent process and distribute it to child processes (as we have done in class).

(b) Let each process generate its own random numbers. To do this, you will need to seed each process with a distinct seed to ensure that you get truly random numbers.

For this problem, use the multiprocessing function "Pool".

4. (**Trapezoidal rule**) Approximate the following definite integral using the Trapezoidal Rule. Your code should run on 1,2,4, and 8 processors.

$$I = \int_{-1}^{1} (x-1)^2 e^{-x^2}\, dx$$

(a) Use Wolfram Alpha (`www.wolframalpha.com`) to find the exact expression for definite integral above. The correct expression will involve e, π, and the function `erf(x)`.

(b) Verify that your code is correct by running it on a sequence of intervals $N = 2^p$, for $p = 10, 11, 12, \ldots, 28$. Verify that your results are correct by showing numerical convergence, and show performance results by showing plots of strong and weak scaling and efficiency. You may use the notebooks on the course website to get started with this.

Use the multiprocessing module and a Pool of workers. Does the scaling improve when you don't have to pass data to each subprocess?