Spring 2019: Advanced Topics in Numerical Analysis: High Performance Computing
Assignment 3 (due Apr. 1, 2019)

Handing in your homework: Hand in your homework as for the previous homework assignment (git repo with Makefile), answering the questions by adding a text or a LaTeX file into your repo. The git repository https://github.com/NYU-HPC19/homework3.git contains the code you can build on for this homework.

1. **Approximating Special Functions Using Taylor Series & Vectorization.** Special functions like trigonometric functions can be expensive to evaluate on current processor architectures which are optimized for floating-point multiplications and additions. In this assignment, we will try to optimize evaluation of $\sin(x)$ for $x \in [-\pi/4, \pi/4]$ by replacing the builtin scalar function in C/C++ with a vectorized Taylor series approximation,

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \cdots$$

The source file `fast-sin.cpp` in the homework repository contains the following functions to evaluate $\{\sin(x_0), \sin(x_1), \sin(x_2), \sin(x_3)\}$ for different $x_0, \ldots, x_3$:

   - `sin4_reference()`: is implemented using the builtin C/C++ function.
   - `sin4_taylor()`: evaluates a truncated Taylor series expansion accurate to about 12-digits.
   - `sin4_intrin()`: evaluates only the first two terms in the Taylor series expansion (3-digit accuracy) and is vectorized using SSE and AVX intrinsics.
   - `sin4_vec()`: evaluates only the first two terms in the Taylor series expansion (3-digit accuracy) and is vectorized using the Vec class.

Your task is to improve the accuracy to 12-digits for **any one** vectorized version by adding more terms to the Taylor series expansion. Depending on the instruction set supported by the processor you are using, you can choose to do this for either the SSE part of the function `sin4_intrin()` or the AVX part of the function `sin4_intrin()` or for the function `sin4_vec()`.

Extra credit: develop an efficient way to evaluate the function outside of the interval $x \in [-\pi/4, \pi/4]$ using symmetries. Explain your idea in words and implement it for the function `sin4_taylor()` and for any one vectorized version. Hint: $e^{i\theta} = \cos\theta + i\sin\theta$ and $e^{i(\theta+\pi/2)} = ie^{i\theta}$.

2. **Parallel Scan in OpenMP.** This is an example where the shared memory parallel version of an algorithm requires some thinking beyond parallelizing for-loops. We aim at parallelizing a scan-operation with OpenMP (a serial version is provided in the homework repo). Given

a (long) vector/array $\boldsymbol{v} \in \mathbb{R}^n$, a scan outputs another vector/array $\boldsymbol{w} \in \mathbb{R}^n$ of the same size with entries

$$w_k = \sum_{i=1}^{k} v_i \text{ for } k = 1, \dots, n.$$

To parallelize the scan operation with OpenMP using p threads, we split the vector into p parts $[v_{k(j)}, v_{k(j+1)-1}]$, $j = 1, \dots, p$, where $k(1) = 1$ and $k(p+1) = n+1$ of (approximately) equal length. Now, each thread computes the scan locally and in parallel, neglecting the contributions from the other threads. Every but the first local scan thus computes results that are off by a constant, namely the sums obtains by all the threads with lower number. For instance, all the results obtained by the the r-th thread are off by

$$\sum_{i=1}^{k(r)-1} v_i = s_1 + \cdots + s_{r-1}$$

which can easily be computed as the sum of the partial sums $s_1, \dots, s_{r-1}$ computed by threads with numbers smaller than r. This correction can be done in serial.

- Parallelize the provided serial code. Run it with different thread numbers and report the architecture you run it on, the number of cores of the processor and the time it takes.