Spring 2017: Advanced Topics in Numerical Analysis: High Performance Computing Assignment 1 (due Feb. 9, 2017)

1. **Describe a parallel application and the algorithms used.** Find and examine an application problem for which high-performance computing has been used. Pick a problem from your own research, or find a problem elsewhere. Prepare a 1–2 page description of the problem and describe where and how successful high-performance computing has been/is used. Consider to include the following:

   (a) What's the application problem being solved?

   (b) Why does the problem require large/fast computation?

   (c) What are the underlying mathematical/parallel algorithms?

   (d) If the application uses a supercomputer, where is that computer on the Top500 list (http://www.top500.org/)? Try to say a few words about the architecture of that machine.

   (e) How well does the algorithm perform? Does it "scale"?

   If you are looking for an application, take a look at the papers from one of the previous Supercomputing conferences.[1] Alternatively, take a look at the National Science Foundation (NSF)-funded supercomputing centers, which usually have science stories with links to papers on their websites.[2] Please hand in this description as a separate PDF file by mailing it to the TA (Bill Bao, yxb201@nyu.edu). I will post all descriptions on Piazza to serve as with an overview of research topics that require HPC resources.

2. **Write a program to solve the Laplace equation in one space dimension.** For a given function $f : [0, 1] \to \mathbb{R}$, we attempt to solve the linear differential equation

$$-u'' = f \text{ in } (0, 1), \text{ and } u(0) = 0, u(1) = 0 \tag{1}$$

   for a function u. In one space dimension[3], this so-called *boundary value problem* can be solved analytically by integrating f twice. In higher dimensions, the analogous problem

[1] See http://sc16.supercomputing.org/full-program/ or http://sc15.supercomputing.org/schedule.html and choose to filter for "papers".

[2] Oark Ridge National Laboratory: https://www.olcf.ornl.gov/;
National Energy Research Scientific Computing Center (NERSC): https://www.nersc.gov/;
San Diego Supercomputing Center: http://www.sdsc.edu/;
Nasa Advanced Supercomputing Division: http://www.nas.nasa.gov/;
Texas Advanced Computing Center (TACC): https://www.tacc.utexas.edu/

[3] The generalization of (1) to two and three-dimensional domains Ω instead of the one-dimensional interval $\Omega = [0, 1]$ is the *Laplace equation*,

$$\begin{aligned} -\Delta u &= f \text{ on } \Omega, \\ u &= 0 \text{ on } \partial\Omega, \end{aligned}$$

which is one of the most important partial differential equations in mathematical physics.

often cannot be solved analytically and one must rely on numerical approximations for u. We use a finite number of grid points in $[0,1]$ and finite-difference approximations for the second derivative to approximate the solution to (1). We choose the uniformly spaced points $\{x_i = ih : i = 0, 1, \ldots, N, N+1\} \subset [0,1]$, with $h = 1/(N+1)$, and approximate $u(x_i) \approx u_i$ and $f(x_i) \approx f_i$, for $i = 0, \ldots, N+1$. Using Taylor expansions of $u(x_i - h)$ and $u(x_i + h)$ about $u(x_i)$ results in

$$-u''(x_i) = \frac{-u(x_i - h) + 2u(x_i) - u(x_i + h)}{h^2} + \text{h.o.t.},$$

where h.o.t. stands for a remainder term that is of higher order in h, i.e., becomes small as h becomes small. We now approximate the second derivative at the point x_i as follows:

$$-u''(x_i) \approx \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2}.$$

This results in the following finite-dimensional approximation of (1):

$$A\boldsymbol{u} = \boldsymbol{f}, \tag{2}$$

where

$$A = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix}, \qquad \boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix}, \qquad \boldsymbol{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix}.$$

Simple methods to solve (2) are the Jacobi and the Gauss-Seidel method, which start from an initial vector $\boldsymbol{u}^0 \in \mathbb{R}^N$ and compute approximate solution vectors $\boldsymbol{u}^k$, $k = 1, 2, \ldots$. The component-wise formula for the Jacobi method is

$$u_i^{k+1} = \frac{1}{a_{ii}}\left(f_i - \sum_{j \neq i} a_{ij} u_j^k\right),$$

where a_{ij} are the entries of the matrix A. The Gauss-Seidel algorithm is given by

$$u_i^{k+1} = \frac{1}{a_{ii}}\left(f_i - \sum_{j<i} a_{ij} u_j^{k+1} - \sum_{j>i} a_{ij} u_j^k\right).$$

If you are unfamiliar with these methods, please take a look at the Wikipedia entries for the Jacobi[4] and the Gauss-Seidel[5] methods. Note that due to the sparsity of the matrix A, the Jacobi and the Gauss-Seidel iterations given above simplify substantially and amount to simple averaging between neighboring grid points.

[4] http://en.wikipedia.org/wiki/Jacobi_method
[5] http://en.wikipedia.org/wiki/Gauss-Seidel_method

(a) Write a program in C that uses the Jacobi or the Gauss-Seidel method to solve (2), where the number of discretization points N is an input parameter, and $f(x) \equiv 1$, i.e., the right hand side vector $\boldsymbol{f}$ is a vector of all ones.

(b) After each iteration, output the norm of the residual $\|A\boldsymbol{u}^k - \boldsymbol{f}\|$ on a new line. Terminate the iteration when the initial residual is decreased by a factor of 10^4 or after a maximum of 1000 iterations. Start the iteration with a zero initialization vector, i.e., $\boldsymbol{u}^0$ is the zero vector.

(c) Compare the number of iterations (or, if the maximum number of iterations is reached, the residual reduction after 1000 iterations) needed for the two different methods for $N = 1000$ and $N = 100,000$ grid points. Compare the run times for $N = 100,000$ using different compiler optimization flags (`-O0` and `-O3`). Hand in the results and a listing of your program. Specify which computer architecture you used for your runs. Make sure you free all the allocated memory before you exit.