Mathematics Department, University of Massachusetts Dartmouth

High Performance Scientific Computing
MTHEAS 520 – Section 01 – Spring 2015
Project 3
OpenMP
Due May 5, 2015

There are two major tasks for this project:

a. Rerun the Lorentz attractor simulation from the previous project, but this time parallelize it using OpenMP.
b. Write a serial and OpenMP-parallelized version of a matrix QR factorization.

As with the previous project, all these should be written to run on the cluster.

Please understand that the cluster is a shared resource – do not hog several nodes for yourself for an extended period of time.

1 Chaos in the Lorentz attractor

Consider the serial Lorentz attractor code from Project 2 (repository `lorentz` on the class git server). Parallelize the code to produce a particle cloud just as in Project 2, but this time using OpenMP.

Your ultimate task is to reproduce, extend, and/or augment the scatterplots in Figure 2 from Project 2.

- You are free to modify the code in the `lorentz` repository as much or as little as you like. But your submitted code must employ OpenMP directives to foster computational parallelism.
- To generate standard normal random variables if you cannot generate them natively (e.g., in C), then you are free to use any external library on the Internet that you wish. However, it may be simpler to use the Box-Muller transform:
  see `http://en.wikipedia.org/wiki/Normal_distribution#Generating_values_from_normal_distribution`,
  `http://en.wikipedia.org/wiki/Box-Muller_transform`.
- The parameter choices $\sigma = 10$, $\beta = \frac{8}{3}$, and $\rho = 28$ are known to produce chaos, but many others do, too. You may change the parameter values to anything you wish.
- Your submission should contain your C or Fortran code with an appropriate makefile, and bash/pbs scripts.
- Your report should describe what problem you are trying to solve and what your code does. It should contain whatever scatterplots you were able to generate, and should give a high-level description of your approach for parallelism.

- For an optional challenge: make a video of the particle cloud evolution. (Upload it onto a data partition of the cluster, do not submit it via git.)

2 The QR factorization

Let A be an $m \times n$ matrix, and for simplicity we'll assume that $m \geq n$ and that A has full rank. There is a matrix decomposition

$$A = QR,$$

where (i) Q is an $m \times n$ matrix whose columns are orthonormal, and (ii) R is a square $n \times n$ matrix that is upper-triangular and invertible. The columns of the matrix Q are an orthonormal basis for the column span of A. Loosely speaking, this decomposition takes the columns of A (that are not necessarily orthogonal) and transforms them into an orthonormal set of vectors that are stored as the columns of Q; the coefficients that transform the columns of Q back into the columns of A are given as the entries of R.

One straightforward way to compute the matrices Q and R is to perform Gram-Schmidt orthogonalization: let $\mathbf{a}_1, \ldots \mathbf{a}_n$ be te columns of A. For $j = 1, \ldots, n$:

- Compute $r_{j,j} = \|\mathbf{a}_j\|$
- Set $\mathbf{q}_j = \mathbf{a}_j / r_{j,j}$
- For $k = j+1, \ldots, n$:
  - Compute $r_{j,k} = \langle \mathbf{a}_k, \mathbf{q}_j \rangle$
  - Set $\mathbf{a}_k \leftarrow \mathbf{a}_k - r_{j,k}\mathbf{q}_j$

At the end of the algorithm, the scalars $r_{j,k}$ are the entries of the matrix R and the vectors $\mathbf{q}_j$ are the columns of the matrix Q.

Your task is to write C or Fortran code that performs a QR factorization using the Gram-Schmidt algorithm above, which uses OpenMP to accelerate computations. You may assume that the input $m \times n$ matrix A has full rank and that $m \geq n$.

- You will probably find it helpful to write a serial version of the pseudocode above first. After that, try to look at your serial code and figure out where you can take advantage of parallelism.
- Your report should describe what problem you are trying to solve and what your code does. It should give a high-level description of your approach for parallelism and, if possible, timings illustrating efficiency of the parallelization. In order to see the benefits of parallelism, you should test your code on relatively large matrices, e.g., a 1000×300 matrix $\mathbf{A}$.
- For an optional challenge: write a *column-pivoted* QR decomposition routine that is parallelized using OpenMP. A column-pivoted factorization generates a decomposition $AP = QR$, with P a permutation matrix that is defined so that the diagonal elements of R satisfy $r_{j,j} \geq r_{j+1,j+1}$ for all j. This can be accomplished by inserting an initial step within the j-counter loop: (i) find the column index ℓ so that $\|\mathbf{a}_\ell\| \geq \|\mathbf{a}_k\|$ for $k = j, \ldots, n$, and (ii) swap columns $\mathbf{a}_\ell \leftrightarrow \mathbf{a}_j$. The permutation matrix $\mathbf{P}$ should keep track of this swap, and you can perform this swap without actually moving any of the elements of $\mathbf{A}$ around.

3 Submit

You must submit (1) your source code (C and/or Fortran), and (2) your LaTeX source code (*not* the dvi/ps/pdf output) via a Git repository. You are submitting to the `john-smith-project-3` repository on the class git server, where you should replace `john-smith` with your first and last name.