

Homework 1: Parallel Matrix Power (via Prefix)

Notes:

- Your code only has to work for numbers of processors that are powers of 2. (You need to support arbitrary matrix powers; however, if you follow the online notes about parallel prefix, this is straightforward.)
- If you are careful, the OpenMP solution should be fairly straightforward (and in-place, after an initial cost to copy data around). The MPI solution can also be done in a uniform way (recursive calls to a prefix-sum procedure) but takes more care.
- Submit the homework as before (see the hw0 instructions). Include a .pdf file called writeup.pdf describing your approach, any problems you encountered, and including the plots called for in question 2. Be sure to comment your code if you want partial credit.

All the files you need for this problem set are available in the Assignments section.

The Problem: You are given an $n \times n$ matrix A and asked to compute all its powers (up to l). That is, you want A, A^2, \dots, A^l . You are to implement parallel solutions to this problem in MPI and in OpenMP and measure their performance. Check the on-line notes and readings for more information on parallel prefix if you need a review.

Question 1 - Basic Implementation

Fill in PrefixMPI.c and PrefixOpenMP.c such that they function correctly on the 5×5 identity matrix and the 5×5 swap matrix. Verify this by running them to compute the 100th powers of each.

The boilerplate code assumes that an input file (specifying the size and contents of the matrix whose powers are being computed) and an integer specifying the highest power are provided on the command line. The input file format is simply a number indicating the number of dimensions of the matrix followed by the matrix contents. The boilerplate code produces an output log (in e.g. mpi-dimension-power-num-processors.out, so running the MPI code on a 4×4 matrix to the 8th power on 2 processors would yield mpi-4-8-2.out) containing the elapsed time for the guts of the computation and the highest power. (Note: Although your code is only outputting the highest power, it will be very clear to us if the intermediate powers are not computed when we examine your source.)

We have provided code to allocate, read and print matrices, represented as double arrays. We have also provided preprocessor macros to get and set their contents. You are free to

use or ignore these. You are free to use the naive $O(n^3)$ algorithm for doing serial matrix multiplication, or come up with something more clever instead.

Question 2 - Performance Measurements

Construct test matrices (e.g. via `randn` and save from matlab) of sizes $n=50$, $n=100$ and $n=500$. Name these `test-50.mat`, `test-100.mat` and `test-500.mat` respectively. Also produce their 10th powers in files `test-n-10.mat` via a trusted program.

For each n , produce clear plots (you may decide how to present the results) showing how performance of your OpenMP and MPI implementations scales as a function of power and number of processors. Note that the more the power dwarfs the number of processors, the more the initial $O(n/p)$ term will dominate the running time. Clearly explain any salient performance trends you see.

Note that your OpenMP tests will probably be very quick, since you can only really choose 2 or 4 processes. Be sure to exercise the full capacity of the cluster for your MPI tests, however.