

CS 140 Assignment 3: Matrix-Vector Multiplication and the Power Method

Assigned January 24, 2008

Due by 11:59 pm Thursday, February 7

This assignment is to write a parallel program to multiply a matrix by a vector, and to use this routine in an implementation of the power method to find the magnitude of the largest eigenvalue of the matrix. You will write two (or more for extra credit) versions of the matrix-vector multiplication routine. You will write separate functions to generate the matrix and to perform the power method, and you will do some timing experiments with the power method routine.

1 The power method

The *power method* estimates the size of the largest eigenvalue of a matrix A (which is also called the *spectral radius* of A .) It works as follows. First, generate a random vector x . Then repeat the following two steps: divide x by its length (or *norm*), then replace x by the matrix-vector product Ax . The length of the vector eventually converges to the spectral radius of A . In your code, you will repeat the matrix-vector product 1000 times.

There is a sequential Matlab code for the power method on the course web page (under Homework 3). You will write a parallel C/MPI code that does the same computation.

2 What to implement

You will write the following C routines:

- **generateMatrix**: Generates a matrix of specified size, with the data distributed across the processors as specified in the next section. The section on experiments below defines the matrices you'll use for experiments.
- **powerMethod**: Implements the power method on a given matrix (which is already distributed across the processors). This routine calls **norm** and **matVec**.
- **norm**: Computes the norm of a given vector.
- **matVec**: Multiplies a given matrix (which is already distributed across the processors) by a given vector. You will actually write two different versions of this routine, as described in the next section.
- **main**: The main routine that calls **generateMatrix** and **powerMethod**, and times **powerMethod**.

3 Where's the data?

You may assume that the number of rows and columns of the matrix is divisible by the number of processors. Distribute the matrix across the processors by rows, with the same number of rows on each processor, as in the `matvec` example in class on January 24. Put the vector on processor 0. Your `generateMatrix` routine should not need to do any communication except for the size of the matrix; each processor can generate its own rows of the matrix, independently of the others, in parallel.

You'll actually test two different versions of `matVec`, one that uses `MPI_Send` and `MPI_Recv` and one that uses `MPI_Bcast` and `MPI_Gather`. (See "Extra credit" below for other data distributions.)

4 What experiments to do

First, debug your code on very small matrices, on one processor, then two processors, then several processors. Two matrices you can use for debugging are the n -by- n matrix of all ones (whose spectral norm is n) and the identity matrix (with ones on the main diagonal and zeros elsewhere), whose spectral radius is 1. For debugging you should probably only do a few iterations of the power method instead of 1000.

Your code should use `MPI_Wtime`, and it should only time the call to `powerMethod`, not the matrix generation.

For your timing experiments you'll use an n -by- n matrix that has the value n on the main diagonal and also on the diagonals above and below the main diagonal, and zeros everywhere else. Here is the matrix for $n = 6$:

$$\begin{pmatrix} 6 & 6 & 0 & 0 & 0 & 0 \\ 6 & 6 & 6 & 0 & 0 & 0 \\ 0 & 6 & 6 & 6 & 0 & 0 \\ 0 & 0 & 6 & 6 & 6 & 0 \\ 0 & 0 & 0 & 6 & 6 & 6 \\ 0 & 0 & 0 & 0 & 6 & 6 \end{pmatrix}$$

Choose a value of n for which your code runs on one processor in a reasonable amount of time, say 30 seconds to a minute. (On my one-processor laptop, $n = 3000$ takes about 40 seconds.) Run your code on this matrix for a range of different numbers of processors, from $p = 1$ to at least $p = 32$. For each run, report the running time and the parallel efficiency. Make plots of the running time versus p , and the parallel efficiency versus p .

Repeat this whole experiment twice, once with your version of `matVec` that uses send and receive, and once with your version that uses broadcast and gather.

Write a report that describes your experiments and your results. What trends do you see? Do the running time and efficiency behave as you would expect? Can you explain your results? (Warning: Experimental timing results on parallel codes are often not nearly as clean as you might expect from the theory!)

5 Extra credit

For extra credit, you can repeat this whole experiment with a different data layout: Distribute the matrix across the processors by columns instead of by rows. Then you will probably want to use `MPI_Scatter` and `MPI_Reduce` as the building blocks of `matVec`. How do your results compare to those of the row-wise layout?

6 What to turn in

Turn in

- Your source code.
- A `Makefile` that compiles your code.
- A report (as a text file, word file, or pdf) containing instructions for compiling and running your code, plus tables of your run time results, plus a report that describes and interprets your results and any conclusions you draw from them.
- If you worked in a team, your report should include the names and perm numbers of both members.
- Don't turn in any executable or `.o` files.

You may do this assignment alone or in groups of two.