

# Homework 2: Parallel Sparse Conjugate Gradients

## Notes:

1. Submit your solutions as before (see instructions for Homeworks 0 or 1). Compile all your conclusions and timing information for both parts into *one* single PDF or PS document.
2. More notes and hints to follow ...

## Problem Statement

In this problem set, you will implement a parallel iterative solver for solving a linear system of equations of the form

$$\mathbf{A} x = b \quad (1)$$

where  $\mathbf{A} \in \mathbb{R}^N \times \mathbb{R}^N$  is *sparse, symmetric* and *positive definite* and  $x, b \in \mathbb{R}^N$ .

The matrix must be stored using either the *compressed row* or *compressed column* formats discussed in the class. Further, in a distributed memory setting, the matrix will be row distributed and you may assume that each process has the same number of rows

(=  $N/NP$ ).

## Compressed Row Storage

Let  $NNZ$  be the number of non-zero entries of  $\mathbf{A}$ . Then  $\mathbf{A}$  can be stored in the compressed sparse row format using three arrays, `vals`, `cols` and `rows` as follows: `vals`

$\in \mathbb{R}^{NNZ}$  stores the non-zero entries in  $\mathbf{A}$  in row-major order; `cols`  $\in \mathbb{R}^{NNZ}$  stores

the column numbers for the non-zero entries for each row; `rows`  $\in \mathbb{R}^{N+1}$  stores the cumulative number of non-zero entries in each row. For example, if

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

then `vals = {2, -1, -1, 2, -1, -1, 2, -1, -1, 2}`, `cols = {0, 1, 0, 1, 2, 1, 2, 3, 2, 3}` and `rows = {0, 2, 5, 8, 10}`.

## The Conjugate Gradient Method

The pseudocode for solving (1) using the conjugate gradient method is given in Appendix B.2 (pp. 50) of Shewchuk's tutorial, "An introduction to the conjugate gradient method without the agonizing pain" available from

<http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf>

You are however free to use the pseudocode from any other reference (e.g. Numerical Recipes, Golub and Van Loan, etc.) for implementing your solver.

## MPI

We have provided a skeleton C program that reads in the rows of  $A$  corresponding to each process and generates the arrays `vals`, `rows` and `cols`. We have also provided a number of utility functions for operating on sparse matrices and vectors. You may write your own or add to the existing set of functions if you wish.

You must complete the functions `MatVec` and `CG` in `SparseMatrix.c`. You are allowed to store only *one* vector of length  $N$  in each process; all other vectors must be of length

$N/NP$ . While this leads to a sub-optimal implementation in terms of storage, it is quite satisfactory for illustrating the basic principles involved.

The driver program, `SparseMatrixTest.c` and `Makefile` are also provided. To test your code, use

```
mpirun -np NP ./SparseMatrixText.exe Problem Epsilon MaxIters
```

where `Problem` is the linear system being solved, `Epsilon` is the tolerance and `MaxIters` is the maximum number of iterations. For consistency, set `Epsilon` to  $1e-7$  and `MaxIters` to a very large number (like 10000) in all your tests.

We have provided the matrices and right-hand sides for three test cases as MAT files. To generate the input files for each run, load the matrices into MATLAB and run `DistributeMatrices(A, b, 'Problem', NP)`. Study the performance of your code

with 2, 4, 8 and 16 processors (where possible, i.e.,  $N/NP$  is an integer) and comment on your results.

## Star-P

Implement a parallel sparse conjugate gradient solver in Matlab\*P using the same pseudocode that you used for your MPI implementation. Study the performance of your

code for 2, 4, 8 and 16 processors and compare it with the performance of your MPI implementation.

Finally, comment on the time it took you to implement and debug your solver using the two approaches. How long did it take you to code and debug each implementation? How much performance gain did you achieve (if any) by using C and MPI vs. Matlab? Was this gain, in your opinion really worth the time and effort spent in programming and debugging low-level message-passing calls?