ECE 420 Parallel and Distributed Programming Lab 3: Solving a Linear System of Equations via Gauss-Jordan Elimination using OpenMP[*]

Winter 2024

1 Background

Consider the problem of solving the following linear system of equations

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n &= b_2 \\ ... &\quad ... \\ a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n &= b_n. \end{aligned}$$

By denoting a coefficient matrix of

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & a_{nn} \end{pmatrix},$$

a constant vector of

$$\vec{b} = \begin{pmatrix} b_1 \\ b_2 \\ ... \\ b_n \end{pmatrix},$$

[*]In this manual, all the indices start from 1.

and a variable vector of

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix},$$

the linear system of equations can be represented as

$$\mathbf{A} \cdot \vec{x} = \vec{b},$$

or characterized as the augmented matrix $\mathbf{G}$,

$$\begin{aligned} \mathbf{G} &= \{\mathbf{A}|\vec{b}\} \\ &= \left(\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right). \end{aligned}$$

Since an augmented matrix can be mapped back to a linear system of equations, and vice versa, any operation on the original system of equations can be mapped to a corresponding equivalent operation on the augmented matrix $\mathbf{G}$. There are 3 types of linear operations (or row operations) that will not change the solution(s) of the linear system of equations:

1. interchanging any two rows;
2. multiplying each element of a row by a nonzero scalar;
3. adding onto a row with a scalar multiple of another row.

For these row operations, we use the following notations:

1. $R_i \leftrightarrow R_j$: interchange the i^{th} row and the j^{th} row;
2. αR_i: multiply each element of row i by a nonzero α;
3. $R_i + \alpha R_j$: add onto row i with the product between scalar α and row j.

To solve the linear system of equations, the basic idea is to transform the original linear system into an equivalent new system through a series of linear operations. The new system should be reduced to a good form such that every equation (row) in the system has exactly one variable with nonzero coefficient, from which we can simply "read" the solutions. In other words, the resultant augmented matrix should be in the following form:

$$\left(\begin{array}{cccc|c} d_{11} & 0 & ... & 0 & b'_1 \\ 0 & d_{22} & ... & 0 & b'_2 \\ ... & ... & ... & ... & ... \\ 0 & 0 & ... & d_{nn} & b'_n \end{array}\right).$$

The resultant augmented matrix is obtained through Gauss-Jordan Elimination.

1.1 Gaussian Elimination with Partial Pivoting

Gaussian Elimination transforms the augmented matrix into its equivalent "upper triangular" form, in which the elements below the main diagonal are all zeros. It iteratively eliminates the elements below the main diagonal from the first column to the last column via row operations. Algorithm 1 describes this procedure.

Note that the partial pivoting is important in this procedure, as it avoids cases where u_{kk} is zero or close to zero. Thus, with partial pivoting, the program becomes more numerically stable. Also, note that in the row replacement operation, j starts from k, since the first $k-1$ elements are always zero in this algorithm.

1.2 Jordan Elimination

After obtaining the "upper triangular" **U** from Gaussian Elimination, Jordan Elimination then transforms the matrix into its final diagonal form. The basic idea is to iteratively eliminate the elements *above* the main diagonal for each column, one after another. Algorithm 2 describes this procedure.

Note that the inner for loop performs row replacement. However, for each row, only d_{ik} and $d_{i(n+1)}$ need to be updated, since elements on the other columns stay the same.

After obtaining **D**, the desired solution $\vec{x}$ can be computed by

$$x_i = d_{i(n+1)}/d_{ii}, \text{for any } i.$$

Algorithm 1 Gaussian Elimination

```
Input: An augmented matrix $\mathbf{G} = \{\mathbf{A}|\vec{b}\}$, where $\mathbf{A} = (a_{ij})$ is an $n \times n$ matrix
and $\vec{b} = (b_i)$ is an $n$-dimensional vector.
Output: The augmented matrix $\mathbf{U}$ (the elements are denoted as $u_{ij}$) that is
equivalent to $\mathbf{G}$ and is in the "upper triangular" form.

Initially, $\mathbf{U} \leftarrow \mathbf{G}$
for $k = 1$ to $n - 1$ do/*eliminate elements below the diagonal to zero one column after another*/
    /*Pivoting*/
    In $\mathbf{U}$, from row $k$ to row $n$, find the row $k_p$ that has the maximum absolute value of the element in the $k^{th}$ column
    Swap row $k$ and row $k_p$
    /*Elimination*/
    for $i = k + 1$ to $n$ do
        $temp = u_{ik}/u_{kk}$
        for $j = k$ to $n + 1$ do
            $u_{ij} \leftarrow u_{ij} - temp \cdot u_{kj}$/*row replacement*/
        endfor
    endfor
endfor
```

Algorithm 2 Jordan Elimination

```
Input: The output of the Gaussian Elimination $\mathbf{U}$ (an $n \times (n+1)$ matrix).
Output: The augmented matrix $\mathbf{D}$ (with elements denoted by $d_{ij}$) that is
equivalent to $\mathbf{G}$ and is in our final target form.

Initially, $\mathbf{D} \leftarrow \mathbf{U}$
for $k = n$ to 2 do/*eliminate elements to zero for each column one after
another*/
    for $i = 1$ to $k-1$ do/*row replacement one row after another*/
        $d_{i(n+1)} \leftarrow d_{i(n+1)} - d_{ik}/d_{kk} \cdot d_{k(n+1)}$
        $d_{ik} \leftarrow 0$
    endfor
endfor
```

1.3 An Example

We give an example here to demonstrate the described algorithms. Consider a linear system of equations:

$$\begin{aligned} 2x_1 + 4x_2 - 2x_3 &= 3 \\ -4x_1 - 8x_2 + 5x_3 &= -4 \\ 4x_1 + 4x_2 - 5x_3 &= 4. \end{aligned}$$

The corresponding augmented matrix is

$$\left(\begin{array}{ccc|c} 2 & 4 & -2 & 3 \\ -4 & -8 & 5 & -4 \\ 4 & 4 & -5 & 4 \end{array}\right)$$

The Gauss-Jordan Elimination with partial pivoting on it will be

$$\left(\begin{array}{ccc|c} 2 & 4 & -2 & 3 \\ -4 & -8 & 5 & -4 \\ 4 & 4 & -5 & 4 \end{array}\right)$$

$$\xrightarrow{R_1 \leftrightarrow R_2} \left(\begin{array}{ccc|c} -4 & -8 & 5 & -4 \\ 2 & 4 & -2 & 3 \\ 4 & 4 & -5 & 4 \end{array}\right) \text{(pivoting)}$$

$$\xrightarrow{R_2 + \frac{1}{2}R_1} \left(\begin{array}{ccc|c} -4 & -8 & 5 & -4 \\ 0 & 0 & \frac{1}{2} & 1 \\ 4 & 4 & -5 & 4 \end{array}\right)$$

$$\xrightarrow{R_3 + R_1} \left(\begin{array}{ccc|c} -4 & -8 & 5 & -4 \\ 0 & 0 & \frac{1}{2} & 1 \\ 0 & -4 & 0 & 0 \end{array}\right)$$

$$\xrightarrow{R_2 \leftrightarrow R_3} \left(\begin{array}{ccc|c} -4 & -8 & 5 & -4 \\ 0 & -4 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 1 \end{array}\right) \text{(pivoting; it happens to be the end of Gaussian Elimination)}$$

$$\xrightarrow{R_1 - 10R_3} \left(\begin{array}{ccc|c} -4 & -8 & 0 & -14 \\ 0 & -4 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 1 \end{array}\right) \text{(starting Jordan Elimination)}$$

$$\xrightarrow{R_1 - 2R_2} \left(\begin{array}{ccc|c} -4 & 0 & 0 & -14 \\ 0 & -4 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 1 \end{array}\right)$$

2 Task and Requirement

Task: Using OpenMP, implement a parallelized program to solve linear systems of equations through Gauss-Jordan Elimination with partial pivoting. The input will be a coefficient matrix $\mathbf{A}$ and a vector $\vec{b}$. The output will be a vector $\vec{x}$, where

$$\mathbf{A} \cdot \vec{x} = \vec{b}.$$

Requirements and Remarks:

- Some helper scripts are provided in "Development Kit Lab 3". Specifically, compile and run "datagen.c" to generate the input data. For marking purposes, use the functions `Lab3LoadInput` to load your input data and `Lab3SaveOutput` to save your output data.
- Time measurement on the Gauss-Jordan Elimination computation time should be implemented. Note that the computation time should account for overhead caused by the parallelization strategy, but should not include the matrix loading or saving time.
- The number of threads should be the only command line argument passed to your program.
- *Optimize* the performance of your implementation. This entails that both your underlying program code and the OpenMP parallelization strategy must be efficient.
- When implementing your program, you do not need to consider the *singular* cases, i.e., a linear system with no solution or an infinite number of solutions. The input data generated by "datagen.c" will avoid such cases.
- You do need to include the partial pivoting procedure in your code to make the computed results correct and numerically stable.
- Make sure that your program works on the VM.

Submission: One member of each team is required to submit a zip file to eClass before the submission deadline. While we do not enforce a naming convention for the submissions, a good template is "user**_lab3.zip" (where ** is the user number). The zip file should contain the following:

1. "Makefile": By executing the `make` command, the solution executable named "main" should be generated. Please do not use any optimization flags (i.e. no `-O3`).
2. Solution source files: You should include all source files necessary to compile your solution executable. Note that you do not need to include "datagen.c". Also, do not include any input/output data file or compiled executable.

3 Marking Guideline

3.1 Marking Session

Each group is required to present a short demo of their code within an appointed in-lab time-slot. Each demo consists of the following components:

1. **Demo:** Upload and compile your eClass submission and verification code (to be provided during the marking session) in your VM. Demonstrate that your code works for different matrix sizes and different thread counts, as specified by the marker (a TA or LI). This includes verifying the correctness of your results and comparing the Gauss-Jordan Elimination computation time achieved relative to the optimal results prepared by the LI and TAs.
2. **Presentation:** Provide a short (1-minutes) verbal explanation of the program design. Focus on the strategy implemented to minimize the runtime of the Gauss-Jordan Elimination program.
3. **Group Response:** The marker will ask some questions to the group. Marks will be assigned to the group based on their collective response to these questions.
4. **Individual Response:** The marker will ask some questions to each group member. Marks will be assigned to each individual based on their individual response (without assistance from other group members) to these questions.

To expedite the marking process, please rehearse your demo beforehand. Be aware that:

1. You should practice the process of loading, unzipping, and compiling your eClass submission in the VM in an efficient manner.
2. The marking process is timed. While there should be ample time for the demo, the marker may cut you off if the demo runs over the time limit.
3. The questions are sampled from a pool, and may be different for different groups.
4. The contents of the questions asked by the marker may include, but are not limited to:
   (a) explaining observations from the results,
   (b) describing anticipated program performance under certain scenarios,
   (c) providing potential improvements on the existing solution to address specific scenarios,
   (d) describing certain components within the solution code,
   (e) explaining concepts and techniques learned in the lecture that are relevant to this lab.

3.2 Marking Rubric

Successful code compilation and execution (generates correct results):	1
Runtime is competitive and comparable to the optimal results:	1
Group questions:	2
Individual questions:	1
Total:	**5**