Parallel Implementations of Gaussian Elimination

Vasilije Perović

Western Michigan University

vasilije.perovic@wmich.edu

January 27, 2012

Linear systems of equations

General form of a linear system of equations is given by

$$
\begin{array}{cccc}
a_{11}x_1 & + \cdots + & a_{1n}x_n & = b_1 \\
a_{21}x_1 & + \cdots + & a_{2n}x_n & = b_2 \\
\vdots & & \vdots & \\
a_{m1}x_1 & + \cdots + & a_{mn}x_n & = b_m
\end{array}
$$

where a_{ij} 's and b_i 's are known and we are solving for x_i 's.

$A\mathbf{x} = \mathbf{b}$

More compactly, we can rewrite system of linear equations in the form

$$A\mathbf{x} = \mathbf{b}$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Why study solutions of $A\mathbf{x} = \mathbf{b}$

1. One of the most fundamental problems in the field of scientific computing
2. Arises in many applications:

Why study solutions of $A\mathbf{x} = \mathbf{b}$

1. One of the most fundamental problems in the field of scientific computing
2. Arises in many applications:
    - Chemical engineering
    - Interpolation
    - Structural analysis
    - Regression Analysis
    - Numerical ODEs and PDEs

Methods for solving $A\mathbf{x} = \mathbf{b}$

1. *Direct methods*
2. *Iterative methods*

Methods for solving $A\mathbf{x} = \mathbf{b}$

1. *Direct methods* – obtain the exact solution (in real arithmetic) in finitely many operations
2. *Iterative methods* – generate sequence of approximations that converge in the limit to the solution

Methods for solving $A\mathbf{x} = \mathbf{b}$

1. *Direct methods* – obtain the exact solution (in real arithmetic) in finitely many operations
   - Gaussian elimination (LU factorization)
   - QR factorization
   - WZ factorization
2. *Iterative methods* – generate sequence of approximations that converge in the limit to the solution
   - Jacobi iteration
   - Gauss-Seidal iteration
   - SOR method (successive over-relaxation)

Methods for solving $A\mathbf{x} = \mathbf{b}$

1. *Direct methods* – obtain the exact solution (in real arithmetic) in finitely many operations
   - $\boxed{\text{Gaussian elimination } (LU \text{ factorization})}$
   - *QR* factorization
   - *WZ* factorization
2. *Iterative methods* – generate sequence of approximations that converge in the limit to the solution
   - Jacobi iteration
   - Gauss-Seidal iteration
   - SOR method (successive over-relaxation)

Gaussian Elimination

When solving $A\mathbf{x} = \mathbf{b}$ we will assume throughout this presentation that A is non-singular and A and $\mathbf{b}$ are known

$$\begin{array}{cccccc} a_{11}x_1 & + \cdots + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + \cdots + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & \\ a_{n1}x_1 & + \cdots + & a_{nn}x_n & = & b_n & . \end{array}$$

Gaussian Elimination

Assuming that $a_{11} \neq 0$ we first subtract a_{21}/a_{11} times the first equation from the second equation to eliminate the coefficient x_1 in the second equation, and so on until the coefficients of x_1 in the last $n-1$ rows have all been eliminated. This gives the modified system of equations

Gaussian Elimination

Assuming that $a_{11} \neq 0$ we first subtract a_{21}/a_{11} times the first equation from the second equation to eliminate the coefficient x_1 in the second equation, and so on until the coefficients of x_1 in the last $n-1$ rows have all been eliminated. This gives the modified system of equations

$$\begin{array}{lllll} a_{11}x_1 + & a_{12}x_2 & + \cdots + & a_{1n}x_n & = b_1 \\ & a_{22}^{(1)}x_2 & + \cdots + & a_{2n}^{(1)}x_n & = b_2^{(1)} \\ & \vdots & & \vdots & \vdots \\ & a_{n1}^{(1)}x_2 & + \cdots + & a_{nn}^{(1)}x_n & = b_n^{(1)} \end{array},$$

where

$$a_{ij}^{(1)} = a_{ij} - a_{1j}\frac{a_{i1}}{a_{11}} \qquad b_i^{(1)} = b_i - b_1\frac{a_{i1}}{a_{11}} \qquad i,j = 1,2,\ldots,n\,.$$

Gaussian Elimination (*Forward Reduction*)

Applying the same process the last $n-1$ equations of the modified system to eliminate coefficients of x_2 in the last $n-2$ equations, and so on, until the entire system has been reduced to the *(upper) triangular form*

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(n-1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2^{(1)} \\ \vdots \\ b_n^{(n-1)} \end{bmatrix}.$$

Gaussian Elimination (*Forward Reduction*)

Applying the same process the last $n-1$ equations of the modified system to eliminate coefficients of x_2 in the last $n-2$ equations, and so on, until the entire system has been reduced to the *(upper) triangular form*

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(n-1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2^{(1)} \\ \vdots \\ b_n^{(n-1)} \end{bmatrix}.$$

• The superscripts indicate the number of times the elements had to be changed.

Gaussian Elimination – Example

- Perform the *forward reduction* the the system given below

$$\begin{bmatrix} 4 & -9 & 2 \\ 2 & -4 & 4 \\ -1 & 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

Gaussian Elimination – Example

- Perform the *forward reduction* the the system given below

$$\begin{bmatrix} 4 & -9 & 2 \\ 2 & -4 & 4 \\ -1 & 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

We start by writing down the augmented matrix for the given system:

$$\left[\begin{array}{rrr|r} 4 & -9 & 2 & 2 \\ 2 & -4 & 4 & 3 \\ -1 & 2 & 2 & 1 \end{array}\right]$$

Gaussian Elimination – Example

We perform appropriate row operations to obtain:

$$\left[\begin{array}{rrr|r} 4 & -9 & 2 & 2 \\ 2 & -4 & 4 & 3 \\ -1 & 2 & 2 & 1 \end{array}\right] \longrightarrow$$

Gaussian Elimination – Example

We perform appropriate row operations to obtain:

$$\left[\begin{array}{rrr|r} 4 & -9 & 2 & 2 \\ 2 & -4 & 4 & 3 \\ -1 & 2 & 2 & 1 \end{array}\right] \xrightarrow[R_3-\left(\frac{-1}{4}\right)R_1 \to R_3]{R_2-\left(\frac{2}{4}\right)R_1 \to R_2} \left[\begin{array}{lrr|r} 4 & -9 & 2 & 2 \\ 0 & 0.5 & 3 & 2 \\ 0 & -0.25 & 2.5 & 1.5 \end{array}\right]$$

Gaussian Elimination – Example

We perform appropriate row operations to obtain:

$$\left[\begin{array}{rrr|r} 4 & -9 & 2 & 2 \\ 2 & -4 & 4 & 3 \\ -1 & 2 & 2 & 1 \end{array}\right] \xrightarrow[R_3-\left(\frac{-1}{4}\right)R_1 \to R_3]{R_2-\left(\frac{2}{4}\right)R_1 \to R_2} \left[\begin{array}{lrr|r} 4 & -9 & 2 & 2 \\ 0 & 0.5 & 3 & 2 \\ 0 & -0.25 & 2.5 & 1.5 \end{array}\right]$$

$$\xrightarrow{R_3-\left(\frac{-0.25}{0.5}\right)R_2 \to R_3} \left[\begin{array}{lrr|r} 4 & -9 & 2 & 2 \\ 0 & 0.5 & 3 & 2 \\ 0 & 0 & 4 & 2.5 \end{array}\right]$$

Gaussian Elimination – Example

Note that the row operations used to eliminate x_1 from the second and the third equations are equivalent to multiplying *on the left* the augmented matrix:

$$\left[\begin{array}{rrr} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0.25 & 0 & 1 \end{array}\right] \cdot \left[\begin{array}{rrr|r} 4 & -9 & 2 & 2 \\ 2 & -4 & 4 & 3 \\ -1 & 2 & 2 & 1 \end{array}\right] = \left[\begin{array}{rrr|r} 4 & -9 & 2 & 2 \\ 0 & 0.5 & 3 & 2 \\ 0 & -0.25 & 2.5 & 1.5 \end{array}\right]$$

Gaussian Elimination – Example

Note that the row operations used to eliminate x_1 from the second and the third equations are equivalent to multiplying *on the left* the augmented matrix:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0.25 & 0 & 1 \end{bmatrix}}_{L_1} \cdot \left[\begin{array}{rrr|r} 4 & -9 & 2 & 2 \\ 2 & -4 & 4 & 3 \\ -1 & 2 & 2 & 1 \end{array}\right] = \left[\begin{array}{lrr|r} 4 & -9 & 2 & 2 \\ 0 & 0.5 & 3 & 2 \\ 0 & -0.25 & 2.5 & 1.5 \end{array}\right]$$

Gaussian Elimination – Example

Note that the row operations used to eliminate x_1 from the second and the third equations are equivalent to multiplying *on the left* the augmented matrix:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0.5 & 1 \end{bmatrix}}_{L_2} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0.25 & 0 & 1 \end{bmatrix}}_{L_1} \cdot [A|\mathbf{b}] = \left[\begin{array}{ccc|c} 4 & -9 & 2 & 2 \\ 0 & 0.5 & 3 & 2 \\ 0 & 0 & 4 & 2.5 \end{array}\right]$$

Gaussian Elimination – Example

$$\underbrace{\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0.5 & 1 \end{bmatrix}}_{L_2} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0.25 & 0 & 1 \end{bmatrix}}_{L_1}}_{\widehat{L} = L_2 \cdot L_1} \cdot [A|\mathbf{b}] = \left[\begin{array}{ccc|c} 4 & -9 & 2 & 2 \\ 0 & 0.5 & 3 & 2 \\ 0 & 0 & 4 & 2.5 \end{array}\right]$$

Thus, we can write

$$(L_2 \cdot L_1) \cdot A = \widehat{L} \cdot A = U$$

LU factorization

In general, we can think of row operations as multiplying with matrices on the left, thus the i^{th} elimination step is equivalent as multiplication on the left by

$$L_i = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{i+1,i} & & \\ & & \vdots & \ddots & \\ & & -l_{n,i} & & 1 \end{bmatrix}.$$

LU factorization

Continuing in this fashion we obtain

$$\widehat{L}A\mathbf{x} = \widehat{L}\mathbf{b}\,, \qquad \widehat{L} = L_{n-1}\cdots L_2 L_1\,,$$

LU factorization

Continuing in this fashion we obtain

$$\widehat{L}A\mathbf{x} = \widehat{L}\mathbf{b}\,, \qquad \widehat{L} = L_{n-1}\cdots L_2 L_1\,,$$

$$A = \widehat{L}^{-1}U = LU\,.$$

LU factorization

Continuing in this fashion we obtain

$$A = \widehat{L}^{-1} U = LU\,.$$

Thus, the Gaussian elimination algorithm for solving $A\mathbf{x} = \mathbf{b}$ is mathematically equivalent to the three-step process:

1. Factor $A = LU$
2. Solve (forward substitution) $L\mathbf{y} = \mathbf{b}$
3. Solve (back substitution) $U\mathbf{x} = \mathbf{y}\,.$

LU factorization

Continuing in this fashion we obtain

$$A = \widehat{L}^{-1} U = LU\,.$$

Thus, the Gaussian elimination algorithm for solving $A\mathbf{x} = \mathbf{b}$ is mathematically equivalent to the three-step process:

1.	Factor	$A = LU$
2.	Solve (forward substitution)	$L\mathbf{y} = \mathbf{b}$
3.	Solve (back substitution)	$U\mathbf{x} = \mathbf{y}\,.$

- Order of operations:
  - $\frac{n^3}{3} + n^2 - \frac{n}{3}$ multiplications/divisions
  - $\frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6}$.

Need for partial pivoting

Apply LU factorization *without pivoting* to

$$A = \begin{bmatrix} 0.0001 & 1 \\ 1 & 1 \end{bmatrix}$$

in *three-decimal-digit* floating point arithmetic.

Need for partial pivoting

$$A = \begin{bmatrix} 0.0001 & 1 \\ 1 & 1 \end{bmatrix}$$

Solution: L and U are easily obtainable and are given by:

$$\begin{aligned}
L &= \begin{bmatrix} 1 & 0 \\ fl(1/10^{-4}) & 1 \end{bmatrix}, \; fl(1/10^{-4}) \text{ rounds to } 10^4, \\
U &= \begin{bmatrix} 10^{-4} & 1 \\ 0 & fl(1-10^4\cdot 1) \end{bmatrix}, \; fl(1-10^4\cdot 1) \text{ rounds to } -10^4 \\
\text{so} \quad LU &= \begin{bmatrix} 1 & 0 \\ 10^4 & 1 \end{bmatrix} \begin{bmatrix} 10^{-4} & 1 \\ 0 & -10^4 \end{bmatrix} = \begin{bmatrix} 10^{-4} & 1 \\ 1 & 0 \end{bmatrix} \\
\text{but} \quad A &= \begin{bmatrix} 10^{-4} & 1 \\ 1 & 1 \end{bmatrix}.
\end{aligned}$$

Need for partial pivoting

$$A = \begin{bmatrix} 10^{-4} & 1 \\ 1 & 1 \end{bmatrix} \qquad \text{but} \qquad LU = \begin{bmatrix} 10^{-4} & 1 \\ 1 & 0 \end{bmatrix}$$

Remark 1: Note that original a_{22} has been entirely "lost" from the computation by subtracting 10^4 from it. Thus if we were to use this LU factorization in order to solve a system there would be no way to guarantee an accurate answer. This is called *numerical instability*.

Need for partial pivoting

$$A = \begin{bmatrix} 10^{-4} & 1 \\ 1 & 1 \end{bmatrix} \qquad \text{but} \qquad LU = \begin{bmatrix} 10^{-4} & 1 \\ 1 & 0 \end{bmatrix}$$

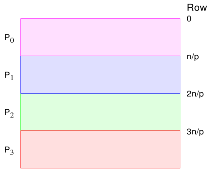
Remark 1: Note that original a_{22} has been entirely "lost" from the computation by subtracting 10^4 from it. Thus if we were to use this LU factorization in order to solve a system there would be no way to guarantee an accurate answer. This is called *numerical instability*.

Remark 2: Suppose we attempted to solve system $A\mathbf{x} = [1, 2]^T$ for $\mathbf{x}$ using this LU decomposition. The correct answer is $x \approx [1, 1]^T$. Instead, if we were to stick with the three-digit-floating point arithmetic we would get an answer $\widehat{\mathbf{x}} = [0, 1]^T$ which is completely erroneous.

Sequential Algorithm

```
for (k = 0; k < n-1; k++) { /* Forward elimination */
   r = max_col(a,k);
   if (k != r) exchange_row(a,b,r,k);
   for (i=k+1; i < n; i++) {
      l[i] = a[i][k]/a[k][k];
      for (j=k+1; j < n; j++)
         a[i][j] = a[i][j] - l[i] * a[k][j];
      b[i] = b[i] - l[i] * b[k];
   }
}
for (k = n-1; k >= 0; k--) { /* Backward substitution */
   sum = 0.0;
   for (j=k+1; j < n; j++)
      sum = sum + a[k][j] * x[j];
   x[k] = 1/a[k][k] * (b[k] - sum);
}
return x;
```

Row-Oriented Algorithm



1. **Determination of the local pivot element,**
2. **Determination of the global pivot element,**
3. **Exchange of the pivot row,**
4. **Distribution of the pivot row,**
5. **Computation of the elimination factors,**
6. **Computation of the matrix elements.**

Row-Oriented Algorithm

Remark 1: The computation of the solution vector **x** in the backward substitution is inherently serial, since the values of x_k depend on each other and are computed one after another. Thus, in step k the processor P_q owning row k computers the value of x_k and sends the value to all other processors by a *single* broadcast operation.

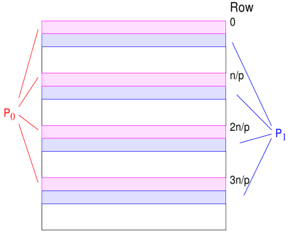
Row-Oriented Algorithm

Remark 1: The computation of the solution vector **x** in the backward substitution is inherently serial, since the values of x_k depend on each other and are computed one after another. Thus, in step k the processor P_q owning row k computers the value of x_k and sends the value to all other processors by a *single* broadcast operation.

Remark 2: Note that the data distribution is not quite adequate. For example, suppose we are at the i^{th} step and that $i > m \cdot n/p$ where m is a natural number. In that case, processors $p_0, p_1, \ldots, p_{m-1}$ are idle since all their work is done until the back substitution step at the very end. Hence there is an issue with load balancing.

Row-Oriented Algorithm

Remark 2: Note that the data distribution is not quite adequate. For example, suppose we are at the i^{th} step and that $i > m \cdot n/p$ where m is a natural number. In that case, processors $p_0, p_1, \ldots, p_{m-1}$ are idle since all their work is done until the back substitution step at the very end. Hence there is an issue with load balancing.



block-cyclic row distribution

Row-Oriented Algorithm

Remark 3: We could also consider *column-oriented* data distribution. In that case, at the k^{th} step the processor that owns k^{th} column would have all needed data to compute the new pivot. On the one hand, this would reduce communication between processors that we had when considering row-oriented data distribution. On the other hand, with column orientation pivot determination is all done serially. Thus, in case when $n >> p$ (which is often case) row oriented data distribution might be more advantageous since the pivot determination is done in parallel

Pipelined Implementation

1. Challenges and possible approaches
    - *Send ahead* strategy

Pipelined Implementation

1. Challenges and possible approaches
    - *Send ahead* strategy
    - Challenges with pivoting

Pipelined Implementation

1. Challenges and possible approaches
   - *Send ahead* strategy
   - Challenges with pivoting
   - *Column pivoting*

Pipelined Implementation

1. Challenges and possible approaches
   - *Send ahead* strategy
   - Challenges with pivoting
   - *Column pivoting*
2. Advantages
   - It allows *asynchronous execution:* "processes can reduce their portions of the augmented matrix as soon as the pivot rows are available"

Pipelined Implementation

1. Challenges and possible approaches
   - *Send ahead* strategy
   - Challenges with pivoting
   - *Column pivoting*
2. Advantages
   - It allows *asynchronous execution:* "processes can reduce their portions of the augmented matrix as soon as the pivot rows are available"
   - It allows processes to overlap communication and computation times.

Final Remarks

- Compute *communication* and *computation* times.

Final Remarks

- Compute *communication* and *computation* times.
  - Largely depends on what kind of distributed system used (usual configurations that have been studying are 1D and 2D meshes as well as hypercube).

Final Remarks

- Compute *communication* and *computation* times.
  - Largely depends on what kind of distributed system used (usual configurations that have been studying are 1D and 2D meshes as well as hypercube).
- Compute *isoefficiency functions* and determine scalability.

Final Remarks

- Compute *communication* and *computation* times.
  - Largely depends on what kind of distributed system used (usual configurations that have been studying are 1D and 2D meshes as well as hypercube).
- Compute *isoefficiency functions* and determine scalability.
  - Row (column) oriented algorithms, stable but not scalable.
  - Pipelined implementation is perfectly scalable.
- Structured systems
  - Symmetric/Hermitian.
  - Banded and upper (lower) triangular.
  - Positive definite (CG method).

Final Remarks

- Compute *communication* and *computation* times.
  - Largely depends on what kind of distributed system used (usual configurations that have been studying are 1D and 2D meshes as well as hypercube).
- Compute *isoefficiency functions* and determine scalability.
  - Row (column) oriented algorithms, stable but not scalable.
  - Pipelined implementation is perfectly scalable.
- Structured systems
  - Symmetric/Hermitian.
  - Banded and upper (lower) triangular.
  - Positive definite (CG method).
- *Sparse* systems and *iterative methods* (numerical PDEs)