Chapter 12

Solving Linear Systems
Outline

- Terminology
- Back substitution
- Gaussian elimination
- Jacobi method
- Conjugate gradient method
Terminology

• System of linear equations
  – Solve $Ax = b$ for $x$

• Special matrices
  – Symmetrically banded
  – Upper triangular
  – Lower triangular
  – Diagonally dominant
  – Symmetric
Symmetrically Banded

\[
\begin{bmatrix}
4 & 2 & -1 & 0 & 0 & 0 \\
3 & -4 & 5 & 6 & 0 & 0 \\
1 & 6 & 3 & 2 & 4 & 0 \\
0 & 2 & -2 & 0 & 9 & 2 \\
0 & 0 & 7 & 3 & 8 & 7 \\
0 & 0 & 0 & 4 & 0 & 2 \\
\end{bmatrix}
\]

Semibandwidth 2
### Upper Triangular

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>-1</td>
<td>5</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>-4</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>-4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
**Lower Triangular**

\[
\begin{array}{ccccccc}
4 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 4 & 3 & 0 & 0 & 0 & 0 \\
2 & 6 & 2 & 3 & 0 & 0 & 0 \\
8 & -2 & 0 & 1 & 8 & 0 & 0 \\
-3 & 5 & 7 & 9 & 5 & 2 & \\
\end{array}
\]
Diagonally Dominant

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>-15</td>
<td>2</td>
<td>0</td>
<td>-3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>22</td>
<td>-1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>13</td>
<td>0</td>
<td>-5</td>
</tr>
<tr>
<td>5</td>
<td>-2</td>
<td>0</td>
<td>1</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>-3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>-32</td>
</tr>
</tbody>
</table>
Symmetric

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>-3</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-1</td>
<td>9</td>
<td>0</td>
<td>-5</td>
</tr>
<tr>
<td>0</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
<td>-5</td>
<td>5</td>
<td>-3</td>
</tr>
</tbody>
</table>
Back Substitution

- Used to solve upper triangular system $Tx = b$ for $x$
- Methodology: one element of $x$ can be immediately computed
- Use this value to simplify system, revealing another element that can be immediately computed
- Repeat
Back Substitution

\[ \begin{align*}
1x_0 + 1x_1 - 1x_2 + 4x_3 &= 8 \\
-2x_1 - 3x_2 + 1x_3 &= 5 \\
2x_2 - 3x_3 &= 0 \\
2x_3 &= 4
\end{align*} \]
Back Substitution

\[ \begin{align*}
1x_0 + 1x_1 - 1x_2 + 4x_3 &= 8 \\
-2x_1 - 3x_2 + 1x_3 &= 5 \\
2x_2 - 3x_3 &= 0 \\
x_3 &= 2 \\
2x_3 &= 4
\end{align*} \]
Back Substitution

\begin{align*}
1x_0 + 1x_1 - 1x_2 &= 0 \\
-2x_1 - 3x_2 &= 3 \\
2x_2 &= 6 \\
2x_3 &= 4
\end{align*}
Back Substitution

\[ \begin{align*}
1x_0 &+ 1x_1 - 1x_2 &= 0 \\
-2x_1 &- 3x_2 &= 3 \\
2x_2 &= 6
\end{align*} \]

\[ x_2 = 3 \quad 2x_3 = 4 \]
Back Substitution

\[\begin{align*}
x_0 + x_1 &= 3 \\
-2x_1 &= 12 \\
2x_2 &= 6 \\
2x_3 &= 4
\end{align*}\]
Back Substitution

\[ \begin{align*}
1x_0 + 1x_1 &= 3 \\
-2x_1 &= 12 \\
2x_2 &= 6 \\
x_1 &= -6 \\
2x_3 &= 4
\end{align*} \]
Back Substitution

\[ 1x_0 = 9 \]
\[ -2x_1 = 12 \]
\[ 2x_2 = 6 \]
\[ 2x_3 = 4 \]
Back Substitution

\[ 1x_0 = 9 \]

\[ -2x_1 = 12 \]

\[ 2x_2 = 6 \]

\[ x_0 = 9 \quad 2x_3 = 4 \]
Pseudocode

for $i \leftarrow n - 1$ down to 1 do
  $x[i] \leftarrow b[i] / a[i,i]$
  for $j \leftarrow 0$ to $i - 1$ do
    $b[j] \leftarrow b[j] - x[i] \times a[j,i]$
  endfor
endfor

Time complexity: $\Theta(n^2)$
Data Dependence Diagram

We cannot execute the outer loop in parallel.
We can execute the inner loop in parallel.
Row-oriented Algorithm

- Associate primitive task with each row of $A$ and corresponding elements of $x$ and $b$
- During iteration $i$ task associated with row $j$ computes new value of $b_j$
- Task $i$ must compute $x_i$ and broadcast its value
- Agglomerate using rowwise interleaved striped decomposition
Interleaved Decompositions

Rowwise interleaved striped decomposition

Columnwise interleaved striped decomposition
Complexity Analysis

- Each process performs about $n / (2\rho)$ iterations of loop $j$ in all
- A total of $n - 1$ iterations in all
- Computational complexity: $\Theta(n^2/\rho)
- One broadcast per iteration
- Communication complexity: $\Theta(n \log \rho)$
Column-oriented Algorithm

- Associate one primitive task per column of $A$ and associated element of $x$
- Last task starts with vector $b$
- During iteration $i$ task $i$ computes $x_i$, updates $b$, and sends $b$ to task $i-1$
- In other words, no computational concurrency
- Agglomerate tasks in interleaved fashion
Complexity Analysis

• Since $b$ always updated by a single process, computational complexity same as sequential algorithm: $\Theta(n^2)$

• Since elements of $b$ passed from one process to another each iteration, communication complexity is $\Theta(n^2)$
Comparison

Message-passing time dominates

<table>
<thead>
<tr>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column-oriented algorithm superior</td>
</tr>
</tbody>
</table>

2

Row-oriented algorithm superior

n

Computation time dominates
Gaussian Elimination

- Used to solve $Ax = b$ when $A$ is dense
- Reduces $Ax = b$ to upper triangular system $Tx = c$
- Back substitution can then solve $Tx = c$ for $x$
Gaussian Elimination

\[ 4x_0 + 6x_1 + 2x_2 - 2x_3 = 8 \]
\[ 2x_0 + 5x_2 - 2x_3 = 4 \]
\[ -4x_0 - 3x_1 - 5x_2 + 4x_3 = 1 \]
\[ 8x_0 + 18x_1 - 2x_2 + 3x_3 = 40 \]
Gaussian Elimination

\[
\begin{align*}
4x_0 + 6x_1 + 2x_2 - 2x_3 &= 8 \\
-3x_1 + 4x_2 - 1x_3 &= 0 \\
+3x_1 - 3x_2 + 2x_3 &= 9 \\
+6x_1 - 6x_2 + 7x_3 &= 24
\end{align*}
\]
Gaussian Elimination

\[\begin{align*}
4x_0 + 6x_1 + 2x_2 - 2x_3 &= 8 \\
-3x_1 + 4x_2 - x_3 &= 0 \\
1x_2 + 1x_3 &= 9 \\
2x_2 + 5x_3 &= 24
\end{align*}\]
Gaussian Elimination

\[\begin{align*}
4x_0 + 6x_1 + 2x_2 - 2x_3 &= 8 \\
-3x_1 + 4x_2 - x_3 &= 0 \\
1x_2 + 1x_3 &= 9 \\
3x_3 &= 6
\end{align*}\]
Iteration of Gaussian Elimination

Elements that will not be changed

Pivot row

Elements already driven to 0

Elements that will be changed
Numerical Stability Issues

- If pivot element close to zero, significant roundoff errors can result
- Gaussian elimination with partial pivoting eliminates this problem
- In step \( i \), we search rows \( i \) through \( n-1 \) for the row whose column \( i \) element has the largest absolute value
- Swap (pivot) this row with row \( i \)
Implementing Partial Pivoting

Without partial pivoting

With partial pivoting
Row–oriented Parallel Algorithm

- Associate primitive task with each row of $A$ and corresponding elements of $x$ and $b$
- A kind of reduction needed to find the identity of the pivot row
- Tournament: want to determine identity of row with largest value, rather than largest value itself
- Could be done with two all–reductions
- MPI provides a simpler, faster mechanism
MPI_MAXLOC, MPI_MINLOC

• MPI provides reduction operators MPI_MAXLOC, MPI_MINLOC

• Provide datatype representing a (value, index) pair
## MPI (value,index) Datatypes

<table>
<thead>
<tr>
<th>MPI_Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_2INT</td>
<td>Two ints</td>
</tr>
<tr>
<td>MPI_DOUBLE_INT</td>
<td>A double followed by an int</td>
</tr>
<tr>
<td>MPI_FLOAT_INT</td>
<td>A float followed by an int</td>
</tr>
<tr>
<td>MPI_LONG_INT</td>
<td>A long followed by an int</td>
</tr>
<tr>
<td>MPI_LONG_DOUBLE_INT</td>
<td>A long double followed by an int</td>
</tr>
<tr>
<td>MPI_SHORT_INT</td>
<td>A short followed by an int</td>
</tr>
</tbody>
</table>
Example Use of MPI_MAXLOC

```c
struct {
    double value;
    int    index;
} local, global;
...
local.value = fabs(a[j][i]);
local.index = j;
...
MPI_Allreduce (&local, &global, 1,
               MPI_DOUBLE_INT, MPI_MAXLOC,
               MPI_COMM_WORLD);
```
Second Communication per Iteration

\[ a[j][i] \]

\[ a[j][k] \]

\[ a[picked][i] \]

\[ a[picked][k] \]
Communication Complexity

- Complexity of tournament: $\Theta(\log p)$
- Complexity of broadcasting pivot row: $\Theta(n \log p)$
- A total of $n - 1$ iterations
- Overall communication complexity: $\Theta(n^2 \log p)$
Isoefficiency Analysis

• Communication overhead: $\Theta(n^2 \rho \log \rho)$
• Sequential algorithm has time complexity $\Theta(n^3)$
• Isoefficiency relation
  \[ n^3 \geq C n^2 \rho \log \rho \Rightarrow n \geq C \rho \log \rho \]

\[ M(C \rho \log \rho) / p = C^2 \rho^2 \log^2 \rho / p = C^2 \rho \log^2 \rho \]
• This system has poor scalability
Column-oriented Algorithm

• Associate a primitive task with each column of $A$ and another primitive task for $b$
• During iteration $i$, task controlling column $i$ determines pivot row and broadcasts its identity
• During iteration $i$, task controlling column $i$ must also broadcast column $i$ to other tasks
• Agglomerate tasks in an interleaved fashion to balance workloads
• Isoefficiency same as row-oriented algorithm
Comparison of Two Algorithms

• Both algorithms evenly divide workload
• Both algorithms do a broadcast each iteration
• Difference: identification of pivot row
  – Row-oriented algorithm does search in parallel but requires all-reduce step
  – Column-oriented algorithm does search sequentially but requires no communication
• Row-oriented superior when $n$ relatively larger and $p$ relatively smaller
Problems with These Algorithms

- They break parallel execution into computation and communication phases
- Processes not performing computations during the broadcast steps
- Time spent doing broadcasts is large enough to ensure poor scalability
Pipelined, Row–Oriented Algorithm

- Want to overlap communication time with computation time
- We could do this if we knew in advance the row used to reduce all the other rows.
- Let’s pivot columns instead of rows!
- In iteration $i$, we can use row $i$ to reduce the other rows.
Communication Pattern

Reducing Using Row 0

Row 0

1

2

3
Communication Pattern

Reducing Using Row 0

Reducing Using Row 0
Communication Pattern

Reducing Using Row 0

Reducing Using Row 0

Reducing Using Row 0

Row 0

Row 0
Communication Pattern

Reducing Using Row 0

Reducing Using Row 0

Reducing Using Row 0

Reducing Using Row 0
Communication Pattern

Reducing Using Row 0

Reducing Using Row 0

Reducing Using Row 0
Communication Pattern

Reducing Using Row 0

Reducing Using Row 1

Reducing Using Row 0
Communication Pattern

Reducing Using Row 0

Reducing Using Row 1

Reducing Using Row 1

Reducing Using Row 1
Communication Pattern

Reducing Using Row 1

Row 1
Communication Pattern

0 Reducing Using Row 1

3 Reducing Using Row 1

1 Reducing Using Row 1

2 Reducing Using Row 1
Analysis (1/2)

- Total computation time: $\Theta(n^3/p)$
- Total message transmission time: $\Theta(n^2)$
- When $n$ large enough, message transmission time completely overlapped by computation time
- Message start-up not overlapped: $\Theta(n)$
- Parallel overhead: $\Theta(np)$
Analysis (2/2)

- Isoefficiency relation:

\[ n^3 \geq Cnp \implies n \geq \sqrt[3]{Cp} \]

- Scalability function:

\[ M(\sqrt{Cp}) / p = Cp / p = C \]

- Parallel system is perfectly scalable
Sparse Systems

• Gaussian elimination not well-suited for sparse systems
• Coefficient matrix gradually fills with nonzero elements
• Result
  – Increases storage requirements
  – Increases total operation count
Example of “Fill”
Iterative Methods

• Iterative method: algorithm that generates a series of approximations to solution’s value

• Require less storage than direct methods

• Since they avoid computations on zero elements, they can save a lot of computations
Jacobi Method

\[ x_{i}^{k+1} = \frac{1}{a_{i,i}} \left( b_{i} - \sum_{j \neq i} a_{i,j} x_{j}^{k} \right) \]

Values of elements of vector \( x \) at iteration \( k+1 \) depend upon values of vector \( x \) at iteration \( k \)

Gauss-Seidel method: Use latest version available of \( x_{i} \)
Jacobi Method Iterations
Rate of Convergence

• Even when Jacobi method and Gauss–Seidel methods converge on solution, rate of convergence often too slow to make them practical

• We will move on to an iterative method with much faster convergence
Conjugate Gradient Method

• $A$ is positive definite if for every nonzero vector $x$ and its transpose $x^T$, the product $x^T A x > 0$

• If $A$ is symmetric and positive definite, then the function

$$ q(x) = \frac{1}{2} x^T A x - x^T b + c $$

has a unique minimizer that is solution to $Ax = b$

• Conjugate gradient is an iterative method that solves $Ax = b$ by minimizing $q(x)$
Conjugate Gradient

Convergence

Finds value of $n$-dimensional solution in at most $n$ iterations
Conjugate Gradient Computations

- Matrix–vector multiplication
- Inner product (dot product)
- Matrix–vector multiplication has higher time complexity
- Must modify previously developed algorithm to account for sparse matrices
Rowwise Block Striped Decomposition of a Symmetrically Banded Matrix
Representation of Vectors

• Replicate vectors
  – Need all-gather step after matrix-vector multiply
  – Inner product has time complexity $\Theta(n)$

• Block decomposition of vectors
  – Need all-gather step before matrix-vector multiply
  – Inner product has time complexity $\Theta(n/p + \log p)$
Comparison of Vector Decompositions

- Replicated Vectors Superior
- Block Decomposition Superior
Summary (1/2)

• Solving systems of linear equations
  – Direct methods
  – Iterative methods

• Parallel designs for
  – Back substitution
  – Gaussian elimination
  – Conjugate gradient method
Summary (2/2)

- Superiority of one algorithm over another depends on size of problem, number of processors, characteristics of parallel computer.
- Overlapping communications with computations can be key to scalability.