Parallel Scientific Computing

- Matrix-vector multiplication.
- Matrix-matrix multiplication.
- Direct method for solving a linear equation. Gaussian Elimination.
- Iterative method for solving a linear equation. Jacobi, Gauss-Seidel.
- Sparse linear systems and differential equations.

Iterative Methods for Solving Ax = b

Ex:

$$
\begin{array}{lrcl}
(1) & 6x_1 - 2x_2 + x_3 & = & 11 \\
(2) & -2x_1 + 7x_2 + 2x_3 & = & 5 \\
(3) & x_1 + 2x_2 - 5x_3 & = & \text{-1}
\end{array}
$$

$\Longrightarrow$

$$
\begin{array}{lcl}
x_1 & = & \frac{11}{6} - \frac{1}{6}(-2x_2 + x_3) \\
x_2 & = & \frac{5}{7} - \frac{1}{7}(-2x_1 + 2x_3) \\
x_3 & = & \frac{1}{5} - \frac{1}{-5}(x_1 + 2x_2)
\end{array}
$$

$\Longrightarrow$

$$
\begin{array}{lcl}
x_1^{(k+1)} & = & \frac{1}{6}(11 - (-2x_2^{(k)} + x_3^{(k)})) \\
x_2^{(k+1)} & = & \frac{1}{7}(5 - (-2x_1^{(k)} + 2x_3^{(k)})) \\
x_3^{(k+1)} & = & \frac{1}{-5}(-1 - (x_1^{(k)} + 2x_2^{(k)}))
\end{array}
$$

Initial Approximation: $x_1 = 0, x_2 = 0, x_3 = 0$

Iter	0	1	2	3	4	$\cdots$	8
x_1	0	1.833	2.038	2.085	2.004	$\cdots$	2.000
x_2	0	0.714	1.181	1.053	1.001	$\cdots$	1.000
x_3	0	0.2	0.852	1.080	1.038	$\cdots$	1.000

Stop when $\| \vec{x}^{(k+1)} - \vec{x}^{(k)} \| < 10^{-4}$

Need to define **norm** $\| \vec{x}^{(k+1)} - \vec{x}^{(k)} \|$.

Iterative methods in a matrix format

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}^{k+1} = \begin{bmatrix} 0 & \frac{2}{6} & -\frac{1}{6} \\ \frac{2}{7} & 0 & -\frac{2}{7} \\ \frac{1}{5} & \frac{2}{5} & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}^{k} + \begin{pmatrix} \frac{11}{6} \\ \frac{5}{7} \\ \frac{1}{5} \end{pmatrix}$$

General iterative method:

Assign an initial value to $\vec{x}^{(0)}$

k=0

Do

$$\vec{x}^{(k+1)} = H * \vec{x}^{(k)} + d$$

until $\| \vec{x}^{(k+1)} - \vec{x}^{(k)} \| < \varepsilon$

Norm of a Vector

Given $x = (x_1, x_2, \cdots x_n)$:

$$\| x \|_1 = \sum_{i=1}^{n} | x_i |$$

$$\| x \|_2 = \sqrt{\sum | x_i |^2}$$

$$\| x \|_\infty = max | x_i |$$

Example:

$$x = (-1, 1, 2)$$

$$\| x \|_1 = 4$$

$$\| x \|_2 = \sqrt{1 + 1 + 2^2} = \sqrt{6}$$

$$\| x \|_\infty = 2$$

Applications:

$$\| Error \| \leq \varepsilon$$

Jacobi Method for Ax = b

$$x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij} x_j^k) \quad i = 1, \cdots n$$

Example:

$$\begin{array}{lrcl}
(1) & 6x_1 - 2x_2 + x_3 & = & 11 \\
(2) & -2x_1 + 7x_2 + 2x_3 & = & 5 \\
(3) & x_1 + 2x_2 - 5x_3 & = & \text{-1}
\end{array}$$

$\Longrightarrow$

$$\begin{array}{lcl}
x_1 & = & \frac{11}{6} - \frac{1}{6}(-2x_2 + x_3) \\
x_2 & = & \frac{5}{7} - \frac{1}{7}(-2x_1 + 2x_3) \\
x_3 & = & \frac{1}{5} - \frac{1}{-5}(x_1 + 2x_2)
\end{array}$$

Jacobi method in a matrix-vector form

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}^{k+1} = \begin{bmatrix} 0 & \frac{2}{6} & -\frac{1}{6} \\ \frac{2}{7} & 0 & -\frac{2}{7} \\ \frac{1}{5} & \frac{2}{5} & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}^{k} + \begin{pmatrix} \frac{11}{6} \\ \frac{5}{7} \\ \frac{1}{5} \end{pmatrix}$$

Parallel Jacobi Method

$$x^{k+1} = D^{-1}Bx^k + D^{-1}b$$

or in general

$$x^{k+1} = Hx^k + d.$$

Parallel solution:

- Distribute rows of H to processors.
- Perform computation based on owner-computes rule.
- Perform all-gather collective communication after each iteration.

Task graph/schedule for $y = H * x + d$

Partitioned code:

for $i = 1$ **to** n **do**

S_i : $y_i = 0;$

for $j = 1$ to n **do**

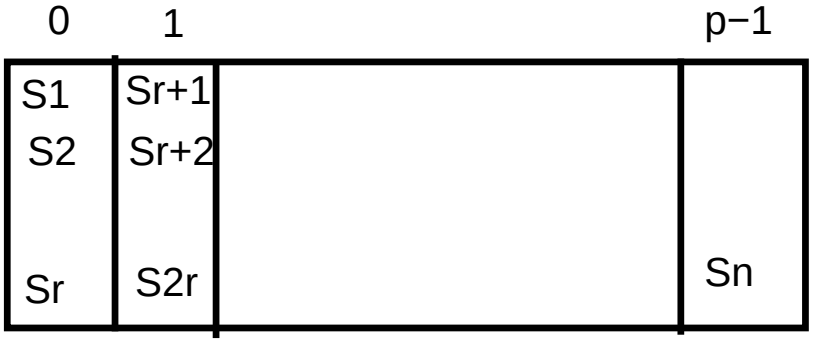
$y_i = y_i + H_{i,j} * x_j + d_j;$

endfor

endfor

Task graph:

Schedule:



Data Mapping for $H * x + d$

Block Mapping function of tasks S_i:

$proc\_map(i) = \lfloor \frac{i-1}{r} \rfloor$ where $r = \lceil \frac{n}{p} \rceil$.

Data partitioning:

Matrix H is divided into n rows $H_1, H_2, \cdots H_n$.

Data mapping:

Row H_i is mapped to processor $proc\_map(i)$.
Vectors y and d are distributed in the same way.
Vector x is replicated to all processors.

Local indexing function is:
$local(i) = (i-1) \bmod r$.

SPMD Code for $y = H * x + d$

me=mynode();

for $i = 1$ **to** n **do**

if $proc\_map(i) == me$, then do S_i:

S_i : $y[i] = 0;$

$i1 = Local(i)$

for $j = 1$ to n **do**

$y[i1] = y[i1] + a[i1][j] * x[j] + d[i1]$

endfor

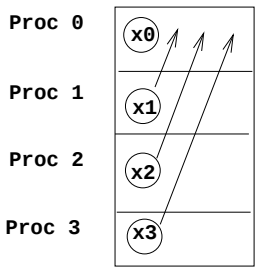
endfor

Cost of Jacobi Method

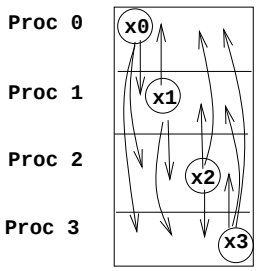
Space cost per processor is $O(n^2/p)$

Time cost:

- T is the number of iterations.
- The all-gather time cost per iteration is C.



(a) Gather



(b) All Gather

- Main cost of S_i: $2n$ multiplication and addition ($2n\omega$).
- Overhead of loop control and code mapping can be reduced, and becomes less significant.

The parallel time is

$$PT \approx T(\frac{n}{p} \times 2n\omega + C) = O(T(n^2/p + C)).$$

Optimizing SPMD Code: Example

SPMP code with block mapping:

```
me=mynode();
For i =0 to rp-1
    if ( proc_map(i) == me) a[Local(i)] = 3.
```

Remove extra loop and branch overhead.

```
me=mynode();
For i = r*me to r*me+r-1
    a[Local(i)] = 3.
```

Further simplification on distributed memory machines

```
me=mynode();
For s = 0 to r-1
    a[s] = 3.
```

If the iterative matrix H is sparse

If it contains a lot of zeros, the code design should take advantage of this:

- Not store too many known zeros.
- Code should explicitly skip those operations applied to zero elements.

Example: $y_0 = y_{n+1} = 0$.

$$y_0 - 2y_1 + y_2 = h^2$$

$$y_1 - 2y_2 + y_3 = h^2$$

$$\vdots$$

$$y_{n-1} - 2y_n + y_{n+1} = h^2$$

This set of equations can be rewritten as:

$$\begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & & \ddots & 1 & \\ & & & 1 & -2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} h^2 \\ h^2 \\ \vdots \\ h^2 \\ h^2 \end{pmatrix}$$

The Jacobi method in a matrix format (right side):

$$0.5* \begin{pmatrix} 0 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & & \ddots & 1 & \\ & & & 1 & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix}^k -0.5* \begin{pmatrix} h^2 \\ h^2 \\ \vdots \\ h^2 \\ h^2 \end{pmatrix}$$

Too time and space consuming if you multiply using the entire iterative matrix!

Optimized solution: write the Jacobi method as:

Repeat

  For i= 1 to n

    $y_i^{new} = 0.5(y_{i-1}^{old} + y_{i+1}^{old} - h^2)$

  Endfor

Until $\| \vec{y}^{new} - \vec{y}^{old} \| < \varepsilon$

Gauss-Seidel Method

Utilize new solutions as soon as they are available.

$$
\begin{aligned}
(1) \quad & 6x_1 - 2x_2 + x_3 = 11 \\
(2) \quad & -2x_1 + 7x_2 + 2x_3 = 5 \\
(3) \quad & x_1 + 2x_2 - 5x_3 = \text{-1}
\end{aligned}
$$

$\Longrightarrow$ Jacobi method.

$$
\begin{aligned}
x_1^{k+1} &= \tfrac{1}{6}(11 - (-2x_2^k + x_3^k)) \\
x_2^{k+1} &= \tfrac{1}{7}(5 - (-2x_1^k + 2x_3^k)) \\
x_3^{k+1} &= \tfrac{1}{-5}(-1 - (x_1^k + 2x_2^k))
\end{aligned}
$$

$\Longrightarrow$ Gauss-Seidel method.

$$
\begin{aligned}
x_1^{k+1} &= \tfrac{1}{6}(11 - (-2x_2^k + x_3^k)) \\
x_2^{k+1} &= \tfrac{1}{7}(5 - (-2x_1^{k+1} + 2x_3^k)) \\
x_3^{k+1} &= \tfrac{1}{-5}(-1 - (x_1^{k+1} + 2x_2^{k+1}))
\end{aligned}
$$

$$\varepsilon = 10^{-4}$$

	0	1	2	3	4	5
x_1	0	1.833	2.069	1.998	1.999	2.000
x_2	0	1.238	1.002	0.995	1.000	1.000
x_3	0	1.062	1.015	0.998	1.000	1.000

It converges faster than Jacobi's method.

The SOR method

SOR (Successive Over Relaxation).

The rate of convergence can be improved (accelerated) by the SOR method:

Step 1: Use the Gauss-Seidel Method.

$$x^{k+1} = Hx^k + d$$

Step 2:

$$x^{k+1} = x^k + w(x^{k+1} - x^k)$$

Convergence of iterative methods

Notation:

$x^* \leftrightarrow$ exact solution

$x^k \leftrightarrow$ solution vector at step k

Definition: Sequence $x^0, x^1, x^2, \cdots, x^n$ converges to the solution x^* with respect to norm $\| \, . \, \|$ if $\| \, x^k - x^* \, \| < \varepsilon$ when k is very large.

i.e. $k \to \infty, \| \, x^k - x^* \, \| \to 0$

A condition for convergence

Let error $e^k = x^k - x^*$

$$x^{k+1} = Hx^k + d$$

$$x^* = Hx^* + d$$

$$x^{k+1} - x^* = H(x^k - x^*)$$

$$e^{k+1} = He^*$$

$$\| e^{k+1} \| \leq \| He^k \| \leq \| H \| \| e^k \|$$

$$\leq \| H \|^2 \| e^{k-1} \| \leq \cdots \leq \| H \|^{k+1} \| e^0 \|$$

Then if $\| H \| < 1$, $\Longrightarrow$ The method converges.

Need to define the matrix norm.

Matrix Norm

Given a matrix of dimension $n \times n$, **Define**

$$\| H \|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} h_{ij} \quad \textbf{max sum row}$$

$$\| H \|_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} h_{ij} \quad \textbf{max sum column}$$

Example:

$$H = \begin{bmatrix} 5 & 9 \\ -2 & 1 \end{bmatrix}$$

$$\| H \|_\infty = max(14, 3) = 14$$

$$\| H \|_1 = max(7, 10) = 10$$

More on Convergence

Can you describe a type of matrix A so that solving $Ax = b$ iteratively can converge?

Theorem: If A is strictly diagonally dominant. Then both Gauss-Seidel and Jacobi methods converge.

Strictly diagonally dominant:

$$| a_{ii} | > \sum_{j=1, j \neq i}^{n} | a_{ij} | \qquad \text{i=1,2,...,n}$$

Example:

$$\begin{pmatrix} 6 & -2 & 1 \\ -2 & 7 & 2 \\ 1 & 2 & -5 \end{pmatrix} x = \begin{pmatrix} 11 \\ 5 \\ -1 \end{pmatrix}$$

A is strictly diagonally dominant:

$$| 6 | > 2 + 1$$

$$7 > 2 + 2$$

$$5 > 1 + 2$$

Then both Jacobi and G.S. methods will converge.

Conclusion

General iterative method:

Assign an initial value to $\vec{x}^{(0)}$

k=0

Do

$$\vec{x}^{(k+1)} = H * \vec{x}^{(k)} + d$$

until $\| \vec{x}^{(k+1)} - \vec{x}^{(k)} \| < \varepsilon$

Convergence condition:

If $\| H \| < 1$, $\Longrightarrow$ Then the method converges.

Handling of sparse iterative matrix: Use of a dense matrix in the implementation can be ineffective. Use of simplified array representation can improve speed and save space.