Iterative Methods and Parallel Algorithms

S. D. Margenov    I. D. Lirkov
margenov@parallel.bas.bg  ivan@parallel.bas.bg

Institute for Parallel Processing, Bulgarian Academy of Sciences, Sofia, Bulgaria

http://parallel.bas.bg/~margenov/
http://parallel.bas.bg/~ivan/
1. Introduction
2. Parallel inner product
3. Sparse matrix vector multiplication
4. Jacobi method
5. Conjugate gradient method
6. Preconditioned conjugate gradient method
7. Circulant Bloick Factorization
8. MIC(0) preconditioning
9. Parallel PCG tests
To establish the theoretical performance, a simple model for non-overlapped arithmetic and communication times is assumed:

- The execution of $M$ a.o. on one processor takes time

$$T_a = Mt_a$$

where $t_a$ is the average unit time to perform one a.o. on one processor.
The communication time to transfer $M$ data elements from one processor to another is approximated by

$$T_{com} = \ell(t_s + Mt_c)$$

where $t_s$ is the start-up time and $t_c$ is the time necessary for each of $M$ elements to be sent, and $\ell$ is the graph distance between the processors.
The standard expressions for parallel speedup $S(N, p)$, and parallel efficiency $E(N, p)$ are used:

$$S(N, p) = \frac{T(N, 1)}{T(N, p)}$$

$$E(N, p) = \frac{S(N, p)}{p}$$

Here, $T(N, p)$ stands for the parallel time to solve the problem on $p$ processors, and $N$ is the discrete size of the problem.
Parallel Speedup and Efficiency

The following theoretical estimates hold:

\[ 0 < S(N, p) \leq p \quad 0 < E(N, P) \leq 1 \]
Iterative methods are techniques to solve systems of linear equations

\[ A\hat{x} = b \]

that generate a sequence of approximations to the solution vector \( \hat{x} \) in the form

\[ x^0, x^1, \ldots, x^k, \ldots \]
Iterative Methods

The process is said to be convergent if the magnitude of the vector

$$g^k = b - Ax^k$$

becomes reasonably small. The vector $g^k$ represents the error in the approximation of $\hat{x}$ and is referred to as the residual after $k$ iterations.
The relative stopping criteria is determined by the inner product of the $k^{th}$ residual

$$\frac{\|g^k\|_2}{\|g^0\|_2} < \epsilon$$

where $\epsilon > 0$ is assumed small, and

$$\|g^i\|_2 = \sqrt{g^iTg^i}.$$
In this general setting, each iteration step includes:

- inner product
- matrix $A$ vector multiplication
The parallel implementation of the inner product is the only step of the considered iterative algorithms which requires global communications.
Parallel Inner Product I

\[ T_{com}^{IP} = (t_s + t_c) \log p \quad \text{d – cube} \]

\[ T_{com}^{IP} = 2(t_s + t_c)\left\lceil \sqrt{p}/2 \right\rceil \quad 2D – mesh \]
Parallel Inner Product II

Communications in the all-to-all like parallel inner product

\[
T_{com}^{IP} = (t_s + t_c) \log p \quad d - cube
\]

\[
T_{com}^{IP} = 2(t_s + t_c)(\sqrt{p} - 1) \quad 2D - mesh
\]
Example A1

\[ A = \begin{bmatrix}
4 & -1 & -1 \\
4 & -1 & -1 \\
4 & -1 & -1 \\
4 & -1 & -1 \\
-1 & -1 & 4 \\
-1 & -1 & 4 \\
-1 & -1 & 4 \\
-1 & -1 & 4 \\
-1 & -1 & 4 \\
\end{bmatrix} \]
Example A2

$$A = \begin{bmatrix}
4 & -1 & -1 \\
-1 & 4 & -1 \\
-1 & 4 & -1 \\
-1 & 4 & -1 \\
-1 & 4 & -1 \\
-1 & 4 & -1 \\
-1 & 4 & -1 \\
-1 & 4 & -1
\end{bmatrix}$$
Consider the model problem

\[-u_{xx} - u_{yy} = f \quad \text{in} \quad \Omega = [0, 1]^2\]

with Dirichlet boundary conditions on \( \Gamma = \partial \Omega \). Let us assume that FDM or FEM is used to solve numerically the problem where \( \omega_h \) is a uniform mesh with mesh-size \( h = 1/(n + 1) \).
If a column-wise numbering of the nodes (unknowns) is used, then

\[ A = \text{blocktridiag}(A_{i,i-1}, A_{i,i}, A_{i,i+1}), \]

\[
A = \begin{bmatrix}
A_{1,1} & A_{1,2} & & \\
A_{2,1} & A_{2,2} & A_{2,3} & \\
& A_{3,2} & A_{3,3} & A_{3,4} \\
& & \cdots & \cdots & \cdots & \cdots & \cdots \\
& & & \cdots & \cdots & \cdots & \cdots \\
& & & & \cdots & \cdots & \cdots & \cdots \\
& & & & & A_{n-1,n-2} & A_{n-1,n-1} & A_{n-1,n} \\
& & & & & & A_{n,n-1} & A_{n,n}
\end{bmatrix},
\]

\[ A_{i,i} = \text{tridiag}(-1, 4, -1), \quad A_{i,i-1} = A_{i,i+1} = -I, \]

\[ A \in \mathbb{N}^{N \times N}, \quad A_{i,j} \in \mathbb{N}^{n \times n}, \quad N = n^2. \]
Matrix-vector multiplication with a block-striped partitioning of the block-tridiagonal sparse matrix

\[ T(N, 1) \approx 9N t_a \]
\[ T_{com} = 2(t_s + nt_c) \]
\[ T(N, p) \approx \frac{9N}{p} t_a + 2(t_s + \sqrt{N} t_c) \]
The \( i^{th} \) equation of the system \( A\hat{x} = b \) can be written in the form

\[
\hat{x}_i = \frac{1}{A_{i,i}} \left( b_i - \sum_{i \neq j} A_{i,j} \hat{x}_j \right).
\]

The iteration step in the Jacobi method is

\[
x_{i}^{k+1} = \frac{1}{A_{i,i}} \left( b_i - \sum_{i \neq j} A_{i,j} x_j^k \right).
\]
The Jacobi Iterative Method

or equivalently

\[ x_{i}^{k+1} = \frac{g_{i}^{k}}{A_{i,i}} + x_{i}^{k}. \]

The method always converges in the class of diagonally-dominant matrices.
Jacoby Algorithm

1. procedure JACOBI\((A, b, x, \epsilon)\)
2. begin
3. \(k := 0;\)
4. Select initial solution vector \(x^0;\)
5. \(g^0 := b - Ax^0;\)
6. while \((\|g^k\|_2 > \epsilon\|g^0\|_2)\) do
   begin
   7. \(k := k + 1;\)
   8. for \(i := 1\) to \(N\) do \(x^k_i := \frac{g^k_i}{A_{i,i}} + x^{k-1}_i;\)
   9. \(g^k := b - Ax^k;\)
   10. endwhile;
11. \(x := x^k;\)
12. endJACOBI

The complexity of one iteration is as follows

\[
N_{it}^{Jac}(A^{-1}b) \approx N(Ad) + N(IP) + 3N
\]

which for the model problem reads as

\[
N_{it}^{CG}(A^{-1}b) \approx 14N.
\]

The related times are simply derived using the related matrix-vector communication estimate.

\[
T_{it}(N, 1) \approx 14N t_a
\]

\[
T_{com}^{it} = 2(t_s + nt_c) + T_{com}^{IP}
\]

\[
T_{it}(N, p) \approx \frac{14N}{p} t_a + 2(t_s + \sqrt{N} t_c)
\]
Conjugate Gradient Algorithm

1. procedure \( CG(A, b, x, \epsilon) \)
2. begin
3. \( k := 0 \);
4. Select initial solution vector \( x^0 \);
5. \( g^0 := Ax^0 - b, \quad d^0 = -g^0 \);
6. while \( (\|g^k\|_2 > \epsilon \|g^0\|_2) \) do
7. begin
8. \( \tau_k = \frac{g^k g^k T g^k}{d^k T A d^k} \);
9. \( x^{k+1} = x^k + \tau_k d^k \);
10. \( g^{k+1} = g^k + \tau_k A d^k \);
11. \( \beta_k = \frac{g^{k+1} g^{k+1} T g^k}{g^k T g^k} \);
12. \( d^{k+1} = -g^{k+1} + \beta_k d^k \);
13. endwhile;
14. \( x := x^k \);
15. end CG

The computational complexity of one CG iteration is as follows:

\[
N_{it}^{CG}(A^{-1}b) \approx N(Ad) + 2N(IP) + 3N(LT),
\]

\[
N_{it}^{CG}(A^{-1}b) \approx 19N.
\]

The related times are derived in a similar way as for Jacobi method.

\[
T_{it}^N(1, 1) \approx 19Nt_a
\]

\[
T_{it}^p = 2(t_s + nt_c) + 2T_{com}^{IP}
\]

\[
T_{it}^N(1, p) \approx \frac{19N}{p} t_a + 2(t_s + \sqrt{N}t_c)
\]
Theorem.

\[ p(\epsilon) \leq \frac{1}{2} \sqrt{\kappa(A)} \ln \left( \frac{2}{\epsilon} \right) + 1, \]

where \( p(\epsilon) \) stands for the smallest number \( k \) such that

\[ \| x^k - \hat{x} \|_A \leq \epsilon \| x^0 - \hat{x} \|_A \quad \forall x^0 \in \mathbb{R}^N. \]

In a very general setting of FEM/FDM sparse matrices, the spectral condition number behaves as \( \kappa(A) = O(N) \).
Therefore $\mathcal{N}^{CG}(A^{-1}b) = O(N^{3/2})$. It is important to note, that the same complexity holds for the best known direct method, namely $\mathcal{N}^{ND}(A^{-1}b) = O(N^{3/2})$ where $ND$ stands for the Nested Dissection Method.
The number of iterations for the model problem of Gauss-Seidel (G-S), Steepest Descent (SD), Conjugate Gradient (CG) and Preconditioned CG (PCG) methods are presented in the table. The implemented PCG algorithm is subject to the next section.

\[
\begin{align*}
n_{it}^{G-S} &= O(N) \\
n_{it}^{SD} &= O(N) \\
n_{it}^{CG} &= O(N^{1/2}) \\
n_{it}^{PCG} &= O(N^{1/4})
\end{align*}
\]
Numerical Tests

<table>
<thead>
<tr>
<th>$n$</th>
<th>$G - S$</th>
<th>$SD$</th>
<th>$CG$</th>
<th>$PCG$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>82</td>
<td>185</td>
<td>26</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>309</td>
<td>698</td>
<td>45</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>1151</td>
<td>2592</td>
<td>91</td>
<td>19</td>
</tr>
<tr>
<td>32</td>
<td>4242</td>
<td>9541</td>
<td>177</td>
<td>27</td>
</tr>
<tr>
<td>64</td>
<td>15529</td>
<td>34818</td>
<td>351</td>
<td>38</td>
</tr>
</tbody>
</table>

Conclusion. The efforts should be addressed to development of scalable parallel algorithms for fast enough iterative solution methods.
The idea of the PCG method is to substitute the original linear system to a new one which is better conditioned:

\[ A\hat{x} = b \quad \Rightarrow \quad C^{-1/2}AC^{-1/2}\hat{y} = \bar{b} \]

The PCG strategy is to construct a preconditioner \( C \) such that:

1. \( \kappa(C^{-1}A) << \kappa(A) \)
2. \( \mathcal{N}(C^{-1}v) << \mathcal{N}(A^{-1}v) \)

The preconditioner is called optimal if \( \kappa(C^{-1}A) = O(1) \) and \( \mathcal{N}(C^{-1}v) = O(N) \).
1. procedure PCG($A, b, x, \epsilon$)
2. begin
3. $k := 0$;
4. Select initial solution vector $x^0$;
5. $g^0 = Ax^0 - b$, $h^0 = C^{-1}g^0$, $d^0 = h^0$;
6. while ($\|g^k\|_{C^{-1}} > \epsilon \|g^0\|_{C^{-1}}$) do
7. begin
8. $\tau_k = \frac{g^k h_k^T}{d^k h_k^T A d^k}$;
9. $x^{k+1} = x^k + \tau_k d^k$;
10. $g^{k+1} = g^k + \tau_k A d^k$;
11. $h^{k+1} = C^{-1}g^{k+1}$;
12. $\beta_k = \frac{g^{k+1} h^{k+1}^T}{g^k h_k^T}$;
13. $d^{k+1} = -h^{k+1} + \beta_k d^k$;
14. endwhile;
15. $x := x^k$;
16. endPCG

Following the structure of our analyses, we estimate the computational complexity of one PCG iteration in the form:

$N_{it}^{PCG}(A^{-1}b) \approx N(C^{-1}g) + N(Ad) + 2N(IP) + 3N(LT)$

$N_{it}^{PCG}(A^{-1}b) \approx N(C^{-1}g) + 19N.$

Then, the related per iteration PCG times are as follows:

$T_{it}(N, 1) \approx T(C^{-1}g)(N, 1) + 19 N t_a,$

$T_{com} = 2(t_s + n t_c) + T_{com}^{(C^{-1}g)} + 2T_{com}^{IP}$
Theorem.

\[ p(\epsilon) \leq \frac{1}{2} \sqrt{\kappa(C^{-1}A)} \ln \left( \frac{2}{\epsilon} \right) + 1, \]

where \( p(\epsilon) \) stands for the smallest number \( k \) such that

\[ \|x^k - \hat{x}\|_A \leq \epsilon \|x^0 - \hat{x}\|_A \quad \forall x^0 \in \mathbb{N} \]
Convergence Rate of PCG Method

Some parallel preconditioning techniques:

- Incomplete Factorization
- Circulant Block Factorization
- Domain Decomposition
- Patched Local Refinement
- Multigrid/Multilevel
- Approximate Inverse
A circulant matrix $C$ has the form

$$C_{k,j} = c_{(j-k) \mod m}$$

$$C = \begin{pmatrix}
  c_0 & c_1 & c_2 & \cdots & c_{m-1} \\
  c_{m-1} & c_0 & c_1 & \cdots & c_{m-2} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  c_1 & c_2 & \cdots & c_{m-1} & c_0
\end{pmatrix}$$

$$C = (c_0, c_1, \ldots, c_{m-1}) = F \Lambda F^*$$

$$\mathcal{N}(C^{-1}v) = \mathcal{O}(m \log m)$$
2D model problem

\[-(a(x, y)u_x)_x - (b(x, y)u_y)_y = f(x, y), \quad \forall (x, y) \in \Omega,\]

\[u(x, y) = 0, \quad \forall (x, y) \in \Gamma = \partial \Omega,\]

\[0 < c_{\min} \leq a(x, y), b(x, y) \leq c_{\max},\]

\[A = \text{tridiag}(-A_{i,i-1}, A_{i,i}, -A_{i,i+1}) \quad i = 1, 2, \ldots n,\]

\[C = \text{tridiag}(-C_{i,i-1}, C_{i,i}, -C_{i,i+1}) \quad i = 1, 2, \ldots n,\]

where \(C_{i,j} = \text{Circulant}(A_{i,j})\) is some given circulant approximation of the corresponding block \(A_{i,j}\).
\[ C = D - L - U \]
\[ C = (X - L)(I - X^{-1}U) \]
\[ X = D - LX^{-1}U \]
\[ X_1 = C_{1,1} \]
\[ X_i = C_{i,i} - C_{i,i-1}X_{i-1}^{-1}C_{i-1,i}, \quad i = 2, \ldots, n \]

\[ C_{i,j} = F \Lambda_{i,j} F^* \]
\[ X_i = F D_i F^* \]
Factorization

\[ D_1^{-1} = \Lambda_{1,1} \]
\[ D_i^{-1} = \Lambda_{i,i} - \Lambda_{i,i-1} D_{i-1} \Lambda_{i-1,i} \cdot \]

Let us denote with \( \Lambda = \text{tridiag}(\Lambda_{i,i-1}, \Lambda_{i,i}, \Lambda_{i,i+1}) \).

Then the following relation holds

\[ Cw = u \iff (I \otimes F)\Lambda(I \otimes F^*)w = u. \]

\[ \hat{u} = (I \otimes F^*)u \]
\[ \Lambda \hat{w} = \hat{u} \]
\[ w = (I \otimes F)\hat{w} \]
\[
\begin{align*}
\hat{v}_1 &= D_1 \hat{u}_1 \\
\hat{v}_i &= D_i (\hat{u}_i - \Lambda_{i,i-1} \hat{v}_{i-1}) \quad i = 2, 3, \ldots n \\
\hat{w}_n &= \hat{v}_n \\
\hat{w}_i &= \hat{v}_i - D_i \Lambda_{i,i+1} \hat{w}_{i+1} \quad i = n - 1, n - 2, \ldots 1
\end{align*}
\]
The CBF preconditioning can be split in three stages. If we use the column-wise mapping for the first and third stage there is no need of communication because we perform block-FFT on blocks which are stored on one processor. For the second stage we have to reorder the vector entries using row-wise mapping.
The CBF preconditioning can be split in three stages. If we use the column-wise mapping for first and third stage there is no need of communication because we perform block-FFT on blocks which are stored on one processor. For second stage we have to reorder the vector entries using row-wise mapping.
### Parallel CBF tests

#### SUN Ultra-Enterprise Symmetric Multiprocessor

<table>
<thead>
<tr>
<th></th>
<th>168 MHz</th>
<th></th>
<th></th>
<th>250 MHz</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(T(p))</td>
<td>(S_p)</td>
<td>(E_p)</td>
<td>(T(p))</td>
<td>(S_p)</td>
</tr>
<tr>
<td>128</td>
<td>1</td>
<td>0.086</td>
<td>1.84</td>
<td>0.92</td>
<td>0.081</td>
<td>1.71</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.047</td>
<td>3.04</td>
<td>0.76</td>
<td>0.047</td>
<td>2.77</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.028</td>
<td>3.04</td>
<td>0.76</td>
<td>0.029</td>
<td>2.77</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.021</td>
<td>4.13</td>
<td>0.52</td>
<td>0.096</td>
<td>0.84</td>
</tr>
<tr>
<td>256</td>
<td>1</td>
<td>0.389</td>
<td>1.88</td>
<td>0.94</td>
<td>0.392</td>
<td>1.88</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.207</td>
<td>1.88</td>
<td>0.94</td>
<td>0.208</td>
<td>3.09</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.109</td>
<td>3.56</td>
<td>0.89</td>
<td>0.127</td>
<td>3.09</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.065</td>
<td>6.02</td>
<td>0.75</td>
<td>0.138</td>
<td>2.83</td>
</tr>
</tbody>
</table>
Parallel CBF tests

SUN Ultra-Enterprise Symmetric Multiprocessor

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>168 MHz</th>
<th></th>
<th></th>
<th>250 MHz</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( T(p) )</td>
<td>( S_p )</td>
<td>( E_p )</td>
<td>( T(p) )</td>
<td>( S_p )</td>
<td>( E_p )</td>
</tr>
<tr>
<td>( n )</td>
<td>( p )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### SUN Ultra-Enterprise Symmetric Multiprocessor

<table>
<thead>
<tr>
<th>$n$</th>
<th>$p$</th>
<th>$T(p)$</th>
<th>$S_p$</th>
<th>$E_p$</th>
<th>$T(p)$</th>
<th>$S_p$</th>
<th>$E_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>420</td>
<td>1</td>
<td>3.718</td>
<td></td>
<td></td>
<td>2.651</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.922</td>
<td>1.93</td>
<td>0.97</td>
<td>1.378</td>
<td>1.92</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.313</td>
<td>2.83</td>
<td>0.94</td>
<td>0.937</td>
<td>2.83</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.990</td>
<td>3.75</td>
<td>0.94</td>
<td>0.714</td>
<td>3.71</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.817</td>
<td>4.55</td>
<td>0.91</td>
<td>1.005</td>
<td>2.64</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.679</td>
<td>5.48</td>
<td>0.91</td>
<td>1.233</td>
<td>2.15</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>0.595</td>
<td>6.25</td>
<td>0.89</td>
<td>1.314</td>
<td>2.02</td>
<td>0.29</td>
</tr>
</tbody>
</table>
Let us rewrite the real matrix $A$ in the form $A = D - L - L^T$. Then, the modified incomplete Cholesky factorization is defined as follows:

$$C_{MIC(0)}(A) = (X - L)X^{-1}(X - L)^T,$$

where $X = diag(x_1, \cdots, x_N)$ provides the equal rowsums condition.
Theorem. Let us assume that

(a) \( L \geq 0 \),  
(b) \( A \leq 0 \),  
(c) \( A \leq + L^t \leq > 0 \), \( \leq = (1, \ldots, 1)^t \in \mathbb{N} \). Then the relation

\[
x_i = a_{ii} - \sum_{k=1}^{i-1} \frac{a_{ik}}{x_k} \sum_{j=k+1}^{N} a_{kj} > 0
\]

gives a stable \( MIC(0) \) factorization of \( A \).
**Remark.** All presented numerical tests are performed using the perturbed $MIC(0)$ algorithm, where the incomplete factorization is applied to the matrix $\tilde{A} = A + \tilde{D}$. The diagonal perturbation $\tilde{D} = \tilde{D}(\xi) = \text{diag}(\tilde{d}_1, \ldots, \tilde{d}_N)$ is defined as follows:

$$
\tilde{d}_i = \begin{cases} 
\xi a_{ii} & \text{if } a_{ii} \geq 2w_i \\
\xi^{1/2} a_{ii} & \text{if } a_{ii} < 2w_i
\end{cases}
$$
where

\[ w_i = - \sum_{j > i} a_{ij}. \]

Here \( 0 < \xi < 1 \) is a constant of the same order as the minimal eigenvalue of \( A \). The computations for the considered model problems are done with \( \xi = h^2 \).
The MIC(0) computational complexity of one PCG iteration is as follows:

\[ \mathcal{N}^{PCG}_{\text{it}}(A^{-1}b) \approx \mathcal{N}(C^{-1}g) + 19N, \quad \mathcal{N}(C^{-1}g) \approx 11N, \]

\[ \mathcal{N}^{PCG}_{\text{it}}(A^{-1}b) \approx 30N. \]

MIC(0) is a cheap preconditioning algorithm. The cost of \( \mathcal{N}(C^{-1}g) \) is almost the same as \( \mathcal{N}(Ad) \).
MIC(0) Complexity

- MIC(0) is a robust preconditioner with respect to local singularities of the problem, where
  \[ \kappa(C^{-1}A) = O(N^{1/4}), \text{ and} \]
  \[ \mathcal{N}^{PCG}(A^{-1}b) = O(N^{5/4}). \]

- MIC(0) is an inherently sequential algorithm.
The model problem is considered again:
\[-u_{xx} - u_{yy} = f \text{ in } \Omega = [0, 1]^2\] with Dirichlet boundary conditions on \( \Gamma = \partial\Omega \).

Since a five point stencil is used in both cases, the accuracy of the regular mesh (ReM) and the alternative skewed mesh (SkM) FDM/FEM approximations are one and the same.
The bottleneck problem of the parallel implementation of MIC(0) algorithm is the solution of problems with triangle matrices $(X - L)$ and $(X - L)^T$. 
The key point of our consideration is, that in the case of skewed mesh, the stiffness matrix has a block structure with diagonal blocks which are diagonal.
Parallel MIC(0) algorithm

MIC(0) PCG algorithm with a block-stripped partitioning: \( N = n^2 + (n - 1)^2 \).

\[
T_{it}^{MIC(0)}(N, 1) \approx 38Nt_a, \quad T_{com} \approx (4t_s + 6t_c)n + 2T_{com}^{IP},
\]

\[
T_{it}^{MIC(0)}(N, p) \approx \frac{38N}{p}t_a + (2t_s + 3t_c)\sqrt{2N}
\]
Parallel MIC(0) Tests

- The presented tests are performed on a Beowulf like cluster of four dual processor Power Macintosh computers with 512 MB RAM each and G4 processors on 450 MHz.
- The parallel MIC(0) algorithm is implemented in C++ using Message Passing Interface (MPI).
- Yellow Dog Linux with LAN MPI are used.
- The size of the problem and the number of the processors are varied to examine the parallel scalability of the code.
<table>
<thead>
<tr>
<th>n</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
<th>1500</th>
</tr>
</thead>
<tbody>
<tr>
<td>S(n,2)</td>
<td>1.21</td>
<td>1.68</td>
<td>1.96</td>
<td>1.85</td>
<td>1.92</td>
<td>2.03</td>
<td>2.02</td>
</tr>
<tr>
<td>S(n,3)</td>
<td>0.24</td>
<td>0.46</td>
<td>0.97</td>
<td>1.72</td>
<td>2.45</td>
<td>2.97</td>
<td>2.86</td>
</tr>
<tr>
<td>S(n,4)</td>
<td>0.22</td>
<td>0.46</td>
<td>1.11</td>
<td>1.97</td>
<td>2.88</td>
<td>3.76</td>
<td>3.95</td>
</tr>
<tr>
<td>S(n,5)</td>
<td>0.20</td>
<td>0.40</td>
<td>0.96</td>
<td>1.99</td>
<td>3.25</td>
<td>4.48</td>
<td>4.86</td>
</tr>
<tr>
<td>S(n,6)</td>
<td>0.18</td>
<td>0.39</td>
<td>1.03</td>
<td>1.99</td>
<td>3.55</td>
<td>5.23</td>
<td>5.73</td>
</tr>
<tr>
<td>S(n,7)</td>
<td>0.19</td>
<td>0.38</td>
<td>0.95</td>
<td>1.78</td>
<td>3.63</td>
<td>6.02</td>
<td>6.31</td>
</tr>
<tr>
<td>S(n,8)</td>
<td>0.19</td>
<td>0.39</td>
<td>1.00</td>
<td>2.28</td>
<td>3.97</td>
<td>6.37</td>
<td>6.76</td>
</tr>
</tbody>
</table>
## Parallel Efficiency

<table>
<thead>
<tr>
<th>n</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
<th>1500</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(n,2)</td>
<td>0.60</td>
<td>0.84</td>
<td>0.98</td>
<td>0.93</td>
<td>0.96</td>
<td>1.02</td>
<td>1.01</td>
</tr>
<tr>
<td>E(n,3)</td>
<td>0.08</td>
<td>0.15</td>
<td>0.32</td>
<td>0.57</td>
<td>0.81</td>
<td>0.99</td>
<td>0.95</td>
</tr>
<tr>
<td>E(n,4)</td>
<td>0.06</td>
<td>0.12</td>
<td>0.27</td>
<td>0.49</td>
<td>0.72</td>
<td>0.94</td>
<td>0.98</td>
</tr>
<tr>
<td>E(n,5)</td>
<td>0.04</td>
<td>0.08</td>
<td>0.19</td>
<td>0.40</td>
<td>0.65</td>
<td>0.90</td>
<td>0.97</td>
</tr>
<tr>
<td>E(n,6)</td>
<td>0.03</td>
<td>0.07</td>
<td>0.17</td>
<td>0.33</td>
<td>0.59</td>
<td>0.87</td>
<td>0.96</td>
</tr>
<tr>
<td>E(n,7)</td>
<td>0.03</td>
<td>0.05</td>
<td>0.14</td>
<td>0.25</td>
<td>0.51</td>
<td>0.86</td>
<td>0.90</td>
</tr>
<tr>
<td>E(n,8)</td>
<td>0.02</td>
<td>0.05</td>
<td>0.13</td>
<td>0.29</td>
<td>0.50</td>
<td>0.80</td>
<td>0.84</td>
</tr>
</tbody>
</table>