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/

CONTENTS

Parallel Performance

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.

Parallel Performance

- 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 ℓ is the graph distance between the processors.

Parallel Speedup and Efficiency

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:

$$0 < S(N,p) \leq p \qquad 0 < E(N,P) \leq 1$$

Iterative Methods

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

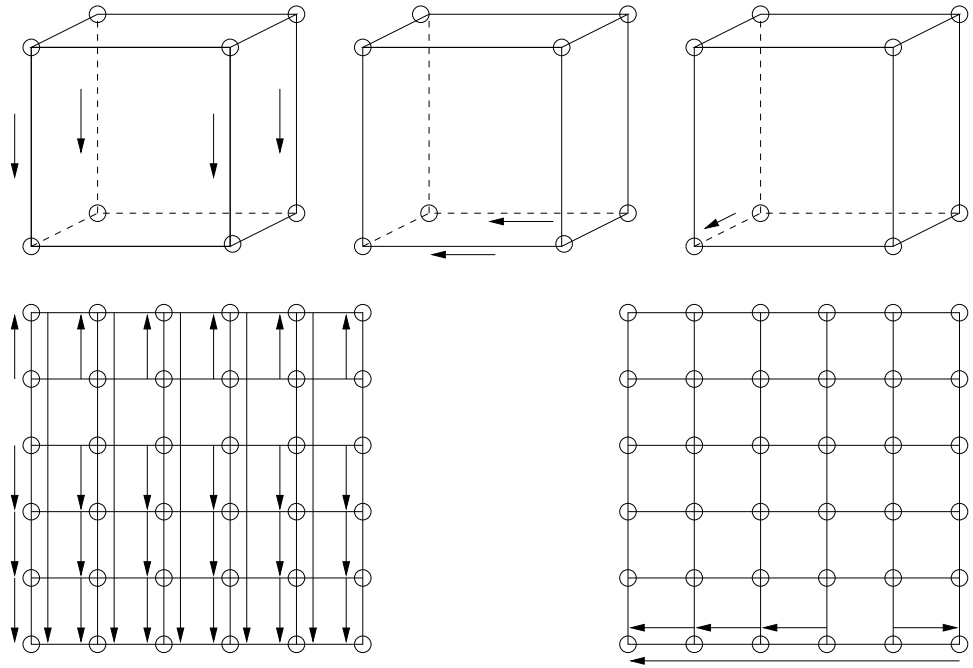
$$\|g^i\|_2 = \sqrt{g^{iT} g^i}.$$

In this general setting, each iteration step includes:

- inner product
- matrix A vector multiplication

Parallel Inner Product I

The parallel implementation of the inner product is the only step of the considered iterative algorithms which requires global communications.

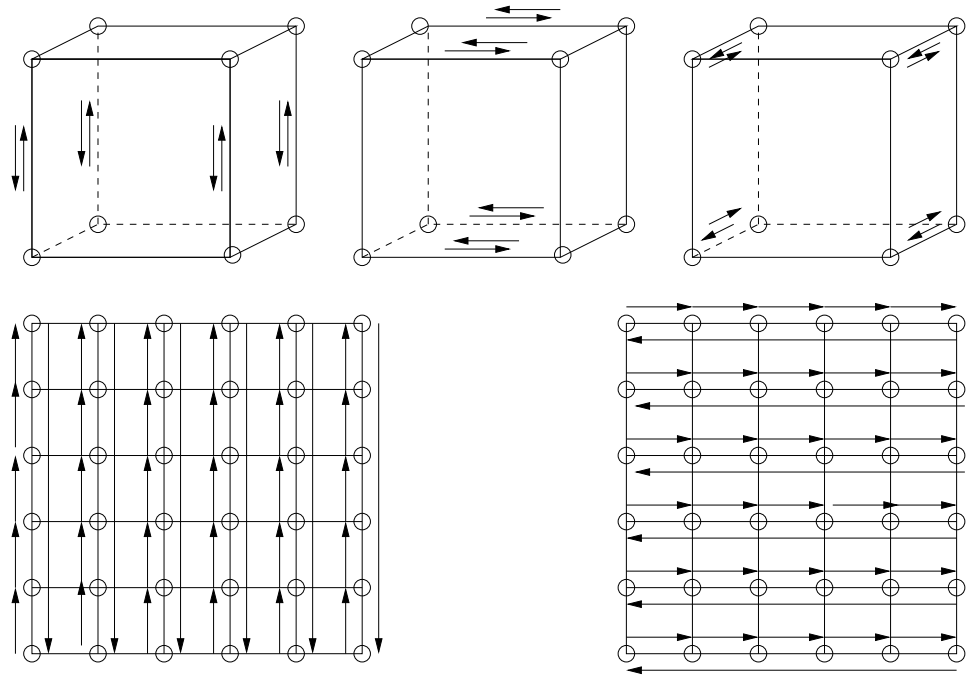


Communications in the **one-to-all like** parallel inner product

Parallel Inner Product I

$$T_{com}^{IP} = (t_s + t_c) \log p \quad d - cube$$
$$T_{com}^{IP} = 2(t_s + t_c)\lceil \sqrt{p}/2 \rceil \quad 2D - mesh$$

Parallel Inner Product II



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

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

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

Sparse Matrices

Example A1

$$A = \begin{bmatrix}
4 & & & & & -1 & & & -1 \\
 & 4 & & & & & & -1 & -1 \\
 & & 4 & & & & -1 & -1 & \\
 & & & 4 & & -1 & -1 & & \\
 & & & & 4 & -1 & -1 & -1 & -1 \\
-1 & & & -1 & -1 & 4 & & & \\
 & & -1 & -1 & -1 & & 4 & & \\
 & -1 & -1 & & -1 & & & 4 & \\
-1 & -1 & & & -1 & & & & 4
\end{bmatrix}$$

Example A2

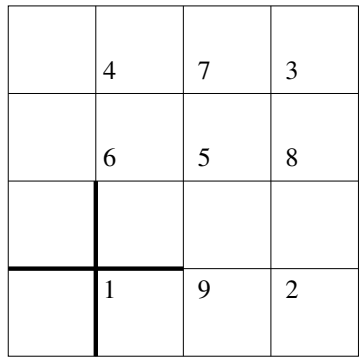
$$A = \begin{bmatrix}
4 & -1 & & -1 & & & & & \\
-1 & 4 & -1 & & -1 & & & & \\
 & -1 & 4 & & & -1 & & & \\
-1 & & & 4 & -1 & & -1 & & \\
 & -1 & & -1 & 4 & -1 & & -1 & \\
 & & -1 & & -1 & 4 & & & -1 \\
 & & & -1 & & & 4 & -1 & \\
 & & & & -1 & & -1 & 4 & -1 \\
 & & & & & -1 & & -1 & 4
\end{bmatrix}$$

FDM/FEM Sparse Matrices I

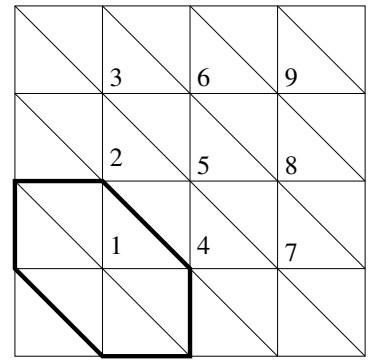
Consider the model problem

$$-u_{xx} - u_{yy} = f \qquad in \;\; \Omega = [0,1]^2$$

with Dirichlet boundary conditions on $\Gamma = \partial\Omega$. Let us assume that FDM or FEM is used to solve numerically the problem where ω_h is a uniform mesh with mesh-size $h = 1/(n+1)$.



(A1)



(A2)

FDM/FEM Sparse Matrices

If a column-wise numbering of the nodes (unknowns) is used, then

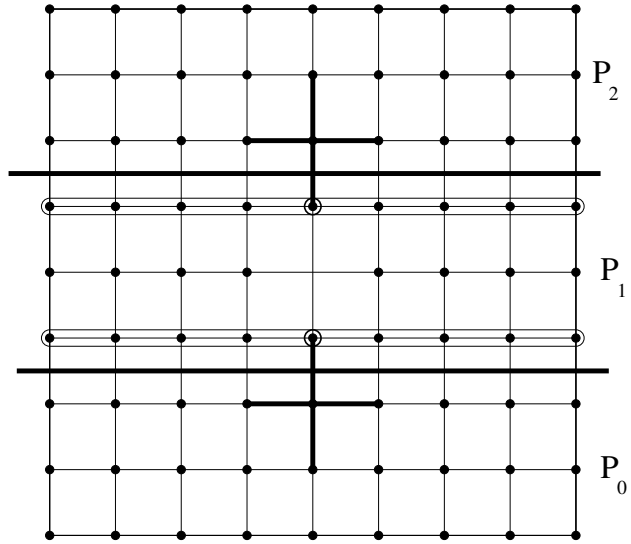
$$A = blocktridiag(A_{i,i-1}, A_{i,i}, A_{i,i+1}),$$

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

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

$$A \in^{N \times N}, \quad A_{i,j} \in^{n \times n}, \quad N = n^2.$$

Matrix-Vector Multiplication I



Matrix-vector multiplication with a block-stripped partitioning of the block-tridiagonal sparse matrix

$$
\begin{aligned}
T(N,1) &\approx 9Nt_a \\
T_{com} &= 2(t_s + nt_c) \\
T(N,p) &\approx \frac{9N}{p}t_a + 2(t_s + \sqrt{N}t_c)
\end{aligned}
$$

Jacobi Iterative Method

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

$$x_i^{k+1} = \frac{1}{A_{i,i}} \left(b_i - \sum_{i \neq j} A_{i,j} x_j^k \right)$$

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, ϵ)
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**
7. **begin**
8. $k := k + 1$;
9. **for** $i := 1$ **to** N **do** $x_i^k := \frac{g_i^{k-1}}{A_{i,i}} + x_i^{k-1}$;
11. $g^k := b - Ax^k$;
12. **endwhile**;
13. $x := x^k$;
14. **end**JACOBI

The complexity of one iteration is as follows

$$\mathcal{N}_{it}^{Jac}(A^{-1}b) \approx \mathcal{N}(Ad) + \mathcal{N}(IP) + 3N$$

which for thte model problem reads as

$$\mathcal{N}_{it}^{CG}(A^{-1}b) \approx 14N.$$

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

$$T^{it}(N, 1) \approx 14Nt_a$$

$$T_{com}^{it} = 2(t_s + nt_c) + T_{com}^{IP}$$

$$T^{it}(N, p) \approx \frac{14N}{p}t_a + 2(t_s + \sqrt{N}t_c)$$

Conjugate Gradient Algorithm

1. **procedure** CG(A, b, x, ϵ)
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. $\tau_k = \frac{{g^k}^T g^k}{{d^k}^T A d^k}$;
9. $x^{k+1} = x^k + \tau_k d^k$;
10. $g^{k+1} = g^k + \tau_k A d^k$;
11. $\beta_k = \frac{{g^{k+1}}^T g^{k+1}}{{g^k}^T g^k}$;
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:

$$\mathcal{N}_{it}^{CG}(A^{-1}b) \approx \mathcal{N}(Ad) + 2\mathcal{N}(IP) + 3\mathcal{N}(LT),$$

$$\mathcal{N}_{it}^{CG}(A^{-1}b) \approx 19N.$$

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

$$T^{it}(N, 1) \approx 19Nt_a$$

$$T_{com}^{it} = 2(t_s + nt_c) + 2T_{com}^{IP}$$

$$T^{it}(N, p) \approx \frac{19N}{p} t_a + 2(t_s + \sqrt{N} t_c)$$

Convergence Rate of CG Method

Theorem.

$$p(\epsilon) \leq \frac{1}{2}\sqrt{\kappa(A)}\ln(2/\epsilon) + 1,$$

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

$$\|x^k - \hat{x}\|_A \leq \epsilon \|x^0 - \hat{x}\|_A \qquad \forall x^0 \in R^N.$$

In a very general setting of FEM/FDM sparse matrices, the spectral condition number behaves as $\kappa(A) = O(N)$.

Convergence Rate of CG Method

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.

Numerical Tests

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.

$$
\begin{aligned}
n_{it}^{G-S} &= O(N) \\
n_{it}^{SD} &= O(N) \\
n_{it}^{CG} &= O(N^{1/2}) \\
n_{it}^{PCG} &= O(N^{1/4})
\end{aligned}
$$

Numerical Tests

n	$G-S$	SD	CG	PCG
4	82	185	26	11
8	309	698	45	15
16	1151	2592	91	19
32	4242	9541	177	27
64	15529	34818	351	38

Conclusion. The efforts should be addressed to development of scalable parallel algorithms for fast enough iterative solution methods.

Preconditioned CG Method

The idea of the PCG method is to substitute the original linear system to a new one which is better conditioned:

$$A\hat{x} = b \quad => \quad C^{-1/2}AC^{-1/2}\hat{y} = \bar{b}$$

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

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

The preconditioner is called optimal if $\kappa(C^{-1}A) = O(1)$ and $\mathcal{N}(C^{-1}v) = O(N)$.

PCG Algorithm

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

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

$$\begin{aligned} \mathcal{N}_{it}^{PCG}(A^{-1}b) &\approx \mathcal{N}(C^{-1}g) + \mathcal{N}(Ad) \\ &+ 2\mathcal{N}(IP) + 3\mathcal{N}(LT) \end{aligned}$$

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

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

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

$$T_{com}^{it} = 2(t_s + nt_c) + T_{com}^{(C^{-1}g)} + 2T_{com}^{IP}$$

Convergence Rate of PCG Method

Theorem.

$$p(\epsilon) \leq \frac{1}{2}\sqrt{\kappa(C^{-1}A)}\ln(2/\epsilon) + 1,$$

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

$$\|x^k - \hat{x}\|_A \leq \epsilon \|x^0 - \hat{x}\|_A \qquad \forall x^0 \in^N$$

Convergence Rate of PCG Method

Some parallel preconditioning techniques:

- Incomplete Factorization
- Circulant Bloick Factorization
- Domain Decomposition
- Patched Local Refinement
- Multigrid/Multilevel
- Approximate Inverse

Circulant Bloick Factorization

A circulant matrix C has the form

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

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

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

$$\aleph(C^{-1}v) = \mathcal{O}(m \log m)$$

2D model problem

$$
\begin{aligned}
-(a(x,y)u_x)_x - (b(x,y)u_y)_y &= f(x,y), \\
&\quad \forall (x,y) \in \Omega, \\
u(x,y) = 0, \qquad \forall (x,y) \in \Gamma &= \partial\Omega, \\
0 < c_{\min} \leq a(x,y),\, b(x,y) &\leq c_{\max},
\end{aligned}
$$

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

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

where $C_{i,j} = Circulant(A_{i,j})$ is some given circulant approximation of the corresponding block $A_{i,j}$.

Factorization

$$
\begin{aligned}
C &= D - L - U \\
C &= (X - L)(I - X^{-1}U) \\
X &= D - LX^{-1}U \\
X_1 &= C_{1,1} \\
X_i &= C_{i,i} - C_{i,i-1}X_{i-1}^{-1}C_{i-1,i}, \quad i = 2, \ldots n
\end{aligned}
$$

$$
\begin{aligned}
C_{i,j} &= F\Lambda_{i,j}F^* \\
X_i &= FD_iF^*
\end{aligned}
$$

Factorization

$$
\begin{aligned}
D_1^{-1} &= \Lambda_{1,1} \\
D_i^{-1} &= \Lambda_{i,i} - \Lambda_{i,i-1} D_{i-1} \Lambda_{i-1,i}.
\end{aligned}
$$

Let us denote with $\Lambda = tridiag(\Lambda_{i,i-1}, \Lambda_{i,i}, \Lambda_{i,i+1})$.
Then the following relation holds

$$
Cw = u \qquad \Longleftrightarrow \qquad (I \otimes F)\Lambda(I \otimes F^*)w = u.
$$

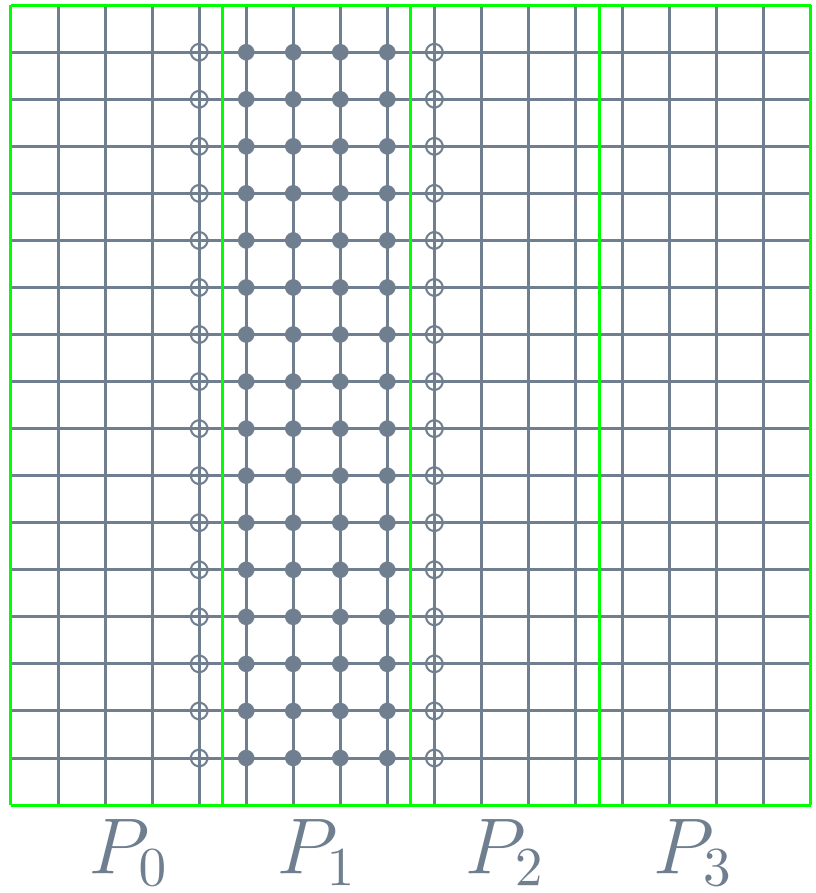
$$
\begin{aligned}
\hat{u} &= (I \otimes F^*)u \\
\Lambda\hat{w} &= \hat{u} \\
w &= (I \otimes F)\hat{w}
\end{aligned}
$$

Factorization

$$\left|\begin{array}{ll} \hat{v}_1 = D_1 \hat{u}_1 & \\ \hat{v}_i = D_i(\hat{u}_i - \Lambda_{i,i-1}\hat{v}_{i-1}) & i = 2, 3, \ldots n \end{array}\right.$$

$$\left|\begin{array}{ll} \hat{w}_n = \hat{v}_n & \\ \hat{w}_i = \hat{v}_i - D_i \Lambda_{i,i+1} \hat{w}_{i+1} & i = n-1, n-2, \ldots 1 \end{array}\right.$$

Parallel algorithm



Distribution of vectors on processors.

Parallel algorithm

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

		168 MHz			250 MHz		
n	p	$T(p)$	S_p	E_p	$T(p)$	S_p	E_p
128	1	0.086			0.081		
	2	0.047	1.84	0.92	0.047	1.71	0.86
	4	0.028	3.04	0.76	0.029	2.77	0.69
	8	0.021	4.13	0.52	0.096	0.84	0.10
256	1	0.389			0.392		
	2	0.207	1.88	0.94	0.208	1.88	0.94
	4	0.109	3.56	0.89	0.127	3.09	0.77
	8	0.065	6.02	0.75	0.138	2.83	0.35

Parallel CBF tests

SUN Ultra-Enterprise Symmetric Multiprocessor

		168 MHz			250 MHz		
n	p	$T(p)$	S_p	E_p	$T(p)$	S_p	E_p
384	1	1.460			1.498		
	2	0.759	1.92	0.96	0.783	1.91	0.96
	3	0.523	2.79	0.93	0.533	2.81	0.94
	4	0.394	3.71	0.93	0.473	3.17	0.79
	6	0.269	5.43	0.90	0.780	1.92	0.32
	8	0.338	4.32	0.54	1.122	1.33	0.17

Parallel CBF tests

SUN Ultra-Enterprise Symmetric Multiprocessor

		168 MHz			250 MHz		
n	p	$T(p)$	S_p	E_p	$T(p)$	S_p	E_p
420	1	3.718			2.651		
	2	1.922	1.93	0.97	1.378	1.92	0.96
	3	1.313	2.83	0.94	0.937	2.83	0.94
	4	0.990	3.75	0.94	0.714	3.71	0.93
	5	0.817	4.55	0.91	1.005	2.64	0.53
	6	0.679	5.48	0.91	1.233	2.15	0.36
	7	0.595	6.25	0.89	1.314	2.02	0.29

MIC(0) Algorithm

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.

MIC(0) Algorithm

Theorem. Let us assume that $(a)\ L \geq 0, \quad (b)\ A\underline{e} \geq 0, \quad (c)\ A\underline{e} + L^t\underline{e} > 0,\ \underline{e} = (1, \cdots, 1)^t \in^N$. 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 $\tilde{D} = \tilde{D}(\xi) = diag(\tilde{d}_1, \ldots \tilde{d}_N)$ is defined as follows:

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

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 considerd model problems are done with $\xi = h^2$.

MIC(0) Complexity

The MIC(0) computational complexity of one PCG iteration is as follows:

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

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

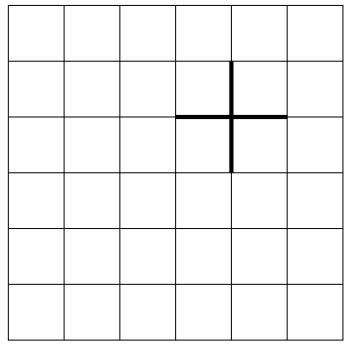
- MIC(0) is a cheap preconditioning algorithm. The cost of $\mathcal{N}(C^{-1}g)$ is almost the same as $\mathcal{N}(Ad)$.

MIC(0) Complexity

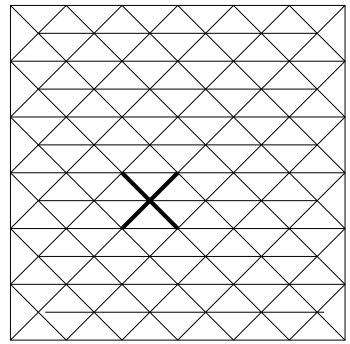
- MIC(0) is a robust preconditioner with respect to local singularities of the problem, where $\kappa(C^{-1}A) = O(N^{1/4})$, and $\mathcal{N}^{PCG}(A^{-1}b) = O(N^{5/4})$.
- MIC(0) is an inherently sequential algorithm.

FDM/FEM Sparse Matrices II

The model problem is considered again: $-u_{xx} - u_{yy} = f$ in $\Omega = [0,1]^2$ with Dirichlet boundary conditions on $\Gamma = \partial\Omega$.



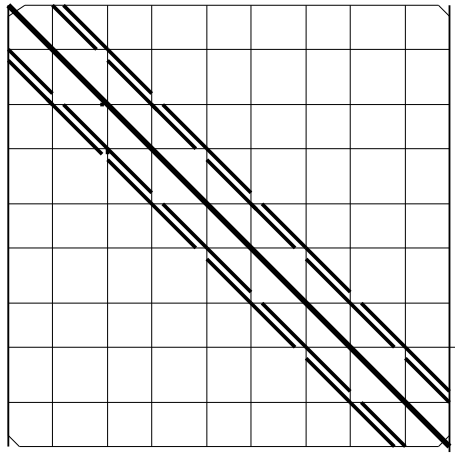
ReM



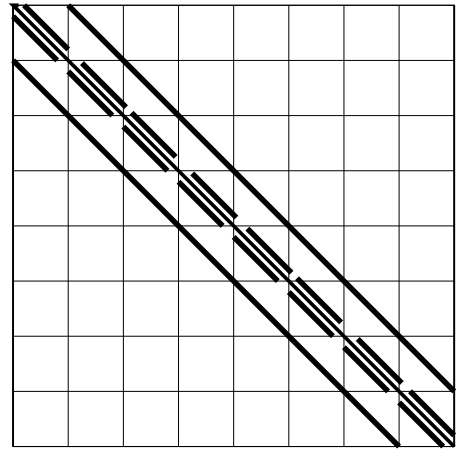
SkM

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.

Block-Structure of the Matrices



SkM



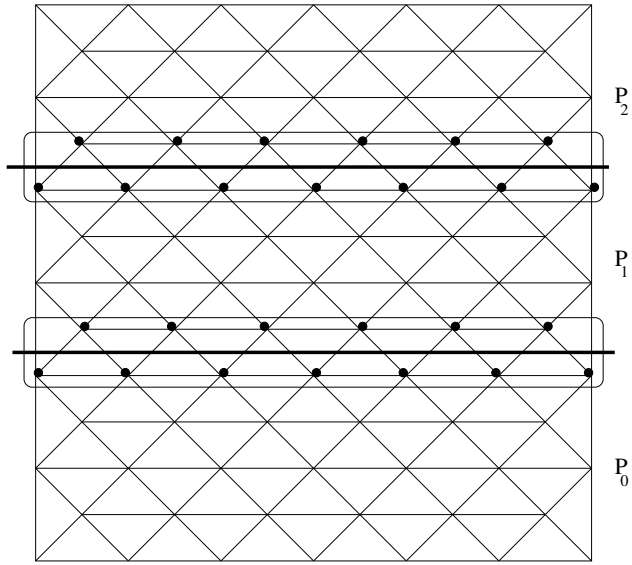
ReM

The bottleneck problem of the parallel implementation of MIC(0) algorithm is the solution of problems with triangle matrices $(X - L)$ and $(X - L)^T$.

Block-Structure of the Matrices

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: $N = n^2 + (n-1)^2$.

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

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

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.

Parallel Speedup

n	32	64	128	256	512	1024	1500
S(n,2)	1.21	1.68	1.96	1.85	1.92	2.03	2.02
S(n,3)	0.24	0.46	0.97	1.72	2.45	2.97	2.86
S(n,4)	0.22	0.46	1.11	1.97	2.88	3.76	3.95
S(n,5)	0.20	0.40	0.96	1.99	3.25	4.48	4.86
S(n,6)	0.18	0.39	1.03	1.99	3.55	5.23	5.73
S(n,7)	0.19	0.38	0.95	1.78	3.63	6.02	6.31
S(n,8)	0.19	0.39	1.00	2.28	3.97	6.37	6.76

Parallel Efficiency

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