Parallel normalized implicit preconditioned conjugate gradient methods for solving biharmonic equations on symmetric multiprocessor systems

George A. Gravvanis*, Konstantinos M. Giannoutakis

Department of Electrical and Computer Engineering, School of Engineering, Democritus University of Thrace, GR 67100 Xanthi, Greece

Abstract

A new class of inner–outer iterative procedures in conjunction with conjugate gradient-type schemes based on normalized approximate factorization procedures for solving sparse linear systems of irregular structure, which are derived from the finite element method of biharmonic equations in three space variables, is introduced. Normalized implicit preconditioned conjugate gradient-type methods are presented, for the efficient solution of linear sparse systems. Applications of the method on a three-dimensional biharmonic problem are discussed and numerical results are given. The parallel implementation on symmetric multiprocessor systems of the forward and backward substitution for the decomposition factors is also investigated.

Keywords: Biharmonic equations; Finite element method; Approximate factorization procedures; Preconditioning; Parallel computations

1. Introduction

Many engineering and scientific problems are described by sparse linear systems of algebraic equations derived from the finite element (FE) discretization of biharmonic equations, which occur in continuum mechanics in both linear elasticity and in fluid flow.

Methods for solving biharmonic equations on a rectangular region have been discussed by many researchers [1–8], and several iterative methods have been examined either considering the biharmonic equation as a 'coupled equation approach' (pair of Poisson equations) or by applying iterative schemes directly to the fourth-order equation.

A new class of inner–outer iterative procedures in conjunction with normalized implicit conjugate gradient-type schemes based on normalized approximate factorization procedures for solving sparse linear systems of irregular structure, which are derived from the finite element method of biharmonic equations in three space variables, is introduced. The parallel implementation of the dominant computational part, which is the forward and backward substitution of the decomposition factors, is also investigated for the efficient solution of sparse linear systems.

Application of the proposed method on a three-dimensional biharmonic problem is discussed and numerical results are given. The improvement of the proposed method is exhibited from its parallel execution results when implementing the forward and backward substitutions for the decomposition factors.

2. Approximate factorization procedures

Let us consider the following biharmonic equation in three space variables, viz.,

$$\Delta^2 u(x,y,z) = f(x,y,z), \quad (x,y,z) \in R \tag{1a}$$

$$u_0 = 0 \quad \text{and } \partial u(x,y,z)/\partial \eta = g_2(x,y,z), \ (x,y,z) \in \partial R \tag{1b}$$

where R is a bounded domain, ∂R is the boundary of R, and f is sufficiently smooth functions on R. Our approach is to consider the 'coupled equation approach', viz. $\nabla^4 = \nabla^2\nabla^2$, by solving

$$c\nabla^2 u = v \quad \text{and} \quad \nabla^2 v = cf \tag{2}$$

* Corresponding author. E-mail: ggravvan@ee.duth.gr

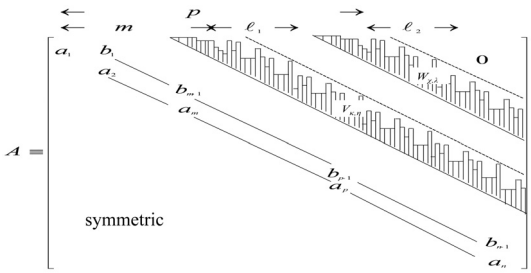
Computational Fluid and Solid Mechanics 2005
K.J. Bathe (Editor)

where two discrete Poisson equations, using a FE approximation scheme, must be solved, usually by an 'inner–outer' iteration method.

Let us consider the finite element linear system, i.e.

$$Au = s \tag{3}$$

where A is a sparse $(n \times n)$ matrix of special structure:



(4)

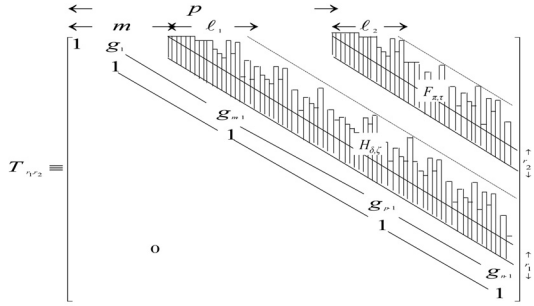
Let us now assume the normalized approximate factorization such that:

$$A \approx D_{r_1,r_2}\, T^t_{r_1,r_2} T_{r_1,r_2} D_{r_1,r_2}, r_1 \in [1,\cdots,m-1),$$
$$r_2 \in [1,\cdots,p-1) \tag{5}$$

where r_1, r_2 are the so-called 'fill-in' parameters, i.e. the number of outermost off-diagonal entries retained in the semi-bandwidths m and p respectively, D_{r_1,r_2} is a diagonal matrix, viz.,

$$D_{r_1,r_2} \equiv \mathrm{diag}\left\{ d_1,\cdots,d_{m-1} \vdots d_m,\cdots,d_{p-1} \vdots d_p,\cdots,d_n \right\} \tag{6}$$

and T_{r_1,r_2} is a sparse upper (with unit diagonal elements) triangular matrix of the same profile as the coefficient matrix A, i.e.



(7)

The elements of the decomposition factors were computed by the finite element approximate normalized factorization procedure (henceforth called the FEANOF-3D algorithm) [9]. The memory requirements of the FEANOF-3D algorithm is $\approx (r_1 + r_2 + 2\ell_1 + 2\ell_2 + 4)n$ words, while the computational work required is $\approx \mathrm{O}[(\mathrm{r}_1 + \ell_1)^2 + (\mathrm{r}_2 + \ell_2)^2]n$ multiplicative operations and n square roots [9].

The computational implementation of the factorization procedure requires the coefficient matrix A to be stored as diagonal, co-diagonals and the V, W submatrices (stored in a band-like scheme, i.e. only ℓ_1 and ℓ_2 vector spaces). In this case the submatrix $V = (v_{\kappa,\eta})$ is to be stored such that $V = (v_{\kappa,\eta})$, $\kappa \in [1,n-m+1]$, $\eta \in [1,\ell_2]$ denotes the elements of the κ-th row and $(\kappa + \eta + m - 2)$-th column of A in its usual arrangement, cf. Eq. (4). In a similar way, the submatrix $W = (w_{\chi,\lambda})$ is to be stored such that $w_{\chi,\lambda}$, $\chi \in [1,n-p+1]$, $\lambda \in [1,\ell_2]$ denotes the elements of the χ-th row and $(\chi + \lambda + p - 2)$-th column of A.

The factorization procedure requires the submatrix $H = (h_{i,j})$, $i \in [1,r_1 + \ell_1 - 1]$, $j \in [1,n-m+1]$ of the matrix T_{r_1,r_2} to be stored such that $h_{i,j}$ (for $i \le m-1$) denotes the elements in the i-th row and the $(m + j - 1)$-th column (if $j \le \ell_1$) or the elements in the $(i + j - \ell_1)$-th row and the $(m + j - 1)$-th column (if $j \rangle \ell_1$) while $h_{i,j}$ (for $i > m - 1$) denotes the elements in the i-th row and the $(i + j)$-th column (if $i + j \le m + \ell_1 - 1$) or the elements in the $(2i + j - m - \ell_1 + 1)$-th row and $(i + j)$-th column (if $i + j \rangle m + \ell_1 - 1$) of the coefficient matrix A in its usual arrangment. The submatrix $F = (f_{i,j})$, $i \in [1, r_2 + \ell_2 - 1], j \in [1, n - p + 1]$ of the matrix T_{r_1},r_2 can be stored such that $f_{i,j}$ (for $i \le p-1$) denotes the elements in the i-th row and the $(p + j - 1)$-th column (if $j \le \ell_2$) or the elements in the $(i + j - \ell_2)$-th row and the $(p + j - 1)$-th column (if $j \langle \ell_2$) while $f_{i,j}$ (for $i > p - 1$) denotes the elements in the i-th row and the $(i + j)$-th column (if $i + j \le p + \ell_2 - 1$) or the elements in the $(2i + j - p - \ell_2 + 1)$-th row and $(i + j)$-th column (if $i + j \rangle p + \ell_2 - 1$) of the coefficient matrix A in its usual arrangment.

3. Normalized implicit preconditioned conjugate gradient methods

The normalized approximate factorization procedures in conjunction with conjugate gradient-type schemes yield a class of efficient normalized implicit preconditioned schemes.

The normalized implicit preconditioned conjugate gradient square (NIPCGS) method can be expressed by the following compact algorithmic scheme:

Let u_0 be an arbitrary initial approximation to the solution vector u. Then,

$$\text{set} \qquad u_0 = 0 \text{ and } e_0 = 0 \tag{8}$$

$$\text{solve} \qquad \left(T^t_{r_1,r_2} T_{r_1,r_2}\right) D_{r_1,r_2} r_0 = D^{-1}_{r_1,r_2}(s - Au_0) \tag{9}$$

$$\text{set} \qquad \sigma_0 = r_0 \text{ and } p_0 = (\sigma_0, r_0) \tag{10}$$

Then, for $i = 0, 1, \ldots$, (until convergence) compute the vectors u_{i+1}, r_{i+1}, σ_{i+1} and the scalar quantities α_i, β_{i+1} as follows:

$$\text{form} \quad q_i = A\sigma_i, \ \text{solve}\left(T^t_{r_1,r_2} T_{r_1,r_2}\right) D_{r_1,r_2} t_i = D^{-1}_{r_1,r_2} q_i \tag{11}$$

$$\text{calculate} \quad \alpha_i = p_i/(\sigma_0, t_i) \tag{12}$$

$$\text{compute} \quad e_{i+1} = r_i + \beta_i e_i - \alpha_i t_i, \quad d_i = r_i + \beta_i e_i + e_{i+1} \tag{13}$$

$$\text{and} \quad u_{i+1} = u_i + \alpha_i d_i, \quad \text{form} \quad q_i = Ad_i \tag{14}$$

$$\text{solve} \quad \left(T^t_{r_1,r_2} T_{r_1,r_2}\right) D_{r_1,r_2} t_i = D^{-1}_{r_1,r_2} q_i \tag{15}$$

$$\text{compute} \quad r_{i+1} = r_i - \alpha_i t_i \tag{16}$$

$$\text{set} \quad p_{i+1} = (\sigma_0, r_{i+1}), \quad \text{evaluate} \quad \beta_{i+1} = p_{i+1}/p_i \tag{17}$$

$$\text{compute} \quad \sigma_{i+1} = r_{i+1} + 2\beta_{i+1} e_{i+1} + \beta^2_{i+1} \sigma_i \tag{18}$$

The normalized implicit preconditioned biconjugate conjugate gradient-STAB (NIPBICG-STAB) method, can be stated by the following algorithmic scheme:

Let u_0 be an arbitrary initial approximation to the solution vector u. Then,

$$\text{set} \quad u_0 = 0, \quad \text{compute} \ r_0 = s - Au_0 \tag{19}$$

$$\text{set} \quad r'_0 = r_0, \ \rho_0 = \alpha = \omega_0 = 1 \quad \text{and} \quad v_0 = p_0 = 0 \tag{20}$$

Then, for $i = 0, 1, \ldots$, (until convergence) compute the vectors u_i, r_i and the scalar quantities α, β, ω_i as follows:

$$\text{calculate} \quad \rho_i = \left(r'_0, r_{i-1}\right), \ \text{and} \ \beta = (\rho_i/\rho_{i-1})/(\alpha/\omega_{i-1}) \tag{21}$$

$$\text{compute} \quad p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1} v_{i-1}) \tag{22}$$

$$\text{solve} \quad \left(T^t_{r_1,r_2} T_{r_1,r_2}\right) D_{r_1,r_2} y_i = D^{-1}_{r_1,r_2} p_i \tag{23}$$

$$\text{form} \quad v_i = Ay_i, \quad \alpha = \rho_i/\left(r'_0, v_i\right), \text{and} \ x_i = r_{i-1} - \alpha v_i \tag{24}$$

$$\text{solve} \quad \left(T^t_{r_1,r_2} T_{r_1 r_2}\right) D_{r_1,r_2} z_i = D^{-1}_{r_1,r_2} x_i \tag{25}$$

$$\text{form} \quad t_i = Az_i, \ \text{solve}\left(T^t_{r_1,r_2} T_{r_1,r_2}\right) D_{r_1,r_2} t_i = D^{-1}_{r_1,r_2} t_i \tag{26}$$

$$\text{set} \quad \omega_i = (t_i, z_i)/(t_i, t_i) \tag{27}$$

$$\text{compute} \quad u_i = u_{i-1} + \alpha y_i + \omega_i z_i \quad \text{and} \ r_i = x_i - \omega_i t_i \tag{28}$$

The computational complexity of the NIPCGS method requires $\approx$ O[$(8\ell_1 + 8\ell_2 + 4r_1 + 4r_2 + 15)n$ mults + $8n$ adds]v operations, while for the NIPBICG-STAB method requires $\approx$ O[$(10\ell_1 + 10\ell_2 + 6r_1 + 6r_2 + 16)n$ mults + $6n$ adds]v operations, where v is the number of iterations required for the convergence to a predetermined tolerance level.

For the parallel exploitation and implementation of the dominant computational part of the methods described above, i.e. the forward and backward substitutions for the T_{r_1,r_2} and $T^t_{r_1,r_2}$ matrices, the simulation software tool environment of Multi-Pascal, [10], has been utilized, where a time unit of the simulated time is approximately equivalent to one microsecond of the real execution time on a general purpose multiprocessor.

The basic statement of the developing environment is the *forall* statement that is responsible for process creation and execution. Furthermore, the balancing of the workload is succeeded through an increase in granularity factor for each process (grouping statement), while the architecture platform is that of a shared memory system consisting of 512 processors. Let us now consider one of the backward substitution relations of the following form:

$$\begin{aligned} s_i &= s_i - g_i s_{i+1} - \sum_{j=m}^{m+\ell_1-1} h_{i,j-m+1}\, s_j \\ &- \sum_{j=m+\ell_1}^{i+m+\ell_1-2} h_{i-j+m+\ell_1-1,j-m+1} s_j \sum_{j=p}^{p+\ell_2-1} f_{i,j-p+1} s_j \\ &- \sum_{j=p+\ell_2}^{i+p+\ell_2-2} f_{i-j+p+\ell_2-1,j-p+1} s_j \end{aligned} \tag{29}$$

or equivalently rewritten as:

$$s_i = s_i - g_i \times s_{i+1} - z_1 - z_2 - z_3 - z_4 \tag{30}$$

Since z_1, z_2, z_3, z_4 can be computed in parallel, then the sequential part is minimized to a subtraction and a multiplication [4].

4. Numerical results

Let us consider the following three-dimensional-model problem:

$$\Delta^2 u(x,y,z) = 1, \quad (x,y,z) \in R \tag{31a}$$

subject to boundary conditions:

$$u(x,y,z) = 0, \text{and} \ \partial u(x,y,z)/\partial\eta = 0, \quad (x,y,z) \in \partial R \tag{31b}$$

where Δ is the Laplacian operator, R is the unity cube and ∂R is the boundary of the domain R. The domain $R \cup \partial R$ was covered by a non-overlapping triangular network, resulting in a hexagonal mesh. The NIPGCGS

Table 1
The convergence behavior of the NIPCGS and NIPBICG-STAB methods for various values of the parameters n, m, p, r_1 and r_2

Method	n	m	p	$r_1 = r_2 = 1$	$r_1 = r_2 = 2$	$r_1 = r_2 = 4$
NIPCGS	729	10	82	10	10	10
	2744	15	197	12	12	12
	6859	20	362	12	12	12
	13824	25	577	12	12	12
	24389	30	842	11	12	12
NIPBICG-STAB	729	10	82	10	10	10
	2744	15	197	11	11	11
	6859	20	362	11	11	11
	13824	25	577	11	11	11
	24389	30	842	11	11	11

and the NIPBICG-STAB methods were terminated when $\|u_{i+1} - u_i\|_\infty < 10^{-6}$.

Numerical results are presented in Table 1 for the NIPCGS and the NIPBICG-STAB method applied to problems (31a)–(31b) in the unit cube for several values of order n, semi-bandwidths m and p, and the fill-in parameters r_1, r_2.

For the parallelization of the forward–backward substitution of the submatrices T_{r_1,r_2} and $T^t_{r_1,r_2}$, two different cases have been considered [4]:

Case 1: The number of processors allocated was free and selected by the system, while the granularity factor used was the result of the square root function over the indices of each *forall* statement, which is nearly the best value for grouping according to the theoretical analysis.

Case 2: The granularity factor is the same as in the previous case, but the number of the processors used is controlled to achieve better efficiency.

Speedups and processor allocation are given in Table 2 and Table 3 for the Case 1 and Case 2 respectively. Additionally, speedups and processor allocation are presented in Fig. 1 and Fig. 2 for the Cases 1 and 2 respectively, while the efficiency and processor allocation is presented in Fig. 3 for the Case 2.

It is obvious that in the case of a restricted shared memory architecture concerning the number of processors available, the values of the relative speedup and efficiency are the best possible obtained.

Finally, it should be noted that, the restrictions imposed by the programming environment used did not allow us to fully explore cases with larger values of the order of the sparse linear system ($n > 1000$).

Table 2
Speedups and processor allocation for Case 1

n	m	p	Procs	r_1,r_2	Speedup
125	6	26	12	1	5.301
				2	5.436
				4	5.708
343	8	50	19	1	8.562
				2	8.683
				4	8.907
729	10	82	27	1	12.207
				2	12.299
				4	12.479

Table 3
Speedups and processor allocation for Case 2

n	m	p	Procs	r_1,r_2	Speedup
125	6	26	3	1	2.731
				2	2.749
				4	2.787
343	8	50	4	1	3.776
				2	3.787
				4	3.805
729	10	82	5	1	4.881
				2	4.894
				4	4.911

References

[1] Axelsson O. Notes on the numerical solution of biharmonic equation, J Inst Maths Applics 1973;11:213–226.

[2] Ehrlich LW. Solving the biharmonic equation as a coupled finite difference equations, SIAM J Numer Anal 1971;8:278–287.

[3] Gravvanis GA. Explicit preconditioning conjugate gradient schemes for solving biharmonic problems, Engineering Computations 2000;17:154–165.

[4] Gravvanis GA, Bekakos MP, Efremides OB. Parallel implicit preconditioned conjugate gradient methods for solving biharmonic equations, Proc of the 6th Hellenic–

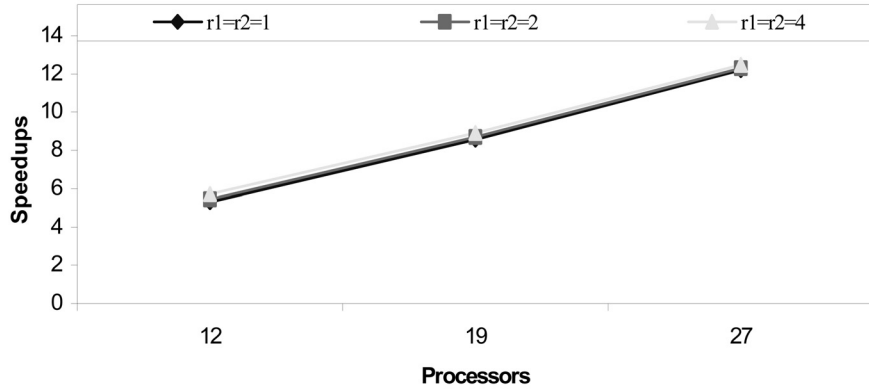


Fig. 1. Speedups and processor allocation for Case 1.

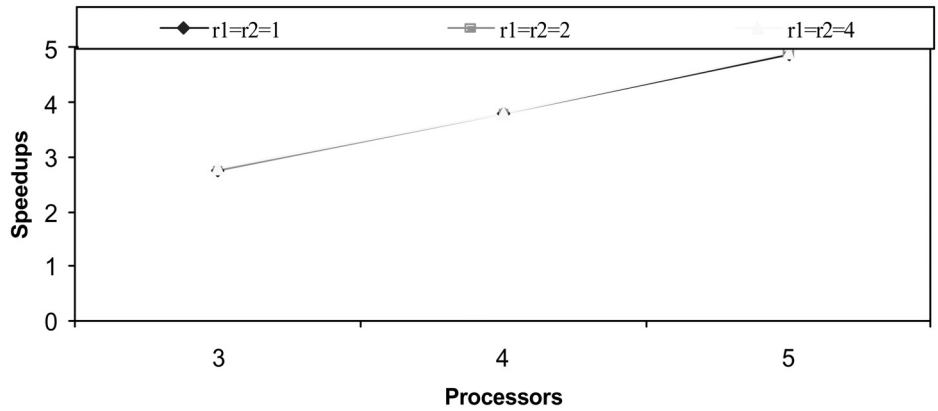


Fig. 2. Speedups and processor allocation for Case 2.

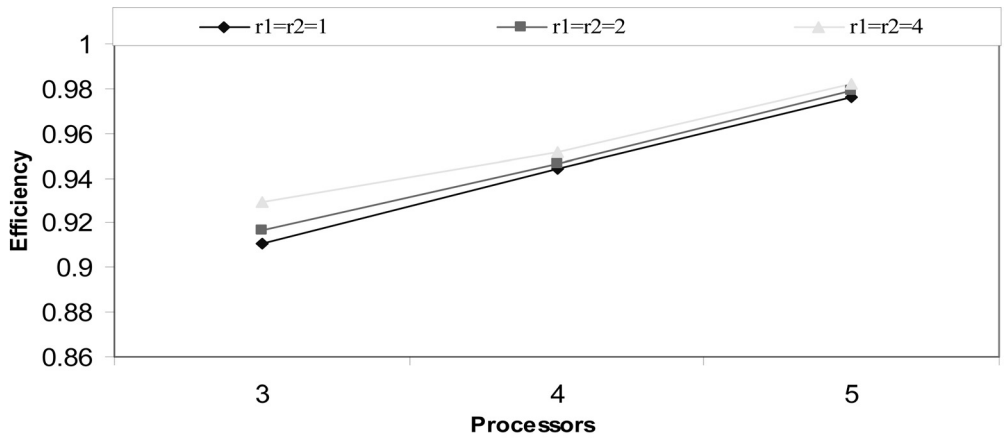


Fig. 3. Efficiency and processor allocation for Case 2.

European Conference on Computer Mathematics and its Applications, EA Lipitakis, editor, vol. 2, LEA Publishers, 2004, 683–694.

[5] Greenspan D, Schultz D. Fast finite difference solution of biharmonic boundary value problem, Comm ACM 1972;15:347–350.

[6] Nodera T, Takahashi H. Preconditioned conjugate gradient algorithm for solving biharmonic equations, Advances in Computer Methods for Partial Differential Equations IV, R Vichnevetsky, RS Stepleman, editors, IMACS, 1981.

[7] Smith J. The coupled equation approach to the numerical solution of the biharmonic equation by finite differences, SIAM J Numerical Analysis 1968;5:323–339.

[8] Yousif WS, Evans DJ. Explicit block iterative method for the solution of biharmonic equation, Numerical Methods for Partial Differential Equations 1993;9:1–12.

[9] Gravvanis GA, Giannoutakis KM. Normalized implicit preconditioned methods based on normalized finite element approximate factorization procedures, Proc of the 3rd MIT Conference on Computational Fluid and Solid Mechanics, KJ Bathe, editor. Elsevier, Oxford pp. 1115–1119.

[10] Lester BP. The Art of Parallel Programming, Prentice Hall, 1993.