Scientific Programming 13 (2005) 79–91
IOS Press


# Parallel preconditioned conjugate gradient square method based on normalized approximate inverses

George A. Gravvanis* and Konstantinos M. Giannoutakis
*Department of Electrical and Computer Engineering, School of Engineering, Democritus University of Thrace, GR 67100 Xanthi, Greece*
*E-mail: {ggravvan, kgiannou}@ee.duth.gr*

**Abstract** A new class of normalized explicit approximate inverse matrix techniques, based on normalized approximate factorization procedures, for solving sparse linear systems resulting from the finite difference discretization of partial differential equations in three space variables are introduced. A new parallel normalized explicit preconditioned conjugate gradient square method in conjunction with normalized approximate inverse matrix techniques for solving efficiently sparse linear systems on distributed memory systems, using Message Passing Interface (MPI) communication library, is also presented along with theoretical estimates on speedups and efficiency. The implementation and performance on a distributed memory MIMD machine, using Message Passing Interface (MPI) is also investigated. Applications on characteristic initial/boundary value problems in three dimensions are discussed and numerical results are given.

Keywords Finite difference method, sparse linear systems, normalized approximate inverse matrix techniques, preconditioning, parallel preconditioned conjugate gradient methods, programming, parallel computations, distributed computations

## 1. Introduction

The solution of sparse linear systems is of central importance to scientific and engineering computations and is the most time-consuming part, cf. [3,4,13]. The solution of sparse linear systems has been obtained by direct or iterative methods, cf. [2–6,9,13].

Let us consider the sparse linear system resulting from Finite Difference (FD) discretization of three dimensional boundary value problems, i.e.

$$Au = s, \tag{1}$$

where $A$ is a sparse, diagonally dominant, positive definite, symmetric $(n \times n)$ matrix of the following form:

*Corresponding author.

ISSN 1058-9244/05/$17.00 

$$A \equiv \begin{bmatrix} a_1 & b_1 & v_1 & & & w_1 & & \\ & a_2 & b_2 & & & & & \\ & & & b_{p-2} & & & & \\ & & & a_{p-1} & b_{p-1} & & w_{n-p+1} & \\ & & & & a_p & b_p & & \\ & & \text{symmetric} & & & & v_{n-m+1} & \\ & & & & & & & b_{n-1} \\ & & & & & & & a_n \end{bmatrix}, \tag{2}$$

and $m$, $p$ are the semi-bandwidths, while $u$ is the FD solution at the nodal points and $s$ is a vector, of which the components result from a combination of source terms and imposed boundary conditions.

For symmetric positive definite problems, the rate of convergence of the conjugate gradient method depends on the distribution of the eigenvalues of the coefficient matrix. Hence the preconditioned matrix will have a smaller spectral condition number, and the eigenvalues clustered around one, cf. [2,6,13]. The preconditioned form of the linear system Eq. (2) is

$$M\,A\,u = M\,s, \tag{3}$$

where $M$ is a preconditioner. The preconditioner $M$ has to satisfy the following conditions: (i) MA should have a "clustered" spectrum, (ii) $M$ can be efficiently computed in parallel and (iii) finally "$M\times$ vector" should be fast to compute in parallel, cf. [1,3,6–8,13]. The effectiveness of the explicit approximate inverse preconditioning methods is related to the fact that the approximate inverses exhibit a similar "fuzzy" structure as the coefficient matrix and are close approximant to the coefficient matrix, cf. [6–8].

The performance and applicability of the new algorithmic schemes for solving 3D elliptic and parabolic P.D.E's is discussed and numerical results are given.

## 2. Normalized optimized approximate inverses

In this section we present a class of normalized approximate inverses based on normalized factorization procedures.

Let us now assume the normalized approximate factorization, cf. [8], such that:

$$A \approx D_{r_1,r_2} T^t_{r_1,r_2} T_{r_1,r_2} D_{r_1,r_2}, \quad r_1 \in [1,\ldots,m-1), \quad r_2 \in [1,\ldots,p-1), \tag{4}$$

where $r_1$, $r_2$ are the "fill-in" parameters, $D_{r_1,r_2}$ is a diagonal matrix and $T_{r_1,r_2}$ is a sparse upper triangular matrix of the same profile as the coefficient matrix A.

Let $M^{\delta l}_{r_1,r_2} = (\mu_{i,j}), i \in [1,n], j \in [\max(1, i-\delta l+1), \min(n, i+\delta l-1)]$, be the normalized approximate inverse of the coefficient matrix A, cf. [8], i.e.

$$M^{\delta l}_{r_1,r_2} = \left(D_{r_1,r_2} T^t_{r_1,r_2} T_{r_1,r_2} D_{r_1,r_2}\right)^{-1} = D^{-1}_{r_1,r_2} \left(T^t_{r_1,r_2} T_{r_1,r_2}\right)^{-1} D^{-1}_{r_1,r_2} = D^{-1}_{r_1,r_2} \hat{M}^{\delta l}_{r_1,r_2} D^{-1}_{r_1,r_2}, \tag{5}$$

The elements of the normalized approximate inverse, by retaining $\delta l$ elements in the lower and upper part of the inverse, cf. [6–8], can be determined by solving recursively the following systems:

$$\hat{M}^{\delta l}_{r_1,r_2} T^t_{r_1,r_2} = \left(T_{r_1,r_2}\right)^{-1} \text{ and } T_{r_1,r_2} \hat{M}^{\delta l}_{r_1,r_2} = \left(T^t_{r_1,r_2}\right)^{-1}, \tag{6}$$

The **Nor**malized **O**ptimized **A**pproximate **I**nverse **M**atrix **–3D** algorithmic procedure (henceforth called the **NOROAIM-3D** algorithm) for computing the elements of the approximate inverse, using a "fish-bone" pattern, can be expressed by the following compact form:

**Let** $nmr1 = n - m + r_1$, $npr2 = n - p + r_2$.

**For** $i = n$ **to** $1$

**for** $j = i$ **to** $\max(1, i - \delta l + 1)$

**if** $j > nmr1$ **then**

**if** $i = j$ **then**

**if** $i = n$ **then**

$$\hat{\mu}_{1,1} = 1 \tag{7}$$

**else**

$$\hat{\mu}_{n-i+1,1} = 1 - e_j \cdot \hat{\mu}_{n-j,\delta l+1} \tag{8}$$

**else**

$$\hat{\mu}_{n-i+1,i-j+1} = -e_j \cdot \hat{\mu}_{n-i+1,i-j} \tag{9}$$

**else**

**if** $j > npr2$ **and** $j \leqslant nmr1$ **then**

**if** $i = j$ **then**

$$\hat{\mu}_{n-i+1,1} = 1 - e_j \cdot \hat{\mu}_{n-j,\delta l+1} - \sum_{k=0}^{nmr1-j} h_{r_1-k,j+k+1-r_1} \cdot \hat{\mu}_{x,y} \tag{10}$$

call **mw** $(n, \delta l, i, m + j + k - r_1, x, y)$

**else**

$$\hat{\mu}_{n-i+1,i-j+1} = -e_j \cdot \hat{\mu}_{n-i+1,i-j} - \sum_{k=0}^{nmr1-j} h_{r_1-k,j+k+1-r_1} \cdot \hat{\mu}_{x,y} \tag{11}$$

call **mw** $(n, \delta l, i, m + j + k - r_1, x, y)$

**else**

**if** $j \geqslant r_1 + 1$ **and** $j \leqslant npr2$ **then**

**if** $i = j$ **then**

**if** $j \geqslant r_2 + 1$ **then**

$$\hat{\mu}_{n-i+1,1} = 1 - e_j \cdot \hat{\mu}_{n-j,\delta l+1} - \sum_{k=0}^{npr2-j} f_{r_2-k,j+k+1-r_2} \cdot \hat{\mu}_{x_1,y_1} - \sum_{k=0}^{nmr1-j} h_{r_1-k,j+k+1-r_1} \cdot \hat{\mu}_{x_2,y_2} \tag{12}$$

call **mw** $(n, \delta l, i, j + k + p - r_2, x_1, y_1)$

call **mw** $(n, \delta l, i, j + k + m - r_1, x_2, y_2)$

**else**

$$\hat{\mu}_{n-i+1,1} = 1 - e_j \cdot \hat{\mu}_{n-j,\delta l+1} - \sum_{k=1}^{j} f_{j-k+1,k} \cdot \hat{\mu}_{x_1,y_1} - \sum_{k=0}^{nmr1-j} h_{r_1-k,j+k+1-r_1} \cdot \hat{\mu}_{x_2,y_2} \tag{13}$$

call **mw** $(n, \delta l, i, k+p-1, x_1, y_1)$

call **mw** $(n, \delta l, i, j+k+m-r_1, x_2, y_2)$

**else**

**if** $j \geqslant r_2 + 1$ **then**

$$\hat{\mu}_{n-i+1,i-j+1} = -e_j \cdot \hat{\mu}_{n-i+1,i-j} - \sum_{k=0}^{npr2-j} f_{r_2-k,j+k+1-r_2} \cdot \hat{\mu}_{x_1,y_1} - \sum_{k=0}^{nmr1-j} h_{r_1-k,j+k+1-r_1} \cdot \hat{\mu}_{x_2,y_2} \tag{14}$$

call **mw** $(n, \delta l, i, j+k+p-r_2, x_1, y_1)$

call **mw** $(n, \delta l, i, j+k+m-r_1, x_2, y_2)$

**else**

$$\hat{\mu}_{n-i+1,i-j+1} = -e_j \cdot \hat{\mu}_{n-i+1,i-j} - \sum_{k=1}^{j} f_{j-k+1,k} \cdot \hat{\mu}_{x_1,y_1} - \sum_{k=0}^{nmr1-j} h_{r_1-k,j+k+1-r_1} \cdot \hat{\mu}_{x_2,y_2} \tag{15}$$

call **mw** $(n, \delta l, i, k+p-1, x_1, y_1)$

call **mw** $(n, \delta l, i, j+k+m-r_1, x_2, y_2)$

**else**

**if** $i = j$ **then**

**if** $j \geqslant r_2 + 1$ **then**

$$\hat{\mu}_{n-i+1,1} = 1 - e_j \cdot \hat{\mu}_{n-j,\delta l+1} - \sum_{k=0}^{npr2-j} f_{r_2-k,j+k+1-r_2} \cdot \hat{\mu}_{x_1,y_1} - \sum_{k=1}^{j} h_{j-k+1,k} \cdot \hat{\mu}_{x_2,y_2} \tag{16}$$

call **mw** $(n, \delta l, i, j+k+p-r_2, x_1, y_1)$

call **mw** $(n, \delta l, i, k+m-1, x_2, y_2)$

**else**

$$\hat{\mu}_{n-i+1,1} = 1 - e_j \cdot \hat{\mu}_{n-j,\delta l+1} - \sum_{k=1}^{j} f_{j-k+1,k} \cdot \hat{\mu}_{x_1,y_1} - \sum_{k=1}^{j} h_{j-k+1,k} \cdot \hat{\mu}_{x_2,y_2} \tag{17}$$

call **mw** $(n, \delta l, i, k+p-1, x_1, y_1)$

call **mw** $(n, \delta l, i, k+m-1, x_2, y_2)$

**else**

**if** $j \geqslant r_2 + 1$ **then**

$$\hat{\mu}_{n-i+1,i-j+1} = -e_j \cdot \hat{\mu}_{n-i+1,i-j} - \sum_{k=0}^{npr2-j} f_{r_2-k,j+k+1-r_2} \cdot \hat{\mu}_{x_1,y_1} - \sum_{k=1}^{j} h_{j-k+1,k} \cdot \hat{\mu}_{x_2,y_2} \tag{18}$$

call **mw** $(n, \delta l, i, j+k+p-r_2, x_1, y_1)$

call **mw** $(n, \delta l, i, k+m-1, x_2, y_2)$

**else**

$$\hat{\mu}_{n-i+1,i-j+1} = -e_j \cdot \hat{\mu}_{n-i+1,i-j} - \sum_{k=1}^{j} f_{j-k+1,k} \cdot \hat{\mu}_{x_1,y_1} - \sum_{k=1}^{j} h_{j-k+1,k} \cdot \hat{\mu}_{x_2,y_2} \tag{19}$$

call **mw** $(n, \delta l, i, k+p-1, x_1, y_1)$

call **mw** $(n, \delta l, i, k+m-1, x_2, y_2)$

**for** $j = i-1$ **to** $\max(1, i-\delta l+1)$

$\hat{\mu}_{n-i+1,\delta l+i-j} = \hat{\mu}_{n-i+1,i-j+1}$

The procedure $mw(n, \delta l, s, q, x, y)$, cf. [6], can then be described as follows:
**procedure mw**$(n, \delta l, s, q, x, y)$
**if** $s \geqslant q$ **then**

$$x = n+1-s; y = s-q+1 \tag{20}$$

**else**

$$x = n+1-q; y = \delta l+q-s. \tag{21}$$

The computational work of the **NOROAIM-3D** algorithm is $O[n\delta l(r_1+r_2+1)]$ multiplicative operations, while the storage of the approximate inverse is only $n \times (2\delta l-1)$-vector spaces, using the optimized storage scheme as described by the above given procedure $mw(n, \delta l, s, q, x, y)$, cf. [6]. This optimized form is particularly effective for solving "narrow-banded" sparse systems of very large order, i.e. $\delta e \ll n/2$, cf. [6–8]. The parallel construction of similar approximate inverses has been studied and implemented in [7], and is under further investigation.

It should be noted that this class of normalized approximate inverse includes various families of approximate inverses according to the requirements of accuracy, storage and computational work, as can be seen by the following diagrammatic relation:

$$\begin{array}{ccc} & \text{class I} & \text{class II} \\ A^{-1} \equiv D^{-1}\hat{M}D^{-1} \leftarrow D^{-1}_{r_1,r_2}\hat{\tilde{M}}^{\delta l}_{r_1=m-1,r_2=p-1} D^{-1}_{r_1,r_2} & \leftarrow D^{-1}_{r_1,r_2}\hat{M}^{\delta l}_{r_1=m-1,r_2=p-1} D^{-1}_{r_1,r_2} & \\ & \text{class III} & \text{class IV} \\ & \leftarrow D^{-1}_{r_1,r_2}\hat{M}^{\delta l}_{r_1,r_2}D^{-1}_{r_1,r_2} & \leftarrow D^{-2}_{r_1,r_2} \end{array} \tag{22}$$

where the entries of the class I inverse have been retained after the computation of the exact inverse ($r_1 = m-1$, $r_2 = p-1$), while the entries of the class II inverse have been computed and retained during the computational procedure of the (approximate) inverse ($r_1 = m-1$, $r_2 = p-1$). The entries of the class III inverse have been retained after the computation of the approximate inverse ($r_1 \leqslant m-1$, $r_2 \leqslant p-1$). Hence an approximate inverse is derived in which both the sparseness of the coefficient matrix is relatively retained and storage requirements are substantially reduced. The class IV of approximate inverse retains only the diagonal elements, i.e. $\delta l = 1$ hence $\hat{M}^{\delta 1}_{r_1,r_2} \equiv I$, resulting in a fast inverse algorithm.

It is known that the larger in magnitude elements of the inverse matrices, in almost every case, are clustered around the diagonals at distances $\rho_1 m$ and $\rho_2 p$ (with $\rho_1 = 1, 2, \ldots, m-1$ and $\rho_2 = 1, 2, \ldots, p-1$) from the main diagonal in a "recurring wave"-like pattern, cf. [6–8]. It is reasonable to assume that the value of the retention parameter $\delta l$ can be chosen as multiples of $m$ and $p$.

It should be noted that, if $w_i = 0$, cf. Eq. (2), then the algorithm reduces to one for solving sparse linear systems, which are encountered in solving 2D boundary value problems by the finite difference method, cf. [5]. When $w_i = 0$ and $v_i = 0$, cf. Eq. (2), then the algorithm is reduced to one for solving tri-diagonal linear systems, which are encountered in solving two-point boundary value problems.

## 3. Parallel normalized explicit preconditioned conjugate gradient methods

In this section we present a class of parallel normalized explicit preconditioned conjugate gradient schemes, based on the derived normalized approximate inverse, for solving sparse linear systems on distributed MIMD parallel systems.

The **N**ormalized **E**xplicit **P**reconditioned **C**onjugate **G**radient (**NEPCG**), **N**ormalized **E**xplicit **P**reconditioned **C**onjugate **G**radient **S**quare (**NEPCGS**) and **N**ormalized **E**xplicit **P**reconditioned **Bi**-conjugate **C**onjugate **G**radient – **STAB** (**NEPBICG-STAB**) methods for solving sparse linear systems have been presented in [8].

Let us now consider that the coefficient matrix A, the normalized optimized approximate inverse $D_{r_1,r_2}^{-1}\hat{M}_{r_1,r_2}^{\delta l}D_{r_1,r_2}^{-1}$ and all the vectors are distributed in a block-row distribution. In this distribution we partition the matrices and vectors into blocks of consecutive rows, and assign a panel of elements to each process. The processors operate with local data, and need synchronization points before computations of inner products and matrix $\times$ vector operations.

Let local n be the number of rows allocated to each processor (i.e. local n:=n/no proc). Then, the **P**arallel form of the **N**ormalized **E**xplicit **P**reconditioned **C**onjugate **G**radient **S**quare (**PNEPCGS**) method can be expressed by the following algorithmic procedure:

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

**for** $j =$ (myrank*local $\mathrm{n}+1$) **to** (myrank* local $\mathrm{n}+$ local n) (myrank is the rank of each process)

$$(u_0)_j = 0, \quad (e_0)_j = 0, \quad (t_0)_j = s_j - A\,(u_0)_j \tag{23}$$

**if** $\delta l = 1$ **then**

**for** $j =$ (myrank*local $\mathrm{n}+1$) **to** (myrank* local $\mathrm{n}+$ local n)

$$(r_0)_j = (t_0)_j \Big/ \left(d^2\right)_j \tag{24}$$

**else**

**gather** distributed $t_0$ onto each process

**for** $j =$ (myrank*local $\mathrm{n}+1$) **to** (myrank* local $\mathrm{n}+$ local n)

$$(r_0)_j = \left( \sum_{k=\max(1,j-\delta l+1)}^{\min(n,j+\delta l-1)} \hat{\mu}_{j,k}\,(t_0)_k\,/d_k \right) \Big/ (d)_j \tag{25}$$

**for** $j =$ (myrank*local $\mathrm{n}+1$) **to** (myrank* local $\mathrm{n}+$ local n)

$$(\sigma_0)_j = (r_0)_j \tag{26}$$

$$p_0 = \left( (\sigma_0)_j, (r_0)_j \right) \tag{27}$$

**Gather** local $p_0$ to root process, compute their sum and **scatter** it to all processes

Then, for $i = 0, 1, \ldots,$ (until convergence) compute in parallel the vectors $u_{i+1}, r_{i+1}, \sigma_{i+1}$ and the scalar quantities $\alpha_i, \beta_{i+1}$ as follows:

**gather** distributed $\sigma_i$ onto each process

**for** $j =$ (myrank*local $\mathrm{n}+1$) **to** (myrank* local $\mathrm{n}+$ local n)

$$(q_i)_j = A\,(\sigma_i)_j \tag{28}$$

**if** $\delta l = 1$ **then**

**for** $j =$ (myrank*local $\mathrm{n}+1$) **to** (myrank* local $\mathrm{n}+$ local n)

$$(t_i)_j = (q_i)_j \Big/ \left(d^2\right)_j \tag{29}$$

**else**

**gather** distributed $q_i$ onto each process

**for** $j =$ (myrank*local $\mathrm{n}+1$) **to** (myrank* local $\mathrm{n}+$ local n)

$$(t_i)_j = \left( \sum_{k=\max(1,j-\delta l+1)}^{\min(n,j+\delta l-1)} \hat{\mu}_{j,k} (q_i)_k / d_k \right) \Big/ (d)_j \tag{30}$$

**for** $j =$ (myrank*local n $+ 1$) **to** (myrank* local n $+$ local n)

$$p_i = \left( (\sigma_0)_j , (t_i)_j \right) \tag{31}$$

**Gather** local $p_0$ to root process, compute their sum and **scatter** it to all processes

$$\alpha_i = p_i / w_i \tag{32}$$

**for** $j =$ (myrank*local n $+ 1$) **to** (myrank* local n $+$ local n)

$$(e_{i+1})_j = (r_i)_j + \beta_i (e_i)_j - \alpha_i (t_i)_j \tag{33}$$

**for** $j =$ (myrank*local n $+ 1$) **to** (myrank* local n $+$ local n)

$$(y_i)_j = (r_i)_j + \beta_i (e_i)_j + (e_{i+1})_j \tag{34}$$

**for** $j =$ (myrank*local n $+ 1$) **to** (myrank* local n $+$ local n)

$$(u_{i+1})_j = (u_i)_j + \alpha_i (y_i)_j \tag{35}$$

**gather** distributed $\sigma_i$ onto each process
**for** $j =$ (myrank*local n $+ 1$) **to** (myrank* local n $+$ local n)

$$(q_i)_j = A (y_i)_j \tag{36}$$

**if** $\delta l = 1$ **then**

**for** $j =$ (myrank*local n $+ 1$) **to** (myrank* local n $+$ local n)

$$(t_i)_j = (q_i)_j \Big/ \left(d^2\right)_j \tag{37}$$

**else**

**gather** distributed $q_i$ onto each process
**for** $j =$ (myrank*local n $+ 1$) **to** (myrank* local n $+$ local n)

$$(t_i)_j = \left( \sum_{k=\max(1,j-\delta l+1)}^{\min(n,j+\delta l-1)} \hat{\mu}_{j,k} (q_i)_k / d_k \right) \Big/ (d)_j \tag{38}$$

**for** $j =$ (myrank*local n $+ 1$) **to** (myrank* local n $+$ local n)

$$(r_{i+1})_j = (r_i)_j - \alpha_i (t_i)_j \tag{39}$$

**for** $j =$ (myrank*local n $+ 1$) **to** (myrank* local n $+$ local n)

$$p_{i+1} = \left( (\sigma_0)_j , (r_{i+1})_j \right) \tag{40}$$

**gather** local $p_{i+1}$ to root process, compute their sum and **scatter** it to all processes

$$\beta_{i+1} = p_{i+1} / p_i \tag{41}$$

**for** $j =$ (myrank*local n $+ 1$) **to** (myrank* local n $+$ local n)

$$(\sigma_{i+1})_j = (r_{i+1})_j + 2\beta_{i+1} (e_{i+1})_j + \beta_{i+1}^2 (\sigma_i)_j \tag{42}$$

The computational complexity of the **PNEPCGS** method is $\approx O[(4\delta l + 27)$ local n mults $+$ 8 local n adds$]\nu$ operations, while the total communication cost, using the butterfly technique for collective communications, is $\approx O[5 \cdot t_s \cdot \log(\text{no proc}) + 4 \cdot \text{local n} \cdot (\text{no proc} - 1) t_w]\nu$, where $\nu$ denotes the number of iterations required for the convergence to a certain level of accuracy, $t_s$ the message latency, and $t_w$ the time necessary for a word to be sent.

Thus, the speedup and efficiency of the **PNEPCGS** method can be defined as follows:

$$S_p = \frac{1}{\frac{1}{\text{no proc}} + \underset{O(\delta l) n t_m}{5 t_s \log(\text{no proc})} + \underset{O(\delta l)}{4(\text{no proc}-1) t_w}}, \tag{43}$$

and

$$E_p = \frac{1}{1 + \underset{O(\delta l) n t_m}{5 t_s \text{no proc} \log(\text{no proc})} + \underset{O(\delta l)}{4(\text{no proc}-1) t_w}}, \tag{44}$$

where $t_m$ denotes the computational time of one multiplication. Hence, for $\delta l \to \infty$, $S_p \to$no proc and $E_p \to 1$, that is optimum.

The total number of arithmetic operations required for the parallel computation of the solution $u_\nu$ is:

$$O\left[n^{1/2} \delta l^{1/2} \text{local n} \log \varepsilon^{-1}\right], \tag{45}$$

while the total communication cost is:

$$O\left[\underset{\delta l^{1/2}}{n^{1/2} \log \varepsilon^{-1}} \left(t_s \log(\text{no proc}) + \text{local n no proc} \, t_w\right)\right]. \tag{46}$$

The implementation of the two most dominating operations of the **PNEPCGS** iterative method, i.e. multiplication of the normalized optimized approximate inverse with a vector, cf. Eqs (29)–(30), and the inner product of two vectors, cf. Eq. (31), is given using the MPI communication library. For communication operations, the collective communication routines MPI Allreduce and MPI Allgather were used for sending and receiving data among distributed processes, cf. [11,12].

/* perform the multiplication of the normalized optimized approximate inverse with a vector, i.e. $t = D^{-1}_{r_1,r_2} \hat{M}^{\delta l}_{r_1,r_2} D^{-1}_{r_1,r_2} \times$ q */

Notation: dl is the "retention" parameter $\delta l$

```
if (dl==1)
    for (i=myrank*local n + 1; i<=myrank*local n+local n; i++)
        t[i] = q[i]/(d[i] * d[i]);
else
{ /* shift operation for the MPI Allgather routine */
    for (i = 1; i <= local n; i++)
        temp[i-1]=q[myrank*local n+i];
    MPI Allgather(temp,local n,MPI DOUBLE,q,local n, MPI DOUBLE,MPI COMM WORLD);
    for (i = n; i >= 1; i--)
        q[i] = q[i - 1];
    for (i=myrank*local n + 1;i<=myrank*local n+local n;i++) {
        s1 = 0;
        for (j= max(1,i-dl+1); j<=i;j++) /*upper part*/
```

```
            s1=s1+m[n+1−i][i+1−j]*(q[j]/d[j]);
        for (j=i+1; j<= min(n,i+dl−1); j++) /*lower part*/
            s1=s1+m[n+1−j][dl+j−i]*(q[j]/d[j]);
        t[i]=s1/d[i]; }
    }
/* perform the inner product of two vectors p=(sigma,t) */
    for(i=myrank*local n + 1; i<=myrank*local n+local n; i++)
        temp1=temp1+sigma[i]*t[i];
    MPI Allreduce(&temp1,&p,1,MPI DOUBLE,MPI SUM, MPI COMM WORLD);
```

It should be noted that the elements of the vectors of each process are stored in their original sequential position. Thus, a shift operation is required for the MPI Allgather routine in order to place the elements in the beginning of each vector.

## 4. Numerical results

In this section we examine the applicability and effectiveness of the proposed schemes for solving characteristic three dimensional boundary value and initial value problems.

**Model Problem I**: Let us consider a 3D-boundary value problem with Dirichlet boundary conditions:

$$u_{xx} + u_{yy} + u_{zz} = F, (x, y, z) \in R \tag{47}$$

$$u(x, y, z) = 0, (x, y, z) \in \partial R, \tag{47a}$$

where $R$ is the unit cube and $\partial R$ denotes the boundary of $R$. The right hand side vector of the system Eq. (1) was computed as the product of the matrix A by the solution vector, with its components equal to unity. The iterative process was terminated when $\|r_i\|_\infty \langle 10^{-5}$.

Numerical results for the new proposed parallel schemes were obtain using a cluster of ten Eq. (10) AMD Athlon 1900 processors running at 1.6 GHz, connected in an 100 Mbps Ethernet network, using MPI.

The speedups and the number of iterations of the **PNEPCGS** method for several values of the "retention" parameter $\delta$l with $n = 8000$, $m = 21$, $p = 401$ and $r_1 = r_2 = 2$ are given in Table 1. In Figs 1, 2 and 3 the speedups and processors allocated for several values of the "retention" parameter $\delta l$, the speedups versus the "retention" parameter $\delta l$ for several numbers of processors and the parallel efficiency for several values of the "retention" parameter $\delta l$ are presented respectively for the **PNEPCGS** method with $n = 8000$, $m = 21$, $p = 401$ and $r_1 = r_2 = 2$. In Fig. 4 the performance evaluation measurements of the **PNEPCGS** method are given with $n = 8000$, $m = 21$, $p = 401$ and $r_1 = r_2 = 2$.

It is observed by the experimental results that the communication cost is responsible for the performance of the **PNEPCGS** method for small values of parameter $\delta l$, in contrast with large values of $\delta l$ where speedups and efficiency tend to become optimum.

**Model Problem II**: Let us consider the following Parabolic P.D.E. in three space variables:

Table 1
Speedups and processors allocated of the **PNEPCGS** method for several values of $\delta l$, with $n = 8000$, $m = 21$ and $p = 401$

| "Retention" parameter | Speedups Number of processors | | | | Number of iterations |
|---|---|---|---|---|---|
| | 2 | 4 | 8 | 10 | |
| $\delta l = 1$ | 0.397 | 0.326 | 0.289 | 0.275 | 44 |
| $\delta l = 2$ | 0.491 | 0.427 | 0.394 | 0.364 | 29 |
| $\delta l = m$ | 1.025 | 1.127 | 1.145 | 1.108 | 27 |
| $\delta l = 2m$ | 1.294 | 1.621 | 1.815 | 1.764 | 22 |
| $\delta l = p$ | 1.778 | 3.150 | 5.317 | 5.830 | 15 |
| $\delta l = 2p$ | 1.816 | 3.264 | 5.965 | 6.803 | 12 |
| $\delta l = 3p$ | 1.854 | 3.251 | 6.124 | 7.059 | 14 |
| $\delta l = 4p$ | 1.979 | 3.284 | 6.344 | 7.471 | 14 |
| $\delta l = 6p$ | 1.987 | 3.323 | 6.686 | 8.286 | 14 |

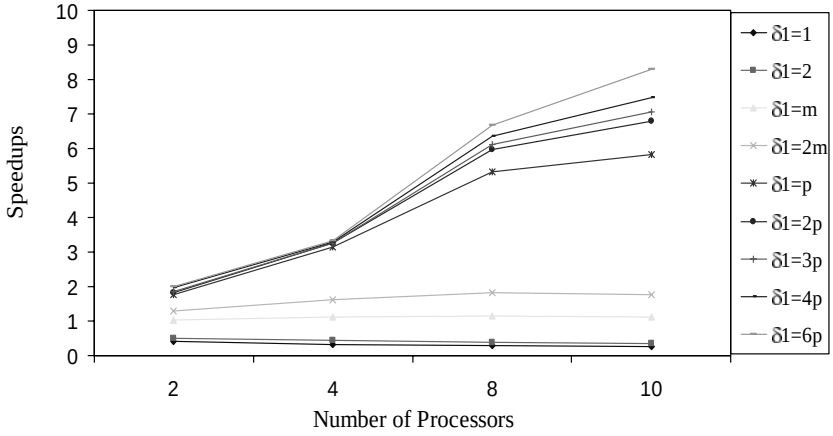


Fig. 1. Speedups and processors allocated of the **PNEPCGS** method for several values of $\delta l$, with $n = 8000$, $m = 21$ and $p = 401$.

$$\frac{\partial u}{\partial t} - \Delta u(x, y, z) = 0, (x, y, z) \in R, t > 0, \tag{48}$$

with initial conditions:

$$u(x, y, z, 0) = g(x, y, z), 0 \leqslant x, y, z \leqslant 1, \tag{48.a}$$

and boundary conditions:

$$u(x, y, z, t) = 0, t > 0, \tag{48.b}$$

where $R$ is the unit cube, $g(x, y, z)$ is given sufficiently smooth function and $\Delta$ is the Laplace operator. Let us assume that a network of mesh-sizes $h_x$, $h_y$, $h_z$ and $\Delta t$ in the $X$, $Y$, $Z$ and $T$ directions respectively is superimposed over $R$.

Then, the time-dependent functions were approximated by certain schemes, cf. [6,7], which can be written in the following parametric form:

$$K\frac{c^{(k+1)} - c^{(k)}}{\Delta t} + \frac{1}{h^2}F(\theta c^{(k+1)} - (1-\theta)c^{(k)} = \theta b^{(k+1)} + (1-\theta)b^{(k)},$$

$$k = 1, \ldots, n, \theta \in [0, 1], \text{ and } Kc^{(0)} = e, \tag{49}$$

where the value of the parameter $\theta$ denotes the various "time" – schemes.

In the case of $\theta = 1$ we have the "time-implicit" scheme, using backward time differences. This scheme is unconditionally valid, i.e. stable and convergent, and independent of the mesh-ratio. A disadvantage of this scheme

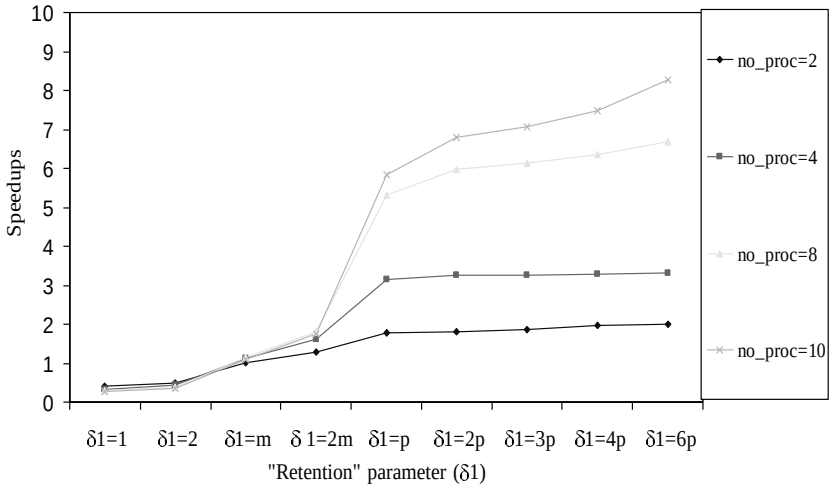


Fig. 2. Speedups versus the "retention" parameter $\delta l$ of the **PNEPCGS** method for several numbers of processors, with $n = 8000$, $m = 21$ and $p = 401$.

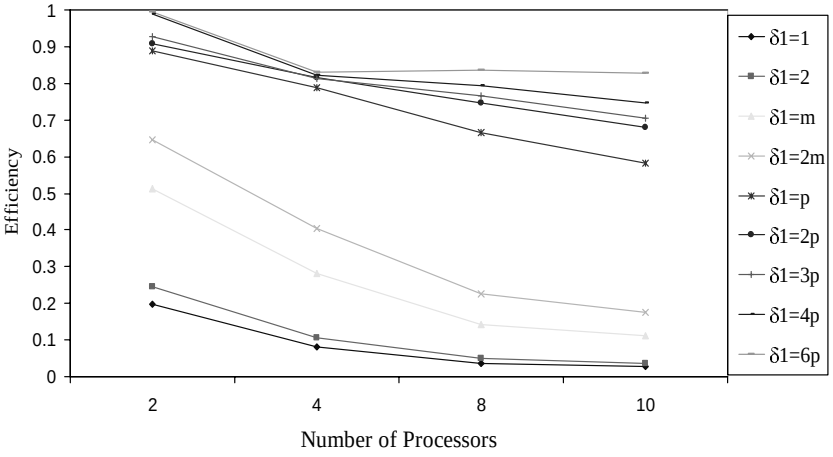


Fig. 3. Parallel efficiency of the **PNEPCGS** method for several values of $\delta l$, with $n = 8000$, $m = 21$ and $p = 401$.

is the error of $O(\Delta t)$ for the time partial derivative implying that the time step $\Delta t$ must be smaller than the mesh size h. While, when $\theta = 1/2$ we have the "time-implicit" Crank-Nickolson scheme, using central time differences, which is unconditionally valid and independent of the mesh-ratio.

Hence, the resulting sparse linear system is of the form Eq. (1). The fill-in parameters were chosen to be $r_1 = r_2 = 2$. The initial guess was $u^{(0)} = 0.05$. The termination criterion for the inner iteration of the NEPCG-type methods was $\|r_i\|_\infty \langle 10^{-5}$, where $r_i$ is the recursive residual. The criterion for the termination of the "steady-state" solution (outer iteration) was $\max_j \left| \left( u_j^{(k+1)} - u_j^{(k)} \right) / \left( u_j^{(k+1)} \right) \right| \langle 10^{-5}$

Numerical results using the "time-implicit" scheme (backward differences, $\theta = 1$) in conjunction with the **NEPCGS** and **NEPBICG-STAB** method, cf. [8], for several values of the "retention" parameter $\delta l$ of the approximate inverse and the time-step $\Delta t$ are given in Table 2.

Finally, the parallel normalized explicit approximate inverse preconditioning method can be efficiently used for solving three dimensional highly non-linear Elliptic and Parabolic P.D.E.'s.

Table 2
The convergence behavior of the **NEPCGS** and **NEPBICG-STAB** method with $r_1 = 2$, $r_2 = 2$ and $\theta = 1$ for various values of the parameter $\delta$l and the time step $\Delta$t

| Method | n | m | p | $\Delta$t | no of outer iter for s.s.s | $\delta l = 1$ | $\delta l = 2$ | $\delta l = m$ | $\delta l = p$ | $\delta l = 2p$ |
|---|---|---|---|---|---|---|---|---|---|---|
| NEPCGS | 343 | 8 | 50 | 0.00100 | 26 | 93 | 79 | 62 | 50 | 42 |
| | | | | 0.00050 | 35 | 103 | 94 | 78 | 62 | 47 |
| | | | | 0.00010 | 70 | 131 | 118 | 114 | 70 | 70 |
| | | | | 0.00005 | 94 | 143 | 128 | 94 | 94 | 94 |
| | 729 | 10 | 82 | 0.00100 | 27 | 118 | 98 | 77 | 59 | 50 |
| | | | | 0.00050 | 35 | 129 | 111* | 84 | 66 | 58 |
| | | | | 0.00010 | 70 | 157** | 136 | 129 | 70 | 70 |
| | | | | 0.00005 | 95 | 177 | 153 | 133*** | 94 | 94 |
| NEPBICG -STAB | 343 | 8 | 50 | 0.00100 | 26 | 62 | 62 | 57 | 47 | 42 |
| | | | | 0.00050 | 35 | 70 | 67 | 63 | 58 | 43 |
| | | | | 0.00010 | 70 | 96 | 88 | 84 | 70 | 70 |
| | | | | 0.00005 | 94 | 109 | 94 | 94 | 94 | 94 |
| | 729 | 10 | 82 | 0.00100 | 27 | 80 | 74 | 66 | 50 | 46 |
| | | | | 0.00050 | 35 | 89 | 78 | 75 | 61 | 55 |
| | | | | 0.00010 | 70 | 101 | 100 | 101 | 70 | 70 |
| | | | | 0.00005 | 94 | 124 | 112 | 94 | 94 | 94 |

*The number of outer iterations was 36 iterations.
**The number of outer iterations was 71 iterations.
***The number of outer iterations was 94 iterations.

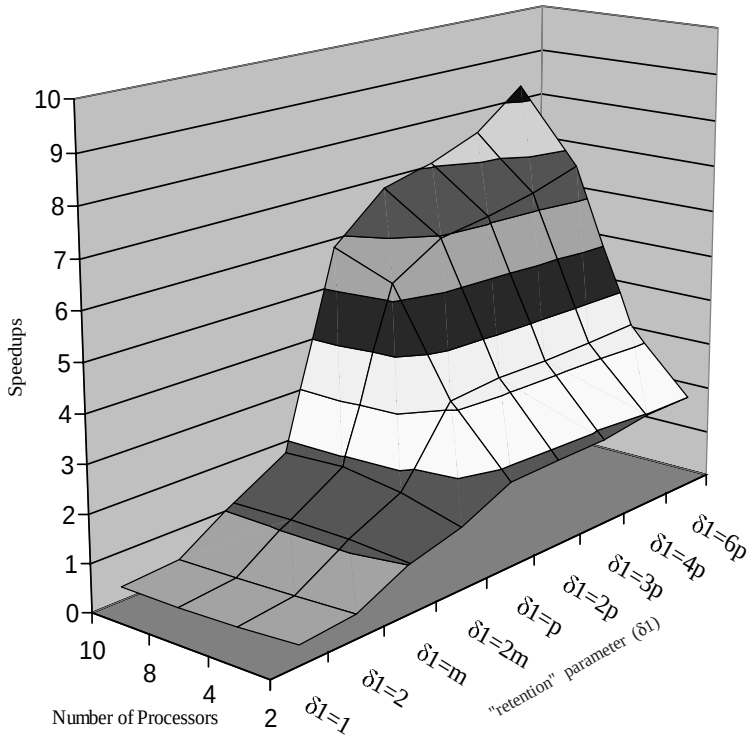


Fig. 4. Performance evaluation measurements of the **PNEPCGS** method, with $n = 8000$, $m = 21$ and $p = 401$.

## Acknowledgment

The authors would like to thank indeed Professor N.M. Missirlis, Department of Informatics and Telecommunications of University of Athens, for his courtesy and the Atmospheric Modeling and Weather Forecasting Group

(AM&WFG) of the School of Physics of the University of Athens for allowing us to use their cluster upon Professor Missirlis recommendation.

## References

[1] S.G. Akl, *Parallel Computation: Models and Methods*, Prentice Hall, 1997.
[2] M. Benzi, C.D. Meyer and M. Tuma, A sparse approximate inverse preconditioner for the conjugate gradient method, *SIAM J. Sci. Comput.* **17** (1996), 1135–1149.
[3] J.J. Dongarra, I. Duff, D. Sorensen and H.A. van der Vorst, *Numerical Linear Algebra for High-Performance Computers*, SIAM, 1998.
[4] I. Duff, The impact of high performance computing in the solution of linear systems: trends and problems, *J. Comp. Applied Math.* **123** (2000), 515–530.
[5] K. Giannoutakis and G.A. Gravvanis, A Normalized Explicit Conjugate Gradient method for solving sparse non-linear systems, in: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA '2002*), (Vol. I), H.R. Arabnia, eds, 2002, pp. 107–113, CSREA Press.
[6] G.A. Gravvanis, Explicit Approximate Inverse Preconditioning Techniques, *Archives of Computational Methods in Engineering* **9**(4) (2002), 371–402.
[7] G.A. Gravvanis, *Parallel matrix techniques, Computational Fluid Dynamics*, K. Papailiou, D. Tsahalis, J. Periaux, C. Hirsch, M. Pandolfi, eds, I, 1998, 472–477, Wiley.
[8] G.A. Gravvanis and K.M. Giannoutakis, Normalized explicit finite element approximate inverse preconditioning, *Intern. J. of Computers and Structures* **82**(28) (2004), 2377–2388.
[9] M.J. Grote and T. Huckle, Parallel preconditioning with sparse approximate inverses, *SIAM J. Sci. Comput.* **18** (1997), 838–853.
[10] T. Huckle, Efficient computations of sparse approximate inverses, *Numer. Linear Alg. with Appl.* **5** (1998), 57–71.
[11] P.S. Pacheco, *Parallel programming with MPI*, Morgan Kaufmann Publishers, 1997.
[12] M.J. Quinn, *Parallel Programming in C with MPI and OpenMP*, Mc-Graw Hill, 2003.
[13] Y. Saad and H.A. van der Vorst, Iterative solution of linear systems in the $20^{th}$ century, *J. Comp. Applied Math.* **123** (2000), 1–33.