 CMES, vol.16, no.2, pp.69-82, 2006

Distributed Finite Element Normalized Approximate Inverse Preconditioning

G.A. Gravvanis[1] and K.M. Giannoutakis[1]

Abstract: A new class of normalized explicit optimized approximate inverse finite element matrix techniques, based on normalized finite element approximate factorization procedures, for solving sparse linear systems resulting from the finite element 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 finite element matrix techniques for solving efficiently sparse finite element linear systems on distributed memory systems is also presented along with theoretical estimates on speedups and efficiency. The performance on a distributed memory machine, using Message Passing Interface (MPI) communication library, is also investigated. Applications on characteristic non-linear initial / boundary value problems in three dimensions are discussed and numerical results are given.

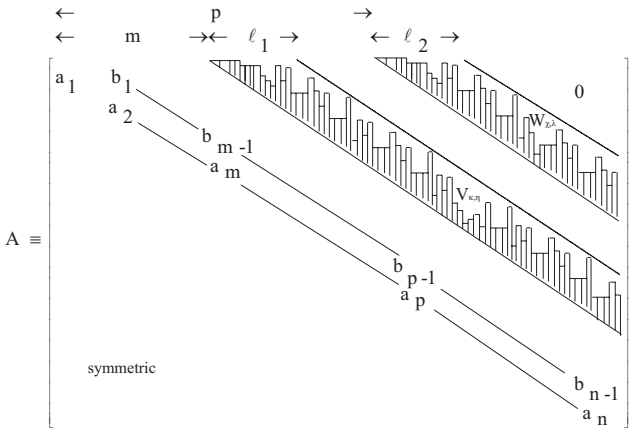
keyword: Finite element method, sparse linear systems, normalized approximate factorization procedures, normalized approximate inverses, preconditioning, parallel preconditioned conjugate gradient methods, parallel computations, distributed computations.

1 Introduction

Let us consider the linear system resulting from the finite element (FE) discretization of an elliptic boundary value problem in three dimensions, i.e.,

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

where A is a non-singular large sparse symmetric positive definite, diagonally dominant $(n \times n)$ matrix, with all the off-center band terms grouped into regular bands of width ℓ_1 and ℓ_2 at semi-bandwidths m and p respectively, viz.,



(2)

while u is the FE solution and s is a vector, of which the components result from a combination of source terms and imposed boundary conditions.

The solution of sparse finite element linear systems is of central importance to scientific and engineering computations. Because of their use in many important applications the need for efficient solvers and the related software has been an important objective for the derivation of parallel and distributed solvers and software, [Akl (1997); Benzi (2002); Dongarra, Duff, Sorensen, and van der Vorst (1998); Gravvanis (2002); Saad and van der Vorst (2000)]. The cost-effectiveness of parallel iterative methods over parallel direct solution methods for solving sparse finite element linear systems is now commonly accepted, especially for three - dimensional problems.

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, [Benzi (2002); Gravvanis (2002); Greenbaum (1997); Saad (1996); Saad and van der Vorst (2000)]. The preconditioned form of the linear system (Eq. 1) is

$$MAu = Ms, \tag{3}$$

[1] Department of Electrical and Computer Engineering, School of Engineering, Democritus University of Thrace, 12 Vas. Sofias street, GR 67100 Xanthi, Greece; Email: {ggravvan, kgiannou}@ee.duth.gr

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, [Benzi (2002); Benzi, Meyer, and Tuma (1996); Gravvanis (2002, 1998); Grote and Huckle (1997); Huckle (1999); Saad (1996); Saad and van der Vorst (2000); van der Vorst (1989)]. Recently, new parallel numerical algorithms and related software have been produced for solving sparse finite element linear systems, by parallel approximate inverse preconditioning methods on multiprocessor and multi-computer systems. 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, [Gravvanis (2002, 1998); Gravvanis and Giannoutakis (2003)].

It is known that finite element approximate factorization procedures and inverse matrix algorithms are in general complicated. However as the demand for solving sparse finite element linear systems grows, the need to use efficient sparse equations solvers based on approximate factorization procedures and inverse matrix algorithms becomes one of great importance, [Gravvanis (2002, 1998); Gravvanis and Giannoutakis (2005)].

In Section 2, we present approximate inverse finite element matrix algorithms, based on the normalized approximate factorization of the coefficient finite element matrix A. In Section 3, parallel normalized explicit preconditioned conjugate gradient - type methods are presented, for distributed memory parallel systems, using the MPI (Message Passing Interface) communication library. The implementation of the proposed parallel method is presented and theoretical estimates on speedups and efficiency is given. Finally, in Section 4 the performance of the parallel normalized explicit preconditioned conjugate gradient method is illustrated by solving sparse finite element linear system on a distributed system using the communication library MPI and speedups are given. Additionally the performance and applicability of the normalized explicit preconditioned conjugate gradient - type methods is illustrated by solving characteristic non-linear initial/boundary value problems in three dimensions and numerical results are given and discussed.

2 Normalized finite element approximate factorization and optimized approximate inverses

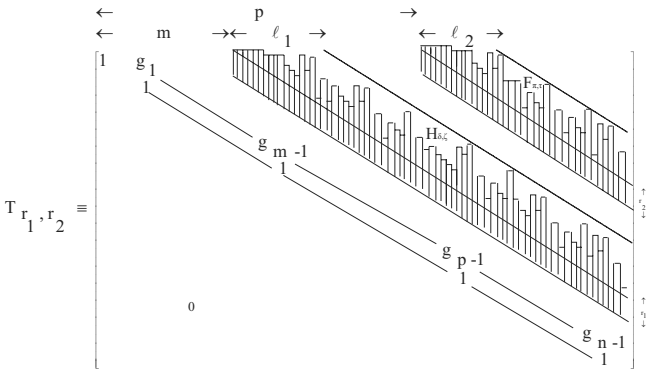
In this section we present the normalized approximate inverse matrix techniques based on an algorithmic procedure of inverting a real $(n\times n)$ matrix A without inverting the decomposition factors, [Gravvanis (2002, 1998); Gravvanis and Giannoutakis (2003)].

Let us now assume the normalized approximate factorization, [Gravvanis and Giannoutakis (2005, 2004, 2003); Lipitakis and Evans (1984)], such that:

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

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

$$D_{r_1,r_2} = \mathrm{diag}\left\{ d_1,\ldots,d_{m-1}\vdots d_m,\ldots,d_{p-1}\vdots d_p,\ldots,d_n \right\}, \tag{5}$$



(6)

Then, the elements of the decomposition factors D_{r_1,r_2} and T_{r_1,r_2}, can be computed by the **F**inite **E**lement **A**pproximate **No**rmalized **F**actorization algorithm (henceforth called the **FEANOF-3D** algorithm), [Gravvanis and Giannoutakis (2005)], and can be expressed by the following compact algorithmic scheme:

$$d_1 = \sqrt{a_1} \tag{7}$$

for $i = 2$ **to** $m-1$

$$d_i = \left(a_i - \left(\frac{b_{i-1}}{d_{i-1}} \right)^2 \right)^{1/2} \tag{8}$$

$$g_{i-1} = \frac{b_{i-1}}{d_{i-1}d_i} \tag{9}$$

let $\mu = r_1 + \ell_1$
for $j = 1$ to $n-m+1$
if $r_2 < p-m+1$ **or** $j < p-m+1$ **then**

The equations to determine the elements of T_{r_1,r_2} proved to be non-linear and a simple iterative Picard-type scheme was used in an inner loop to determine the values of $d_m,\ldots,d_{p-1}$, as the direct solution of these equations proved to be intractable, [Lipitakis and Evans (1984)].

Find the first non-zero element of submatrix $V_{\kappa,\eta}$ of the j-th column of A (the procedure IXNOS can be used, [Lipitakis and Evans (1984)]).

$$d_{m+j-1} = d_{m+j-2} \tag{10}$$
if $j < \ell_1$ **then**
$$h_{k,j} = \frac{v_{k,n}}{d_k d_{m+j-1}} \tag{11}$$
else
$$h_{k+1-j,j} = \frac{v_{k,n}}{d_k d_{m+j-1}} \tag{12}$$
$$g_{m+j-2} = \frac{b_{m+j-2}}{d_{m+j-2}d_{m+j-1}} \tag{13}$$
if $j < r_1 + \ell_1 - 1$ **then**
if $(j < \ell_1)$ **and** $(k < r_1)$ **then**
for $i = k+1$ **to** r_1
$$h_{i,j} = -g_{i-1}h_{i-1,j} \tag{14}$$
if $i < j+1$ **then**
$$h_{i,j} = h_{i,j} + \frac{v_{i,j-i+1}}{d_i d_{m-j+1}} \tag{15}$$
else
for $i = k+\ell_1-j+1$ **to** $\mu - j$
$$h_{i,j} = -g_{i+j-\ell_1-1}h_{i-1,j} \tag{16}$$
if $i < \ell_1 + 1$ **then**
$$h_{i,j} = h_{i,j} + \frac{v_{i+j-\ell_1,\ell_1-i+1}}{d_{i+j-\ell_1}d_{m+j-1}} \tag{17}$$
if $j < \mu - 1$ **then**
$$p = \mu - j + 1 \tag{18}$$
else
$$p = k + \ell_1 - j + 1 \tag{19}$$
if $(j < \mu - 1)$**and**$(k > m-1)$ **then**
$$p = k + \ell_1 - j + 1 \tag{20}$$
if $j > \ell_1 - 1$ **then**
for $i = p$ **to** $\mu - 1$
if $(i > m + 2\ell_1 - j - 1)$**and**$(j > \ell_1 + 1)$ **then**
$$h_{i,j} = -g_{i+j-\ell_1-1}h_{i-1,j} - \sum_{\lambda=1}^{i-1} h_{\lambda-i+\mu,i+j-\mu}h_{\lambda,j} \tag{21}$$
else
$$h_{i,j} = -g_{i+j-\ell_1-1}h_{i-1,j} - \sum_{\lambda=1}^{i-1} h_{\lambda+j-\ell_1,i+j-\mu}h_{\lambda,j} \tag{22}$$
if $i < \ell_1 + 1$ **then**
$$h_{i,j} = h_{i,j} + \frac{v_{i+j-\ell_1,\ell_1-i+1}}{d_{i+j-1}d_{m+j-1}} \tag{23}$$
else
if $k < r_1 + 1$ **then**
$$p = r_1 + 1 \tag{24}$$
else
$$p = k + 1 \tag{25}$$
for $i = p$ **to** $r_1 - j + 1$
$$h_{i,j} = -g_{i-1}h_{i-1,j} - \sum_{\lambda=1}^{i-1} h_{\lambda,i-r_1}h_{\lambda,j} \tag{26}$$
if $i < j+1$ **then**
$$h_{i,j} = h_{i,j} + \frac{v_{i,j-i+1}}{d_i d_{m+j-1}} \tag{27}$$
if $j < \ell_1$ **then**
$$d_{m+j-1} = \sqrt{\frac{a_{m+j-1}}{1+\sum_{\lambda=1}^{r_1+j-2}\left(h_{\lambda,j}\right)^2+\left(h_{r_1+j-1,j}+g_{m+j-2}\right)^2}} \tag{28}$$
else
$$d_{m+j-1} = \sqrt{\frac{a_{m+j-1}}{1+\sum_{\lambda=1}^{r_1+\ell_1-2}\left(h_{\lambda,j}\right)^2+\left(h_{r_1+\ell_1-1,j}+g_{m+j-2}\right)^2}} \tag{29}$$
if $(r_2 < p-m+1)$ **or** $(j \geq p-m+1)$ **then**

The equations to determine the elements of T_{r_1,r_2} proved to be non-linear and a simple iterative Picard-type scheme was used in an inner loop to determine the values of $d_p,\ldots,d_n$, as the direct solution of these equations proved to be intractable, [Lipitakis and Evans (1984)].

Find the first non-zero element of submatrix $W_{\chi,\lambda}$ of the $(j-p+m)$th column of A (the procedure IXNOS can be used, [Lipitakis and Evans (1984)]).

$$d_{m+j-1} = d_{m+j-2} \tag{30}$$
if $j - p + m \leq \ell_2$ **then**
$$f_{\chi,j-p+m} = \frac{w_{\chi,\lambda}}{d_k d_{m+j-1}} \tag{31}$$
else
$$f_{\chi+\ell_2-j+p-m,j-p+m} = \frac{w_{\chi,\lambda}}{d_k d_{m+j-1}} \tag{32}$$
if $j - p + m \leq r_2 + \ell_2 - 2$ **then**
if $j - p + m \leq \ell_2$ **then**
$$f_{i,j-p+m} = -d_{i-1}f_{i-1,j-p+m} \tag{33}$$
if $i < j - p + m + 1$ **then**
$$f_{i,j-p+m} = f_{i,j-p+m} - \frac{w_{i,j-p+m-i+1}}{d_i d_{m+j-1}} \tag{34}$$
for $i = \chi+1,\ldots,r_2+j-p+m-1$
else
$$f_{i,j-p+m} = -g_{i+j-p+m-\ell_2-1}f_{i-1,j-p+m} \tag{35}$$
if $i \leq \ell_2$ **then**
$$f_{i,j-p+m} = f_{i,j-p+m} - \frac{w_{i+j-p+m-\ell_2,\ell_2-i+1}}{d_i d_{m+j-1}} \tag{36}$$
for $i = \chi + \ell_2 - j + p - m + 1,\ldots,$
$r_2 + \ell_2 - j + p - m + 1$
if $j - p + m > \ell_2$ **then**
if $(i \geq p + 2\ell_2 - j + p - m)$ **and**

$(j-p+m \geq \ell_2+2)$ **then**

$$f_{i,j-p+m} = -g_{i+j-p+m-\ell_2-1} f_{i-1,j-p+m} - \sum_{k=1}^{i-1} h_{k-i+r_2+\ell_2,i+j-p+m-r_2-\ell_2} f_{k,j-p+m} \quad (37)$$

else

$$f_{i,j-p+m} = -g_{i+j-p+m-\ell_2-1} f_{i-1,j-p+m} - \sum_{k=1}^{i-1} h_{k+j-p+m-\ell_2,i+j-p+m-r_2-\ell_2} f_{k,j-p+m} \quad (38)$$

then, if $i \leq \ell_2$

$$f_{i,j-p+m} = f_{i,j-p+m} + \frac{w_{i+j-p+m-\ell_2,\ell_2-i+1}}{d_{i+j-p+m-\ell_2} d_{m+j-1}} \quad (39)$$

for either $i = r_2+\ell_2-j+1+p-m,\ldots,r_2+\ell_2-1$

and $\chi < p$

or $i = \chi+\ell_2-j+p-m+1,\ldots,r_2+\ell_2-1$

and $\chi \geq p$, **with** $j < r_2+\ell_2-1$

or $i = \chi+\ell_2-j+p-m+1,\ldots,r_2+\ell_2-1$

for all $j-p+m \geq r_2+\ell_2-1$

if $j-p+m \leq \ell_2$ **then**

$$d_{m+j-1} = \sqrt{\frac{w_{m+j-1}}{1+\sum_{k=1}^{r_1+\ell_1-1} h_{k,j}^2 + \sum_{k=1}^{r_2+j-p+m-1} f_{k,j-p+m}^2}} \quad (40)$$

else

$$d_{m+j-1} = \sqrt{\frac{w_{m+j-1}}{1+\sum_{k=1}^{r_1+\ell_1-1} h_{k,j}^2 + \sum_{k=1}^{r_2+\ell_2-1} f_{k,j-p+m}^2}} \quad (41)$$

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 $\approx \left[(r_1+\ell_1)^2+(r_2+\ell_2)^2\right]n$ multiplicative operations and n square roots, [Gravvanis and Giannoutakis (2005)].

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 $V=(v_{\kappa,\eta})$, $\kappa \in [1,n-m+1]$, $\eta \in [1,\ell_1]$ denotes the elements of the κ-th row and $(\kappa+\eta+m-2)$-th column of A in its usual arrangement. In a similar way the submatrix $W=\left(w_{\chi,\lambda}\right)$ 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 $i \leq m-1$) denotes the elements in the i-th row and the $(m+j-1)$-th column (if $j \leq \ell_1$) or the elements in the $(i+j-\ell_1)$-th row and the $(m+j-1)$-th column (if $j > \ell_1$) while $h_{i,j}$ (for $i > m-1$) denotes the elements in the i-th row and the $(i+j)$-th column (if $i+j \leq m+\ell_1-1$) or the elements in the $(2i+j-m-\ell_1+1)$-th row and $(i+j)$-th column (if $i+j > m+\ell_1-1$) of the coefficient matrix A in its usual arrangement. 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 $i \leq p-1$) denotes the elements in the i-th row and the $(p+j-1)$-th column (if $j \leq \ell_2$) or the elements in the $(i+j-\ell_2)$-th row and the $(p+j-1)$-th column (if $j > \ell_2$) while $f_{i,j}$ (for $i > p-1$) denotes the elements in the i-th row and the $(i+j)$-th column (if $i+j \leq p+\ell_2-1$) or the elements in the $(2i+j-p-\ell_2+1)$-th row and $(i+j)$-th column (if $i+j > p+\ell_2-1$) of the coefficient matrix A in its usual arrangement.

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

$$M_{r_1,r_2}^{\delta l} = \left(D_{r_1,r_2} T_{r_1,r_2}^t T_{r_1,r_2} D_{r_1,r_2}\right)^{-1} = D_{r_1,r_2}^{-1} \left(T_{r_1,r_2}^t T_{r_1,r_2}\right)^{-1} D_{r_1,r_2}^{-1} = D_{r_1,r_2}^{-1} \widehat{M}_{r_1,r_2}^{\delta l} D_{r_1,r_2}^{-1}, \quad (42)$$

where

$$\widehat{M}_{r_1,r_2}^{\delta l} = \left(T_{r_1,r_2}^t T_{r_1,r_2}\right)^{-1}. \quad (43)$$

Then, the elements of the inverse $\widehat{M}_{r_1,r_2}^{\delta l}$ can be computed, by retaining δl elements in the lower and upper part of the inverse and using an optimized storage scheme, [Gravvanis (2002)], and by solving recursively the systems:

$$\widehat{M}_{r_1,r_2}^{\delta l} T_{r_1,r_2}^t = \left(T_{r_1,r_2}\right)^{-1} \text{ and } T_{r_1,r_2} \widehat{M}_{r_1,r_2}^{\delta l} = \left(T_{r_1,r_2}^t\right)^{-1}, \quad (44)$$

without inverting the decomposition factors T_{r_1,r_2} and T_{r_1,r_2}^t.

The storage requirements of the normalized optimized approximate inverse are only $n \times (2\delta l-1)$-vector spaces. The **Nor**malized **O**ptimized **B**anded **A**pproximate **I**nverse **F**inite **E**lement **M**atrix algorithmic procedure (henceforth called the **NOROBAIFEM-3D** algorithm) for computing the elements of the approximate inverse, using a "fish-bone" pattern, can be expressed by the following compact form:

Let $r\ell_1 = r_1+\ell_1$; $r\ell_2 = r_2+\ell_2$;

$r\ell 11 = r\ell 1 - 1$; $r\ell 21 = r\ell 2 - 1$; $mr1 = m - r_1$;
$pr2 = p - r_2$; $m\ell 1 = m + \ell_1$; $p\ell 2 = p + \ell_2$;
$nmr1 = n - m + r_1$; $npr2 = n - p + r_2$.

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**

$$\widehat{\mu}_{1,1} = 1 \tag{45}$$

else

$$\widehat{\mu}_{n-i+1,1} = 1 - g_j \cdot \widehat{\mu}_{n-j,\delta l+1} \tag{46}$$

else

$$\widehat{\mu}_{n-i+1,i-j+1} = -g_j \cdot \widehat{\mu}_{n-i+1,i-j} \tag{47}$$

else
if $j > npr2$ **and** $j \leq nmr1$ **then**
if $i = j$ **then**

$$\widehat{\mu}_{n-i+1,1} = 1 - g_j \cdot \widehat{\mu}_{n-j,\delta l+1} - \sum_{k=0}^{nmr1-j} h_{r\ell 11-k,j+k+1-r_1} \cdot \widehat{\mu}_{x,y} \tag{48}$$

call $\mathbf{mw}(n, \delta l, i, j + mr1 + k, x, y)$

else

$$\widehat{\mu}_{n-i+1,i-j+1} = -g_j \cdot \widehat{\mu}_{n-i+1,i-j} - \sum_{k=0}^{nmr1-j} h_{r\ell 11-k,j+k+1-r_1} \cdot \widehat{\mu}_{x,y} \tag{49}$$

call $\mathbf{mw}(n, \delta l, i, j + mr1 + k, x, y)$

else
if $j \geq r\ell 1$ **and** $j \leq npr2$ **then**
if $i = j$ **then**

$$\widehat{\mu}_{n-i+1,1} = 1 - g_j \cdot \widehat{\mu}_{n-j,\delta l+1} - \sum_{k=0}^{npr2-j} f_{r\ell 21-k,j+k+1-r_2} \cdot \widehat{\mu}_{x_1,y_1} - \sum_{k=0}^{nmr1-j} h_{r\ell 11-k,j+k+1-r_1} \cdot \widehat{\mu}_{x_2,y_2} \tag{50}$$

call $\mathbf{mw}(n, \delta l, i, j + k + pr2, x_1, y_1)$
call $\mathbf{mw}(n, \delta l, i, j + k + mr1, x_2, y_2)$

else

$$\widehat{\mu}_{n-i+1,i-j+1} = -g_j \cdot \widehat{\mu}_{n-i+1,i-j} - \sum_{k=0}^{npr2-j} f_{r\ell 21-k,j+k+1-r_2} \cdot \widehat{\mu}_{x_1,y_1} - \sum_{k=0}^{nmr1-j} h_{r\ell 11-k,j+k+1-r_1} \cdot \widehat{\mu}_{x_2,y_2} \tag{51}$$

call $\mathbf{mw}(n, \delta l, i, j + k + pr2, x_1, y_1)$
call $\mathbf{mw}(n, \delta l, i, j + k + mr1, x_2, y_2)$

else
if $i = j$ **then**
if $= 1$ **then**

$$\widehat{\mu}_{n,1} = 1 - g_1 \cdot \widehat{\mu}_{n-1,\delta l+1} - \sum_{k=1}^{\ell_2} f_{1,k} \cdot \widehat{\mu}_{x_1,y_1} - \sum_{k=1}^{\ell_1} h_{1,k} \cdot \widehat{\mu}_{x_2,y_2} \tag{52}$$

call $\mathbf{mw}(n, \delta l, 1, p + k - 1, x_1, y_1)$
call $\mathbf{mw}(n, \delta l, 1, m + k - 1, x_2, y_2)$

else

$$\widehat{\mu}_{n-i+1,1} = 1 - g_j \cdot \widehat{\mu}_{n-j,\delta l+1} - \sum_{k=1}^{j-1} h_{j-k,\ell_1+k} \cdot \widehat{\mu}_{x_1,y_1} - \sum_{k=j+1-r_1}^{\ell_1} h_{j,k} \cdot \widehat{\mu}_{x_2,y_2} - \sum_{k=1}^{j-1} f_{j-k,\ell_2+k} \cdot \widehat{\mu}_{x_3,y_3} - \sum_{k=j+1-r_2}^{\ell_2} f_{j,k} \cdot \widehat{\mu}_{x_4,y_4} \tag{53}$$

call $\mathbf{mw}(n, \delta l, i, m\ell 1 + k - 1, x_1, y_1)$
call $\mathbf{mw}(n, \delta l, i, m + k - 1, x_2, y_2)$
call $\mathbf{mw}(n, \delta l, i, p\ell 2 + k - 1, x_3, y_3)$
call $\mathbf{mw}(n, \delta l, i, p + k - 1, x_4, y_4)$

else

$$\widehat{\mu}_{n-i+1,i-j+1} = -g_j \cdot \widehat{\mu}_{n-i+1,i-j} - \sum_{k=1}^{j-1} h_{j-k,\ell_1+k} \cdot \widehat{\mu}_{x_1,y_1} - \sum_{k=j+1-r_1}^{\ell_1} h_{j,k} \cdot \widehat{\mu}_{x_2,y_2} - \sum_{k=1}^{j-1} f_{j-k,\ell_2+k} \cdot \widehat{\mu}_{x_3,y_3} - \sum_{k=j+1-r_2}^{\ell_2} f_{j,k} \cdot \widehat{\mu}_{x_4,y_4} \tag{54}$$

call $\mathbf{mw}(n, \delta l, i, m\ell 1 + k - 1, x_1, y_1)$
call $\mathbf{mw}(n, \delta l, i, m + k - 1, x_2, y_2)$
call $\mathbf{mw}(n, \delta l, i, p\ell 2 + k - 1, x_3, y_3)$
call $\mathbf{mw}(n, \delta l, i, p + k - 1, x_4, y_4)$

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

$$\widehat{\mu}_{n-i+1,\delta l+i-j} = \widehat{\mu}_{n-i+1,i-j+1} \tag{55}$$

The procedure $\mathbf{mw}(n, \delta l, s, q, x, y)$, [Gravvanis (2002)], can then be described as follows:

procedure $\mathbf{mw}(n, \delta l, s, q, x, y)$
if $s \geq q$ **then**

$$x = n + 1 - s \tag{56}$$

$$y = s - q + 1 \tag{57}$$

else

$$x = n + 1 - q \tag{58}$$

$$y = \delta l + q - s \tag{59}$$

The computational work of the **NOROBAIFEM-3D**

algorithm is $O[n\delta l\,(r_1+r_2+\ell_1+\ell_2+1)]$ multiplicative operations, while the storage of the approximate inverse is only $n\times(2\delta l-1)$-vector spaces. This optimized form is particularly effective for solving "narrow-banded" sparse systems of very large order , i.e. $\delta l << n/2$. The construction of parallel inverses has been studied in [Gravvanis (1998)], and currently 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:

$$A^{-1}\equiv\ D^{-1}\widehat{M}D^{-1}\leftarrow D^{-1}_{r_1,r_2}\underbrace{\widehat{\widehat{M}}^{\delta l}_{r_1=m-1,r_2=p-1}}_{\text{class I}}D^{-1}_{r_1,r_2}\leftarrow D^{-1}_{r_1,r_2}\underbrace{\widehat{M}^{\delta l}_{r_1=m-1,r_2=p-1}}_{\text{class II}}D^{-1}_{r_1,r_2}\leftarrow D^{-1}_{r_1,r_2}\underbrace{\widehat{M}^{\delta l}_{r_1,r_2}}_{\text{class III}}D^{-1}_{r_1,r_2}\leftarrow\underbrace{D^{-2}_{r_1,r_2}}_{\text{class IV}} \tag{60}$$

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 ($r_1\leq m-1$, $r_2\leq p-1$). 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 $\widehat{M}^{\delta l}_{r_1,r_2}=I$, resulting in a fast inverse algorithm.

It is known that the larger in magnitude elements of the inverse matrices resulting from the discretization of P.D.E's in three space variables, 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, [Gravvanis (2002, 1998); Gravvanis and Giannoutakis (2003)]. It is reasonable to assume that the value of the retention parameter δl can be chosen as multiples of m and p.

3 Parallel Normalized explicit preconditioned conjugate gradient methods

In this section we present a class of normalized explicit preconditioned conjugate gradient schemes, based on the derived **NOROBAIFEM-3D** algorithm. The **N**ormalized **E**xplicit **P**reconditioned **C**onjugate **G**radient (**NEPCG**) method for solving linear systems has been presented in [Gravvanis and Giannoutakis (2003)]. The computational complexity of the **NEPCG** method is $\approx O[(2\delta l+2\ell_1+2\ell_2+11)n$ mults $+3n$ adds$]\nu$ operations, where ν denotes the number of iterations required for convergence to a predetermined tolerance level.

The **N**ormalized **E**xplicit **P**reconditioned **C**onjugate **G**radient **S**quare (**NEPCGS**) method can be expressed by the following compact algorithmic scheme:

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

$$\text{set}\quad u_0=0,\text{ and }e_0=0, \tag{61}$$
$$\text{solve}\quad r_0=D^{-1}_{r_1,r_2}\widehat{M}^{\delta l}_{r_1,r_2}D^{-1}_{r_1,r_2}(s-Au_0), \tag{62}$$
$$\text{set}\quad \sigma_0=r_0,\text{ and }p_0=(\sigma_0,r_0), \tag{63}$$

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:

$$\text{form}\quad q_i=A\sigma_i, \tag{64}$$
$$\text{calculate}\quad \alpha_i=p_i/\left(\sigma_0,D^{-1}_{r_1,r_2}\widehat{M}^{\delta l}_{r_1,r_2}D^{-1}_{r_1,r_2}q_i\right), \tag{65}$$
$$\text{compute}\quad e_{i+1}=r_i+\beta_i e_i-\alpha_i D^{-1}_{r_1,r_2}\widehat{M}^{\delta l}_{r_1,r_2}D^{-1}_{r_1,r_2}q_i, \tag{66}$$
$$d_i=r_i+\beta_i e_i+e_{i+1}, \tag{67}$$
$$\text{form}\quad u_{i+1}=u_i+\alpha_i d_i,\text{ and }q_i=Ad_i, \tag{68}$$
$$\text{compute}\quad r_{i+1}=r_i-\alpha_i D^{-1}_{r_1,r_2}\widehat{M}^{\delta l}_{r_1,r_2}D^{-1}_{r_1,r_2}q_i, \tag{69}$$
$$\text{set}\quad p_{i+1}=(\sigma_0,r_{i+1})\text{ and }\beta_{i+1}=p_{i+1}/p_i, \tag{70}$$
$$\text{compute}\quad \sigma_{i+1}=r_{i+1}+2\beta_{i+1}e_{i+1}+\beta^2_{i+1}\sigma_i. \tag{71}$$

The computational complexity of the **NEPCGS** method is $\approx O[(4\delta l+4\ell_1+4\ell_2+19)n$ mults $+8n$ adds$]\nu$ operations, where ν denotes the number of iterations required for convergence to a predetermined tolerance level.

The **N**ormalized **E**xplicit **P**reconditioned **BI**conjugate **C**onjugate **G**radient-**STAB** (**NEPBICG-STAB**) method, has also been presented in [Gravvanis and Giannoutakis (2003)]. The computational complexity of the **NEPBICG-STAB** method is

$\approx O[(6\delta l+4\ell_1+4\ell_2+22)n \text{ mults}+6n \text{ adds}]\nu$ operations, where ν denotes the number of iterations required for convergence to a predetermined tolerance level. The effectiveness of the normalized explicit preconditioned schemes, using the **NOROBAIFEM-3D** algorithm, is related to the fact that the normalized approximate inverse exhibits a similar "fuzzy" structure as the coefficient matrix A.

Let us consider that the coefficient matrix A, the normalized optimized approximate inverse $D^{-1}_{r_1,r_2}\widehat{M}^{\delta l}_{r_1,r_2}D^{-1}_{r_1,r_2}$ and the vectors s, u_i, r_i, σ_i, q_i, are distributed in a block-row distribution. In this balanced 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 matrix $\times$ vector multiplications and after inner products.

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**) can be expressed by the following algorithmic procedure:
Let u_0 be an arbitrary initial approximation to the solution vector u. Then,

for $(j = myrank*local\_n+1)$ **to** $(myrank*local\_n+local\_n)$
($myrank$ is the rank of each process)

$$(r_0^*)_j = s_j - A(u_0)_j \tag{72}$$

if $\delta l = 1$ **then**
  for $(j = myrank*local\_n+1)$ **to** $(myrank*local\_n+local\_n)$

$$(r_0)_j = (r_0^*)_j / \left(d^2\right)_j \tag{73}$$

else
  gather distributed r_0^* onto each process
  for $(j = myrank*local\_n+1)$ **to** $(myrank*local\_n+local\_n)$

$$(r_0)_j = \left(\sum_{k=\max(1,j-\delta l+1)}^{\min(n,j+\delta l-1)} \widehat{\mu}_{j,k}(r_0^*)_k/d_k\right)/(d)_j \tag{74}$$

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

$$(\sigma_0)_j = (r_0)_j \tag{75}$$

$$(p_0)_j = \left((\sigma_0)_j,(r_0)_j\right) \tag{76}$$

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}, σ_{i+1} and the scalar quantities α_i, β_{i+1} as follows:

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

$$(q_i)_j = A(\sigma_i)_j \tag{77}$$

if $\delta l = 1$ **then**
  for $(j = myrank*local\_n+1)$ **to** $(myrank*local\_n+local\_n)$

$$(g_i)_j = (q_i)_j / \left(d^2\right)_j \tag{78}$$

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

$$(g_i)_j = \left(\sum_{k=\max(1,j-\delta l+1)}^{\min(n,j+\delta l-1)} \widehat{\mu}_{j,k}(q_i)_k/d_k\right)/(d)_j \tag{79}$$

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

$$(t_i)_j = \left((\sigma_0)_j,(g_i)_j\right) \tag{80}$$

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

$$a_i = p_i/t_i \tag{81}$$

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

$$(e_{i+1})_j = (r_i)_j + b_i(e_i)_j - a_i(g_i)_j \tag{82}$$

$$(f_i)_j = (r_i)_j + b_i(e_i)_j + (e_{i+1})_j \tag{83}$$

$$(u_{i+1})_j = (u_i)_j + a_i(f_i)_j \tag{84}$$

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

$$(q_i)_j = A(f_i)_j \tag{85}$$

if $\delta l = 1$ **then**
  for $(j = myrank*local\_n+1)$ **to** $(myrank*local\_n+local\_n)$

$$(g_i)_j = (q_i)_j / \left(d^2\right)_j \tag{86}$$

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

$$(g_i)_j = \left(\sum_{k=\max(1,j-\delta l+1)}^{\min(n,j+\delta l-1)} \widehat{\mu}_{j,k}(q_i)_k/d_k\right)/(d)_j \tag{87}$$

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

$$(r_{i+1})_j = (r_i)_j - a_i(g_i)_j \tag{88}$$

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

$$(p_{i+1})_j = \left((\sigma_0)_j, (r_{i+1})_j\right)^{(myrank*local\_n+local\_n)} \quad (89)$$

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

$$b_{i+1} = p_{i+1}/p_i \quad (90)$$

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

$$(\sigma_{i+1})_j = (r_{i+1})_j + 2b_{i+1}(e_{i+1})_j + b_{i+1}^2(\sigma_i)_j^{(myrank*local\_n+local\_n)} \quad (91)$$

The computational complexity of the **PNEPCGS** method is $\approx O[(4\delta l+4\ell_1+4\ell_2+19)n\ local\_n$ mults $+8local\_n$ adds$]\nu$ operations, while the total communication cost is $\approx O[5t_s\log(no\_proc)+2local\_n(no\_proc-1)t_w]\nu$, where ν 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:

$$S_p = \frac{1}{\frac{1}{no\_proc}+\frac{5t_s\log(no\_proc)}{O(\delta l)nt_m}+\frac{2(no\_proc-1)t_w}{O(\delta l)}} \quad (92)$$

and

$$E_p = \frac{1}{1+\frac{5t_s no\_proc\log(no\_proc)}{O(\delta l)nt_m}+\frac{2(no\_proc-1)t_w}{O(\delta l)}} \quad (93)$$

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_ν is:

$$O\left(n^{1/2}(\delta l+\ell_1+\ell_2)^{1/2}local\_n\log\varepsilon^{-1}\right), \quad (94)$$

while the total communication cost is:

$$O\left[\frac{n^{1/2}\log\varepsilon^{-1}}{(\delta l+\ell_1+\ell_2)^{1/2}}(t_s\log(no\_proc) + local\_n\ no\_proc\ t_w)\right]. \quad (95)$$

4 Numerical Results

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

Model problem I: Let us consider the following 3D elliptic partial differential equation subject to Dirichlet boundary conditions:

$$u_{xx}+u_{yy}+u_{zz}+u = F, \quad (x,y,z)\in R, \quad (96)$$

$$u(x,y,z) = 0, \quad (x,y,z)\in \partial R, \quad (96.a)$$

where R is the unit cube and ∂R denotes the boundary of R. The domain is covered by a non-overlapping triangular network resulting in a hexagonal mesh. 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 "fill-in" parameters were set to $r_1 = r_2 = 2$ and the width parameters were set to $\ell_1 = \ell_2 = 3$. The iterative process was terminated when $\|r_i\|_\infty < 10^{-5}$.

The parallel numerical results were performed on a Beowulf cluster, which consists of forty eight (48) AMD Athlon XP, running at 1.7 GHz with 1.5GB RAM and 20Gb local boot drive connected in a 2Gbit network using Myrinet switch.

For communication operations, the MPI collective communication routines MPI_Allreduce and MPI_Allgather were used for sending and receiving data among distributed processes, [Pacheco (1997); Quinn (2003)].

The speedups and the number of iterations of the **PNEPCGS** method for several values of the "retention" parameter δl with $n = 8000$, $m = 21$, $p = 401$ with $r_1 = r_2 = 2$ and $\ell_1 = \ell_2 = 3$ are given in Tab. 1.

In Fig. 1, Fig. 2 and Fig. 3 the speedups and processors allocated for several values of the "retention" parameter δl, the speedups versus the "retention" parameter δl for several numbers of processors and the parallel efficiency for several values of the "retention" parameter δl are presented respectively for the **PNEPCGS** method with $n = 8000$, $m = 21$, $p = 401$ with $r_1 = r_2 = 2$ and $\ell_1 = \ell_2 = 3$.

In Fig. 4 the performance evaluation measurements of the **PNEPCGS** method are given with $n = 8000$, $m = 21$, $p = 401$ with $r_1 = r_2 = 2$ and $\ell_1 = \ell_2 = 3$.

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 δl, where the choice of symmetric multiprocessor systems is recommended, in contrast with large values of δl where speedups and efficiency tend to become optimum.

Model problem II: Let us consider the following non-

Table 1 : Speedups and processors allocated of the **PNEPCGS** method for several values of δl, with $n = 8000$, $m = 21$ and $p = 401$, with $r_1 = r_2 = 2$ and $\ell_1 = \ell_2 = 3$.

"Retention" parameter	**Speedups**					**Number of PNEPCGS iterations**
	Number of processors					
	2	4	8	16	32	
$\delta l = 1$	0.5669	0.3405	0.2267	0.1409	0.0852	11
$\delta l = 2$	0.4948	0.2678	0.1758	0.1066	0.0645	11
$\delta l = m$	0.7593	0.4770	0.2822	0.1974	0.1246	9
$\delta l = 2m$	0.9759	0.6869	0.4070	0.3024	0.1955	7
$\delta l = p$	1.8004	3.0555	3.2012	3.3146	3.3813	6
$\delta l = 2p$	1.8591	3.2250	5.0216	5.1399	5.3410	5
$\delta l = 3p$	1.8784	3.3414	5.2458	5.5970	8.2806	5
$\delta l = 4p$	1.8834	3.4010	6.9428	7.1225	9.6978	5
$\delta l = 6p$	1.8901	3.4112	6.9970	7.2610	9.8880	5

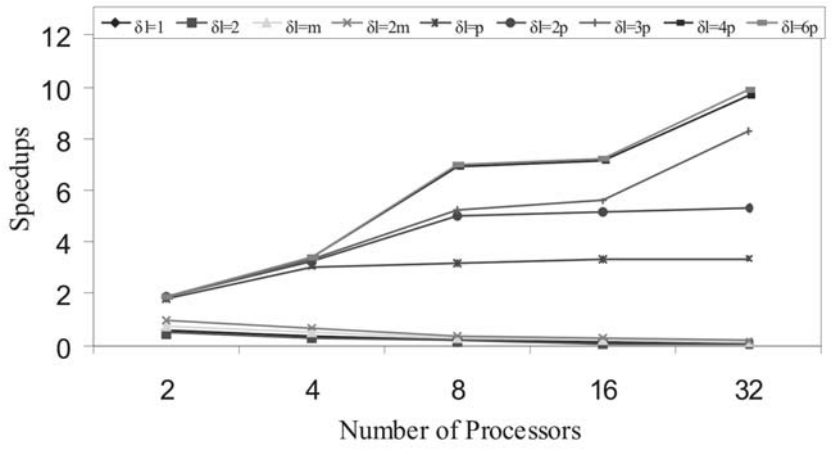


Figure 1 : Speedups and processors allocated of the **PNEPCGS** method for several values of δl, with $n = 8000$, $m = 21$ and $p = 401$, with $\ell_1 = \ell_2 = 3$ and $r_1 = r_2 = 2$.

linear P.D.E. in three space variables:

$$\varepsilon_L \Delta u = \alpha e^{\beta u}, \quad \varepsilon_L \to 0_+, \quad (x,y,z) \in \Omega, \tag{97}$$

with boundary conditions:

$$u(x,y,z) = 0. \tag{97.a}$$

A non-overlapping triangular network resulting in a hexagonal mesh covered the domain. The width parameters at semi-bandwidths m and p were chosen to be $\ell_1 = \ell_2 = 3$ and the "fill-in" parameters were set to $r_1 = r_2 = 2$. The initial guess was $u^{(0)} = 1.0$.

The quasi-linearized Newton scheme is of the following form:

$$-L_h u^{(k+1)} + \frac{h^2}{\varepsilon_L}\alpha\beta e^{u^{(k)}} u^{(k+1)} = -\frac{h^2}{\varepsilon_L}\alpha e^{\beta u^{(k)}} + \frac{h^2}{\varepsilon_L} u^{(k)} \alpha\beta e^{\beta u^{(k)}}, \tag{98}$$

with L_h denoting the discretized finite element operator.

The termination criterion for the normalized explicit preconditioned conjugate gradient - type schemes was $\|r_i\|_\infty < 10^{-4}$, where r_i is the recursive residual.

The criterion for the termination of the Newton method was $\max_j \left| \left(u_j^{(k+1)} - u_j^{(k)} \right) / \left(1 + u_j^{(k+1)} \right) \right| < 10^{-4}$, $j \in [1,n]$.

Numerical results for the Newton compact iterations in

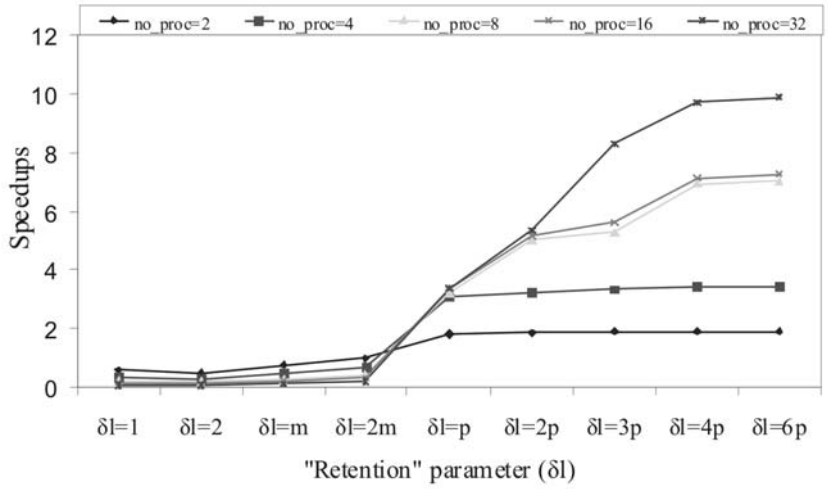


Figure 2 : Speedups versus the "retention" parameter δl of the **PNEPCGS** method for several numbers of processors, with $n = 8000$, $m = 21$ and $p = 401$, with $\ell_1 = \ell_2 = 3$ and $r_1 = r_2 = 2$.

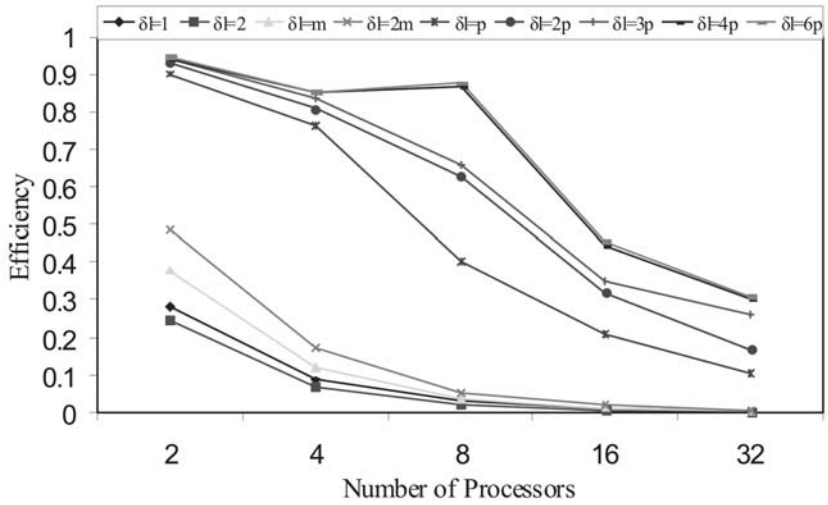


Figure 3 : Parallel efficiency of the **PNEPCGS** method for several values of δl, with $n = 8000$, $m = 21$ and $p = 401$, with $\ell_1 = \ell_2 = 3$ and $r_1 = r_2 = 2$.

conjunction with the **NEPCG** and **NEPCGS** method based on the **NOROBAIFEM-3D** algorithm, for several values of the parameters ε_L, α, β and the "retention" parameter δl, with $n = 3375$, $m = 16$, $p = 226$, $r_1 = r_2 = 2$ and $\ell_1 = \ell_2 = 3$, are presented in Tab. 2 and Tab. 3 respectively.

Model problem III: Let us consider the following non-linear parabolic P.D.E. in three space variables:

$$\varepsilon_t \frac{\partial u}{\partial t} - \varepsilon_L \Delta u = \alpha e^{\beta u}, \varepsilon_t, \varepsilon_L \to 0_+, (x,y,z) \in \Omega, t > 0, \quad (99)$$

with initial conditions:

$$u(x,y,z,0) = g(x,y,z), \quad 0 \le x,y,z \le 1, \quad (99.a)$$

and non-linear boundary conditions:

$$u(x,y,z,t) = 0, \quad t > 0. \quad (99.b)$$

A non-overlapping triangular network resulting in a hexagonal mesh covered the region, for the computation of an approximate solution of the perturbation problem. Then, by using backward differences for u_t and the finite

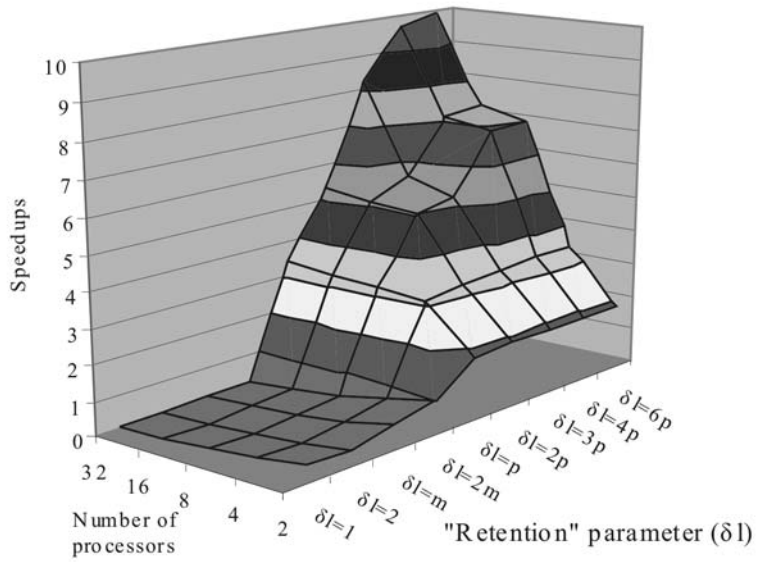


Figure 4 : Performance evaluation measurements of the **PNEPCGS** method, with $n = 8000$, $m = 21$ and $p = 401$, with $\ell_1 = \ell_2 = 3$ and $r_1 = r_2 = 2$.

Table 2 : The convergence behavior for solution of the **Newton** method in conjunction with the **NEPCG** method for various values of ε_L, α, β, δl, with $n = 3375$, $m = 16$, $p = 226$, $\ell_1 = \ell_2 = 3$ and $r_1 = r_2 = 2$.

ε_L	$\alpha = \beta$	**Newton method (outer iter.)**	**number of NEPCG iterations**				
			$\delta l = 1$	$\delta l = 2$	$\delta l = m$	$\delta l = 2m$	$\delta l = p$
1.0	1.0	3	21	21	18	17	15
	0.01	2	15	15	13	12	11
	0.000001	2	15	15	13	12	11
0.01	1.0	4	29	28	24	22	20
	0.01	2	15	15	13	12	11
	0.000001	2	15	15	13	12	11
0.000001	1.0	10	42	42	40	37	35
	0.01	3	26	26	23	20	19
	0.000001	2	15	15	13	12	11

element scheme, with a row-wise ordering used such that the width parameters ℓ_1, ℓ_2 of the bands were kept to low values, i.e. $\ell_1 = \ell_2 = 3$, then the resulting coefficient matrix is of the form given in (2). The "fill-in" parameters were set to $r_1 = r_2 = 2$. The initial guess was $u^{(0)} = 1.0$. The termination criterion for the normalized explicit preconditioned conjugate gradient - type schemes was $\|r_i\|_\infty < 10^{-4}$, where r_i is the recursive residual. The criterion for the termination of the Newton and the steady state solution was $\max_j \left| \left(u_j^{(k+1)} - u_j^{(k)} \right) / \left(1 + u_j^{(k+1)} \right) \right| < 10^{-4}$, $j \in [1, n]$.

The quasi-linearized Newton scheme is of the following form:

$$\frac{h^2}{\varepsilon_L} \left(\frac{\varepsilon_t}{\Delta t} - \alpha\beta e^{\beta u^{(k)}} \right) u_{i,j}^{(k+1)} - L_h u_{i,j}^{(k+1)} = \frac{h^2}{\varepsilon_L} \left(\frac{\varepsilon_t u^{(k)}}{\Delta t} + \left(1 - \beta u^{(k)} \right) \alpha e^{\beta u^{(k)}} \right), \tag{100}$$

with L_h denoting here the corresponding discretized finite element operator.

Numerical results for the steady state solution using backward differences combined with Newton compact iterations in conjunction with the **NEPCG** and **NEPCGS**

Table 3 : The convergence behavior for solution of the **Newton** method in conjunction with the **NEPCGS** method for various values of ε_L, α, β, δl, with $n = 3375$, $m = 16$, $p = 226$, $\ell_1 = \ell_2 = 3$ and $r_1 = r_2 = 2$.

ε_L	$\alpha = \beta$	**Newton method (outer iter.)**	**number of NEPCGS iterations**				
			$\delta l = 1$	$\delta l = 2$	$\delta l = m$	$\delta l = 2m$	$\delta l = p$
1.0	1.0	3	13	13	11	9	7
	0.01	2	9	9	8	6	6
	0.000001	2	9	9	8	6	6
0.01	1.0	4	17	16	15	12	10
	0.01	2	9	9	8	6	6
	0.000001	2	9	9	8	6	6
0.000001	1.0	10	18	18	17	16	15
	0.01	3	15	16	14	11	10
	0.000001	2	9	9	8	6	6

Table 4 : The convergence behavior for steady state solution of the **Newton** method in conjunction with the **NEPCG** method for various values of ε_t, ε_L, α, β, δl and the time step Δt, with $n = 3375$, $m = 16$, $p = 226$, $\ell_1 = \ell_2 = 3$ and $r_1 = r_2 = 2$.

ε_t	$\varepsilon_L = \alpha = \beta$	Δt	**no of outer iter for s.s.s.**	**Newton method inner iter.**	**number of NEPCG iterations**				
					$\delta l = 1$	$\delta l = 2$	$\delta l = m$	$\delta l = 2m$	$\delta l = p$
10^{-1}	1	0.05	3	6	23	23	21	18	17
		0.01	4	8	31	30	28	25	22
	0.1	0.05	4	7	28	28	24	22	19
		0.01	6	11	42	41	37	34	30
	0.01	0.05	7	13	50	50	44	41	38
		0.01	16	31	88	88*	87	81	74
10^{-2}	1	0.05	3	6	19	18	17	14	13
		0.01	3	6	21	21	21	18	15
	0.1	0.05	3	5	18	18	16	14	13
		0.01	4	7	25	25	22	20	18
	0.01	0.05	4	7	28	27	24	22	19
		0.01	6	11	42	41	37	34	30
10^{-3}	1	0.05	2	4	11	10	10	10	7
		0.01	3	6	16	17	15	14	11
	0.1	0.05	3	5	14	14	13	11	10
		0.01	3	5	17	17	15	13	12
	0.01	0.05	3	5	18	17	15	14	13
		0.01	4	7	25	24	22	20	18
10^{-6}	1	0.05	2	4	9	9	9	8	7
		0.01	2	4	9	9	9	8	7
	0.1	0.05	2	3	9	9	8	7	6
		0.01	2	3	9	9	8	7	6
	0.01	0.05	2	3	8	8	8	7	6
		0.01	2	3	9	9	8	7	6

*The number of outer and inner (Newton method) iterations was 15 and 29 iterations respectively

method, based on the **NOROBAIFEM-3D** algorithm, for several values of the parameters ε_t, ε_L, α, β the time-step Δt and the "retention" parameter δl, with $n = 3375$, $m = 16$, $p = 226$, $r_1 = r_2 = 2$ and $\ell_1 = \ell_2 = 3$, are pre-

Table 5 : The convergence behavior for steady state solution of the **Newton** method in conjunction with the **NEPCGS** method for various values of ε_t, ε_L, α, β, δl and the time step Δt, with $n = 3375$, $m = 16$, $p = 226$, $\ell_1 = \ell_2 = 3$ and $r_1 = r_2 = 2$.

ε_t	$\varepsilon_L = \alpha = \beta$	Δt	**no of outer iter for s.s.s.**	**Newton method inner iter.**	**number of NEPCGS iterations**				
					$\delta l = 1$	$\delta l = 2$	$\delta l = m$	$\delta l = 2m$	$\delta l = p$
10^{-1}	1	0.05	3	6	13	17	12	11	9
		0.01	4	8	19	18	18	13	13
	0.1	0.05	4	7	18	18	16	14	12
		0.01	6	11	27	28	25	20	20
	0.01	0.05	7	13	32	32	28	24	22
		0.01	16	31	53	55	52	48	43
10^{-2}	1	0.05	3	6	10*	11	10	9	9
		0.01	3	6	12*	15	10	9	9
	0.1	0.05	3	5	11	13	10	9	7
		0.01	4	7	17	15	15	11	11
	0.01	0.05	4	7	18	18	16	14	12
		0.01	6	11	27	28	25	20	20
10^{-3}	1	0.05	2	4	6**	7	5	5	5
		0.01	3	6	8*	9	8	8	7
	0.1	0.05	3	5	8○	10	8	8	7
		0.01	3	5	9○	12	8	8	7
	0.01	0.05	3	5	11	13	10	9	7
		0.01	4	7	17	15	15	11	11
10^{-6}	1	0.05	2	4	6**	7	5	5	5
		0.01	2	4	6**	7	5	5	5
	0.1	0.05	2	3	5	6	4	4	4
		0.01	2	3	5○○	6	4	4	4
	0.01	0.05	2	3	5○○	6	4	4	4
		0.01	2	3	5○○	6	4	4	4

* The number of inner iterations (Newton method) was 7 iterations
** The number of inner iterations (Newton method) was 5 iterations
○ The number of inner iterations (Newton method) was 6 iterations
○○ The number of inner iterations (Newton method) was 4 iterations

sented in Tab. 4 and Tab. 5 respectively.

It should be noted that the convergence behavior of the **NEPCG** and **NEPCGS** method for the "retention" parameter $\delta l = 2p$ was almost of the same order as for the "retention" parameter $\delta l = p$.

Finally, the parallel normalized explicit approximate inverse finite element preconditioning methods can be efficiently used for solving three dimensional highly nonlinear elliptic and parabolic partial differential equations.

Acknowledgement: The authors would like to thank indeed Professor M.P. Bekakos, Director of the Digital Systems Laboratory of Democritus University of Thrace for proposing to use the Beowulf cluster of University of Ioannina, and Professor I.E. Lagaris, Department of Computer Science, University of Ioannina for allowing us to use the Beowulf cluster.

References

Akl, S. (1997): *Parallel Computation: Models and Methods*. Prentice Hall.

Benzi, M. (2002): Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, vol. 182, pp. 418–477.

Benzi, M.; Meyer, C.; Tuma, M. (1996): A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, vol. 17, pp. 1135–1149.

Dongarra, J.; Duff, I.; Sorensen, D.; van der Vorst, H. (1998): *Numerical Linear Algebra for High-Performance Computers*. SIAM.

Gravvanis, G. (1998): Parallel matrix techniques. In Papailiou, K.; Tsahalis, D.; Periaux, J.; Hirsch, C.; Pandolfi, M.(Eds): *Computational Fluid Dynamics*, volume I, pp. 472–477. Wiley.

Gravvanis, G. (2002): Explicit Approximate Inverse Preconditioning Techniques. *Archives of Computational Methods in Engineering*, vol. 9(4), pp. 371–402.

Gravvanis, G.; Giannoutakis, K. (2003): Normalized Explicit Finite Element Approximate Inverses. *I. J. Differential Equations and Applications*, vol. 6(3), pp. 253–267.

Gravvanis, G.; Giannoutakis, K. (2004): On the rate of convergence and computational complexity of normalized implicit preconditioning for solving finite difference equations in three space variables. *I. J. of Computational Methods*, vol. 1(2), pp. 367–386.

Gravvanis, G.; Giannoutakis, K. (2005): Normalized implicit preconditioned methods based on normalized finite element approximate factorization procedures. In K.J.Bathe(Ed): *Computational Fluid and Solid Mechanics 2005*, volume 2, pp. 1115–1119. Elsevier.

Greenbaum, A. (1997): *Iterative methods for solving linear systems*. SIAM.

Grote, M.; Huckle, T. (1997): Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, vol. 18, pp. 838–853.

Huckle, T. (1999): Approximate sparsity patterns for the inverse of a matrix and preconditioning. *Applied Numerical Mathematics*, vol. 30, pp. 291–303.

Lipitakis, E.; Evans, D. (1984): Solving linear finite element systems by normalized approximate matrix factorization semi-direct methods. *Computer Methods in Applied Mechanics & Engineering*, vol. 43, pp. 1–19.

Pacheco, P. (1997): *Parallel programming with MPI*. Morgan Kaufmann Publishers.

Quinn, M. (2003): *Parallel Programming in C with MPI and OpenMP*. Mc-Graw Hill.

Saad, Y. (1996): *Iterative methods for sparse linear systems*. PWS Publishing.

Saad, Y.; van der Vorst, H. (2000): Iterative solution of linear systems in the 20th century. *J. Comp. Applied Math*, vol. 123, pp. 1–33.

van der Vorst, H. (1989): High performance preconditioning. *SIAM J. Sci. Stat. Comput.*, vol. 10, pp. 1174–1185.