 CMES, vol.71, no.4, pp.305-330, 2011

Finite Element Approximate Inverse Preconditioning for solving 3D Biharmonic Problems on Shared Memory Systems

G.A. Gravvanis[1], and K.M. Giannoutakis[2]

Abstract: In this paper we present parallel explicit approximate inverse matrix techniques for solving sparse linear systems on shared memory systems, which are derived using the finite element method for biharmonic equations in three space variables. Our approach for solving such equations is by considering the biharmonic equation as a "coupled equation approach" (pair of Poisson equation), using a FE approximation scheme, yielding an "inner-outer" iteration method. Additionally, parallel approximate inverse matrix algorithms are introduced for the efficient solution of sparse linear systems, based on an anti-diagonal computational approach that eliminates the data dependencies. Parallel explicit preconditioned conjugate gradient-type schemes in conjunction with parallel approximate inverse matrix algorithms are presented for the efficient solution of sparse linear systems. Theoretical estimates on computational complexity of the parallel explicit preconditioned conjugate gradient method along with theoretical speedups and efficiency are also presented. Applications of the proposed methods on characteristic biharmonic problems are discussed and numerical results are given.

Keywords: Biharmonic equations, finite element method, sparse linear systems, approximate factorization procedures, parallel approximate inverse matrix algorithms, parallel preconditioned conjugate gradient methods, shared memory systems.

[1] Department of Electrical and Computer Engineering, School of Engineering, Democritus University of Thrace, University Campus, Kimmeria, GR 67100, Xanthi, Greece

[2] Centre for Research and Technology Hellas, Informatics and Telematics Institute, 6th km Charilaou-Thermi, GR 57001, Thessaloniki, Greece

1 Introduction

Let us consider a class of problems defined by the following biharmonic equation in three space variables:

$$\nabla^4 u(x,y,z) = \frac{\partial^4 u}{\partial x^4} + \frac{\partial^4 u}{\partial y^4} + \frac{\partial^4 u}{\partial z^4} + 2\frac{\partial^4 u}{\partial x^2 \partial y^2} + 2\frac{\partial^4 u}{\partial x^2 \partial z^2} + 2\frac{\partial^4 u}{\partial y^2 \partial z^2} =$$

$$f(x,y,z), (x,y,z) \in R, \tag{1}$$

where R is a three dimensional bounded domain, subject to the boundary conditions:

$$u(x,y,z) = g_1(x,y,z), (x,y,z) \in \partial R, \tag{2}$$

$$\frac{\partial u(x,y,z)}{\partial \eta} = g_2(x,y,z), (x,y,z) \in \partial R, \tag{3}$$

and $\partial/\partial\eta$ is the derivative in the direction of the outward normal to the boundary, while f and g_1, g_2 are given functions defined on R and ∂R respectively. Important applications in engineering and science are described by such equations, which occur in elasticity and in fluid flow, etc.

By considering the Finite Element (FE) method, in general we have to solve a large sparse linear system of algebraic equations. Hence sparse matrix computations, which have inherent parallelism, are therefore of central importance in scientific and engineering computing and furthermore the need for high performance computing, which is about 70% of supercomputer time, has had some effect on the design of modern computer systems.

Methods for solving biharmonic equations on a rectangular region have been discussed by many researchers, [Axelsson (1973); Buzbee and Dorr (1974); Ehrlich (1973, 1971); Gravvanis and Giannoutakis (2005); Greenspan and Schultz (1972); Nodera and Takahashi (1981); Smith (1968); Yousif and Evans (1993)], and several iterative methods have been examined either considering the biharmonic equation as a "coupled equation approach" (pair of Poisson equation) or by applying iterative schemes directly to the fourth order equation.

Our approach is to consider the "coupled equation approach" viz $\nabla^4 = \nabla^2\nabla^2$, by solving

$$c\nabla^2 u = v \text{ and } \nabla^2 v = cf, \tag{4}$$

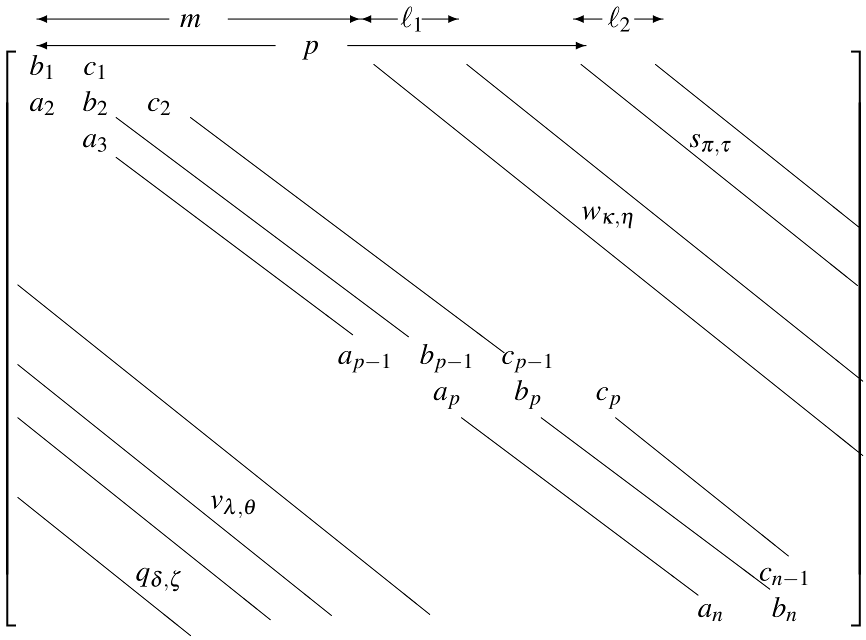
where two discrete Poisson equations, using the FE method, must be solved yielding an "inner-outer" iteration method, [Ehrlich (1971); Smith (1968)].

Let us consider the resulting finite element linear system for the discrete Poisson equation, i.e.,

$$Au = s \tag{5}$$

where A is a non-singular sparse unsymmetric $(n \times n)$ matrix of irregular non-zero structure (where all the off-center band terms are grouped in regular bands of width ℓ_1 and ℓ_2 at semi-bandwidths m and p respectively), Eq. 6, while u is a FE 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.

$A \equiv$



(6)

An important achievement over the last decades is the appearance and use of Preconditioning Methods for the numerical solution of Partial Differential Equations, [Benzi, Meyer, and Tuma (1996); Evans (1983); Saad (1996)]. The well known

preconditioning methods based on Incomplete factorization techniques or Approximate Inverses by minimizing the Frobenious norm of the error or the residual for fixed sparsity pattern, [Benzi, Meyer, and Tuma (1996); Kolotilina and Yeremin (1993); Saad (1996); Saad and van der Vorst (2000)], are very difficult to implement them on parallel systems.

The emergence of Krylov subspace methods, based on projection methods, has placed explicit preconditioned generalized conjugate gradient - type methods in much competitive demand for solving sparse systems, [Saad and van der Vorst (2000)]. The suitable preconditioner M required is an optimized form of the explicit finite element generalized approximate inverse matrix algorithms for computing explicitly various families of approximate inverses of the preconditioned linear system:

$$MAu = Ms \tag{7}$$

The preconditioner M has therefore 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, Meyer, and Tuma (1996); Evans (1983); Gravvanis (2002); Grote and Huckle (1997); Huckle (1999, 1998); Kolotilina and Yeremin (1993); Saad (1996); Saad and van der Vorst (2000)].

Our main motive is the derivation of suitable inherent parallel preconditioned methods for which several forms of the approximate inverse are produced, according to a "fish-bone" computational procedure by using either the "Location-Principle" or the "Magnitude- Principle". Such preconditioning iterative methods are particularly suitable on parallel systems. Optimized forms of the approximate inverse algorithm, in which both sparseness of the original matrix is relatively retained and storage requirements are substantially reduced, have been efficiently used for solving sparse systems.

The challenge encountered when computing parallel approximate inverses is its internal data dependencies, which create both a critical path and an order of computations, such that any computational strategy adopted should abide by those dependencies. For the parallel construction of the approximate inverse preconditioner, a transformation of the sequential "fish-bone" pattern to an antidiagonal wave pattern has been carried out in order to overcome the data dependencies. The elements located on an antidiagonal are independent, as the computation of each element requires at least its right element and can be computed concurrently by the available processors. Thus, a consecutive antidiagonal movement through the banded

matrix would eliminate all dependencies. The computation of each antidiagonal is assigned to the available processors by continuous blocks of elements. The degree of parallelism depends on the "retention" parameter (length of the antidiagonal) and the number of processors. By increasing the value of the "retention" parameter, the workload per process overcomes the parallelization overheads, and the obtained speedups tend to the upper theoretical bound. The inherently parallel linear operations between vectors and matrices involved in the explicit preconditioned conjugate gradient schemes exhibit significant amounts of loop-level parallelism that can lead to high performance gain on shared address space systems, [Giannoutakis and Gravvanis (2009); Gravvanis and Giannoutakis (2008)].

The cost-effectiveness of explicit preconditioned iterative schemata over parallel direct solution methods, [Gravvanis (2002); Saad and van der Vorst (2000)], for solving sparse linear systems is now commonly accepted especially for three dimensional problems. It is known that approximate factorization procedures and inverse matrix algorithms are in general complicated. However as the demand for solving biharmonic equations 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, 2000, 1999a, 1997); Gravvanis and Giannoutakis (2005); Gravvanis and Lipitakis (1996a); Lipitakis and Gravvanis (1995)].

In Section 2, we present a new class of finite element parallel approximate inverse matrix algorithms, based on an antidiagonal wave pattern in order to overcome the data dependencies. In Section 3, explicit preconditioned conjugate gradient type methods along with theoretical results on the computational complexity are presented. Additionally, parallel explicit preconditioned conjugate gradient methods with theoretical estimates on the speedup and efficiency are also given. Finally in Section 4, the performance and applicability of the proposed methods, are illustrated by solving characteristic 3D biharmonic problem, is discussed and numerical results are given.

2 Parallel approximate inverse finite element matrix techniques

In this section we present parallel explicit approximate inverse matrix techniques for inverting a sparse $(n \times n)$ matrix A by computing the elements of a class of inverses, [Gravvanis (2009, 2002, 1999a); Gravvanis and Giannoutakis (2008); Gravvanis and Lipitakis (1996a,b); Lipitakis and Gravvanis (1995)].

Let us now consider the approximate factorization of the coefficient matrix A, [Gravvanis (1999b)], i.e.:

$$A \cong L_{r_1,r_2} U_{r_1,r_2}, r_1 \in [1,\ldots,m-1), r_2 \in [1,\ldots,p-1). \tag{8}$$

where r_1, r_2 are the so-called "fill-in" parameters, i.e. the number of outermost off-diagonal entries retained in semi-bandwidth m and p respectively, and L_{r_1,r_2}, U_{r_1,r_2} are sparse strictly lower and upper (with main diagonal unity elements) triangular matrices respectively (of the same profile as the coefficient matrix A), as shown in Eq. 9 and Eq. 10.

The elements of the L_{r_1,r_2}, U_{r_1,r_2} factors can be obtained by using the Finite Element Approximate LU-type factorization procedure (henceforth called FEALUFA-3D algorithm), [Gravvanis (1999b)]. The memory requirements of the FEALUFA-3D algorithm are $\approx O\left(2r_1+2r_2+4\ell_1+4\ell_2+3\right)n$ words, while the computational work required by the factorization process is $O\left[2\left(r_1+\ell_1-1\right)^2+2\left(r_2+\ell_2-1\right)^2+3\right]n$ multiplicative operations), [Gravvanis (1999b)].

Efficient computational implementation issues for the storage techniques of the coefficient matrix and the decomposition factors and in particular the submatrices involved have been given in [Gravvanis (1999a)]. Stability analysis of LU-type factorization has been studied in [Elman (1986)].

$$L_{r_1,r_2} \equiv \begin{bmatrix}
\omega_1 & & & & & & & & & & & \\
\beta_1 & \omega_2 & & & & & & & & & & \\
 & \beta_2 & & & & & & & & & & \\
 & & & & & & & & & & & \\
e_{1,1} & \cdots & e_{r_1,1} & & & & & & & & & \\
\vdots & & & & & 0 & & & & & & \\
e_{1,\ell_1} & & & & & & & & & & & \\
 & & & & & & \beta_{p-2} & \omega_{p-1} & & & & \\
 & & & & & & & \beta_{p-1} & \omega_p & & & \\
 & & & & & & & & & & & \\
f_{1,1} & \cdots & f_{r_2,1} & & & & & & & & & \\
\vdots & & & & & & & & & & & \\
f_{1,\ell_2} & & & & & & & & & & & \\
 & f_{1,n-p+1} & \cdots & f_{r_2+\ell_2-1,n-p+1} & & e_{1,n-m+1} & \cdots & e_{r_1+\ell_1-1,n-m+1} & & \beta_{n-1} & \omega_n
\end{bmatrix} \tag{9}$$

$$U_{r_1,r_2} \equiv \begin{bmatrix}
1 & g_1 & h_{1,1} & \cdots & h_{1,\ell_1} & t_{1,1} & \cdots & t_{1,\ell_2} & & \\
 & 1 & g_2 & \vdots & & \vdots & & & t_{1,n-p+1} & \\
 & & & h_{r_1,1} & & t_{r_2,1} & & & \vdots & \\
 & & & & & & & & & t_{r_2+\ell_2-1,n-p+1} \\
 & & & & 1 & g_{p-1} & & & & \\
 & & & & & 1 & g_p & & & \\
 & & & & & & & & & h_{1,n-m+1} \\
 & & 0 & & & & & & & \vdots \\
 & & & & & & & & & h_{r_1+\ell_1-1,n-m+1} \\
 & & & & & & & & & g_{n-1} \\
 & & & & & & & & & 1
\end{bmatrix} \tag{10}$$

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 approximate inverse of the coefficient matrix A. Then, the elements of a class of banded forms of the approximate inverse, by retaining δl and $\delta l - 1$ elements in the lower and upper part of the inverse, can be computed by solving recursively the systems:

$$M^{\delta l}_{r_1,r_2} = (L_{r_1,r_2} U_{r_1,r_2})^{-1}, \tag{11}$$

or equivalently

$$M^{\delta l}_{r_1,r_2} L_{r_1,r_2} = (U_{r_1,r_2})^{-1} \text{ and } U_{r_1,r_2} M^{\delta l}_{r_1,r_2} = (L_{r_1,r_2})^{-1}, \delta l \in [1, \rho p], \tag{12}$$

where $\rho = 1, 2, \ldots, p-1$. Then, the elements of the optimized approximate inverse, by using an optimized storage scheme, [Gravvanis (2000, 1999a, 1997); Gravvanis and Lipitakis (1996b)], were computed by the so-called Optimized Generalized Approximate Inverse Finite Element Matrix algorithmic procedure (OGAIFEM-3D algorithm), [Gravvanis (1997)].

The memory requirements of the OGAIFEM-3D algorithm are $n \times (2\delta l - 1)$ words and the computational work involved is of $\approx O[n \times (2\delta l - 1) \times (2r_1 + 2r_2 + 2\ell_1 + 2\ell_2 + 1)]$ multiplicative operations, [Gravvanis (1997)]. This form of the optimized approximate inverse is particularly effective for solving "banded" sparse FE

systems of large order, i.e. $\delta l < n/2$, or "narrow-banded" sparse FE systems of very large order, i.e. $\delta l << n/2$, [Gravvanis (2002, 1996); Lipitakis and Gravvanis (1995)].

According to the proposed computational strategy this class of approximate inverses can be considered that includes various families of approximate inverses having in mind the desired requirements of accuracy, storage and computational work as can be seen by the following diagrammatic scheme, [Gravvanis (1999a, 1997); Lipitakis and Gravvanis (1995)]:

$$A^{-1} \equiv M \leftarrow \underline{\text{class I}} \quad \tilde{M}^{\delta l}_{r_1=m-1,r_2=p-1} \leftarrow \underline{\text{class II}} \quad M^{\delta l}_{r_1=m-1,r_2=p-1} \leftarrow \underline{\text{class III}} \quad M^{\delta l}_{r_1,r_2} \leftarrow \underline{\text{class IV}} \quad M_i \tag{13}$$

where the entries of $\tilde{M}^{\delta l}_{r_1=m-1,r_2=p-1}$ have been retained after the computation of the exact inverse $(r_1 = m-1, r_2 = p-1)$, while the entries of $M^{\delta l}_{r_1=m-1,r_2=p-1}$ have been computed and retained during the computational procedure of the (approximate) inversion. The entries of $M^{\delta l}_{r_1,r_2}$ have been retained after the computation of the approximate inverse $(r_1 < m-1, r_2 < p-1)$. The M_i class of inverse retains only the diagonal elements of the pseudo-inverse, i.e. $\delta l = 1$, that is we invert the diagonal elements of L_{r_1,r_2}, i.e. a fast inverse algorithm.

Note that the largest in magnitude elements of the approximate inverse matrix are clustered around the diagonals at distances $\rho_1 m$ and $\rho_2 p$, ($\rho_1 1 = 1,2,\ldots,m-1$ and $\rho_2 = 1,2,\ldots,p-1$), from the main diagonal in a "recurring wave"-like pattern, [Gravvanis (1999a, 1997); Gravvanis and Lipitakis (1996a,b); Lipitakis and Gravvanis (1995)]. Therefore, it is reasonable to assume, the value of the "retention" parameter δl can be chosen as multiples of the semi-bandwidths m and p.

It should be also noted that if $\ell_1 = \ell_2 = 1$, Eq. 6, the algorithm is reduced to one for solving unsymmetric finite difference linear systems of semi-bandwidth m and p, which is usually encountered in solving 3D boundary value problems by FD discretization, [Gravvanis and Lipitakis (1996b)]. If $\ell_2 = 0$, Eq. 6, the algorithm is reduced to one for solving unsymmetric finite element linear systems of semi-bandwidth m, which is usually encountered in solving 2D boundary value problems by FE discretization, [Gravvanis and Lipitakis (1996a); Lipitakis and Gravvanis (1995)]. If $\ell_1 = 1$ and $\ell_2 = 0$, Eq. 6, the algorithm is reduced to one for solving unsymmetric finite difference linear systems of semibandwidth m, which is usually encountered in solving 2D boundary value problems by FD discretization, [Gravvanis (2002, 2000)]. In general, by setting appropriately the values of the semibandwidth parameters m and p as well as the values of the width parameters ℓ_1 and ℓ_2 of the coefficient matrix A, Eq. 6, then the presented algorithmic approach can be used for solving linear systems resulting from more general type problems.

For the parallelization of the OGAIFEM-3D algorithm, an antidiagonal motion (wave-like pattern), starting from the element $\mu_{n,n}$ down to $\mu_{1,1}$, has been used, because of the dependency of the elements of the inverse during its construction, [Giannoutakis and Gravvanis (2009); Gravvanis and Giannoutakis (2008)]. More specifically, any element within the banded approximate inverse requires its corresponding right or lower element to be computed first. This sequence of computations, without any loss of generality and for simplicity reasons, is shown for the banded approximate inverse in Eq. 14, Eq. 15, (with $n = 8$, $\delta l = 4$). The values of the parentheses at the superscript of each element (e.g. $\mu_{i,j}^{(k)}$), indicate that the element $\mu_{i,j}$ was computed at the (k)-th sequential step of the algorithm (k-th antidiagonal), while the elements with the same superscript (i.e. (k)) were computed concurrently.

Banded Approximate Inverse

$$M_{r_1,r_2}^{\delta l} = \begin{bmatrix}
\overleftarrow{\qquad} \delta l \overrightarrow{\qquad} & & & & & & & \\
\mu_{1,1}^{(15)} & \mu_{1,2}^{(14)} & \mu_{1,3}^{(13)} & \mu_{1,4}^{(12)} & & & & \\
\mu_{2,1}^{(14)} & \mu_{2,2}^{(13)} & \mu_{2,3}^{(12)} & \mu_{2,4}^{(11)} & \mu_{2,5}^{(10)} & & & \\
\mu_{3,1}^{(13)} & \mu_{3,2}^{(12)} & \mu_{3,3}^{(11)} & \mu_{3,4}^{(10)} & \mu_{3,5}^{(9)} & \mu_{3,6}^{(8)} & & \\
\mu_{4,1}^{(12)} & \mu_{4,2}^{(11)} & \mu_{4,3}^{(10)} & \mu_{4,4}^{(9)} & \mu_{4,5}^{(8)} & \mu_{4,6}^{(7)} & \mu_{4,7}^{(6)} & \\
 & \mu_{5,2}^{(10)} & \mu_{5,3}^{(9)} & \mu_{5,4}^{(8)} & \mu_{5,5}^{(7)} & \mu_{5,6}^{(6)} & \mu_{5,7}^{(5)} & \mu_{5,8}^{(4)} \\
 & & \mu_{6,3}^{(8)} & \mu_{6,4}^{(7)} & \mu_{6,5}^{(6)} & \mu_{6,6}^{(5)} & \mu_{6,7}^{(4)} & \mu_{6,8}^{(3)} \\
 & & & \mu_{7,4}^{(6)} & \mu_{7,5}^{(5)} & \mu_{7,6}^{(4)} & \mu_{7,7}^{(3)} & \mu_{7,8}^{(2)} \\
 & & & & \mu_{8,5}^{(4)} & \mu_{8,6}^{(3)} & \mu_{8,7}^{(2)} & \mu_{8,8}^{(1)}
\end{bmatrix} \tag{14}$$

For the parallel construction of the optimized form of the approximate inverse, as diagrammatically shown in Eq. 14, Eq. 15, a simple transformation of the indexes of the elements of the approximate inverse is used, [Gravvanis (2002, 1999b); Gravvanis and Lipitakis (1996a)].

$$M^{\delta l}_{r_1,r_2} = \overbrace{\longleftarrow \quad \delta l \quad \longrightarrow}^{\text{Optimized Approximate Inverse}} \left[\begin{array}{cccc|ccc} \mu^{(1)}_{8,8} & \mu^{(2)}_{8,7} & \mu^{(3)}_{8,6} & \mu^{(4)}_{8,5} & \mu^{(2)}_{7,8} & \mu^{(3)}_{6,8} & \mu^{(4)}_{5,8} \\ \mu^{(3)}_{7,7} & \mu^{(4)}_{7,6} & \mu^{(5)}_{7,5} & \mu^{(6)}_{7,4} & \mu^{(4)}_{6,7} & \mu^{(5)}_{5,7} & \mu^{(6)}_{4,7} \\ \mu^{(5)}_{6,6} & \mu^{(6)}_{6,5} & \mu^{(7)}_{6,4} & \mu^{(8)}_{6,3} & \mu^{(6)}_{5,6} & \mu^{(7)}_{4,6} & \mu^{(8)}_{3,6} \\ \mu^{(7)}_{5,5} & \mu^{(8)}_{5,4} & \mu^{(9)}_{5,3} & \mu^{(10)}_{5,2} & \mu^{(8)}_{4,5} & \mu^{(9)}_{3,5} & \mu^{(10)}_{2,5} \\ \mu^{(9)}_{4,4} & \mu^{(10)}_{4,3} & \mu^{(11)}_{4,2} & \mu^{(12)}_{4,1} & \mu^{(10)}_{3,4} & \mu^{(11)}_{2,4} & \mu^{(12)}_{1,4} \\ \mu^{(11)}_{3,3} & \mu^{(12)}_{3,2} & \mu^{(13)}_{3,1} & & \mu^{(12)}_{2,3} & \mu^{(13)}_{1,3} & \\ \mu^{(13)}_{2,2} & \mu^{(14)}_{2,1} & & & \mu^{(14)}_{1,2} & & \\ \mu^{(15)}_{1,1} & & & & & & \end{array}\right] \tag{15}$$

Let us consider that the command forall denotes the parallel for instruction (forks / joins threads), for executing parallel loops. Then, the algorithm for the implementation of the Parallel ANti Diagonal OGAIFEM-3D algorithm (henceforth called the PANDOGAIFEM- 3D algorithm), on shared memory computer systems, can be described as follows:

```
// lower triangle-shaped zone
for k = 1 to δl
        forall ℓ = 1 to k
                call inverse(n − ℓ + 1, n − k + ℓ)
m = 2
// middle antidiagonal length zone
for k = (δl + 1) to n
        forall ℓ = m to (k − m + 1)
                call inverse(n − ℓ + 1, n − k + ℓ)
        if (k − δl) mod 2 = 0 then
                m = m + 1
m = m − 1
for k = (n − 1) downto (δl + 1)
        forall ℓ = m to (k − m + 1)
                call inverse(ℓ, k − ℓ + 1)
        if (k − δl) mod 2 = 1 then
                m = m − 1
// upper triangle-shaped zone
for k = δl downto 1
```

forall $\ell = 1$ to k
call inverse$(\ell, k-\ell+1)$

where the function inverse(i,j), computes the element $\mu_{i,j}$ according to the OGAIFEM-3D algorithm:

$r\ell 1 = r_1 + \ell_1$, $r\ell 2 = r_2 + \ell_2$, $r\ell 11 = r\ell 1 - 1$, $r\ell 21 = r\ell 2 - 1$, $mr1 = m - r_1$
$pr2 = p - r_2$, $m\ell 1 = m + \ell_1$, $p\ell 2 = p + \ell_2$, $nmr1 = n - m + r_1$, $npr2 = n - p + r_2$
if $i \geq j$ then
  if $j > nmr1$ then
    if $i = j$ then
      if $j = n$ then
        $\mu_{1,1} = 1/\omega_n$,
      else
        $$\mu_{n-i+1,i-j+1} = \left(1 - \beta_j \mu_{n-j,\delta l-i+j+1}\right)/\omega_j \tag{16}$$
    else
      $$\mu_{n-i+1,i-j+1} = -\beta_j \mu_{n-i+1,i-j}/\omega_j \tag{17}$$
  else
    if $j > npr2$ and $j \leq nmr1$ then
      if $i = j$ then
        $$\mu_{n-i+1,i-j+1} = \Big(1 - \beta_j \mu_{n-j,\delta l-i+j+1} - \sum_{k=0}^{nmr1-j} e_{r\ell 11-k,j+k+1-r_1} \mu_{x,y}\Big)/\omega_j \tag{18}$$
        call rs$(n, \delta l, i, j+mr1+k, x, y)$
      else
        $$\mu_{n-i+1,i-j+1} = \left(-\beta_j \mu_{n-i+1,i-j} - \sum_{k=0}^{nmr1-j} e_{r\ell 11-k,j+k+1-r_1} \mu_{x,y}\right)/\omega_j \tag{19}$$
        call rs$(n, \delta l, i, j+mr1+k, x, y)$
    else
      if $j \leq npr2$ and $j \geq r\ell 1$ then
        if $i = j$ then
          $$\mu_{n-i+1,i-j+1} = \left(1 - \beta_j \mu_{n-j,\delta l-i+j+1} - \sum_{k=0}^{nmr1-j} e_{r\ell 11-k,j+k+1-r_1} \mu_{x_1,y_1} - \sum_{k=0}^{npr2-j} f_{r\ell 21-k,j+k+1-r_2} \mu_{x_2,y_2}\right)/\omega_j \tag{20}$$
          call rs$(n, \delta l, i, j+mr1+k, x_1, y_1)$ call rs$(n, \delta l, i, j+pr2+k, x_2, y_2)$
        else
          $$\mu_{n-i+1,i-j+1} = \left(-\beta_j \mu_{n-i+1,i-j} - \sum_{k=0}^{nmr1-j} e_{r\ell 11-k,j+k+1-r_1} \mu_{x_1,y_1}\right.$$

$$- \sum_{k=0}^{npr2-j} f_{r\ell 21-k,j+k+1-r_2}\mu_{x_2,y_2}\Bigg) / \omega_j \tag{21}$$

call rs$(n,\delta l,i,j+mr1+k,x_1,y_1)$ call rs$(n,\delta l,i,j+pr2+k,x_2,y_2)$

else

if $i=j$ then

if $i=1$ then

$$\mu_{1,1} = \Bigg(1-\beta_1\mu_{n-j,\delta l-i+j+1} - \sum_{k=1}^{\ell_1} e_{1,k}\mu_{n+2-m-k,\delta l+m+k-2} - \sum_{k=1}^{\ell_2} f_{1,k}\mu_{n+2-p-k,\delta l+p+k-2}\Bigg) / \omega_1 \tag{22}$$

else

$$\mu_{n-i+1,i-j+1} = \Bigg(1-\beta_j\mu_{n-j,\delta l-i+j+1} - \sum_{k=j+1-r_1}^{\ell_1} e_{j,k}\mu_{x_1,y_1} - \sum_{k=1}^{j-1} e_{j-k,\ell_1+k}\mu_{x_2,y_2} - \sum_{k=j+1-r_2}^{\ell_2} f_{j,k}\mu_{x_3,y_3} - \sum_{k=1}^{j-1} f_{j-k,\ell_2+k}\mu_{x_4,y_4}\Bigg) / \omega_j \tag{23}$$

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

else

$$\mu_{n-i+1,i-j+1} = \Bigg(-\beta_j\mu_{n-i+1,i-j} - \sum_{k=j+1-r_1}^{\ell_1} e_{j,k}\mu_{x_1,y_1} - \sum_{k=1}^{j-1} e_{j-k,\ell_1+k}\mu_{x_2,y_2} - \sum_{k=j+1-r_2}^{\ell_2} f_{j,k}\mu_{x_3,y_3} - \sum_{k=1}^{j-1} f_{j-k,\ell_2+k}\mu_{x_4,y_4}\Bigg) / \omega_j \tag{24}$$

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

if $i \le j$ then

if $j > nmr1$ then

$$\mu_{n-i+1,\delta l+i-j} = -g_j\mu_{x,y} \tag{25}$$

call rs$(n,\delta l,j+1,i,x,y)$

else

if $j > npr2$ and $j \le nmr1$ then

$$\mu_{n-i+1,\delta l+i-j} = -g_j\mu_{x_1,y_1} - \sum_{k=0}^{nmr1-j} h_{r\ell 11-k,j+k+1-r_1}\mu_{x_2,y_2} \tag{26}$$

call rs$(n,\delta l,j+1,i,x_1,y_1)$ call rs$(n,\delta l,j+mr1+k,i,x_2,y_2)$

else

if $j \le npr2$ and $j \ge r\ell 1$ then

$$\mu_{n-i+1,\delta l+i-j} = -g_j\mu_{x_1,y_1} - \sum_{k=0}^{nmr1-j} h_{r\ell 11-k,j+k+1-r_1}\mu_{x_2,y_2}$$

$$-\sum_{k=0}^{npr2-j} t_{r\ell 21-k,j+k+1-r_2}\mu_{x_3,y_3} \tag{27}$$
call rs$(n,\delta l,j+1,i,x_1,y_1)$ call rs$(n,\delta l,j+mr1+k,i,x_2,y_2)$
call rs$(n,\delta l,j+pr2+k,i,x_3,y_3)$

else

$$\mu_{n-i+1,\delta l+i-j} = -g_j\mu_{x_1,y_1} - \sum_{k=j+1-r_1}^{\ell_1} h_{j,k}\mu_{x_2,y_2} - \sum_{k=1}^{j-1} h_{j-k,\ell_1+k}\mu_{x_3,y_3}$$
$$-\sum_{k=j+1-r_2}^{\ell_2} t_{j,k}\mu_{x_4,y_4} - \sum_{k=1}^{j-1} t_{j-k,\ell_2+k}\mu_{x_5,y_5} \tag{28}$$
call rs$(n,\delta l,j+1,i,x_1,y_1)$ call rs$(n,\delta l,m+k-1,i,x_2,y_2)$
call rs$(n,\delta l,m\ell 1+k-1,i,x_3,y_3)$ call rs$(n,\delta l,p+k-1,i,x_4,y_4)$
call rs$(n,\delta l,p\ell 2+k-1,i,x_5,y_5)$

The procedure rs$(n,\delta l,s,q,x,y)$, [Gravvanis (2002, 2000)], can then be described as follows :

if $s \geq q$ then
$$x = n+1-s \tag{29}$$
$$y = s-q+1 \tag{30}$$
else
$$x = n+1-q \tag{31}$$
$$y = \delta l+q-s \tag{32}$$

The computational process is logically divided into $2n-1$ sequential steps representing the $2n-1$ antidiagonals in a matrix of order n, while synchronization between processes is needed after the computation of each antidiagonal, to ensure that the elements of the matrix are computed correctly. The workload on each antidiagonal varies between 1 and δl elements for the lower and upper triangle-shaped zones, while for the middle antidiagonal length zone interchanges between $\delta l-1$ and δl elements, see Eq. 14, Eq. 15. Thus, the parallel computational complexity for the lower or upper triangle-shaped zones is $\sum_{i=1}^{\delta l}\left\lceil \frac{i}{no\_procs}\right\rceil O(2r_1+2r_2+2\ell_1+2\ell_2+1)$ multiplications, while for the middle zone is $(2n-2\delta l-1)\left\lceil \frac{\delta l}{no\_procs}\right\rceil O(2r_1+2r_2+2\ell_1+2\ell_2+1)$ multiplications, where *no_procs* denotes the number of processors. Since the elements of each, which can lead to low efficiencies on some platforms where excessively fine granularity antidiagonal are partitioned between the processors (*no_procs*), adding more processors results in finer granularity makes it harder to amortize the parallelization overheads.

The theoretical speedup and efficiency of the PAND-OGAIFEM-3D algorithm are:

$$S_p^{\delta l} = \frac{t_{serial}}{t_{parallel}} = \frac{1}{\frac{1}{no\_procs} + \frac{t_1}{\delta l O(2r_1+2r_2+2\ell_1+2\ell_2+1)}} \tag{33}$$

and

$$E_p^{\delta l} = \frac{1}{1 + \frac{t_1 no\_procs}{\delta l O(2r_1+2r_2+2\ell_1+2\ell_2+1)}} \tag{34}$$

where t_1 is the latency of one fork / join operation and t_m denotes the computational time of one multiplication. It is obvious that for $\delta l \to \infty$ then $S_p^{\delta l} \to no\_procs$ and $E_p^{\delta l} \to 1$.

3 Explicit preconditioned conjugate gradient methods

In this section we present a class of explicit preconditioned conjugate gradient-type schemes based on the derived Optimized Generalized Approximate Inverse Finite Element Matrix algorithm (OGAIFEM-3D algorithm) of Section 2. The use of the approximate inverse matrix techniques in the preconditioned conjugate gradient schemes eliminates the implicitness due to the forward-backward substitutions required and allows the derivation of parallel explicit preconditioned conjugate gradient schemes, [Giannoutakis and Gravvanis (2009); Gravvanis and Giannoutakis (2008, 2005)].

The Explicit Preconditioned Generalized Conjugate Gradient Square (EPGCGS) algorithm can be expressed by the following compact algorithmic scheme:

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

$$\text{set} \quad u_0 = 0, \text{ and } e_0 = 0, \tag{35}$$
$$\text{solve} \quad r_0 = M_{r_1,r_2}^{\delta l}(s - Au_0), \tag{36}$$
$$\text{set} \quad \sigma_0 = r_0, \text{ and } p_0 = (\sigma_0, r_0). \tag{37}$$

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:

$$\text{form} \quad q_i = A\sigma_i, \tag{38}$$
$$\text{calculate} \quad \alpha_i = p_i / \left(\sigma_0, M_{r_1,r_2}^{\delta l} q_i\right), \tag{39}$$
$$\text{compute} \quad e_{i+1} = r_i + \beta_i e_i - \alpha_i M_{r_1,r_2}^{\delta l} q_i, \tag{40}$$
$$d_i = r_i + \beta_i e_i + e_{i+1}, \tag{41}$$

$$\text{form} \quad u_{i+1} = u_i + \alpha_i d_i, \text{ and } q_i = Ad_i, \tag{42}$$
$$\text{compute} \quad r_{i+1} = r_i - \alpha_i M^{\delta l}_{r_1,r_2} q_i, \tag{43}$$
$$\text{set} \quad p_{i+1} = (\sigma_0, r_{i+1}) \text{ and } \beta_{i+1} = p_{i+1}/p_i, \tag{44}$$
$$\text{compute} \quad \sigma_{i+1} = r_{i+1} + 2\beta_{i+1} e_{i+1} + \beta^2_{i+1}\sigma_i. \tag{45}$$

The computational complexity of the EPGCGS method is $\approx O[(4\delta l + 4\ell_1 + 4\ell_2 + 15)n \text{ mults } +8n \text{ adds}]\nu$ operations, where ν is the number of iterations required for convergence to a certain level of accuracy, while ℓ_1, ℓ_2 are the width parameters of the coefficient matrix A at semi-bandwidth m and p respectively.

In the following we present the Explicit Preconditioned Generalized BIconjugate Conjugate Gradient-STAB (EPGBICG-STAB) method, which can be expressed by the following compact scheme:

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

$$\text{set} \quad u_0 = 0 \quad \text{compute } r_0 = s - Au_0, \tag{46}$$
$$\text{set} \quad r'_0 = r_0, \rho_0 = \alpha = \omega_0 = 1 \text{ and } v_0 = p_0 = 0. \tag{47}$$

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:

$$\text{calculate } \rho_i = \left(r'_0, r_{i-1}\right), \tag{48}$$
$$\beta = (\rho_i/\rho_{i-1})/(\alpha/\omega_{i-1}), \tag{49}$$
$$\text{compute } p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1} v_{i-1}), \tag{50}$$
$$\text{form} \quad y_i = M^{\delta l}_{r_1,r_2} p_i, \; v_i = Ay_i, \tag{51}$$
$$\alpha = \rho_i / \left(r'_0, v_i\right), \tag{52}$$
$$\text{compute } x_i = r_{i-1} - \alpha v_i, \; z_i = M^{\delta l}_{r_1,r_2} x_i, \tag{53}$$
$$t_i = Az_i, \tag{54}$$
$$\text{set} \quad \omega_i = \frac{\left(M^{\delta l}_{r_1,r_2} t_i, M^{\delta l}_{r_1,r_2} x_i\right)}{\left(M^{\delta l}_{r_1,r_2} t_i, M^{\delta l}_{r_1,r_2} t_i\right)}, \tag{55}$$
$$\text{compute } u_i = u_{i-1} + \alpha y_i + \omega_i z_i \text{ and } r_i = x_i - \omega_i t_i. \tag{56}$$

The computational complexity of the EPGBICG-STAB method is $\approx O[(6\delta l + 4\ell_1 + 4\ell_2 + 14)n \text{ mults } +6n \text{ adds}]\nu$ operations, where ν is the number of iterations required for convergence to a certain level of accuracy.

The effectiveness of the explicit preconditioned conjugate gradient - type methods using the OGAIFEM-3D algorithm is related to the fact that the approximate inverse exhibits a similar "fuzzy" structure as the original coefficient matrix A and is a close approximant to the coefficient matrix A.

The basic properties of the preconditioned conjugate gradient method along with similar convergence analysis have been presented in [Gravvanis (1996); Notay (1993); van der Sluis and van der Vorst (1986)]. Then, the following Theorem on the rate of convergence and computational complexity of the EPGCGS method can be stated as follows:

Theorem 3.1 *Let $\Omega^{\delta l}_{r_1,r_2} = M^{\delta l}_{r_1,r_2} A$ be the preconditioning matrix of the Explicit Preconditioned Generalized Conjugate Gradient Square (EPGCGS) scheme. Suppose there exist positive numbers ξ_1, ξ_2, such that $[\xi_1, \xi_2] \supset [\lambda_{\min}, \lambda_{\max}]$, where ξ_1 is independent of the mesh size and $\xi_2 = O\left(n\left(\delta l + \ell_1 + \ell_2\right)^{-1}\right)$. Then, the number of iterations of the EPGCGS method required to reduce the L_∞ - norm of the error by a factor $\varepsilon > 0$ is given by*

$$\nu = O\left(kn^{1/2}\left(\delta l + \ell_1 + \ell_2\right)^{-1/2} \log \varepsilon^{-1}\right). \tag{57}$$

Furthermore, the total number of arithmetic operations required for the computation of the solution u_ν is given by

$$O\left(kn^{3/2}\left(\delta l + \ell_1 + \ell_2\right)^{-1/2} \log \varepsilon^{-1}\right). \tag{58}$$

It should be noted that according to the convergence analysis of the explicit approximate inverse preconditioning, the rate of convergence is improved when the value of the "retention" parameter δl is chosen as multiples of m and p, [Gravvanis (1996)]. It is evident that when the value of δl is chosen as multiples of m and p, then the required overall computational complexity can be prohibitively high. Thus the determination of the optimum value of the "retention" parameter δl is important and the value of $\delta l = 1$ is recommended, which gives the lowest overall computational complexity and minimum storage requirements. Hence, the class V of the approximate inverse requires only the diagonal elements of L_{r_1,r_2}, and the approximate inverse matrix-vector product in the CG-type method is reduced to a vector-vector product, which is optimal.

Let *no_procs* denote the number of processors available. Then, the two most computationally dominating operations of the explicit preconditioned conjugate gradient - type schemes (i.e. multiplication of the optimized approximate inverse with a vector and inner products), can be computed in parallel by partitioning the approximate inverse matrix and the vectors by a block - row distribution.

The Parallel Explicit Preconditioned Generalized Conjugate Gradient Square (PE PGCGS) algorithm can be expressed by the following compact scheme:

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

forall $j = 1$ to n

$$(r_0^*)_j = s_j - A(u_0)_j \tag{59}$$

forall $j = 1$ to n

$$(r_0)_j = \sum_{k=\max(1,j-\delta l+1)}^{\min(n,j+\delta l-1)} \mu_{j,k} (r_0^*)_k \tag{60}$$

forall $j = 1$ to n

$$(\sigma_0)_j = (r_0)_j \tag{61}$$

forall $j = 1$ to n (reduction $+p_0$)

$$p_0 = (\sigma_0)_j * (r_0)_j \tag{62}$$

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:

forall $j = 1$ to n

$$(q_i)_j = A(\sigma_i)_j \tag{63}$$

forall $j = 1$ to n

$$(g_i)_j = \sum_{k=\max(1,j-\delta l+1)}^{\min(n,j+\delta l-1)} \mu_{j,k} (q_i)_k \tag{64}$$

forall $j = 1$ to n (reduction$+t_i$)

$$t_i = (\sigma_0)_j * (g_i)_j \tag{65}$$

$$\alpha_i = p_i / t_i \tag{66}$$

forall $j = 1$ to n

$$(e_{i+1})_j = (r_i)_j + \beta_i (e_i)_j - \alpha_i (g_i)_j \tag{67}$$

$$(f_i)_j = (r_i)_j + \beta_i (e_i)_j + (e_{i+1})_j \tag{68}$$

$$(u_{i+1})_j = (u_i)_j + \alpha_i (f_i)_j \tag{69}$$

forall $j = 1$ to n

$$(q_i)_j = A(f_i)_j \tag{70}$$

forall $j = 1$ to n

$$(g_i)_j = \sum_{k=\max(1,j-\delta l+1)}^{\min(n,j+\delta l-1)} \mu_{j,k} (q_i)_k \tag{71}$$

forall $j = 1$ to n

$$(r_{i+1})_j = (r_i)_j - \alpha_i (g_i)_j \tag{72}$$

forall $j = 1$ to n (reduction$+p_{i+1}$)

$$p_{i+1} = (\sigma_0)_j * (r_{i+1})_j \tag{73}$$

$$\beta_{i+1} = p_{i+1} / p_i \tag{74}$$

forall $j = 1$ to n

$$(\sigma_{i+1})_j = (r_{i+1})_j + 2\beta_{i+1} (e_{i+1})_j + \beta_{i+1}^2 (\sigma_i)_j \tag{75}$$

The computational complexity of the PEPGCGS method is $\approx O[(4\delta l + 4\ell_1 + 4\ell_2 + 15)local\_n$ multiplications $+$ $8local\_n$ adds$]\nu$ operations, where ν denotes the

number of iterations required for convergence to a predetermined tolerance level where $local\_n = n/no\_procs$ denotes the number of rows assigned to each processor. The algorithm for the PEPGBICG-STAB method can be implemented in a similar manner.

It should be noted that the parallelization of the coefficient matrix $A\times$vector operation has been implemented by taking advantage of the sparsity of the coefficient matrix A.

In our implementation, the parallel for pragma in OpenMP with static scheduling has been used in order to generate code that forks/joins threads.

The theoretical speedup and efficiency of the PEPGCGS algorithm are:

$$S_p^{\delta l} = \frac{1}{\frac{1}{no\_procs} + \frac{11t_1}{O(4\delta l+4\ell_1+4\ell_2+15)nt_m}} \tag{76}$$

and

$$E_p^{\delta l} = \frac{1}{1 + \frac{11t_1 no\_procs}{O(4\delta l+4\ell_1+4\ell_2+15)nt_m}} \tag{77}$$

where t_1 is the latency of one fork / join operation and t_m denotes the computational time of one multiplication. It is obvious that for $\delta l \to \infty$ then $S_p^{\delta l} \to no\_procs$ and $E_p^{\delta l} \to 1$.

4 Numerical results

In this final section the applicability and effectiveness of the derived parallel finite element approximate inverse preconditioning is shown.

Let us consider the following biharmonic problem in three dimensions:

$$\nabla^4 u(x,y,z) = \frac{\partial^4 u}{\partial x^4} + \frac{\partial^4 u}{\partial y^4} + \frac{\partial^4 u}{\partial z^4} + 2\frac{\partial^4 u}{\partial x^2 \partial y^2} + 2\frac{\partial^4 u}{\partial x^2 \partial z^2} + 2\frac{\partial^4 u}{\partial y^2 \partial z^2}$$

$$= 1, (x,y,z) \in R, \tag{78}$$

where R is a three dimensional domain, subject to the boundary conditions

$$u(x,y,z) = \partial u(x,y,z)/\partial \eta = 0, (x,y,z) \in \partial R \tag{79}$$

The domain is covered by a non-overlapping triangular network resulting in a hexagonal mesh. 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 numerical results presented in this section were obtained on an SMP machine consisting of 8 (dual 4 core machine) 2.0 GHz, 64-bit xeons, with 8 GB RAM running Debian GNU/Linux (University College at Cork). For the parallel implementation of the algorithms presented, the Intel Fortran Compiler with OpenMP directives has been utilized with no optimization enabled at the compilation level. It should be noted that due to administrative policies, we were not able to explore the full processor resources (i.e. more than 8 threads). In our implementation, the parallel for pragma has been used. Additionally, static scheduling has been used (*schedule(static)*), whereas the use of dynamic scheduling has not produced improved results.

The convergence behavior of the EPGCGS and EPGBI-CGSTAB method, using the "coupled equation approach" based on FEALUFA-3D and OPTGAIFEM-3D algorithms, for several values of the order n and the "retention" parameter δl is given in Tab. 1 and Tab. 2 respectively.

Taking into consideration the resources available and the memory requirements of the proposed method, the maximum order of the linear system was chosen to be $n = 4826809$, with semi-bandwidths $m = 170$ and $p = 28562$ with the retention parameter set to $\delta l = 1$ or $\delta l = 2$.

The convergence behavior of the EPGCGS and EPGBI-CGSTAB method for large order sparse linear systems is given in Tab. 3.

Table 1: The convergence behavior of the EPGCGS method for several values of n, m, p and δl.

			Retention parameter δl							
n	m	p	1	2	m	$2m$	$3m$	$4m$	p	$2p$
729	10	82	14	14	12	10	10	10	8	8
2744	15	197	20	18	16	12	12	12	10	10
6859	20	362	18	18	16	12	12	12	10	10
24389	30	842	18	18	16	12	12	12	10	10

The speedups and efficiencies of the PAND-OGAIFEM-3D algorithm for several values of the "retention" parameter δl with $n = 24389$, $m = 30$, $p = 842$ are given in Tab. 4 and Tab. 5 respectively. In Fig. 1 the speedups and processors allocated for several values of the "retention" parameter δl is presented for the PAND-OGAIFEM-3D algorithm along with theoretical estimates, with $n = 24389$, $m = 30$, $p = 842$.

Table 2: The convergence behavior of the EPGBI-CGSTAB method for several values of n, m, p and δl.

			Retention parameter δl							
n	m	p	1	2	m	$2m$	$3m$	$4m$	p	$2p$
729	10	82	9	9	8	7	7	7	6	5
2744	15	197	10	10	9	7	7	7	6	6
6859	20	362	11	11	9	8	8	8	7	6
24389	30	842	11	11	9	8	8	8	7	6

Table 3: The convergence behavior of the EPGCGS and EPGBICG-STAB method for large linear systems.

			EPGCGS		EPGBI-CGSTAB	
n	m	p	$\delta l = 1$	$\delta l = 2$	$\delta l = 1$	$\delta l = 2$
205379	60	3482	20	20	11	11
493039	80	6242	20	20	11	11
970299	100	9802	20	20	11	11
3307949	150	22202	24	22	11	11
4826809	170	28562	26	22	12	11

Table 4: Speedups and processors allocated of the PAND-OGAIFEM-3D algorithm, for several values of δl, with $n = 24389$, $m = 30$ and $p = 842$.

Speedups for the PAND-OGAIFEM-3D algorithm			
Retention	Number of processors		
parameter	no_procs =2	no_procs =4	no_procs =8
$\delta l = m$	1.600	2.667	3.200
$\delta l = 2m$	1.692	2.750	3.667
$\delta l = 3m$	1.700	3.091	4.250
$\delta l = 4m$	1.846	3.200	4.800
$\delta l = p$	1.879	3.245	5.950
$\delta l = 2p$	1.955	3.406	6.949

The class IV approximate inverse M_i , Eq. 13, retains only its diagonal elements which are computed based on the inversion of the diagonal elements of the lower triangular matrix L_{r_1,r_2}. It should be mentioned that the parallel speedup of this class of approximate inverse was obtained to be equal to the number of processors, which is the theoretical estimate for speedup, indicating that this class is a fast

Table 5: Efficiencies and processors allocated of the PAND-OGAIFEM-3D algorithm, for several values of δl, with $n = 24389$, $m = 30$ and $p = 842$.

Efficiencies for the PAND-OGAIFEM-3D algorithm			
Retention parameter	Number of processors		
	$no\_procs = 2$	$no\_procs = 4$	$no\_procs = 8$
$\delta l = m$	0.800	0.667	0.400
$\delta l = 2m$	0.846	0.688	0.458
$\delta l = 3m$	0.850	0.773	0.531
$\delta l = 4m$	0.923	0.800	0.600
$\delta l = p$	0.939	0.811	0.744
$\delta l = 2p$	0.977	0.851	0.869

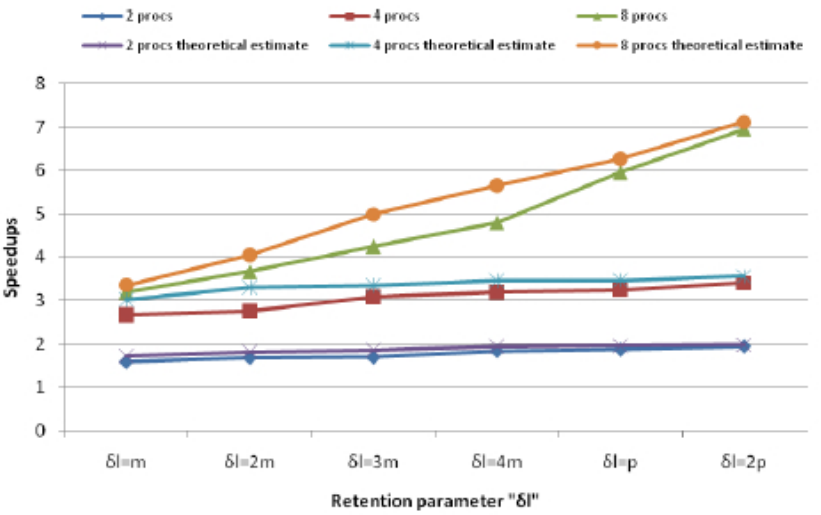


Figure 1: Speedups and processors allocated of the PAND-OGAIFEM-3D algorithm, for several values of δl, with $n = 24389$, $m = 30$ and $p = 842$.

inverse matrix algorithm.

Additionally, the speedups and efficiencies of the PEPGCGS method for several values of the "retention" parameter δl with $n = 24389$, $m = 30$, $p = 842$ are given in Tab. 6 and Tab. 7 respectively. In Fig. 2 the speedups and processors allocated for several values of the "retention" parameter δl is presented along with theoretical estimates for the PEPGCGS method, with $n = 24389$, $m = 30$, $p = 842$.

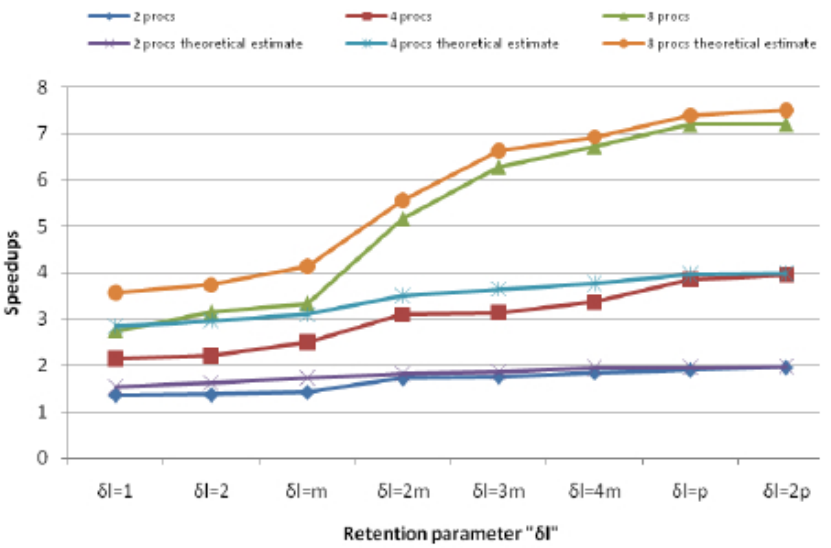


Figure 2: Speedups and processors allocated of the PEPGCGS method along with theoretical estimates, for several values of δl, with $n = 24389$, $m = 30$ and $p = 842$.

Table 6: Speedups and processors allocated of the PEPGCGS method, for several values of δl, with $n = 24389$, $m = 30$ and $p = 842$.

Speedups for the PEPGCGS algorithm			
Retention parameter	Number of processors		
	no_procs =2	*no_procs* =4	*no_procs* =8
δl=1	1.365	2.146	2.748
δl=2	1.372	2.215	3.143
δl=m	1.429	2.500	3.333
δl=2m	1.722	3.100	5.167
δl=3m	1.760	3.143	6.286
δl=4m	1.850	3.364	6.727
δl=p	1.910	3.859	7.195
δl=2p	1.963	3.951	7.209

It can be observed, that due to coarse granularity and the reduced overheads of the parallel construction of the approximate inverse, the parallel efficiency is almost close to the upper theoretical bound for all values of the "retention" parameter δl that are multiples of the semi-bandwidths m and p.

Table 7: Efficiencies and processors allocated of the PEPGCGS method, for several values of δl, with $n = 24389$, $m = 30$ and $p = 842$.

Efficiencies for the PEPGCGS algorithm			
Retention	Number of processors		
parameter	*no_procs* =2	*no_procs* =4	*no_procs* =8
δl=1	0.683	0.537	0.343
δl=2	0.686	0.554	0.393
δl=m	0.714	0.625	0.417
δl=2m	0.861	0.775	0.646
δl=3m	0.880	0.786	0.786
δl=4m	0.925	0.841	0.841
δl=p	0.955	0.965	0.899
δl=2p	0.981	0.988	0.901

Additionally for large values of the "retention" parameter, i.e. multiples of the semibandwidths m and p, the speedups and the efficiency tend to the upper theoretical bound, for the parallel preconditioned conjugate gradient type method, since the coarse granularity amortizes the parallelization overheads.

5 Conclusions

The design of parallel explicit approximate inverses and preconditioned conjugate gradienttype schemes results in efficient parallel methods for solving sparse linear systems on symmetric multiprocessor systems. The main advantage of the proposed method is that the approximate inverse is computed explicitly and can be efficiently used in conjunction with parallel preconditioned conjugate gradienttype methods for solving biharmonic problems in three space variables, using an "inner-outer" iteration scheme.

Finally, we state that the new parallel finite element approximate inverse preconditioning, can be efficiently used for solving non-linear biharmonic problems on shared memory computer systems. Further parallel algorithmic techniques will be investigated in order to improve the parallel performance of the explicit approximate inverse preconditioning methods on shared memory computer systems, particularly by increasing the computational work output per processor and eliminating process synchronization and any associated latencies.

Acknowledgement: The authors would like to thank indeed Dr. John Morrison of the Department of Computer Science, University College of Cork for the provi-

sion of computational facilities and support through the WebCom-G project funded by Science Foundation Ireland.

References

Axelsson, O. (1973): Notes on the numerical solution of the biharmonic equation. *IMA Journal of Applied Mathematics*, vol. 11, no. 2, pp. 213–226.

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

Buzbee, B. L.; Dorr, F. W. (1974): The direct solution of the biharmonic equation on rectangular regions and the poisson equation on irregular regions. *SIAM Journal on Numerical Analysis*, vol. 11, no. 4, pp. 753–763.

Ehrlich, L. W. (1971): Solving the biharmonic equation as coupled finite difference equations. *SIAM Journal on Numerical Analysis*, vol. 8, no. 2, pp. 278–287.

Ehrlich, L. W. (1973): Solving the biharmonic equation in a square: a direct versus a semidirect method. *Commun. ACM*, vol. 16, pp. 711–714.

Elman, H. C. (1986): A stability analysis of incomplete lu factorizations. *Math. Comput.*, vol. 47, pp. 191–217.

Evans, D. J. (1983): *Preconditioning Methods: Theory and Applications*. Gordon and Breach Science Publishers, Inc., Newark, NJ, USA, 1st edition.

Giannoutakis, K. M.; Gravvanis, G. A. (2009): Design and implementation of parallel approximate inverse classes using openmp. *Concurrency and Computation: Practice and Experience*, vol. 21, no. 2, pp. 115–131.

Gravvanis, G. A. (1996): A note on the rate of convergence of explicit approximate inverse preconditioning. *Journal of Computational Mathematics*, vol. 60, pp. 77–89.

Gravvanis, G. A. (1997): Solving 3d time-dependent problems by pseudoinverses. In Lewis, R.; Cross, J.(Eds): *Numerical Methods in Thermal Problems*, volume X, pp. 910–921. Pineridge Press.

Gravvanis, G. A. (1999): Generalized approximate inverse finite element matrix techniques. *Neural, Parallel Sci. Comput.*, vol. 7, no. 4, pp. 487–500.

Gravvanis, G. A. (1999): Preconditioned iterative methods for solving 3d boundary value problems. *International Journal of Computer Mathematics*, vol. 71, pp. 117–136.

Gravvanis, G. A. (2000): Explicit preconditioning conjugate gradient schemes for solving biharmonic problems. *Engineering Computations*, vol. 17, pp. 154–165.

Gravvanis, G. A. (2002): Explicit approximate inverse preconditioning techniques. *Archives of Computational Methods in Engineering*, vol. 9, pp. 371–402.

Gravvanis, G. A. (2009): High performance inverse preconditioning. *Archives of Computational Methods in Engineering*, vol. 16, pp. 77–108.

Gravvanis, G. A.; Giannoutakis, K. M. (2005): Parallel normalized implicit preconditioned conjugate gradient methods for solving biharmonic equations. In Bathe, K.(Ed): *Computational Fluid and Solid Mechanics 2005 (Proceedings of the Third MIT Conference on Computational Fluid and Solid Mechanics)*, pp. 1120–1125. Elsevier.

Gravvanis, G. A.; Giannoutakis, K. M. (2008): Fast parallel finite element approximate inverses. *CMES: Computer Modeling in Engineering & Sciences*, vol. 32, no. 1, pp. 35–44.

Gravvanis, G. A.; Lipitakis, E. A. (1996): An explicit sparse unsymmetric finite element solver. *Communications in Numerical Methods in Engineering*, vol. 12, no. 1, pp. 21–29.

Gravvanis, G. A.; Lipitakis, E. A. (1996): A three-dimensional explicit preconditioned solver. *Computers & Mathematics with Applications*, vol. 32, no. 2, pp. 111 – 131.

Greenspan, D.; Schultz, D. (1972): Fast finite-difference solution of biharmonic problems. *Commun. ACM*, vol. 15, pp. 347–350.

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

Huckle, T. (1998): Efficient computation of sparse approximate inverses. *Numerical Linear Algebra with Applications*, vol. 5, no. 1, pp. 57–71.

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

Kolotilina, L. Y.; Yeremin, A. Y. (1993): Factorized sparse approximate inverse preconditioning. *SIAM Journal on Matrix Analysis and Applications*, vol. 14, pp. 45–58.

Lipitakis, E.; Gravvanis, G. (1995): Explicit preconditioned iterative methods for solving large unsymmetric finite element systems. *Computing*, vol. 54, pp. 167–183.

Nodera, T.; Takahashi, H. (1981): Preconditioned conjugate gradient algorithm for solving biharmonic equations. In Vichnevetsky, R.; Stepleman, R.(Eds): *Advances in Computer Methods for Partial Differential Equations, IMACS*, volume IV.

Notay, Y. (1993): On the convergence rate of the conjugate gradients in presence of rounding errors. *Numerische Mathematik*, vol. 65, pp. 301–317.

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

Saad, Y.; van der Vorst, H. A. (2000): Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, vol. 123, pp. 1–33.

Smith, J. (1968): The coupled equation approach to the numerical solution of the biharmonic equation by finite Differences. *SIAM Journal on Numerical Analysis*, vol. 5, pp. 323–339.

van der Sluis, A.; van der Vorst, H. A. (1986): The rate of convergence of conjugate gradients. *Numerische Mathematik*, vol. 48, pp. 543–560.

Yousif, W. S.; Evans, D. J. (1993): Explicit block iterative method for the solution of the biharmonic equation. *Numerical Methods for Partial Differential Equations*, vol. 9, no. 1, pp. 1–12.