Fast Parallel Finite Element Approximate Inverses

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

Abstract: A new parallel normalized optimized approximate inverse algorithm, based on the concept of the "fish bone" computational approach with cyclic distribution of the processors satisfying an antidiagonal data dependency, for computing classes of explicit approximate inverses, is introduced for symmetric multiprocessor systems. The parallel normalized explicit approximate inverses are used in conjunction with parallel normalized explicit preconditioned conjugate gradient square schemes, for the efficient solution of finite element sparse linear systems. The parallel design and implementation issues of the new proposed algorithms are discussed and the parallel performance is presented, using OpenMP.

Keyword: Sparse linear systems, preconditioning, parallel normalized approximate inverses, parallel preconditioned conjugate gradient method, parallel computations, symmetric multiprocessor systems, OpenMP.

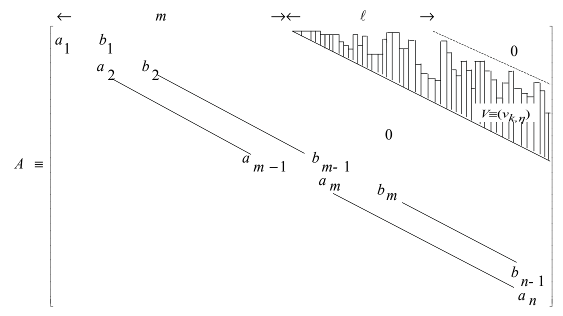
1 Introduction

Let us consider the linear system derived from the discretization of many scientific and engineering two dimensional problems by the finite element (FE) method, i.e.

$$Au = s \qquad (1)$$

where the coefficient matrix A is a non-singular large, sparse, symmetric, positive definite, diagonally dominant $(n \times n)$ matrix of irregular structure (where all the off-center band terms are grouped in regular band of width ℓ at semi-bandwidth m), Eq. (2), while u is the 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.

The solution of sparse non-linear systems, because of its applicability to real-life problems, has attracted the attention of many researchers and has been obtained either by direct or iterative methods, [Duff (2000); Lipitakis and Evans (1987)]. Recently, a direct solution method for the quasi-unsymmetric sparse matrix arising in the Meshless Local Petrov-Galerkin method has been proposed in [Yuan, Chen, and Liu (2007)], while parallel Seidel-type domain decomposition iterative method, based on a hybridization of a nonconforming mixed finite element method, was introduced in [Ha, Seo and Sheen (2006)].



(2)

An important achievement over the last decades is the appearance and use of Explicit Preconditioned Methods for solving sparse linear systems, and the preconditioned form of the linear system (1) is

$$MAu = Ms \qquad (3)$$

where M is a suitable preconditioner, [Lipitakis and Evans (1987); Lipitakis and Gravvanis (1995)]. The preconditioner M has therefore to satisfy the following conditions: (i) MA should

[1] Department of Electrical and Computer Engineering, School of Engineering, Democritus University of Thrace, 12, Vas. Sofias street, GR 671 00 Xanthi, Greece; Email: {ggravvan; kgiannou}@ee.duth.gr

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, [Gravvanis (2000); Huckle (1999); Huckle (1998); Saad and Vorst (2000)]. In recent years many researchers have derived preconditioners based on various techniques, which are difficult to be implemented on parallel systems, [Akcadogan and Dag (2003); Benzi (2002); Benzi, Meyer and Tuma (1996); Chan and Vorst (1997); Cosgrove, Dias and Griewank (1992); Dubois, Greenbaum and Rodrigue (1979); Grote and Huckle (1977); Kolotilina and Yeremin (1993); Saad and Vorst (2000)]. Our main motive for the derivation of the new Parallel Normalized Approximate Inverse Finite Element Matrix techniques is that they can be efficiently used in conjunction with normalized explicit preconditioned conjugate gradient – type schemes on symmetric multiprocessor systems.

For the parallel construction of the normalized approximate inverse preconditioner the "fish-bone" computational approach, with respect to the antidiagonal data dependency pattern, has been considered. Cyclic distribution of the processors for any associated row and column (inverted L-shaped block) has been used, in order to increase the granularity and to overcome the parallelization overheads, yielding an efficient approach for any choice of the "retention" parameter (number of elements kept in the approximate inverse) and number of processors.

The inherently parallel linear operations between vectors and matrices involved in the normalized explicit preconditioned conjugate gradient schemes exhibit significant amounts of loop-level parallelism that can lead to high performance gain on shared address space systems, [Akl (1997); Dongarra, Duff, Sorensen and Vorst (1998); Grama, Gupta, Karypis and Kumar (2003)].

For the implementation of the parallel programs, the OpenMP application programming interface has been used. OpenMP has emerged as a shared-memory programming standard and it consists of compiler directives and functions for supporting both data and functional parallelism. The *parallel for* pragma with static scheduling has been used for the parallelization of loops on both the construction of the approximate inverse and the conjugate gradient schemes.

In Section 2, a new parallel finite element normalized approximate inverse algorithm is introduced, based on the "fish bone" computational pattern satisfying an antidiagonal data dependency. In Section 3, a parallel normalized preconditioned conjugate gradient square method for solving sparse finite element systems is presented. Finally in Section 4, the performance and applicability of the new proposed parallel approximate inverse preconditioning is discussed and the parallel performance on a symmetric multiprocessor system is given, using OpenMP.

2 Parallel Normalized Finite Element Approximate Inverses

In this section we present a parallel implementation of the finite element normalized approximate inverse algorithm, based on the "fish bone" computational approach, for solving linear systems on symmetric multiprocessor systems.

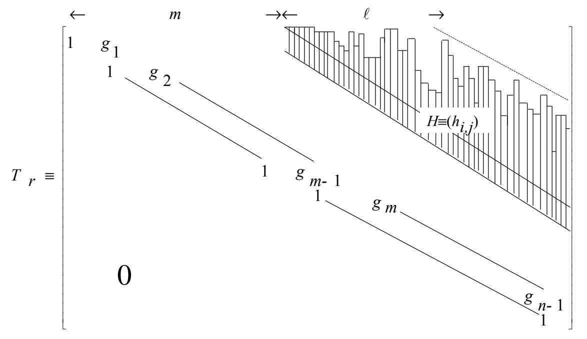
Let us assume the normalized sparse approximate factorization, [Gravvanis and Giannoutakis (2006); Gravvanis and Giannoutakis (2003); Lipitakis and Evans (1987); Lipitakis and Evans (1984)], of the coefficient matrix A, such that:

$$A \approx D_r T_r^t T_r D_r, \quad r \in [1, \dots, m-1), \tag{4}$$

where r is the "fill-in" parameter, i.e. the number of outermost off-diagonal entries retained at semi-bandwidths m, D_r is a diagonal matrix

$$D_r \equiv diag\left(d_1, \dots, d_{m-1} \vdots d_m, \dots, d_{p-1} \vdots d_p, \dots, d_n\right), \tag{5}$$

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



(6)

The elements of the decomposition factors were computed by the **FEANOF-2D** algorithm, [Lipitakis and Evans (1987); Lipitakis and Evans (1984)].

Let $M_r^{\delta l} = D_r^{-1}\hat{M}_r^{\delta l}D_r^{-1} = (\mu_{i,j})$, $i \in [1,n]$, $j \in [\max(1, i-\delta l+1), \min(n, i+\delta l-1)]$, be the banded form of the normalized approximate inverse of the coefficient matrix A. The elements of the banded form of the approximate inverse can be determined by retaining a certain number of elements of the inverse, i.e. only δl elements in the lower part and δl-1 elements in the upper part of the inverse next to the main diagonal. Then the elements can be computed by solving recursively the following systems:

$$\hat{M}_r^{\delta l}T_r^t = (T_r)^{-1} \text{ and } T_r\hat{M}_r^{\delta l} = \left(T_r^t\right)^{-1}, \tag{7}$$

without inverting the decomposition factors, by the **Nor**malized **B**anded **A**pproximate **I**nverse **F**inite **E**lement **M**atrix algorithmic procedure (henceforth called the **NORBAIFEM-2D** algorithm).

The computational work of the **NORBAIFEM-2D** algorithm is $\mathrm{O}[n\delta l(r+\ell)]$ multiplicative operations, while the storage of the approximate inverse in only $n \times (2\delta l - 1)$-vector spaces is achieved using an optimized storage scheme, based on a moving window shifted from bottom to top, [Gravvanis and Giannoutakis (2003)]. The optimized **NORBAIFEM-2D** algorithm (henceforth called the **NOROBAIFEM-2D** algorithm) is particularly effective for solving "narrow-banded" sparse systems of very large order, i.e. $\delta \ll ln/2$.

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, [Gravvanis and Giannoutakis (2003)]:

$$A^{-1} \leftarrow \overbrace{D_r^{-1}\hat{\hat{M}}{}_{r=m-1}^{\delta l}D_r^{-1}}^{\text{class I}} \leftarrow \overbrace{D_r^{-1}\hat{M}_{r=m-1}^{\delta l}D_r^{-1}}^{\text{class II}},$$
$$\leftarrow \overbrace{D_r^{-1}\hat{M}_r^{\delta l}D_r^{-1}}^{\text{class III}} \leftarrow \overbrace{D_r^{-2}}^{\text{class IV}} \tag{8}$$

where the entries of the class I inverse have been retained after the computation of the exact inverse ($r = m-1$), while the entries of the class II inverse have been computed and retained during the computational procedure of the (approximate) inverse ($r = m-1$). The entries of the class III inverse have been retained after the computation of the approximate inverse ($r \leq m$-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. δl=1, hence $\hat{M}_r^{\delta l} \equiv I$, resulting in a fast inverse algorithm.

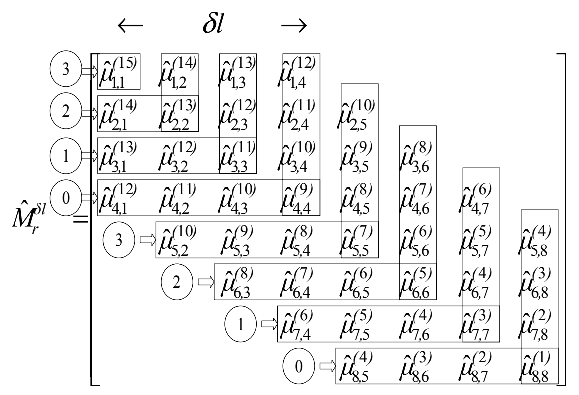
For the parallelization of the **NORBAIFEM-2D** algorithm, the "fish bone" computational pattern has been used. Each inverted L-shaped block, cf. (9), is assigned onto one processor, i.e. at most ($2\delta l$-1) elements per processor. If the number of processors is less than the order of the matrix n, then a cyclic distribution of the processors on the inverted L-shaped blocks (i.e. i-th row and i-th column) has been used. For example, without loss of generality, the pattern is shown in equation (9), for n=8, δl=4 and number of processors=4 (the numbers in the circles indicate the rank of each processor).

The computation of each inverted L-shaped block can not start concurrently by the processors, because of the data dependency of the elements of the approximate inverse. More specific, the processor with rank (k), can not start the computation of the element $\hat{\mu}_{i,j}$, until the processor with rank (k-1) has finished the computation of the element

$\hat{\mu}_{i+1,j}$ and its symmetric counterpart (if they exist within the band of the approximate inverse). The superscript of each element in equation (9) indicate the priority of the computations, i.e. the element $\hat{\mu}_{7,6}$ of the processor with rank 1, will start to be computed after processor with rank 0 has computed the element $\hat{\mu}_{8,6}$ and $\hat{\mu}_{6,8}$. When processor 1 has finished the computation of the element $\hat{\mu}_{7,6}$ and its symmetric counterpart, then the processor with rank 2 is ready to start the computation of the element $\hat{\mu}_{6,6}$. The data dependency pattern follows the antidiagonal motion (wave pattern approach) described in [Gravvanis and Giannoutakis (2006)].

It should be noted that when $\delta l = 1$ we have $\hat{M}_r^{\delta l} \equiv I$ and the approximate inverse for $\delta l = 2$ can not be constructed in parallel, due to the data dependency pattern and the symmetric property of the approximate inverse.

For the parallel construction of the optimized form of the approximate inverse, a simple transformation of the indexes (based on a moving window shifted from bottom to top) of the elements of the approximate inverse has been used, cf. [Gravvanis (1998)].



(9)

Let us consider that the command **forall** denotes the parallel for instruction (forks/joins threads), for executing parallel loops, and *myrank* is the rank of each process. Then, the **Pa**rallel **Fi**sh-**Bo**ne **NOROBAIFEM-2D** algorithm (henceforth called the **PaFiBo-NOROBAIFEM-2D** algorithm), on symmetric multiprocessor systems, can be described as follows:

```
forall $i = n$ downto 1 (cyclic distribution with chunk size = 1)
  call inverse($i$)
```

where the function inverse(i), computes the i-th inverted L-shaped block according to the **NOROBAIFEM-3D** algorithm, [Gravvanis and Giannoutakis (2003)]:

```
function inverse ($i$)
Let $r\ell = r+\ell$, $m\ell = m+\ell$, $mr = m-r$, $nmr = n-mr$ and $nm\ell = n-m\ell$.
for $j = i$ downto $\max(i-\delta l+1,1)$
  if $i <> n$ then
  wait (until $(myrank-1)(j) > myrank(j)$)
  if $j > nmr$ then
    if $i = j$ then
      if $i$=$n$ then

$\hat{\mu}_{1,1} = 1$   (10)

      else

$\hat{\mu}_{n-i+1,1} = 1 - g_j \cdot \hat{\mu}_{n-j,\delta l+1}$   (11)

    else

$\hat{\mu}_{n-i+1,i-j+1} = -g_j \cdot \hat{\mu}_{n-i+1,i-j}$   (12)

  else
if $j \geq r\ell$ and $j \leq nmr$ then
  if $i = j$ then

$\hat{\mu}_{n-i+1,1} = 1 - g_j \cdot \hat{\mu}_{n-j,\delta l+1} - \sum_{k=0}^{nmr-j} h_{r\ell-1-k,j+1-r+k} \cdot \hat{\mu}_{x,y}$   (13)

  call mw ($n$,$\delta l$, $i$, $mr$+$j$+$k$,$x$,$y$)
  else

$\hat{\mu}_{n-i+1,i-j+1} = -g_j \cdot \hat{\mu}_{n-i+1,i-j} - \sum_{k=0}^{nmr-j} h_{r\ell-1-k,j+1-r+k} \cdot \hat{\mu}_{x,y}$   (14)

  call mw ($n$,$\delta l$, $i$, $mr$+$j$+$k$,$x$,$y$)
  else
  if $j > nm\ell+1$ and $j \leq r\ell-1$ then
    if $i = j$ then
      $\hat{\mu}_{n-i+1,1} = 1 - g_j \cdot \hat{\mu}_{n-j,\delta l+1} - \sum_{\substack{k=j+1-r\\k>0}}^{\ell} h_{j,k} \cdot \hat{\mu}_{x1,y1} - \sum_{\lambda=0}^{nm\ell} h_{j-1-\lambda,\ell+1+k} \cdot \hat{\mu}_{x2,y2}$   (15)
      call mw $(n,\delta l,i,m+k\text{-}1,x1,y1)$ call mw $(n,\delta l,i,m\ell+\lambda,x2,y2)$
    else
      $\hat{\mu}_{n-i+1,i-j+1} = -g_j \cdot \hat{\mu}_{n-i+1,i-j} - \sum_{\substack{k=j+1-r\\k>0}}^{\ell} h_{j,k} \cdot \hat{\mu}_{x1,y1} - \sum_{\lambda=0}^{nm\ell} h_{j-1-\lambda,\ell+1+k} \cdot \hat{\mu}_{x2,y2}$   (16)
  call mw $(n,\delta l,i,m+k\text{-}1,x1,y1)$ call mw $(n,\delta l,i,m\ell+\lambda,x2,y2)$
  else
  if $j \leq nm\ell+1$ then
    if $i = j$ then
      if $i=1$ then
        $\hat{\mu}_{n,1} = 1 - g_1 \cdot \hat{\mu}_{n-1,\delta l+1} - \sum_{k=1}^{\ell} h_{1,k} \cdot \hat{\mu}_{x,y}$   (17)
        call mw $(n,\delta l,1,m+k\text{-}1,x,y)$
      else
        $\hat{\mu}_{n-i+1,1} = 1 - g_j \cdot \hat{\mu}_{n-j,\delta l+1} - \sum_{\substack{k=j+1-r\\k>0}}^{\ell} h_{j,k} \cdot \hat{\mu}_{x1,y1} - \sum_{\lambda=1}^{j-1} h_{j-\lambda,\ell+\lambda} \cdot \hat{\mu}_{x2,y2}$   (18)
    call mw $(n,\delta l,i,m+k-\ell,x1,y1)$ call mw $(n,\delta l,i,m\ell+\lambda\text{-}1,x2,y2)$
    else
      $\hat{\mu}_{n-i+1,1} = -g_j \cdot \hat{\mu}_{n-j,\delta l+1} - \sum_{\substack{k=j+1-r\\k>0}}^{\ell} h_{j,k} \cdot \hat{\mu}_{x1,y1} - \sum_{\lambda=1}^{j-1} h_{j-\lambda,\ell+\lambda} \cdot \hat{\mu}_{x2,y2}$   (19)
      call mw $(n,\delta l,i,m+k-\ell,x1,y1)$ call mw $(n,\delta l,i,m\ell+\lambda\text{-}1,x2,y2)$
    else
  if $i <> j$ then
    $\hat{\mu}_{n-i+1,\delta l+i-j} = \hat{\mu}_{n-i+1,i-j+1}$   (20)
```

while, the procedure mw$(n,\delta l,s,q,x,y)$, [Gravvanis (1998)], can be described as follows:

```
procedure mw$(n,\delta l,s,q,x,y)$
if $s \geq q$ then
  $x = n+1-s; \quad y = s-q+1$   (21)
else
  $x = n+1-q; \quad y = \delta l+q-s.$   (22)
```

In order to follow the data dependency pattern during the parallel construction of the approximate inverse, a shared variable has been used for synchronizing the processes according to their private variable j (the private variable j of processor k should be less than the private variable j of processor k-1 for the computation of the element $\hat{\mu}_{i,j}$ and its symmetric counterpart of processor k). The function **wait** is a custom synchronization utility, which compares the private variable j of process with rank *myrank* ((*myrank*)(j)), and private variable j of process with rank *myrank*-1 ((*myrank*-1)(j)). The cyclic distribution of the inverted L-shaped blocks on the processors is implemented with OpenMP by using static scheduling with chunk size = 1 in the *parallel for* pragma.

3 Parallel Normalized Preconditioned Conjugate Gradient – type method

In this section we present a class of parallel Normalized Explicit Preconditioned Conjugate Gradient - type method, based on the derived parallel optimized approximate inverse, designed for symmetric multiprocessor systems.

The **P**arallel **N**ormalized **E**xplicit **P**reconditioned **C**onjugate **G**radient **S**quare (**PNEPCGS**) algorithm, for solving linear systems, can then be described as follows:

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

```
forall $j$=1 to $n$
    $$(r_0^*)_j = s_j - A(u_0)_j \quad (23)$$
if $\delta l$=1 then
    forall $j$=1 to $n$
        $$(r_0)_j = (r_0^*)_j / (d^2)_j \quad (24)$$
else
    forall $j$=1 to $n$
        $$(r_0)_j = \left( \sum_{k=\max(1,j-\delta l+1)}^{j} \hat{\mu}_{n+1-i,i+1-k} (r_0^*)_k / d_k + \sum_{k=j+1}^{\min(n,j+\delta l-1)} \hat{\mu}_{n+1-k,\delta l+k-j} (r_0^*)_k / d_k \right) \Big/ (d)_j \quad (25)$$
forall $j$=1 to $n$
    $$(\sigma_0)_j = (r_0)_j \quad (26)$$
forall $j$=1 to $n$ (reduction $+p_0$)
    $$p_0 = (\sigma_0)_j * (r_0)_j \quad (27)$$
```

Then, for i = 0, 1, ..., (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 $n$
    $$(q_i)_j = A(\sigma_i)_j \quad (28)$$
if $\delta l$=1 then
    forall $j$=1 to $n$
        $$(g_i)_j = (q_i)_j / (d^2)_j \quad (29)$$
else
    forall $j$=1 to $n$
        $$(g_i)_j = \left( \sum_{k=\max(1,j-\delta l+1)}^{j} \hat{\mu}_{n+1-i,i+1-k} (q_i)_k / d_k + \sum_{k=j+1}^{\min(n,j+\delta l-1)} \hat{\mu}_{n+1-k,\delta l+k-j} (q_i)_k / d_k \right) \Big/ (d)_j \quad (30)$$
forall $j$=1 to $n$ (reduction $+t_i$)
    $$t_i = (\sigma_0)_j * (g_i)_j \quad (31)$$
$$a_i = p_i / t_i \quad (32)$$
forall $j$=1 to $n$
    $$(e_{i+1})_j = (r_i)_j + b_i (e_i)_j - a_i (g_i)_j \quad (33)$$
    $$(f_i)_j = (r_i)_j + b_i (e_i)_j + (e_{i+1})_j \quad (34)$$
    $$(u_{i+1})_j = (u_i)_j + a_i (f_i)_j \quad (35)$$
forall $j$=1 to $n$
    $$(q_i)_j = A(f_i)_j \quad (36)$$
if $\delta l$=1 then
```

```
        forall $j$=1 to $n$
            $(g_i)_j = (q_i)_j / (d^2)_j$    (37)
    else
        forall $j$=1 to $n$
            $(g_i)_j = \left( \sum_{k=\max(1,j-\delta l+1)}^{j} \hat{\mu}_{n+1-i,i+1-k} (q_i)_k / d_k + \sum_{k=j+1}^{\min(n,j+\delta l-1)} \hat{\mu}_{n+1-k,\delta l+k-j} (q_i)_k / d_k \right) \Big/ (d)_j$    (38)
    forall $j$=1 to $n$
        $(r_{i+1})_j = (r_i)_j - a_i (g_i)_j$    (39)
    forall $j$=1 to $n$ (reduction +$p_{i+1}$)
        $p_{i+1} = (\sigma_0)_j * (r_{i+1})_j$    (40)
    $b_{i+1} = p_{i+1} / p_i$    (41)
    forall $j$=1 to $n$
        $(\sigma_{i+1})_j = (r_{i+1})_j + 2b_{i+1} (e_{i+1})_j + b_{i+1}^2 (\sigma_i)_j$    (42)
```

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 with static scheduling has been used in order to generate code that forks/joins threads.

4 Numerical Results

In this section we examine the applicability and effectiveness of the new proposed parallel finite element approximate inverse preconditioning for solving sparse linear systems.

The numerical results presented in this section were obtained on an SMP machine consisting of 16 2.2 GHz Dual Core AMD Opteron processors, with 32 GB RAM running Debian GNU/Linux (National University Ireland Galway). For the parallel implementation of the algorithms presented, the Intel C Compiler v9.0 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).

Let us consider a 2D-boundary value problem with Dirichlet boundary conditions:

$$\begin{aligned} &u_{xx} + u_{yy} + u = F, \quad (x,y) \in R, \\ &u(x,y) = 0, \quad (x,y) \in \partial R, \end{aligned} \tag{43}$$

where R is the unit square 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 (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=2 and the width parameters were set to ℓ=3. The iterative process was terminated when $\|r_i\|_\infty \langle 10^{-5}$.

Table 1: Speedups and processors allocated of the **PaFiBo-NOROBAIFEM-2D** algorithm, for several values of δl, with n=10000 and m=101.

"Retention" parameter	**Speedups for the PaFiBo-NOROBAIFEM-2D algorithm**		
	Number of processors		
	2	**4**	**8**
δl=m	1.987	3.936	7.690
δl= 2m	1.992	3.967	7.809
δl=4m	1.995	3.971	7.842
δl=6m	1.999	3.972	7.880

The speedups and efficiencies of the **PaFiBo-NOROBAIFEM-2D** algorithm for several values of the "retention" parameter δl with n=10000, m=101, are given in Table 1 and 2 respectively.

Table 2: Efficiencies and processors allocated of the **PaFiBo-NOROBAIFEM-2D** algorithm, for several values of δl, with n=10000 and m=101.

"Retention" parameter	**Efficiencies for the PaFiBo-NOROBAIFEM-2D algorithm**		
	Number of processors		
	2	**4**	**8**
δl=m	0.994	0.984	0.961
δl= 2m	0.996	0.992	0.976
δl=4m	0.997	0.993	0.980
δl=6m	0.999	0.993	0.985

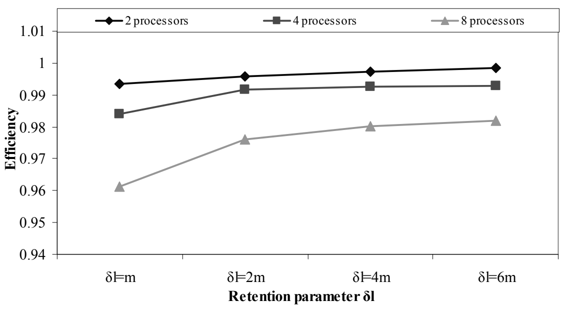


Figure 2: Parallel efficiency of the **PaFiBo-NOROBAIFEM-2D** algorithm, for several values of δl, with n=10000 and m=101.

In Fig. 1 and 2 the speedups and processors allocated for several values of the "retention" parameter δl, and the parallel efficiency for several values of the "retention" parameter δl are presented respectively for the **PaFiBo-NOROBAIFEM-2D** algorithm with n=10000, m=101.

The speedups and efficiencies of the **PNEPCGS** method for several values of the "retention" parameter δl with n=10000, m=101 are given in Table 3 and 4 respectively. In Fig. 3 and 4 the speedups and processors allocated for several values of the "retention" parameter δl, and the parallel efficiency for several values of the "retention" parameter δl are presented respectively for the **PNEPCGS** method with n=10000, m=101.

Table 3: Speedups and processors allocated of the **PNEPCGS** method, for several values of δl, with n=10000 and m=101.

"Retention" parameter	**Speedups for PNEPCGS**		
	Number of processors		
	2	**4**	**8**
δl= 1	1.480	1.537	1.677
δl= 2	1.901	2.691	3.353
δl= m	1.932	3.576	6.559
δl= 2m	1.939	3.625	6.608
δl= 4m	1.945	3.697	6.613
δl= 6m	1.956	3.736	6.700

Table 4: Parallel Efficiency of the **PNEPCGS** method, for several values of δl, with n=10000 and m=101.

"Retention" parameter	**Efficiency for PNEPCGS**		
	Number of processors		
	2	**4**	**8**
δl= 1	0.740	0.384	0.210
δl= 2	0.951	0.673	0.419
δl= m	0.966	0.894	0.820
δl= 2m	0.969	0.906	0.826
δl= 4m	0.973	0.924	0.827
δl= 6m	0.978	0.934	0.837

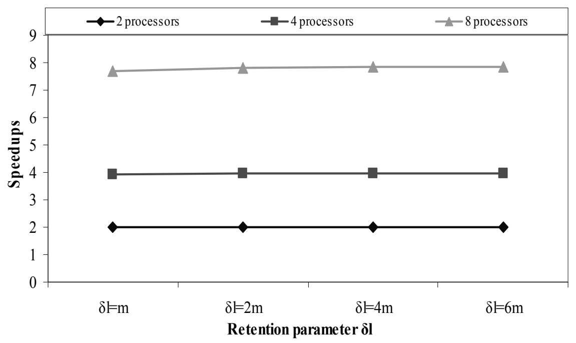


Figure 1: Speedups and processors allocated of the **PaFiBo-NOROBAIFEM-2D** algorithm, for several values of δl, with n=10000 and m=101.

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.

Additionally for large values of the "retention" parameter, i.e. multiples of the semi-bandwidth m, the speedups and the efficiency tend to the up-

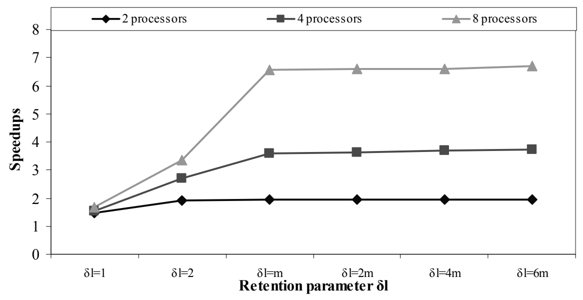


Figure 3: Speedups and processors allocated of the **PNEPCGS** method, for several values of δl, with n=10000 and m=101.

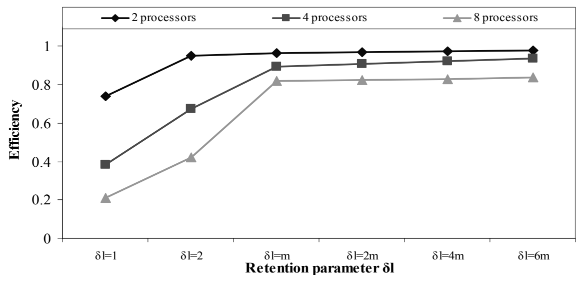


Figure 4: Parallel efficiency of the **PNEPCGS** method, for several values of δl, with n=10000 and m=101.

per theoretical bound, for the parallel normalized preconditioned conjugate gradient method, since the coarse granularity amortizes the parallelization overheads.

Finally, we state that the new parallel normalized approximate inverse preconditioned method, can be efficiently used for solving non-linear initially/boundary value problems on symmetric multiprocessor systems, and investigate the implementation on different computational platforms.

Acknowledgement: The authors would like to thank indeed Dr. John Morrison of the Department of Computer Science, University College of Cork for the provision of computational facilities and support through the WebCom-G project funded by Science Foundation Ireland.

References

Akcadogan, C.; Dag, H. (2003): A parallel implementation of chebyshev preconditioned conjugate gradient method. *Second International Symposium on Parallel and Distributed Computing*, IEEE, pp. 1-8.

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

Chan, T. F.; van der Vorst, H. A. (1997): Approximate and Incomplete Factorizations. In: Keyes, D. E., Sameh, A. and Venkatakrishnan, V. (eds), *Parallel Numerical Algorithms, ICASE/LaRC Interdisciplinary Series in Science and Engineering*, pp. 167-202. Kluwer, Dordrecht.

Cosgrove, J. D. F.; Dias, J. C.; Griewank, A. (1992): Approximate inverse preconditioning for sparse linear systems. *Inter. J. Comp. Math.*, vol. 44, pp. 91–110.

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

Dubois, P.; Greenbaum, A.; Rodrigue, G. (1979): Approximating the inverse of a matrix for use in iterative algorithms on vector processors. *Computing*, vol. 22, pp. 257–268.

Duff, I. (2000): The impact of high performance computing in the solution of linear systems: trends and problems. *J. Comp. Applied Math.*, vol. 123, pp. 515–530.

Grama, A.; Gupta, A.; Karypis, G.; Kumar, V. (2003): *Introduction to Parallel Computing*. Addison-Wesley.

Gravvanis, G. A. (2000): Explicit preconditioned generalized domain decomposition methods. *Inter. J. Appl. Math.*, vol. 4(1), pp. 57–71.

Gravvanis, G. A. (1998): Parallel matrix techniques. In: Papailiou, K., Tsahalis, D., Periaux, J., Hirsch, C., and Pandolfi, M., (eds), *Computational Fluid Dynamics*, Wiley, vol. 1, pp. 472-477.

Gravvanis, G. A.; Giannoutakis, K. M. (2006): Distributed finite element normalized approximate inverse preconditioning. *CMES: Computer Modeling in Engineering and Sciences*, vol. 16(2), pp. 69-82.

Gravvanis, G. A.; Giannoutakis, K. M. (2006): Parallel exact and approximate arrow-type inverses on symmetric multiprocessor systems. *ICCS 2006*, part I, LNCS 3991, pp. 506–513.

Gravvanis, G. A.; Giannoutakis, K. M. (2003): Normalized finite element approximate inverse preconditioning for solving non-linear boundary value problems. In: Bathe, K.J. (ed), *Computational Fluid and Solid Mechanics 2003*, Elsevier, vol. 2, pp. 1963-1967.

Grote, M. J.; Huckle, T. (1977): Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, vol. 18, pp. 838-853.

Ha, T.; Seo, S.; Sheen, D. (2006): Parallel iterative procedures for a computational electromagnetic modeling based on a nonconforming mixed finite element method. *CMES: Computer Modeling in Engineering and Sciences*, vol. 14(1), pp. 57-76.

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

Huckle, T. (1998): Efficient computations of sparse approximate inverses. *Numer. Linear Alg. with Appl.*, vol. 5, pp. 57-71.

Kolotilina, L. Y. and Yeremin, A. Y. (1993): Factorized sparse approximate inverse preconditioning. *SIAM J. Matrix Anal. Appl.*, vol. 14, pp. 45–58.

Lipitakis, E. A.; Evans, D. J. (1987): Explicit semi-direct methods based on approximate inverse matrix techniques for solving boundary value problems on parallel processors. *Math. and Comput. in Simulation*, vol. 29, pp. 1-17.

Lipitakis, E. A.; Evans, D. J. (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.

Lipitakis, E. A.; Gravvanis, G. A. (1995): Explicit preconditioned iterative methods for solving large unsymmetric finite element systems. *Computing*, vol. 54(2), pp. 167-183.

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

Yuan, W.; Chen, P.; Liu, K. (2007): A New Quasi-Unsymmetric Sparse Linear Systems Solver for Meshless Local Petrov-Galerkin Method (MLPG). *CMES: Computer Modeling in Engineering and Sciences*, vol.17(2), pp. 115-134.