On the Performance of Parallel Normalized Explicit Preconditioned Conjugate Gradient Type Methods

George A. Gravvanis and Konstantinos M. Giannoutakis

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

Abstract

A new class of parallel normalized preconditioned conjugate gradient type methods in conjunction with normalized approximate inverses algorithms, based on normalized approximate factorization procedures, for solving sparse linear systems of irregular structure, which are derived from the finite element method of a two dimensional boundary value problem, is introduced. Parallel normalized explicit preconditioned conjugate gradient - type methods for distributed memory systems based on the block – row distribution (for the vectors and the explicit approximate inverse), using Message Passing Interface (MPI) communication library, is also presented with theoretical estimates on speedups and efficiency, in order to examine the parallel behavior of these methods using normalized explicit approximate inverses as the suitable preconditioner. Collective communications have been utilized at the synchronization points and non – blocking communications have been used, where the exchanging of messages can be overlapped with computations, where applicable. Application of the methods on a two dimensional boundary value problem is discussed and numerical results are given, concerning the parallel performance in terms of speedups and efficiency.

1. Introduction

Many engineering and scientific problems are described by sparse linear systems of algebraic equations derived from the Finite Element (FE) discretization of partial differential 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 supercomputing time, has had some effect on the design of modern computer systems.

An important achievement over the last decades is the appearance and use of preconditioned iterative methods, for solving a linear system Au=s, [2, 4, 9, 10, 11, 17, 18, 21, 22]. The preconditioned form of the linear system is MAu=Ms, where M is a suitable preconditioner, satisfying the following conditions: (i) MA should have a "clustered" spectrum, (ii) M can be efficiently computed in parallel and (iii) finally "M × vector" should be fast to compute in parallel, [9, 11, 12, 17].

Many researchers have tried to provide preconditioned iterative methods, based on splitting techniques, factorization techniques (based on modifications of Gaussian Elimination), level-scheduling or wavefront approach, polynomial preconditioners, red-black ordering and factorized sparse approximate inverses, which were either difficult to implement on parallel systems or of limited potential and success [21]. Further many researchers have proposed and discussed parallel issues of conjugate gradient methods [2, 6, 7, 8, 9, 10, 21, 22]. Additionally sparse approximate inverses by minimizing the Frobenious norm of the error have been presented and can be implemented on parallel systems [17, 21]. In recent years explicit preconditioned methods, based on approximate inverse matrix algorithms, have been used for solving efficiently sparse linear systems, [11]. The effectiveness of the explicit approximate inverse preconditioning method is related to the fact that the derived classes of approximate inverses exhibit a similar "fuzzy" structure as the coefficient matrix and are close approximants to the coefficient matrix, [11, 13].

The cost-effectiveness of parallel explicit preconditioned iterative schemata over parallel direct solution methods for solving large sparse linear systems is now commonly accepted. It is known that adaptive approximate factorization and approximate inverse matrix algorithms are in general tediously complicated. However as the demand for solving boundary value problems grows, the need to use efficient sparse finite element (FE) linear equations solvers based on approximate

1-4244-0054-6/06/$20.00

factorization procedures and approximate inverse algorithms becomes one of great importance, [11].

The main motive for the derivation of the approximate inverse matrix algorithms is that they can be efficiently used in conjunction with explicit preconditioned conjugate gradient – type schemes. The computationally dominant part of operations involved in these methods (i.e. approximate inverse × vector) can be efficiently parallelized on multiprocessor and multicomputer systems, since the approximate inverse matrix has been computed explicitly avoiding the usage of forward-backward substitution, which can not parallelize well, [21]. For the parallelization of the conjugate gradient - type methods on multicomputer systems, the block – row distribution has been used for the vectors and the explicit approximate inverse. Collective communications have been utilized at the synchronization points and non – blocking communications have been used, where the exchanging of messages can be overlapped with computations, where applicable.

In Section 2, normalized approximate inverse finite element matrix algorithmic methods are presented, based on normalized approximate factorization procedures of the coefficient finite element matrix. In Section 3, parallelization issues of the normalized explicit preconditioned conjugate gradient type methods are discussed, for distributed memory parallel systems, using the MPI communication library. Finally, in Section 4 the performance in terms of speedups and efficiency of the parallel normalized explicit preconditioned conjugate gradient variants is illustrated by solving sparse finite element linear system on a distributed system.

2. Normalized Approximate Inverses

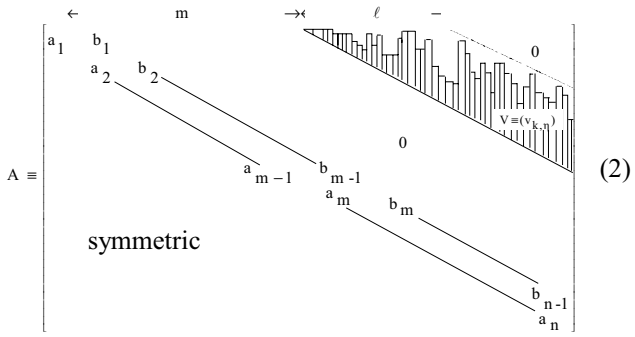
In this section we present normalized explicit approximate inverse finite element matrix techniques by computing the elements of a class of normalized approximate inverses, [13, 14, 15, 16].

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

$$A\,u = s \qquad (1)$$

where A is a sparse, diagonal dominant, positive definite, symmetric (n×n) matrix of irregular structure (with all the off-center band terms grouped into a regular band of width ℓ), 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 boundary conditions.

Let us now assume the normalized approximate factorization of the coefficient matrix A such that, viz.

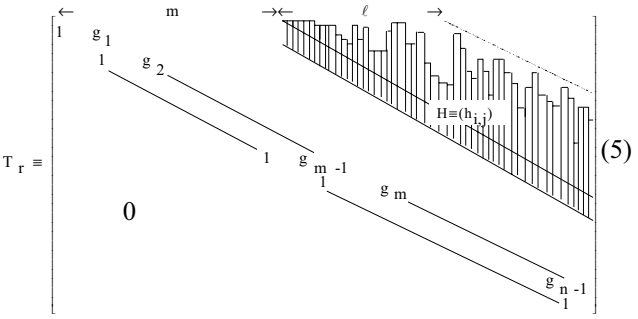


(2)

$$A \approx D_r T_r^t T_r D_r, \quad r \in [1,\ldots,m-1), \qquad (3)$$

where r is the "fill-in" parameter, i.e. the number of outermost off-diagonal entries at semi-bandwidth m, D_r is a diagonal matrix, T_r is a sparse upper triangular matrix of the same profile as the coefficient matrix A, i.e.

$$D_r \equiv \mathrm{diag}(d_1, d_2, \ldots, d_n) \qquad (4)$$



(5)

The elements of the decomposition factors D_r and T_r can be computed by the **FEANOF-2D** algorithm, [18]. The memory requirements of the **FEANOF-2D** algorithm are $\approx (r + 2\ell + 2)n$ words, while the computational work required is $\approx 1/2\,(r + \ell)(r + \ell + 3)n$ multiplicative operations + n square roots, [18].

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

$$M_r^{\delta l} = \left(D_r T_r^t T_r D_r \right)^{-1} = D_r^{-1} \left(T_r^t T_r \right)^{-1} D_r^{-1} = D_r^{-1} \hat{M}_r^{\delta l} D_r^{-1} \qquad (6)$$

where δl is the "retention" parameter, i.e. the additional mumber of diagonals retained next to the main diagonal in the lower and upper part of the inverse, [11, 12, 13, 14, 15, 16].

A class of approximate inverses can be obtained by solving recursively the following systems:

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

Then, the elements of the normalized approximate inverse can be computed by the **Nor**malized **O**ptimized **A**pproximate **I**nverse **F**inite **E**lement **M**atrix (**NOROAIFEM-2D**) algorithm, [15]. The memory requirements of the **NOROAIFEM-2D** algorithm are $(2\delta l-1)\times n$ words, using a moving window shifted from bottom to top. The computational work of the **NOROAIFEM-2D** algorithm is $\approx O[(r+\ell+1)\delta l]n$ multiplicative operations.

It should be noted that this class of 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:

$$\underline{\text{class I}} \qquad \underline{\text{class II}}$$

$$A^{-1} \equiv D^{-1}\hat{M}D^{-1} \leftarrow D_r^{-1}\hat{M}_{r=m-1}^{\delta l}D_r^{-1}$$

$$\underline{\text{class III}} \qquad \underline{\text{class IV}} \qquad \underline{\text{class V}}$$

$$\leftarrow D_r^{-1}\hat{M}_{r=m-1}^{\delta l}D_r^{-1} \leftarrow D_r^{-1}\hat{M}_r^{\delta l}D_r^{-1} \leftarrow D_r^{-2}, \quad (8)$$

where the entries of the class I inverse results in a direct method, i.e. $r=m-1$ and $\delta l=n$, with the disadvantage of high memory requirements and computational work for large order systems. The entries of the class II inverse have been retained after the computation of the exact inverse ($r=m-1$). The entries of the class III inverse have been computed and retained during the computational procedure of the (approximate) inverse ($r=m-1$), while the entries of the class IV inverse have been retained after the computation of the approximate inverse ($r \le m-1$). The class V of the normalized approximate inverse retains only the diagonal elements, i.e. $\delta l=1$ hence $\hat{M}_r^{\delta l} \equiv I$, resulting in a fast inverse algorithm.

The parallel construction of similar approximate inverses has been studied and implemented in [12], and is under further investigation.

It should be noted that if the width-parameter is set to $\ell = 1$, (2), then the above mentioned approximate factorization and inverse algorithms reduce to the corresponding ones, namely the **NOBAR-2D** and **NOROBAIM-2D** algorithm, for solving linear systems of semi-bandwidth m, which is encountered usually in solving 2D boundary value problems by the finite difference method.

3. Parallel Normalized Explicit Preconditioned Conjugate Gradient Type Methods

In this section we present a class of normalized explicit preconditioned conjugate gradient - type schemes, based on the normalized finite element approximate inverses, yielding a class of efficient parallel explicit preconditioned schemes, [13, 15, 16].

In the following we present a modified form of the Chronopoulos-Gear variant of the Conjugate Gradient method, [5, 9], henceforth called the **N**ormalized **Ex**plicit **P**reconditioned **C**onjugate **G**radient – **Ch**ronopoulos **G**ear **variant** (**NEPCG-ChG-variant**) method, for solving linear systems and can be expressed by the following compact scheme:

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

compute $r_0 = s - Au_0$, (9)

set $q_{i-1} = p_{i-1} = 0$ and $\beta_{-1} = 0$ (10)

form $w_0 = D_r^{-1}\hat{M}_r^{\delta l}D_r^{-1}r_0$, (11)

$s_0 = Aw_0$ (12)

calculate $\rho_0 = (r_0, w_0)$, $\mu_0 = (s_0, w_0)$ (13)

and $\alpha_0 = \rho_0/\mu_0$ (14)

Then, for i=0, 1, ..., (until convergence) compute the vectors p_i, q_i, u_{i+1}, r_{i+1}, w_{i+1}, s_{i+1} and the scalar quantities α_{i+1}, β_i, μ_{i+1}, ρ_{i+1} as follows:

compute $p_i = w_i + \beta_{i-1}p_{i-1}$, (15)

$q_i = s_i + \beta_{i-1}q_{i-1}$, (16)

$u_{i+1} = u_i + \alpha_i p_i$, (17)

$r_{i+1} = r_i - \alpha_i q_i$ (18)

Then, form $w_{i+1} = D_r^{-1}\hat{M}_r^{\delta l}D_r^{-1}r_{i+1}$ (19)

form $s_{i+1} = Aw_{i+1}$ (20)

set $\rho_{i+1} = (r_{i+1}, w_{i+1})$, (21)

$\mu_{i+1} = (s_{i+1}, w_{i+1})$ (22)

evaluate $\beta_i = \rho_{i+1}/\rho_i$, (23)

$$\alpha_{i+1} = \frac{\rho_{i+1}}{\mu_{i+1} - \rho_{i+1}\beta_i/a_i} \quad (24)$$

The computational work required for the **NEPCG - ChG - variant** method is $\approx O[(2\delta l+2\ell+10)n$ mults + $5n$ adds$]\nu$, 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, can be expressed by the following compact scheme:

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

set $u_0 = 0$ (25)

compute $r_0 = s - Au_0$, (26)

set $r_0^{'} = r_0,\ \rho_0 = \alpha = \omega_0 = 1$, (27)

$v_0 = p_0 = 0.$ (28)

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

calculate $\rho_i = \left(r_0^{'}, r_{i-1}\right)$ (29)

$\beta = \left(\rho_i / \rho_{i-1}\right) / \left(\alpha / \omega_{i-1}\right)$ (30)

compute $p_i = r_{i-1} + \beta\left(p_{i-1} - \omega_{i-1} v_{i-1}\right)$, (31)

form $y_i = D_r^{-1} \hat{M}_r^{\delta l} D_r^{-1} p_i$, (32)

and $v_i = Ay_i,\ \alpha = \rho_i \Big/ \left(r_0^{'}, v_i\right)$, (33)

$x_i = r_{i-1} - \alpha v_i$, (34)

form $z_i = D_r^{-1} \hat{M}_r^{\delta l} D_r^{-1} x_i \qquad t_i = Az_i$, (35)

set $\omega_i = \dfrac{\left(D_r^{-1} \hat{M}_r^{\delta l} D_r^{-1} t_i,\ D_r^{-1} \hat{M}_r^{\delta l} D_r^{-1} x_i\right)}{\left(D_r^{-1} \hat{M}_r^{\delta l} D_r^{-1} t_i,\ D_r^{-1} \hat{M}_r^{\delta l} D_r^{-1} t_i\right)}$, (36)

compute $u_i = u_{i-1} + \alpha y_i + \omega_i z_i$ (37)

$r_i = x_i - \omega_i t_i$. (38)

The computational complexity of the **NEPBICG-STAB** method is $\approx O[(6\delta l + 4\ell + 22)n$ mults + $6n$ 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 **NOROAIFEM** algorithm is related to the fact that the normalized approximate inverse exhibits a similar "fuzzy" structure as the coefficient matrix A.

The convergence analysis and computational complexity of normalized approximate inverse preconditioning has been presented in [14].

For the parallel implementation of the **NEPCG-ChG-variant** and **NEPBICG-STAB** methods (henceforth called **PNEPCG-ChG-variant** and **PNEPBICG-STAB** respectively), the Message Passing Interface (MPI) communication library was utilized.

Let no_proc denote the number of processors available. Then, the two most computationally dominating operations of the normalized explicit preconditioned conjugate gradient - type schemes (i.e. multiplication of the normalized 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. Each processor is assigned to a strip of elements (from the (myrank*local_n+1)-th to the (myrank* local_n + local_n)-th row) of the normalized approximate inverse and vectors, and performs all the necessary operations, where local_n=n/no_proc.

During each iteration, communication operations are required before matrix × vector and after inner product computations. The collective communication routines MPI_Allreduce and MPI_Allgather were used for sending and receiving data among distributed processes, [19, 20]. The parallel algorithm of the **NEPBICG-STAB** method has been presented in [13].

An essential modification in order to improve performance, by overlapping some computations during exchanging messages, can only be adopted for the **PNEPCG - ChG - variant** method, [9]. For example, the computations of equation (22) can be concurrently executed with the reduction operation required for gathering the "partial" sums of the inner product (21). Then, a simple collective communication step is used to broadcast the final sum to all processors.

The theoretical estimates on speedups and efficiency for the **PNEPCG-ChG-variant** and **PNEPBICG-STAB** methods can be similarly obtained, as in [13]. Thus,

$$S_p = \frac{1}{\frac{1}{no\_proc} + \frac{at_s \log(no\_proc)}{O(\delta l) nt_m} + \frac{b(no\_proc - 1)t_w}{O(\delta l) no\_proc}}, \quad (39)$$

and

$$E_p = \frac{1}{1 + \frac{at_s\, no\_proc\, \log(no\_proc)}{O(\delta l) nt_m} + \frac{b(no\_proc - 1)t_w}{O(\delta l)}}, \quad (40)$$

where t_s denotes the message latency, t_w the time necessary for a word to be sent, t_m the computational time of one multiplication, and a, b are parameters whose values depend on the number of collective communications required during each iteration (e.g. for the **PNEPBICG-STAB**, a=5 and b=2), [13].

Hence, for $\delta l \to n$ and $n \to \infty$, it is evident that $S_p \to no\_proc$ and $E_p \to 1$, which are the theoretical upper bounds, [1, 19, 20].

4. Numerical Results

In this section we examine the applicability and

effectiveness of the normalized explicit preconditioned conjugate gradient – type schemes for solving characteristic two dimensional boundary value problems, on distributed memory machines, using Message Passing Interface communication library (MPI), [19, 20].

Let us consider the following 2D-model problem

$$\Delta u(x, y) + u(x, y) = f(x, y), \quad (x, y) \in R, \qquad (41)$$

subject to boundary conditions

$$u(x,y) = 0, \quad (x,y) \in \partial R, \qquad (41.a)$$

where Δ is the Laplacian operator, R is the unity square and ∂R is the boundary of the domain R. The domain $R \cup \partial R$ is covered by a non-overlapping triangular network. The **PNEPCG-ChG-variant** and the **PNEPBICG-STAB** methods were terminated when $\left\| r_i \right\|_\infty < 10^{-5}$.

The numerical test runs were performed on a cluster, which consists of fifty (50) dual Intel Pentium III servers, running at 1.26 GHz with 512K cache, 1GB ram and 133MHz bus. Each server has a gigabit network interface and is located in a chassis (9 total chassis, 8 with 6 servers and one chassis with only 2 servers). Within each chassis the servers are connected by a LOM (Lan On Motherboard). This LOM has 6 Ethernet ports shared amongst the servers in the chassis. Each chassis has an external 4 port gigabit switch, which is connected to a gigabit switch (and internally on the 6 port LOM).

The speedups and the number of iterations of the **PNEPCG-ChG-variant** and the **PNEPBICG-STAB** method for several values of the "retention" parameter δl with n=10000, m=101, r=2 and $\ell = 3$, are given in Table 1 and 2 respectively. It should be noted that these speedups do not take into account the performance of the construction of the approximate inverse. It should be also noted that these results presented are in qualitative agreement with the theoretical results given.

In Figure 1, 2, 3, 5, 6 and 7 the speedups and processors allocated for several values of δl, the speedups versus the "retention" parameter δl for several numbers of processors and the parallel efficiency for several values of δl is presented for the **PNEPCG-ChG-variant** and the **PNEPBICG-STAB** method respectively, with n=10000, m=101, r=2 and $\ell = 3$. In Figure 4 and 8 the overall performance evaluation measurements of the **PNEPCG-ChG-variant** and the **PNEPBICG-STAB** method are given respectively, with n=10000, m=101, r=2 and $\ell = 3$.

As it is verified by our experimental results, the parallel behavior of the **PNEPCG-ChG-variant** is better than the **PNEPBICG-STAB** method for large values of the "retention" parameter δl, as it was expected, because of the non-blocking communications that was adopted, [9]. Additionally, in the **PNEPCG-ChG-variant** all the vectors need to be loaded only once during each iteration, which leads to a better exploitation of the data (improved data locality), [5, 9]. For large values of the "retention" parameter, i.e. multiples of the semi-bandwidth m, the speedups and the efficiency tend to become optimum, which is in qualitative agreement with the theoretical results presented. For small values of the "retention" parameter δl, the communication cost is responsible for such performance.

Since there is increased communication cost for small values of the "retention parameter" δl and further taking into account the time consumption of the application it is recommended that large values of the "retention parameter" δl should be chosen.

Based on the derived theoretical estimates, in order to approach the optimum value of speedup as δl increases, the utilized number of processors should be increased.

When coarse grain parallelism is adopted, i.e. when δl is increased, a reduction in iterations was achieved along with an increase in the overall speedup. This is because by increasing δl, the approximate inverse tends to become the exact inverse (class I), (8).

"Retention" parameter	**Speedups**				**Number of iterations**
	Number of processors				
	2	**4**	**8**	**16**	
δl=1	1.3058	0.5849	0.5700	0.4972	19
δl=2	1.3121	0.7682	0.7338	0.6502	18
δl=m	1.9153	3.0099	4.4234	5.8597	13
δl=2m	1.9652	3.8352	5.8905	9.6369	9
δl=4m	1.9786	3.9559	7.3386	12.6621	6
δl=6m	1.9999	3.9690	7.5111	13.5700	5

Table 1. Speedups and processors allocated of the PNEPCG-ChG-variant method, for several values of δl, with n=10000, m=101 and r=2.

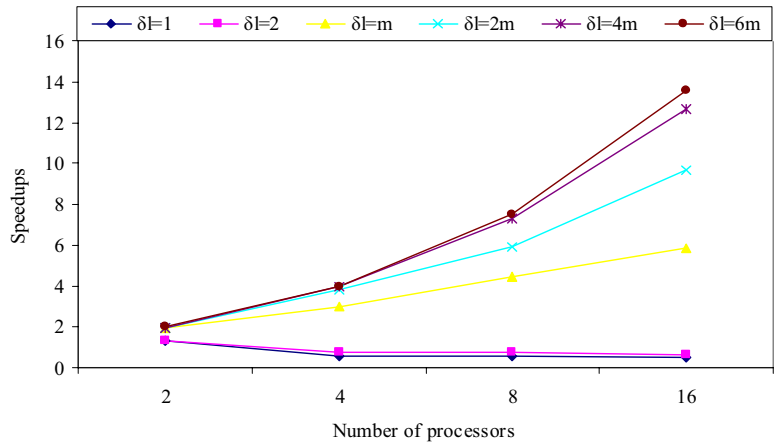


Figure 1. Speedups and processors allocated of the PNEPCG-ChG-variant method for several values of δl, with n=10000 and m=101.

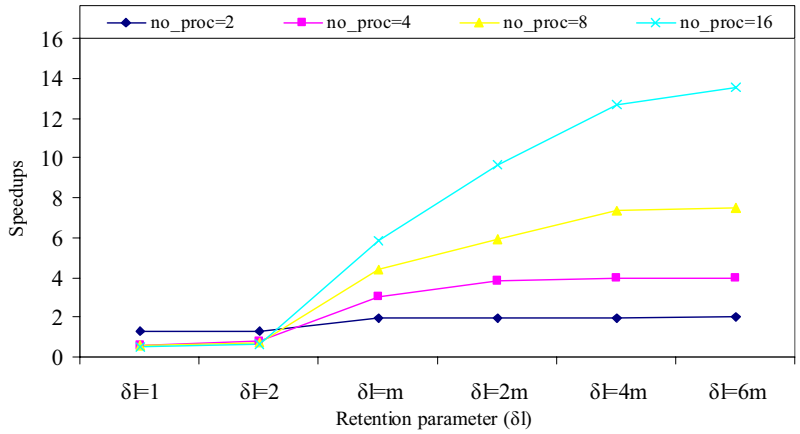


Figure 2. Speedups versus the "retention" parameter δl of the PNEPCG-ChG-variant method for several numbers of processors, with n=10000 and m=101.

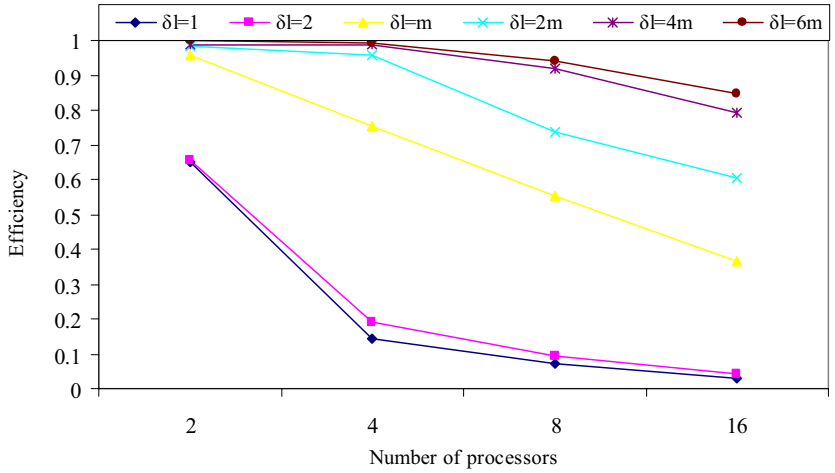


Figure 3. Parallel efficiency and processors allocated of the PNEPCG-ChG-variant method for several values of δl, with n=10000 and m=101.

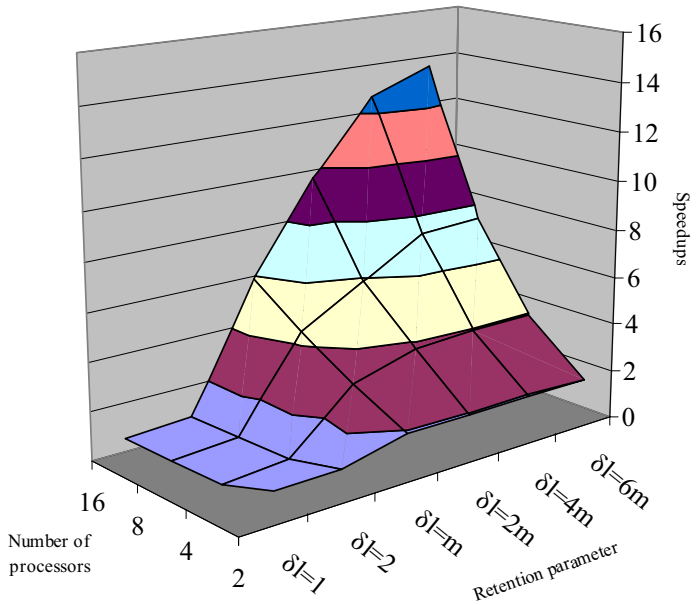


Figure 4. Performance evaluation measurements of the PNEPCG-ChG-variant method, with n=10000 and m=101.

"Retention" parameter	Speedups				Number of iterations
	Number of processors				
	2	4	8	16	
δl=1	1.2912	1.1752	1.0633	0.9288	13
δl=2	1.3387	1.1844	1.1069	0.9437	11
δl=m	1.9360	3.1828	5.3622	6.5735	8
δl=2m	1.9349	3.3452	5.9481	7.2508	5
δl=4m	1.9540	3.6554	6.4035	9.8555	3
δl=6m	1.9636	3.7162	7.1375	11.1503	3

Table 2. Speedups and processors allocated of the PNEPBICG-STAB method, for several values of δl, with n=10000, m=101 and r=2.

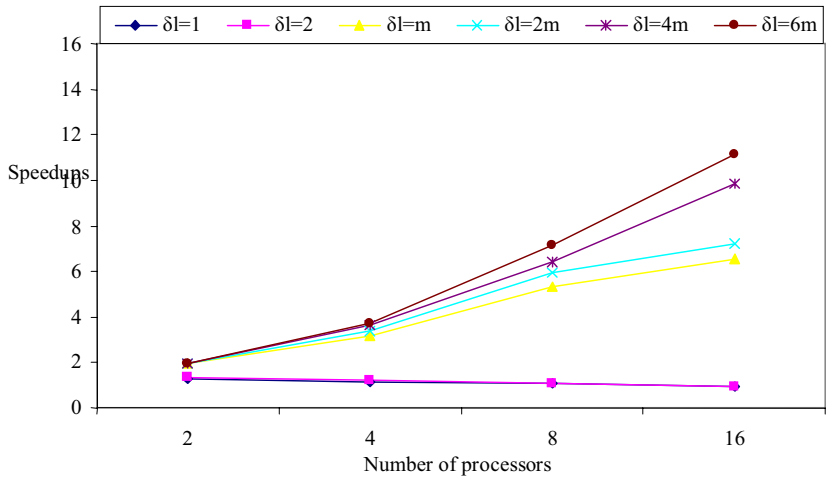


Figure 5. Speedups and processors allocated of the PNEPBICG-STAB method for several values of δl, with n=10000 and m=101.

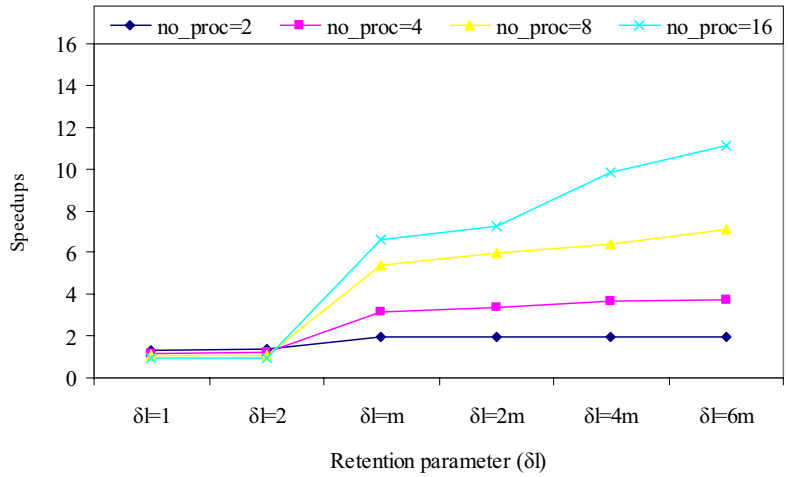


Figure 6. Speedups versus the "retention" parameter δl of the PNEPBICG-STAB method for several numbers of processors, with n=10000 and m=101.

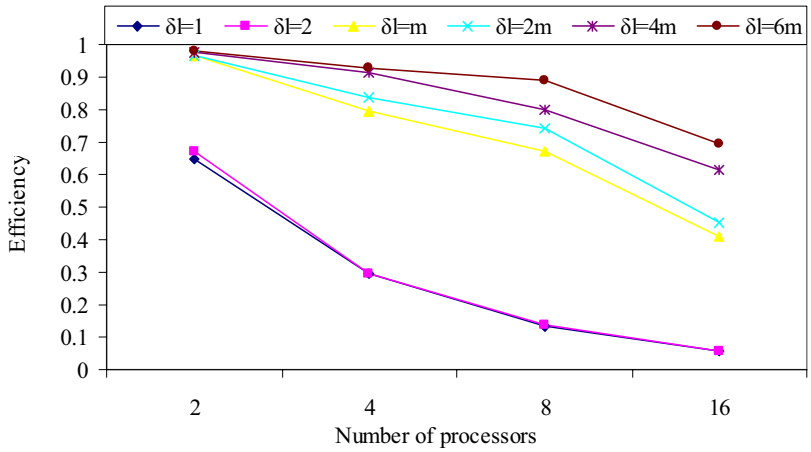


Figure 7. Parallel efficiency and processors allocated of the PNEPBICG-STAB method for several values of δl, with n=10000 and m=101.

Similar results concerning the speedups and efficiency are expected for other problems and other platforms. It should be mentioned that similar results concerning the speedups and efficiency have been presented using MPICH on different operating systems (Linux, Windows), MPICH over Globus environment and Remote Internet Interface (over Microsoft Windows operating system), [3].

Finally, in order to overcome inefficiencies in terms of the performance of the parallel normalized explicit preconditioned conjugate gradient methods, for small values of the "retention" parameter δl, symmetric multiprocessor systems are recommended, [16].

Acknowledgment

The authors would like to thank indeed Dr. John Morrison of the Department of Computer Science, University College of Cork for proposing and allowing us to perform the test runs on the Departments' cluster.

References

[1] S. G. Akl. Parallel computation: models and methods. Prentice Hall, 1997.

[2] O. Axelsson, G. F. Carey and G. Lindskog. On a class of preconditioned iterative methods for parallel computers. *Inter. J. Numer. Meth. Eng.*, 27: 637-654, 1989.

[3] M. P. Bekakos, G. A. Gravvanis, E. S. Bazoukis and K. M. Giannoutakis. Towards a pilot grid platform for internet high performance computations, In "Grid Technologies: Emerging from Distributed Architectures to Virtual Organizations", M.P. Bekakos, G.A. Gravvanis and H.R. Arabnia (eds.), WIT Press, 2006, to appear.

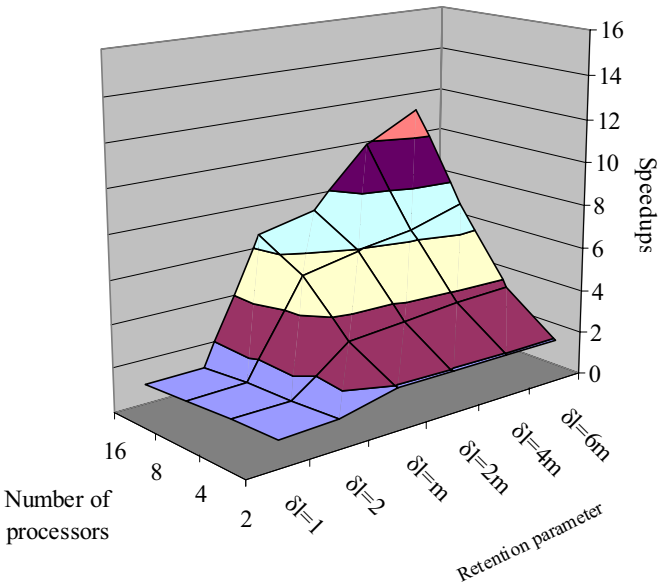


Figure 8. Performance evaluation measurements of the PNEPBICG-STAB method, with n=10000 and m=101.

[4] M. Benzi. Preconditioning techniques for large linear systems: A survey. *J. of Comp. Physics*, 182: 418-477, 2002.

[5] A. T. Chronopoulos and C. W. Gear. s-Step iterative methods for symmetric linear systems. *J. Comput. Appl. Math.*, 25: 153-168, 1989.

[6] E. D'Azevedo, V. Eijkhout and C. Romine. A matrix framework for conjugate gradient methods and some variants of CG with less synchronization overhead. *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing,* pages 644-646, 1993.

[7] C. R. de Brozolo and Y. Robert. Parallel and vector conjugate gradient-like algorithms for sparse nonsymmetric linear systems. *Proceedings of the 2nd International Conference on Supercomputing*, pages 478-487, 1988.

[8] J. Demmel, M. Heath and H. van der Vorst. Parallel numerical linear albegra. In *Acta Numerica 1993*, Cambridge University Press, Cambridge, 1993.

[9] J. J. Dongarra, I. Duff, D. Sorensen and H. A. van der Vorst. Numerical Linear Algebra for High-Performance Computers. SIAM, 1998.

[10] G. H. Golub and C. van Loan. Matrix Computations, The Johns Hopkins University Press, 1996.

[11] G. A. Gravvanis. Explicit approximate inverse preconditioning techniques. *Archives of Computational Methods in Engineering*, 9(4): 371-402, 2002.

[12] G. A. Gravvanis. Parallel matrix techniques. *Computational Fluid Dynamics*. K. Papailiou, D. Tsahalis, J. Periaux, C. Hirsch, M. Pandolfi eds., volume I, pages 472-477, Wiley, 1998.

[13] G. A. Gravvanis and K.M. Giannoutakis. Normalized Explicit Preconditioned methods for solving 3D boundary value problems on uniprocessor and distributed systems. *Inter. J. Numer. Meth. Eng.*, 65(1): 84-110, 2006.

[14] G. A. Gravvanis and K.M. Giannoutakis. On the rate of convergence and complexity of finite element normalized explicit approximate inverse preconditioning. In: K.J. Bathe, (editor), *Computational Fluid and Solid Mechanics 2003*, volume 2, pages 1963-1967, Elsevier, 2003.

[15] G. A. Gravvanis and K.M. Giannoutakis. Normalized finite element approximate inverse preconditioning for solving non-linear boundary value problems. In: K.J. Bathe, (editor), *Computational Fluid and Solid Mechanics 2003*, volume 2, pages 1958-1962, Elsevier, 2003.

[16] G. A. Gravvanis, K.M. Giannoutakis and M. P. Bekakos. Parallel finite element approximate inverse preconditioning on symmetric multiprocessor systems. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, H. R. Arabnia eds, volume I, pages 168-175, CSREA Press, 2004.

[17] M. J. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.* 18: 838-853, 1997.

[18] E. A. Lipitakis and D. J. Evans. Solving linear finite element systems by normalized approximate matrix factorization semi-direct methods. *Computer Methods in Applied Mechanics & Engineering*, 43: 1-19, 1984.

[19] P. S. Pacheco. Parallel programming with MPI. Morgan Kaufmann Publishers, 1997.

[20] M. J. Quinn. Parallel Programming in C with MPI and OpenMP. Mc-Graw Hill, 2003.

[21] Y. Saad and H. A. van der Vorst. Iterative solution of linear systems in the 20th century. *J. Comp. Applied Math.* 123: 1-33, 2000.

[22] H. A. van der Vorst. High performance preconditioning, *SIAM J. Sci. Stat. Comput.*, 10: 1174-1185, 1989.