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 “$M \times \text{vector}$” 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
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 \( \times \) 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 inverse finite element matrix techniques by

\[ A = D_r T_r^T D_r, \quad r \in [1, n-1], \]

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.

\[ D_r = \text{diag}(d_1, d_2, \ldots, d_n) \]

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 \( = (r + 2\ell + 2)n \) words, while the computational work required is \( = 1/2(r + \ell)(r + \ell + 3)n \) multiplicative operations + \( n \) square roots, [18].

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

\[ M^l r = \left( D_r T_r^T D_r \right)^{-1} = D_r^{-1} T_r^T D_r^{-1} = D_r^{-1} M_r^l D_r^{-1} \]

where \( \delta l \) is the “retention” parameter, i.e. the additional number 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:

\[ M^l r T_r^T r = \left( T_r \right)^{-1} \quad \text{and} \quad T_r M^l r = \left( T_r \right)^{-1} \]

\[ (1) \]

\[ (2) \]

\[ (3) \]

\[ (4) \]

\[ (5) \]

\[ (6) \]

\[ (7) \]
Then, the elements of the normalized approximate inverse can be computed by the Normalized Optimized Approximate Inverse Finite Element Matrix (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 \(=O((r+\ell+1)\delta l)n\) 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:

\[
\begin{align*}
\text{class I} & \quad \text{class II} \\
A^{-1} = D^{-1}M^{-1} & \quad D^{-1}M^{-1} \\
\text{class III} & \quad \text{class IV} & \quad \text{class V} \\
D^{-1}M^{-1}r_{m-1} & \quad D^{-1}M^{-1}r_{m-1} & \quad D^{-1}M^{-1}r_{m-1} \\
& \quad D^{-2}r & \quad D^{-2}r
\end{align*}
\]

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 (\(r \leq m-1\)). The class V of the normalized approximate inverse retains only the diagonal elements, i.e. \(\delta l = 1\) hence \(M^{-1}r = 1\), 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-parameters 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 Normalized Explicit Preconditioned Conjugate Gradient – Chronopoulos Gear 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** \(r_0 = s - Au_0\),
- **set** \(q_{i-1} = p_{i-1} = 0\) and \(\beta_{i-1} = 0\),
- **form** \(w_0 = D^{-1}M^{-1}D^{-1}r_0\),
- **calculate** \(\rho_0 = (q_0, w_0)\), \(\mu_0 = (s_0, w_0)\),
- **and** \(a_0 = \rho_0/\mu_0\).

Then, for \(i = 0, 1, \ldots\), (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 \(\alpha_{i+1}\), \(\beta_{i+1}\), \(\mu_{i+1}\), \(\rho_{i+1}\) as follows:

- **compute** \(p_i = w_i + \beta_{i-1}p_{i-1}\), \(q_i = s_i + \beta_{i-1}q_{i-1}\), \(u_{i+1} = u_i + \alpha_ip_i\), \(r_{i+1} = r_i - \alpha iq_i\),
- **form** \(w_{i+1} = D^{-1}M^{-1}D^{-1}r_{i+1}\),
- **set** \(\rho_{i+1} = (q_{i+1}, w_{i+1})\), \(\mu_{i+1} = (s_{i+1}, w_{i+1})\),
- **evaluate** \(\beta_{i+1} = \rho_{i+1}/\mu_{i+1}\), \(\alpha_{i+1} = \frac{\rho_{i+1}}{\mu_{i+1} - \beta_{i+1}a_{i+1}}\),

The computational work required for the NEPCG-ChG-variant method is \(=O((2\delta l + 2\ell + 10)n\text{mults} + 5n\text{adds})\), where \(v\) denotes the number of iterations required for convergence to a predetermined tolerance level.

The Normalized Explicit Preconditioned BiConjugate Gradient-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,
Then, for i=0, 1, ..., (until convergence) compute the vectors \( u_i, r_i \) and the scalar quantities \( \alpha, \beta, \omega_i \) as follows:

\[
\text{calculate} \quad \rho_i = \left( r_i^0, r_i^{-1} \right) \\
\text{compute} \quad p_i = r_i + \beta \left( \rho_i, r_i^{-1} \right) \\
\text{form} \quad y_i = D^{-1}r_i M^{-1}r_i p_i , \\
\text{and} \quad v_i = A y_i , \quad \alpha = \rho_i \left( r_i^0, v_i \right) , \\
x_i = r_i - \alpha v_i , \\
\text{form} \quad z_i = D^{-1}r_i M^{-1}D^{-1}y_i , \quad t_i = A z_i , \\
\text{set} \quad \omega_i = \left( D^{-1}M^{-1}D^{-1}, D^{-1}M^{-1}D^{-1} \right) , \\
\text{compute} \quad u_i = u_i + \alpha y_i + \omega_i z_i , \\
r_i = x_i - \omega_i t_i .
\]

The computational complexity of the NEPBICG-STAB method is \( O((l+4)^2) \) in mults + 6in adds\( v \) operations, where \( v \) 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 \( \text{no\_proc} \) 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 \( (\text{myrank} \times \text{local}_n + 1) \)-th to the \( (\text{myrank} \times \text{local}_n + \text{local}_n) \)-th row) of the normalized approximate inverse and vectors, and performs all the necessary operations, where \( \text{local}_n = \text{no\_proc} \).

During each iteration, communication operations are required before matrix \( \times \) 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 NEPCG-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 NEPCG-ChG-variant and NEPBICG-STAB methods can be similarly obtained, as in [13]. Thus,

\[
S_p = \frac{1}{1 + \frac{\text{at \ log(\text{no\_proc})}}{\text{no\_proc}} + \frac{b(\text{no\_proc} - 1)t_w}{O(\delta l)\text{no\_proc}}} , \quad (39)
\]

and

\[
E_p = \frac{1}{1 + \frac{\text{at \ log(\text{no\_proc})}}{\text{no\_proc}} + \frac{b(\text{no\_proc} - 1)t_w}{O(\delta l)}} , \quad (40)
\]

where \( t_m \) 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 NEPBICG-STAB, \( a=5 \) and \( b=2 \), [13].

Hence, for \( \delta l \to \infty \) and \( n \to \infty \), it is evident that \( S_p \to \text{no\_proc} \) and \( E_p \to \infty \), 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

\[
\Delta u(x, y) + u(x, y) = f(x, y), \quad (x, y) \in \mathbb{R}, \tag{41}
\]

subject to boundary conditions

\[
u(x, y) = 0, \quad (x, y) \in \partial \mathbb{R}, \tag{41.a}
\]

where \(\Delta\) is the Laplacian operator, \(\mathbb{R}\) is the unity square and \(\partial \mathbb{R}\) is the boundary of the domain \(\mathbb{R}\). The domain \(\mathbb{R} \cup \partial \mathbb{R}\) is covered by a non-overlapping triangular network. The \textit{PNEPCG-ChG-variant} and the \textit{PNEPBICG-STAB} methods were terminated when \(\| \mathbf{r} \|_\infty < 10^{-5}\).

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 \textit{PNEPCG-ChG-variant} and the \textit{PNEPBICG-STAB} method for several values of the “retention” parameter \(\delta l\), 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 \(\delta l\), the speedups versus the “retention” parameter \(\delta l\) for several numbers of processors and the parallel efficiency for several values of \(\delta l\) is presented for the \textit{PNEPCG-ChG-variant} and the \textit{PNEPBICG-STAB} method respectively, with \(n=10000\), \(m=101\), \(r=2\) and \(\ell = 3\). As it is verified by our experimental results, the parallel behavior of the \textit{PNEPCG-ChG-variant} is better than the \textit{PNEPBICG-STAB} method for large values of the “retention” parameter \(\delta l\), as it was expected, because of the non-blocking communications that was adopted, [9]. Additionally, in the \textit{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 \(\delta l\), the communication cost is responsible for such performance.

Since there is increased communication cost for small values of the “retention parameter” \(\delta l\) and further taking into account the time consumption of the application it is recommended that large values of the “retention parameter” \(\delta l\) should be chosen.

Based on the derived theoretical estimates, in order to approach the optimum value of speedup as \(\delta l\) increases, the utilized number of processors should be increased. When coarse grain parallelism is adopted, i.e. when \(\delta l\) is increased, a reduction in iterations was achieved along with an increase in the overall speedup. This is because by increasing \(\delta l\), the approximate inverse tends to become the exact inverse (class I), (8).

<table>
<thead>
<tr>
<th>“Retention” parameter</th>
<th>Speedups</th>
<th>Number of processors</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\delta l) = 1</td>
<td>1.3058</td>
<td>0.5849</td>
<td>0.5700</td>
</tr>
<tr>
<td>(\delta l) = 2</td>
<td>1.321</td>
<td>0.7682</td>
<td>0.7338</td>
</tr>
<tr>
<td>(\delta l) = m</td>
<td>1.9153</td>
<td>3.0099</td>
<td>4.4234</td>
</tr>
<tr>
<td>(\delta l) = 2m</td>
<td>1.9652</td>
<td>3.8352</td>
<td>5.8905</td>
</tr>
<tr>
<td>(\delta l) = 4m</td>
<td>1.9786</td>
<td>3.9559</td>
<td>7.3386</td>
</tr>
<tr>
<td>(\delta l) = 6m</td>
<td>1.9999</td>
<td>3.9690</td>
<td>7.5111</td>
</tr>
</tbody>
</table>

Table 1. Speedups and processors allocated of the \textit{PNEPCG-ChG-variant} method, for several values of \(\delta l\), with \(n=10000\), \(m=101\) and \(r=2\).
Figure 1. Speedups and processors allocated of the PNEPCG-ChG-variant method for several values of $\delta l$, with $n=10000$ and $m=101$.

Figure 2. Speedups versus the “retention” parameter $\delta l$ 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 $\delta l$, with $n=10000$ and $m=101$. 
Figure 4. Performance evaluation measurements of the PNEPCG-ChG-variant method, with $n=10000$ and $m=101$.

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

<table>
<thead>
<tr>
<th>“Retention” parameter</th>
<th>Speedups</th>
<th>Number of processors</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta l=1$</td>
<td>1.2912</td>
<td>1.1752</td>
<td>1.0633</td>
</tr>
<tr>
<td>$\delta l=2$</td>
<td>1.3387</td>
<td>1.1844</td>
<td>1.1069</td>
</tr>
<tr>
<td>$\delta l=m$</td>
<td>1.9360</td>
<td>3.1828</td>
<td>5.3622</td>
</tr>
<tr>
<td>$\delta l=2m$</td>
<td>1.9349</td>
<td>3.3452</td>
<td>5.9481</td>
</tr>
<tr>
<td>$\delta l=4m$</td>
<td>1.9540</td>
<td>3.6554</td>
<td>6.4035</td>
</tr>
<tr>
<td>$\delta l=6m$</td>
<td>1.9636</td>
<td>3.7162</td>
<td>7.1375</td>
</tr>
</tbody>
</table>

Figure 5. Speedups and processors allocated of the PNEPBICG-STAB method for several values of $\delta l$, 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 \( \delta l \), 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**


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


