Parallel Exact and Approximate Arrow-Type Inverses on Symmetric Multiprocessor Systems

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. In this paper we present new parallel inverse arrow-type matrix algorithms based on the concept of sparse factorization procedures, for computing explicitly exact and approximate inverses, on symmetric multiprocessor systems. The parallel implementation of the new inversion algorithms is discussed and numerical results are presented, using the simulation tool of Multi-Pascal.

1 Introduction

Sparse matrix computations, which have inherent parallelism, are of central importance, because of the applicability to real-life problems and are the most time-consuming part in computational science and engineering computations. Hence research efforts were focused on the production of efficient parallel computational methods and related software suitable for multiprocessor systems, [1, 2, 5, 11, 12, 14].

Until recently, direct methods have been effectively used, but the increase of size, even with the use of modern computer systems, has become a barrier to such methods, [2, 3]. Additionally, the solution of sparse linear systems, because of its applicability to real-life problems, is obtained by iterative methods, which are in competitive demand after the emergence of Krylov subspace methods, [5, 11, 14].

An important achievement over the last decades is the appearance and use of Explicit Preconditioned Methods, [4], for solving sparse linear systems, and the preconditioned form of a linear system $Au = s$ is $MAu = Ms$, where M is preconditioner, [1, 2, 5, 11, 14]. 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, [2, 5, 11, 12, 14].

Hence, the derivation of parallel methods was the main objective for which several families of parallel inverses of an arrow-type matrix, are proposed. The main motive for the derivation of the parallel exact and approximate inverse matrix algorithms is that they result in parallel direct methods and in parallel iterative methods in conjunction with parallel preconditioned conjugate gradient-type schemes respectively, for solving arrow-type linear systems on multiprocessor systems.

2 Parallel Exact and Approximate Inverses

Let us consider the arrow - type linear system, i.e.,

$$Au = s, \quad (1)$$

where A is a sparse arrow - type $(n \times n)$ matrix of the following form:

$$A \equiv \left[\begin{array}{ccc} b_1 & c_1 & p_1 \\ a_2 & & \\ & 0 & \\ & & p_{n-2} \\ & & c_{n-1} \\ v_1 & & a_n \\ & v_{n-2} & b_n \end{array} \right]. \quad (2)$$

Arrow-type matrices occur in practice for example, in the course of the Lanczos method for solving the eigenvalue problem for large sparse matrices, in the eigenstructure problems of arrowhead matrices which arise from applications in molecular physics and in the application of the finite element or finite difference method over a region by removing part of the region. This class of arrow-type linear systems captures the class of linear systems, which can be considered as a special tridiagonal linear systems, which occur when solving certain constant-coefficient elliptic partial differential equations by the Fourier method, or using finite difference methods to solve linear constant-coefficient boundary value problems, [6, 7, 8].

Let us now assume the factorization of the coefficient matrix A , i.e.

$$A = LU, \quad (3)$$

where L and U are sparse strictly lower and upper triangular matrices of the same profile as the matrix A , [6, 7, 8]. The elements of the L and U decomposition factors were computed by the so-called **Arrow** - **type Approximate LU-type Factorization** algorithm (henceforth **ATALUFA** algorithm), [6]. The memory requirements of the **ATALUFA** algorithm are $O(5n)$ words and the computational work is $O(6n)$ multiplicative operations, [6].

Let $M = (\mu_{i,j})$, $i, j \in [1, n]$, be the exact inverse of the coefficient matrix A . The elements of $M = (\mu_{i,j})$ can be computed by solving recursively the following systems, [6]:

$$ML = U^{-1} \quad \text{and} \quad UM = L^{-1}. \quad (4)$$

The **Exact Inverse Arrow - type Matrix** algorithm (henceforth called the **EIATM** algorithm) for computing the elements of the exact inverse, has been presented in [6]. Similarly, by retaining δl elements next to the main diagonal, the elements of the approximate inverse $M^{\delta l} = (\mu_{i,j})$, $i \in [1, n]$, $j \in [\max(1, i - \delta l + 1), \min(n, i + \delta l - 1)]$, were computed by the **Banded**

Approximate Inverse Arrow - type Matrix algorithm (henceforth called the **BAIATM** algorithm), [6].

For the parallelization of the **EIATM** and the **BAIATM** algorithms on symmetric multiprocessor systems, the **ATALUFA** algorithm was used as a “front-end” computational procedure. The elements of the matrix M were computed in a sequence, as shown in Eq. 5 (for $n = 8$ and $\delta l = 3$ without loss of generality), because of the dependence of the elements of the inverse during the construction. The values of the parentheses at the superscript of elements (e.g. $\mu_{i,j}^{(k)}$), indicate that the element $\mu_{i,j}$ was computed at the k -th step of the algorithm, while the elements with the same superscript (i.e. (k)) were computed concurrently. The pattern for the sequence of computations was considered as an anti-diagonal motion, starting from the element $\mu_{n,n}$ down to $\mu_{1,1}$, where the elements on an anti-diagonal were computed in parallel.

$$M = \begin{bmatrix} \mu_{1,1}^{(15)} & \mu_{1,2}^{(14)} & \mu_{1,3}^{(13)} & \mu_{1,4}^{(12)} & \mu_{1,5}^{(11)} & \mu_{1,6}^{(10)} & \mu_{1,7}^{(9)} & \mu_{1,8}^{(8)} \\ \mu_{2,1}^{(14)} & \mu_{2,2}^{(13)} & \mu_{2,3}^{(12)} & \mu_{2,4}^{(11)} & \mu_{2,5}^{(10)} & \mu_{2,6}^{(9)} & \mu_{2,7}^{(8)} & \mu_{2,8}^{(7)} \\ \mu_{3,1}^{(13)} & \mu_{3,2}^{(12)} & \mu_{3,3}^{(11)} & \mu_{3,4}^{(10)} & \mu_{3,5}^{(9)} & \mu_{3,6}^{(8)} & \mu_{3,7}^{(7)} & \mu_{3,8}^{(6)} \\ \mu_{4,1}^{(12)} & \mu_{4,2}^{(11)} & \mu_{4,3}^{(10)} & \mu_{4,4}^{(9)} & \mu_{4,5}^{(8)} & \mu_{4,6}^{(7)} & \mu_{4,7}^{(6)} & \mu_{4,8}^{(5)} \\ \mu_{5,1}^{(11)} & \mu_{5,2}^{(10)} & \mu_{5,3}^{(9)} & \mu_{5,4}^{(8)} & \mu_{5,5}^{(7)} & \mu_{5,6}^{(6)} & \mu_{5,7}^{(5)} & \mu_{5,8}^{(4)} \\ \mu_{6,1}^{(10)} & \mu_{6,2}^{(9)} & \mu_{6,3}^{(8)} & \mu_{6,4}^{(7)} & \mu_{6,5}^{(6)} & \mu_{6,6}^{(5)} & \mu_{6,7}^{(4)} & \mu_{6,8}^{(3)} \\ \mu_{7,1}^{(9)} & \mu_{7,2}^{(8)} & \mu_{7,3}^{(7)} & \mu_{7,4}^{(6)} & \mu_{7,5}^{(5)} & \mu_{7,6}^{(4)} & \mu_{7,7}^{(3)} & \mu_{7,8}^{(2)} \\ \mu_{8,1}^{(8)} & \mu_{8,2}^{(7)} & \mu_{8,3}^{(6)} & \mu_{8,4}^{(5)} & \mu_{8,5}^{(4)} & \mu_{8,6}^{(3)} & \mu_{8,7}^{(2)} & \mu_{8,8}^{(1)} \end{bmatrix}. \quad (5)$$

If we consider that the command **forall** is responsible for process creation and execution, then the algorithm for the **Parallel implementation of the EIATM algorithm** (henceforth called the **PEIATM** algorithm) is:

```

For  $k = 1$  to  $n$ 
  forall  $l = 1$  to  $k$ 
    call  $inverse(n - l + 1, n - k + l)$ 
for  $k = n - 1$  downto  $1$ 
  forall  $l = 1$  to  $k$ 
    call  $inverse(l, k - l + 1)$ 

```

while the algorithm for the **Parallel implementation of the BAIATM algorithm** (henceforth called the **PBAIATM** algorithm) is:

```

For  $k = 1$  to  $\delta l$ 
  forall  $l = 1$  to  $k$ 
    call  $inverse(n - l + 1, n - k + l)$ 
 $m = 2$ 

```

```

for  $k = (\delta l + 1)$  to  $n$ 
    forall  $l = m$  to  $(k - m + 1)$ 
        call  $inverse(n - l + 1, n - k + l)$ 
    if  $(k - \delta l) \bmod 2 = 0$  then
         $m = m + 1$ 
 $m = m - 1$ 
for  $k = (n - 1)$  downto  $(\delta l + 1)$ 
    forall  $l = m$  to  $k - m + 1$ 
        call  $inverse(l, k - l + 1)$ 
    if  $(k - \delta l) \bmod 2 = 1$  then
         $m = m - 1$ 
for  $k = \delta l$  downto  $1$ 
    forall  $l = 1$  to  $k$ 
        call  $inverse(l, k - l + 1)$ 

```

where the function $inverse(i, j)$ computes the element $\mu_{i,j}$ of the exact inverse, and can be described as follows, [6]:

```

function  $inverse(i, j)$ 
if  $(i \geq j)$  then
    if  $(i = n)$  then
        if  $(j = n)$  then
             $\mu_{n,n} = 1/w_n$ 
        else
            if  $(j = n - 1)$  then
                 $\mu_{n,n-1} = (-d_j \cdot \mu_{n,n}) / w_j$ 
            else
                 $\mu_{n,j} = (-d_j \cdot \mu_{n,j+1} - e_j \cdot \mu_{n,n}) / w_j$ 
    else
        if  $(i = j)$  then
            if  $(j = n - 1)$  then
                 $\mu_{n-1,n-1} = (1 - d_{n-1} \cdot \mu_{i,n}) / w_j$ 
            else
                 $\mu_{i,j} = (1 - d_j \cdot \mu_{i,j+1} - e_j \cdot \mu_{i,n}) / w_j$ 
        else
             $\mu_{i,j} = (-d_j \cdot \mu_{i,j+1} - e_j \cdot \mu_{i,n}) / w_j$ 
else
    if  $(j = n)$  then
        if  $(i = n - 1)$  then
             $\mu_{n-1,n} = -g_{n-1} \cdot \mu_{n,n}$ 
        else
             $\mu_{i,n} = -g_i \cdot \mu_{i+1,n} - h_i \cdot \mu_{n,n}$ 
    else
         $\mu_{i,j} = -g_i \cdot \mu_{i+1,j} - h_i \cdot \mu_{n,j}$ 

```

It should be mentioned that the parallel construction of exact inverses on distributed systems has been studied and implemented in [9], and is under further investigation.

3 Numerical Results

For the parallel implementation of the **PEIATM** and **PBAIATM** algorithm described above, the simulation tool of *Multi-Pascal* has been utilized, where a time unit of the simulated time is approximately equivalent to one microsecond of the real execution time on a general purpose multiprocessor, [13]. The architecture platform is that of a shared memory system consisted of 512 processors.

The relative speedups and efficiency, for several values of the order n and number of processors for the **PEIATM** algorithm are presented in Table 1. In Fig. 1 and 2 the relative speedups versus the order n for several number of processors and the relative speedups versus the number of processors for several values of the order n , are presented respectively for the **PEIATM** algorithm.

Table 1. Speedups and efficiency for the **PEIATM** algorithm, for various values of the order n and number of processors

n	no_proc=2		no_proc=4		no_proc=8		no_proc=13	
	Speedup	Effi/ncy	Speedup	Effi/ncy	Speedup	Effi/ncy	Speedup	Effi/ncy
15	1.825	0.913	2.867	0.717	3.505	0.438	3.645	0.280
20	1.856	0.928	2.975	0.744	3.898	0.487	4.155	0.320
40	1.894	0.947	3.083	0.771	4.931	0.616	5.361	0.412
60	1.902	0.951	3.102	0.776	5.494	0.687	5.957	0.458
80	1.905	0.953	3.106	0.777	5.870	0.734	6.314	0.486

Fig. 1. Speedups versus the order n for the **PEIATM** algorithm for several number of processors

The relative speedups and efficiency for several values of the order n , the retention parameter δl and number of processors for the **PBAIATM** algorithm are presented in Table 2. In Fig. 3 and 4 the relative speedups versus the retention parameter δl for several number of processors with $n = 100$, and the relative speedups versus the order n for several number of processors with $\delta l = n/2$, are presented respectively for the **PBAIATM** algorithm.

It should be noted that the restrictions imposed by the developing environment used, did not allow us to fully explore cases for $n > 80$ for the exact inverse

Fig. 2. Speedups versus the number of processors for the **PEIATM** algorithm for several values of the order n

Table 2. Speedups and efficiency for the **PBAIATM** algorithm, for various values of the order n , δl and number of processors

n	δl	no_proc=2		no_proc=4		no_proc=5	
		Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency
20	2	1.104	0.552				
	4	1.477	0.739	1.760	0.440	1.760	0.352
	8	1.637	0.819	2.201	0.550	2.307	0.461
	$n/2$	1.669	0.835	2.376	0.594	2.421	0.484
40	2	1.107	0.554				
	4	1.484	0.742	1.784	0.446	1.784	0.357
	8	1.650	0.825	2.258	0.565	2.370	0.474
	$n/2$	1.793	0.897	2.809	0.702	2.943	0.589
60	2	1.108	0.554				
	4	1.486	0.743	1.792	0.448	1.792	0.358
	8	1.654	0.827	2.275	0.569	2.389	0.478
	$n/2$	1.783	0.892	2.953	0.738	3.151	0.630
80	2	1.108	0.554				
	4	1.487	0.744	1.796	0.449	1.795	0.359
	8	1.656	0.828	2.283	0.571	2.398	0.480
	$n/2$	1.768	0.884	3.118	0.780	3.262	0.652
100	2	1.108	0.554				
	4	1.487	0.744	1.798	0.450	1.798	0.360
	8	1.657	0.829	2.288	0.572	2.403	0.481
	$n/2$	1.762	0.881	3.218	0.805	3.351	0.670

algorithm and for $n > 100$ for the approximate inverse algorithm. Additionally, the number of processors $no_proc = 13$ for the exact inverse and $no_proc = 5$ for the approximate inverse, was the maximum number of processors allowed by the simulation environment in each case (for $\delta l = 2$ the maximum number was $no_proc = 2$).

Fig. 3. Speedups versus the retention parameter δl for the **PBAIATM** algorithm for several number of processors with $n = 100$

Fig. 4. Speedups versus the order n for the **PBAIATM** algorithm for several number of processors with $\delta l = n/2$

We observe that the relative speedups and efficiency are increasing as the order n increases for various values of the number of processors. Although the relative speedups increase when adding more processors on the system, the efficiency is decreasing, but remains high for large values of n .

Numerical results for such arrow-type linear systems have been presented in [6, 7, 8] for solving boundary value problems and in [10] for solving problems considered in reliability engineering and in particular performability evaluation of multitasking and multiprocessor systems.

4 Conclusion

The design of parallel explicit exact and approximate inverses results in efficient parallel direct and iterative methods for solving arrow-type linear systems on multiprocessor systems. Despite the restrictions of the simulated environment used as the experimental platform, the experimental results obtained tend to the upper theoretical bounds.

Finally, further parallel algorithmic techniques will be investigated in order to improve the speedups and efficiency of the parallel construction of explicit exact and approximate inverses, on symmetric multiprocessor systems.

References

- [1] Akl, S.G.: *Parallel Computation: Models and Methods*. Prentice Hall (1997)
- [2] Dongarra, J.J., Duff, I., Sorensen, D., van der Vorst, H.A.: *Numerical Linear Algebra for High-Performance Computers*. SIAM (1998)
- [3] Duff, I.: The impact of high performance computing in the solution of linear systems: trends and problems. *J. Comp. Applied Math.* **123** (2000) 515-530
- [4] Evans, D.J.: *Preconditioning Methods: Theory and Applications*. Gordon and Breach Science Publishers (1983)
- [5] Gravvanis, G.A.: Explicit Approximate Inverse Preconditioning Techniques. *Archives of Computational Methods in Engineering* **9(4)** (2002) 371-402
- [6] Gravvanis, G.A.: Explicit isomorphic iterative methods for solving arrow-type linear systems. *Inter. J. Comp. Math.* **74(2)** (2000) 195-206
- [7] Gravvanis, G.A.: Solving parabolic and nonlinear 1D problems with periodic boundary conditions. CD-ROM Proceedings of the European Congress on Computational Methods in Applied Sciences and Engineering 2000
- [8] Gravvanis, G.A.: Parallel preconditioned algorithms for solving special tridiagonal systems. *Proceedings of Dynamical Systems and Applications* **III**, (1999) 241-248 Dynamic Publishers
- [9] Gravvanis, G.A.: Parallel matrix techniques. In: K. Papailiou, D. Tsahalis, J. Periaux, C. Hirsch, M. Pandolfi eds. *Computational Fluid Dynamics I* (1998) 472-477 Wiley
- [10] Gravvanis, G.A., Platis, A.N., Giannoutakis, K.M., Violentis, J.B., Lipitakis, E.A.: Performability evaluation of multitasking and multiprocessor systems by explicit approximate inverses. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, H.R. Arabnia and Youngsong Mun eds. **III** (2003) 1324-1331 CSREA Press
- [11] Grote, M.J., Huckle, T.: Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.* **18** (1977) 838-853
- [12] Grote, M.J., Simon, H.D.: Parallel preconditioning and approximate inverses on the connection machine. In: R.F. Sincovec, D.E. Keyes, L.R. Petzold and D.A. Reed, eds. *Parallel Processing for Scientific Computing* **2** (1993) 519-523 SIAM
- [13] Lester, B.P.: *The Art of Parallel Programming*. Prentice-Hall Int. Inc (1993)
- [14] Saad, Y., van der Vorst, H.A.: Iterative solution of linear systems in the 20th century. *J. Comp. Applied Math.* **123** (2000) 1-33