International Journal of Pure and Applied Mathematics

Volume 22 No. 2 2005, 267-279

A PARALLEL COMPUTATIONAL KERNEL FOR SPARSE NONSYMMETRIC EIGENVALUE PROBLEMS ON MULTICOMPUTERS

M.R. Guarracino[1], F. Perla[2 §], P. Zanetti[3]

[1]Institute for High Performance Computing and Networking – ICAR-CNR
Italian National Research Council
Via P. Castellino, Naples, 111 - 80131, ITALY
e-mail: mario.guarracino@icar.cnr.it

[2,3] University of Naples Parthenope
Via Medina, Naples, 40 - 80133, ITALY
[2]e-mail: francesca.perla@uniparthenope.it
[3]e-mail: paolo.zanetti@uniparthenope.it

Abstract: The aim of this paper is to show an effective reorganization of the nonsymmetric block lanczos algorithm efficient, portable and scalable for multiple instructions multiple data (MIMD) distributed memory message passing architectures. Basic operations implemented here are matrix-matrix multiplications, eventually with a transposed and a sparse factor, LU factorisation and triangular systems solution. Since the communication overhead of the algorithm inhibits an efficient parallel implementation, we propose a reorganization of the block algorithm which reduces the total amount of communication involved in linear algebra operations. Then, we develop an efficient parallelization of the matrix-matrix multiplication when one of the factor is sparse. Some other linear algebra operations are performed using *ScaLAPACK* library. The parallel eigensolver has been tested on a cluster of PCs. All reported results show the proposed algorithm is efficient and scalable on the target architectures for problems of adequate dimension, and it can be the computational kernel of a robust

Received: June 28, 2005

§Correspondence author

software for large sparse eigenvalue problems.

AMS Subject Classification: 65F15, 65F50, 68W10
Key Words: nonsymmetric eigensolver, parallel block Lanczos algorithm, distributed memory architectures

1. Introduction

Driven by technological advances, scientific computing has emerged as powerful tool to solve increasingly more complex problems. For this reason, a wide effort in developing efficient numerical software is under way, especially for parallel architectures. At present, little robust parallel software is available for large sparse eigenvalue problems, as the existing algorithms do not lead to an efficient parallel implementation and new algorithms have yet to be developed for the target architectures. Among them, there is the PARPACK package [16], which implements the implicitly restarted Arnoldi method, and represents the state of the art in the field.

It is worth noting a broad class of economic applications, involving large-scale macroeconometric models (e.g. Leontiev and Von Neumann models) and dynamical systems, which consists of problems involving a large sparse operator and requires the computation of few extreme eigenvalues [8], [9], [19]. For those problems, widely used algorithms are the *Lanczos algorithms* [15], which are based on *Krylov subspace methods* [10], [18]. The reasons reside in the demonstrated computational efficiency and the good convergence properties which can be achieved by these procedures [5].

In this work we consider the *nonsymmetric block Lanczos* algorithm (see for example [2]) for real matrices. With respect to the single vector algorithm, it is more robust and efficient for matrices with multiple or closely clustered eigenvalues; moreover it exploits a large grain parallelism. Since the communication overhead of the algorithm inhibits an efficient parallel implementation, we propose a reorganization of the block algorithm which reduces the total amount of communication involved in linear algebra operations. Its implementation has lead to parallel numerical software which is efficient, portable and scalable for MIMD *distributed memory message passing architectures*.

The rest of this work is organized as follows: Section 2 describes the basic building blocks. Section 3 shows how we reorganize the algorithm in order to reduce data communication and it deals with its parallel implementation on MIMD distributed memory message passing architectures, assuming a 2-D mesh connection topology and a scatter decomposition of matrices. Finally,

Section 4 discusses results of a performance evaluation carried out on a cluster of PCs.

2. Nonsymmetric Block Lanczos Algorithm

The Lanczos algorithm for computing eigenvalues of a nonsymmetric matrix $A \in \mathbb{R}^{m\times m}$ is an oblique projection method that allows to obtain a representation of the operator in the Krylov subspace K and orthogonal to another subspace L. In subspace K the representation of a nonsymmetric matrix is always tridiagonal. Assuming $m = rs$, the *block Lanczos algorithm*, if no breakdown occurs, generates a nonsymmetric block tridiagonal matrix T having the same eigenvalues as A:

$$T = \begin{pmatrix} M_1 & B_1 & & & \\ D_1 & M_2 & B_2 & & \\ & \ddots & \ddots & \ddots & \\ & & D_{r-2} & M_{r-1} & B_{r-1} \\ & & & D_{r-1} & M_r \end{pmatrix},$$

where $M_j \in \mathbb{R}^{s\times s}$, $B_j \in \mathbb{R}^{s\times s}$ and $D_j \in \mathbb{R}^{s\times s}$ are lower and upper triangular, respectively. T is such that

$$W^T A V = T,$$

where

$$V = [V_1\ V_2\ \ldots\ V_r], \quad V_i \in \mathbb{R}^{m\times s}, \quad W = [W_1\ W_2\ \ldots\ W_r], \quad W_i \in \mathbb{R}^{m\times s}$$

are bi-orthogonal matrices, and their columns are the vectors spanning the two subspaces K and L. A direct way to evaluate M_j, B_j, D_j, V_j and W_j is described in Algorithm 1.

At step j, Algorithm 1 produces a nonsymmetric block tridiagonal matrix T_j of order $(j+1)\times s$ satisfying the equivalence

$$[W_1\ W_2\ \ldots\ W_{j+1}]^T A [V_1\ V_2\ \ldots\ V_{j+1}] = T_j\,,$$

where $[W_1\ W_2\ \ldots\ W_{j+1}]^T[V_1\ V_2\ \ldots\ V_{j+1}] = I$. In fact, when j grows T_j extreme eigenvalues, called *Ritz values* of A, are increasingly better approximation of A extreme eigenvalues.

Block algorithms allow approximations of eigenvalues with multiplicity greater than one, while in single vector algorithms difficulties can be expected since the

Choose $V_1, W_1 \in \mathbb{R}^{m \times s}$ such that $W_1^T V_1 = I_s$,

Let $B_0 = D_0 \equiv 0 \in \mathbb{R}^{s \times s}$ e $V_0, W_0 \in \mathbb{R}^{m \times s}$ such that $W_0 = V_0 \equiv 0$,

$M_1 = W_1^T A V_1$

$j = 1$

Repeat:

$\tilde{V}_{j+1} = AV_j - V_{j-1}B_{j-1} - V_j M_j$

$\tilde{W}_{j+1} = A^T W_j - W_{j-1} D_{j-1}^T - W_j M_j^T$

$\tilde{W}_{j+1}^T \tilde{V}_{j+1} = B_j D_j$ (LU factorisation)

Solve $V_{j+1} D_j = \tilde{V}_{j+1}$

Solve $W_{j+1} B_j^T = \tilde{W}_{j+1}$

$M_{j+1} = W_{j+1}^T A V_{j+1}$

$j = j + 1$

until: $j = r$

Algorithm 1: Nonsymmetric block Lanczos algorithm (I version)

projected operator, in finite precision, is unreduced tridiagonal, which implies it cannot have multiple eigenvalues [10].

In this paper we do not consider the application of some reorthogonalization techniques of the Lanczos vectors, since it goes beyond the aim of this work.

3. Parallel Implementation

In a previous work [12] we proposed a parallel implementation of the symmetric block Lanczos algorithm for MIMD distributed memory architectures. The symmetric algorithm is build upon the same computational kernels of Algorithm 1 – i.e. matrix-matrix products and factorisations. In [13] we showed that a direct parallelization of the symmetric algorithm has efficiency values that deteriorates when sparsity decreases. This loss of efficiency is due to the amount of communication, with respect to computational complexity, required in the

Choose $V_1, W_1 \in \mathbb{R}^{m\times s}$ such that $W_1^T V_1 = I_s$,

Let $B_0 = D_0 \equiv 0 \in \mathbb{R}^{s\times s}$ e $V_0, W_0 \in \mathbb{R}^{m\times s}$ such that $W_0 = V_0 \equiv 0$,

$M_1^T = V_1^T A^T W_1$

$j = 1$

Repeat:

$\tilde{V}_{j+1}^T = V_j^T A^T - B_{j-1}^T V_{j-1}^T - M_j^T V_j^T$

$\tilde{W}_{j+1}^T = W_j^T A - D_{j-1} W_{j-1}^T - M_j W_j^T$

$\tilde{W}_{j+1}^T \tilde{V}_{j+1} = B_j D_j$ (LU factorisation)

Solve $V_{j+1}^T D_j^T = \tilde{V}_{j+1}^T$

Solve $W_{j+1}^T B_j = \tilde{W}_{j+1}^T$

$M_{j+1}^T = V_{j+1}^T A^T W_{j+1}$

$j = j + 1$

until: $j = r$

Algorithm 2: Nonsymmetric block Lanczos algorithm (II version)

matrix-matrix multiplication, when the first factor is A sparse, since, following *ScaLAPACK* implementation choices for matrix-matrix operations, the first factor is involved in global communications, while the second only in one-to-one communications.

Then, to avoid this phenomena, we reorganized the symmetric block algorithm in such a way the sparse A is the second factor in all matrix-matrix products, so that it is not involved in global communications. This was achieved formally substituting each matrix appearing in the algorithm by its transpose. Applying the same idea to Algorithm 1, we obtain the version of the nonsymmetric block Lanczos algorithm, depicted in Algorithm 2.

Substituting the sparse matrix-matrix operations AV_j and $A^T W_j$ by the evaluation of the products $V_j^T A^T$ and $W_j^T A$, we obtain that V_j^T and W_j^T are involved in global communications instead of A and therefore, there was a reduction in terms of communication complexity, execution times, and a gain in *speed-up* and *efficiency*, as we showed in [13] for the symmetric version.

Algorithm 2 has the same numerical properties of Algorithm 1, since the use of transposed factors does not alter its behaviour with respect to round-off errors, as described in [14].

To obtain good performances on different MIMD distributed memory architectures, in particular on massively parallel machines, we have to consider a suitable connection topology, and, consequently, an appropriate data distribution of the matrices among the nodes.

It is well known that *2D mapping* of matrices onto a logical grid of nodes leads to efficient, scalable, and flexible routines. For this reason we assume the target architecture to consist of p nodes, logically configured as a $P \times Q$ grid, indexed by an ordered pair (I, J), where $0 \leq I < P$ and $0 \leq J < Q$. Each node is equipped with CPU and local memory. The nodes are connected by some communication network that allows broadcasting of messages within rows and columns, in addition to point-to-point communication. In this environment it is natural to develop a parallel algorithm in terms of loosely synchronous processes performing the same task on different nodes.

Since in Algorithm 2 basic operations are level *3 BLAS*, see [6], and the considered connection topology is 2-D mesh, we choose the *block scatter decomposition*, see [3]. This mapping is obtained by partitioning a matrix $B \in \mathbb{R}^{n \times n}$ like

$$B = \begin{pmatrix} B_{1,1} & \dots & B_{1,m} \\ B_{2,1} & \dots & B_{2,m} \\ \vdots & \vdots & \vdots \\ B_{m,1} & \dots & B_{m,m} \end{pmatrix},$$

where each subblock is at most $nb \times nb$ and at least 1×1, and $n \leq nb \times m$. These blocks are mapped to nodes by assigning $B_{i,j}$ to node $((i-1) \mathrm{mod} P, (j-1) \mathrm{mod} Q)$.

Block scatter decomposition is used for all the matrices involved in Algorithm 2. For the storage scheme of A sparse we use a data structure, per process, usually referenced as *CSR-compress sparse row*, see [3], consisting of three arrays, respectively containing:

1. the non-zero entries of A row parts in the subblocks that are assigned to the process;
2. the column indices in A of each element in the first array;
3. pointers to the position in the first array of the first non-zero entry of each row part.

Therefore, the global sparse matrix storage is a block scattered *CSR*. This storage scheme allows a faster memory access to get data, an easier localization

of a whole row and a decrease in global communication.

A popular choice for sparse matrix distribution is 1D partitioning, since it is more natural to sparse matrices, due to the existence of well known graph partitioning algorithms, and it is much easier to implement. Nevertheless, a 2D layout strikes a good balance among locality (by blocking), load balance (by cyclic mapping), and lower communication volume (by 2D mapping), see [17].

In Algorithm 2, linear algebra operations are essentially matrix-matrix multiplications, eventually with a transposed factor or a sparse factor, LU factorisation and triangular systems solution.

Our implementation uses standard message passing libraries, i.e. *BLACS* [7] and *MPI* [11] and de facto standard numerical linear algebra software, *PBLAS* and *ScaLAPACK* [3], [4], obtaining a software as portable as *PARPA- CK*. Since all matrices involved in the algorithm are distributed among processing nodes, memory is used efficiently, since no replication of data occurs. On single node, using level 3 *BLAS* and *LAPACK* [1] routines enables both its efficient use and a favorable computation/communication ratio.

The main routine of *PBLAS* used in the implementation of Algorithm 2 is *PSGEMM* to evaluate matrix-matrix multiplications with dense factors. The current model implementation of the *PBLAS* assumes the matrix operands to be distributed according to the block scatter decomposition.

Routines for matrices factorisation are not included in *PBLAS*, but they are covered by *ScaLAPACK*. The evaluation of the LU factorisation $B_j D_j$ of $\tilde{W}_j^T \tilde{V}_j$, at each iteration, is then performed by using the routine *PSGETRF*. The solution of the triangular systems evaluated by *PSTRSM*.

We developed *PSGEMMS* routine to calculate matrix-sparse matrix product $C = B * \mathrm{op}(A)$, where $\mathrm{op}(A)$ is either sparse matrix A or its transpose, and all matrices are distributed on the the 2D mesh of processors. *PSGEMMS* has been obtained from *ScaLAPACK* routine *PSGEMM* substituting, in each node program, *BLAS3 SGEMM* matrix-matrix product, with an implementation of its counterpart matrix-sparse matrix product, that has been named *SGEMMS*, in which the sparse factor has a block *CSR* storage format. Since sparse factor is not involved in communication, the advantage is the overhead does not depend on sparsity. On the other hand, performance depends on the distribution of the nonzero entries in the sparse matrix; if those elements are uniformly distributed, each processor will execute a comparable number of operations, thus balancing the workload.

Finally, the operation count of Algorithm 2 is exactly the same as Algorithm 1. Thanks to computational characteristics of linear algebra kernels, the parallel implementation of Algorithm 2 has a computational cost on p nodes

that is exactly $1/p$ of the sequential one, and a communication complexity of one order magnitude less that computational one.

4. Numerical Experiments

The proposed parallel Nonsymmetric Block Lanczos algorithm is implemented in *Fortran 77*; *PSGEMMS* is partially implemented in *C* language. All the tests reported refer to a cluster of 8 AMD Athlon 1.2GHz processors with 128MB DDR RAM connected by a 100 Megabit/s FastEthernet switch, operated by the University of Naples Parthenope; clustering middleware is Scyld 27BZ8, which includes *egcs-2.91.66*, *MPICH 1.2.1*, *BLACS 1.1* and *SCALAPACK 1.6*.

The following tests have been performed, on randomly generated matrices of order $m = 8192, 16384, 32768$, with a percentage of non-zero entries $nzp = .5\%, 1\%$, and four values for the number of Lanczos vectors $s = 32, 64, 128, 256$. The performance of the algorithm is evaluated using $p = 1, 2, 4, 8$ nodes logically configured as a grid of 1×1, 1×2, 2×2 and 2×4 nodes, respectively. In the tables 1-6 we report the minimal execution times, $T_p(m, s, nzp)$, in seconds, with respect to nb, of one complete step on p nodes for a fixed size problem. All the execution times have been obtained using the *Fortran 77* routine *SECOND()*. Further, to evaluate the effect of parallelization, we use the classical parameter *efficiency* (E_p):

$$E_p = T_1(m, s, nzp)/pT_p(m, s, nzp).$$

The efficiency values are reported in brackets in Tables 1-6.

	p			
s	1	2	4	8
32	2.45	1.93 (.63)	1.70 (.36)	1.50 (.20)
64	5.86	4.00 (.73)	3.17 (.46)	2.16 (.34)
128	12.98	9.60 (.67)	6.70 (.48)	4.05 (.40)
256	35.60	22.00 (.80)	15.30 (.58)	9.70 (.46)

Table 1: $T_p(m, s, nzp)$ (E_p), $m = 8192$, $nzp = .5\%$

	p			
s	1	2	4	8
32	3.37	2.53 (.67)	2.17 (.38)	2.02 (.20)
64	10.77	6.80 (.79)	4.22 (.63)	2.74 (.49)
128	22.60	14.50 (.78)	8.80 (.64)	4.66 (.61)
256	49.76	31.60 (.79)	20.00(.62)	10.05 (.59)

Table 2: $T_p(m, s, nzp)$ (E_p), $m = 8192$, $nzp = 1\%$

	p			
s	1	2	4	8
32	10.00	7.00 (.71)	5.20 (.48)	4.60 (.27)
64	21.31	14.00 (.76)	9.10 (.59)	5.70 (.47)
128	56.98	29.20 (.97)	18.74 (.76)	10.54 (.67)
256	-	-	45.90 (-)	24.01 (.95)

Table 3: $T_p(m, s, nzp)$ (E_p), $m = 16384$, $nzp = .5\%$

	p			
s	1	2	4	8
32	19.47	12.00 (.81)	6.30 (.77)	5.5 (.44)
64	40.90	23.40 (.87)	14.2 (.70)	7.38 (.69)
128	86.05	48.60 (.88)	28.60 (.76)	15.50 (.69)
256	-	-	61.63 (-)	34.01 (.90)

Table 4: $T_p(m, s, nzp)$ (E_p), $m = 16384$, $nzp = 1\%$

	p			
s	1	2	4	8
32	38.54	24.50 (.78)	15.30 (.63)	12.26 (.39)
64	-	- (-)	29.5 (-)	18.18 (.81)
128	-	- (-)	67.00 (-)	33.64 (.99)
256	-	-	-	-

Table 5: $T_p(m, s, nzp)$ (E_p), $m = 32768$, $nzp = .5\%$

	p			
s	1	2	4	8
32	-	- (-)	25.40 (-)	13.30 (.95)
64	-	- (-)	49.70 (-)	24.01 (.98)
128	-	- (-)	- (-)	- (-)
256	-	-	-	-

Table 6: $T_p(m, s, nzp)$ (E_p), $m = 32768$, $nzp = 1\%$

Values of m smaller than 8192 have not been tested since the obtained performance would not be significant due to problem size.

It was not possible to test some problems (gaps in tables), due to memory shortage. In those cases the efficiency value in brackets is obtained considering the ratio between the two execution times in the same row. That means it is possible to solve problems on an increasing number of processors that can not be managed before.

In Table 1 and Table 2 low efficiency values E_p are reported for small problems on 8 processors; nevertheless those values rapidly increase for larger m, reaching a value of 0.69 (Table 3 and Table 4). Based on the trends in table, if it was possible to run the missing jobs, the efficiency values would reach approximately 0.70. The obtained values show that, for fixed m and nzp, efficiency grows when s increases, accordingly to the fact that *ScaLAPACK* routines gain better performances for sufficient dense problem granularity on each node. Further, the efficiency values grows, fixed m and s, when nzp increases: this is an expected result, since one-to-all communication does not involve sparse factor A.

Furthermore, shorter execution times on 8 nodes than on 1 node for a twofold value of m and a fixed value of $s \geq 64$ and of nzp, that is for a fourfold

problem (e.g. Table 1 and Table 3, rows 2-3-4). This means that, for the considered dimensions, it is possible to solve a problem on 8 nodes four times bigger than on 1 node, in a shorter time.

5. Summary

In present work we present a parallel software based on a variant of the nonsymmetric block Lanczos algorithm for the real, nonsymmetric eigenvalue problem. Results of a performance evaluation, in terms of wall clock times and efficiency, confirm the qualities of the proposed algorithm, showing performance increases as the size of the problem grows.

All results seem to show that the proposed algorithm is efficient and scalable on the target architectures for problems of adequate dimension, and it can be the computational kernel of a robust software for large sparse eigenvalue problems. Nevertheless, further attention is required to implement suitable techniques to enhance the numerical behaviour of the algorithm. Besides, the aim of future work will be to obtain an implementation that can be efficient even in computational environments, such as computational grids, in which network speed and latency variability and use of heterogeneous processors pose more stringent constraints on the nature of algorithms.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J.D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen, *LAPACK Users' Guide*, Second Edition, SIAM, Philapdelphia (1995).

[2] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: a Practical Guide*, SIAM, Philapdelphia (2000).

[3] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, R.V. Whaley, ScaLAPACK: A portable linear algebra library for distributed memory computers – design and performance, *Computer Physics Communications*, **97** (1996), 1-15.

[4] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, R.C. Whaley, A proposal for a set of parallel basic linear algebra subprograms, Technical Report, UT-CS-95-292 (1995).

[5] J.K. Cullum, R.A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, Birkhäuser (1985).

[6] J. Dongarra, J. Du Croz, S. Hammarling, I. Duff, A set of level 3 basic linear algebra subprograms, *ACM Trans. Math. Soft*, **16** (1990), 1-17.

[7] J. Dongarra, R.C. Whaley, A User's Guide to the BLACS v1.1, *Technical Report*, UT-CS-95-281 (1997).

[8] M. Garbely, M. Gilli, Qualitative decomposition of the eigenvalue problem in a dynamic system, *Journal of Economic Dynamics and Control*, **15** (1991), 539-548.

[9] M. Gilli, G. Pauletto, Sparse direct methods for model simulation, *Journal of Economics Dynamics and Control*, **21** (1997), 1093-1111.

[10] G.H. Golub, C.F. Van Loan, *Matrix Computations*, Second Edition, Johns Hopkins Univ. Press (1989).

[11] W. Gropp, E. Lusk, A. Skjellum, *Using MPI, Second Edition: Portable Parallel Programming with the Message Passing Interface*, The MIT Press (1999).

[12] M.R. Guarracino, F. Perla, A parallel block lanczos algorithm for distributed memory architectures, *Parallel Algorithms and Applications*, **4** (1994), 211-221.

[13] M.R. Guarracino, F. Perla, A parallel modified block Lanczos' algorithm for distributed memory architectures, In: *Proc. Euromicro Workshop on Parallel and Distributed Processing* IEEE Computer Society (1995), 424-431.

[14] M.R. Guarracino, F. Perla, A Sparse, Symmetric eigensolver for distributed memory architectures: Parallel implementation and numerical stability, *Technical Report*, CPS (1995).

[15] C. Lanczos, An iteration method for the solution of the eigenvalue problem of linear differential and integral operators, *J. Res. Nat. Bur. Stand.*, **45** (1950), 225-281.

[16] R.B. Lehoucq, D.C. Sorensen, C. Yang, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, SIAM, Phildelphia (1998).

[17] X. Li, J. Demmel, SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems, *ACM Transactions on Mathematical Software*, **29** (2003), 110-140.

[18] B.N. Parlett, *The Symmetric Eigenvalues Problem*, Prentice-Hall (1980).

[19] Y. Saad, *Numerical Methods for Large Eigenvalues Problems*, Manchester Univ. Press, Halsted Press (1992).