Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

A parallel Block Lanczos algorithm and its implementation for the evaluation of some eigenvalues of large sparse symmetric matrices on multicomputers

M. R. Guarracino, F. Perla, P. Zanetti

RT-ICAR-NA-2005-14 **Novembre 2005**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Napoli, Via P. Castellino 111, I-80131 Napoli, Tel: +39-0816139508, Fax: +39-0816139531, e-mail: napoli@icar.cnr.it, URL: *www.na.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

A parallel Block Lanczos algorithm and its implementation for the evaluation of some eigenvalues of large sparse symmetric matrices on multicomputers [1]

M. R. Guarracino[a] F. Perla[b] P. Zanetti[b]

Rapporto Tecnico N.: **RT-ICAR-NA-2005-14**

Data: **Novembre 2005**

---

[1] sottomesso a *AMCS*

[a] ICAR-CNR
[2] University of Naples Parthenope

A parallel Block Lanczos algorithm and its implementation for the evaluation of some eigenvalues of large sparse symmetric matrices on multicomputers

Mario Rosario Guarracino [a] Francesca Perla [b,*] Paolo Zanetti [b]

[a] *Institute for High Performance Computing and Networking - ICAR-CNR*
Via P. Castellino, 111 - 80131 Naples, Italy
e-mail: mario.guarracino@icar.cnr.it

[b] *University of Naples Parthenope*
Via Medina, 40 -80133 Naples, Italy
e-mail: {paolo.zanetti, francesca.perla}@uniparthenope.it

Abstract

In the present work we describe HPEC (High Performance Eigenvalues Computation), a parallel software for the evaluation of some eigenvalues of a large sparse symmetric matrix. It implements a Block Lanczos algorithm efficient and portable for distributed memory multicomputers. HPEC is based on basic linear algebra operations for sparse and dense matrices, some of which have been derived by ScaLAPACK library modules. Numerical experiments have been carried out to evaluate HPEC performance on a cluster of workstations with test matrices from Matrix Market and Higham's collections. A comparison with a PARPACK routine is also detailed. Finally, parallel performance is evaluated on random matrices, using standard parameters.

Key words: Symmetric Block Lanczos Algorithm, Sparse Matrices, Parallel Eigensolver, Cluster architecture

* Corresponding author. Email address: francesca.perla@uniparthenope.it

1 Introduction

The eigenvalue problem has a deceptively simple formulation and the background theory has been known for many years, yet the determination of accurate solutions presents a wide variety of challenging problems. That has been stated by Wilkinson [27] some forty years ago, and it is still true.

Eigenvalue problems are the computational kernel of a wide spectrum of applications ranging from structural dynamics and computer science, to economy. The relevance of those applications has lead to a wide effort in developing numerical software in sequential environment. The results of this intensive activity are both single routines in general and special purpose libraries, such as Nag, IMSL and Harwell Library, and specific packages, such as LANCZOS [7], LANSO [23], LANZ [18], TRLAN [29], and IRBLEIGS [1].

If we look at sequential algorithms, we see that nearly all are based on well known *Krylov subspace methods* [22,24]. The reasons of this are the demonstrated computational efficiency and the excellent convergence properties which can be achieved by these procedures. In spite of some numerical difficulties arising from their implementation, they form the most important class of methods available for computing eigenvalues of large, sparse matrices. It is worth noting that a broad class of applications consists of problems that involve a symmetric matrix and requires the computation of few extremal eigenvalues. For this class the *Lanczos algorithm* [19] appears to be the most promising solver.

The availability and widespread diffusion of low cost, off the shelf clusters of workstations have increased the request of black box computational solvers, which can be embedded in easy to use problem solving environments. To achieve such goal, it is necessary to provide simple application programming interfaces and support routines for input/output operation and data distribution. At present, little robust software is available, and a straightforward implementation of the existing algorithms does not lead to an efficient parallelization, and new algorithms have yet to be developed for the target architecture. The existing packages for sparse matrices, such as PARPACK package [20], PNLASO [28], SLEPc [16] and TRLAN, implement Krylov projection methods and exploit parallelism at matrix-vector products level, i.e. level 2 BLAS operation. Nevertheless, for dense matrices, some packages have been implemented with level 3 BLAS [26].

In this work we present an efficient and portable software for the computation of few extreme eigenvalues of a large sparse symmetric matrix based on a reorganization of block Lanczos algorithm for distributed memory multicomputers, which allows to exploit a larger grain parallelism and to harness the

computational power of the target architecture.

The rest of this work is organized as follows: in section 2 we describe the block version considered and we show how we reorganize the algorithm in order to reduce data communication. In section 3 we deal with its parallel implementation, providing computational and communication complexity and implementation details. In section 4 we focus on the specification and architecture of the implemented software. In section 5, we present the numerical experiments that have been carried out to compare the performance of HPEC with PARPACK software on test matrices from Matrix Market and Higham's collections. Finally, parallel performance evaluation, in terms of efficiency on random matrices, is also shown.

2 Block Lanczos Algorithm

The *Lanczos algorithm* for computing eigenvalues of a symmetric matrix $A \in \mathbb{R}^{m\times m}$ is a projection method that allows to obtain a representation of the operator in the Krylov subspace spanned by the set of orthonormal vectors, called *Lanczos vectors*. In this subspace the representation of a symmetric matrix is always tridiagonal. Assuming $m = rs$, the considered *Block Lanczos algorithm*, proposed in [11], generates a symmetric banded block tridiagonal matrix T having the same eigenvalues of A:

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

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

$$Q^T A Q = T,$$

where

$$Q = [X_1\ X_2\ \ldots\ X_r], \quad X_i \in \mathbb{R}^{m\times s}$$

is an orthonormal matrix, and its columns are the Lanczos vectors. A direct way to evaluate M_j, B_j and X_j [11,24] is described below.

Choose $X_1 \in \mathbb{R}^{m\times s}$ *such that*

$X_1^T X_1 = I_s$, $X_0 \equiv 0$, $B_0 \equiv 0$

$M_1 = X_1^T A X_1$

for $j = 1$ **to** $r - 1$

$R_j = AX_j - X_{j-1}B_{j-1}^T - X_j M_j$

$R_j = X_{j+1}B_j$ (*QR factorization*)

$M_{j+1} = X_{j+1}^T A X_{j+1}$

endfor

Algorithm 1: Block Lanczos Algorithm (I version)

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

$$Q_j^T A Q_j = T_j \quad (Q_j = [X_1\ X_2\ \ldots\ X_{j+1}]),$$

where $Q_j^T Q_j = I$. In facts, when j grows T_j extremal eigenvalues, called *Ritz values* of A, are increasingly better approximation of A extremal eigenvalues.

Block versions allow approximations of eigenvalues with multiplicity greater than one, while in single vector algorithms difficulties can be expected since the projected operator, in finite precision, is unreduced tridiagonal, which implies it cannot have multiple eigenvalues [11].
Numerical stability of Block Lanczos algorithm [2,21] can derived from the one for single vector version. As we have shown in [15], the following theorem holds:

Theorem 2.1 *(Block Lanczos Error Analysis)*
Let A be a $m \times m$ real symmetric matrix with at most nza nonzero entries in any row and column. Suppose the Block Lanczos algorithm with starting matrix $X_1 \in R^{m\times s}$, implemented in floating-point arithmetic with machine precision ϵ_M, reaches the j-th step without breakdown. Let the computed $\tilde{M}_j$, $\tilde{B}_j$ and $\tilde{X}_{j+1}$ satisfy

$$A\tilde{Q}_j = \tilde{Q}_j\tilde{T}_j + \tilde{X}_{j+1}\tilde{B}_j E_j^T + F_j$$

where $E_j = [0, 0, \ldots, I_s]^T \in R^{s\times sj}$,

$$\tilde{Q}_j = [\tilde{X}_1, \ldots, \tilde{X}_j],$$

and

$$\tilde{T}_j = \begin{pmatrix} \tilde{M}_1 & \tilde{B}_1^T & & & \\ \tilde{B}_1 & \tilde{M}_2 & \tilde{B}_2^T & & \\ & \ddots & \ddots & \ddots & \\ & & \tilde{B}_{j-2} & \tilde{M}_{j-1} & \tilde{B}_{j-1}^T \\ & & & \tilde{B}_{j-1} & \tilde{M}_j \end{pmatrix}.$$

Then,

$$|F_j| \leq [m(1+sG_1)+3]\gamma|A||\tilde{Q}_j| + [s(1+sG_1)+3]\gamma|\tilde{Q}_j||\tilde{T}_j| + O(\epsilon_M^2)$$

with $\|G_1\|_F = 1$ and $\gamma = max\{\epsilon_M, \frac{cm\epsilon_M}{1-cm\epsilon_M}\}$, where c is a small integer constant whose value is unimportant.

3 Parallel Implementation

3.1 Modified Block Lanczos Algorithm

In previous works [13,14] we proposed a parallel implementation of the symmetric Block Lanczos algorithm for MIMD distributed memory architectures configured as a 2-D mesh. We showed that a direct parallelization of the Algorithm 1 has efficiency values, in the considered computational environments, that deteriorate when sparsity decreases. This loss of efficiency is due to the amount of communication, with respect to computational complexity, required in the matrix-matrix multiplication, when the first factor is A sparse. This behavior depends on ScaLAPACK [4] implementation choices for matrix-matrix operations, since the first matrix is involved in global communications and the second one only in one-to-one communications. Then, to avoid this phenomena, we reorganized the 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 1 by its transpose. Since A is a symmetric matrix, and so M_j, $j = 1, \ldots, r-1$, we obtained the following version of the Block Lanczos algorithm.

Choose $X_1^T \in \mathbb{R}^{s \times m}$ *such that*

$X_1^T X_1 = I_s, X_0 \equiv 0, B_0 \equiv 0$

$M_1 = X_1^T A X_1$

for $j = 1$ **to** $r - 1$

$R_j^T = X_j^T A - B_{j-1} X_{j-1}^T - M_j X_j^T$

$R_j = X_{j+1} B_j \quad (QR\ fact.,\ obtaining\ X_{j+1}^T)$

$M_{j+1} = X_{j+1}^T A X_{j+1}$

end for

Algorithm 2: Block Lanczos Algorithm (II version)

Substituting the sparse matrix-matrix operation AX_j by the evaluation of the product $X_j^T A$, we obtained that X_j^T was involved in communication instead of A and therefore, there was a reduction in terms of execution times, communication complexity and a gain in terms of *speed-up* and *efficiency*, as shown in [14]. Algorithm 2 has the same numerical properties of Algorithm 1, since the use of transposed factors does not alter its behavior with respect to round-off errors.

3.2 Data distribution

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

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 [8], and the considered connection topology is 2-D mesh, we choose the *block scatter decomposition* [4] for all matrices involved in the algorithm, including the sparse one, since this strategy allows the use of ScaLAPACK. For the memorization scheme of A sparse we use a data structure, per process, usually referenced as *CSC - Compress Sparse Column* (see for example [24]), consisting of three arrays, respectively containing:

(1) the non-zero entries of A columns parts in the subblocks that are assigned to the process;
(2) the rows 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 column part.

Therefore, the global sparse matrix storage is a block scattered CSC. This memorization scheme is redundant for a symmetric matrix, but it allows a faster memory access to data, an easier localization of a whole column and a decrease in global communication.

If we look at Algorithm 2, we see that the linear algebra operations involved are essentially matrix-matrix multiplications, eventually with a transposed factor or a sparse factor, and a QR factorization. Before implementing Algorithm 2, according to the described 2-D mesh approach, we observe that a global transposition of the matrix R_j^T is neeeded at each iteration before evaluating the QR factorization. Since transposition operations in a message passing environment are extremely time consuming, due to the accesses needed to non local memories, we substitute the QR factorization by the LQ factorization that allows to access the matrix R_j^T without transposition. Then, the implemented algorithm is the following:

Choose $X_1^T \in \mathbb{R}^{s \times m}$ *such that*

$X_1^T X_1 = I_s, X_0 \equiv 0, B_0 \equiv 0$

$M_1 = X_1^T A X_1$

for $j = 1$ **to** $r - 1$

$R_j^T = X_j^T A - B_{j-1} X_{j-1}^T - M_j X_j^T$

$R_j^T = B_j X_{j+1}^T$ (**LQ fact.**)

$M_{j+1} = X_{j+1}^T A X_{j+1}$

end for

Algorithm 3: Block Lanczos Algorithm (III version)

The proposed Algorithm 3 represents the computational kernel of the software, which will be described in detail in the next section.

3.4 Computational and communication complexity

In this section we deal with the computational and communication aspects of the implemented Block Lanczos algorithm.
Let nza be the number of non-zero entries of the sparse $A \in \mathbb{R}^{m \times m}$ and s the number of Lanczos vectors, the operation count for each complete step of the sequential Block algorithm asymptotically is:

$$7m \times s^2 + 2nza \times s - 2s^3/3 \; floating - point \; operations.$$

The cost of a complete step of the parallel implementation of Algorithm 3, for each of p computing nodes, is:

$$(7m \times s^2 + 2nza \times s - 2s^3/3)/p \; floating - point \; operations,$$

$$4m \times s + 2nza \qquad one - to - one \; communications,$$

$$m \times s + s^2 \qquad one - to - all \; communications.$$

When nza increases the number of operations involving the sparse factor becomes dominant. It is also worth noticing that the operation count is not affected by parallelization. Furthermore, the communication complexity is one order of magnitude less than computational complexity, that is generally considered a target in linear algebra parallel algorithms.

4 Software Description

4.1 Software architecture

HPEC uses standard message passing libraries, i.e. BLACS [9] and MPI [12] and standard numerical linear algebra software, PBLAS [5] and ScaLAPACK, obtaining a software as portable as PARPACK.

PBLAS routine used in Algorithm 3 is PDGEMM to evaluate matrix-matrix multiplications with dense factors.

Routines for matrices factorization are not included in PBLAS, but they are covered by ScaLAPACK. The evaluation of B_j, at each iteration, is then performed by using the routine PDGELQF, and the routine PDORGLQ is used

to compute the matrix X_{j+1}^T of the LQ factorization.

We developed PDMASPMA routine to evaluate matrix-sparse matrix products, using the same scattered decomposition on which PDGEMM is based. On each node the computational kernel is a sequential matrix-sparse matrix product in which the access to the elements of the sparse factor is done accordingly to the data layout scheme. The advantage is that, since sparse factor is not involved in communication, 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.

Since all matrices involved in the algorithm are distributed among processing nodes, no replication of data occurs.

4.2 Software specification

The proposed HPEC is implemented in C and Fortran 77. It uses reverse communication strategy for the sparse matrix A. The driver routine is named LANCZOS, and its stub is the following:

```
    SUBROUTINE LANCZOS (S, M, A, LMA, AI, AJ, XT1, LDXT1, MB, CONTXT,
1               NUMSEA, NUMAUT, W, ORFAC, ABSTOL, NMAX, IFND, IIFAIL)
    DOUBLE PRECISION   A(*), XT1(LDXT1,*), W(*), ORFAC, ABSTOL
    INTEGER            S, M, LMA, AI(*), AJ(*), LDXT1, MB, CONTXT
    INTEGER            NUMSEA, NUMAUT, NMAX, IFND(*), IIFAIL
```

User needs to provide the sparse matrix `A` in block CSC format, an initial block `XT1` consisting of `S` Lanczos' vectors, and the required absolute tolerance `ABSTOL`. With respect to other packages, such as PARPACK, which require an user supplied matrix-vector product, HPEC user needs to provide the sparse matrix A either in a data file, or via a function to compute A blocks, for given rows and columns indexes.
HPEC supports different input sparse matrix formats:

Compress Sparse Column : described in section 3.2.
Coordinate Storage Scheme : it records each nonzero entry together with its row and column index.
RSA Harwell-Boeing format : each column is held as a sparse vector, represented by a list of row indices of the entries in a integer array and a list

of corresponding values in a separate array; a multiple line header contains information about the matrix.

HPEC has two utility routines for the management of distributed matrices: PDMATDIS implements the block scattered decomposition and distribution of a dense matrix on a 2D mesh topology, while PDSPMDIS executes the same operations on a sparse matrix.

5 Numerical Experiments

All numerical experiments described in the present section refer to a cluster of 8 AMD Athlon XP 2400+ processors with 384MB DDR RAM connected by a 100 Megabit/s Fast Ethernet switch, operated by the University of Naples *Parthenope*; clustering middleware is Oscar 3.0, which includes gcc-3.3.2, MPICH 1.2.5.10, BLACS 1.1 and ScaLAPACK 1.7.
We firstly compare numerical results and execution times of HPEC and `PDSDRV1` PARPACK driver on two test matrices. We briefly recall PARPACK is a parallel version of ARPACK software and it is targeted for multicomputers. It is written in Fortran 77 and implements Implicitly Restarted Arnoldi Method for solving large sparse eigenvalue problems. PARPACK uses single-vector version of the algorithm, thus exploiting parallelism at matrix-vector products level. `PDSDRV1` needs a parallel sparse matrix-vector routine coherent with PARPACK internal data distribution. Among available software, we decided to use `F11XBFP` routine from the *de facto* standard NAg Parallel Library (http://www.nag.com/numeric/FD/manual/html/FDlibrarymanual.asp). Our choice has been motivated by the fact it uses the same cyclic row block distribution as PARPACK. We wish to emphasize that for an unskilled user the task of finding and using a parallel sparse matrix-vector code can be difficult, since software publicly available, such as P-SPARSLIB [25] and PSBLAS [10], has been motivated by particular numerical problems and implemented within larger software projects, and thus computational kernels are not easy to include in other packages. Parallel software libraries that contain general purpose low level modules, such as NAg parallel software library and PESSL (http://publib.boulder.ibm.com/doc_link/en_US/a_doc_lib/sp34/essl/essl02.html), are commercial products.

Numerical experiments have been performed on two test matrices taken from *Matrix Market* [3] and *Test Matrix Toolbox* for Matlab [17].
Matrix Market provides convenient access to a repository of test data for use in comparative studies of algorithms for numerical linear algebra. Matrices as well as matrix generation software and services are provided. Test Matrix Toolbox has been implemented by N.J. Higham. Not only does it contain test matrices, but also provides various tools for visualising and generating test

Table 1
Test matrices characteristics

Name	Size	nza	average nza per col.	longest col. (nza)	shortest col. (nza)	Frobenius norm
PLATZ1919	1919	32399	17	682 (19)	63 (3)	22
WATHEN(100,100)	30401	471601	8.2	3 (11)	30401 (1)	1.5 e+04

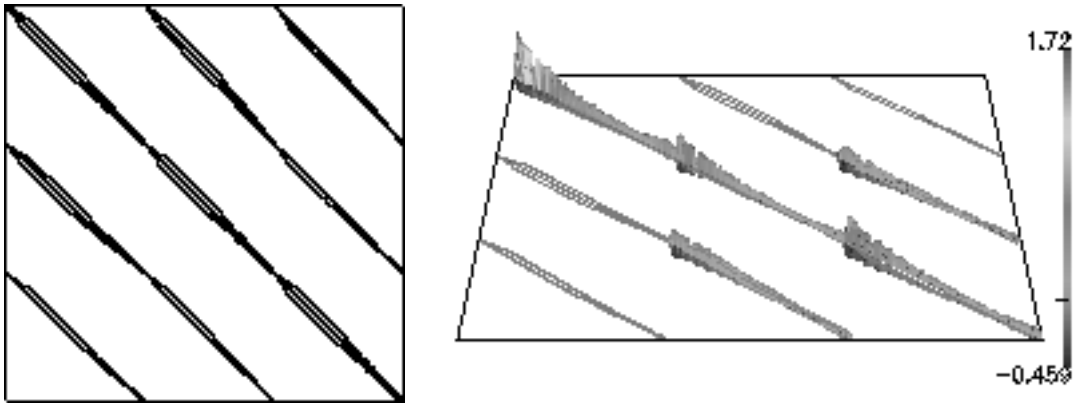


Fig. 1. PLATZ1919 pattern and elements magnitude

problems in Matlab.

The two selected matrices have size, number of non zero entries, sparsity patterns and conditioning, as shown in Table 1.

PLATZ matrix [6] is a finite-difference model for the shallow wave equations for the Atlantic and Indian Oceans (Fig. 1). The original matrix is derived as the (negative) square of a purely imaginary skew-symmetric matrix. Hence, the eigenvalues occur in pairs (except for an isolated singleton at zero).

Tables 2 and 3 show execution times in seconds obtained on 1 and 4 processors respectively, to seek 1, 2, 4 and 10 largest eigenvalues in magnitude of PLATZ1919 with PARPACK and HPEC, with a fixed user tolerance in the order of machine (double) precision for the computed eigenvalues.

We report the best execution time of PARPACK using 16, 32, 64 and 128 Arnoldi's vectors, and in brackets, the number of vectors for which it has been obtained. A similar methodology has been used to determine the number of Lanczos' vectors for HPEC, and very often the best choice is 16 vectors. The block algorithm implemented by HPEC allows to computed multiple eigenvalues at the same time, as it can be observed in Table 2.

We observe that in all cases execution times of the two tested software are comparable. Finally, since the problem is very small, no significant performance

Table 2
Execution times in seconds for PLATZ1919 matrix on 1 processor

Eigenvalues	1	2	4	10
PARPACK	12.02 (32)	12.25 (64)	16.89 (32)	31.51 (32)
HPEC	9.40(16)	9.40 (16)	9.40 (16)	10.24 (32)

Table 3
Execution times in seconds for PLATZ1919 matrix on 4 processors

Eigenvalues	1	2	4	10
PARPACK	4.60 (64)	4.65 (64)	6.34 (32)	10.04 (32)
HPEC	6.42 (16)	6.42 (16)	7.17 (16)	10.24 (16)

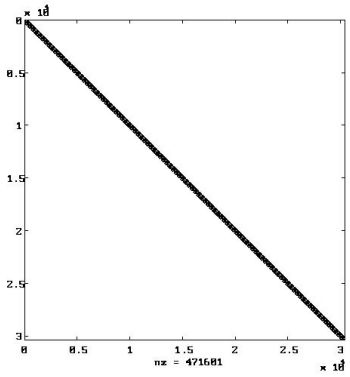


Fig. 2. WATHEN(100,100) nonzero pattern

gain has been observed on a larger number of processors.

WATHEN(NX,NY) is a sparse random N-by-N finite element matrix where N = 3*NX*NY + 2*NX + 2*NY + 1.
It is precisely the 'consistent mass matrix' for a regular NX-by-NY grid of 8-node (serendipity) elements in 2 space dimensions. WATHEN(NX,NY) is symmetric positive definite for any (positive) values of the 'density', RHO(NX,NY), which is chosen randomly in this examples. In particular, if D = DIAG(DIAG(A)), then $0.25 \leq EIG(INV(D) * A) \leq 4.5$ for any positive integers NX and NY and any densities RHO(NX,NY).

Since the eigenproblems have a worst conditioning with respect to the previous examples, user tolerance has been set to .1D-7, to limit the number of iterations.

Table 4
Execution times in seconds for WATHEN(100,100) matrix on 1 processor

	1	2	4	10
PARPACK	6042 (32)	8642 (32)	15611 (32)	24691 (32)
HPEC	1210 (32)	1533 (32)	2874 (64)	5402 (32)

Table 5
Execution times in seconds for WATHEN(100,100) matrix on 8 processors

	1	2	4	10
PARPACK	782 (64)	1104 (32)	1507 (64)	6567 (64)
HPEC	580 (32)	732 (32)	1320 (32)	2504 (64)

As reported in Tables 4 and 5, excution time decreases as the number of processors increases. On one processor, HPEC execution time is sensibly less than PARPACK, due both to greater granularity in dense operations and the absence of reorthogonalization steps.

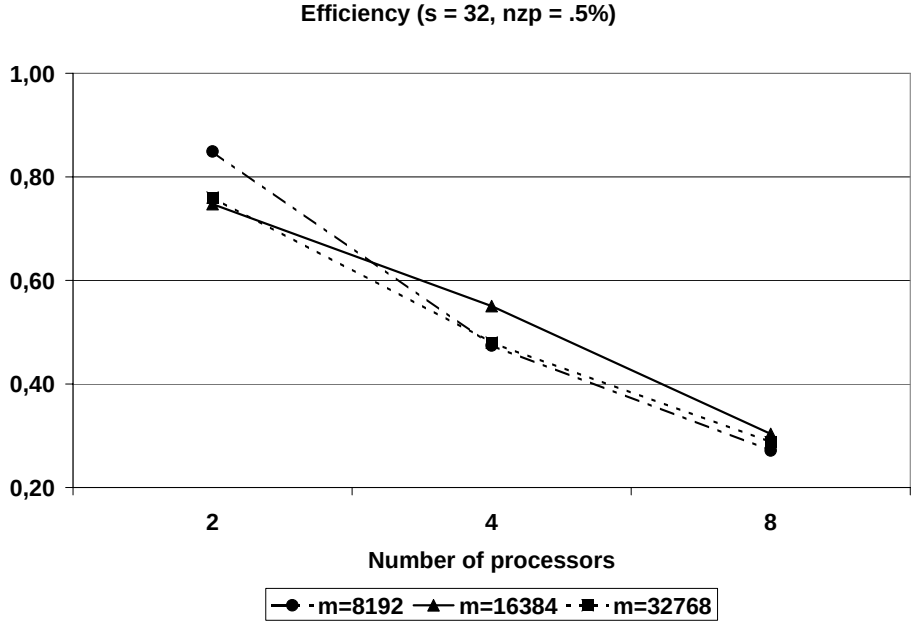
We note the number of Arnoldi's and Lanczos' vectors to optimize the execution time, is something strictly related to the problem and cannot be estimated a priori, as we can see from Tables 4 and 5, and indeed software libraries usually leave to users the choice.
All tests confirm that the algorithm implemented in HPEC preserves the same well known numerical properties of the Block Lanczos algorithm, and in particular the ability to evaluate multiple eigenvalues, and the capability of evaluate eigenvalues of ill conditioned problems.

5.1 Parallel performance evaluation

In this section we evaluate the performance of the parallel implementation, using standard parameters. In particular we want to estimate the gain, in terms of execution time, when an increasing number of processors is used, fixed the problem size, sparsity and number of Lanczos' vectors. Since their number cannot be chosen to fit all problems, we tested different block sizes.

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 three values for the number of Lanczos' vectors $s = 32, 64, 128$. 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. To evaluate the effect of parallelization, we use the classical parameter *efficiency*

Fig. 3. Efficiency values on $p = 2, 4, 8$ nodes, for $s = 32$, $nzp = .5\%$, $m = 8192, 16384, 32768$



(E_p):

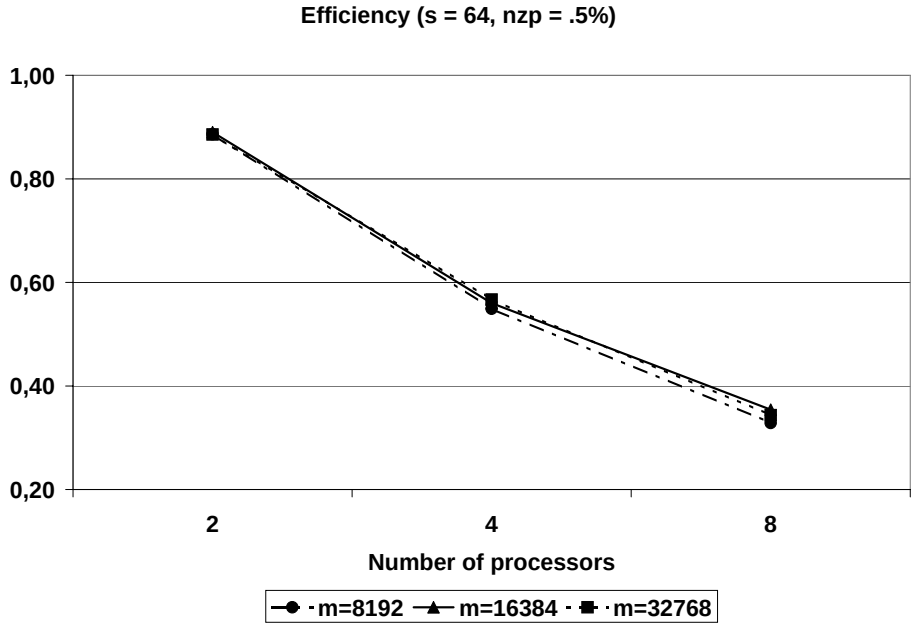
$$E_p = \frac{T_1(m, s, nzp)}{p \cdot T_p(m, s, nzp)},$$

where $T_j(m, s, nzp)$ is the execution time of the 10^{th} iteration on j nodes for a fixed size problem [1] . As we showed in section 3.4, the operation count for each iteration of the algorithm depends on s^2, and therefore the execution time of the single iteration increases accordingly. For this reason, we choose the 10^{th} iteration, which provides a sufficient granularity for the dense operations to justify the use of multiple processors.

We note efficiency values on 2 nodes for all tests, never drop below 0.75, while it is at least 0.47 on 4 nodes and at most 0.60 on 8 processors. The efficiency values grows, fixed m and s, when nzp increases: this is an expected result,

[1] All the execution times have been obtained using MPICH routine *MPI_WTIME()*.

Fig. 4. Efficiency values on $p = 2, 4, 8$ nodes, for $s = 64$, $nzp = .5\%$, $m = 8192, 16384, 32768$



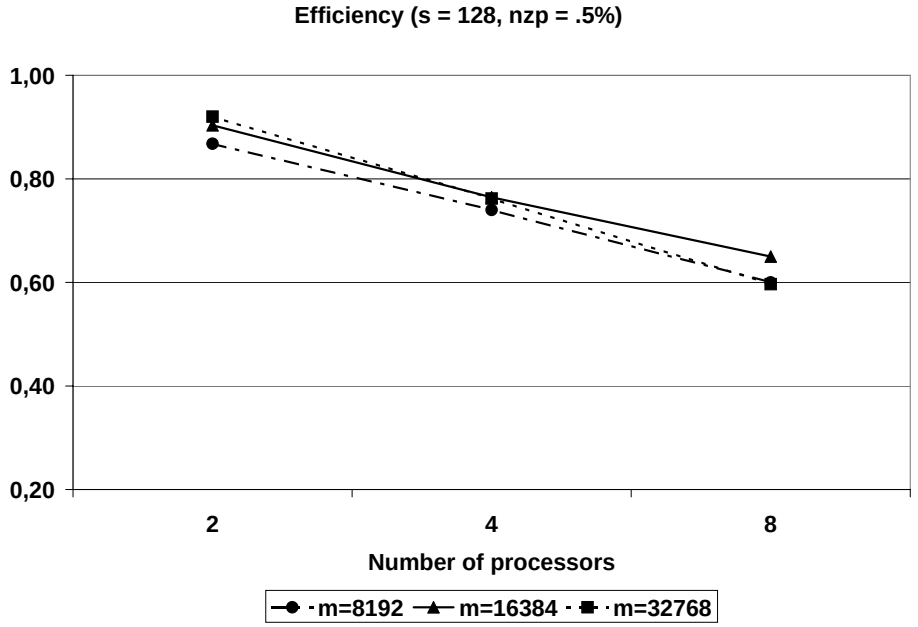
since in the analysis of communication and computation complexity shown in section 3.4, one to all communication does not involve sparse factor A. Further, efficiency increases for larger values of m, which provides a hint software can be efficient for a growing number of processors.

All results show the implemented parallelization strategy allows to reduce the execution times using more than 1 processor and that HPEC is efficient on the target architectures for problems of adequate dimension.

6 Summary

In present work we present HPEC, a freely available parallel software, based on a variant of the Block Lanczos algorithm for the real, sparse symmetric eigenvalue problem. The software is based on the linear algebra library ScaLAPACK and BLACS communication library. It provides a simple application

Fig. 5. Efficiency values on $p = 2, 4, 8$ nodes, for $s = 128$, $nzp = .5\%$, $m = 8192, 16384, 32768$



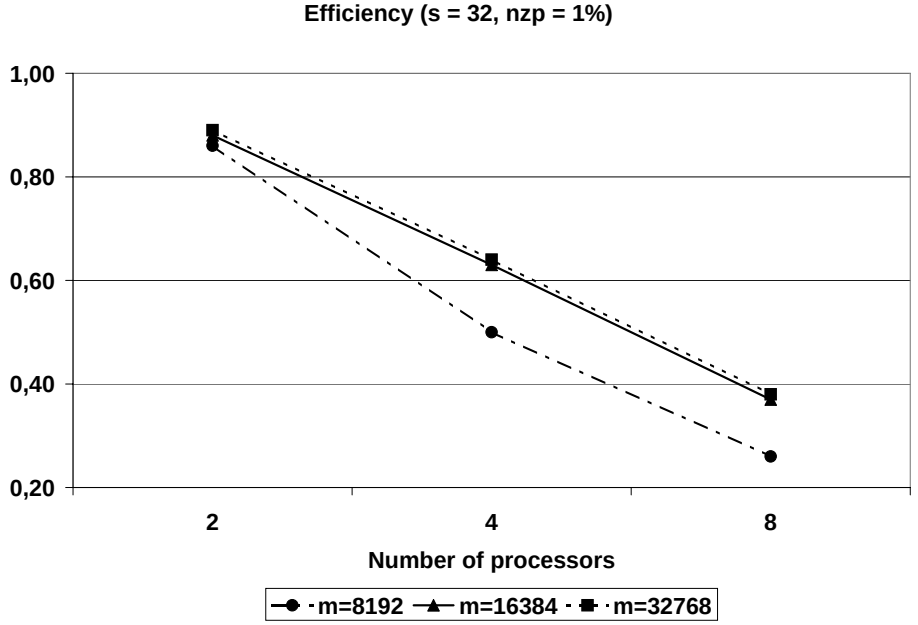
programming interface and supplies decomposition and distribution routines for dense and sparse matrices.

Results of numerical experiments and performance evaluation, confirm the numerical and efficiency qualities of the proposed software.

7 Acknowledgements

This work has been partially supported by Center for Research Enrico Fermi under Project *Matematica e Diagnosi Medica*. Authors would like to thank Nag staff for supporting this work and referees to greatly improve the quality of present work.

Fig. 6. Efficiency values on $p = 2, 4, 8$ nodes, for $s = 32$, $nzp = 1\%$, $m = 8192, 16384, 32768$



References

[1] J. Baglama, D. Calvetti and L. Reichel, Irbleigs: A MATLAB program for computing a few eigenpairs of a large sparse Hermitian matrix, *ACM TOMS* **29** (2003) 337-348.

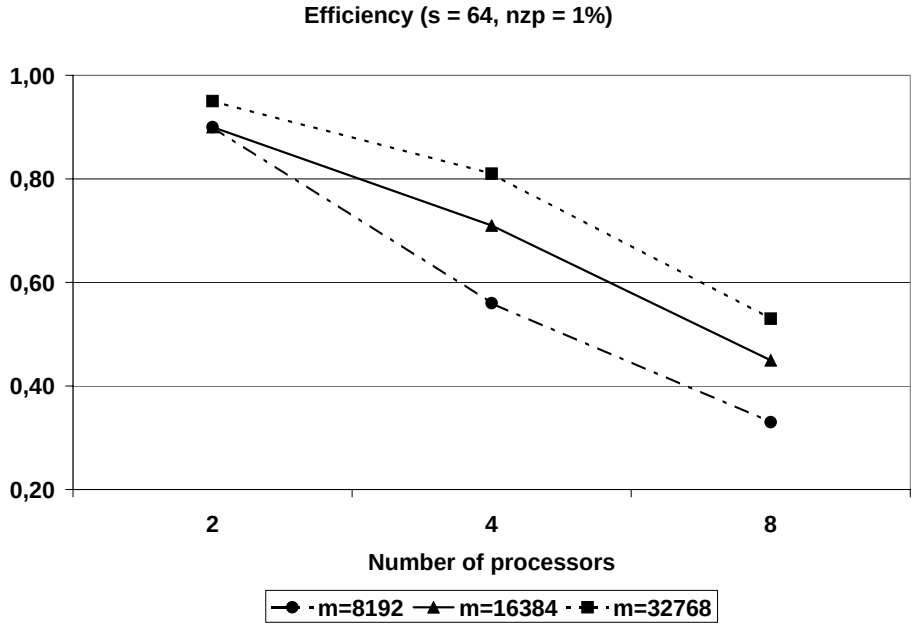
[2] Z. Bai, "Error Analysis of the Lanczos Algorithm for the Nonsymmetric Eigenvalues Problem", *Math. Comp.*, Vol. 62, n. 205, pp. 209-226, 1994.

[3] R. F. Boisvert, R. Pozo, K. Remington, R. Barrett and J. Dongarra, The Matrix Market: A web resource for test matrix collections, in: Ronald F. Boisvert, editor, *Quality of Numerical Software, Assessment and Enhancement*, (Chapman & Hall, 1997), 125-137.

[4] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker and R.C. Whaley, ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design and Performance, *Computer Physics Communications* **97** (1996) 1-15.

[5] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker and R. C. Whaley,

Fig. 7. Efficiency values on $p = 2, 4, 8$ nodes, for $s = 64$, $nzp = 1\%$, $m = 8192, 16384, 32768$



A Proposal for a Set of Parallel Basic Linear Algebra Subprograms, UT-CS-95-292, (1995).

[6] A. K. Cline, G. H. Golub, and G. W. Platzman, Calculations of normal modes of oceans using a Lanczos method, in: J. R. Bunch and D. J. Rose eds., *Sparse Matrix Computations*, (Academic Press, London and New York, 1976).

[7] J. K. Cullum and R. A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations Vol.I: Theory*, (SIAM, Phildelphia, 2002).

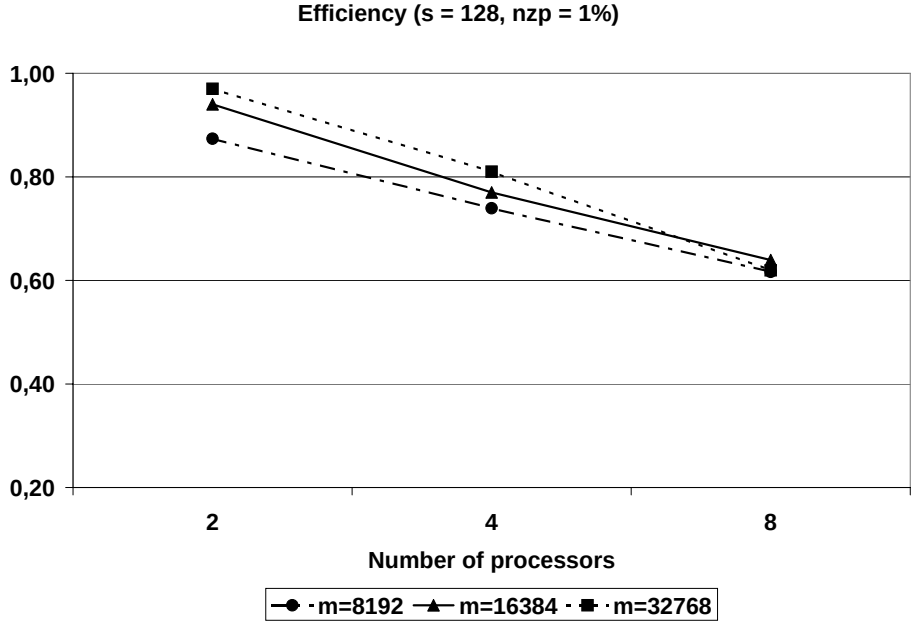
[8] J. Dongarra, J. Du Croz, S. Hammarling and I. Duff, A Set of Level 3 Basic Linear Algebra Subprograms *ACM Trans. Math. Soft* **16** (1990) 1-17.

[9] J. Dongarra and R. C. Whaley, A User's Guide to the BLACS v1.1, UT-CS-95-281, (1995).

[10] S. Filippone and M. Colajanni, "PSBLAS: A library for parallel linear algebra computation on sparse matrices", *ACM Trans. on Math. Software*, Vol. 26, n. 4, pp. 527-550, 2000.

[11] G. H. Golub and C. F. Van Loan, *Matrix Computations, 2nd edition*, (Johns Hopkins Univ. Press, 1989).

Fig. 8. Efficiency values on $p = 2, 4, 8$ nodes, for $s = 128$, $nzp = 1\%$, $m = 8192, 16384, 32768$



[12] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface*, (The MIT Press, 1999).

[13] M.R. Guarracino and F. Perla, A Parallel Block Lanczos Algorithm for Distributed Memory Architectures, *J. Par. Alg. Appl.*, **4** (1994) 211-221.

[14] M.R. Guarracino and 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.

[15] M.R. Guarracino and F. Perla, A Sparse, Symmetric Eigensolver for Distributed Memory Architectures: Parallel Implementation and Numerical Stability, Tech. Rep. CPS 11 (1995).

[16] V. Hernandez, J. E. Roman and V. Vidal, SLEPc: Scalable Library for Eigenvalue Problem Computations, *Lecture Notes in Computer Science* **2565** (2003) 377-391.

[17] N. J. Higham, The Test Matrix Toolbox for MATLAB (version 3.0), Manchester Centre for Computational Mathmatics, Tech. Rep. 276, (1995).

[18] M.T. Jones and M.L. Patrick, The Use of Lanczos's Method to Solve the Large Generalized Symmetric Definite Eigenvalue Problem, Tech. Rep. ICASE 89-67 (1989).

[19] 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.

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

[21] C. Paige, "Error Analysis of the Lanczos Algorithm for Tridiagonalizing a Symmetric Matrix", *J. Inst. Math. Appl.*, Vol. 18, pp. 341-349, 1976.

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

[23] B. N. Parlett and D. S. Scott. The Lanczos algorithm with selective orthogonalization, *Math. Comp.* **33** (1979) 217-238.

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

[25] Y. Saad and A.V. Malevsky, P-SPARSLIB: a portable library of distributed memory sparse iterative solvers, in: V. E. Malyshkin, et. al. eds., *Proceedings of Parallel Computings Technologies (PaCT-95), 3rd International Conference, St. Petersburg*, (1995).

[26] F. Webster and G. Lo. Projective block Lanczos algorithm for dense, Hermitian eigensystems, *J. Comput. Phys.*, **124** (1996) 146-161.

[27] J. H. Wilkinson, *The Algebraic Eigenvalue Problem* (Clarendon Press, Oxford, 1965).

[28] K. Wu and H. Simon, Parallel Efficiency of the Lanczos Method for Eigenvalue Problems, Tech. Rep. LBNL-42828 (1999).

[29] K. Wu and H. Simon, TRLAN user guide, Lawrence Berkeley National Laboratory, Tech. Rep. LBNL-42953 (1999).