A PARALLEL BLOCK LANCZOS ALGORITHM AND ITS IMPLEMENTATION FOR THE EVALUATION OF SOME EIGENVALUES OF LARGE SPARSE SYMMETRIC MATRICES ON MULTICOMPUTERS

MARIO ROSARIO GUARRACINO[*], FRANCESCA PERLA[**], PAOLO ZANETTI[**]

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

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

In the present work we describe HPEC (High Performance Eigenvalues Computation), a parallel software package for the evaluation of some eigenvalues of a large sparse symmetric matrix. It implements an efficient and portable Block Lanczos algorithm 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.

Keywords: symmetric block Lanczos algorithm, sparse matrices, parallel eigensolver, cluster architecture

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. This has been stated by Wilkinson (1965), and 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 painstaking efforts in developing numerical software in sequential environments. 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 (Cullum and Willoughby, 2002), LANSO (Parlett and Scott, 1979), LANZ (Jones and Patrick, 1989), TRLAN (Wu and Simon, 1999b), and IRBLEIGS (Baglama *et al.*, 2003).

If we look at sequential algorithms, we see that nearly all of them are based on the well-known *Krylov subspace methods* (Parlett, 1980; Saad, 1992). The reasons for 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 the eigenvalues of large, sparse matrices. It is worth noting that a broad class of applications consists of problems that involve a symmetric matrix and require the computation of few extremal eigenvalues. For this class the *Lanczos algorithm* (Lanczos, 1950) 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 this goal, it is necessary to provide simple application programming interfaces and support routines for input/output operations 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 (Lehoucq *et al.*, 1998), PNLASO (Wu and Simon, 1999a), SLEPc (Hernandez *et al.*, 2003) and TRLAN, implement Krylov projection methods and exploit parallelism at the matrix-vector products level, i.e., level 2 BLAS operations. Nevertheless, for dense matrices, some packages have been implemented with level 3 BLAS (Webster and Lo, 1996).

In this work we present efficient and portable software for the computation of few extreme eigenvalues of a large sparse symmetric matrix based on a reorganization of the block Lanczos algorithm for distributed memory multicomputers, which allows us 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 the eigenvalues of a symmetric matrix $A \in \mathbb{R}^{m\times m}$ is a projection method that allows us to obtain a representation of an 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 analysed *block Lanczos algorithm*, proposed in (Golub and Van Loan, 1989), 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

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

is an orthonormal matrix, and its columns are Lanczos vectors. A direct way to evaluate M_j, B_j and X_j (Golub and Van Loan, 1989; Saad, 1992) is described as Algorithm 1. At the step j, Algorithm 1 produces a symmetric banded block tridiagonal matrix T_j of the order $(j+1)\times s$ satisfying

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

Algorithm 1: Block Lanczos algorithm (1-st version).

```
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)
    $M_{j+1} = X_{j+1}^T A X_{j+1}$
  end for
```

where $Q_j^T Q_j = I$. In fact, when j increases, extremal eigenvalues of T_j, called the *Ritz values* of A, form an increasingly better approximation of extremal eigenvalues of A.

Block versions permit approximations of eigenvalues with multiplicities greater than one, while in single vector algorithms difficulties can be expected since the projected operator, in finite precision, is unreduced tridiagonal, which implies that it cannot have multiple eigenvalues (Golub and Van Loan, 1989).

The numerical stability of the block Lanczos algorithm (Bai, 1994; Paige, 1976) can be derived from the one for the single vector version. As we have shown in (Guarracino and Perla, 1995b), the following theorem holds:

Theorem 1. *(Block Lanczos Error Analysis)* Let A be an $m \times m$ real symmetric matrix with at most nza nonzero entries in any row and column. Suppose that the block Lanczos algorithm with the starting matrix $X_1 \in \mathbb{R}^{m\times s}$, 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

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

where

$$E_j = [0, 0, \dots, I_s]^T \in \mathbb{R}^{s\times sj},$$
$$\tilde{Q}_j = [\tilde{X}_1, \dots, \tilde{X}_j],$$

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

$$\boxed{\begin{aligned}|F_j| \le{}& [m(1+sG_1)+3]\gamma|A||\tilde{Q}_j|\\ &+[s(1+sG_1)+3]\gamma|\tilde{Q}_j||\tilde{T}_j| + O(\epsilon_M^2)\end{aligned}}$$

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 (Guarracino and Perla, 1994;1995a), 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 Algorithm 1 in the computational environments considered has efficiency values that deteriorate when sparsity decreases. This loss of efficiency is due to the amount of communication with respect to computational complexity required in matrix-matrix multiplication when the first factor is A sparse. This kind of behavior depends on ScaLAPACK (Choi *et al.*, 1996) 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 these phenomena, we reorganized the algorithm in such a way that 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 Algorithm 1 by its transpose. Since A is a symmetric matrix, and so M_j, $j = 1,\ldots,r-1$, we obtained the version of the block Lanczos algorithm embodied by Algorithm 2.

Algorithm 2: Block Lanczos algorithm (2-nd version).

```
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 $r-1$
    $R_j^T = X_j^T A - B_{j-1}X_{j-1}^T - M_j X_j^T$
    $R_j = X_{j+1}B_j$   (QR fact., obtaining $X_{j+1}^T$)
    $M_{j+1} = X_{j+1}^T A X_{j+1}$
  end for
```

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 and communication complexity, and a gain in terms of *speed-up* and *efficiency*, as shown in (Guarracino and Perla, 1995a). Algorithm 2 has the same numerical properties as 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 matrices among 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 $0 \le I < P$ and $0 \le J < Q$. Each node is equipped with a CPU and local memory. The nodes are connected by some communication network that allows broadcasting 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 the basic operations are level 3 BLAS (Dongarra *et al.*, 1990), and considered connection topology the is a 2-D mesh, we choose the *block scatter decomposition* (Choi *et al.*, 1996) for all matrices involved in the algorithm, including the sparse one, since this strategy allows using ScaLAPACK. For the memorization scheme of sparse A we use a data structure, per process, usually referenced as the *CSC – Compress Sparse Column* (see, e.g., Saad, 1992), consisting of three arrays, containing respectively:

(i) non-zero entries of A columns parts in the subblocks that are assigned to the process;

(ii) rows indices in A of each element in the first array;

(iii) 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 provides faster memory access to data, an easier localization of a whole column and a decrease in global communications.

3.3. Implemented Algorithm. 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 global transposition of the matrix R_j^T is needed at each iteration before evaluating the QR factorization. Since transposition operations in a message passing environment are extremely time consuming, due to the access needed to non-local memories, we substitute the QR factorization by the LQ factorization, which allows us to access the matrix R_j^T without transposition. Then, the implemented version is given as Algorithm 3. It represents the computational kernel of the software, which will be described in detail in the next section.

Algorithm 3: Block Lanczos algorithm (3-rd version).

```
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 $r - 1$
    $R_j^T = X_j^T A - B_{j-1} X_{j-1}^T - M_j X_j^T$
    $\boxed{R_j^T = B_j X_{j+1}^T \quad \textbf{(LQ fact.)}}$
    $M_{j+1} = X_{j+1}^T A X_{j+1}$
  end for
```

3.4. Computational and Communication Complexities. 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.

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

$$(7m \times s^2 + 2nza \times s - 2s^3/3)/p$$

floating-point operations,

$$4m \times s + 2nza$$

one-to-one communications,

$$m \times s + s^2$$

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 by one order of magnitude less than computational complexity, which is generally considered a target in parallel algorithms for linear algebra.

4. Software Description

4.1. Software Architecture. HPEC uses standard message passing libraries, i.e., BLACS (Dongarra and Whaley, 1995) and MPI (Gropp *et al.*, 1999), and standard numerical linear algebra software, PBLAS (Choi *et al.*, 1995) and ScaLAPACK, obtaining software as portable as PARPACK.

A PBLAS routine used in Algorithm 3 to evaluate matrix-matrix multiplications with dense factors is PDGEMM. Routines for matrix factorization are not included in PBLAS, but they are covered by ScaLAPACK. At each iteration, the evaluation of B_j 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 the 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 according to the data layout scheme. Since the sparse factor is not involved in communication, the advantage is that the overhead does not depend on sparsity. On the other hand, the performance depends on the distribution of 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 a reverse communication strategy for the sparse matrix A. The driver routine is named LANCZOS, and its header is the following:

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

    DOUBLE PRECISION   A(*), XT1(LDXT1,*),
1                  W(*), ORFAC, ABSTOL

    INTEGER    S, M, LMA, AI(*), AJ(*),
1                  LDXT1, MB, CONTXT

    INTEGER    NUMSEA, NUMAUT, NMAX,
1                  IFND(*), IIFAIL
```

Table 1. Characteristics of test matrices.

Name	Size	nza	Average nza per col.	Longest (nza) col.	Shortest (nza) col.	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

The user needs to provide the sparse matrix `A` in the block CSC format, an initial block `XT1` consisting of `S` Lanczos vectors, and the required absolute tolerance `ABSTOL`. With respect to other packages, e.g., PARPACK, which require a user supplied matrix-vector product, the HPEC user needs to provide the sparse matrix A either in a data file, or via a function to compute A blocks, for given row and column indexes.

HPEC supports different input sparse matrix formats:

Compress Sparse Column: described in Section 3.2.

Coordinate Storage Scheme: 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 an integer array and a list of the 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 2-D 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 the `PDSDRV1` PARPACK driver on two test matrices. We briefly recall that PARPACK is a parallel version of the ARPACK software and it is targeted for multicomputers. It is written in Fortran 77 and implements the Implicitly Restarted Arnoldi Method for solving large sparse eigenvalue problems. PARPACK uses a single-vector version of the algorithm, thus exploiting parallelism on a matrix-vector product level. `PDSDRV1` needs a parallel sparse matrix-vector routine coherent with the PARPACK internal data distribution. Among the available software, we decided to use the `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 the 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 publicly available software, such as P-SPARSLIB (Saad and Malevsky, 1995) and PSBLAS (Filippone and Colajanni, 2000), 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 the 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 were performed on two test matrices taken from the *Matrix Market* (Boisvert *et al.*, 1997) and the *Test Matrix Toolbox* for Matlab (Higham, 1995). 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. The Test Matrix Toolbox was implemented by N.J. Higham. Not only does it contain test matrices, but it also provides various tools for visualising and generating test problems in Matlab.

The two selected matrices have the sizes, numbers of nonzero entries, sparsity patterns and conditioning properties shown in Table 1. The PLATZ matrix (Cline *et al.*, 1976) is a finite-difference model for 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 one and four processors, respectively, to seek 1, 2, 4 and 10 largest eigenvalues as regards the magnitude of PLATZ1919 with PARPACK and HPEC, with a fixed user tolerance in the order of machine (double) precision for the computed eigenvalues.

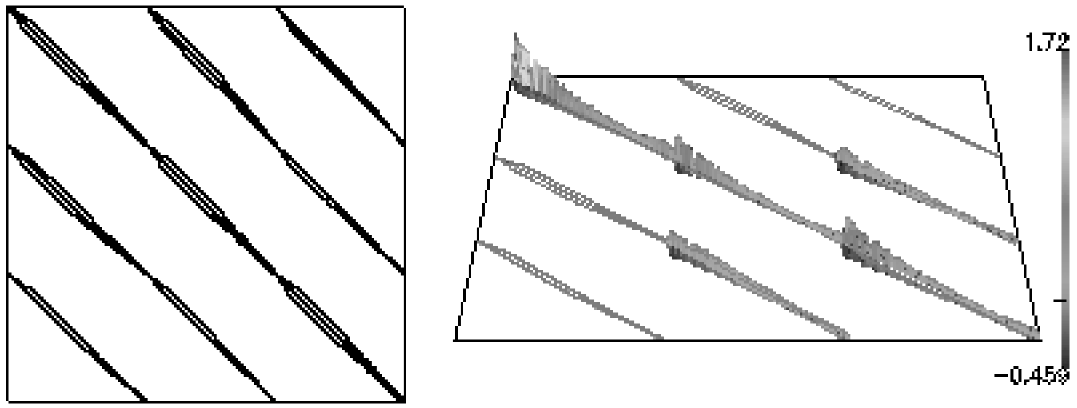

Fig. 1. PLATZ1919 pattern and element magnitudes.

Table 2. Execution times in seconds for the PLATZ1919 matrix on one 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 the 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)

We report the best execution time of PARPACK using 16, 32, 64 and 128 Arnoldi vectors and, in brackets, the number of vectors for which it was obtained. A similar methodology was 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 us to compute multiple eigenvalues at the same time, as can be observed in Table 2.

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

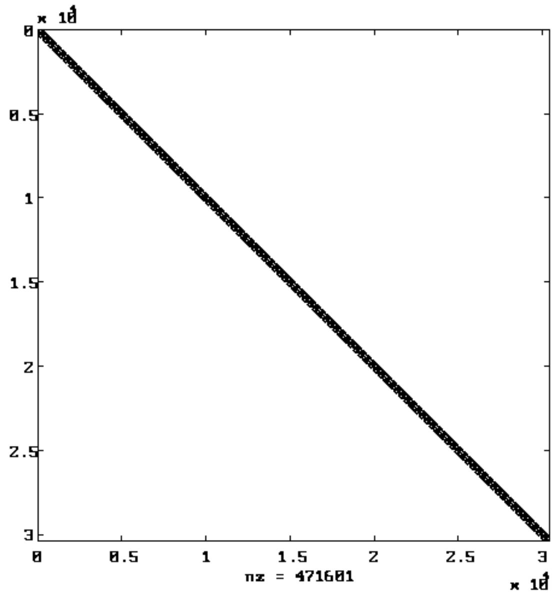

Fig. 2. WATHEN(100,100) nonzero pattern.

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

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

We note the number of Arnoldi and Lanczos vectors to optimize execution time, which 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 the choice to users.

All tests confirm that the algorithm implemented in HPEC preserves the same well-known numerical proper-

Table 4. Execution times in seconds for the 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 the 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)

ties of the block Lanczos algorithm and, in particular, the ability to evaluate multiple eigenvalues, and the capability to evaluate eigenvalues of ill-conditioned problems.

5.1. Parallel Performance Evaluation. In this section we evaluate the performance of parallel implementation, using standard parameters. In particular, we wish to estimate the gain in terms of execution time when an increasing number of processors is used, while fixing 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 were performed on randomly generated matrices of the order $m = 8192, 16384, 32768$, with the 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* (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 a single iteration increases accordingly. For this reason, we choose the 10-th iteration, which provides sufficient granularity for the dense operations to justify the use of multiple processors.

We note that the efficiency values on 2 nodes for all tests never drop below 0.75, while they are at least 0.47 on 4 nodes and at most 0.60 on 8 processors. The efficiency values grow for fixed m and s when nzp increases: this is an expected result, since in the analysis of communication and computation complexities shown in Section 3.4, one-to-all communication does not involve the sparse factor A. Further, efficiency increases for larger values of m, which provides a hint that software can be efficient for a growing number of processors.

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

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

6. Summary

We have presented HPEC, a freely available parallel software based on a variant of the block Lanczos algorithm for real, sparse symmetric eigenvalue problems. The software is based on the linear algebra library ScaLAPACK and the BLACS communication library. It provides a simple application programming interface and supplies decomposition and distribution routines for dense and sparse matrices. The results of numerical experiments and performance evaluation confirm numerical and efficiency qualities of the proposed software.

Acknowledgments

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

References

Baglama J., Calvetti D. and Reichel L. Irbleigs (2003): *A MATLAB program for computing a few eigenpairs of a large sparse Hermitian matrix*. — ACM TOMS, Vol. 29, No. 5, pp. 337–348.

Bai Z. (1994): *Error analysis of the Lanczos algorithm for the nonsymmetric eigenvalues problem*. — Math. Comp., Vol. 62, No. 205, pp. 209–226.

Boisvert R.F., Pozo R., Remington K., Barrett R. and Dongarra J. (1997): *The Matrix Market: A web resource for test matrix collections*, In: Quality of Numerical Software, Assessment and Enhancement (Ronald F. Boisvert, Ed.). — London: Chapman & Hall, pp. 125–137.

Choi J., Demmel J., Dhillon I., Dongarra J., Ostrouchov S., Petitet A., Stanley K., Walker D. and Whaley R.C., ScaLAPACK (1996): *A portable linear algebra library for distributed memory computers: Design and performance*. — Comput. Phys. Comm., Vol. 97, No. 1–2, pp. 1–15.

Choi J., Dongarra J., Ostrouchov S., Petitet A., Walker D. and Whaley R.C. (1995): *A proposal for a set of parallel basic linear algebra subprograms*. — Tech. Rep. UT-CS-95-292, University of Tennessee, Knoxville.

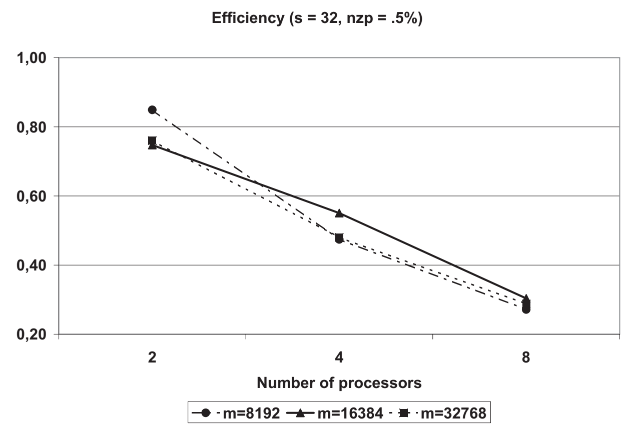

Fig. 3. Efficiency values on $p = 2, 4, 8$ nodes, for $s = 32$, $nzp = .5\%$, $m = 8192, 16384, 32768$.

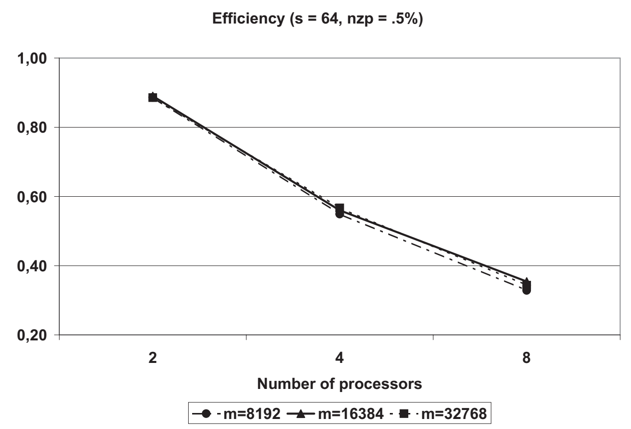

Fig. 4. Efficiency values on $p = 2, 4, 8$ nodes, for $s = 64$, $nzp = .5\%$, $m = 8192, 16384, 32768$.

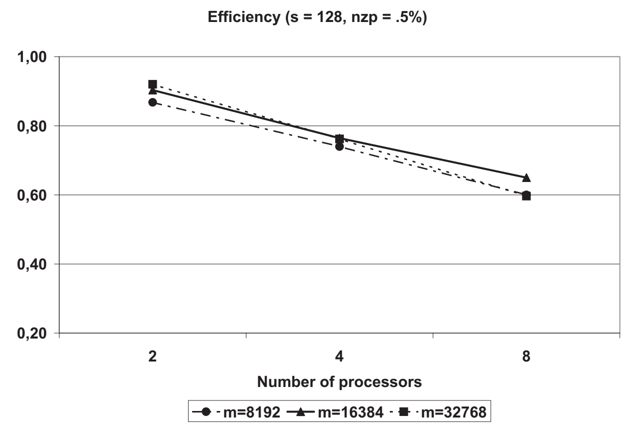

Fig. 5. Efficiency values on $p = 2, 4, 8$ nodes, for $s = 128$, $nzp = .5\%$, $m = 8192, 16384, 32768$.

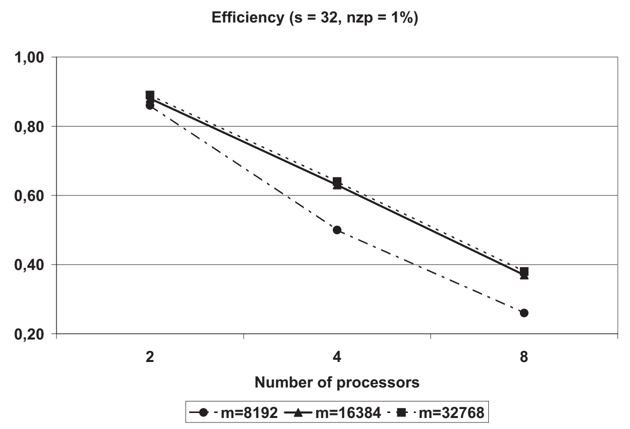

Fig. 6. Efficiency values on $p = 2, 4, 8$ nodes, for $s = 32$, $nzp = 1\%$, $m = 8192, 16384, 32768$.

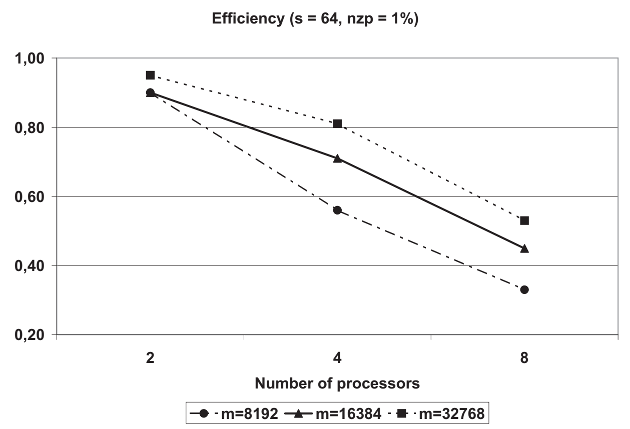

Fig. 7. Efficiency values on $p = 2, 4, 8$ nodes, for $s = 64$, $nzp = 1\%$, $m = 8192, 16384, 32768$.

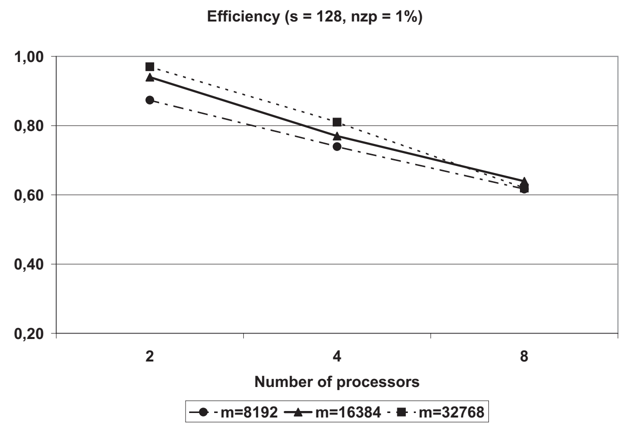

Fig. 8. Efficiency values on $p = 2, 4, 8$ nodes, for $s = 128$, $nzp = 1\%$, $m = 8192, 16384, 32768$.

Cline A.K., Golub G.H. and Platzman G.W. (1976): *Calculations of normal modes of oceans using a Lanczos method*, In: Sparse Matrix Computations (J.R. Bunch and D.J. Rose, Eds.). — London: Academic Press.

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

Dongarra J., Du Croz J., Hammarling S. and Duff I. (1990): *A set of level 3 basic linear algebra subprograms*. — ACM Trans. Math. Software, Vol. 16, No. 1, pp. 1–17.

Dongarra J. and Whaley R.C. (1995): *A user's guide to the BLACS v1.1*. — Tech. Rep. UT-CS-95-281, University of Tennessee, Knoxville.

Filippone S. and Colajanni M. (2000): *PSBLAS: A library for parallel linear algebra computation on sparse matrices*. — ACM Trans. Math. Software, Vol. 26, No. 4, pp. 527–550.

Golub G.H. and Van Loan C.F. (1989): *Matrix Computations, 2nd Ed.* — Baltimore, MD: Johns Hopkins Univ. Press.

Gropp W., Lusk E. and Skjellum A. (1999): *Using MPI — 2nd Edition: Portable Parallel Programming with the Message Passing Interface*. — Cambridge, MA: MIT Press.

Guarracino M.R. and Perla F. (1994): *A parallel block Lanczos algorithm for distributed memory architectures*. — J. Par. Alg. Appl., Vol. 4, No. 1–2, pp. 211–221.

Guarracino M.R. and Perla F. (1995a): *A parallel modified block Lanczos' algorithm for distributed memory architectures*, In: *Proc. Euromicro Workshop* Parallel and Distributed Processing, San Remo, Italy, pp. 424–431.

Guarracino M.R. and Perla F. (1995b): *A sparse, symmetric eigensolver for distributed memory architectures: Parallel implementation and numerical stability*. — Tech. Rep. 11, Center for Research on Parallel Computing and Supercomputers, Naples, Italy.

Hernandez V., Roman J.E. and Vidal V. (2003): *SLEPc: Scalable library for eigenvalue problem computations*. — Lect. Notes Comput. Sci., Vol. 2565, pp. 377–391.

Higham N.J. (1995): *The test matrix toolbox for MATLAB (version 3.0)* — Tech. Rep. Vol. 276, Manchester Centre for Computational Mathematics.

Jones M.T. and Patrick M.L. (1989): *The use of Lanczos's method to solve the large generalized symmetric definite eigenvalue problem*. — Tech. Rep. ICASE 89–67, Langley Research Center.

Lanczos C. (1950): *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. — J. Res. Nat. Bur. Stand., Vol. 45, No. 4, pp. 225–281.

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

Paige C. (1976): *Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix*. — J. Inst. Math. Appl., Vol. 18, No. 3, pp. 341–349.

Parlett B.N. (1980): *The Symmetric Eigenvalues Problem*. — Englewood Cliffs, NJ: Prentice-Hall.

Parlett B.N. and Scott D.S. (1979): *The Lanczos algorithm with selective orthogonalization*. — Math. Comp., Vol. 33, No. 145, pp. 217–238.

Saad Y. (1992): *Numerical Methods for Large Eigenvalues Problems*. — Manchester: Manchester Univ. Press, Halsted Press.

Saad Y. and Malevsky A.V., P-SPARSLIB (1995): *A portable library of distributed memory sparse iterative solvers*, In: Proceedings of Parallel Computings Technologies (V.E. Malyshkin *et. al.*, Eds.) (PaCT-95), 3rd International Conference, St. Petersburg. — Heidelberg: Springer.

Webster F. and Lo G. (1996): *Projective block Lanczos algorithm for dense, Hermitian eigensystems*. — J. Comput. Phys., Vol. 124, No. 1, pp. 146–161.

Wilkinson J.H. (1965): *The Algebraic Eigenvalue Problem*. — Oxford: Clarendon Press.

Wu K. and Simon H. (1999a):*Parallel efficiency of the Lanczos method for eigenvalue problems*. — Tech. Rep. LBNL–42828.

Wu K. and Simon H. (1999b): *TRLAN user guide*. — Tech. Rep. Lawrence Berkeley National Laboratory, LBNL–42953.

Received: 13 April 2005
Revised: 16 November 2005