# A parallel algorithm for the generalized symmetric eigenvalue problem on a transputer network

Kai Rothe

*Institute for Applied Mathematics,*
*University of Hamburg,*
*Federal Republic of Germany*

**abstract**

Conventional routines from standard libraries (e.g. EISPACK) for solving the generalized symmetric eigenvalue problem $Ax = \lambda Bx$ cannot work efficiently on parallel computers with distributed memory because of their sequential conception. The algorithm presented here parallizes the given problem very efficiently on a transputer network.

# 1 Introduction

The discretisation of boundary eigenvalue problems e.g. by finite elements results in a large sparse generalized symmetric matrix eigenvalue problem $Ax = \lambda Bx$ with positive definite $B$. In [9] a substructure concept for the parallel computation of these problems is developed. An additional condensation technique reduces the large system to a *dense* one of moderate size. A partial step in the parallel frame program calculates a small number of eigenpairs in the lower part of the spectrum of this dense system. This part turns out to be the bottle neck since conventional solvers from standard libraries e.g. EISPACK cannot work efficiently on parallel computers with distributed memory because of their sequential conception.

Several parallizing concepts for eigenvalue problems have been developed recently (cf. [4],[6] and [8]). However they are not advantageously applicable to our special problem. The algorithm presented here parallizes the given problem on a ring structure. It should be pointed out that the main feature of an efficient parallel algorithm on distributed memory computers is the data handling. Which means that data restructuring during the run time must be avoided. Therefore in this report we emphasize the data structure.

In a first section we describe the standard algorithm as it is used in the EISPACK library. Next we define the process topology for our parallel version of the standard algorithm. Finally in the main section we describe the program which parallizes the several steps of the standard algorithm by corresponding subroutines. We measure the efficiency with which the parallel version is implemented at each step and collectively. A comparison with the corresponding EISPACK sequence results in a considerable reduction of the whole computational time.

## 2 The standard algorithm

First we briefly review the standard algorithm for solving the dense generalized symmetric $n \times n$ eigenvalue problem (1) with positiv definite $B$ as it is used in the EISPACK library (cf. [3]).

$$Ax = \lambda Bx \tag{1}$$

Problem (1) is transformed first to the standard eigenvalue problem (5). Therefore we compute the Cholesky decomposition (2) of $B$.

$$LL^T := B \tag{2}$$

With the Cholesky decomposition we compute the matrix $\tilde{A}$ in steps (3) and (4). This procedure is described in detail in [7].

$$LC := A \tag{3}$$

$$\tilde{A}L^T := C \tag{4}$$

The reduction (2) - (4) is implemented in the EISPACK routine REDUC and results in the standard form (5) of the eigenvalue problem.

$$\tilde{A}y = \lambda y \tag{5}$$

Next we reduce the standard form (5) via Householder transformations (6) to tridiagonal form (7) which can be done by TRED1.

$$T := H^{(n-2)} \cdots H^{(1)} \tilde{A} H^{(1)} \cdots H^{(n-2)} \tag{6}$$

$$Tz = \lambda z \tag{7}$$

Then the requested eigenvalues of the tridiagonal problem (7) are computed. If we are interested in a small number of eigenvalues e.g. in the lower part of the spectrum we can use here the routine TRIDIB. In some applications the corresponding eigenvectors are required as well. A matching routine for this is TINVIT. At last the eigenvectors are transformed back first to those of the standard problem (5) via (8) which is implemented in TRBAK1

$$y = H^{(1)} \cdots H^{(n-2)} z \tag{8}$$

and second by (9) to the eigenvectors of the original problem (1) which is done by the EISPACK routine REBAK.

$$y = L^T x \tag{9}$$

The following matrices are used as examples in each partial step which is parallized in the presented report.

$$A = \begin{pmatrix} 27.0 & 54.0 & 93.6 & 153.8 & 182.8 & 211.8 & 240.8 & 269.8 & 298.8 \\ 54.0 & 2508.0 & 2827.2 & 3337.6 & 4013.6 & 4329.6 & 4645.6 & 4961.6 & 5277.6 \\ 93.6 & 2827.2 & 10026.6 & 11026.8 & 12463.2 & 13946.4 & 14736.6 & 15526.8 & 16317.0 \\ 153.8 & 3337.6 & 11026.8 & 23445.4 & 25507.3 & 27964.6 & 30340.4 & 31741.2 & 33142.0 \\ 182.8 & 4013.6 & 12463.2 & 25507.3 & 42184.8 & 45310.7 & 48714.6 & 51941.5 & 53996.6 \\ 211.8 & 4329.6 & 13946.4 & 27964.6 & 45310.7 & 64210.2 & 68267.2 & 72444.8 & 76406.4 \\ 240.8 & 4645.6 & 14736.6 & 30340.4 & 48714.6 & 68267.2 & 86956.8 & 91723.6 & 96444.2 \\ 269.8 & 4961.6 & 15526.8 & 31741.2 & 51941.5 & 72444.8 & 91723.6 & 107819.2 & 113026.9 \\ 298.8 & 5277.6 & 16317.0 & 33142.0 & 53996.6 & 76406.4 & 96444.2 & 113026.9 & 124539.0 \end{pmatrix}$$

$$B = \begin{pmatrix} 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 8.0 & 9.0 \\ 2.0 & 104.0 & 116.0 & 128.0 & 140.0 & 152.0 & 164.0 & 176.0 & 188.0 \\ 3.0 & 116.0 & 454.0 & 486.0 & 518.0 & 550.0 & 582.0 & 614.0 & 646.0 \\ 4.0 & 128.0 & 486.0 & 1146.0 & 1206.0 & 1266.0 & 1326.0 & 1386.0 & 1446.0 \\ 5.0 & 140.0 & 518.0 & 1206.0 & 2231.0 & 2326.0 & 2421.0 & 2516.0 & 2611.0 \\ 6.0 & 152.0 & 550.0 & 1266.0 & 2326.0 & 3722.0 & 3858.0 & 3994.0 & 4130.0 \\ 7.0 & 164.0 & 582.0 & 1326.0 & 2421.0 & 3858.0 & 5600.0 & 5782.0 & 5964.0 \\ 8.0 & 176.0 & 614.0 & 1386.0 & 2516.0 & 3994.0 & 5782.0 & 7820.0 & 8052.0 \\ 9.0 & 188.0 & 646.0 & 1446.0 & 2611.0 & 4130.0 & 5964.0 & 8052.0 & 10317.0 \end{pmatrix}$$

$$L = \begin{pmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2.0 & 10.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3.0 & 11.0 & 18.0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4.0 & 12.0 & 19.0 & 25.0 & 0 & 0 & 0 & 0 & 0 \\ 5.0 & 13.0 & 20.0 & 26.0 & 31.0 & 0 & 0 & 0 & 0 \\ 6.0 & 14.0 & 21.0 & 27.0 & 32.0 & 36.0 & 0 & 0 & 0 \\ 7.0 & 15.0 & 22.0 & 28.0 & 33.0 & 37.0 & 40.0 & 0 & 0 \\ 8.0 & 16.0 & 23.0 & 29.0 & 34.0 & 38.0 & 41.0 & 43.0 & 0 \\ 9.0 & 17.0 & 24.0 & 30.0 & 35.0 & 39.0 & 42.0 & 44.0 & 45.0 \end{pmatrix}$$

$$C = \begin{pmatrix} 27.0 & 54.0 & 93.6 & 153.8 & 182.8 & 211.8 & 240.8 & 269.8 & 298.8 \\ 0 & 240.0 & 264.0 & 303.0 & 364.8 & 390.6 & 416.4 & 442.2 & 468.0 \\ 0.7 & 1.4 & 380.1 & 401.8 & 439.0 & 500.8 & 524.1 & 547.4 & 570.7 \\ 1.3 & 8.6 & 10.5 & 462.4 & 482.3 & 516.6 & 576.9 & 598.2 & 619.5 \\ 0 & 12.0 & 22.2 & 23.9 & 490.6 & 507.3 & 536.0 & 591.7 & 609.6 \\ 0 & 0 & 19.8 & 30.9 & 32.4 & 465.9 & 479.4 & 501.5 & 551.2 \\ 0 & 0 & 0 & 25.0 & 35.3 & 36.6 & 397.9 & 408.2 & 423.0 \\ 0 & 0 & 0 & 0 & 27.9 & 36.0 & 37.1 & 296.2 & 303.3 \\ 0 & 0 & 0 & 0 & 0 & 28.8 & 33.6 & 34.5 & 170.4 \end{pmatrix}$$

$$\tilde{A} = \begin{pmatrix} 27.0 & 0 & 0.7 & 1.3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 24.0 & 0 & 0.6 & 1.2 & 0 & .0 & 0 & 0 \\ 0.7 & 0 & 21.0 & 0 & 0.5 & 1.1 & 0 & 0 & 0 \\ 1.3 & 0.6 & 0 & 18.0 & 0 & 0.4 & 1.0 & 0 & 0 \\ 0 & 1.2 & 0.5 & 0 & 15.0 & 0 & 0.3 & 0.9 & 0 \\ 0 & 0 & 1.1 & 0.4 & 0 & 12.0 & 0 & 0.2 & 0.8 \\ 0 & 0 & 0 & 1.0 & 0.3 & 0 & 9.0 & 0 & 0.1 \\ 0 & 0 & 0 & 0 & 0.9 & 0.2 & 0 & 6.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.8 & 0.1 & 0 & 3.0 \end{pmatrix}$$

$$T = \begin{pmatrix} 27.0 & -1.476 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1.476 & 18.674 & 1.855 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.855 & 16.421 & -5.435 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -5.435 & 14.524 & 3.103 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3.103 & 16.846 & -7.327 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -7.327 & 12.955 & -5.337 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -5.337 & 7.898 & 0.935 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.935 & 13.228 & -3.381 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3.381 & 7.455 \end{pmatrix}$$

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| 2.927 | 5.903 | 8.876 | 11.917 | 14.908 | 17.897 | 21.093 | 24.211 | 27.267 |

Table 1: Eigenvalues of the exemplarily treated matrices

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| .35e-3 | .19e-3 | -.79e-2 | .11e-1 | .34e-2 | .14e+0 | .11e+0 | -.44e-1 | .98e+0 |
| .87e-4 | -.66e-2 | -.82e-2 | .22e-2 | -.13e+0 | .91e-1 | .24e-1 | .99e+0 | .29e-1 |
| -.54e-2 | -.57e-2 | -.60e-3 | -.12e+0 | -.77e-1 | -.11e-1 | -.98e+0 | .12e-1 | .11e+0 |
| -.35e-2 | .58e-4 | .11e+0 | -.64e-1 | .99e-2 | -.98e+0 | .35e-1 | .88e-1 | .14e+0 |
| .23e-3 | .10e+0 | .48e-1 | .10e-1 | .98e+0 | .25e-1 | -.77e-1 | .13e+0 | .76e-2 |
| .89e-1 | .33e-1 | -.10e-1 | .99e+0 | -.21e-1 | -.69e-1 | -.12e+0 | .41e-2 | .12e-1 |
| .17e-1 | -.10e-1 | -.99e+0 | -.18e-1 | .52e-1 | -.11e+0 | .99e-3 | .83e-2 | .79e-2 |
| -.58e-2 | -.99e+0 | .14e-1 | .35e-1 | .99e-1 | .71e-3 | -.61e-2 | .65e-2 | .43e-3 |
| -.99e+0 | .86e-2 | -.18e-1 | .88e-1 | -.10e-2 | -.44e-2 | -.52e-2 | .19e-3 | .42e-3 |

Table 2: Corresponding eigenvectors of $\tilde{A}x = \lambda x$ (orthonormalized)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| .13e-1 | .14e-1 | .64e-2 | -.29e-2 | .76e-2 | .15e+0 | .15e+0 | -.25e+0 | .97e+0 |
| .99e-3 | .33e-3 | -.13e-3 | .66e-2 | -.96e-2 | .11e-1 | .63e-1 | .98e-1 | -.43e-2 |
| .76e-4 | -.89e-4 | -.44e-2 | -.44e-2 | -.52e-2 | .41e-1 | -.56e-1 | -.31e-2 | .28e-3 |
| -.34e-4 | -.32e-2 | .30e-2 | -.31e-2 | -.33e-1 | -.40e-1 | .40e-2 | -.86e-3 | .54e-2 |
| -.25e-2 | .24e-2 | .19e-2 | -.28e-1 | .32e-1 | .28e-2 | .91e-3 | .41e-2 | -.94e-4 |
| .21e-2 | .12e-2 | .25e-1 | .28e-1 | -.19e-2 | .89e-3 | -.33e-2 | -.10e-3 | .13e-3 |
| .59e-3 | .23e-1 | -.25e-1 | -.13e-2 | -.11e-2 | -.27e-2 | .17e-3 | .54e-4 | .19e-3 |
| .23e-1 | -.23e-1 | .75e-3 | -.12e-2 | .23e-2 | .12e-3 | -.24e-4 | .15e-3 | .48e-6 |
| -.22e-1 | .19e-3 | -.41e-3 | .20e-2 | -.22e-4 | -.99e-4 | -.12e-3 | .43e-5 | .94e-5 |

Table 3: Corresponding eigenvectors of $Ax = \lambda Bx$ ($B$ orthonormalized)

# 3 Process topology

For the parallel version of the standard algorithm we take a ring of processes with distributed memory as shown in Figure 1 since we want to use already existing and efficiently working algorithms for the Cholesky decomposition [5] and for the Householder reduction [2]. Each process can be identified with a corresponding driver routine named 'slave' respectively 'master' which is responsible for the initialisation. These drivers communicate bidirectionally in the ring by message passing where the input and output streams are shown in Figure 1. The stream numbers can be set as parameters in the driver routine (cf. next paragraph).

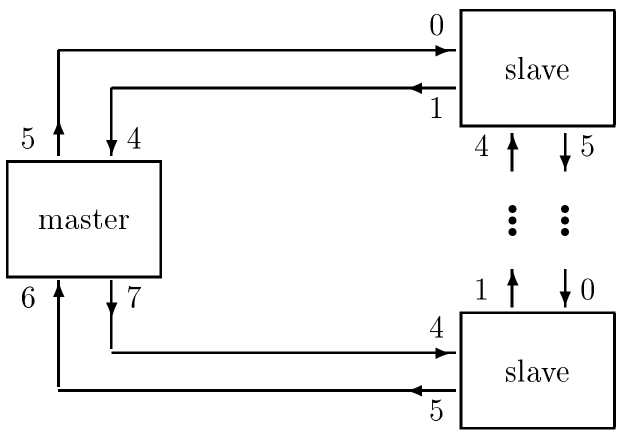


Figure 1: Distributed memory process ring with stream numbers

The 'cdl'-compiler (cf. [1]) which runs under the the distributed memory system 'helios' of the transputer network links these drivers as is shown in the following cdl program.

The task force definition and the command 'streams' assemble the ring structure. The command 'code' appoints the driver code to the process, while 'processor' and 'memory' assign the resources. Finally 'puid' maps the processes to the hardware processor topologie which in our example is a mapping one to one. For large process numbers this point requires an extra investigation.

**cdl-program:** bidirectionally communicating ring (4 processes)

called by: cdl filename

component declaration

```
component ma [ code /helios/users/rothe/fortran/eigenw/master;
               processor T800;
               puid /Cluster/01;
               memory 3000000;
               streams , , , , , , < |d2, > |d1; ]
component s1 [ code /helios/users/rothe/fortran/eigenw/slave;
               processor T800;
               puid /Cluster/02;
               memory 3000000; ]
component s2 [ code /helios/users/rothe/fortran/eigenw/slave;
               processor T800;
               puid /Cluster/00;
               memory 3000000; ]
component s3 [ code /helios/users/rothe/fortran/eigenw/slave;
               processor T800;
               puid /Cluster/03;
               memory 3000000;
               streams , , , , < |d1, > |d2; ]
```

task force definition

```
ma <> s1 <> s2 <> s3
```

# 4 The parallel program

The main feature of an efficient parallel algorithm is the data handling. In our case we start with the wrapped mapping of the columns of $A$ and $B$ which is given in Figure 2. The algorithm then continues with the wrapped mapping of the columns. It should be pointed out, that data restructuring during the run time must be avoided. Therefore the data output of one parallized step has to be the data input for the next. We will emphazise this in the following discussion.

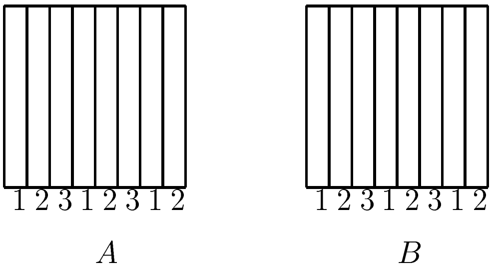


Figure 2: Columnwise wrapped mapping of $A$ and $B$ to $p = 3$ processes

The step by step parallelisation of the standard algorithm is done by a corresponding sequence of subroutines which are coordinated by the driver routines. This means that all drivers simultaneously call the subroutine which parallizes e.g. the Cholesky decomposition. The synchronisation is guaranteed by the synchronous communication commands

PSX_WRITE(STREAM,BUFFER,LENGHT),
PSX_READ(STREAM,BUFFER,LENGHT).

Where STREAM is an integer containing the stream number (cf. Figure 1), BUFFER is a variable containing the data to be sent or received and LENGHT is an integer containing the amount of data (bytes) to be transmitted. In the following sections we discuss the parallel working subroutines in the sequence in which they appear. First however we have a look at the two driver routines 'master' and 'slave'. They differ in the subroutine calls INIT1 respectively INIT2 which initialize the ring and COLLECT1 respectively COLLECT2 which collect the requested eigenpairs in the master process.

In the initializing step the master assigns to each slave its 'id' number and the local parts of the matrices $A$ and $B$ clockwise in ascending order. For technical reasons these local parts constist of $m = \text{int}(n/p) + 1$ columns. Thus one last nonrelevant zero column may occur in some processes. The subroutines INIT1, INIT2 and RINGPARAM are given in the appendix.

```
      PROGRAM MASTER
C     P: number of processes
C     N: dimension of the problem
C     M: maximum number of local columns
C     ANZGES: global number of required eigenvalues
C     I1,O1: input,output stream (clockwise, cf. Figure 1)
C     I2,O2: input,output stream (counter-clockwise)
      IMPLICIT NONE
      CHARACTER*3 DATEI
      INTEGER ID,N,M,P, ANZGES,ANZ,IND, I1,I2,O1,O2,KANAL
     1        ,SPALTEN,L,RIGHT,ALPHA,BETA
      REAL A,AA,B,BB,G,U,X,D,E,E2,T,EV,Z
      PARAMETER(N=52,P=4,ANZGES=5,M=N/P+1)
      PARAMETER(I1=6,O1=5,I2=4,O2=7)
      DIMENSION AA(N,N),A(N,M),BB(N,N),B(N,M),SPALTEN(M),G(N,P)
     1          ,U(N+1),D(N),E(N),E2(N),T(M,2), IND(M),EV(M),Z(N,M)
      ID = 1
      CALL INIT1(ID,P,N,M,AA,A,BB,B,DATEI,KANAL,I1,O1)
      CALL RINGPARAM(ID,P,N,M,L,BETA,RIGHT,SPALTEN)
      OPEN( UNIT=KANAL , FILE=DATEI , STATUS='NEW')
      CALL CHOLESKY(P,N,M,B,G,L,BETA,RIGHT,SPALTEN,I1,O1)
      CALL FORWARD(P,N,M,ID,A,B,G,I2,O2)
      CALL ALTEQC(P,N,M,A,B,G,L,BETA,RIGHT,SPALTEN,I1,O1)
      CALL HOUSEHOLDER(ID,P,N,M,A,T,G,U,L,BETA,RIGHT,SPALTEN
     1                 ,I1,I2,O1,O2)
      CALL EIGENVALUES(ANZGES,ANZ,ID,P,N,M,T,D,E,E2,EV,IND
     1                 ,L,SPALTEN,I2,O2,KANAL)
      CALL EIGENVECTORS(N,D,E,E2,IND,ANZ,EV,Z,KANAL)
      CALL TRANSBAK1(P,N,M,ID,A,G,Z,ANZ,I2,O2)
      CALL TRANSBAK2(P,N,M,ID,SPALTEN,L,B,G,Z,ANZGES,ANZ
     1                ,I2,O2,KANAL)
      CALL COLLECT1(ANZGES,ID,P,N,M,E,EV,U,Z,I2,O2,KANAL)
      CLOSE( UNIT=KANAL , STATUS='KEEP')
      STOP
      END
```

```
      PROGRAM SLAVE

C     P: number of processes
C     N: dimension of the problem
C     M: maximum number of local columns
C     ANZGES: global number of required eigenvalues
C     I1,O1: input,output stream (clockwise, cf. Figure 1)
C     I2,O2: input,output stream (counter-clockwise)
      IMPLICIT NONE
      CHARACTER*3 DATEI
      INTEGER ID,N,M,P, ANZGES,ANZ,IND, I1,I2,O1,O2,KANAL
     1        ,SPALTEN,L,RIGHT,ALPHA,BETA
      REAL A,B,G,U,X,D,E,E2,T,EV,Z
      PARAMETER(N=52,P=4,ANZGES=5,M=N/P+1)
      PARAMETER(I1=0,O1=5,I2=4,O2=1)
      DIMENSION A(N,M),B(N,M),SPALTEN(M),G(N,P),U(N+1)
     1          ,D(N),E(N),E2(N),T(M,2), IND(M),EV(M),Z(N,M)
      ID = 1
      CALL INIT2(ID,P,N,M,A,B,DATEI,KANAL,I1,O1)
      CALL RINGPARAM(ID,P,N,M,L,BETA,RIGHT,SPALTEN)
      OPEN( UNIT=KANAL , FILE=DATEI , STATUS='NEW')
      CALL CHOLESKY(P,N,M,B,G,L,BETA,RIGHT,SPALTEN,I1,O1)
      CALL FORWARD(P,N,M,ID,A,B,G,I2,O2)
      CALL ALTEQC(P,N,M,A,B,G,L,BETA,RIGHT,SPALTEN,I1,O1)
      CALL HOUSEHOLDER(ID,P,N,M,A,T,G,U,L,BETA,RIGHT,SPALTEN
     1                 ,I1,I2,O1,O2)
      CALL EIGENVALUES(ANZGES,ANZ,ID,P,N,M,T,D,E,E2,EV,IND
     1                 ,L,SPALTEN,I2,O2,KANAL)
      CALL EIGENVECTORS(N,D,E,E2,IND,ANZ,EV,Z,KANAL)
      CALL TRANSBAK1(P,N,M,ID,A,G,Z,ANZ,I2,O2)
      CALL TRANSBAK2(P,N,M,ID,SPALTEN,L,B,G,Z,ANZGES,ANZ
     1                ,I2,O2,KANAL)
      CALL COLLECT2(ANZGES,ID,P,N,M,EV,Z,I2,O2)
      CLOSE( UNIT=KANAL , STATUS='KEEP')
      STOP
      END
```

## 4.1 Cholesky decomposition

For the Cholesky decomposition (2) we choose an algorithm that is recommded by Golub and van Loan [5]. We start here with the columnwise wrapped mapping of $B$ which is the data input for CHOLESKY. The data output given by $L$ is of the kind visualized for $p = 3$ processes in Figure 3.

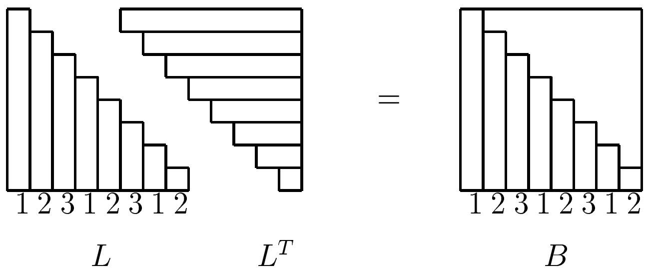


Figure 3: Cholesky decomposition wrapped mapping of $L$ and $B$ for $p = 3$ processes

**mode of working of CHOLESKY:**

Process 1 (the master) computes $\ell_1$ the first column of the Cholesky factor $L$. The only necessary information for that is the first column of $B$ which is part of its local memory. Then process 1 sends $\ell_1$ clockwise through the ring to its right neighbor (process 2, a slave). Afterwards it dates up the rest of its part of $B$ with $\ell_1$. Process 2 receives $\ell_1$ and sends it to its right neighbor if it is not process 1. Then it dates up its local part of $B$ with $\ell_1$. After that it computes $\ell_2$ the second column of $L$, the necessary information is the second (updated) column of $B$ which is part of its local memory. Then it sends $\ell_2$ to its right neighbor and continues with updating the rest of its part of $B$ with $\ell_2$. The processes 3 to $p$ proceed in the same way. The algorithm stops if the last column of $L$ which consists only of the element $L_{n,n}$ is computed.

| processor | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | 2 | | | 3 | | | 4 | | |
| 1.0 | 5.0 | 9.0 | 2.0 | 6.0 | 0 | 3.0 | 7.0 | 0 | 4.0 | 8.0 | 0 |
| 2.0 | 140.0 | 188.0 | 10.0 | 152.0 | 0 | 116.0 | 164.0 | 0 | 128.0 | 176.0 | 0 |
| 3.0 | 518.0 | 646.0 | 11.0 | 550.0 | 0 | 18.0 | 582.0 | 0 | 486.0 | 614.0 | 0 |
| 4.0 | 1206.0 | 1446.0 | 12.0 | 1266.0 | 0 | 19.0 | 1326.0 | 0 | 25.0 | 1386.0 | 0 |
| 5.0 | 31.0 | 2611.0 | 13.0 | 2326.0 | 0 | 20.0 | 2421.0 | 0 | 26.0 | 2516.0 | 0 |
| 6.0 | 32.0 | 4130.0 | 14.0 | 36.0 | 0 | 21.0 | 3858.0 | 0 | 27.0 | 3994.0 | 0 |
| 7.0 | 33.0 | 5964.0 | 15.0 | 37.0 | 0 | 22.0 | 40.0 | 0 | 28.0 | 5782.0 | 0 |
| 8.0 | 34.0 | 8052.0 | 16.0 | 38.0 | 0 | 23.0 | 41.0 | 0 | 29.0 | 43.0 | 0 |
| 9.0 | 35.0 | 45.0 | 17.0 | 39.0 | 0 | 24.0 | 42.0 | 0 | 30.0 | 44.0 | 0 |

Table 4: Local memory of $B$ after CHOLESKY

After termination the full lower triangle of $B$ is overwritten by $L$ and the strict upper triangle of $B$ remains unchanged. Thus the local memorys of the processes contain $L$ in the same columnwise wrapped fashion as previously $B$. This is demonstrated in Table 4 for our exemplarily treated matrices and four processes.

The technique of receiving and immediately sending the column $\ell_i$ in connection with the wrapped mapping of the matrix leads to a good load balancing and thus we get a satisfying efficiency as shown in Table 5. Another fact that influences the efficiency is the access to the memory. Depending on the hardware the access to smaller arrays leads to a superlinear speedup. That is why we get an efficiency of 100 per cent for $n = 400$ and $p = 2$. The sequential computational effort of this step is about 10 per cent of the total sequential effort ($n \geq 100$) which can be interpreted as weighting factor for the lateron summarized efficiency in Table 13.

| $n\backslash p$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 100 | 100.0 | 88.4 | 69.0 | 57.1 |
| 200 | 100.0 | 97.7 | 89.0 | 82.2 |
| 300 | 100.0 | 99.5 | 95.1 | 90.9 |
| 400 | 100.0 | 100.0 | 97.8 | 95.2 |

Table 5: Efficiency of CHOLESKY in per cent

```
      SUBROUTINE CHOLESKY(P,N,M,B,G,L,BETA,RIGHT,SPALTEN,IN,OUT)
C     P: number of processes
C     N: dimension of the problem
C     M: maximum number of local columns
C     B: array with local part of B
C     G: temporary storage array
C     L: number of local colummns
C     BETA: global index of last column of right neighbor process
C     RIGHT: ID of right neighbor process
C     SPALTEN: array with global indices of local columns
C     IN: input stream (cf. Figure 1)
C     OUT: output stream
      IMPLICIT NONE
      INTEGER I,J,K, N,M,P, Q,R, SPALTEN,L,RIGHT,BETA, ALPHA, IN,OUT
      REAL B,G,X
      DIMENSION B(N,M),SPALTEN(M),G(N,P)
```

```
      Q=1
      DO 40 J=1,N
         IF(J .EQ. SPALTEN(Q)) THEN
            X=B(J,Q)
            X=SQRT(X)
            B(J,Q)=X
            DO 10 I=J+1,N
               B(I,Q)=B(I,Q)/X
10          CONTINUE
            IF(J .LT. N) THEN
               CALL PSX_WRITE( OUT , B(J+1,Q) , 4*(N-J))
            ENDIF
            DO 20 K=Q+1,L
               R=SPALTEN(K)
               X=B(R,Q)
               DO 20 I=R,N
                  B(I,K) = B(I,K) - X*B(I,Q)
20          CONTINUE
            Q=Q+1
         ELSE
            IF(Q .LE. L) THEN
               CALL PSX_READ( IN , G(J+1,1) , 4*(N-J))
               ALPHA=MOD(J,P)
               IF(ALPHA .EQ. 0) ALPHA=P
               IF( (RIGHT .NE. ALPHA) .AND. (J .LT. BETA) ) THEN
                  CALL PSX_WRITE( OUT , G(J+1,1) , 4*(N-J))
               ENDIF
               DO 30 K=Q,L
                  R=SPALTEN(K)
                  X=G(R,1)
                  DO 30 I=R,N
                     B(I,K) = B(I,K) - X*G(I,1)
30             CONTINUE
            ENDIF
         ENDIF
40    CONTINUE
      RETURN
      END
```

## 4.2 Reduktion to standard form

We use the Cholesky decomposition to solve the forward substitution (3). The data input $A$ and $L$ are given in a columnwise wrapped mapping which has to be used without restructuring. At the end of FORWARD we get $C$ in the same columnwise wrapped fashion as shown in Figure 4.

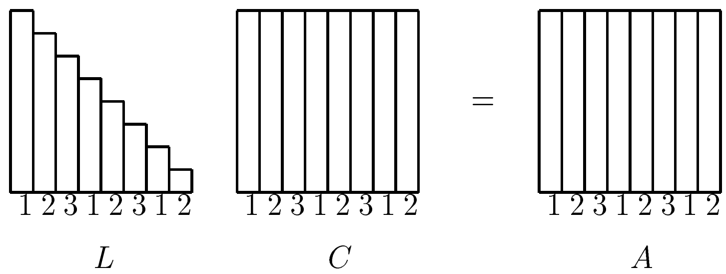


Figure 4: Forward substitution and wrapped mapping of $L, C$ and $A$ for $p = 3$ processes

**mode of working of FORWARD:**

First the processes rotate the local columns $\ell_1, \cdots, \ell_p$ counter-clockwise through the ring in $p - 1$ cycles. At the end each process holds the full information $\ell_1, \cdots, \ell_p$ with which it computes fully in parallel the first $p$ forward substitution steps of (3) on its local part of $A$. Then $\ell_{p+1}, \cdots, \ell_{2p}$ are rotated through the ring and the next $p$ forward substitution steps are performed.

At the end of FORWARD the matrix $A$ is overwritten by the unsymmetric matrix $C$, as demonstrated for the exemplarily treated matrices and four processes in Table 6.

| processor | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | 2 | | | 3 | | | 4 | | |
| 27.0 | 182.8 | 298.8 | 54.0 | 211.8 | 0 | 93.6 | 240.8 | 0 | 153.8 | 269.8 | 0 |
| 0 | 364.8 | 468.0 | 240.0 | 390.6 | 0 | 264.0 | 416.4 | 0 | 303.0 | 442.2 | 0 |
| 0.7 | 439.0 | 570.7 | 1.4 | 500.8 | 0 | 380.1 | 524.1 | 0 | 401.8 | 547.4 | 0 |
| 1.3 | 482.3 | 619.5 | 8.6 | 516.6 | 0 | 10.5 | 576.9 | 0 | 462.4 | 598.2 | 0 |
| 0 | 490.6 | 609.6 | 12.0 | 507.3 | 0 | 22.2 | 536.0 | 0 | 23.9 | 591.7 | 0 |
| 0 | 32.4 | 551.2 | 0 | 465.9 | 0 | 19.8 | 479.4 | 0 | 30.9 | 501.5 | 0 |
| 0 | 35.3 | 423.0 | 0 | 36.6 | 0 | 0 | 397.9 | 0 | 25.0 | 408.2 | 0 |
| 0 | 27.9 | 303.3 | 0 | 36.0 | 0 | 0 | 37.1 | 0 | 0 | 296.2 | 0 |
| 0 | 0 | 170.4 | 0 | 28.8 | 0 | 0 | 33.6 | 0 | 0 | 34.5 | 0 |

Table 6: Local memory of $A$ after FORWARD

The sequential computational effort of this step is about 40 per cent of the total sequential effort ($n \geq 100$) which means that the parallel version of this step has significant influence on the summarized efficiency in Table 13. Table 7 shows the satisfying degree of parallelisation.

| $n\backslash p$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 100 | 100.0 | 93.5 | 87.4 | 77.0 |
| 200 | 100.0 | 97.7 | 96.1 | 91.1 |
| 300 | 100.0 | 98.7 | 97.3 | 95.0 |
| 400 | 100.0 | 99.0 | 98.4 | 96.6 |

Table 7: Efficiency of FORWARD in per cent

At this step we unfortunately have to compute the full matrix $C$ although we only need the lower triangle of $C$. Computing only the lower triangle of $C$ on the other hand necessitates tremendous costs of communication since the relevant information is not suitably distributed. So we get a loss of performance relative to the sequential algorithm from EISPACK.

```
      SUBROUTINE FORWARD(P,N,M,ID,A,B,G,IN,OUT)
C     P: number of processes
C     N: dimension of the problem
C     M: maximum number of local columns
C     ID: identity number of the calling process
C     A: array with local part of A
C     B: array with local part of L
C     G: temporary storage array
C     IN: input stream (cf. Figure 1)
C     OUT: output stream
      IMPLICIT NONE
      INTEGER H,HJ,I,J,K, P,N,M,ID, Q,Q1, IN,OUT
      REAL A,B,G
      DIMENSION A(N,M),B(N,M),G(N,P)
```

```
      DO 40 J=1,M
         HJ=(J-1)*P+1
         DO 10 I=HJ,N
            G(I,ID)=B(I,J)
10       CONTINUE
         DO 20 I=1,P-1
            Q=ID+I
            Q1=Q-1
            IF(Q .GT. P) Q=Q-P
            IF(Q1 .GT. P) Q1=Q1-P
            IF(ID .EQ. 1) THEN
               CALL PSX_WRITE( OUT, G(HJ,Q1), 4*(N-HJ+1))
               CALL PSX_READ( IN , G(HJ,Q) , 4*(N-HJ+1))
            ELSE
               CALL PSX_READ( IN , G(HJ,Q) , 4*(N-HJ+1))
               CALL PSX_WRITE( OUT, G(HJ,Q1), 4*(N-HJ+1))
            ENDIF
20       CONTINUE
         DO 40 H=1,P
            HJ=(J-1)*P+H
            IF(HJ.LE.N) THEN
               DO 30 K=1,M
                  A(HJ,K)=A(HJ,K)/G(HJ,H)
                  DO 30 I=HJ+1,N
                     A(I,K)=A(I,K) - A(HJ,K)*G(I,H)
30             CONTINUE
            ENDIF
40    CONTINUE
      RETURN
      END
```

Next we have to solve equation (4) with the data structure from the preceding routines as input namely the columnwise wrapped mapping of $C$ and $L$. As output we get $\tilde{A}$ in the same columnwise wrapped fashion as it is shown in Figure 5.

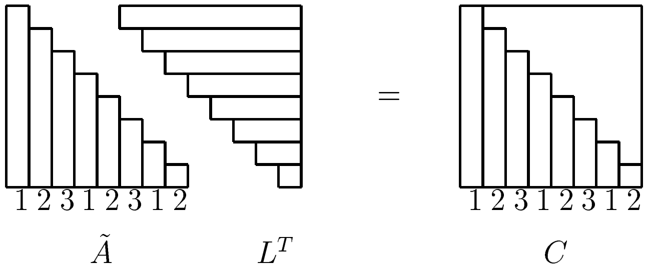


Figure 5: ALTEQC and wrapped mapping of $\tilde{A}$, $L^T$ and $C$ for $p = 3$ processes

ALTEQC works in the same way as CHOLESKY with the slight difference that at each communication step two vectors are involved namely $\tilde{a}_i$ the $i$-th actually computed column of $\tilde{A}$ and $\ell_i$ as before. At the end of this step the full lower triangle of $C$ which was originally the full lower triangle of $A$ is overwritten by $\tilde{A}$. We demonstrate this for our exemplarily treated matrices and four processes in Table 8.

| processor | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | 2 | | | 3 | | | 4 | | |
| 27.0 | 182.8 | 298.8 | 54.0 | 211.8 | 0 | 93.6 | 240.8 | 0 | 153.8 | 269.8 | 0 |
| 0 | 364.8 | 468.0 | 24.0 | 390.6 | 0 | 264.0 | 416.4 | 0 | 303.0 | 442.2 | 0 |
| 0.7 | 439.0 | 570.7 | 0 | 500.8 | 0 | 21.0 | 524.1 | 0 | 401.8 | 547.4 | 0 |
| 1.3 | 482.3 | 619.5 | 0.6 | 516.6 | 0 | 0 | 576.9 | 0 | 18.0 | 598.2 | 0 |
| 0 | 15.0 | 609.6 | 1.2 | 507.3 | 0 | 0.5 | 536.0 | 0 | 0 | 591.7 | 0 |
| 0 | 0 | 551.2 | 0 | 12.0 | 0 | 1.1 | 479.4 | 0 | 0.4 | 501.5 | 0 |
| 0 | 0.3 | 423.0 | 0 | 0 | 0 | 0 | 9.0 | 0 | 1.0 | 408.2 | 0 |
| 0 | 0.9 | 303.3 | 0 | 0.2 | 0 | 0 | 0 | 0 | 0 | 6.0 | 0 |
| 0 | 0 | 3.0 | 0 | 0.8 | 0 | 0 | 0.1 | 0 | 0 | 0 | 0 |

Table 8: Local memory of $A$ after ALTEQC

The sequential computational effort of this step is about 10 per cent of the total sequential effort ($n \geq 100$). The efficiency given in Table 9 reflects in comparison with Table 5 the influence of the increased communication of ALTEQC.

| $n\backslash p$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 100 | 100.0 | 78.5 | 57.1 | 46.2 |
| 200 | 100.0 | 90.6 | 77.9 | 70.0 |
| 300 | 100.0 | 95.0 | 86.5 | 80.9 |
| 400 | 100.0 | 96.3 | 90.5 | 86.2 |

Table 9: Efficiency of ALTEQC in per cent

```
      SUBROUTINE ALTEQC(P,N,M,A,B,G,L,BETA,RIGHT,SPALTEN,IN,OUT)
C     P: number of processes
C     N: dimension of the problem
C     M: maximum number of local columns
C     A: array with local part of C
C     B: array with local part of L
C     G: temporary storage array
C     L: number of local colummns
C     BETA: global index of last column of the right neighbor process
C     RIGHT: ID of right neighbor process
C     SPALTEN: array with global indices of columns
C     IN: input stream (cf. Figure 1)
C     OUT: output stream
```

```
      IMPLICIT NONE
      INTEGER I,J,K, N,M,P, Q,R, SPALTEN,L,RIGHT,BETA, ALPHA, IN,OUT
      REAL A,B,G,X
      DIMENSION A(N,M),B(N,M),G(N,P),SPALTEN(M)
      Q=1
      DO 50 J=1,N
         IF(J .EQ. SPALTEN(Q)) THEN
            X=B(J,Q)
            DO 10 I=J,N
               A(I,Q)=A(I,Q)/X
10          CONTINUE
            IF(J .LT. N) THEN
               DO 20 I=J+1,N
                  G(I,1)=A(I,Q)
                  G(I,2)=B(I,Q)
20             CONTINUE
               CALL PSX_WRITE( OUT , G(J+1,1) , 4*(2*N-J))
            ENDIF
            DO 30 K=Q+1,L
               R=SPALTEN(K)
               X=B(R,Q)
               DO 30 I=R,N
                  A(I,K) = A(I,K) - X*A(I,Q)
30          CONTINUE
            Q=Q+1
         ELSE
            IF(Q .LE. L) THEN
               CALL PSX_READ( IN , G(J+1,1) , 4*(2*N-J))
               ALPHA=MOD(J,P)
               IF(ALPHA .EQ. 0) ALPHA=P
               IF( (RIGHT .NE. ALPHA) .AND. (J .LT. BETA) ) THEN
                  CALL PSX_WRITE( OUT , G(J+1,1) , 4*(2*N-J))
               ENDIF
               DO 40 K=Q,L
                  R=SPALTEN(K)
                  X=G(R,2)
                  DO 40 I=R,N
                     A(I,K) = A(I,K) - X*G(I,1)
40             CONTINUE
            ENDIF
         ENDIF
50    CONTINUE
      RETURN
      END
```

## 4.3 Reduction to tridiagonal form

For reduction to the tridiagonal form (6) we specialize the more general formulated algorithm that is recommended by Dongarra and van de Geijn [2] to our ring topologie. Before describing the the parallel algorithm we review the mathematical background.

The matrix $\tilde{A}^{(0)} := \tilde{A}$ is reduced via $n-2$ Householder transformations

$$\tilde{A}^{(k)} := H^{(k)}\tilde{A}^{(k-1)}H^{(k)} \qquad k = 1, \cdots, n-2$$

to tridiagonal form $T := \tilde{A}^{(n-2)}$. The Householdermatrix in step $k$ is of the form $H^{(k)} := I - \beta u u^T$ with

$$\begin{aligned} \eta &= ||(0, \cdots, 0, \tilde{A}^{(k-1)}_{k+1,k}, \cdots, \tilde{A}^{(k-1)}_{n,k})^T||_2 \\ u &= (0, \cdots, 0, \tilde{A}^{(k-1)}_{k+1,k} \pm \eta, \tilde{A}^{(k-1)}_{k+2,k}, \cdots, \tilde{A}^{(k-1)}_{n,k})^T \\ \beta &= 2/u^T u, \end{aligned}$$

where we suppress the index $k$ as far as possible in this first description. With these notations we are able to compute $\tilde{A}^{(k)} = \tilde{A}^{(k-1)} - \beta u v^T - \beta v u^T$ with

$$\begin{aligned} y &:= \tilde{A}^{(k-1)}u \\ v &:= y - \frac{\beta}{2}u^T y u. \end{aligned}$$

It should be pointed out here that reduction to tridiagonal form is not unique. Our reductional process starts with the transformation of the first column of $\tilde{A}$ and chooses the Householder matrix which avoids numerical instability. The input for HOUSEHOLDER is the full lower triangle of $\tilde{A}$ which is stored in a columnwise wrapped fashion. On output $T$ stored in the same way as it is shown in Figure 6.

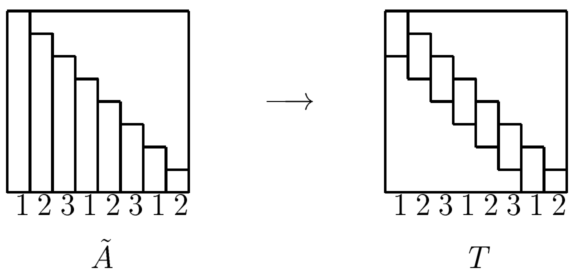


Figure 6: Householder reduction and wrapped mapping of $\tilde{A}$ and $T$ for $p = 3$ processes

**mode of working of HOUSEHOLDER:**

The process which houses the $k$-th global column of $\tilde{A}^{(k-1)}$ in its local memory computes $u$ and $\beta$ and sends it clockwise through the ring. After that each process computes its partial sum of $y = \tilde{A}^{(k-1)}u$ fully in parallel. The algorithm continues by summing up these partial sums counter-clockwise in $p-1$ cycles. Then each process computes $v$ and dates up its local part of $\tilde{A}^{(k-1)}$ fully in parallel.

After termination the first $n-2$ columns of the lower triangle of $A$ are overwritten by the Householder information $\beta^{(1)}, u^{(1)}, \cdots, \beta^{(n-2)}, u^{(n-2)}$ while the diagonal and subdiagonal elements of $T$ are stored on a seperate array. For the exemplarily treated matrices and four processes the contents of $A$ is given in Table 10.

| processor | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | 2 | | | 3 | | | 4 | | |
| 0.459 | 182.800 | 298.800 | 54.000 | 211.800 | 0 | 93.600 | 240.800 | 0 | 153.800 | 269.800 | 0 |
| 1.476 | 364.800 | 468.000 | 0.199 | 390.600 | 0 | 264.000 | 416.400 | 0 | 303.000 | 442.200 | 0 |
| 0.700 | 439.000 | 570.700 | -2.707 | 500.800 | 0 | 0.023 | 524.100 | 0 | 401.800 | 547.400 | 0 |
| 1.300 | 482.300 | 619.500 | 1.059 | 516.600 | 0 | 8.051 | 576.900 | 0 | 0.100 | 598.200 | 0 |
| 0 | 0.011 | 609.600 | -0.237 | 507.300 | 0 | -0.475 | 536.000 | 0 | -3.217 | 591.700 | 0 |
| 0 | 12.132 | 551.200 | -0.874 | 0.028 | 0 | 2.014 | 479.400 | 0 | -0.304 | 501.500 | 0 |
| 0 | 3.394 | 423.000 | -0.880 | 6.651 | 0 | 4.265 | 0.795 | 0 | 2.771 | 408.200 | 0 |
| 0 | 0.592 | 303.300 | 0 | 0.219 | 0 | -0.209 | -1.347 | 0 | -0.343 | 13.228 | 0 |
| 0 | 4.327 | 7.455 | 0 | 5.169 | 0 | -0.424 | -0.839 | 0 | -1.314 | -3.381 | 0 |

Table 10: Local memory of $A$ after HOUSEHOLDER

The sequential computational effort of this step costs about 38 per cent of the total sequential effort ($n \geq 100$) which again requires a good degree of parallelization. How satisfying HOUSEHOLDER parallizes here is shown in Table 11.

| $n \backslash p$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 100 | 100.0 | 86.9 | 69.3 | 54.0 |
| 200 | 100.0 | 94.5 | 86.2 | 76.1 |
| 300 | 100.0 | 96.7 | 91.6 | 84.6 |
| 400 | 100.0 | 97.6 | 93.4 | 88.9 |

Table 11: Efficiency of HOUSEHOLDER in per cent

```
      SUBROUTINE HOUSEHOLDER(ID,P,N,M,A,T,G,U,L,BETA,RIGHT
     1                                ,SPALTEN,I1,I2,O1,O2)
C     ID: identity number of the calling process
C     P: number of processes
C     N: dimension of the problem
C     M: maximum number of local columns
C     A: array with local part of Ã
C     T: array for T
C     G,U: temporay storage arrays
C     L: number of local colummns
C     BETA: global index of last column of right neighbor process
C     RIGHT: ID of right neighbor process
C     SPALTEN: array with global indices of columns
C     I1,I2: input streams (cf. Figure 1)
C     O1,O2: output streams
      IMPLICIT NONE
      INTEGER I,J,K, N,M,P, Q,R, SPALTEN,L,RIGHT,BETA, ALPHA
     1                ,I1,I2,O1,O2,Q1,ID
      REAL A,T,G,U,ETA,X,Z,RELABW,EPS
      PARAMETER(EPS=1.0E-6)
      DIMENSION A(N,M),T(M,4),G(N,P),SPALTEN(M),U(N+1)
      Q=1
      DO 130 J=1,N-2
C      computing the Householdermatrix H^(j)
         IF(J .EQ. SPALTEN(Q)) THEN
            DO 10 I=J,N+1
               U(I)=0.0
10          CONTINUE
            ETA=0.0
            DO 20 I=J+1,N
               ETA=ETA+A(I,Q)*A(I,Q)
20          CONTINUE
            X=ABS( A(J+1,Q) )
            RELABW=0.0
            IF(ETA .GT. 1.0E-20) THEN
               RELABW=ABS( X-ETA ) / ETA
            ENDIF
            IF( (ETA .GT. 1.0E-20) .AND. (RELABW .GT. EPS) ) THEN
               ETA=SQRT(ETA)
               U(J+1)=A(J+1,Q)+SIGN(1.0,A(J+1,Q))*ETA
               U(N+1)=U(J+1)*U(J+1)
```

```
            DO 30 I=J+2,N
               U(I) = A(I,Q)
               U(N+1)= U(N+1)+U(I)*U(I)
30          CONTINUE
            U(N+1)=2.0/U(N+1)
         ENDIF
         CALL PSX_WRITE(O1, U(J+1) , 4*(N+1-J))
      ELSE
         IF(Q .LE. L) THEN
            CALL PSX_READ (I1 , U(J+1) , 4*(N+1-J))
            ALPHA=MOD(J,P)
            IF(ALPHA .EQ. 0) ALPHA=P
            IF( (RIGHT .NE. ALPHA) .AND. (J .LT. BETA) ) THEN
               CALL PSX_WRITE(O1, U(J+1) , 4*(N+1-J))
            ENDIF
         ENDIF
      ENDIF
C     computing the partial sums of y = Au
      Q1=Q
      DO 60 K=J,N
         G(K,ID)=0.0
         DO 40 I=Q,Q1-1
            G(K,ID)=G(K,ID)+A(K,I)*U(SPALTEN(I))
40       CONTINUE
         IF(K .EQ. SPALTEN(Q1)) THEN
            DO 50 I=K,N
               G(K,ID)=G(K,ID)+A(I,Q1)*U(I)
50          CONTINUE
            Q1=Q1+1
         ENDIF
60    CONTINUE
C     summing up the partial sums of y
      DO 70 K=1,P-1
         R=ID+K
         Q1=R-1
         IF(R .GT. P) R=R-P
         IF(Q1 .GT. P) Q1=Q1-P
         IF(ID .EQ. 1) THEN
            CALL PSX_WRITE(O2, G(J,Q1) , 4*(N+1-J))
            CALL PSX_READ (I2, G(J,R) , 4*(N+1-J))
         ELSE
            CALL PSX_READ (I2, G(J,R) , 4*(N+1-J))
            CALL PSX_WRITE(O2, G(J,Q1) , 4*(N+1-J))
         ENDIF
70    CONTINUE
```

```
          DO 80 I=2,P
             DO 80 K=J,N
                G(K,1)=G(K,1)+G(K,I)
80        CONTINUE
C       computing v = y - β/2u^T yu
          X=0.0
          DO 90 I=J+1,N
             X=X+U(I)*G(I,1)
90        CONTINUE
          X=U(N+1)*X/2.0
          DO 100 I=J+1,N
             G(I,1)=G(I,1)-X*U(I)
100       CONTINUE
C       updating of the remaining columns A = A - βuv^T - βvu^T
          DO 110 K=Q,L
             R=SPALTEN(K)
             X=U(N+1)*G(R,1)
             Z=U(N+1)*U(R)
             DO 110 I=R,N
                A(I,K)=A(I,K)-X*U(I)-Z*G(I,1)
110       CONTINUE
C       storing T and β,u on A
          IF(J.EQ.SPALTEN(Q)) THEN
             T(Q,1)=A(J,Q)
             T(Q,2)=A(J+1,Q)
             A(J,Q)=U(N+1)
             DO 120 I=J+1,N
                A(I,Q)=U(I)
120          CONTINUE
             Q=Q+1
          ENDIF
C       end of outer loop
130     CONTINUE
C       storing the rest of T
        IF(N-1.EQ.SPALTEN(L)) T(L,1)=A(N-1,L)
        IF(N-1.EQ.SPALTEN(L)) T(L,2)=A(N ,L)
        IF(N .EQ.SPALTEN(L)) T(L,1)=A(N ,L)
        RETURN
        END
```

## 4.4 Computing the eigenvalues

Now we are able to compute the (e.g. $s$ smallest) eigenvalues of problem (7). The columns of $T$ are stored in a wrapped fashion in the local memory at the end of HOUSEHOLDER, so first each process receives the full information by rotating the columns of $T$ in $p-1$ cycles through the ring. Then each process computes $\text{int}(s/p)$ respectively $\text{int}(s/p)+1$ eigenvalues specified by their indices fully in parallel by the EISPACK routine TRIDIB and writes the result to an output file.

The efficiency of EIGENVALUES is not exactly 100 per cent because of the occuring communication costs for spreading $T$ through the ring. Furthermore generally not all processes compute exactly the same number of eigenvalues. In the case of the exemplarily treated matrices four processes compute all eigenvalues in the way shown in Table 12.

| process | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | | | 2 | | 3 | | 4 | |
| 2.927 | 5.903 | 8.876 | 11.917 | 14.908 | 17.897 | 21.093 | 24.211 | 27.267 |

Table 12: Eigenvalue computation of the processors

Another point that influences the efficiency is the required accuracy of the eigenvalues. The higher the required accuracy is the higher are the computational costs and the more unimportant are the communication costs. In Table 13 we show the efficiency of the first five steps of the presented parallel algorithm for computing the twentyfive smallest eigenvalues. If one wants to achieve an efficiency for small $p$ of more than 90 per cent one has to choose $n \approx 100 * p$.

| $n\backslash p$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 100 | 100.0 | 88.7 | 74.0 | 60.8 |
| 200 | 100.0 | 95.7 | 89.3 | 81.5 |
| 300 | 100.0 | 97.6 | 93.6 | 88.8 |
| 400 | 100.0 | 98.3 | 95.7 | 92.2 |

Table 13: Efficiency of CHOLESKY - EIGENVALUES in per cent (25 eigenvalues)

In Table 14 we show the speedup from CHOLESKY to EIGENVALUES compared with the recommended EISPACK sequence REDUC, TRED1 and TRIDIB. We remark that the efficiency decreases to about 60 per cent. This fact is due to the loss of performance which occurs with the parallelisation of FORWARD. Nevertheless the total computational time can be reduced considerably by the presented parallel algorithm as the speedups show.

| $n\backslash p$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 100 | 0.658 | 1.168 | 1.461 | 1.601 |
| 200 | 0.646 | 1.237 | 1.731 | 2.078 |
| 300 | 0.643 | 1.255 | 1.805 | 2.283 |
| 400 | 0.642 | 1.262 | 1.844 | 2.369 |

Table 14: Speedup relative to EISPACK (25 eigenvalues)

```
      SUBROUTINE EIGENVALUES(ANZGES,ANZ,ID,P,N,M,T,D,E,E2,EV
     1                         ,IND,L,SPALTEN,IN,OUT,KANAL)
C     ANZGES: global number s of required eigenvalues
C     ANZ: local number of eigenvalues
C     ID: identity number of the calling process
C     P: number of processes
C     N: dimension of the problem
C     M: maximum number of local columns
C     T: array for T
C     D: array for diagonal elements of T
C     E: array for subdiagonal elements of T
C     E2,IND: arrays for TRIDIB
C     EV: array for eigenvalues
C     L: number of local columns
C     SPALTEN: array with global indices of columns
C     IN: input stream (cf. Figure 1)
C     OUT: output stream
C     KANAL: stream for output file
      IMPLICIT NONE
      INTEGER I,J, L,N,GN,M,M11,P,Q,R, ANZGES,ANZ,ID,IERR,IND
     1            ,SPALTEN,IN,OUT,KANAL
      REAL T,D,E,E2,EPS,EV,LB,UB,RV4,RV5
      PARAMETER(EPS=1.0E-3,GN=700)
      DIMENSION D(N),E(N),E2(N),EV(M),IND(M)
     1            ,T(M,2),SPALTEN(M),RV4(GN),RV5(GN)
      DO 10 I=1,L
         R= SPALTEN(I)
         D(R) = T(I,1)
         IF(R+1.LE.N) E(R+1) = T(I,2)
10    CONTINUE
```

```
      DO 20 I=1,P-1
         CALL PSX_WRITE( OUT, T(1,1) , 8*M )
         CALL PSX_READ ( IN, T(1,1) , 8*M )
         Q=ID+I
         IF(Q .GT. P) Q=Q-P
         DO 200 J=1,M
            R=(J-1)*P+Q
            IF(R .LE. N) D(R )=T(J,1)
            IF(R+1 .LE. N) E(R+1)=T(J,2)
20    CONTINUE
      ANZ=ANZGES/P
      R=MOD(ANZGES,P)
      IF( ID .LE. R) THEN
         ANZ=ANZ+1
         M11=(ID-1)*ANZ+1
      ELSE
         M11=(ID-1)*ANZ+R+1
      ENDIF
      DO 30 I=1,N
         E2(I)=E(I)*E(I)
30    CONTINUE
      CALL TRIDIB(N,EPS,D,E,E2,LB,UB,M11,ANZ,EV,IND,IERR,RV4,RV5)
      WRITE(KANAL,1100)
      DO 40 I=1,ANZ
         WRITE(KANAL , 1000) M11-1+I,EV(I)
40    CONTINUE
1000  FORMAT (I3,1000E13.6 )
1100  FORMAT ('Nr. Eigenwerte ')
      RETURN
      END
```

## 4.5 Computing the eigenvectors

After the determination of the $s$ smallest eigenvalues each process fully in parallel computes the corresponding eigenvectors of the tridiagonal system (7) by EIGENVECTORS which uses in our case the EISPACK routine TINVIT.

The efficiency of this step is less than 100 per cent if and only if the processes have to compute a different number of eigenvectors. Table 15 demonstrates for our exemplarily treated matrices how the eigenvectors are computed.

| processor | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | | | 2 | | 3 | | 4 | |
| (2.927) | (5.903) | (8.876) | (11.917) | (14.908) | (17.897) | (21.093) | (24.211) | (27.267) |
| .35e-3 | .19e-3 | -.79e-2 | .11e-1 | .34e-2 | .14e+0 | .11e+0 | -.44e-1 | .98e+0 |
| .57e-2 | .26e-2 | -.97e-1 | .11e+0 | .28e-1 | .87e+0 | .43e+0 | -.83e-1 | -.18e+0 |
| -.48e-1 | -.18e-1 | .51e+0 | -.41e+0 | -.54e-1 | -.25e+0 | .65e+0 | -.28e+0 | -.40e-1 |
| -.12e+0 | -.34e-1 | .67e+0 | -.30e+0 | -.55e-2 | .36e+0 | -.41e+0 | .38e+0 | .19e-1 |
| .35e+0 | .63e-1 | -.33e+0 | -.46e+0 | -.95e-1 | -.44e-1 | .27e+0 | .68e+0 | .97e-2 |
| .62e+0 | .80e-1 | -.78e-1 | -.44e+0 | -.27e-1 | .16e+0 | -.33e+0 | -.53e+0 | -.55e-2 |
| .68e+0 | .19e-1 | .40e+0 | .55e+0 | .14e+0 | -.88e-1 | .14e+0 | .17e+0 | .15e-2 |
| -.82e-1 | .41e+0 | -.30e-1 | -.13e+0 | .90e+0 | -.23e-1 | .18e-1 | .16e-1 | .11e-3 |
| -.61e-1 | .90e+0 | .71e-1 | .10e+0 | -.41e+0 | .74e-2 | -.45e-2 | -.32e-2 | -.18e-4 |

Table 15: Eigenvectors computed by EIGENVECTORS (orthonormalized)

Computing the eigenvectors requires an extra step since in some applications only the eigenvalues are of interest and thus the algorithm stops with the routine EIGENVALUES.

```
      SUBROUTINE EIGENVECTORS(N,D,E,E2,IND,ANZ,EV,Z,KANAL)
C     N: dimension of the problem
C     D: array for diagonal elements of T
C     E: array for subdiagonal elements of T
C     E2,IND: input arrays for TINVIT from TRIDIB
C     ANZ: local number of required eigenvalues
C     EV: array for eigenvalues
C     Z: array for eigenvectors
C     KANAL: stream for output file
      IMPLICIT NONE
      INTEGER I,J,N,GN ,IND,ANZ, IERR, KANAL
      REAL D,E,E2,EV,Z, RV1,RV2,RV3,RV4,RV6
      PARAMETER(GN=700)
      DIMENSION D(N),E(N),E2(N),IND(ANZ),EV(ANZ),Z(N,ANZ)
     1          ,RV1(GN),RV2(GN),RV3(GN),RV4(GN),RV6(GN)
      CALL TINVIT(N,N,D,E,E2,ANZ,EV,IND,Z,IERR,RV1,RV2,RV3,RV4,RV6)
      WRITE(KANAL,1000) IERR
1000  FORMAT (' Eigenvektoren: IERR = ',I5 )
      RETURN
      END
```

## 4.6 Backtransformation of the eigenvectors

First the eigenvectors of $T$ have to be transformed back via the stored Householder matrices to those of the standard form (5) by equation (8). For this step we need the pairs $\beta^{(k)}$, $u^{(k)}$ which have been stored in the lower triangle of $A$ during the procedure HOUSEHOLDER as shown in Figure 7.

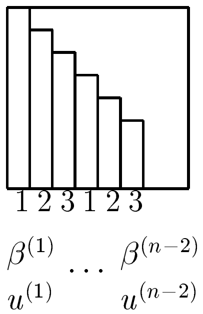


Figure 7: Wrapped mapping of $\beta^{(k)}$, $u^{(k)}$ in $A$ for $p = 3$ processes

**mode of working of TRANSBAK1:**

In the first step of the subroutine the processes rotate the local columns with $(\beta^{(n-2)}, u^{(n-2)})$, $\cdots$, $(\beta^{(n-p-1)}, u^{(n-p-1)})$ counter clockwise through the ring in $p-1$ cycles. At the end each process holds the full information of $H^{(n-2)}, \cdots, H^{(n-p-1)}$ with which it transforms back via (8) fully in parallel its set of eigenvectors. Then $(\beta^{(n-p-2)}, u^{(n-p-2)})$, $\cdots$, $(\beta^{(n-2p-1)}, u^{(n-2p-1)})$ are rotated through the ring and the next $p$ Householder backtransformation steps are performed. After the backtransformation the eigenvectors of (7) are overwritten by those of (5). Table 16 demonstrates the result for our exemplarily treated matrices.

| processor | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | | | 2 | | 3 | | 4 | |
| (2.927) | (5.903) | (8.876) | (11.917) | (14.908) | (17.897) | (21.093) | (24.211) | (27.267) |
| .35e-3 | .19e-3 | -.79e-2 | .11e-1 | .34e-2 | .14e+0 | .11e+0 | -.44e-1 | .98e+0 |
| .87e-4 | -.66e-2 | -.82e-2 | .22e-2 | -.13e+0 | .91e-1 | .24e-1 | .99e+0 | .29e-1 |
| -.54e-2 | -.57e-2 | -.60e-3 | -.12e+0 | -.77e-1 | -.11e-1 | -.98e+0 | .12e-1 | .11e+0 |
| -.35e-2 | .58e-4 | .11e+0 | -.64e-1 | .99e-2 | -.98e+0 | .35e-1 | .88e-1 | .14e+0 |
| .23e-3 | .10e+0 | .48e-1 | .10e-1 | .98e+0 | .25e-1 | -.77e-1 | .13e+0 | .76e-2 |
| .89e-1 | .33e-1 | -.10e-1 | .99e+0 | -.21e-1 | -.69e-1 | -.12e+0 | .41e-2 | .12e-1 |
| .17e-1 | -.10e-1 | -.99e+0 | -.18e-1 | .52e-1 | -.11e+0 | .99e-3 | .83e-2 | .79e-2 |
| -.58e-2 | -.99e+0 | .14e-1 | .35e-1 | .99e-1 | .71e-3 | -.61e-2 | .65e-2 | .43e-3 |
| -.99e+0 | .86e-2 | -.18e-1 | .88e-1 | -.99e-3 | -.44e-2 | -.52e-2 | .19e-3 | .42e-3 |

Table 16: Eigenvectors after TRANSBAK1 (orthonormalized)

```
      SUBROUTINE TRANSBAK1(P,N,M,ID,A,G,Z,ANZ,IN,OUT)
C     P: number of processes
C     N: dimension of the problem
C     M: maximum number of local columns
C     ID: identity number of the calling process
C     A: array with Householder matrix information
C     G: temporary storage array
C     Z: array with eigenvectors
C     ANZ: number of eigenvectors
C     IN,OUT:input, output stream (cf. Figure 1)
      IMPLICIT NONE
      INTEGER H,I,J,K, P,N,M,ID, Q,Q1,R, IN,OUT, ANZ, AN,LA
      REAL A,G,Z,SUM
      DIMENSION A(N,M),G(N,P),Z(N,M)
      R=N+P-MOD(N,P)
      DO 50 K=M,1,-1
         AN=MIN(N,(K-1)*P+1)
         LA=MAX(0,N-(K-1)*P)
         DO 10 I=AN,N
            G(I,ID)=A(I,K)
10       CONTINUE
         DO 20 I=1,P-1
            Q=ID+I
            Q1=Q-1
            IF(Q .GT. P) Q=Q-P
            IF(Q1 .GT. P) Q1=Q1-P
            IF(ID .EQ. 1) THEN
               CALL PSX_WRITE( OUT, G(AN,Q1), 4*LA)
               CALL PSX_READ( IN , G(AN,Q) , 4*LA)
            ELSE
               CALL PSX_READ( IN , G(AN,Q) , 4*LA)
               CALL PSX_WRITE( OUT, G(AN,Q1), 4*LA)
            ENDIF
20       CONTINUE
         DO 50 I=P,1,-1
            IF( (1.LE.R) .AND. (R.LE.(N-2)) ) THEN
               DO 40 H=1,ANZ
                  SUM=0.0
                  DO 30 J=R+1,N
                     SUM=SUM+G(J,I)*Z(J,H)
30                CONTINUE
                  DO 40 J=R+1,N
                     Z(J,H)=Z(J,H)-SUM*G(R,I)*G(J,I)
40             CONTINUE
            ENDIF
            R=R-1
50    CONTINUE
      RETURN
      END
```

In a second step the eigenvectors of a process $y_1, \cdots, y_k$ of $\tilde{A}$ have to be transformed back to those of the original problem (1) $x_1, \cdots, x_k$ by equation (9). We again need the Cholesky factor $L$ which has been stored in the lower triangle of $B$ during the procedure CHOLESKY .

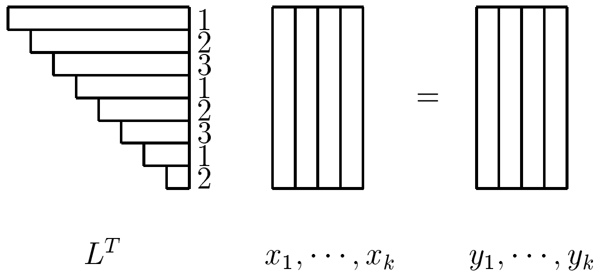


Figure 8: TRANSBAK2 and wrapped mapping of $L^T$ for $p = 3$ processes

**mode of working of TRANSBAK2:**

First the processes rotate the local columns with $\ell_n, \cdots, \ell_{n-p+1}$ counter-clockwise through the ring in $p-1$ cycles. At the end each process holds the full information of $\ell_n, \cdots, \ell_{n-p+1}$ with which it transforms back its set of eigenvectors via (9) fully in parallel. Then $\ell_{n-p}, \cdots, \ell_{n-2p+1}$ are rotated through the ring and the next $p$ backward substitution steps are performed. After the backtransformation the eigenvectors of (5) are overwritten by those of (1). Table 17 demonstrates the result for our exemplarily treated matrices.

| processor | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | | | 2 | | 3 | | 4 | |
| (2.927) | (5.903) | (8.876) | (11.917) | (14.908) | (17.897) | (21.093) | (24.211) | (27.267) |
| .13e-1 | .14e-1 | .64e-2 | -.29e-2 | .76e-2 | .15e+0 | .15e+0 | -.25e+0 | .97e+0 |
| .99e-3 | .33e-3 | -.13e-3 | .66e-2 | -.96e-2 | .11e-1 | .63e-1 | .98e-1 | -.43e-2 |
| .76e-4 | -.89e-4 | -.44e-2 | -.44e-2 | -.52e-2 | .41e-1 | -.56e-1 | -.31e-2 | .28e-3 |
| -.34e-4 | -.32e-2 | .30e-2 | -.31e-2 | -.33e-1 | -.40e-1 | .40e-2 | -.86e-3 | .54e-2 |
| -.25e-2 | .24e-2 | .19e-2 | -.28e-1 | .32e-1 | .28e-2 | .91e-3 | .41e-2 | -.94e-4 |
| .21e-2 | .12e-2 | .25e-1 | .28e-1 | -.19e-2 | .89e-3 | -.33e-2 | -.10e-3 | .13e-3 |
| .59e-3 | .23e-1 | -.25e-1 | -.13e-2 | -.11e-2 | -.27e-2 | .17e-3 | .54e-4 | .19e-3 |
| .23e-1 | -.23e-1 | .75e-3 | -.12e-2 | .23e-2 | .12e-3 | -.24e-4 | .15e-3 | .48e-6 |
| -.22e-1 | .19e-3 | -.41e-3 | .20e-2 | -.22e-4 | -.99e-4 | -.12e-3 | .43e-5 | .94e-5 |

Table 17: Eigenvectors after TRANSBAK2 ($B$ orthonormalized)

```
      SUBROUTINE TRANSBAK2(P,N,M,ID,B,G,Z,ANZGES,ANZ,IN,OUT,KANAL)
C     P: number of processes
C     N: dimension of the problem
C     M: maximum number of local columns
C     ID: identity number of the calling process
C     SPALTEN: array with global indices of columns
C     L: number of local columns
C     B: local part of L
C     Z: array with eigenvectors
C     ANZGES: global number of eigenvectors
C     ANZ: local number of eigenvectors
C     IN: input stream (cf. Figure 1)
C     OUT: output stream
C     KANAL: stream for output file
      IMPLICIT NONE
      INTEGER H,I,J,K, P,N,M,ID, Q,Q1,R, IN,OUT, ANZGES,ANZ ,KANAL,AN,LA
      REAL B,G,Z
      DIMENSION B(N,M),G(N,P),Z(N,M)
      R=N+P-MOD(N,P)
      DO 50 K=M,1,-1
         AN=MIN(N,(K-1)*P+1)
         LA=MAX(0,N-(K-1)*P)
         DO 10 I=AN,N
            G(I,ID)=B(I,K)
10       CONTINUE
         DO 20 I=1,P-1
            Q=ID+I
            Q1=Q-1
            IF(Q .GT. P) Q=Q-P
            IF(Q1 .GT. P) Q1=Q1-P
            IF(ID .EQ. 1) THEN
               CALL PSX_WRITE( OUT, G(AN,Q1), 4*LA)
               CALL PSX_READ( IN , G(AN,Q) , 4*LA)
            ELSE
               CALL PSX_READ( IN , G(AN,Q) , 4*LA)
               CALL PSX_WRITE( OUT, G(AN,Q1), 4*LA)
            ENDIF
20       CONTINUE
```

```
          DO 50 I=P,1,-1
             IF((1.LE.R).AND.(R.LE.N)) THEN
                DO 40 H=1,ANZ
                   DO 30 J=N,R+1,-1
                      Z(R,H)=Z(R,H)-G(J,I)*Z(J,H)
30                 CONTINUE
                   Z(R,H)=Z(R,H)/G(R,I)
40              CONTINUE
             ENDIF
             R=R-1
50        CONTINUE
          R=MOD(ANZGES,P)
          IF(ID .LE. R) Q=(ID-1)*ANZ
          IF(ID .GT. R) Q=(ID-1)*ANZ+R
          WRITE(KANAL,1100)
          DO 60 K=1,ANZ
             WRITE(KANAL,1000) Q+K,(Z(I,K),I=1,N)
60        CONTINUE
1000      FORMAT (I3,1000E9.3 )
1100      FORMAT( ' Eigenvektoren' )
          RETURN
          END
```

If the eigenvectors are orthonormal at the end of EIGENVECTORS which is the case if we use the EISPACK routine TINVIT the output of TRANSBAK1 is orthonormal and the output of TRANSBAK2 is $B$ orthonormal.

The backtransformation has no remarkable influence on the total computational time, since we only compute a small number of eigenpairs in the lower part of the spectrum where only $O(n^2)$ operations are needed instead of the previous $O(n^3)$ processes. Thus it is not necessary to examine the efficiency of TRANSBAK1 and TRANSBAK2.

## 4.7 Collection of the eigenpairs

In some applications, it is not sufficient to obtain the eigenpairs in a distributed form as is the case at the end of TRANSBAK2 but collected in the master process eg. if the parallel algorithm is linked as subroutine to a global frame program. COLLECT1 respectively COLLECT2 are corresponding subroutines for the master respectively slave process.

**mode of working of COLLECT1/2:**

The eigenpairs are sent counter-clockwise through the ring. The master collects them on special arrays while getting the information of the slaves in ascending order from the slave with $id = 2$. Each slave sends its eigenpairs counter-clockwise to its left neighbor and passes on the information of slaves with higher $id$'s. After the collection the master houses in its arrays E and U the eigenvalues and the corresponding eigenvectors which is demonstrated for our exemplarily treated matrices in Table 1 and 3.

```
      SUBROUTINE COLLECT2(ANZGES,ID,P,N,M,EV,Z,IN,OUT)
C     ANZGES: global number s of required eigenvalues
C     ID: identity number of the calling process
C     P: number of processes
C     N: dimension of the problem
C     M: maximum number of local columns
C     EV: array for local computed eigenvalues (input)
C     Z: array for local computed eigenvectors (input)
C     IN: input stream (cf. Figure 1)
C     OUT: output stream
      IMPLICIT NONE
      INTEGER ID,K,N,M,P,R, ANZGES, MAXL ,IN,OUT
      REAL EV,Z
      DIMENSION EV(M),Z(N,M)
      MAXL=ANZGES/P
      R=MOD(ANZGES,P)
      IF( R .GT. 0) MAXL=MAXL+1
      DO 10 K=ID,P
         CALL PSX_WRITE ( OUT, EV(1) , 4*MAXL )
         CALL PSX_WRITE ( OUT, Z(1,1) , 4*MAXL*N )
         IF(K.LT.P) THEN
            CALL PSX_READ ( IN, EV(1) , 4*MAXL )
            CALL PSX_READ ( IN, Z(1,1) , 4*MAXL*N )
         ENDIF
   10 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE COLLECT1(ANZGES,ID,P,N,M,E,EV,U,Z,IN,OUT,KANAL)
C     ANZGES: global number s of required eigenvalues
C     ID: identity number of the calling process
C     P: number of processes
C     N: dimension of the problem
C     M: maximum number of local columns
C     E: array for all computed eigenvalues (output)
C     EV: array for local computed eigenvalues (input)
C     U: array for all computed eigenvectors (output)
C     Z: array for local computed eigenvectors (input)
C     IN: input stream (cf. Figure 1)
C     OUT: output stream
C     KANAL: stream for output file
      IMPLICIT NONE
      INTEGER I,J,K, N,M,P, R,ANZGES,ANZ,ID, SUM,MAXL, IN,OUT,KANAL
      REAL EV,Z,E,U
      DIMENSION EV(M),Z(N,M),E(N),U(N,N)
      MAXL=ANZGES/P
      R=MOD(ANZGES,P)
      IF( R .GT. 0) MAXL=MAXL+1
      SUM=0
      DO 20 K=1,P
         ANZ=ANZGES/P
         IF( K .LE. R ) ANZ=ANZ+1
         DO 10 I=1,ANZ
            E(SUM+I)=EV(I)
            DO 10 J=1,N
               U(J,SUM+I)=Z(J,I)
10       CONTINUE
         SUM=SUM+ANZ
         IF( K.LT.P ) THEN
            CALL PSX_READ ( IN, EV(1) , 4*MAXL )
            CALL PSX_READ ( IN, Z(1,1) , 4*MAXL*N )
         ENDIF
20    CONTINUE
      RETURN
      END
```

# 5 Appendix

```
      SUBROUTINE INIT1(ID,P,N,M,AA,A,BB,B,DATEI,KANAL,I1,O1)
C     ID: identity number of the calling process
C     P: number of processes
C     N: dimension of the problem
C     M: maximum number of local columns
C     AA: matrix A
C     A: local part of A (columnwise wrapped mapping)
C     BB: matrix B
C     B: local part of B (columnwise wrapped mapping)
C     DATEI: name of output file
C     KANAL: stream for output file
C     I1: input stream (cf. Figure 1)
C     O1: output stream
      IMPLICIT NONE
      CHARACTER*3 DATEI
      INTEGER ID,P,N,M,I1,O1,KANAL, I,J,K,R,PP
      REAL AA,A,BB,B
      DIMENSION AA(N,N),A(N,M),BB(N,N),B(N,M)
      DATEI='pa'
      KANAL=10
C     initialisation of the ring
      CALL PSX_WRITE( O1, ID , 4)
      CALL PSX_READ( I1, PP , 4)
C     option
      CALL STARTMAT(N,AA,BB)
C     distribution of the matrices A and B
      DO 30 K=P,1,-1
         DO 20 J=1,M
            R=P*(J-1)+K
            IF(R .LE. N) THEN
               DO 10 I=1,N
                  A(I,J)=AA(I,R)
                  B(I,J)=BB(I,R)
 10            CONTINUE
            ENDIF
 20      CONTINUE
         IF(K .GT. 1) THEN
            CALL PSX_WRITE( O1, A(1,1) , 4*M*N)
            CALL PSX_WRITE( O1, B(1,1) , 4*M*N)
         ENDIF
 30   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE INIT2(ID,P,N,M,A,B,DATEI,KANAL,I1,O1)
C     ID: identity number of the calling process
C     P: number of processes
C     N: dimension of the problem
C     M: maximum number of local columns
C     A: local part of A (columnwise wrapped mapping)
C     B: local part of B (columnwise wrapped mapping)
C     DATEI: name of output file
C     KANAL: stream for output file
C     I1: input stream (cf. Figure 1)
C     O1: output stream
      IMPLICIT NONE
      CHARACTER*3 DATEI,IDDAT
      INTEGER ID,P,N,M,I1,O1,KANAL, I
      REAL A,B
      DIMENSION A(N,M),B(N,M)
C     initialisation of the ring
      CALL PSX_READ( I1, ID , 4)
      ID = ID + 1
      CALL PSX_WRITE(O1, ID , 4)
      IDDAT=CHAR(ID+96)
      DATEI='p'//IDDAT
      KANAL=10+ID
C     distribution of the matrices A and B
      DO 10 I=1,P-1
         IF(I .LE. P-ID+1) THEN
            CALL PSX_READ( I1, A(1,1) , 4*M*N)
            CALL PSX_READ( I1, B(1,1) , 4*M*N)
         ENDIF
         IF(I .LE. P-ID) THEN
            CALL PSX_WRITE(O1, A(1,1) , 4*M*N)
            CALL PSX_WRITE(O1, B(1,1) , 4*M*N)
         ENDIF
   10 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE RINGPARAM(ID,P,N,M,L,BETA,RIGHT,SPALTEN)
C     L: number of local colummns
C     BETA: global index of last column of right neighbor process
C     RIGHT: ID of right neighbor process
C     SPALTEN: array with global indices of columns
      IMPLICIT NONE
      INTEGER I,J, ID,P,N,M, L,BETA,RIGHT,SPALTEN
      DIMENSION SPALTEN(M)
      L=0
      DO 100 J=1,M
        I=(J-1)*P+ID
        IF(I .LE. N) THEN
          SPALTEN(J)=I
          L=L+1
        ENDIF
  100 CONTINUE
      IF(ID .NE. P) THEN
        RIGHT=ID+1
      ELSE
        RIGHT=1
      ENDIF
      IF( SPALTEN(L) .EQ. N) THEN
        BETA=SPALTEN(L)+1-P
      ELSE
        BETA=SPALTEN(L)+1
      ENDIF
      RETURN
      END
```

# References


[1] **Distributed Software Limited**
The cdl guide, Bristol 1990

[2] **J.J. Dongarra and R.A. van de Geijn**
Reduction to condensed form for the Eigenvalue problem on distributed memory architectures, Parallel Computing 18 (1992) 973-982

[3] **B.S. Garbow, J.M. Boyle, J.J. Dongarra and C.B. Moler**
Matrix Eigensystem Routines-EISPACK Guide Extension, Lecture Notes in Computer Science, volume 51, Springer, New York, 1977

[4] **G.A. Geist and G.J. Davis**
Finding eigenvalues and eigenvectors of unsymmetric matrices using a distributed memory multiprocessor, Parallel Computing 13 (1990) 199-209

[5] **G.H. Golub and C.F. van Loan**
Matrix Computations, The John Hopkins University Press, Baltimore and London, 1989

[6] **T.Z. Kalamboukis**
A parallel algorithm for the dense symmetric eigenvalue problem on a transputer array, Parallel Computing 18 (1992) 207-212

[7] **R.S. Martin and J.H. Wilkinson**
Reduction of the symmetric eigenproblem $Ax = \lambda Bx$ and related problemsto the standard form, in: J.H. Wilkinson and C. Reinsch, eds., Linear Algebra, Springer, New York, 1971

[8] **R. Natarajan**
A parallel algorithm for the generalized symmetric eigenvalue problem on a hybrid multiprocessor, Parallel Computing 14 (1990) 129-150

[9] **K. Rothe and H. Voss**
Improving dynamic substructuring on distributed memory MIMD-Computers, in C.A. Brebbia and H. Power (eds.): Applications of Supercomputers in Engineering III, Computational Mechanics Publications, Southhampton, Boston and Elsevier Science Publishers, London, New York, 1993, 339-352