

ABCLib Working Notes No.8

ABCLib_DRSSSED

Manual version 1.00

October 2004

Takahiro KATAGIRI

Department of Information Network Science
Graduate School of Information Systems
The University of Electro-Communications

"Information Infrastructure and Application",
PRESTO, Japan Science and Technology Agency

Foreword

This manual describes the usage of the Dense Real Symmetric Standard Eigenvalue Decomposition (*ABCLib_DRSSSED*) developed as part of the Automatically Blocking and Communication-adjustment Library (ABCLib) project [B-4].

ABCLib is a linear mathematics library that achieves high performance in parallel-processor computers having hierarchical memory as processing elements (PEs). Specifically, ABCLib is a parallel-computing library designed to deliver high performance in PC clusters, in which inexpensive PCs are networked as PEs, as well as on supercomputers.

The ABCLib project is developing this library under the following guiding principles:

- Active use of blocking algorithms in order to achieve high-performance numerical calculation in computers having hierarchical memory.
- Active use of communications encapsulation algorithms, in order to achieve high performance in slow networking environments.
- Inclusion of auto-tuning features to support a wide range of parallel-processing environments, from PC clusters to supercomputers.
- The inclusion of auto-tuning features makes the library easy to use, by vastly reducing the number of user-specified parameters over previous libraries. In addition, the auto-tuning features are implemented in such a way that having fewer parameters does not degrade performance.
- Directives are provided to support additional auto tuning, for developers of mathematics libraries with auto-tuning features.
- Benchmarking software using the auto-tuning framework is provided in order to evaluate the performance of the compiler, system, etc.

The following linear-calculation functions are planned to be added to the ABCLib.

- For dense matrices:
 - Direct solution of simultaneous linear equations
 - Direct solution of eigenvalue problems
 - Iterative solution of eigenvalue problems
- For sparse matrices:
 - Direct solution of simultaneous linear equations

- Iterative solution of simultaneous linear equations
- Iterative solution of eigenvalue problems

The following features are also provided.

- A tool to support the addition of auto-tuning features:
 - *ABCLibCodeGen*: A preprocessor for ABCLibScript directives to support the addition of auto-tuning features using the FIBER method
- Benchmarking software:
 - *ABCLibBench*: Benchmarking using ABCLib

This library has the following features:

- Upon installation, before execution starts, and during execution, the parameter optimization framework automatically adjusts (tunes) performance-related parameters.

This auto-tuning software configuration method is called Framework of Install-time, Before Execute-time, and Run-time optimization layers (FIBER).

ABCLib_DRSSSED was developed as a test type implementing some features of the FIBER software architecture.

ABCLib_DRSSSED uses the subroutine format. It can be called from the FORTRAN language.

ABCLib_DRSSSED can perform the following processing in parallel.

- Conversion of dense real symmetric matrices into tridiagonal matrices via similarity transformation
- Calculation of user-defined number of Eigenvalues (from 1 to all) for a dense real symmetric matrix
- Calculation of user-defined number of Eigenvalues and Eigenvectors (from 1 to all) for a dense real symmetric matrix
- Calculation of all Eigenvalues for a hermitian matrix
- Calculation of all Eigenvalues and Eigenvectors for a hermitian matrix
- Orthogonalize all members of a given group of real vectors

The purpose of the release of this test version is to enable its use by many different users, in order to improve its utility, find and correct bugs, and provide feedback for new development. Consequently, please be aware that there may still be problems with the library, including deficiencies in the manual, poor code readability, and sudden changes in specifications.

Please feel free to contact the author regarding any questions or comments you may have.

1. Introduction

The ABCLib_DRSED library can be used to solve dense symmetric standard Eigenvalue problems. Solving a dense symmetric standard Eigenvalue problem means for a coefficient matrix $A \in Re^{n \times n}$, where $A = A^T$, the Eigenvector is a matrix held in a column vector $X \in Re^{n \times n}$, and all Eigenvalues are a matrix held in diagonal elements $\Lambda \in Re^{n \times n}$, finding the unique solution such that $A = X \Lambda X$.

Additionally, it is possible to use the solution to the dense symmetric standard Eigenvalue problem to calculate all Eigenvalues and Eigenvectors of the hermitian matrix $A \in Im^{n \times n}$, where $A = A^H$.

This library performs tridiagonal conversion using the Householder method. In order to complete the unique solution, all Eigenvalues from the tridiagonal matrix T we have obtained are calculated via dichotomy, and all Eigenvectors via reverse recursion; and using the scalars/vectors obtained during Householder transformation, Householder inverse transformation is performed on all Eigenvectors.

The multi-processor version of this library uses Message Passing Interface 1 (MPI-1), and so will run on computers where MPI-1 is implemented.

2. List of Libraries (Subroutines) Provided

Below are described the functions provided by the test version of ABCLib_DRSED.

- Tridiagonalization routine for dense real symmetric matrices [general version] (*ABCLibTriRed*):
Obtains a tridiagonalized matrix using similarity transformation. Also obtains information about matrix H required for similarity transformation.
- Tridiagonalization routine for dense real symmetric matrices [specialized version] (*ABCLibTriRedR*):
Obtains a tridiagonalized matrix using similarity transformation. Note that this routine cannot obtain information about matrix H required for similarity transformation. However, this routine executes faster than *ABCLibTriRed*.
- Eigenvalue calculation routine for dense real symmetric matrices (*ABCLibDRSAIEig*):
This routine calculates a user-defined number of Eigenvalues in a dense real symmetric matrix. It can calculate a series of m Eigenvalues, starting with the Eigenvalue with

the greatest absolute value.

- Eigenvalue and Eigenvector calculation routine for dense real symmetric matrices (*ABCLibDRSAIEigVec*):
This routine calculates a user-defined number of Eigenvalues and Eigenvectors in a dense real symmetric matrix. When the Eigenvalues corresponding to the Eigenvectors are sorted and numbered in order of greatest absolute value, a series of m Eigenvalues and Eigenvectors can be calculated. This specification assigns the start and end numbers of the sort-numbered Eigenvalues.
- Routine for solving all Eigenvalues in a hermitian matrix (*ABCLibHerAllEig*):
Calculates all Eigenvalues for a hermitian matrix.
- Routine for the calculation of all Eigenvalues and Eigenvectors in a hermitian matrix (*ABCLibHerAllEigVec*):
Calculates all Eigenvalues and Eigenvectors for a hermitian matrix.
- Gram-Schmidt orthogonalization routine (*ABCLibQRD*):
Orthogonalizes two or more real vectors input.

2.1 Householder Similarity Transformation

The *ABCLib_DRSSSED* library makes use of the well known Householder similarity transformation algorithm, shown below.

Theorem: For a given vector $x \in Re^n$, there exist vector $u \in Re^n$ and scalar $alpha \in Re$, such that:

$$(I - \alpha u u^T)x = (v_1, \dots, v_k, \sigma, 0, \dots, 0)^T, \text{ where } \sigma = \|x_{k+1:n}\|_2.$$

Vector $u = (0, \dots, 0, v_{k+1} \pm \sigma, v_{k+2}, \dots, v_n)^T$ and scalar $\alpha = 1 / (\sigma^2 + \|v_{k+1} \ \sigma\|)$ fulfill the above theorem. Additionally,

$$\|u\|_2^2 = (v_{k+1} \pm \sigma)^2 + v_{k+2}^2 + \dots + v_n^2 = \sigma^2 + \sigma^2 + 2 \|v_{k+1} \ \sigma\| = 2 (\sigma^2 + \|v_{k+1} \ \sigma\|) = 2 / \alpha;$$

therefore, $\alpha u^T u = 2$. In order to avoid loss of digits when calculating vector u , the sign of σ is the same as the sign of v_{k+1} . Here, the transformation $(I - \alpha u u^T)x$ is expressed as $H^{(k)}(x)$. This transformation does not operate on elements v_1, \dots, v_k . Furthermore, the vector-scalar pair needed for transformation $H^{(k)}(x)$ are expressed as (u, α) .

Consider the transformation of $A(1) = A$ to the tridiagonal matrix $A(n-2) = T$. By applying $H^{(k)} = I - \alpha u u^T$ to the $k+1$ th iteration of matrix A , we are able to obtain the following formula:

$$\begin{aligned}
A^{(k+1)} &= H^{(k)} A^{(k)} H^{(k)} \\
&= A^{(k)} - \alpha A^{(k)} u u^T - \alpha u u^T A^{(k)} - \alpha^2 u u^T A^{(k)} u u^T \\
&= A^{(k)} - x u^T - u y^T + \alpha u u^T x u^T \\
&= A^{(k)} - u y^T + u \mu u^T - x u^T = A^{(k)} - u (y^T - \mu u^T) - x u^T
\end{aligned}$$

Where $x = \alpha A^{(k)} u$, $y^T = \alpha u^T A^{(k)}$, and $\mu = \alpha u^T x$.

Now, as A is symmetric, $x = y$, yielding the following formula:

$$A^{(k+1)} = A^{(k)} - u (x^T - \mu u^T) - x u^T$$

Note that for the k^{th} recursion, vector $A_{k:n, k}$ is required for matrix $A_{k:n, k:n}$. Here, vector $A_{k:n, k}$ is called the **pivot vector**.

Below is an outline of the tridiagonalization algorithm.

c Pmyidx, myidy owns row set P_i and column set Γ of $n \times n$ matrix A

```

1: do k=1, n-2
2:   if (k ∈ Gamma) then
3:     Broadcast(A(k)_ {P_i, k}) to PEs sharing rows P_i.
4:   else
5:     receive(A(k)_ {P_i, k})
6:   endif
7:   Computation of (u_Pi, alpha).
8:   if (I have diagonal elements of A) then
9:     Broadcast(u_Pi) to PEs sharing columns Gamma.
10:  else
11:    receive(u_Gamma)
12:  endif
13:  do j=k, n
14:    if (j ∈ Gamma) x_Pi = x_Pi + alpha A(k)_ {P_i, j} u_j endif
15:  enddo
16:  Global summation of x_Pi to PEs sharing rows P_i.
17:  if (I have diagonal elements of A) then
18:    Broadcast(x_Pi) to PEs sharing columns Gamma.
19:  else
20:    receive(x_Gamma)

```

```

21: endif
22: do j=k, n
23:   mu = alpha u_Pi^T x_Pi enddo
24:   Global summation of mu to PEs sharing rows Pi.
25:   do j=k, n
26:     do i=k, n
27:       if (i ∈ Pi .and. j ∈ Gamma) then
28:         A(k+1)_i, j = A(k)_i, j - ui (y^T_j - mu u^T_j) - y_i u^T_j
29:       endif enddo enddo
   c Remove k from active columns and rows.
30:   if (k ∈ Gamma) Gamma = Gamma - {k} endif
31:   if (k ∈ Pi) Pi = Pi - {k} endif
32: enddo

```

In this algorithm, the pivot vector is sent in lines 2. to 12.; the matrix-vector product $x = \alpha A^{(k)}u$ is taken in 13: to 15: ;

In 22: to 24:, the inner product $\mu = \alpha u^T x$ is calculated; and in 25: to 29:, the matrix is updated:

$$A^{(k+1)} = A^{(k)} - u(x^T - \mu u^T) - x u^T.$$

2.2 Cyclic-Cyclic Distribution

For the parallelization of the calculation of the tridiagonalization and all Eigenvalues and Eigenvectors based thereon, the distribution of matrix A is assumed to be a cyclic-cyclic distribution*.

The input data is the elements of matrix A in the standard Eigenvalue problem $A X = X \Lambda$. These elements are distributed to each PE by means of the cyclic-cyclic distribution method, and used as the input data.

Below is a concrete description of the cyclic-cyclic distribution method. First, let the one-dimensional number of $nprocs$ processors be pe_num . Here, let the scope be

* For the solution of the hermitian matrix, however, data is not distributed; it is passed sequentially using the same interface as the program.

$pe_num : 0, 1, \dots, nprocs-1$

Additionally, let $nprocs$ be

$$nprocs = ncelx \times ncely$$

and

- $ncelx < ncely$, where $ncelx$ must not be 1, and
- $ncely / ncelx$ is divisible.

Here, the one-dimensional number pe_num is converted into the two-dimensional processor number $(nmyidx, nmyidy)$ using the following subprogram.

```
subroutine cid2xy(pe_num, ncelx, nmyidx, nmyidy)
  integer pe_num, ncelx, nmyidx, nmyidy
  nmyidx = modulo(pe_num, ncelx)
  nmyidy = (pe_num - modulo(pe_num, ncelx))/ncelx
return
end
```

Now, we have obtained the two-dimensional processor number $(nmyidx, nmyidy)$. Next, in order to distribute matrix $n \times n$ matrix A to $nprocs$ processors, we write the following (however, although the implementation is easy, this method is not very computationally efficient):

Now, let the subscript of matrix A change from 0 to $n-1$.

```
do i=0, n-1
  li = LOC(i)
  if (nmyidx .eq. OWNER(i)) then
    do j=0, n-1
      if (nmyidy .eq. OWNER(j)) then
        A(li, LOC(j)) = n - i
      endif
    enddo
  endif
enddo
```

Here, OWNER(i) is the array of two-dimensional numbers of the processor group having the i^{th} column, and LOC(i) is the array of local subscripts of the processors having the i^{th} column. This list is created as follows:

```
do i=0, n-1
  LOC(i) = CYCLIC_LOC(i, ncelx)
  OWNER(i) = CYCLIC_OWNER(i, ncelx)
enddo
```

Functions CYCLIC_LOC(i, ncelx) and CYCLIC_OWNER(i, ncelx) are as follows:

```
integer function CYCLIC_OWNER(i, nprocs)
  integer i, nprocs
  CYCLIC_OWNER = modulo(i, nprocs)
return
end
```

```
integer function CYCLIC_LOC(i, nprocs)
  integer i, nprocs
  CYCLIC_LOC = idfloor(i/nprocs)
return
end
```

If we know how to calculate the two-dimensional number of the processor owning the i^{th} column, the same holds for the two-dimensional number of the processor owning the j^{th} column. Thus, there is no need to calculate ownership information relating to matrix direction.

Specifically, we write as follows: (nprocs=4, n=3)

Matrix A = (a_{ij}), i, j=1,...,3

↓(i) →(j)

a₁₁, a₁₂, a₁₃

a₂₁, a₂₂, a₂₃

a₃₁, a₃₂, a₃₃

OWNER(i), owner information (nmyidx, nmyidy)

0, (0,0) (0,1) (0,0)

1, (1,0) (1,1) (1,0)

0, (0,0) (0,1) (0,0)

LOCAL(i), internal subscript information (local_i, local_j)

0, (0,0) (0,0) (0,1)

0, (0,0) (0,0) (0,1)

1, (1,0) (1,0) (1,1)

PE pe_num, (nmyidx, nmyidy)

0, (0, 0)

1, (1, 0)

2, (0, 1)

3, (1, 1)

Here, "ownership information" means the two-dimensional PE number that owns matrix a_{ij} .

"Internal subscript information" means the local subscript number to substitute for matrix a_{ij} .

For example, we know from the ownership information that the 0th processor (0, 0) owns elements:

a_{11} , a_{31} , a_{13} , a_{33}

And we know from the internal subscript information to substitute these locally where necessary with the following:

(0,0), (1,0), (0,1), (1,1)

3. Auto Tuning Features

This section describes the auto-tuning features implemented by this library.

3.1 Auto Tuning Types Using FIBER

The FIBER software architecture classifies auto-tuning into the following three categories.

- Install-time auto tuning:
Auto tuning performed when the library is installed
- Pre-execution auto tuning:
Auto tuning performed after basic information provided by the user (e.g. number of processors and problem size) has been specified
- Run-time auto tuning:
Auto tuning performed when the library runs

The test version of ABCLib_DRSSSED implements a subset of the features of the FIBER software architecture. Specifically, the library can perform some of the install-time and pre-execution auto-tuning features[†].

3.2 Library Structure

The library is made up of the following four main modules:

- 1 . Householder tridiagonalization module
- 2 . Eigenvalue calculation module using dichotomy
- 3 . Eigenvector calculation module using reverse recursion
- 4 . Eigenvector calculation module using Householder inverse transformation

The current version includes auto-tuning features in modules 1 and 4.

Additionally, the current version of the library also includes an orthogonalization routine using the Gram-Schmidt Method that can be used for QR iteration and other linear-algebraic calculations, although it does not use it for the calculation of Eigenvalues or

[†]In the current version, pre-execution auto tuning is performed manually (see section 6.4, "Executing Auto Tuning" for details).

Eigenvectors. This orthogonalization routine also includes auto-tuning features.

3.3 The Householder Tridiagonalization Module

The following auto-tuning is performed in the Householder tridiagonalization module:

- Number of levels of loop unrolling to perform for matrix-vector products
- Number of levels of loop unrolling to perform for matrix updating
- Communication method

For matrix-vector product loop unrolling, there are 16 versions of the second-innermost loop in the loop for calculating the matrix-vector product, consisting of 1, 2, ..., 16 levels. The number of levels is automatically adjusted.

For matrix-update loop unrolling, there are 16 versions of the second-innermost loop in the loop for updating matrices, consisting of 1, 2, ..., 16 levels. The number of levels is automatically adjusted.

Communication-method auto tuning consists of automatically selecting the communication method that enables the fastest calculation of inter-processor vector addition necessary for tridiagonalization, from the following two methods.

- Binary tree-structure communications:
This method uses the MPI_Recv and MPI_Send synchronous communications functions in the MPI library.
- MPI functions:
This method uses the MPI_Allreduce function in the MPI library.

3.4 Eigenvector Calculation Module Using Householder Inverse Transformation

The following auto-tuning is performed in the Eigenvector calculation module using Householder inverse transformation:

- Number of levels of loop unrolling to perform for matrix updating
- Communication method

For matrix-update loop unrolling, there are 16 versions of the second-innermost loop,

consisting of 1, 2, ..., 16 levels. The number of levels is automatically adjusted.

Communication-method auto tuning consists of automatically selecting the communication method that enables the fastest calculation of processing required for freeing vectors, from the following three methods:

- MPI functions:
This method uses the MPI_Bcast function in the MPI library.
- Binary tree-structure communications using synchronous communications:
This method uses the MPI_Recv and MPI_Send synchronous communications functions in the MPI library.
- Binary tree-structure communications using asynchronous communications:
This method uses the MPI_Recv synchronous communications function and the MPI_Isend asynchronous communications function in the MPI library.

3.5 The Gram-Schmidt Orthogonalization Routine

The following auto-tuning is performed in the Gram-Schmidt orthogonalization routine:

- Block width
- Number of levels of loop unrolling to perform for matrix updating for pivot PE
- Number of levels of loop unrolling to perform for main matrix updating

The Gram-Schmidt orthogonalization routine uses a blocking algorithm, whose block width affects communications and calculation performance. A block width of 1, 2, 3, 4, 5, 6, 8, or 16 can be specified; this width is adjusted automatically.

For pivot PE matrix updating, the outermost loop can have 1, 2, 3, or 4 levels, and the second-outermost loop can have 1, 2, 3, 4, 5, 6, 8, or 16 levels, for a total of $4 \times 8 = 32$ permutations. The numbers of levels are adjusted automatically.

As with pivot PE matrix updating, for main-matrix updating, the outermost loop can have 1, 2, 3, or 4 levels, and the second-outermost loop can have 1, 2, 3, 4, 5, 6, 8, or 16 levels, for a total of $4 \times 8 = 32$ permutations. The numbers of levels are adjusted automatically.

4. Installation

4.1 Procedures

Decompress the provided file `abclib_r_x.tar.gz` (where x is the version).

Under UNIX, the procedure would be as follows:

```
% gzip -d abclib_r_x.tar.gz
% tar xvf abclib_r_x.tar
```

After the above decompression, execute the command `abclib_install‡` located in the `ABCLib_Rx` directory, as follows:

```
% cd ABCLib_Rx
% ./abclib_install
```

The `abclib_install` command generates all sample programs and the static library. The static library generated by the `abclib_install` command is `ABCLib.a`, located in the `ABCLib_Rx/lib` directory.

[‡] Note that `tests/Makefile` is specially for when the PGI compiler is used. Run this file after editing the compiler commands and optimization options appropriately.

4.2 Directory Structure

After decompression, the ABCLib_DRSSSED directory structure is as follows:

```
ABCLib_Rx/ --- DRSSSED/ --- src/ --- *.f
|
|__ doc/ --- *.ps
|    |__ *.pdf
|
|__ include/ --- ABCLib*.h
|
|__ tests/ --- Makefile
|    |__ ABCLibTest*.f
|
|__ COPYRIGHT
|__ KnownBugs
|__ README
|__ abclib_install
|__ abclib_uninstall
|__ lib – ABCLib.a (Note: created after execution of abclib_install)
```

The DRSSSED/ directory contains the Eigenvalue solver sourcecode.

The doc/ directory contains documentation related to ABCLIB_DRSSSED.

The include/ directory contains include files needed by ABCLib.

The tests/ directory contains sample programs for using the library. You can compile these sample programs to confirm usage, and make sure that the library will start.

The lib/ directory contains the static library. It is generated after the abclib_install command is executed. It is deleted after abclib_uninstall is executed.

5. Sample Programs

5.1 Solving Eigenvalues and Eigenvectors in Symmetric Real Dense Matrices

The sample program for calculating Eigenvalues and Eigenvectors in symmetric real dense matrices, tests/ABCLibTestDRSAllEigVec.f, is as follows:

```

    program main
    include 'ABCLibDRSSED.h'
    common /ABCLibpval/ nprocs, myid, ncelx, ncely, nx, ny, ierrcode
c    === real*8 definition
    real*8 A(0:ABCLibMAXNX-1, 0:ABCLibMAXNY-1)
    real*8 eig(1:ABCLibMAXNDIVP)
    real*8 X(1:ABCLibMAXN, 1:ABCLibMAXNDIVP)
c    === for checking answer
    real*8 dtemp
    real*8 EigErr, VecErr, SysErr
c    === for measuring time
    real*8 t1, t2, bt, t_all
c    =====
c    === integer definition
c    === for parallel control
    integer ncelx, ncely, nx, ny
    integer myid, nmyidx, nmyidy
    integer n, m, ms, me, nprocs
c    === for error flag
    integer ierrcode
c    === for selecting test matrices
    integer isw_mat
c    =====
c    === Program start
c    === MPI Init.
    call MPI_INIT( ierr )
    call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
    call MPI_COMM_SIZE( MPI_COMM_WORLD, nprocs, ierr )
c    =====
c    === Print title
```

```

if (myid .eq. 0) then
  print *, ""
  print *, "==== Parallel Eigensolver Check Program ===="
  print *, "Parallel Eigensolver on MPI"
  print *, "ABCLibDRSAllEigVec"
  call ABCLibPrintVer()
  print *, ""
endif

c =====
c === Setting Test Condition
if (myid .eq. 0) then
  write(6,*) "Problem size >"
  read(5,*) n
  write(6,*) "Start number of eigenvectors >"
  read(5,*) ms
  write(6,*) "End number of eigenvectors >"
  read(5,*) me
  m = me - ms + 1
c   === set matrix type
  isw_mat = 1
endif

c =====
c == Set Parameters
call MPI_BCAST(n,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
call MPI_BCAST(m,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
call MPI_BCAST(ms,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
call MPI_BCAST(me,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
call MPI_BCAST(isw_mat,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

c =====
c === Set Parallel Control Values
if (myid .eq. 0) then
  write(6,*) " ncelx="
  read(5,*) ncelx
  write(6,*) " ncely ="
  read(5,*) ncely
endif

```

```

call MPI_BCAST(ncelx, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
call MPI_BCAST(ncely, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
call cid2xy2(myid, ncelx, ncely, nmyidx, nmyidy)
nx = ceiling(dfloat(n)/dfloat(ncelx))
ny = ceiling(dfloat(n)/dfloat(ncely))
c =====
c === Generate Test Matrix
call MakeTestMat2(A, n, nmyidx, nmyidy, isw_mat)
c =====
c === Initialize ABCLibTriRed routine
call ABCLibDRSAIEigVec_Init()
c =====
c === Call Tridiagonalization Routine
call MPI_BARRIER(MPI_COMM_WORLD, ierr)
t1 = MPI_WTIME()
call ABCLibDRSAIEigVec(A, n, eig, X, ms, me)
call MPI_BARRIER(MPI_COMM_WORLD, ierr)
t2 = MPI_WTIME()
t_all = t2 - t1
call MPI_REDUCE(t_all, bt, 1, MPI_DOUBLE_PRECISION, MPI_MAX, 0,
& MPI_COMM_WORLD, ierr)
t_all = bt
c =====
c === Calculate Eigenvalues & Check Its Error
if (ierrcode .eq. 0) then
  if (myid .eq. 0) print *, "Normal return"
else
  print *, "myid=",myid,"Abnormal return: error code",ierrcode
endif
if (isw_mat .eq. 1) then
  if (myid .eq. 0) print *,
& "Check of maximal error for eigenvalues..."
  call ChkEigErr(eig, n, EigErr, ms, me)
  if (myid .eq. 0) print *, "End of checking."
endif
  if (myid .eq. 0) print *,

```

```

&      "Check of error for eigenvectors...."
      call TestVec(X, n, VecErr, m)
      if (myid .eq. 0) print *, "End of checking."
      if (myid .eq. 0) print *,
&      "Check of error for the eigensystem...."
      call TestEigSys(eig, X, n, isw_mat, SysErr, m)
      if (myid .eq. 0) print *, "End of checking."
c      =====
c      === Output Results
      if (myid .eq. 0) then
        print *, ""
        print *, "=== Result "
        print *, "===== "
        print *, "n = ", n, ", Number of processors = ", nprocs
        print *, "Number of eigenvalues and eigenvectors = ", m
        print *, "Start Eigenvalue No.", ms, "/ End Eigenvalue No.", me
        print *, "Test Matrix:"
        if (isw_mat .eq. 1) print *, "Frank"
        if (isw_mat .eq. 2) print *, "Rnd (-32768 to 32765)"
        if (isw_mat .eq. 4) print *, "Glud Wilkinson W_21, d=1.0d-14"
        if (isw_mat .eq. 5) print *, "Unit Matrix"
        if (isw_mat .eq. 6) print *, "Wilkinson W_n+"
        print *, ""
        if (isw_mat .eq. 1) then
          print *, "Eigenvalue Max Error = ", EigErr
        endif
        print *, "Eigenvector Error of ||X^T X - I|| = ", VecErr
        print *,
&      "Eigensystem Error max_i | A x_i - A_i x_i | = ", SysErr
      write(*, 1000) t_all
1000  format(" Total Calculation Time = ",F10.3," [sec]")
      print *, ""
      endif
c      =====
c      === Finalize ABCLibDRSAIEig routine
      call ABCLibDRSAIEigVec_Finalize()

```

```

c =====
c ===== MPI finazize
c call MPI_FINALIZE(ierr)
c =====
c -----
c stop
c end

```

5.1.1 Description of Sample Program

To use ABCLib_DRSSSED, add the following header file and common statement.

```
include 'ABCLibDRSSED.h'
```

```
common /ABCLibpval/ nprocs, myid, ncelx,ncely, nx, ny, ierrcode
```

The parameters in the common statement are all integer variables. The parameters to set and their meanings are as follows.

- nprocs: The number of processors to use
- myid: Your processor numbers (0,...,nprocs-1)
- ncelx,ncely: Values such that nprocs = ncelx * ncely
- nx,ny: The values such that nx = n / ncelx and ny = n / ncely, when the problem size to use is n
- ierrcode: Contains the error code when the library returns. This value does not need to be set.

Library calls have the following format:

```
call ABCLibDRSAllEigVec(A, n, eig, X, ms, me)
```

The meanings of each parameter are as follows.

- A: Coefficient matrix (0:NdivP-1, 0:NdivP-1) upon which cyclic-cyclic distribution has been performed
- N: The dimensionality of the coefficient matrix. An integer value.
- Eig: Block distributed, calculated Eigenvalues (1: NdivP)
- X: Calculated Eigenvectors block-distributed in the column direction (1:n, 1:NdivP-1)

- Ms: The start number of the Eigenvalues corresponding to the requested Eigenvector (counting from greatest absolute value)
- Me: The end number of the Eigenvalues corresponding to the requested Eigenvector (counting from greatest absolute value)

5.2 Solving All Eigenvalues and Eigenvectors of a Hermitian Matrix

The sample program for calculating all Eigenvalues and Eigenvectors in a hermitian matrix (tests/ABCLibTestHerAllEigVec.f) is as follows:

```

program main
include 'ABCLibDRSSED.h'
common /ABCLibpval/ nprocs, myid, ncelx, ncely, nx, ny, ierrcode
c    === real*8 definition
c    === for tridiagonalization
c        === The AR is real part, and the AI is imaginary part of
c            the Hermitian Matrix of A
real*8  AR(1:ABCLibMAXN, 1:ABCLibMAXN)
real*8  AI(1:ABCLibMAXN, 1:ABCLibMAXN)
c        === The number of eigenvalues of the matrix A is 2 times
c            larger than original one, because of conjugate eigenvectors.
real*8  eig(1:ABCLibMAXNDIVP*2)
real*8  X(1:ABCLibMAXN*2, 1:ABCLibMAXNDIVP*2)
c    === for checking answer
real*8  dtemp
real*8  EigErr, VecErr1, VecErr2, SysErr
c    === for measuring time
real*8  t1, t2, bt, t_all
c    =====
c    === integer definition
c    === for parallel control
integer  ncelx, ncely, nx, ny
integer  myid, nmyidx, nmyidy
integer  n, nprocs

```

```

c      === for error flag
integer   ierrcode
c      === for selecting test matrices
integer   isw_mat
c      =====
c      === Program start
c      === MPI Init.
call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, nprocs, ierr )
c      =====
c      === Print title
if (myid .eq. 0) then
  print *, ""
  print *, "=== Parallel Eigensolver Check Program ==="
  print *, "Parallel Eigensolver on MPI"
  print *, "ABCLibHerAllEigVec"
  call ABCLibPrintVer()
  print *, ""
endif
c      =====
c      === Setting Test Condition
c      === set matrix type
isw_mat = 1
call MPI_BCAST(isw_mat,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
if (isw_mat .ge. 5) then
  if (myid .eq. 0) then
    write(6,*) "Problem size of the Hermitian matrix >"
    read(5,*) n
  endif
endif
c      =====
c      === Set Parallel Control Values
if (myid .eq. 0) then
  write(6,*) " ncelx ="
  read(5,*) ncelx

```

```

        write(6,*) " ncely ="
        read(5,*) ncely
    endif
    call MPI_BCAST(ncelx, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
    call MPI_BCAST(ncely, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
c
c =====
c
c === Generate Test Matrix
    call HerTestMatDim(n, isw_mat)
    call HerMakeTestMat(AR, AI, n, isw_mat)
c
c =====
c
c === Set Parallel Control Values
    call MPI_BCAST(n,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
    nx = ceiling(dfloat(2*n)/dfloat(ncelx))
    ny = ceiling(dfloat(2*n)/dfloat(ncely))
c
c =====
c
c === Initialize ABCLibHerAllEig routine
    call ABCLibHerAllEigVec_Init()
c
c =====
c
c === Call Hermitian Eigenvalue Computation Routine
    call MPI_BARRIER(MPI_COMM_WORLD, ierr)
    t1 = MPI_WTIME()
    call ABCLibHerAllEigVec(AR, AI, n, eig, X)

    call MPI_BARRIER(MPI_COMM_WORLD, ierr)
    t2 = MPI_WTIME()
    t_all = t2 - t1
    call MPI_REDUCE(t_all, bt, 1, MPI_DOUBLE_PRECISION, MPI_MAX, 0,
&                MPI_COMM_WORLD, ierr)
    t_all = bt
c
c =====
c
c === Check Its Error
    if (ierrcode .eq. 0) then
        print *, myid, "Normal return"
    else
        print *, "myid=",myid,"Abnormal return: error code",ierrcode
    endif
endif

```

```

if (isw_mat .le. 3) then
  if (myid .eq. 0) then
    print *,
&      "Check of maximal error for eigenvalues...."
  endif
  call HerChkEigErr(eig, n, EigErr, isw_mat)
  if (myid .eq. 0) then
    if (myid .eq. 0) print *, "End of checking."
  endif
endif
if (myid .eq. 0) print *,
&  "Check of error for eigenvectors...."
call TestVec(X, 2*n, VecErr1, 2*n)
call TestHerVec(X, eig, n, VecErr2)
if (myid .eq. 0) print *, "End of checking."
if (myid .eq. 0) print *,
&  "Check of error for the eigensystem...."
call TestHerEigSys(AR, AI, eig, X, n, isw_mat, SysErr)
if (myid .eq. 0) print *, "End of checking."
c  =====
c  === Output Results
if (myid .eq. 0) then
  print *, ""
  print *, "=== Result "
  print *, "===== "
  print *, "n = ", n, ", Number of processors = ", nprocs
  print *, "Test Matrix No:", isw_mat
  print *, ""
  if (isw_mat .le. 3) then
    print *, "Eigenvalue Max Error = ", EigErr
  endif
  print *, "Eigenvector Error of Real ||X^T X - I|| = ", VecErr1
  print *, "Eigenvector Error of Her ||X^H X - I|| = ", VecErr2
  print *, "Eigensystem Error of Hermitian:"
  print *, "  max_i ||A x_i - A_i x_i|| = ", SysErr
  write(*, 1000) t_all

```

```

1000  format(" Total Calculation Time = ",F10.3," [sec]")
      print *, ""
      endif
c     =====
c     === Finalize ABCLibHerAllEig routine
      call ABCLibHerAllEigVec_Finalize()
c     =====
c     ===== MPI finazize
      call MPI_FINALIZE(ierr)
c     =====
c     -----
      stop
      end

```

5.2.1 Description of Sample Program

Add the same header file and `common` statement.

Library calls have the following format:

```
call ABCLibHerAllEigVec(AR, AI, n, eig, X)
```

The meanings of each parameter are as follows.

- AR: The reals of a non-distributed, hermitian coefficient matrix (1:n, 1:n)
- AI: The imaginary part of a non-distributed, hermitian coefficient matrix (1:n, 1:n)
- n: The dimensionality of the coefficient matrix. An integer value.
- eig: Block distributed, calculated Eigenvalues (1:2*NdivP)
- X: Calculated Eigenvectors block-distributed in the column direction (1:2*n, 1:2*NdivP-1)

5.3 Orthogonalization Using the Gram-Schmidt Method

The sample program for orthogonalization via the Gram-Schmidt method (tests/ABCLibTestQRD.f) is as follows:

```

      program main
      include 'ABCLibDRSSED.h'
      common /ABCLibpval/ nprocs, myid, ncelx, ncely, nx, ny, ierrcode
c     === real*8 definition
      real*8 X(1:ABCLibMAXN, 1:ABCLibMAXNDIVP)
c     === for checking answer
      real*8 dtemp
      real*8 VecErr
c     === for measuring time
      real*8 t1, t2, bt, t_all
c     =====
c     === integer definition
c     === for parallel control
      integer ncelx, ncely, nx, ny
      integer myid, nmyidx, nmyidy
      integer n, nprocs
c     === for error flag
      integer ierrcode
c     === for selecting test matrices
      integer isw_mat
c     =====
c     === Program start
c     === MPI Init.
      call MPI_INIT( ierr )
      call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
      call MPI_COMM_SIZE( MPI_COMM_WORLD, nprocs, ierr )
c     =====
c     === Print title
      if (myid .eq. 0) then
        print *, ""
        print *, "=== Parallel Eigensolver Check Program ==="
        print *, "Parallel QR Decomposition on MPI"
      end if

```

```

    print *, "ABCLibDRSAllEigVec"
    call ABCLibPrintVer()
    print *, ""
endif
c =====
c === Setting Test Condition
if (myid .eq. 0) then
    write(6,*) "problem size >"
    read(5,*) n
c    === set matrix type
    isw_mat = 1
endif
c =====
c == Set Parameters
call MPI_BCAST(n,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
call MPI_BCAST(isw_mat,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
c =====
c === Set Parallel Control Values
if (myid .eq. 0) then
    write(6,*) " ncelx="
    read(5,*) ncelx
    write(6,*) " ncely ="
    read(5,*) ncely
endif
call MPI_BCAST(ncelx, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
call MPI_BCAST(ncely, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
call cid2xy2(myid, ncelx, ncely, nmyidx, nmyidy)
nx = ceiling(dfloat(n)/dfloat(ncelx))
ny = ceiling(dfloat(n)/dfloat(ncely))
c =====
c === Generate Test Matrix
call MakeMatSeqCB(X, n, isw_mat)
c =====
c === Initialize ABCLib_QRD routine
call ABCLibQRD_Init()
c =====

```

```

c      === Call Tridiagonalization Routine
      call MPI_BARRIER(MPI_COMM_WORLD, ierr)
      t1 = MPI_WTIME()
      call ABCLib_QRD(X, n)
      call MPI_BARRIER(MPI_COMM_WORLD, ierr)
      t2 = MPI_WTIME()
      t_all = t2 - t1
      call MPI_REDUCE(t_all, bt, 1, MPI_DOUBLE_PRECISION, MPI_MAX, 0,
&                MPI_COMM_WORLD, ierr)
      t_all = bt
c      =====
c      === Orthogonalized Vectors & Check Its Error
      if (ierrcode .eq. 0) then
        if (myid .eq. 0) print *, "Normal return"
      else
        print *, "myid=",myid,"Abnormal return: error code",ierrcode
      endif
      if (myid .eq. 0) print *,
&    "Check of error for vectors...."
      call TestVec(X, n, VecErr, n)
      if (myid .eq. 0) print *, "End of checking."
c      =====
c      === Output Results
      if (myid .eq. 0) then
        print *, ""
        print *, "=== Result "
        print *, "===== "
        print *, "n = ", n, ", Number of processors = ", nprocs
        print *, "Test Matrix:"
        if (isw_mat .eq. 1) print *, "Random"
        print *, ""
        print *, "Orthogonal Error of  $||X^T X - I|| =$ ", VecErr
        write(*, 1000) t_all
1000  format(" Total Calculation Time = ",F10.3," [sec]")
        print *, ""
      endif

```

```

c =====
c === Finalize ABCLib_QRD routine
c call ABCLibQRD_Finalize()
c =====
c ===== MPI finazize
c call MPI_FINALIZE(ierr)
c =====
c -----
c
c stop
c end

```

5.3.1 Description of Sample Program

Add the same header file and common statement.

Library calls have the following format:

```
call ABCLib_QRD(X, n)
```

The meanings of each parameter are as follows.

- X: Matrix comprising vectors to orthogonalization, block-distributed in the column direction (1:n, 1:NdivP-1)
Note that the orthogonalized vectors are also overwritten onto this matrix.
- N: The dimensionality of the coefficient matrix. An integer value.

6. Sample Execution

6.1 Library Execution for Calculation of Eigenvalues and Eigenvectors in Dense Real Symmetric Matrix

Below is a sample execution of the `ABCLibTestDRSAllEigVec.f` test program, which calculates the Eigenvalues and Eigenvectors in a dense real symmetric matrix. Note that the format of the sample output below may change due to version modifications or the like.

In this example, the specification file `.ABCLibDRSAllEigVec` (described in section 6.4.1) is as follows:

```
-ort CGS -autotune no -starttunesize 128 -maxtunesize 512
-tunestrade100 128 -tunestrade1000 1000 -print yes
```

Below are the results of the calculation of 51 Eigenvalues (10th through 60th, counting from greatest absolute value) and their corresponding 51 Eigenvectors, for a 100-dimension Frank matrix using 4 PEs.

```
$ mpirun -np 4 -machinefile nodeinfo ABCLibTestDRSAllEigVec
```

```
=== Parallel Eigensolver Check Program ===
```

```
Parallel Eigensolver on MPI
```

```
ABCLibDRSAllEigVec
```

```
-----
```

```
ABCLib Ver. 1.00
```

```
Composed by T. Katagiri, October 2004
```

```
-----
```

```
Problem size >
```

```
100
```

```
Start number of eigenvectors >
```

```
10
```

```
End number of eigenvectors >
```

```
60
```

```
ncelx=
```

```
2
```

```

ncely =
2
!!!! Warning !!!!
There is no 'autotuneTRD.dat' file.
I will set AD-HOC parameters for TriRed.
I strongly recommend doing auto-tuning.
!!!! Warning !!!!
There is no 'autotuneHIT.dat' file.
I will set AD-HOC parameters for HIT.
I strongly recommend doing auto-tuning.
Auto-tuned :(Tri)          100          0          8          6
Auto-tuned (HIT):         100          1          1
Tridiagonalization Time =    0.298 [sec]
Re-distribution Time =     0.002 [sec]
Calculating Eigenvalues Time =    0.003 [sec]
----- All Eigenvalues Cal. Time =    0.303 [sec] -----
Gathering Eigenvalues Time =    0.038 [sec]
SubMatrix Indexes :         1          100
Number of Groups :         1
Max clustered eigenvalues :          51
Constant for convergence :   4.9652297362006328E-012
Distance for orthogonalization :   6.391867118793334
fact :   1.0000000000000000E-003
Method for ort. : Peters-Wilkinson
Inverse Iteration : CGS
Calculating Eigenvectors Time =    0.629 [sec]
Householder Inverse Iteration Time =    0.084 [sec]
myid=          0 Abnormal return: error code          4000
Check of maximal error for eigenvalues....
End of checking.
Check of error for eigenvectors....
End of checking.
Check of error for the eigensystem....
End of checking.
=== Result
=====

```

```

n =          100 ,Number of processors =          4
Number of eigenvalues and eigenvectors =          51
Start Eigenvalue No.          10 / End Eigenvalue No.          60
Test Matrix:
Frank
Eigenvalue Max Error =    5.1249242025868220E-014
Eigenvector Error of ||X^T X - I|| =    7.7850102906234229E-014
Eigensystem Error max_i ||A x_i - A_i x_i|| =    1.6640934173348136E-011
Total Calculation Time =    1.059 [sec]

```

6.2 Library Execution for Calculation of Eigenvalues and Eigenvectors in Hermitian Matrix

Below is a sample execution of the ABCLibTestHerAllEigVec.f test program, which calculates all Eigenvalues and Eigenvectors in a hermitian matrix. Note that the format of the sample output below may change due to version modifications or the like.

In this example, the specification file .ABCLibHerAllEigVec (described in section 6.4.1) is as follows:

```

-ort MGS -autotune no -starttunesize 128 -maxtunesize 512
-tunestrade100 128 -tunestrade1000 1000 -print yes

```

Below are the results using 4 PEs.

```
$ mpirun -np 4 -machinefile nodeinfo ABCLibTestHerAllEigVec
```

```
=== Parallel Eigensolver Check Program ===
```

```
Parallel Eigensolver on MPI
```

```
ABCLibHerAllEigVec
```

```
-----
```

```
ABCLib ver. 1.00
```

```
Composed by T. Katagiri, October 2004
```

```
-----
```

```

ncelx =
2
ncely =
2
!!!! Warning !!!!
There is no 'autotuneTRD.dat' file.
I will set AD-HOC parameters for TriRed.
I strongly recommend doing auto-tuning.
!!!! Warning !!!!
There is no 'autotuneHIT.dat' file.
I will set AD-HOC parameters for HIT.
I strongly recommend doing auto-tuning.
Auto-tuned :(Tri)           8           0           8           6
Auto-tuned (HIT):          8           1           1
Tridiagonalization Time =   0.965 [sec]
Re-distribution Time =     0.007 [sec]
Calculating Eigenvalues Time = 0.002 [sec]
----- All Eigenvalues Cal. Time = 0.974 [sec] -----
Gathering Eigenvalues Time = 0.038 [sec]
SubMatrix Indexes :         1           3
Number of Groups :         3
Max clustered eigenvalues :           0
Constant for convergence :   1.2414855315468873E-014
Distance for orthogonalization : 2.7931648995851659E-002
fact : 1.0000000000000000E-003
Method for ort. : Peters-Wilkinson
Inverse Iteration : MGS
Calculating Eigenvectors Time = 0.000 [sec]
Householder Inverse Iteration Time = 0.003 [sec]
myid=          0 Abnormal return: error code          4000
Check of maximal error for eigenvalues....
End of checking.
Check of error for eigenvectors....
End of checking.
Check of error for the eigensystem....
End of checking.

```

=== Result

```
=====
n =          4 ,Number of processors =          4
Test Matrix No:          1

Eigenvalue Max Error =    1.4924678768188513E-015
Eigenvector Error of Real ||X^T X - I|| =    2.9447165805096462E-011
Eigenvector Error of Her ||X^H X - I|| =    3.7867080775839170E-011
Eigensystem Error of Hermitian:
  max_i ||A x_i - A_i x_i|| =    4.9365182464366839E-011
Total Calculation Time =    1.062 [sec]
```

6.3 Library Execution Relating to Orthogonalization Using the Gram-Schmidt Method

Below is a sample execution of the ABCLibTestQRD.f test program, which performs orthogonalization via the Gram-Schmidt method. Note that the format of the sample output below may change due to version modifications or the like.

In this example, the specification file .ABCLibQRD (described in section 5.4.1) is as follows:

```
-ort MGS -autotune no -starttunesize 128 -maxtunesize 512
-tunestrade100 128 -tunestrade1000 1000 -print yes
```

Below is a sample execution of the orthogonalization of 100 100-dimension random vectors using 4 PEs.

```
$ mpirun -np 4 -machinefile nodeinfo ABCLibTestQRD
=== Parallel Eigensolver Check Program ===
Parallel QR Decomposition on MPI
ABCLibDRSAllEigVec
-----
ABCLib ver. 1.00
Composed by T. Katagiri, October 2004
-----
```

```

problem size >
100
  ncelx=
2
  ncely =
2
  !!!! Warning !!!!
  There is no 'autotuneMGSAO.dat' file.
  I will set AD-HOC parameters for TriRed.
  I strongly recommend doing auto-tuning.
Auto-tuned :           4           4           8           4           8
Triagonalization Time =      0.798 [sec]
----- Orthogonalization Time =      0.798 [sec] -----
Normal return
Check of error for vectors....
End of checking.

=== Result
=====
n =          100 ,Number of processors =          4
Test Matrix:
Random

Orthogonal Error of ||X^T X - I|| =      5.8215990320764321E-013
Total Calculation Time =      0.862 [sec]

```

6.4 Executing Auto Tuning

6.4.1 Specifying Auto Tuning and Contents of Specification File

Auto tuning in ABCLib_DRSSSED is specified by means of the specification files whose names are determined by each of the library routines.

Below are each of the library routines, and their corresponding specification file names.

- Tridiagonalization routine for real symmetric dense matrices [general version] (ABCLibTriRed):
Specification file: **.ABCLibTriRed**

- Tridiagonalization routine for real symmetric dense matrices [specialized version] (ABCLibTriRedR):
Specification file: **.ABCLibTriRedR**
- Eigenvalue calculation routine for real symmetric dense matrices (ABCLibDRSAllEig):
Specification file: **.ABCLibDRSSEDAIEig**
- Eigenvalue and Eigenvector calculation routine for real symmetric dense matrices (ABCLibDRSAllEigVec):
Specification file: **.ABCLibDRSSEDAIEigVec**
- Routine for solving all Eigenvalues in a hermitian matrix (ABCLibHerAllEig):
Specification file: **.ABCLibHerAllEig**
- Routine for the calculation of all Eigenvalues and Eigenvectors in a hermitian matrix (ABCLibHerAllEigVec):
Specification file: **.ABCLibHerAllEigVec**
- Gram-Schmidt orthogonalization routine (ABCLibQRD):
Specification file: **.ABCLibQRD**

The contents of the specification files are as follows. Note that the specification files should have at least 80 characters of white space, including the text below.

- **-autotune**
Specifies whether to perform auto tuning. Specify "yes" or "no." Performs all implemented auto tuning. To perform all auto tuning, specify "1"; to perform auto tuning in the Householder tridiagonalization module only, specify "2"; to perform auto tuning in the Eigenvector calculation module using Householder inverse transformation only, specify "3"; and to perform auto tuning in the Gram-Schmidt orthogonalization routine only, specify "4."
- **-starttunesize**
Specifies the starting number of matrix dimensions when performing auto tuning.
- **-maxtunesize**
Specifies the ending number of matrix dimensions when performing auto tuning.
- **-tunestr100**
Specifies the additional size for dimensions below 1,000 when performing install-time auto tuning.
- **-tunestr1000**
Specifies the additional size for dimensions over 1,000 when performing install-time auto tuning.

- **-ort**

Specifies the orthogonalization method when calculating Eigenvectors.

The orthogonalization method specified here affects Eigenvector precision and execution time. The orthogonalization types provided in the test version are as follows:

- **MGS**: Orthogonalization using the Modified Gram-Schmidt Method (MGS)
Although this is the most precise method, it cannot be performed in parallel, and is not suited to high-speed processing. This method is recommended if you want to ensure precision without regard to execution time.
- **CGS**: Orthogonalization using the Classical Gram-Schmidt Method (CGS), although MGS is used for PE-internal orthogonalization. Although this method is imprecise, it can be performed in parallel, making it suited to high-speed processing.
- **RB_MGS**: Orthogonalization via the Modified Gram-Schmidt Method, but with the aim of proving suitability to parallel processing through re-distribution in a row-direction distribution. This method was developed independently by the author [A-23]. Although this depends on the performance of the parallel computers, it is capable of performing roughly four times faster than MGS. The precision is identical to MGS.
- **RB_CGS**: Orthogonalization via the Classical Gram-Schmidt Method, but with the aim of proving suitability to parallel processing through re-distribution in a row-direction distribution. This method was developed independently by the author [A-23]. On most parallel computers, this method is faster than CGS. The precision is identical to CGS.
- **HCGS**: Orthogonalization via a combination of the Classical Gram-Schmidt Method (CGS) and Classical Gram-Schmidt Method (CGS).
Although this method has low precision due to the use of the Classical Gram-Schmidt Method, it may be more precise than pure CGS. It can be performed in parallel, making it suited to high-speed processing.
- **IRCGS**: Orthogonalization via the Iterative Refinement Classical Gram-Schmidt Method, which performs the Classical Gram-Schmidt Method twice for improved precision. Although this method has low precision, it may be more precise than pure CGS. It can be performed in parallel, making it suited to high-speed processing.
- **SCGS**: Orthogonalization via the pure Classical Gram-Schmidt Method. This method has low precision, and is less precise than the theoretical CGS method precision. Normally, this method must not be specified for anything but

comparative evaluation and other numerical experiments.

- **NoOrt**: Using this method, no orthogonalization whatsoever is performed. Although this method may destroy precision, it is extremely well suited to parallel processing, making it suited to high-speed execution. This method is recommended if quick calculation is more important than precision.

- **-imp_acc_ort** (extended feature)

This is an attempt to improve orthogonal precision by tweaking the Gram-Schmidt calculation method for re-orthogonalization. This feature is off by default.

- **NO**: Do not apply the orthogonal precision improvement method.
- **SORT**: Attempt to improve the precision of the orthogonalization calculation via the Classical Gram-Schmidt Method by sorting with the inner product as the key. This increases memory usage and calculation time.
- **CMP**: Attempt to improve the precision of the orthogonalization calculation via the Gram-Schmidt Method by guaranteeing the precision of the inner-product calculation via two variables. Applied to both the Classical Gram-Schmidt Method and Modified Gram-Schmidt Method.
- **BOTH**: Using the Classical Gram-Schmidt Method, attempts to improve the precision of the orthogonalization calculation via the Gram-Schmidt Method by sorting with the inner product as the key, and guaranteeing the precision of the inner-product calculation via two variables. Meanwhile, using the Modified Gram-Schmidt Method, identical to CMP.

- **-beo**

Specifies whether to perform pre-execution optimization. Specify "yes" to perform optimization, and "no" to not. Note that pre-execution optimization does not infer parameters via a polynomial equation.

- **-print**

Specifies whether to display internal execution times and other detailed information. Specify "yes" to display verbose information, and "no" to not. The display level can also be specified between 1 and 10. If "yes" is specified, the display level is 1.

- **-heterogeneous** (extended feature)

Specifies whether to adjust the calculation load via static load-distribution mechanism in a heterogeneous PC cluster environment. Specify "yes" or "no." The default is "no."

Below are sample specification files.

```
Example 1: -autotune yes -starttunesize 100 -maxtunesize 8000  
-tunestride100 100 -tunestride1000 1000 -ort MGS -print yes
```

In example 1, install-time auto tuning is performed using matrix-size sample points in increments of 100 from 100 to 1,000 dimensions, in increments of 1,000 past 1,000 dimensions, and up to 8,000 dimensions.

```
Example 2: -autotune no -starttunesize 100 -maxtunesize 8000  
-tunestride100 100 -tunestride1000 1000 -ort MGS -print yes
```

Example 2 executes the library using the MGS orthogonalization method, and displaying the library's internal execution times. It specifies that auto tuning is not to be performed.

6.4.2 Descriptions of Auto-generated Files

The ABCLib_DRSSSED has additional auto-tuning features in three areas: the Householder tridiagonalization module; the Eigenvector calculation module using Householder inverse transformation; and Gram-Schmidt orthogonalization processing.

When auto tuning is performed, files containing information relating to optimized parameters are generated automatically. These are called parameter information files. Parameter information files are generated for each library routine that is a candidate for auto tuning.

These parameter information files are as shown below.

The Householder tridiagonalization module:

- **autotuneTRD.dat**

This file stores parameters relating to the optimum number of matrix-update unrolling levels, number of matrix-vector product unrolling levels, and communication method.

- **TrdUpdLSM.dat**

Coefficient information via the least squares method, relating to the optimum number of unrolling levels for matrix update.

- **TrdMatVecLSM.dat**
Coefficient information via the least squares method, relating to the optimum number of unrolling levels for matrix-vector products.
- **TrdCommLSM.dat**
Coefficient information via the least squares method, relating to the optimum communication method.
- **autotuneTRDana.dat**
This file stores analysis results relating to auto tuning.

Eigenvector calculation module using Householder inverse transformation:

- **autotuneHIT.dat**
This file stores parameters relating to the optimum number of matrix-update unrolling levels and communication method.
- **HITKerLSM.dat**
Coefficient information via the least squares method, relating to the optimum number of unrolling levels for matrix update.
- **HITCommLSM.dat**
Coefficient information via the least squares method, relating to the optimum communication method.
- **autotuneHITana.dat**
This file stores analysis results relating to auto tuning.

Gram-Schmidt orthogonalization processing:

- **autotuneMGSAO.dat**
This file stores parameters relating to the optimum block width and number of unrolling levels.
- **MGSAObLSM.dat**
Coefficient information via the least squares method, relating to the optimum block width.
- **autotuneMGSAOana.dat**
This file stores analysis results relating to auto tuning.

Note that the least squares method can be specified in detail in the file `include/ABCLibLSM.h`.

6.4.3 Executing Install-time Auto Tuning

Install-time auto tuning is optimization that is performed when the library is installed.

Install-time auto tuning is performed by configuring the specification files corresponding to each library routine as described in the preceding section, then calling each routine.

Note that performing install-time auto tuning automatically generates the parameter information files described in the preceding section. The optimum parameters are automatically set by referring to these files. Consequently, it is generally sufficient to perform install-time auto tuning one time after installation[§].

Sample Execution

Below are some sample executions of install-time auto tuning when ABCLibDRSAllEigVec is configured.

```
-ort MGS -autotune yes -starttunesize 128 -maxtunesize 512 -tunestride100 128  
-tunestride1000 1000 -print yes
```

Execute the command below after placing the above file in the execution directory.

```
$ mpirun -np 4 -machinefile nodeinfo ABCLibTestDRSAllEigVec
```

After executing the above command, you will be prompted for the problem size and other information. Input the values shown below.

Note that since the values in ABCLibDRSAllEigVec are used to specify the problem size, start number of Eigenvalues, and end number of Eigenvalues, any value can be entered here. Note that the format of the sample output below may change due to version modifications or the like.

[§]It is also necessary to run install-time auto tuning again if the number of processors used changes, the communications network or other hardware configuration changes, or the like.

```

Problem size >
100
Start number of eigenvectors >
2
End number of eigenvectors >
2
ncelx=
2
ncely =
2

```

After input is complete, the following log is displayed, and the program enters auto-tuning mode.

```

!!!! Warning !!!!
There is no 'autotuneTRD.dat' file.
I will set AD-HOC parameters for TriRed.
I strongly recommend doing auto-tuning.
!!!! Warning !!!!
There is no 'autotuneHIT.dat' file.
I will set AD-HOC parameters for HIT.
I strongly recommend doing auto-tuning.
===== AUTO Tuning Mode =====
-----
ABCLib ver. 1.00
Composed by T. Katagiri, October 2004
-----
=== Auto-tuning for Tridiagonalization ===
-----
ABCLib ver. 1.00
Composed by T. Katagiri, October 2004
-----

Now start tuning...
          1          1 Initial switches:          0          8
          6

```

n=	128	Time=	0.5255729999999992	iupdate_sw=	1
n=	128	Time=	0.2111440000000004	iupdate_sw=	2
n=	128	Time=	0.1672940000000001	iupdate_sw=	3
n=	128	Time=	0.1705660000000000	iupdate_sw=	4
n=	128	Time=	0.1708849999999993	iupdate_sw=	5
n=	128	Time=	0.1729319999999990	iupdate_sw=	6
n=	128	Time=	0.1736940000000011	iupdate_sw=	7
n=	128	Time=	0.1667370000000008	iupdate_sw=	8
n=	128	Time=	0.1737360000000008	iupdate_sw=	9
n=	128	Time=	0.1685940000000001	iupdate_sw=	10
n=	128	Time=	0.1730620000000007	iupdate_sw=	11
n=	128	Time=	0.1696140000000010	iupdate_sw=	12
n=	128	Time=	0.1680330000000017	iupdate_sw=	13
n=	128	Time=	0.1737040000000021	iupdate_sw=	14
n=	128	Time=	0.1699410000000019	iupdate_sw=	15
n=	128	Time=	0.1740300000000010	iupdate_sw=	16

....

After all auto tuning is complete, the following file is automatically generated.

```

$ ls
ABCLibTestDRSA11EigVec      autotuneTRDana.dat      autotuneHITana.dat
autotuneTRDana_tmp.dat     MGSAOb1LSM.dat         autotuneHITana_tmp.dat
autotuneTRD.dat            nodeinfo                autotuneHIT.dat
TrdCommLSM.dat             autotuneMGSAOana.dat   TrdMatVecLSM.dat
autotuneMGSAOana_tmp.dat   HITCommLSM.dat         TrdUpdLSM.dat
autotuneMGSAO.dat          HITKerLSM.dat

```

Here, the contents of parameter information file autotuneTRD.dat, which stores the auto-tuning results for the tridiagonalization routine (optimum parameters) are as follows:

```

128    1    8    8
256    0    8    3
384    0    7    4
512    0    4    3
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6
-1    0    8    6

```

The contents of the file autotuneMGSAOana.dat, which stores the results of the auto-tuning log analysis for the Gram-Schmidt orthogonalization routine, are as follows:

=== ABCLib_QRD Tuning Log Analysis Result

=====

Number of PE: 4/Tuning Time: 104.148 [Sec.]

=====

ProblemSize	Best [Sec.] (Parameter)	Worst [Sec.] (Parameter)	C/B	W/B
128	0.0250(4, 4, 8, 4, 8)	0.0798(1, 4, 8, 4, 8)		
	0.0245(5, 1, 3, 4, 8)		1.022	3.255
256	0.0940(4, 4, 8, 4, 8)	0.0988(1, 4, 8, 4, 8)		

	0.0937(5, 1, 1, 3, 2)	1.003	1.055
384	0.2083(4, 4, 8, 4, 8)	0.2173(1, 4, 8, 4, 8)	
	0.2042(5, 2, 5, 4, 5)	1.020	1.064
512	0.3729(4, 4, 8, 4, 8)	0.4359(1, 4, 8, 4, 8)	
	0.3664(6, 2, 4, 3, 6)	1.018	1.190

6.4.4 Executing Pre-execution Auto Tuning

Pre-execution auto tuning is a more precise form of optimization than install-time auto tuning, which is performed when the problem size and coefficient matrix information are fixed.

Below is a sample pre-execution auto tuning.

```
Example 3: -autotune yes -starttunesize 1234 -maxtunesize 1234 -tunestriderange 100 100
           -tunestriderange1000 1000 -ort MGS -beo yes -print yes
```

In example 3, both `-starttunesize` and `-maxtunesize` are set to 1,234 dimensions, so only the 1,234 parameter is optimized. After this auto-tuning is performed, specifying 1,234 dimensions at execution sets the non-inferred optimum parameters.

This should allow the library to execute faster than when optimization was performed using install-time auto tuning.

7. Future Plans

This section briefly describes the future plans of the ABCLIB_DRSSSED development project.

- Addition of blocked tridiagonalization routine:

The test version does not have a tridiagonalization routine that uses a blocking algorithm. On PC clusters and other computers having hierarchical memory, block algorithms are the only way to improve calculation efficiency. The project plans to implement a blocking algorithm, and add an auto-tuning feature.

- Improvement of auto-tuning method:

The test version uses a polynomial equation as an evaluation function (cost-definition function), and optimizes parameters via the least squares method. For this reason, parameter inference precision may not always be adequate. The project aims to further improve parameter inference precision by adopting a more precise evaluation function.

- Integration with sparse-matrix recursion method:

This test version uses similarity transformation as a direct solution for dense matrices, and a recursive solution for Eigenvector calculation. It is known that for sparse matrices, it is more efficient for recursive solutions to be used for all processes. Additionally, it is known that for band matrices, it is more efficient to use a similarity transformation for band matrices. Thus, the project aims to further improve speed and usability by researching and developing auto-tuning features (pre-execution and run-time) that automatically detect the shape of non-zero elements in the input matrix, and automatically select the solution method.

- Compatibility with existing libraries:

The project plans to further improve usability and performance by providing interfaces similar to such packages as Basic Linear Algebra Subprograms (BLAS) and ARPACK, using them as auto-tuning commands within ABCLib.

Conclusion

This library has described the usage of the test version of ABCLib_DRSED. Please see <http://www.abc-lib.org/> for future information about the ABCLib project.

References

- [A-1] Brewer, A.E., Portable High-Performance Supercomputing: High-Level Platform-Dependent Optimization, Ph.D Thesis, Massachusetts Institute of Technology (1994)
- [A-2] Bilmes, J., Asanovic, K, Chin, C.-W. and Demmel, J.: Optimizing Matrix Multiply using PHiPAC: a Portable, High-Performance, ANSI C Coding Methodology, Proceedings of International Conference on Supercomputing 97, pp.340–347 (1997)
- [A-3] 高田広章：特集「組み込みシステム開発の現状」、情報処理、Vol.38、No.10、pp.870—903 (1997)
- [A-4] 黒田久泰、片桐孝洋、佃良生、金田康正：自動チューニング機能付き並列数値計算ライブラリ構築の試み 対称疎行列用の連立一次方程式ソルバを例にして、情報処理学会第57回全国大会講演論文集(1)、pp.1-10–1-11 (1998)
- [A-5] Frigo, M.: A Fast Fourier Transform Compiler, Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation, Atlanta, Georgia, pp.169–180 (1999)
- [A-6] Whaley, R., Petitet, A. and Dongarra, J.J.: Automated Empirical Optimizations of Software and the ATLAS project, Parallel Computing, Vol.27, pp. 3–35 (2001)
- [A-7] 片桐孝洋、黒田久泰、大澤清、工藤誠、金田康正：自動チューニング機構が並列数値計算ライブラリに及ぼす効果、情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム、Vol.42、No.SIG 12 (HPS 4)、pp. 60–76 (2001)
- [A-8] 直野健、山本有作：単一メモリ型インタフェースを有する自動チューニング並列ライブラリの構成方法、情報処理学会研究報告、No. 2001-HPC-87、pp.25–30 (2001)
- [A-9] Ribler, R.L., Simitci, H. and Reed, D.A.: The Autopilot Performance-Directed Adaptive Control System, Future Generation Computer Systems, Special Issue (Performance Data Mining), Vol.18, No.1, pp. 175–187 (2001)
- [A-10] Kuroda, H., Katagiri, T., and Kanada, Y.: Knowledge Discovery in Auto-tuning

Parallel Numerical Library, Progress in Discovery Science, Final Report of the Japanese Discovery Science Project. Lecture Notes in Computer Science 2281 Springer 2002, ISBN 3-540-43338-4, pp.628–639 (2002)

[A-11] Tapus, C., Chung, I.-H. and Hollingsworth, J. K.: Active Harmony : Towards Automated Performance Tuning, Proceedings of High Performance Networking and Computing (SC2002), Baltimore, USA (2002)

[A-12] 片桐孝洋、吉瀬謙二、本多弘樹、弓場敏嗣：実行起動前最適化層を有する自動チューニングソフトウェア構成方式の提案、2003年先進的計算基盤システムシンポジウム (Symposium on Advanced Computing Systems and Infrastructures (SACSIS)、SACSIS 2003 論文集、pp.159—160 (2003)

[A-13] Katagiri,T., Kise,K., Honda,H., and Yuba,T., : FIBER: A Framework of Installation, Before Execution-invocation, and Run-time Optimization Layers for Auto-tuning Software, 電気通信大学情報システム学研究科技術報告, UEC-IS-2003-3 (2003)

[A-14] 片桐孝洋、吉瀬謙二、本多弘樹、弓場敏嗣：FIBER：汎用的な自動チューニング機能の付加を支援するソフトウェア構成方式、第94回ハイパフォーマンスコンピューティング (HPC) 研究会、平成15年6月13日 (金) 情報処理学会研究報告 2003-HPC-94, pp. 1—6 (2003)

[A-15] Katagiri, T., Kise K., Honda, H., and Yuba,T.,: FIBER: A General Framework for Auto-Tuning Software, Springer LNCS 2858, pp.146–159, The Fifth International Symposium on High Performance Computing (ISHPC-V), Tokyo Fashion Town Building, Tokyo International Trade Center (Odaiba, Tokyo, JAPAN), October 20-22 (2003)

[A-16] 今村俊幸、直野健：性能安定化を目指した自動チューニング型固有値ソルバーについて、先進的計算基盤システムシンポジウム SACSIS 2003 論文集、pp.145—152, (2003)

[A-17] 須田礼仁：ERXPP-数値ライブラリにより並列計算性能を簡易かつ適応的に引き出す方式の提案、情報処理学会研究報告、2003-HPC-96、pp.19—24 (2003)

[A-18] 高田広章：組み込みシステム開発の現状と課題、先進的計算基盤システムシンポジ

ウムSACSIS2003チュートリアル資料 (2003)

[A-19] Katagiri,T., Kise,K., Honda,H., and Yuba,T.: Effect of Auto-tuning with User's Knowledge for Numerical Software, Proceedings of ACM Computing Frontiers (CF) 04, pp.12–25, Island of Ischia, Italy, 14–16 (2004)

[A-20] Cuenca, J., Gimenez, D., and Gonzalez, J.: Architecture of an Automatically Tuned Linear Algebra Library, Parallel Computing, Vol.30, pp.187–210 (2004)

[A-21] 今村俊幸、直野健：キャッシュ競合を制御する性能安定化機構内蔵型数値計算ライブラリについて、ハイパフォーマンスコンピューティングと計算科学シンポジウムHPC S2004論文集、pp.173–180 (2004)

[A-22] 直野健、今村俊幸、恵木正史：GRIDコンピューティング環境における行列ライブラリ向け性能保証方式の検討、ハイパフォーマンスコンピューティングと計算科学シンポジウムHPC S2004論文集、pp.51–58 (2004)

[A-23] 石井良規、片桐孝洋、本多弘樹、弓場敏嗣：Autopilotを用いた疎行列ソルバにおける実行時自動チューニング機構の設計、電子情報通信学会総合大会論文集、D-3-9、pp. 28、(2004)

[A-24] Eijkhout,V., Fuentes,E: Statistical Techniques for Algorithm Ranking In Self-Adapting Numerical Software, Eleventh SIAM Conference on Parallel Processing for Scientific Computing (PP04), Hyatt at Fisherman's Wharf, San Francisco, CA, USA, February 25, 2004, Organized Session of MS22, Multimethod Sparse Solvers for Large Scale Simulations (2004)

[A-25] 渡辺政彦：組み込みソフトウェア向け開発支援環境、情報処理、Vo.45、No.1、pp.10–15 (2004)

[A-26] 片桐孝洋、吉瀬謙二、本多弘樹、弓場敏嗣：自動チューニング処理記述用ディレクティブ ABCLibScript の設計と実装、2004年先進的計算基盤システムシンポジウム (Symposium on Advanced Computing Systems and Infrastructures (SACSIS)、SACSIS2004論文集、pp.43–52 (2004)、および、自動チューニング処理記述用ディレクティブ ABCLibScript、電気通信大学情報システム学研究科技術報告、UEC-IS-2004-1 (2004) :

[A-27] 片桐孝洋、吉瀬謙二、本多弘樹、弓場敏嗣：自動チューニング処理記述用ディレクティブ ABCLibScript の設計と実装、2004年先進的計算基盤システムシンポジウム (Symposium on Advanced Computing Systems and Infrastructures (SACSIS)、SACSIS 2004 論文集、pp.43—52 (2004)

[A-28] 直野健、恵木正史：GRIDコンピューティング環境下での行列ライブラリにおける性能保証ライン算出の改良型アルゴリズム、2004年先進的計算基盤システムシンポジウム (Symposium on Advanced Computing Systems and Infrastructures (SACSIS)、SACSIS 2004 論文集、pp.295—304 (2004)

[A-29] 片桐孝洋、吉瀬謙二、本多弘樹、弓場敏嗣：ユーザ知識を活用するソフトウェア自動チューニングについて、第 回 ハイパフォーマンスコンピューティング (HPC) 研究会、平成 15 年 8 月 1 日 (金)、情報処理学会研究報告 2004-EVA-10、pp.19—24 (2004)

[A-30] マイクロソフト SQL Server 2000 :
<http://www.microsoft.com/japan/sql/default.msp>

[A-31] 東京大学 生産技術研究所 桑原研究室：交通流シミュレーション、ネットワークシミュレーションの容量パラメタの自動調整法：<http://www.transport.iis.u-tokyo.ac.jp/>

[A-32] 電子情報通信学会、ディペンダブルコンピューティング研究会、
<http://www.ieice.org/iss/dc/jpn/>

[A-33] 片桐孝洋、吉瀬謙二、本多弘樹、弓場敏嗣：データ再分散を行う並列 Gram-Schmidt 再直交化、情報処理学会論文誌：コンピューティングシステム、Vol.45、No. SIG 6 (ACS 6)、pp.75—85 (2004) :

[A-34] 直野健、猪貝光祥：マルチカラー逆反復法による直交化法とその性能、日本応用数理学会年会、オーガナイズドセッション：数値線形代数、中央大学後楽園キャンパス、2004年9月16日～18日

Projects:

[B-1] PHiPAC プロジェクト ; <http://www.icsi.berkeley.edu/~bilmes/philpac/>

[B-2] ATLAS プロジェクト ; <http://www.netlib.org/atlas/index.html>.

[B-3] ILIB プロジェクト (HINTS プロジェクト); <http://www.hints.org/>

[B-4] ABCLib プロジェクト ; <http://www.abc-lib.org/>

[B-5] FIBER プロジェクト ;

http://www.yuba.is.uec.ac.jp/~katagiri/FIBER_Prj/FIBER-Prj.html

[B-6] Autopilot プロジェクト :

<http://www-pablo.cs.uiuc.edu/Project/Autopilot/AutopilotOverview.htm>

[B-7] Active Harmony プロジェクト : <http://www.dyninst.org/harmony/>

[B-8] SANS プロジェクト ; <http://icl.cs.utk.edu/sans/index.html>

[B-9] SALSA プロジェクト ; <http://icl.cs.utk.edu/salsa/>

[B-10] BeBOP プロジェクト ; <http://bebop.cs.berkeley.edu/>

[B-11] Sparsity プロジェクト ; <http://www.cs.berkeley.edu/~yelick/sparsity/>

Patents:

[C-1] 片桐孝洋 : プログラム、記録媒体およびコンピュータ、 日本国特許出願、
特願 2003-022792 (平成 15 年 1 月 30 日)

[C-2] 片桐孝洋: 計算装置、計算方法、プログラムおよび記録媒体、 日本国特許出願、
特願 2003-092592 (平成 15 年 3 月 28 日)。

[C-3] 片桐孝洋 : 計算装置、計算方法、プログラムおよび記録媒体、日本国特許出願、特願
2003-149701、平成 15 年 5 月 27 日、(特願 2003-92592 の国内優先
権出願)