ABCLib_DRSSED: A Parallel Eigensolver with an Auto-tuning Facility

Takahiro Katagiri [a,b,*], Kenji Kise [a,b] Hiroki Honda [a]
Toshitsugu Yuba [a]

[a] *Graduate School of Information Systems, The University of Electro-Communications*

[b] *Japan Science and Technology Agency, PRESTO*
1-5-1 Choufu-gaoka, Choufu-shi, Tokyo 182-8585, JAPAN
Phone: +81-424-43-5642 FAX: +81-424-43-5644

Abstract

Conventional auto-tuning numerical software has the drawbacks of (1) fixed sampling points for the performance estimation; (2) no function to adapt to heterogeneous environments. To solve these drawbacks, we developed ABCLib_DRSSED, which is a parallel eigensolver with an auto-tuning facility. ABCLib_DRSSED has (1) functions for the sampling points, which consist of an end-user interface and new auto-tuning optimization timing, called Before Execute-time Optimization (BEO); (2) a load-balancer for the data to be distributed. In our performance evaluation of the BEO, we obtained speedup factors from 10% to 90%, and 340% in the case of a failed estimation. In addition, in our evaluation of the load-balancer, we could improve the performance by 220%.

Key words: ABCLib, Auto-Tuning Facility, FIBER, Sampling Point, Load-Balancer

* Corresponding author.
Email addresses: katagiri@is.uec.ac.jp (Takahiro Katagiri), kis@is.uec.ac.jp (Kenji Kise), honda@is.uec.ac.jp (Hiroki Honda), yuba@is.uec.ac.jp (Toshitsugu Yuba).

[1] This paper is IS Technical Reports UEC-IS-2004-8, Graduate School of Information Systems, The University of Electro-Communications, in 3rd December of 2004. This paper is also submitted to Journal of Parallel Computing.

1 Introduction

Recently, many numerical libraries with an "auto-tuning facility" have been developed, e.g., PHiPAC [3], ATLAS [1], FFTW [9], and I-LIB [15]. We refer to a library with an auto-tuning facility as a *SATF* (*Software with Auto-Tuning Facility.*) Early research on these libraries established the effectiveness of SATFs [14]. Although SATFs are generally effective from the viewpoint of performance, there are following two performance problems with the numerical auto-tuning software.

First, all conventional numerical SATFs need fixed values for the sampling points to estimate their execution behavior. As an example, the dimensions for the input matrices are needed. If the SATF developer chooses the wrong sampling points, the performance may be unstable except for the sampling points. Unfortunately, conventional SATFs have no function enabling the end-user to modify the sampling points. To resolve this issue, we supply an end-user interface for specifying the sampling points. In addition, we also supply a new auto-tuning timing, named Before Execution-time Optimization, to make sure the auto-tuning uses end-user specified sampling points.

Second, SATFs need to be adapted to heterogeneous environments, which are increasing due to the progress of the PC cluster and GRID technologies. In these environments, conventional methods for a numerical library should be modified by taking into account the characteristics of the heterogeneous parallel environments. The key issue for the numerical library is how to distribute the input data for the matrices. To resolve this issue, we need a load-balancer for numerical libraries as part of the auto-tuning facility. The idea for a load-balancer in a numerical library is not new. For example, a load-balancer can be built as an extension of the ScaLAPACK library[5]. Load-balancing for dense linear algebra kernels has been studied in [2]. We emphasize, however, that the idea for a load-balancing function in auto-tuning is a new approach. The focus on heterogeneous environments in this paper is limited to either of the following: the same arithmetic architectures with different execution speeds, i.e., CPU clocks, or an environment with different system loads, but with the same CPU clocks. These situations are easily caused by multiple users in the system.

The organization of this paper is as follows: Section 2 explains the ABCLib basics, which are the design policy, functions, and FIBER auto-tuning framework. Section 3 explains the ABCLib_DRSSED details, such as the supplied functions, software architecture, and end-user interface. In Section 4, the details for the auto-tuning facility are described. In Section 5, we evaluate the auto-tuning facility by using several kinds of parallel machines. Section 6 explains related work. Finally, Section 7 summarizes the observations of this

study.

2 ABCLib Library

2.1 Design Policy

ABCLib (*Automatically Blocking-and-Communication adjustment Library*) is a parallel numerical library with an auto-tuning facility.

ABCLib is a parallel numerical library for attaining high performance in several parallel machines, including those with hierarchical memory architectures as processing elements (PE). We apply the library to both PC clusters, which consist of low-cost PCs through low-cost networks, and to supercomputers.

The design policy is listed as follows.

- *Blocked Algorithm*: The blocked algorithm is adapted to attain high performance for computers, including those with hierarchical memory architectures.
- *Hiding Communication Latency*: Hiding implementations for communication latency are applied for low-performance networking environments.
- *Auto-tuning Facility*: An auto-tuning facility is adapted by the library to establish high performance in several kinds of machine environments.
- *Less Argument Interface*: A smaller number of arguments for the library interface are needed. This makes the interface easier to use and avoids a decrease in performance if the end-user specifics the wrong parameters.

2.2 Functions to be Supplied

The plan to develop subroutines in the ABCLib project is shown below.

- For dense matrices:
  - Direct solver for liner equations.
  - Direct solver for eigenvalue problems.
  - Iterative solver for eigenvalue problems.
- For sparse matrices:
  - Direct solver for linear equations.
  - Iterative solver for linear equations.
  - Iterative solver for eigenvalue problems.
- Tools for auto-tuning facility addition:

  - *ABCLibScript*: A directive to support easy construction of the auto-tuning facility based on the FIBER auto-tuning framework.
  - *ABCLibCodeGen*: A pre-processor to process the directive of ABCLibScript.
- Benchmark Software:
  - *ABCLibBench*: Benchmark software using the auto-tuning facility in ABCLib.

2.3 The FIBER Auto-Tuning Framework

For the auto-tuning facility, we apply the FIBER framework [13,12] to ABCLib. The FIBER Framework has three kinds of optimization timings: Install-time, Before Execute-time, and Run-time Optimizations.

In addition, there are two kinds of system parameters to simplify the auto-tuning process. The system parameters are called the *Basic Parameter* (*BP*) and *Performance Parameter* (*PP*). The BP is a fixed parameter for optimizing the PP. The PP is a target parameter for optimizing. By using these parameters, we can define the auto-tuning process—that is, the minimizing of a function by varying the parameters of PP with the fixed parameters of BP. The function is called **Cost Definition Function** in the FIBER framework.

3 ABCLib_DRSSED Library Details

3.1 Supplied Functions

Currently, ABCLib_DRSSED supports the following routines.

- `ABCLibDRSAllEigVec(A, n, eig, X, ms, me)`
  - `Function`: An arbitrary number of eigenvalues and eigenvectors are calculated for real symmetric matrices.
  - `A`: Matrix coefficients (1:NdivP, 1:NdivP), which are distributed by cyclic-cyclic distribution, where NdivP is the amount of distributed data.
  - `n`: The dimension of the matrix.
  - `eig`: Calculated eigenvalues (1:NdivP), which are distributed in a blocked manner.
  - `X`: Calculated eigenvectors (1:n, 1:NdivP), which are distributed in a row-wise blocked manner.
  - `ms`: The first eigenvalue number to compute (the count starts from the largest absolute value.)

  - `me`: The last eigenvalue number to compute (the count starts from the largest absolute value.)

- `ABCLibHerAllEigVec(AR, AI, n, eig, X)`:
  - `Function`: An arbitrary number of eigenvalues and eigenvectors are calculated for Hermitian matrices.
  - `AR`: Coefficients of the real part of the Hermitian matrix (1:n,1:n) which is not distributed.
  - `AI`: Coefficients of the imaginary part of the Hermitian matrix (1:n,1:n) which is not distributed.
  - `n`: The dimension of the matrix.
  - `eig`: Calculated eigenvalues (1:2*NdivP), which are distributed in a blocked manner.
  - `X`: Calculated eigenvectors (1:2*n, 1:2*NdivP-1), which are distributed in a row-wise blocked manner.

- `ABCLib_QRD(X, n)`
  - `Function`: Input vectors are orthogonalized. This means QR decomposition is performed for the input matrix.
  - `X`: Input matrix-formed vectors to be orthogonalized are distributed in a row-wise blocked manner (1:n, 1:NdivP). The orthogonalized vectors are updated in this matrix.
  - `n`: The dimension of the matrix.

3.2 Software Architecture

Figure 1 shows the software architecture of ABCLib_DRSSED.

As shown as Figure 1, ABCLib_DRSSED has two kinds of auto-tuning layers and one system layer.

- **Run-time Auto-tuning Layer**: This is the part of the auto-tuning process when the library runs. The time for the object of auto-tuning is measured at run-time, and the best parameter is selected or estimated.
- **Static Auto-tuning Layer**: This is the part of the auto-tuning process when the library is installed or before the library is invocated. The time for the object of auto-tuning is measured, and the best parameter is selected or estimated. The computation to estimate the execution time is performed. Then, the data is saved into a file named the Parameter Information File. The effect of auto-tuning is also analyzed, and the report is stored in a file named the Parameter Analysis File.
- **Parameter Selection Layer**: This part selects or estimates the best parameter. This layer is a kind of run-time routines. The layer selects the best parameter based on the measured or estimated parameter information in

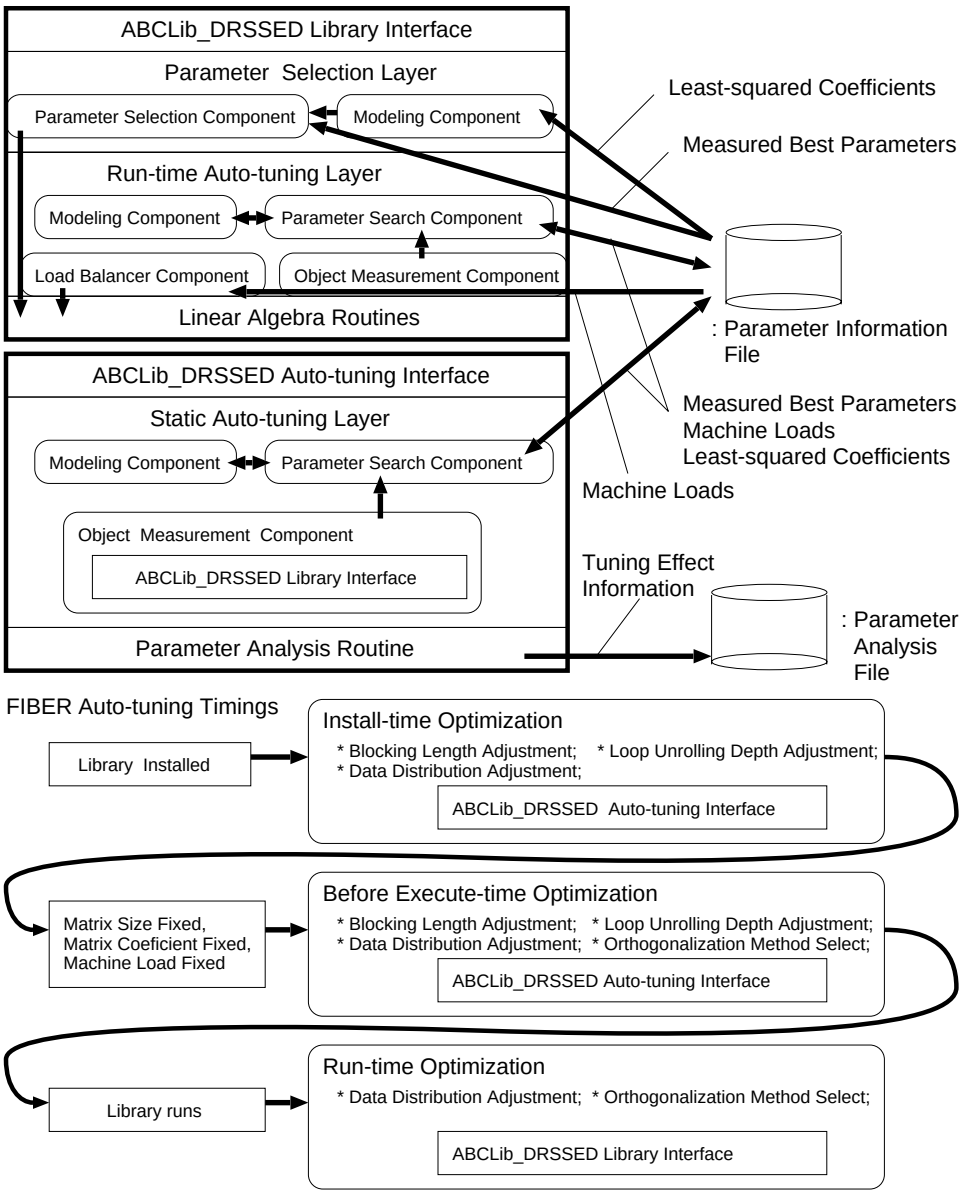

Fig. 1. Software Architecture in ABCLib_DRSSED.

the Parameter Information File.

The Run-time and Static Auto-tuning Layers in Figure 1 have three kinds of components. They are the **Object Measure Component**, **Parameter Search Component**, and **Modeling Component**. These components differ between the Run-time and Static Auto-tuning Layers in that the Object Measurement Component in the Run-time Auto-tuning Layer does not call the ABCLib_DRSSED library interface. This is because the target parameters for the Run-time Auto-tuning Layer are obvious and can be specified directly at run-time.

The Object Measure Component measures the execution time of the target.

The time data is sent to the Parameter Search Component. The Parameter Search Component determines the search method for the target parameters to optimize. The search method is also determined by the sampling point from the end-users. Finally, the execution behavior is modeled based on the information from the Parameter Search Component. The order for the modeling formula from the end-user is also used in the Modeling Component.

In the Run-time Auto-tuning Layer, there is the **Load Balancer Component**. This component determines the amount of the distributed data based on the machine load information in the Parameter Information File. The amount is sent to the Linear Algebra Routines.

From the end-user's point of view, auto-tuning is performed in the three kinds of timings in Figure 1 based on the FIBER framework. The end-user in the FIBER framework is the **system administrator** or the **library user**.

First, Install-time Optimization is performed by invocating the ABCLib_DRSSED Auto-tuning Interface, when the library is installed. The main purposes of this optimization are the adjustments of the blocking length and loop unrolling depth. If the end-user is in a heterogeneous computer environment and the performance is permanently stable, the adjustment of the data distribution amount can be an optimization target. The end-user in the Install-time Optimization is the system administrator.

Second, Before Execute-time Optimization is performed by invocating the ABCLib_DRSSED Auto-tuning Interface, when the system parameters are defined by the end-user. For example, the matrix dimension to execute is a system parameter. The main purposes of this optimization are the adjustments of the blocking length and loop unrolling depth. If the end-user knows that the performance in the computer environment is stable when the library executes in this timing, the adjustment of the data distribution amount can be an optimization target. If the end-user knows that the coefficients of the input matrix do not change whenever the library calls, the selection of the orthogonalization methods can be an optimization target. The end-user in the Before Execute-time Optimization is a library user.

Finally, Run-time Optimization is performed by invocating the ABCLib_DRSSED Library Interface at run-time. The main purposes of this optimization are the adjustments of the data distribution amount. Most loads in the computer environment cannot be fixed before the library runs. The optimization, hence, is needed to adjust the data distribution amount based on the machine loads at run-time. If the coefficients of the input matrix are fixed at run-time, the selection of the orthogonalization method is also a choice of optimization. However, the function is not implemented in the current version.

The use of the above three optimization timings for auto-tuning is a typical

scenario in the FIBER framework.

3.3 End-user Interface

3.3.1 Run-time Options

The end-user can specify the run-time options in a file named the Specification File. For example, `.ABCLibDRSSEDAllEigVec` is the file which specifies the run-time options for the `ABCLibDRSAllEigVec` routine.

The system administrator can specify the auto-tuning in Install-time with sampling points for the matrix dimensions, which range from 128 to 2048 stridden by 128, as follows:

```
-autotune yes       -starttunesize 128    -maxtunesize 2048
-tunestride100 128  -tunestride1000 1028
```

If the library user would like to specify the Before Execute-time Optimization in the dimension of 1234, the library user can specify the following:

```
-autotune yes       -starttunesize 1234   -maxtunesize 1234
-tunestride100 128  -tunestride1000 1028  -beo yes
```

If the library user would like to invocate the load-balancer in Before Execute-time, the library user can write the options as follows:

```
-autotune yes       -starttunesize 128    -maxtunesize 2048
-tunestride100 128  -tunestride1000 1028  -beo yes
-heterogeneous yes
```

Finally, the auto-tuning is finished. To execute the library with tuned parameters, the library user can specify the options as follows:

```
-ort CGS            -autotune no
-starttunesize 128  -maxtunesize 2048  -tunestride100 128
```

In this case, the re-orthogonalization method in the inverse iteration method is selected as the classical Gram-Schmidt method.

3.3.2 Tuned Parameter Analyzer

In ABCLib_DRSSED, an analyzing function for auto-tuned parameters is included. The function analyzes the effect of auto-tuning, and stores the result in the Parameter Analysis File.

For example, the effect of auto-tuning for `ABCLib_QRD`, which is a QR routine in ABCLib_DRSSED, is stored in the file `autotuneMGSAOana.dat`. The contents of the file are as follows:

```
=== ABCLib_QRD Tuning Log Analysis Result
=========================================
Number of PE: 4 / Tuning Time: 104.148 [s]
=========================================
ProblemSize | Const [s] (Parameter) | Worst [s] (Parameter)
              | Best [s] (Parameter)  s| 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
```

The above output shows the effect of auto-tuning from the viewpoint of speedups to execution with the defaults, and the worst parameters to execution with the best parameter.

4 Auto-tuning Facilities

4.1 The Method for Optimization

Although we can choose several methods for optimizing the parameters in the FIBER framework, we adapt the least-squares method with QR decomposition[7] with a partial pivoting to estimate the parameters because of the easy implementation.

The formula for estimating the execution time—we call this formula the *Cost Definition Function* in the FIBER Framework—is the k-th order linear polynomial function in the implementation of ABCLib_DRSSED.

4.2 The Method for Parameter Estimation

Let Φ be a set of sampling points. There are two sampling points for the FIBER framework. They are the sampling points for BP and PP. The target sampling points for BP are the dimensions for the input matrix. We denote

the set of sampling points as Φ_n. The target sampling points for PP are the depth of the unrolling and the width of the blocking size. We denote the set for p_i, which is the i-th parameter of PP, as Φ_{p_i}.

The i-th argument of the set Φ is denoted as ϕ_i. We also denote the number of arguments for the set as $|\cdot|$. For example, the number of arguments for the set of Φ_n is $|\Phi_n|$.

Figure 2 shows the method for the parameter estimation in ABCLib_DRSSED. The routine of $LSM(x, y)$ in Figure 2 returns estimated parameters in the area of x by the least-squares method with the measured time of y. The formula for this estimation is a linear polynomial function, whose order is specified by the library developer. The routine of $LSM(x, y)$ also calculates and stores the coefficients of the formula into the Parameter Information File. The implemented routine of Figure 2 forms a part of the Modeling Component in Figure 1.

```
⟨1⟩  do kp=1, Σ_i |Φ_{p_i}|
⟨2⟩    do n_loop=1, |Φ_n|
⟨3⟩      do l_loop=1, |Φ_{p_kp}|
⟨4⟩        t = Measured Executing Time of the Target Routine with
                 the Dimension φ_{n_loop} and the Parameter φ_{p_kp l_loop}.
⟨5⟩        x(l_loop) = l_loop;
⟨6⟩        y(l_loop) = t;
⟨7⟩      enddo
c        === Estimation of execution time for the parameters of p_kp
c        ===   with the fixed sampling points of φ_{n n_loop}.
⟨8⟩      if (|Φ_{p_kp}| .eq. The Number of All Definition Area for p_kp) then
⟨9⟩        Parameter = the measured best in ⟨3⟩ – ⟨7⟩.
⟨10⟩       xx(n_loop, *) = φ_{n n_loop};   yy(n_loop, *) = y(*);
⟨11⟩     else
⟨12⟩       Parameter = LSM(x, y);
⟨13⟩       do i=1, The Number of All Definition Area for p_kp
⟨14⟩             xx(n_loop, i) = φ_{n i};
⟨15⟩       enddo
⟨16⟩       yy(n_loop, *) = The Estimated Values in the LSM(x, y);
⟨17⟩     endif
⟨18⟩   enddo
c      === Estimation of execution time for the parameters of all area for n.
⟨19⟩   do l_loop=1, The Number of All Definition Area for p_kp
⟨20⟩     call LSM(xx(1, l_loop), yy(1, l_loop));
⟨21⟩   enddo
⟨22⟩ enddo
```

Fig. 2. Method for parameter auto-tuning in `ABCLib_DRSSED`.

In Figure 2, there are two phases to estimate the parameters: (1) Estimation of PP with $\Phi_{p_{kp}}$ for a fixed BP parameter (in this case, the dimension n of the matrix) in $\langle 2 \rangle$–$\langle 18 \rangle$; (2) Estimation of time in all areas of the BP parameter (in this case, the dimension n of matrix) in $\langle 19 \rangle$–$\langle 21 \rangle$;

In Phase (1), if the sampling points for PP are the same as the number of all the definition areas of the parameter p_{kp}, the measured best parameter is chosen (See $\langle 9 \rangle$ in Figure 2.) Otherwise, a parameter is estimated by the least-squares method (See $\langle 12 \rangle$ in Figure 2.)

In Phase (2), since the all sampling points for the parameter p_{kp} are measured or estimated by the least-squares method, the execution time for varying the dimension fixed by the parameter of p_{kp} is estimated. This estimation is performed by all definition areas of p_{kp}. For this reason, all the best parameters for p_{kp} can be estimated in the arbitrary dimension of n, which is specified by the library user at run-time.

Finally, the library user specifies the dimension at run-time of the library. The best parameter is estimated by using the method in Figure 3. The routine $EstLSM(n, p)$ in Figure 3 estimates the best parameters for the parameter of p_{in} in the dimension of n. In this estimation, the k-th order linear polynomial with calculated coefficients is used. The order of k is specified by the library developer. The coefficients are stored in the Parameter Information File. The implemented routine of Figure 3 is part of the Modeling Component in the Parameter Selection Layer in Figure 1.

```
c   === Library user specifies the dimension n_in for the parameter p_in.
⟨1⟩  if (n_in ∈ Φ_n) then
⟨2⟩    Parameter = the Measured Best or Estimated Parameter for p_in,
                   which is stored in the Parameter Information File.
⟨3⟩  else
⟨4⟩    Parameter = EstLSM(n_in, p_in)
⟨5⟩  endif
```

Fig. 3. Method for the estimated parameter in `ABCLib_DRSSED`.

As shown as Figure 3, if the library user specifies the measured sampling dimensions, the system sets the best parameter. If the library user specifies a dimension not sampled, the system estimates the best parameter using the least-squares method. In this estimation, the calculated coefficients in the routine of Figure 2 are used.

Figure 4 shows the measurement part of the machine loads. Figure 4 forms a part of the Object Measurement Component in Figure 1.

⟨1⟩ t_i = Measured Time with a Benchmark Program in PE i.
⟨2⟩ T = Summation of t_i $(i = 1, 2, ..., p)$ and share the result;
⟨3⟩ $R_i = t_i$ / T;
⟨4⟩ Store the R_i $(i = 1, 2, ..., p)$ to the Parameter Information File;

Fig. 4. Measurement part for the machine loads in `ABCLib_DRSSED`.

Figure 5 shows the data distribution length decision part for the load-balancer. Figure 5 forms a part of the Load-balancer Component in Figure 1.

c === Give the data length of n. The following is for PE i process.
⟨1⟩ Load R_i $(i = 1, 2, ..., p)$ from the Parameter Information File;
⟨2⟩ $iend_0 = 0$;
⟨3⟩ **do** $k = 1, p$
⟨4⟩ $istart_k = iend_{k-1} + 1$;
⟨5⟩ $iend_k = istart_k + n \times R_k$;
⟨6⟩ **enddo**
⟨7⟩ $iend_p = n$;
⟨8⟩ **return** ($istart_i$, $iend_i$) to distribute the data;

Fig. 5. Data distribution decision part for the load-balancer in `ABCLib_DRSSED`.

Figure 4 and Figure 5 indicate that the data is distributed according to the ratio of each execution time in PE i to the summation of all execution times by the benchmark program. We used the DAXPY program in the Basic Linear Algebra Subprograms (BLAS) for the benchmark program. The target routines for `ABCLib_DRSSED` are also linear algebra programs; hence, we believe the benchmark program can work well to estimate the behavior of the target programs.

5 Performance Evaluation

5.1 Machine Environments

We used the following three kinds of parallel computers to evaluate the FIBER optimization facilities.

- HITACHI SR8000/MPP
  - System configuration: The HITACHI SR8000/MPP nodes have 8 PEs. The theoretical maximum performance of each node is 14.4 GFLOPS. Each node has 16 GB of memory, and the inter-connection topology is a three-dimensional hypercube. Its theoretical throughput is 1.6 Gbytes/s for one-way communication, and 3.2 Gbytes/s for two-way. For the communication library, the HITACHI optimized MPI was used.
  - Compiler: The HITACHI Optimized Fortran90 V01-04 compiler with specified options, *-opt=4 -parallel=0* and *-opt=0 -parallel=0*, was used.
- Fujitsu VPP800/63
  - System configuration: This machine is a vector-parallel style of supercomputer. The Fujitsu VPP800/63 at the Academic Center for Computing and Media Studies, Kyoto University was used. The total number of nodes for the VPP800 is 63. The theoretical maximum performance of each node is 8 GFLOPS for vector processing, and 1 GFLOPS for scalar processing. Each node has 8 GB of memory, and the inter-connection topology is a crossbar. Its theoretical throughput is 3.2 Gbytes/s. For the communication library, the Fujitsu optimized MPI was used.
  - Compiler: The Fujitsu optimized UXP/V Fortran/VPP V20L20 with compiler specified options, *-O5 -X9* and *-O0 -X9*, was used.
- PC Cluster
  - System configuration: The Intel Pentium4 (2.0 GHz), as a node of a PC cluster, was used. The number of PEs for the PC cluster is 4, and each node has 1 GB (Direct RDRAM/ECC 256 MB*4) of memory. The system hardware board is the ASUSTek P4T-E+A (Socket 478). The network is the Onboard Broadcom Gigabit Ethernet*2. Linux 2.4.9-34 and MPICH 1.2.1 are used as the operating system and communication library.
  - Compiler: The PGI Fortran90 4.0-2 compiler with specified options, *-fast* and *-O0*, was used.

5.2 Hypothesis of the Cost Definition Function

The following hypothesis is made in this performance evaluation.

- Cost Definition Function: 5th-order linear polynomial function of $a_1x^5 + a_2x^4 + a_3x^3 + a_4x^2 + a_5x^1 + a_6$.

Since we found that the 5-th order polynomial function had the least errors in the PC Cluster in the experiment with the Cost Definition Functions from the 0-th to 5-th orders [11], we set this hypothesis.

5.3 Target Processes

We will evaluate auto-tuning facilities in the FIBER framework by using the following four processes for the eigensolver.

1. The Householder tridiagonalization routine in `ABCLibDRSAllEigVec`:
   $PP = \{$ `ictr`, `imv`, `iud` $\}$
2. The inverse iteration routine in `ABCLibDRSAllEigVec`:
   $PP = \{$ `kort` $\}$
3. The Householder inverse transformation routine in `ABCLibDRSAllEigVec`:
   $PP = \{$ `ichit`, `ihit` $\}$
4. The QR decomposition routine with the Gram-Schmidt method
   in `ABCLib_QRD`: $PP = \{$ `ibl`, `iop`, `isp`, `ioo`, `iso` $\}$

5.4 Sampling Point Errors

The sampling points for this evaluation are listed as follows.

- Definition area of `iud`: { 1, 2, ..., 16 }
- Sampling Points of `iud`: $\Phi_{iud} \equiv$ { 1, 2, 3, 4, 8, 16 } (Sampling Points 1)
- Sampling Points of `iud`: $\Phi_{iud} \equiv$ { 1, 2, ..., 16 } (Sampling Points 2)
- Sampling Points for Problem Size of `n`: $\Phi_n \equiv$ { 200, 400, 800, 2000, 4000, 6000, 8000 }

Table 1 shows the estimated errors with the sampling points 1.

The sampling points were well estimated in these supercomputer environments according to Table 1, since the relative errors for the execution times between the best and the estimated parameters were 7% at least.

The relative errors, however, ranged from 18% to 219% in the PC Cluster, and the fluctuation was large. This means that we need another approach in this environment. To solve this problem, we must modify the sampling points, the cost definition function, and the optimization method.

Table 2 shows the effect of the optimization in Before Execute-time. The notation "Est. Param.1" means the estimated parameter with the sampling points 1, and "Est. Param.2" means the estimated parameter with the sampling points 2. The values in "Best Param." indicate the optimized parameters in the Before Execute-time Optimization.

The results of Table 2 tell us that there is no room to improve the accuracy of the estimate in supercomputer environments. Increasing sampling points,

Table 1
Errors in the Install-time Optimization

(a) HITACHI SR8000/MPP (1 Node, 8 PEs)

Sampled Dimension	Est. Param.1 (ET [s])	Err. in Sampling Points	Best Param. (BT [s])	Rel. Err. $(ET-BT)/BT$
200	14 (5.905E-2)	8.749E-18	6 (5.867E-2)	0.64 %
400	14 (0.1665)	1.410E-16	14 (0.1665)	0 %
800	14 (0.6198)	1.167E-15	14 (0.6198)	0 %
2000	14 (5.833)	9.125E-14	16 (5.824)	0.15 %
4000	14 (41.22)	7.251E-12	15 (41.00)	0.54 %
8000	13 (314.6)	4.362E-10	15 (314.4)	0.04 %

(b) Fujitsu VPP800/63 (8 PEs)

Sampled Dimension	Est. Param.1 (ET [s])	Err. in Sampling Points	Best Param. (BT [s])	Rel. Err. $(ET-BT)/BT$
200	7 (3.073E-2)	2.283E-18	2 (3.058E-2)	0.49 %
400	7 (6.558E-2)	5.277E-17	5 (6.530E-2)	0.44 %
800	7 (0.1521)	1.456E-16	10 (0.1515)	0.40 %
2000	5 (0.6647)	2.175E-15	4 (0.6644)	0.04 %
4000	6 (3.418)	2.414E-14	2 (3.203)	6.7 %
8000	7 (23.06)	1.412E-12	4 (22.40)	2.9 %

(c) PC Cluster (4 PEs)

Sampled Dimension	Est. Param.1 (ET [s])	Err. in Sampling Points	Best Param. (BT [s])	Rel. Err. $(ET-BT)/BT$
200	7 (0.2786)	1.391E-16	13 (0.2345)	18.8 %
400	6 (2.149)	3.079E-15	4 (0.6739)	218 %
800	6 (5.603)	6.102E-14	14 (2.7176)	106 %
2000	6 (20.38)	1.533E-12	2 (15.89)	28.3 %
4000	6 (106.5)	6.107E-11	2 (88.96)	19.7 %
8000	2 (583.2)	1.901E-9	2 (583.2)	0 %

however, can be improve the accuracy of the estimate in the PC cluster. This is because the relative error in the dimension of 1234 in Table 2(c) is reduced from 28.7% to 3.1%. This reduction cannot be ignored. On the other hand, we can reduce the execution time of 1%–30% by using the library user-specified dimension information. However, the effect depends on the number of sampling

Table 2
Effect of sampling points in Before Execute-time Optimization.

(a) HITACHI SR8000/MPP (1 Node, 8 PEs)

End-User Specified Dimension	Est. Param.1 ($ET1$ [s])	Est.Param.2 ($ET2$ [s])	Best Param. (BEO Opt. , Eff. BT [s])	Rel. Err.1 $(ET1 - BT)/BT$	Rel. Err.2 $(ET2 - BT)/BT$
123	14 (0.0333)	11 (0.0341)	6 (0.0333)	0.00 %	2.4 %
1234	14 (1.668)	16 (1.662)	16 (1.662)	0.36 %	0 %
9012	16 (440.6)	12 (447.0)	16 (440.6)	0 %	1.4 %

(b) Fujitsu VPP800/63 (8 PEs)

End-User Specified Dimension	Est. Param.1 ($ET1$ [s])	Est.Param.2 ($ET2$ [s])	Best Param. (BEO Opt. , Eff. BT [s])	Rel. Err.1 $(ET1 - BT)/BT$	Rel. Err.2 $(ET2 - BT)/BT$
123	7 (0.0183)	1 (0.0182)	10 (0.0181)	1.1 %	0.5 %
1234	6 (0.2870)	6 (0.2870)	4 (0.2847)	0.8 %	0.8 %
9012	14 (34.67)	16 (34.29)	4 (32.03)	8.2 %	8.2 %

(c) PC Cluster (4 PEs)

End-User Specified Dimension	Est. Param.1 ($ET1$ [s])	Est.Param.2 ($ET2$ [s])	Best Param. (BEO Opt. , Eff. BT [s])	Rel. Err.1 $(ET1 - BT)/BT$	Rel. Err.2 $(ET2 - BT)/BT$
123	14 (0.1286)	4 (0.1285)	10 (0.1269)	1.3 %	1.2 %
1234	6 (7.838)	5 (6.2835)	10 (6.090)	28.7 %	3.1 %
9012	6 (973.6)	1 (867.0)	2 (845.6)	15.1 %	2.5 %

points.

Generally, the end-user can reduce the sampling points from 16 to 6 for the parameter *iud*, and the optimization time is decreased to a factor of $6/16 = 3/8$. However, the quality of the estimation may decrease, as in the PC cluster case in Table 2(c). This is a typical trade-off issue in auto-tuning software.

5.5 The Effect of Before Execute-time Optimization

In this evaluation, we set the following hypotheses.

- Sampling points for problem sizes of n:

· The SR8000:
Compiler Option -*opt=4*: $\Phi_n \equiv \{100, 200, ..., 1000, 2000, ..., 6000\}$
Compiler Option -*opt=0*: $\Phi_n \equiv \{100, 200, ..., 1000, 2000\}$
· The VPP800:
Compiler Option -*O5*: $\Phi_n \equiv \{100, 200, ..., 1000, 2000, ..., 6000\}$
Compiler Option -*O0*: $\Phi_n \equiv \{100, 200, ... ,900\}$
· PC Cluster:
Compiler Option -*fast*: $\Phi_n \equiv \{100, 200, ..., 1000, 2000, ..., 9000, 10000\}$
Compiler Option -*O0*: $\Phi_n \equiv \{100, 200, ..., 2000\}$
These values were determined from the limits of the computation time for each machine.

The sampling points for the PPs in this example are:

- $\Phi_{ictr} \equiv$ { `one`, `red` }: The communication method for the reduction operation in the Householder tridiagonalization routine. ϕ_{ictr_1}=`one` means using an implementation with one-to-one communication libraries (`MPI_SEND`, `MPI_RECV`). ϕ_{ictr_2}=`red` means using an implementation with an MPI released library (`MPI_ALLREDUCE`).
- $\Phi_{imv} \equiv \{ 1, 2, ..., 16 \}$: The unrolling depth for the outer loop of a matrix-vector product in the Householder tridiagonalization. The kernel is formed as a double nested loop, BLAS2.
- $\Phi_{iud} \equiv \{ 1, 2, ..., 16 \}$: Unrolling depth for the outer loop of an updating process in the Householder tridiagonalization. The kernel is formed as a double nested loop, BLAS2.
- $\Phi_{ichit} \equiv$ { `broad`, `blk`, `non-blk` }: The communication method for the gathering operation in the Householder inverse transformation routine. ϕ_{ichit_1}=`broad` means an implementation using an MPI broadcast routine (`MPI_BCAST`). ϕ_{ichit_2}=`blk` means an MPI blocking one-to-one communication routine (`MPI_SEND`, `MPI_RECV`). ϕ_{ichit_3}=`non-blk` means an MPI non-blocking one-to-one communication routine (`MPI_SEND`, `MPI_IRECV`).
- $\Phi_{ihit} \equiv \{ 1, 2, ..., 16 \}$: The unrolling depth for the outer loop of the Householder inverse transformation routine. The kernel is formed as a double nested loop, and this is classified as BLAS1.
- $\Phi_{ibl} \equiv \{ 1, 2, 3, 4, 8, 16 \}$: The blocking length for the blocked algorithm of the QR decomposition with the Gram-Schmidt method. The blocking length can also control the communication frequency and volume. The kernel is formed as a triple nested loop, and this is classified as BLAS3.
- $\Phi_{iop} \equiv \{ 1, 2, 3, 4 \}$: The unrolling depth of the outer loop for the pivot PEs in the QR decomposition routine.
- $\Phi_{isp} \equiv \{ 1, 2, 3, 4, 8, 16 \}$: The unrolling depth of the second loop for the pivot PEs in the QR decomposition routine.
- $\Phi_{ioo} \equiv \{ 1, 2, 3, 4 \}$: The unrolling depth of the outer loop for the updating process in the QR decomposition routine.
- $\Phi_{iso} \equiv \{ 1, 2, 3, 4, 8, 16 \}$: The unrolling depth of the second loop for the

updating process in the QR decomposition routine.

We only implemented the auto-tuning function for Φ_{ibl} in `ABCLib_QRD`. This is because the auto-tuning function for the nested parameters of (Φ_{iop}, Φ_{isp}) and (Φ_{ioo}, Φ_{iso}) needs a different cost definition function except for the k-th order linear polynomial functions adapted to ABCLib_DRSSED.

Figures 3 and 4 show the IO (Install-time Optimization) and BEO (Before Execute-time Optimization) effects in this experiment. The following can be seen from the results.

- The IO estimated parameters are not always optimal, but they are sufficient in many cases. This indicates that the 5th-order polynomial function is a sufficient estimation in this case.
- BEO effects are about 10%–90% speedups compared to the IO estimated parameters.
- We found, however, that in one case the IO estimations totally fail. In this case, a speedup factor of 340% for IO estimated parameters is obtained.

The above points indicate that BEO is a needed facility for the following reasons: (1) BEO can improve the speed by about 10%–90% compared to IO optimization results; (2) To avoid the failures of the IO estimation, BEO should be performed. This also indicates that BEO can assure the users about the library performance.

5.6 The Effect of the Load-Balancer

If the end-user uses a PC cluster which consists of different performance computer architectures, namely, a heterogeneous environment, the amount of distributed data should be changed to attain high performance.

The install-time when the system administrator wants to install the library, or when the library user knows the performance is stable, but there are different loads while the library runs, are evaluated in this experiment. By using Install-time Optimization, or Before Execute-time Optimization, load-balancing is established in ABCLib_DRSSED. The target of the load-balancer is the output matrix for the calculated eigenvectors and the output array to store the eigenvalues in `ABCLibDRSAllEigVec`.

We added the AMD Opteron Processor 244 (1.8 GHz) * 2PE with 2 GB of memory (DDR-SDRAM/ECC 512NV*4) as a node of the PC Cluster. The PC Cluster, hence, consists of three nodes of the Pentium4 (2 GHz), which are named Node1 through Node3, and 1 node of the AMD Opteron (1.8 GHz), which is named Node4. The master node is Node1, that is, the Pentium4.

We also added extended loads with a program to the master node to increase the heterogeneity of the PC cluster. The target routine is the Householder inverse transformation routine in `ABCLibDRSAllEigVec`.

Table 5 shows the result.

Surprisingly, speed decreases of 30% were observed in Table 5(a). One reason is the worse cache blocking for Node4, because the amount of distributed data is increased by using the load-balancing function. We need more a detailed analysis for this problem of load-balancing discovered through this experiment. The result also indicates that applying the load-balancing function will be a choice of auto-tuning.

We obtained speedup factors from 130% to 220%, except for the speedup factors in Table 5(a). The effect was increased in the case of larger dimensions and high heterogeneity (See Table 5(c).) In the case of high heterogeneity, the effect of the load-balancing function was high. Hence, in this case, the function is useful.

6 Related Work

We can classify the conventional auto-tuning software into the following three categories.

Complete Run-time Optimization Software: In this category, the software performs the parameter adjustments at run-time. For example, to tune computer system parameters such as the I/O buffer size, Active Harmony[18] and Autopilot[17] can be used.

The SANS [8] project provides a framework based on run-time optimization by a network agent. This project also includes an install-time optimization scenario [8]. However, the agent approach is basically classified as run-time tuning, because it decides the appropriate parameters at run-time.

Complete Install-time Optimization Software: In this category, the software performs the parameter adjustments at the install-time. For example, PHiPAC [3], ATLAS and the paradigm of AEOS (Automated Empirical Optimization of Software) [1,19], and FFTW [9] can automatically tune the performance parameters for the computation kernels of their routines when they are installed. Later, in the SOLAR framework [6], the implementation of hierarchy routines for the computation kernels is considered an install-time optimization.

For formalization of an auto-tuning facility in this category, Naono and Yamamoto formulated the install-time optimization in the SIMPL [16] auto-

tuning software framework, which is a paradigm for parallel numerical libraries. To reduce the search time, a theory for optimal parameters in an eigensolver was studied by Imamura and Naono [10].

Hybrid Install-time and Run-time Optimization Software: In ILIB [14,15], the facility of install-time and run-time optimizations is implemented.

The concepts of an execute-time optimization layer with the user's knowledge, in order to improve parameter accuracy and to generalize auto-tuning facilities, are not clear and rarely discussed in the conventional auto-tuning software mentioned above. We therefore believe that BEO in FIBER is a very new concept.

We have another concern for using a numerical library in a heterogeneous environment.

Numerical Dangers of Heterogeneous Environments: Numerical libraries in heterogeneous environment have another issue—that is, the reliability issue [4]. This is caused by the different computer arithmetic architectures. To avoid this issue is basically difficult. Hence, the function for heterogeneous environments should be limited. As we demonstrated in this paper, the load-balancing function should be used in PC clusters which have almost the same computer architectures but different current loads.

7 Conclusion

In this paper, we solved the issue of sampling points for auto-tuning software by supplying an end-user interface and the new optimization timing, called Before Execute-time Optimization (BEO), in the FIBER framework. We also developed a load-balancing function for data distribution to improve the performance in heterogeneous environments. Our performance evaluation indicated that speedup factors from 10% to 90% (and 340% in a failed estimation case in the Install-time Optimization) by invoking the BEO. In addition, we can improve the performance to 220% by using the load-balancer in a heterogeneous environment. Thus, the functions proposed this paper are crucial factors for improving performance.

By opening the interface of sampling points to end-users, the library can improve the stability of parameter estimation, and can also reduce the optimization time. Generally speaking, if an end-user reduces the sampling points by half to the default parameters, the optimization time will be reduced by half as well. But, the quality of the estimation may decrease. Hence, we have a trade-off between the quality and time in auto-tuning. The solution for the

problem, taking into account of the trade-off, will be important future work in auto-tuning software.

The information of ABCLib project and the source codes of ABCLib_DRSSED have been opened in the WWW page of `www.abc-lib.org`.

Acknowledgments

This study is supported by the Japan Science and Technology Agency, "Information Infrastructure and Applications," PRESTO.

References

[1] ATLAS project; http://www.netlib.org/atlas/index.html.

[2] Olivier Beaumont, Arnaud Legrand, Fabrice Rastello, and Yves Robert. Dense linear algebra kernels on heterogeneous platforms: Redistribution issues. *Parallel Computing*, 28(2):155–185, 1997.

[3] Jeff Bilmes, Krste Asanović, Chee-Whye Chin, and Jim Demmel. Optimizing matrix multiply using PHiPAC: a portable, high-performance, ANSI C coding methodology. *Proceedings of International Conference on Supercomputing 97*, pages 340–347, 1997.

[4] L.S. Blackford, A. Cleary andA. Petitet, R.C. Whaley, J. Demmel, I. Dhillon, H. Ren, K. Stanley, J. Dongarra, and S. Hammarling. Practical experience in the numerical dangers of heterogeneous computing. *ACM Transactions on Mathematical Software*, 23(2):133–147, 1997.

[5] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S.Hammarling, G.Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK Users' Guide*. SIAM, 1997.

[6] Javier Cuenca, Domingo Gimenez, and Jose Gonzalez. Architecture of an automatically tuned linear algebra library. *Parallel Computing*, 30:187–210, 2004.

[7] James W. Demmel. *Applied Numerical Linear Algebra*, pages 105–117. SIAM, 1997.

[8] Jack Dongarra and Vector Eijkhout. Self-adapting numerical software for next generation applications. *The International Journal of High Performance Computing and Applications*, 17(2):125–131, 2003.

[9] Matteo Frigo. A Fast Fourier Transform compiler. In *Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 169–180, Atlanta, Georgia, May 1999.

[10] Toshiyuki Imamura and Ken Naono. An evaluation towards an automatic tuning eigensolver with performance stability. In *Proceedings of Symposium on Advanced Computing Systems and Infrastructures (SACSIS)2003*, pages 145–152, 2003.

[11] Takahiro Katagiri, Kenji Kise, Hiroki Honda, and Toshitsugu Yuba. FIBER: A software framework to support automatically addition of generalized auto-tuning facilities. *IPSJ SIG Notes*, 2003-HPC-94:1–6, 2003.

[12] Takahiro Katagiri, Kenji Kise, Hiroki Honda, and Toshitsugu Yuba. Effect of auto-tuning with user's knowledge for numerical software. In *Proceedings of ACM Computing Frontiers 04*, pages 12–25, Island of Ischia, Italy, April 2004.

[13] Takahiro Katagiri, Kenji Kise, Hiroki Honda, and Toshitsugu Yuba. FIBER: A general framework for auto-tuning software. *Proceedings of The Fifth International Symposium on High Performance Computing*, Springer Lecture Notes in Computer Science(2858):146–159, 2003.

[14] Takahiro Katagiri, Hisayasu Kuroda, Kiyoshi Ohsawa, Makoto Kudoh, and Yasumasa Kanada. Impact of auto-tuning facilities for parallel numerical library. *IPSJ Transaction on High Performance Computing Systems*, 42(SIG 12 (HPS 4)):60–76, 2001.

[15] Hisayasu Kuroda, Takahiro Katagiri, and Yasumasa Kanada. 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*, 2281:628–639, 2002.

[16] Ken Naono and Yuusaku Yamamoto. A framework for development of the library for massively parallel processors with auto-tuning function and with the single memory interface. *IPSJ SIG Notes*, (2001-HPC-87):25–30, 2001.

[17] Randy L. Ribler, Huseyin Simitci, and Daniel A. Reed. The Autopilot performance-directed adaptive control system. *Future Generation Computer Systems, special issue (Performance Data Mining)*, 18(1):175–187, 2001.

[18] Cristian Tapus, I-Hsin Chung, and Jeffery K. Hollingsworth. Active Harmony : Towards automated performance tuning. In *Proceedings of High Performance Networking and Computing (SC2002)*, Baltimore, USA, November 2003.

[19] R.Clint Whaley, Antoone Petitet, and Jack J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27:3–35, 2001.

Table 3
IO and BEO effects for the eigensolver on the three kinds of parallel machines. One second is the unit of execution time.

(a) HITACHI SR8000/MPP
(a-1) Compiler Option (*-opt=4*)

Dim.	Def. Param. Time [s]	IO-Est. Time [s]	Param. (ictr, imv, iud) (ihit, ichit)	BEO-Opt. Time [s]	Param. (ictr, imv, iud) (ihit, ichit)	Eff.1 (Def. /BEO)	Eff.2 (IO /BEO)
512	0.607	0.588	(one, 14, 6) (3, non-blk)	0.534	(one, 14, 14) (8, blk)	1.13	1.10
5123	247	223	(one, 16, 16) (2, blk)	195	(one, 15, 15) (7, broad)	1.26	1.14
6123	485	370	(one, 16, 16) (15, broad)	331	(one, 15, 4) (16, broad)	1.46	1.11

(a-2) Compiler Option (*-opt=0*)

Dim.	Def. Param. Time [s]	IO-Est. Time [s]	Param. (ictr, imv, iud) (ihit, ichit)	BEO-Opt. Time [s]	Param. (ictr, imv, iud) (ihit, ichit)	Eff.1 (Def. /BEO)	Eff.2 (IO /BEO)
512	9.15	3.73	(one, 14, 6) (3, non-blk)	3.42	(one, 5, 5) (16, blk)	2.67	1.09
1234	102	41.8	(one, 13, 16) (5, blk)	40.8	(one, 6, 14) (5, blk)	2.50	1.02
2345	731	270	(one, 13, 16) (5, blk)	273	(one, 13, 14) (7, blk)	2.67	0.98

(b) Fujitsu VPP800/63
(b-1) Compiler Option (*-O5*)

Dim.	Def. Param. Time [s]	IO-Est. Time [s]	Param. (ictr, imv, iud) (ihit, ichit)	BEO-Opt. Time [s]	Param. (ictr, imv, iud) (ihit, ichit)	Eff.1 (Def. /BEO)	Eff.2 (IO /BEO)
512	0.815	0.771	(red, 2, 1) (16, non-blk)	0.757	(red, 10, 9) (16, blk)	1.07	1.01
5123	71.8	60.2	(red, 16, 2) (14, non-blk)	60.3	(red, 16, 2) (14, broad)	1.19	0.99
6123	110	92.01	(red, 16, 2) (14, non-blk)	91.9	(red, 16, 4) (14, non-blk)	1.19	1.00

(b-2) Compiler Option (*-O0*)

Dim.	Def. Param. Time [s]	IO-Est. Time [s]	Param. (ictr, imv, iud) (ihit, ichit)	BEO-Opt. Time [s]	Param. (ictr, imv, iud) (ihit, ichit)	Eff.1 (Def. /BEO)	Eff.2 (IO /BEO)
123	0.653	0.620	(red, 9, 8) (14, broad)	0.616	(red, 9, 9) (15, blk)	1.06	1.00
512	20.4	18.1	(one, 11, 9) (12, non-blk)	17.9	(red, 10, 9) (16, non-blk)	1.13	1.01
912	107	93.9	(red, 15, 9) (13, non-blk)	93.6	(red, 16, 9) (12, blk)	1.14	1.00

(c) PC Cluster
(c-1) Compiler Option (*-fast*)

Dim.	Def. Param. Time [s]	IO-Est. Time [s]	Param. (ictr, imv, iud) (ihit, ichit)	BEO-Opt. Time [s]	Param. (ictr, imv, iud) (ihit, ichit)	Eff.1 (Def. /BEO)	Eff.2 (IO /BEO)
512	2.88	3.32	(red, 10, 4) (1, blk)	2.68	(red, 5, 2) (1, broad)	1.07	1.00
5123	396	359	(red, 5, 2) (1, non-blk)	366	(one, 5, 1) (1, blk)	1.08	0.98
10123	2804	2497	(red, 5, 2) (1, non-blk)	2526	(red, 5, 3) (1, blk)	1.11	0.98

(c-2) Compiler Option (*-O0*)

Dim.	Def. Param. Time [s]	IO-Est. Time [s]	Param. (ictr, imv, iud) (ihit, ichit)	BEO-Opt. Time [s]	Param. (ictr, imv, iud) (ihit, ichit)	Eff.1 (Def. /BEO)	Eff.2 (IO /BEO)
512	3.55	3.45	(one, 13, 11) (7, broad)	3.31	(red, 13, 3) (3, broad)	1.07	1.04
1234	17.6	19.00	(one, 13, 8) (14, broad)	16.7	(red, 5, 8) (4, non-blk)	1.05	1.13
2345	97.4	98.6	(red, 14, 15) (4, non-blk)	84.5	(one, 6, 6) (4, non-blk)	1.15	1.16

Table 4
IO and BEO effects for the QR Decomposition Routine on the three kinds of parallel machines. One second is the unit of execution time.

(a) HITACHI SR8000/MPP
(a-1) Compiler Option (*-opt=4*)

Dim.	Def. Param. Time [s]	IO-Est. Time [s]	Param. (ibl, iop, isp, ioo, iso)	BEO-Opt. Time [s]	Param. (ibl, iop, isp, ioo, iso)	Eff.1 (Def. /BEO)	Eff.2 (IO /BEO)
512	0.217	0.290	(6, 4, 8, 4, 8)	0.171	(8, 2, 2, 4, 4)	1.26	1.69
5123	391	149	(16, 4, 8, 4, 8)	146	(8, 4, 1, 4, 4)	2.67	1.02
6123	762	270	(16, 4, 8, 4, 8)	276	(8, 2, 5, 2, 8)	2.76	0.97

(a-2) Compiler Option (*-opt=0*)

Dim.	Def. Param. Time [s]	IO-Est. Time [s]	Param. (ibl, iop, isp, ioo, iso)	BEO-Opt. Time [s]	Param. (ibl, iop, isp, ioo, iso)	Eff.1 (Def. /BEO)	Eff.2 (IO /BEO)
512	2.42	1.05	(8, 4, 8, 4, 8)	0.982	(8, 1, 3, 4, 8)	2.45	1.06
1234	33.9	15.2	(8, 4, 8, 4, 8)	16.9	(8, 2, 3, 3, 8)	2.00	0.89
2345	240	119	(16, 4, 8, 4, 8)	114	(8, 2, 5, 4, 8)	2.10	1.04

(b) Fujitsu VPP800/63
(b-1) Compiler Option (*-O5*)

Dim.	Def. Param. Time [s]	IO-Est. Time [s]	Param. (ibl, iop, isp, ioo, iso)	BEO-Opt. Time [s]	Param. (ibl, iop, isp, ioo, iso)	Eff.1 (Def. /BEO)	Eff.2 (IO /BEO)
512	0.061	0.026	(16, 4, 8, 4, 8)	0.026	(8, 1, 5, 4, 8)	2.34	1.00
5123	31.3	19.0	(8, 4, 8, 4, 8)	18.2	(16, 4, 5, 1, 16)	1.71	1.04
6123	51.4	30.6	(16, 4, 8, 4, 8)	30.2	(8, 1, 1, 3, 8)	1.70	1.01

(b-2) Compiler Option (*-O0*)

Dim.	Def. Param. Time [s]	IO-Est. Time [s]	Param. (ibl, iop, isp, ioo, iso)	BEO-Opt. Time [s]	Param. (ibl, iop, isp, ioo, iso)	Eff.1 (Def. /BEO)	Eff.2 (IO /BEO)
123	0.292	0.290	(3, 4, 8, 4, 8)	0.152	(8, 1, 3, 3, 4)	1.92	1.90
512	19.6	7.11	(8, 4, 8, 4, 8)	7.05	(8, 1, 3, 4, 8)	2.78	1.00
912	110	40.6	(8, 4, 8, 4, 8)	40.5	(8, 1, 4, 4, 8)	2.71	1.00

(c) PC Cluster
(c-1) Compiler Option (*-fast*)

Dim.	Def. Param. Time [s]	IO-Est. Time [s]	Param. (ibl, iop, isp, ioo, iso)	BEO-Opt. Time [s]	Param. (ibl, iop, isp, ioo, iso)	Eff.1 (Def. /BEO)	Eff.2 (IO /BEO)
512	0.520	0.590	(6, 4, 8, 4, 8)	0.466	(3, 1, 2, 1, 3)	1.11	1.26
5123	261	151	(8, 4, 8, 4, 8)	152	(8, 3, 4, 4, 8)	1.71	0.99
10123	2017	2079	(8, 4, 8, 4, 8)	1320	(16, 4, 16, 3, 4)	1.52	1.57

(c-2) Compiler Option (*-O0*)

Dim.	Def. Param. Time [s]	IO-Est. Time [s]	Param. (ibl, iop, isp, ioo, iso)	BEO-Opt. Time [s]	Param. (ibl, iop, isp, ioo, iso)	Eff.1 (Def. /BEO)	Eff.2 (IO /BEO)
512	1.11	0.523	(16, 4, 8, 4, 8)	0.532	(16, 2, 3, 4, 16)	2.08	0.98
1234	12.7	3.60	(8, 4, 8, 4, 8)	3.59	(8, 2, 6, 4, 8)	3.53	1.00
2345	86.0	87.3	(6, 4, 8, 4, 8)	25.4	(8, 3, 1, 4, 8)	3.38	3.43

Table 5
Effect for the Load-Balancer in a Heterogeneous PC Cluster. (4 nodes, Dimension: 8000)

(a) Case of 0 Load in the Master Node.
(Data distribution ratio: Node1: 23%; Node2: 23%; Node3: 23%; Node4: 31%;)

# of Calculated Eigenvectors	Load-Balancer OFF [s]	Load-Balancer ON [s]	Speedup
100	18.5	18.0	1.02
500	55.4	64.9	0.8
1000	94.6	123	0.7
4000	354	474	0.7
8000	690	935	0.7

(b) Case of 1 Load in the Master Node.
(Data distribution ratio: Node1: 12%; Node2: 26%; Node3: 27%; Node4: 35%;)

# of Calculated Eigenvectors	Load-Balancer OFF [s]	Load-Balancer ON [s]	Speedup
100	33.5	25.6	1.3
500	108	76.6	1.4
1000	199	129	1.5
4000	680	468	1.4
8000	1326	915	1.3

(c) Case of 2 Loads in the Master Node.
(Data distribution ratio: Node1: 9%; Node2: 27%; Node3: 27%; Node4: 37%;)

# of Calculated Eigenvectors	Load-Balancer OFF [s]	Load-Balancer ON [s]	Speedup
100	51.7	32.0	1.6
500	170	90.9	1.8
1000	279	147	1.8
4000	941	534	1.7
8000	2374	1068	2.2