FIBER: A Framework of Installation, Before Execution-invocation, and Run-time Optimization Layers for Auto-tuning Software*

Takahiro Katagiri[1 2], Kenji Kise[1 2], Hiroaki Honda[1], and Toshitsugu Yuba[1]

[1] Department of Network Science, Graduate School of Information Systems, The University of Electro-Communications
[2] PRESTO, Japan Science and Technology Corporation(JST)
1-5-1 Choufu-gaoka, Choufu-shi, Tokyo 182-8585, JAPAN
Phone: +81-424-43-5642, FAX: +81-424-43-5644
{ katagiri, kis, honda, yuba }@is.uec.ac.jp

Abstract. In this paper we propose a new paradigm, named FIBER. The FIBER software architecture framework has three kinds of parameter optimization layers—installation, before execution-invocation, and run-time, to generalize auto-tuning facilities and to obtain highly estimated parameters. FIBER framework also provides a loop unrolling function, which needs code generation and parameter registration processes, to support code development by users. The example of an eigensolver to apply the FIBER framework is shown. We also optimize the eigensolver parameters using FIBER's installation and before-execution layers in the three kinds of parallel computers, which are the HITACHI SR8000/MPP, Fujitsu VPP800/63, and Pentium4 PC cluster.

Keywords: Auto-tuning; Parameter optimization; Numerical library; Performance modeling; Eigensolver;

1 Introduction

Recently, many packages of numerical computation software with auto-tuning facility (SATF), such as PHiPAC[2], ATLAS[1, 9] and FFTW[3] have been developed. There are two reasons for this. First, the library arguments should be reduced to make the interface easier to use. A facility is needed to maintain high performance in all computer environments. Second, tuning work on complicated machine environments, such as parallel computers, is time-consuming. Automated adjustment facility for the parameters is thus needed.

Many SATF have been evaluated. The results indicate the facility of SATF is very important to maintain high performance in several kinds of computer architectures [4]. From the viewpoint of general applicability, however, the conventional facility of SATF has a limitation, i.e. the auto-tuning facility uses dedicated methods defined in each package. An example of this limitation is

* IS Technical Reports, UEC-IS-2003-3, Graduate School of Information Systems, The University of Electro-Communications (13 May, 2003)

that several general numerical libraries contain direct solvers, iterative solvers, dense solvers, and sparse solvers. There is no software framework, however, to adapt SATF to these solvers.

To solve the above problem, we propose a new and general software framework for SATF. The framework contains the following three component layers:

(1) Installation Optimization Layer (IOL) : This layer is called when software is installed.
(2) Before Execution-invocation Optimization Layer (BEOL) : This layer is called when user defined parameters are fixed (for example, problem sizes).
(3) Run-time Optimization Layer (ROL) : This layer is called at run-time to optimize the object.

We call this new software framework FIBER (Framework of Installation, Before Execution-invocation, and Run-time optimization layers).

2 Components and definitions of auto-tuning in FIBER

2.1 Components of FIBER

FIBER is a framework for software, which contains the following two kinds of functions:

– **Code development support function:** The function performs automatically code generation to grantee auto-tuning, parameterization, and its registration by specifying an instruction from users. The instruction is specified to library or sub-routine interfaces, or other parts of the program, by using a dedicated language.
– **Three kinds of parameter optimization functions:** The functions perform the optimizations of specified parameters in Parameter Tuning Layer (PTL). There are three kinds of timing for the optimizations (See Figure 2.)

Figure 1 and Figure 2 show software components, and how to optimize parameters in FIBER, respectively.

The parameters in Figure 1 are added in the library, sub-routines, or other parts of the program, described by library developers or users. One language is used to specify the parameters. The PTL optimizes the parameters to minimize a function. Please note that we can specify the parameters even in computer system libraries, such as MPI (Message Passing Interface), if the interface is open to users. PTL in FIBER, thus, can access system parameters.

There are three kinds of optimization layers to optimize the specified parameters in PTL : IOL, BEOL, and ROL. These layers can access a limited number of parameters to optimize. For instance, IOL-tuned parameters can be accessed in BEOL and ROL. BEOL-tuned parameters, however, can only be accessed by ROL. The main reason of this is to obtain highly estimated parameters in lower optimization layers.

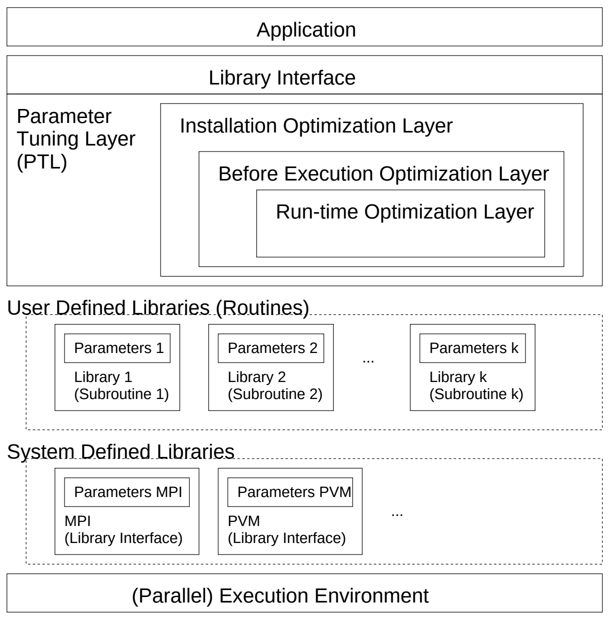

Fig. 1. Software components of FIBER.

Figure 2 is a snapshot of the optimization process using the three kinds of optimization layers in PTL.

In Figure 2, the parameters are separated into IOP (Installation Optimization Parameters) accessed in IOL, BEOP (Before Execution-invocation Optimization Parameters) accessed in BEOL, and ROP (Run-time Optimization Parameters) accessed in ROL.

The optimization procedure is performed in the following order:

(1) IOL : After software installation.
(2) BEOL : After specified the special parameters, such as problem sizes, by users.
(3) ROL : When run-time at target library, sub-routine, or other parts of the program.

The optimized parameters in the above procedure are stored in the parameter information file. Lower level optimization layers can access the parameters stored in the parameter information file to perform their optimization.

2.2 The aim of auto-tuning

In this section, we present the aim of auto-tuning facility for FIBER. To understand the auto-tuning facility of FIBER, a library interface for parallel eigenvalue computation is shown.

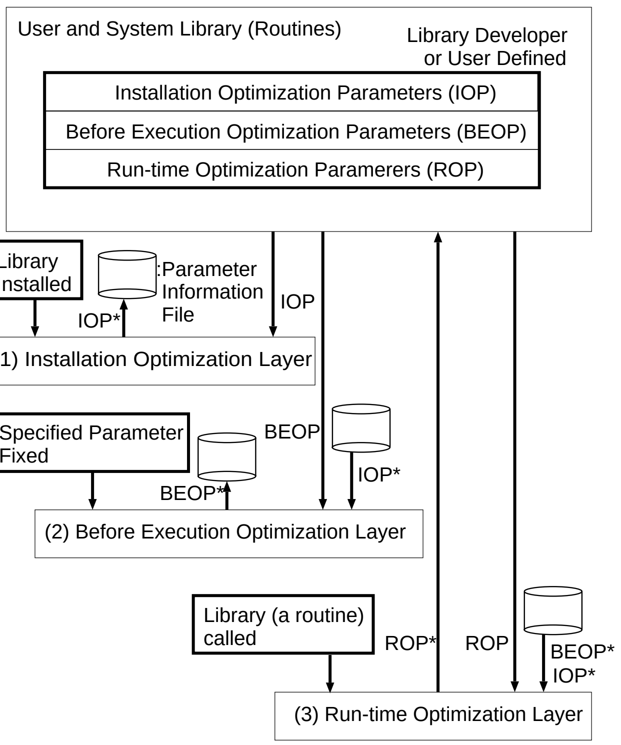
User and System Library (Routines)
Library Developer or User Defined
Installation Optimization Parameters (IOP)
Before Execution Optimization Parameters (BEOP)
Run-time Optimization Paramerers (ROP)
Library Installed
:Parameter Information File
IOP
IOP*
(1) Installation Optimization Layer
Specified Parameter Fixed
BEOP
BEOP*
IOP*
(2) Before Execution Optimization Layer
Library (a routine) called
ROP*
ROP
BEOP*
IOP*
(3) Run-time Optimization Layer

Fig. 2. Processes of the three kinds of optimization layers in FIBER.

[**Example 1**] A conventional parallel numerical library interface:

```
call PEigVecCal(
        A, x, lambda, n,                  ...(i)
        nprocs, myid, iDistInd,           ...(ii)
        imv, iud, ihit, icomm, kbi, kort, ...(iii)
        MAXITER, deps                     ...(iv)  )
```

For this paper, the arguments in Example 1 are called as (i) Basic information parameters, (ii) Parallel control parameters, (iii) Performance parameters, and (iv) Algorithm parameters. For example, the dimension sizes of matrix A are specified in the parameters of (i), data distribution information stored in the parameters of (ii), unrolling depth or block size specified in the parameters of (iii), and maximum iterative numbers defined in the parameters of (iv). Generally speaking, these arguments can be removed to design a better library interface for (ii), and to analyze numerical characteristics for (iv). The parameters of (iii), however, cannot be removed in the conventional frameworks which do not have the auto-tuning facility.

The aim of SATF is to maintain performance and removing the parameters of (iii). Using SATF, the interface can be simplified as:

```
call PEigVecCal(A, x, lambda, n)
```

2.3 The definition of auto-tuning

The auto-tuning facility is described is as follows[1] .

Let the parameter set for all library arguments be AP. The parameter set of (i) the basic information parameter defined in Example 1, and the set of the other parameters except for (i), are defined as BP and $OP \equiv AP/BP$, respectively.

Thus :

$$AP = BP \cup OP, \tag{1}$$

Where $BP \cap OP = \phi$.

The sets of AP and BP are defined as the following:

[**Definition 1**] Set AP (All Parameters)

The set of parameters for input, output, and performance is defined as AP, for all subroutine interfaces in the library, sub-routine, or other parts of the program. □

[**Definition 2**] Set BP (Basic Parameters)

The set of parameters for basic information in AP, such as matrix sizes for input or output matrices, and target machine environments information, such as the number of processors (PEs), is defined as BP. These parameters are specified by users before the target libraries, sub-routines, or other parts of the program run. □

The following assumption is made:

[**Assumption 1**] The execution time of the target library, sub-routine, or other parts of the program can be estimated by using the function F, which is derived from the set of AP. □

By Assumption 1, we obtain the execution time t for the target as

$$t = F(m), \tag{2}$$

where $m \subseteq AP$.

Let the set of PP, where $PP \subseteq OP$ be the set of (iii) performance parameters in Example 1. Then, the PP is defined as follows.

[1] Similar definition was done by Naono and Yamamoto[6]. In their definition, performance parameters are defined as CP (Critical Parameter). They defined two kinds of CP parameters, thus, UCP (Users' Critical Parameter) and ICP (Internal Critical Parameter). The UCPs are parameters specified in library arguments by users, and ICPs are parameters which do not appear library interface. The ICPs are defined in the internal library.

[**Definition 3**] Set PP (Performance Parameters)

The set of parameters in OP, which can affect the whole performance of the library, sub-routine, or other parts of the program is defined as PP, when the set of BP is fixed. Users do not need to specify the parameters in PP, but the performance of the library, sub-routine, or other parts of the program is controlled by the parameters. $\square$

Let the set of OPP be $OPP \equiv OP/PP$.

Consequently,

$$OP = PP \cup OPP, \tag{3}$$

where $PP \cap OPP = \phi$.

For the OOP, the following assumption is made.

[**Assumption 2**] The parameters in OPP do not affect the other parameters in AP, except for the parameters in BP. $\square$

Now (i) basic parameter of BP is fixed as $\bar{l} \subseteq BP$, and by using Assumption 1 and 2, the execution time t of target is estimated as:

$$t = F(\bar{l}, g), \tag{4}$$

where $g \subseteq PP$.

Thus, the auto-tuning facility is defined as the following:

[**Definition 4**] Framework of auto-tuning facility

Auto-tuning facility is defined as the optimization procedure to minimize the function of F for execution time, when the set of BP is fixed.

In other words, we can define the facility as the following optimization problem: Finding the set of $g \subseteq PP$ in the condition of $\bar{l} \subseteq BP$, such that minimizing the function of F:

$$\min_g F(\bar{l}, g) = \min_g F'(g). \qquad \square \tag{5}$$

We have defined the function of F as the execution time of the target process. In general, there are many parameter optimization problems, such as minimization of fee for computer use, and the size of memory spaces used. Taking into account these problems, we enhance the definition of F. The function of F is the target function to minimize in the optimization process, then, we call the function of F as a **cost definition function**.

2.4 The definition of auto-tuning in FIBER

In FIBER, the set of performance parameters PP is separated as the following three kinds of parameters:

$$PP = IOP \cup BEOP \cup ROP. \tag{6}$$

The parameters of IOP, $BEOP$, and ROP are defined as optimization parameters for installation, before execution-invocation, and run-time, respectively.

The definition of FIBER's auto-tuning facility is defined as follows:

[**Definition 5**] FIBER's auto-tuning facility

FIBER's auto-tuning facility is defined as an optimization procedure to estimate parameter set PP fixing a part of BP in installation, before-execution, and run-time optimization layers.

In detail, FIBER's auto-tuning facility is a procedure to minimize the cost definition function F, which is defined as users' programs and instructions in the three kinds of layers, to estimate the parameter set PP when a part of parameter set BP is fixed in each layer. □

Definition 5 indicates that FIBER's installation optimization is an estimation procedure for PP when a part of BP, which is affected by machine environments, is fixed. FIBER's before-execution optimization can be explained as an estimation procedure for PP when a part of BP, which is determined by users' knowledge for the target process, is fixed by using optimized parameters in FIBER's installation optimization layer. For this reason, the parameters estimated by FIBER's before-execution optimization have higher accuracy than pentameters estimated by FIBER's installation optimization. For the parameters of BP, which can not be fixed in the two optimization layers, FIBER's run-time optimization layers determines their values to estimate PP at run-time.

3 Example of auto-tuning for FIBER

3.1 Specification of performance parameters by users

In FIBER framework, users can instruct detail instructions to specify the performance parameters of PP and target areas of auto-tuning in their programs. Hereafter, we call the target area as *tuning region*. The follows are examples.

Specification format In source program, the line of `!ABCLib$` is regarded as FIBER's instruction. The overall notation is figured in Figure 3.

```
!ABCLib$ ⟨Auto-tuning Type⟩ ⟨Function Name⟩ [ (Target Variables) ] region start
 [ !ABCLib$ ⟨Detail of Function⟩ [ sub region start ] ]

                              Tuning Region

 [ !ABCLib$ ⟨Detail of Function⟩ [ sub region end ] ]
!ABCLib$ ⟨Auto-tuning Type⟩ ⟨Function Name⟩ [ (Target Variables) ] region end
```

Fig. 3. Instruction format of auto-tuning in FIBER.

The ⟨Auto-tuning Type⟩ and ⟨Function Name⟩ in Figure 3 are called as instruction operators. The instruction operator of ⟨Auto-tung Type⟩ can specify the three kinds of timing—installation optimization (`install`), before execution-invocation optimization (`static`), and run-time optimization (`dynamic`). How to process the target code and details for auto-tuning method can be specified by the instruction operator of ⟨Function Name⟩.

The typical instruction operators for function name, named as *unrolling instruction operator* (`unroll`) and *selection instruction operator* (`select`), are shown in the following sections.

Example of unrolling instruction operator The following code shows an example of unrolling instruction operator.

```
!ABCLib$ install unroll (j) region start
!ABCLib$ varied (j) from 1 to 16
!ABCLib$ fitting polynomial 5 sampled (1-4,8,16)
 do j=0, local_length_y-1
    tmpu1 = u_x(j)
    tmpr1 = mu * tmpu1 - y_k(j)
    do i=0, local_length_x-1
       A(i_x+i, i_y+j) = A(i_x+i, i_y+j)
          + u_y(i)*tmpr1 - x_k(i)*tmpu1
    enddo
 enddo
!ABCLib$ install unroll (j) region end
```

The above code shows that the loop unrolled codes for j-loop, which adapts unrolling to the tuning region of `region start` – `region end`, are automatically generated. The depth of the loop unrolling is also automatically parameterized as PP. The instruction shows that installation optimization is performed for this tuning region.

The instruction operator, which can specify detailed function for the target instruction operator, is called as sub-instruction operator. The sub-instruction operator of `varied` defines the defined area of target variables. In this example, the defined area is `{1,..,16}`. For cost definition function, the types of them can be specified by the sub-instruction operator of `fitting`. In this example, a 5-th order linear polynomial function is specified. The sub-instruction operator of `sampled` is for definition of sampling points to estimate the cost definition function. This example of sampling is `{1-4,8,16}`.

Example of selection instruction operator The following code shows an example of selection instruction operator.

```
!ABCLib$ static select region start
!ABCLib$ parameter (in CacheS, in NB, in NPrc)
!ABCLib$   select sub region start
!ABCLib$   according estimated
!ABCLib$         (2.0d0*CacheS*NB) / (3.0d0*NPrc)
```

```
              Target Process 1
!ABCLib$   select sub region end
!ABCLib$   select sub region start
!ABCLib$   according estimated
!ABCLib$         (4.0d0*CacheS*dlog(NB))/(2.0d0*NPrc)
              Target Process 2
!ABCLib$   select sub region end
!ABCLib$ static select region end
```

The above code shows that the selection procedure from several tuning regions, which are specified by `sub region start` – `sub region end`, is performed based on the values of formulas, which are specified by the sub-instruction operator of `according estimated`.

The variables referring in the formulas are defined as the sub-instruction operator of `parameter`. The sub-instruction operator of `in` shows that the target variables are input variables. The values of the variables should be stored in an installation optimization layer by using the sub-instruction operator of `out` to parameter information file, since this example is defined as before execution-invocation optimization.

Please note that the selection of Target Process 1 or Target Process 2 is parameterized as PP in this example.

3.2 Example of Installation Optimization Layer (IOL)

In this section, we adapt the auto-tuning facility of FIBER to an eigensolver. This is implemented using the Householder-bisection-inverse iteration method for computing all eigenvalues and eigenvectors in dense real symmetric matrices.

We define the cost definition function as the execution time for the solver. For target parallel computers, the HITACHI SR8000/MPP at the Information Technology Center, The University of Tokyo is used[2] .

Let the interface of the target library in the Householder-bisection-inverse iteration method be the same interface as `PEigVecCal`, as shown in Example 1. The main arguments of this library are shown as follows:

– $PP \equiv$ { `imv`, `iud`, `icomm`, `ihit`, `kbi`, `kort` }
– $BP \equiv$ { `n`, `nprocs` }

The library interface of `PEigVecCal` consists of the following four kinds of performance parameters for PP in this library.

[2] The nodes of the HITACHI SR8000/MPP have 8 PEs. The theoretical maximum performance of each node is 14.4 GFLOPS. Each node has 16 GB memory, and interconnection topology is three dimensional hyper-cube. Its theoretical throughput is 1.6 Gbytes/s for one-way, and 3.2 Gbytes/s for both-way. In this example, the HITACHI Optimized Fortran90 V01-04 compiler specified option of *-opt=4 -parallel=0* was used. For the communication library, the HITACHI optimized MPI (Message Passing Interface) was used.

1. Hoseholder tridiagonalization routine:
   $PP = \{$ `imv`, `iud`, `icomm` $\}$
2. Bisection routine : $PP = \{$ `kbi` $\}$
3. Inverse iteration routine: $PP = \{$ `kort` $\}$
4. Householder inverse transformation routine:
   $PP = \{$ `ihit` $\}$

The definition area and process in each parameter are define as:

- `imv` $\equiv$ { 1,2,...,16 } : Unrolling depth for the outer loop of a matrix-vector product in the Householder tridiagonalization (a double nested loop, BLAS2).
- `iud` $\equiv$ { 1,2,...,16 } : Unrolling depth for the outer loop of an updating process in the Householder tridiagonalization (a double nested loop, BLAS2).
- `kbi` $\equiv$ { vec, non-vec } : Types of implementation in a bisection routine. `vec` means a vectorized implementation, and `non-vec` means a non-vectorized implementation.
- `kort` $\equiv$ { MG-S, CG-S, IRCG-S, NoOrt } : Types of algorithms in inverse iteration routine for re-orthogonalization algorithms to calculate eigenvectors corresponding to clustered eigenvalues.
- `ihit` $\equiv$ { 1,2,...,16 } : Unrolling depth for the outer loop of the Householder inverse transformation routine (a double nested loop, the kernel is classified as BLAS1.)

A set of the parameters which can optimize in IOL is

- $IOP \equiv \{$ `imv`, `iud`, `kbi`, `ihit` $\}$.

How to estimate parameters in IOL We can determine the parameters when the target computer systems, such as computer hardware architecture or compilers, are fixed, since these parameters can be affected by factors of computer hardware information, such as the number of registers, the size of caches, and the feature of vector processing.

The parameters are determined in the following way. First of all, the parameters in BP are fixed, and several points for execution time at the target process are sampled (refined to as sampled data). The cost definition function is then determined by using the sampled data.

[**Example 2**] The installation optimization of the parameter of `iud` in Example 1.

Let the execution time be approximated by a linear pronominal formula. The sampling points to estimate the best parameter for `iud` are { 1,2,3,4,8,16 }. For the parameters of BP, the number of PE is fixed as 8, and the parameters of `n`, which are the problem sizes sampled as { 200, 400, 800, 2000, 4000, 8000 }. The code of tuning region, cost definition function, and sampling points are same as the example of unrolling instruction operator in Section 3.1.

Table 1 shows the execution time at the HITACHI SR8000/MPP with the above sampled points.

Table 1. The execution time of the HITACHI SR8000/MPP 8PE. [sec.]

`n\iud`	1	2	3	4	8	16
200	.0628	.0628	.0629	.0623	.0621	.0625
400	.1817	.1784	.1763	.1745	.1723	.1719
800	.7379	.6896	.6638	.6550	.6369	.6309
2000	7.535	6.741	6.333	6.240	6.013	5.846
4000	54.06	48.05	44.85	44.36	42.89	41.19
8000	413.2	366.5	349.2	344.1	327.6	315.5

In this experiment, the basic parameter of `n` is fixed to estimate the function $f_n(iud)$ for `iud`, where $f_n(iud)$ is a k-th order polynomial function of $f_n(iud) = a_1 \cdot iud^k + a_2 \cdot iud^{k-1} + a_3 \cdot iud^{k-2} + \cdots + a_k \cdot iud + a_{k+1}$. By using an appropriate optimization method, we can determine the coefficients of $a_1, .., a_{k+1}$.

Table 2 shows the average of relative errors between the execution time with estimated parameters with k-th order cost definition functions [3] and the execution time with the best parameters measuring all area of definition, in each sampling points of the problem size `n`.

Table 2. Average values of relative errors between execution time of estimated parameters with linear polynomial functions and execution time of the best parameters. (HITACHI SR8000/MPP 8PE)

0-th	1-st	2-nd	3-rd	4-th	5-th
19.2	0.51	1.73	0.87	0.82	0.23

Table 2 indicated that the 5-th order polynomial function is the best in the viewpoint of averaged relative error. Thus, we use the 5-th order polynomial function as the cost deification function.

We chose the least square method to estimate the coefficients of the 5-th oder polynomial function. Table 3 shows the determined coefficients by using sampled data from Table 1. The least square method with the Householder QR decomposition is used to obtain the coefficients of Table 3.

We can determine the best parameters of `iud` in the all definition area of { 1,2,...,16 } by using the coefficients of Table 3 in each problem size.

Figure 4 shows the overall of estimated function $f_n(iud)$ by using the coefficients of Table 3. For this optimization, the sampled problem sizes of `n` are not always specified by the users at run-time. The estimated parameters in Figure 4, hence, cannot be used in any of the cases. If users specify a different problem size

[3] It is from 0-th to 5-th. The 0-th order shows the function returns the value of the smallest parameters. In this case, the measured execution time of `iud`=1 returns for all definition area.

Table 3. The coeffcients of the estimated cost definition function. (Fixed the problem size)

f_n	a_1	a_2	a_3	a_4	a_5	a_6
f_{200}	-2.2E-6	6.7E-5	-6.5E-4	2.4E-3	-3.8E-3	6.4E-2
f_{400}	-1.3E-6	4.2E-5	-4.6E-4	2.3E-3	-7.8E-3	1.8E-1
f_{800}	9.6E-6	-2.3E-4	7.9E-4	1.1E-2	-8.4E-2	8.1E-1
f_{2000}	2.4E-4	-6.3E-3	3.5E-2	1.1E-1	-1.3E-0	8.6E-0
f_{4000}	3.0E-3	-8.3E-2	6.1E-1	-4.9E-1	-7.7eE0	6.1E+1
f_{8000}	-1.3E-2	5.0E-1	-7.0E+0	4.5E+1	-1.4E+2	5.1E+2

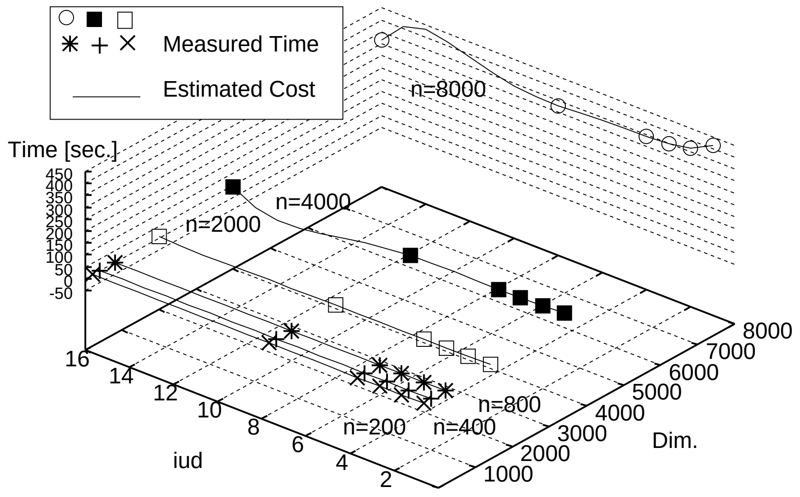

Fig. 4. The estimation of optimized parameter of **iud**. (Fixed the problem sizes)

against the sampling dimensions at run-time, FIBER estimates the appropriate values by using the coefficients of Table 3 to calculate the estimation costs for the defined area of `iud`. The method of this estimation is explained as follows.

First, the costs for fixing the parameter `iud` can be calculated by varying the problem sizes of `n` as { 200, 400, 800, 2000, 4000, 8000 }. Hence, we can obtain the all estimation costs in the `iud` defined area of { 1,2,...,16 } for the problem sizes `n` of { 200, 400, 800, 2000, 4000,8000 }.

Second, these estimation costs can be regarded as new sampling points. By using the new sampling points, we can estimate the function $f_{iud}(n)$ for the number of problems. We also chose the least square method to estimate the function $f_{iud}(n)$. Let the $f_{iud}(n)$ also be approximated by a 5-order polynomially function of $f_{iud}(n) = \bar{a_1} \cdot n^5 + \bar{a_2} \cdot n^4 + \bar{a_3} \cdot n^3 + \bar{a_4} \cdot n^2 + \bar{a_5} \cdot n + \bar{a_6}$.

Figure 5 shows the coefficients varied from the sampled number of `n` as { 200, 400, 800, 2000, 4000, 8000}.

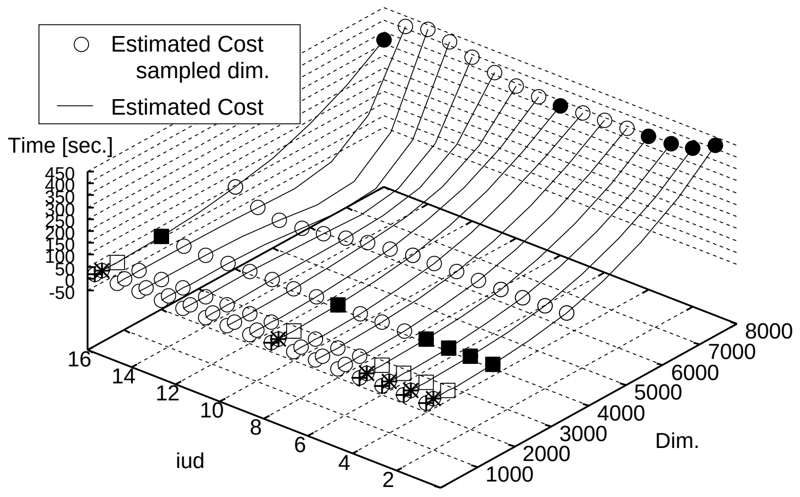

Fig. 5. The estimation of the optimized parameter of **iud**. (A case that the problem sizes are not fixed until run-time.)

Figure 5 indicates that the function $f_{iud}(n)$ for `n` is determined in the entire definition area { 1,2,...,16 } of `iud`, thus, the parameters of `iud` to minimize the cost function can be determined by substituting the `n` at run-time.

This is the parameter optimization procedure of FIBER's IOL.

Experiments of the effect of IOL We evaluate an effect of IOL for the parameters of `iud` by using several kinds of machine environments. The details of machine environments are summarized as the follows.

- HITACHI SR8000/MPP
  - System configuration and compiler: Explained in Section 3.2.
- Fujitsu VPP800/63
  - System configuration: This machine is a vector-parallel style super-computer. The Fujitsu VPP800/63 at the Academic Center for Computing and Media Studies, Kyoto University is used. The total number of nodes for the VPP800 is 63 nodes. The theoretical maximum performance of each node is 8 GFLOPS for vector processing, and 1 GOPS for scalar processing. Each node has 8 GB memory, and inter-connection topology is a cross bar. 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 compiler specified option of *-O5 -X9* was used.
- PC Cluster
  - System configuration: As a node of PC cluster, the Intel Pentium4 2.0GHz is used. The number of PEs for the PC cluster is 4, and each node has

1GB (Direct RDRAM/ECC 256MB*4) memory. The system hardware board is the ASUSTek P4T-E+A (Socket478). The network card is the Intel EtherExpressPro100+. The 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 specified option of *-fast* was used.

[**Experiment 1**] We evaluate the estimated errors in IOL. Figure 4 shows the estimated errors and relative errors in the sampled points of `iud`.

Table 4. Estimated and relative errors for the sampled points in IOL with 5-th order polynomial cost definition function.

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

Sampled Dim.	Estimated Param.1 (Exe.T. ET[Sec])	Estimated Error	Best Param. (Exe.T. BT[Sec])	Rel. Err $(ET - BT)/BT * 100$
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 (8PEs)

Sampled Dim.	Estimated Param.1 (Exe.T. ET[Sec])	Estimated Error	Best Param. (Exe.T. BT[Sec])	Rel. Err $(ET - BT)/BT * 100$
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 (4PEs)

Sampled Dim.	Estimated Param.1 (Exe.T. ET[Sec])	Estimated Error	Best Param. (Exe.T. BT[Sec])	Rel. Err $(ET - BT)/BT * 100$
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 %

The estimated errors in Figure 4 were calculated by the sum of power of subtraction for the measured time and the estimated time for the sampled points. The relative errors in Figure 4 were calculated by using the execution time for estimated parameters and the execution time for best parameters.

Figure 4 indicated that the relative errors in PC cluster are huge compared to the other errors in super computers. The main reason is that the fluctuation for the execution time was observed in PC cluster. The estimation of parameter, hence, is sensitive compared to the case of super computers.

3.3 Example of Before Execution-invocation Optimization Layer (BEOL)

In FIBER, the optimization of BEOL is performed when the parameters in BP are specified before executing the target process. This section explains several adoptions for this layer.

[**Situation 1**] In Example 1, the users know the number of processors (=8 PEs), and matrix sizes (=8192) for the eigenvalue computation. ($\texttt{n} \equiv 8192$, $\texttt{nprocs} \equiv 8$)

In Situation 1, the parameters to optimize in BEOL are

– $BEOP \equiv \{ \texttt{imv}, \texttt{iud}, \texttt{ihit}, \texttt{kbi} \}$.

Please note that the IOL uses the totally estimated parameters or estimated parameters using the user-specified sampled data. In BEOP, however, users know the number of problems n, and specify the value to inform the FIBER optimization system. Hence, the accuracy of parameter estimated in BEOP is better than that of IOP.

BEOP can be used in processes which need highly estimated parameters.

Experiments of the effect of BEOL We evaluate FIBER's BEOL by using the three kinds of parallel computers. Firstly, we will evaluate the following Experiment 2.

[**Experiment 2**] Evaluate the case that users can know the problem sizes to execute the library. The sizes are 123, 1234, and 9012.

Figure 5 shows that the execution time of FIBER's IOL-estimated parameters with the sampling points of `iud` (Est.Param.1), the execution time of FIBER's IOL-estimated parameters with all definition area points (Est.Param.2), and the execution time of FIBER's BEOL-estimated parameters (Best Param.).

The results of Figure 5 indicated that (1) modifying sampling points makes better estimated accuracy; (2) FIBER's BEOL has 0.5% – 28.7% effectiveness compared to FIBER's IOL;

Let's consider the following Situation 2.

[**Situation 2**] Users know the matrix coefficients are not changed in the invocations.

In Situation 2, the parameters for BEOP can be defined as

Table 5. Effect of optimization in BEOL and sampling points.

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

Specified Dim.	Est.Param.1 (Exe.T.$ET1$[Sec])	Est.Param.2 (Exe.T.$ET2$[Sec])	Best Param. (BEOL Opt.Res., Exe.T.BT[Sec])	Rel.Err1 $(ET1 - BT)$ $/BT * 100$	Rel.Err2 $(ET2 - BT)$ $/BT * 100$
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 (8PEs)

Specified Dim.	Est.Param.1 (Exe.T.$ET1$[Sec])	Est.Param.2 (Exe.T.$ET2$[Sec])	Best Param. (BEOL Opt.Res., Exe.T.BT[Sec])	Rel.Err1 $(ET1 - BT)$ $/BT * 100$	Rel.Err2 $(ET2 - BT)$ $/BT * 100$
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 (4PEs)

Specified Dim.	Est.Param.1 (Exe.T.$ET1$[Sec])	Est.Param.2 (Exe.T.$ET2$[Sec])	Best Param. (BEOL Opt.Res., Exe.T.BT[Sec])	Rel.Err1 $(ET1 - BT)$ $/BT * 100$	Rel.Err2 $(ET2 - BT)$ $/BT * 100$
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 %

– $BEOP \equiv$ { `imv`, `iud`, `ihit`, `kbi`, `kort` }.

In Situation 2, the process of re-orthogonalization in the inverse iteration method can be optimized.

[**Experiment 3**] Let the users know the information that target matrix coefficients are not changed same as Situation 2. In this situation, evaluate the effect of FIBER's BEOL for the parameter of `kort`.

The following is the example of BEOL optimization, where a user wants to solve the eigenvalues and eigenvectors of a Frank matrix order 10,000 by using the HITACHI SR8000/MPP. Table 6 and Table 7 show the results for the execution time and accuracy of the eigenvector with the different parameters of `kort`.

Table 6 shows that the execution time was different according to re-orthogonalization methods in the Frank matrix. With taking account of numerical stability, the MG-S method is selected as a *de facto* parameter in several libraries.

If users can specify the accuracy of eigenvectors in this situation, i.e. less $1.5E-12$, the system can determine the suitable re-orthogonalization method. In this case, CG-S was the best parameter. Consequently, the BEOL can determine the parameter of `kort` as CG-S in this situation using the value of accuracy from

Table 6. Execution time of each re-orthogonalization method in inverse iteration method. (HITACHI SR8000/MPP, $n = 10,000$) The unit is second. The notation of $>$ means the iteration was not converged in a limitation for execution in the super-computer environments.

#PEs (nprocs)	CG-S	MG-S	IRCG-S	NoOrt
8	6,604	38,854	12,883	23
16	3,646	24,398	6,987	12
32	2,061	28,050	3,906	7
64	1,633	27,960	3,059	3
128	2,091	$>$	3,978	1

Table 7. The accuracy of eigenvectors calculated with each re-orthogonalization method in inverse iteration method. (HITACHI SR8000/MPP, $n = 10,000$) The unit is the norm of Frobenius.

#PEs (nprocs)	CG-S	MG-S	IRCG-S	NoOrt
8	6.4E-13	6.6E-13	6.4E-13	1.4
16	6.6E-13	6.6E-13	6.6E-13	1.4
32	6.8E-13	6.6E-13	6.8E-13	1.4
64	9.4E-13	6.6E-13	9.4E-13	1.4
128	1.5E-12	–	1.5E-12	1.4

users. The accuracy of eigenvectors and the code of algorithm selection in the re-orthogonalization methods can be implemented by using the selection instruction operation in Section 3.1.

Figure 8 shows the speedup ratio compared to the case of normal default parameter (the MG-S method) in Situation 2. Figure 8 indicated that we can

Table 8. Speedup ratio to specified the normal default parameter of MG-S method, when users specify the accuracy of eigenvectors as less $1.5E - 12$ in Situation 2. (HITACHI SR8000/MPP)

#PEs (nprocs)	8	16	32	64	128
Speedup	5.8	6.6	13.6	17.1	–

obtain 5.8%–17.1% speedups in this case. This is a typical case to apply the BEOL optimization in FIBER.

This speedup is crucial, hence, we can conclude that FIBER's BEOL is an important function to optimize algorithms according to user's knowledge.

3.4 Example of Run-time Optimization Layer (ROL)

For Example 1, the ROP (Run-time Optimization Parameters) in ROL is

- $IOP \equiv \{$ `kort` $\}$.

In this situation, the size of the matrix and the characteristics of the matrix inputed are not fixed. The suitable re-orthogonalization method depends on user specified conditions and the characteristics of the matrix at run-time.

One approach for the optimization of ROL is the following. In BEOL, the history of suitable parameters for `kort` is stored. Next at run-time, the history is used to determine the parameters of `kort`. For example, the most often selected parameter is chosen in ROL. If the parameter does not satisfy the user specified accuracy, the most stable method, such as MG-S, is selected. This implementation, of course, has an overhead to re-try the computation. But the specification of the parameter `kort` is true, and the execution time can be shortened.

Almost all sparse solvers can apply the ROL optimization. This is because, the best algorithm depends on the location of a non-zero value of the inputed matrix. The location is defined at run-time. For example, the block sizes in a block algorithm for sparse solvers are determined by the location of non-zero elements for the target matrix.

4 Related work

There are two kinds of paradigms for auto-tuning software.

The first paradigm is a computer system software. For tuning computer system parameters, such as IO buffer size, Active Harmony[8] and Autopilot[7] are known.

The second paradigm is a numerical library. PHiPAC[2], ATLAS and the paradigm of AEOS (Automated Empirical Optimization of Software)[1, 9], and FFTW[3] can automatically tune the performance parameters of their routines when they are installed. In ILIB[4, 5], the facility of installation and run-time optimization is implemented. However, the concepts of (1) execution-invocation optimization layer to estimate PP fixing parameters of BP from usrs' knowledge, and (2) the FIBER's parameter reference procedure, to improve parameter accuracy and to generalize auto-tuning facilities, are not clear and rarely discussed in these libraries. We therefore believe that these two characteristics for FIBER are quite new concepts.

For formalization of an auto-tuning facility, Naono and Yamamoto formulated the installation optimization in the SIMPL[6] auto-tuning software framework, which is a paradigm for parallel numerical libraries.

5 Conclusion

In this paper, the authors propose a new framework for auto-tuning software, named FIBER. FIBER has three kinds of parameter optimization layers — installation, execution-invocation, and run-time optimization, to generalize auto-tuning facilities and to improve parameter accuracy. The key point of the FIBER framework is how to determine the cost definition function of F according to the characteristics of libraries, sub-routines, or other parts of the program.

The authors evaluate the cost definition function of F with a linear polynomial function. Arbitrary processes, however, cannot estimate their costs by using the function. To extend the adaption of auto-tuning and to obtain high quality estimated parameters, more sophisticated methods to estimate parameters are needed. An important task for us in the future is to evaluate these methods.

The authors have developed a parallel eigensolver, named `ABCLibDRSSED`, which contains a part of FIBER's functions. The source code and manual for the version alpha of `ABCLibDRSSED` are open to public through the following WWW page: `http://www.abc-lib.org/`.

The authors also have a plan to develop a language, named `ABCLibScript`. It supports code generation, parameterization, and its registration for auto-tuning facilities based on the FIBER's concept, which are shown in Section 3.1.

Acknowledgments

The authors would like to thank to Associate Professor Reiji Suda at the University of Tokyo, for giving us useful comments and discussions for FIBER. This study was partially supported by PRESTO, Japan Science and Technology cooperation (JST).

References

1. Atlas project; http://www.netlib.org/atlas/index.html.
2. J. Bilmes, K. Asanović, C.-W. Chin, and J. 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.
3. M. 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.
4. T. Katagiri, H. Kuroda, K. Ohsawa, M. Kudoh, and Y. 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.
5. H. Kuroda, T. Katagiri, and Y. 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.
6. K. Naono and Y. 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.
7. R. L. Ribler, H. Simitci, and D. A. Reed. The autopilot performance-directed adaptive control system. *Future Generation Computer Systems, special issue (Performance Data Mining)*, 18(1):175–187, 2001.
8. C. Tapus, I.-H. Chung, and J. K. Hollingsworth. Active harmony : Towards automated performance tuning. In *Proceedings of High Performance Networking and Computing (SC2002)*, Baltimore, USA, November 2003.
9. R. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimizations of software and the atlas project. *Parallel Computing*, 27:3–35, 2001.