Scientific Programming 22 (2014) 299–307
DOI 10.3233/SPR-140395
IOS Press

Implementation of d-Spline-based incremental performance parameter estimation method with ppOpen-AT

Teruo Tanaka [a,*], Ryo Otsuka [a], Akihiro Fujii [a], Takahiro Katagiri [b] and Toshiyuki Imamura [c]
[a] *Faculty of Information, Kogakuin University, Tokyo, Japan*
E-mails: {teru, fujii}@cc.kogakuin.ac.jp
[b] *Information Technology Center, The University of Tokyo, Tokyo, Japan*
E-mail: katagiri@kata-lab.itc.u-tokyo.ac.jp
[c] *RIKEN Advanced Institute for Computational Science, Kobe, Japan*
E-mail: imamura.toshiyuki@riken.jp

Abstract. In automatic performance tuning (AT), a primary aim is to optimize performance parameters that are suitable for certain computational environments in ordinary mathematical libraries. For AT, an important issue is to reduce the estimation time required for optimizing performance parameters. To reduce the estimation time, we previously proposed the Incremental Performance Parameter Estimation method (*IPPE method*). This method estimates optimal performance parameters by inserting suitable sampling points that are based on computational results for a fitting function. As the fitting function, we introduced *d-Spline*, which is highly adaptable and requires little estimation time. In this paper, we report the implementation of the IPPE method with ppOpen-AT, which is a scripting language (set of directives) with features that reduce the workload of the developers of mathematical libraries that have AT features. To confirm the effectiveness of the IPPE method for the runtime phase AT, we applied the method to sparse matrix–vector multiplication (SpMV), in which the block size of the sparse matrix structure blocked compressed row storage (BCRS) was used for the performance parameter. The results from the experiment show that the cost was negligibly small for AT using the IPPE method in the runtime phase. Moreover, using the obtained optimal value, the execution time for the mathematical library SpMV was reduced by 44% on comparing the compressed row storage and BCRS (block size 8).

Keywords: Automatic performance tuning, fitting function, SpMV, performance parameter estimation, mathematical library

1. Introduction

One of the main aims of automatic performance tuning (AT) is to optimize performance parameters that are suitable for certain computational environments in ordinary mathematical libraries [2,3,6–8,11,12]. In this study, we discuss the optimization of a single performance parameter. In a conventional estimation of performance parameters, the following procedure is applied to obtain the optimal value in a mathematical library:

Step 1. Choose static sampling points from the values of the performance parameter.
Step 2. Run the target mathematical library to obtain the execution time (executed value) at each sampling point.
Step 3. Define a fitting function and fit it to the executed values.
Step 4. Search for a minimum value of the fitting function that corresponds to the optimal value of the performance parameter.

In general, this approach to conventional estimation of performance parameters has been used in AT.

The accuracy of the estimated value of the cost function depends on the number of sampling points, although fixed-point sampling is generally used in AT. The target mathematical library can also be executed at

*Corresponding author. E-mail: teru@cc.kogakuin.ac.jp.

any sampling point. Moreover, sampling points can be incremented dynamically. Because the sampling cost is proportional to the number of sampled points, we should consider alternative approaches, such as a dynamic incremental method, in which adequate points are inserted successively.

For performance parameter estimation, we have proposed the Incremental Performance Parameter Estimation method (*IPPE method*). In this method, optimal performance parameters are estimated by inserting suitable sampling points that refer to the computational results for a fitting function, *d-Spline*, which is highly adaptable and requires little estimation time [9,10].

When designing the IPPE method, we considered two major issues. The first was the selection of a fitting function under the following conditions: (1) It should be calculated using a small number of sampling points because the estimation begins from the least number of sampling points. (2) The calculation cost should be small because it is necessary to calculate a fitting function for every sampling point, which is incremented dynamically. The second issue was the criteria for incrementing sampling points; i.e., (1) terminating and (2) selecting sampling points.

We have now applied the IPPE method to ppOpen-AT, a scripting language for AT [4,5]. In particular, ppOpen-AT is a scripting language (set of directives) with the purpose of automatically generating the code required for AT by placing annotations in the source program to reduce the workload of library developers. We applied the IPPE method to ppOpen-AT for both the install-time and runtime phases. For the runtime phase, the IPPE method was implemented in a mathematical library, such as BLAS, which is within the loop structure of the user program. The IPPE method is executed to improve performance every time the mathematical library in the loop structure is executed. To confirm the effectiveness of AT for the runtime phase using the IPPE method, we have applied it to sparse matrix–vector multiplication (SpMV).

This paper is organized as follows. Section 2 of this paper introduces the d-Spline-based IPPE method. Section 3 describes the procedure used in the IPPE method. Section 4 provides an outline of ppOpen-AT. Section 5 describes the implementation of the IPPE method with ppOpen-AT. Section 6 presents an example of the application of the IPPE method to ppOpen-AT for SpMV. Finally, Section 7 provides a summary, conclusions and planned future work.

2. Incremental performance parameter estimation based on d-Spline

2.1. Fitting function d-Spline

Figure 1 illustrates the fitting function d-Spline with sampling points and executed values. Regarding the first issue of the IPPE method, we define a fitting function d-Spline (discrete spline function), which has the high flexibility required to adapt to data sets and can be easily computed. The function d-Spline is represented by the values of the discrete points $f_j = f(x_j)$, $1 \leqslant j \leqslant n$, i.e., $\mathbf{f} = (f_1, f_2, f_3, \ldots, f_j, \ldots, f_n)^t$, where t means transposition (Fig. 1). Each x_j has the same interval. Parameter values are a portion of $\mathbf{x}$. For smoothness, the number of $x_j(n)$ must be sufficiently greater than the number of parameter values (N). The executed value y_i ($1 \leqslant i \leqslant k, k \leqslant N$) for k sampling points out of N is the execution time of the target mathematical program. The executed values are for sampling points $\mathbf{y} = (y_1, y_2, y_3, \ldots, y_i, \ldots, y_n)^t$. The smoothness of $\mathbf{f}$ is achieved by $|f_{j-1} - 2f_j + f_{j+1}|$, $2 \leqslant j \leqslant n-1$, and $\mathbf{f}$ is selected to minimize the following expression (1):

$$\min_f \left(\|\mathbf{y} - E\mathbf{f}\|^2 + \alpha^2 \|D\mathbf{f}\|^2 \right), \tag{1}$$

where α is sufficiently small to adapt well to the executed values $\mathbf{y}$. Figure 2 illustrates matrices E and D of sizes $k \times n$ and $(n-2) \times n$, respectively. The function d-Spline is not a spline function because the continuity of the derivatives is not guaranteed; thus, it is named as d-Spline for discrete spline.

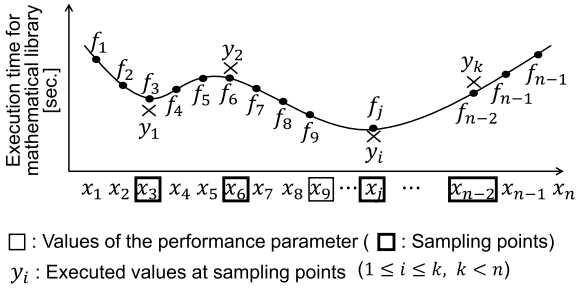

Fig. 1. Example of the fitting function d-Spline.

$$\boldsymbol{E} = \underbrace{\begin{bmatrix} 1 & & & 0 \\ & 1 & 0 & \\ & 0 & 1 & \\ 0 & & & 1 \end{bmatrix}}_{n} \Big\} k, \quad \boldsymbol{D} = \underbrace{\begin{bmatrix} 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & 1 & -2 & 1 \\ & & & \cdots & \\ & & & 1 & -2 & 1 \end{bmatrix}}_{n} \Big\} n-2$$

Fig. 2. Structure of matrices E and D in (1).

2.2. Analysis of d-Spline using the Givens method

To solve (1), we first solve the following least-squares problem:

$$\min_{f}\left(\|\mathbf{b} - Z\mathbf{f}\|^2\right). \tag{2}$$

Figure 3 shows the structures of the matrix Z and vector $\mathbf{b}$. To suppress fill-ins, which change zero elements to nonzero elements, $\|\mathbf{y} - E\mathbf{f}\|$ is transformed to $\|E^t\mathbf{y} - E^tE\mathbf{f}\|$. The matrix E^tE has nonzero values on some diagonal elements, while all values on off-diagonal elements are zero. To solve the least-squares problem, the Householder method is generally used; however, for a sparse matrix, the Givens method is better. Moreover, in our parameter estimation method, the Givens method is better for suppressing fill-ins because of the special structure of the matrix Z.

When we apply the Givens method, Z is transformed to R, which is a tridiagonal matrix (half bandwidth is 3) with a computational complexity of $O(n)$, where n is the number of parameter values. In comparison, the computational complexity of the mathematical program is $O(L^2)$ or $O(L^3)$, where L is the matrix size. Therefore, the computational complexity for solving a d-Spline can be ignored. Figure 4 illustrates the addition of a new row at the incrementing sampling point. When a new sampling point x_s is incremented, a Givens transformation is performed to remake R (obtained from existing points; see Fig. 4) by adding a new row. However, a Givens transformation need not be applied for Z. Therefore, the same $O(n)$ requires less computation (approximately $1/5$) in the increment phase. This shows that the method of adding sampling points together with the fitting function d-Spline makes an efficient combination.

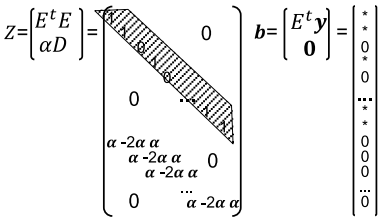

Fig. 3. Structure of the matrix Z and vector $\mathbf{b}$ in (2).

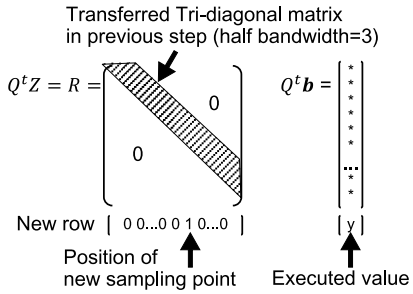

Fig. 4. Addition of a new row at the incrementing sampling point.

3. Procedure for incremental performance parameter estimation method

This section describes the procedure for estimating the optimal parameter value by applying d-Spline. The second issue in using IPPE method is the criteria for incrementing sampling points, i.e., (1) terminating and (2) selecting sampling points. This issue is considered here. Figure 5 shows the procedure for the IPPE method:

Step 1. Select four initial sampling points from $\mathbf{x}$, including both end points.
Step 2. Execute the target mathematical library for each point x_j to obtain y.
Step 3. Compute d-Spline $\mathbf{f}$, which is adaptable to executed values $\mathbf{y}$, and estimate x_s, which corresponds to f_s, the smallest of $\mathbf{f}$.
Step 4. If x_s is the same for successive t times ($t =$ 2, 3, 4, 5), exit (criterion 1); otherwise, go to Step 5.
Step 5. If the estimated x_s is not among the sampling points, add x_s as a new sampling point; otherwise, select another x_s to make f_s such that $\max_j |f_{j-1} - 2f_j - f_{j+1}|$, $1 \leqslant j \leqslant n$ (criterion 2), and return to Step 2.

In Step 4, the number of successive t values is selected by balancing the accuracy required for estimation against computational efficiency.

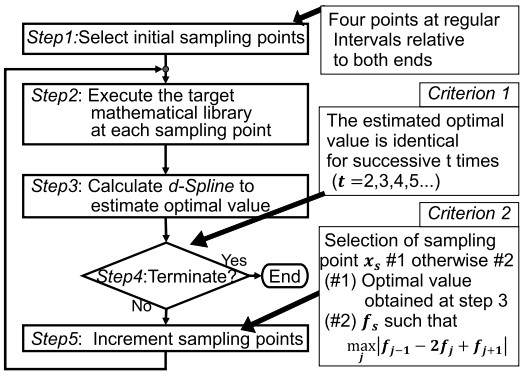

Fig. 5. Procedure for the IPPE method.

4. Outline of ppOpen-AT

The software ppOpen-AT is a scripting language (set of directives) designed to efficiently develop mathematical libraries for parallel computations. In particular, ppOpen-AT automatically generates the code required for AT by placing annotations in the source program, thereby reducing the workload of library developers. Moreover, ppOpen-AT provides a code generator, in the manner of a preprocessor, which is based on its defined directives.

4.1. Timing of automatic performance tuning (AT)

There are two optimization phases in AT: the install-time and runtime phases (see Fig. 6). AT for the install-time phase occurs when mathematical libraries are installed on the computer.

The runtime phase occurs when the mathematical library is run via the user program. Note that AT must be executed in the loop structure of the user program. Because AT information is updated in the loop structure, it is called dynamic AT. AT must be executed quickly so it does not affect the user program. In the runtime phase, information is obtained about characteristics of the user program, such as the size or structure of matrices or both.

4.2. Major functions of ppOpen-AT

ppOpen-AT provides the following three types of AT functions:

Unroll: loop unrolling depth adjustment to loop unrolled code.
Variable: blocking of length adjustment to blocked algorithms.
Select: algorithm selection based on user's knowledge.

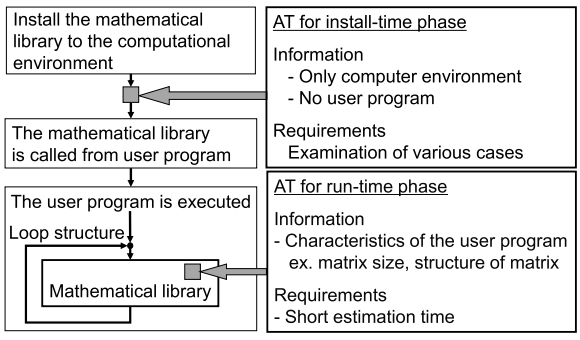

Fig. 6. Install-time and runtime optimization phases for AT.

4.3. Example of ppOpen-AT

An example of the input and output of ppOpen-AT is shown in Fig. 7. The input is a user program. The annotations of ppOpen-AT (#pragma OAT) are placed in the user program. The output is the user program with AT features, and ppOpen-AT provides a code generator in the manner of a preprocessor.

The user program, as shown in Fig. 7, performs matrix–matrix multiplication, which is applied in the adjustment function for the unrolling depth in the install-time phase optimization. In this program, the variable n specifies the matrix dimension.

Because this example uses the pragma operator *unroll* and specifies the loop variable i, the loop unrolling depth of the outer loop i is selected as the performance parameter. Because the pragma cooperator *varied* is used, the range for the depth is selected from 1 to 16. The fitting function for the execution time of the target AT region for optimization is a fifth-order linear polynomial, which, in this example, is selected by the pragma co-operator *fitting*.

In the AT system, the best values for the performance parameter are estimated by fixing n. For the performance parameter in this example, sampling points, which are equal to the depth of unrolling to fix n, are selected by the pragma cooperator *sampled*. In this ex-

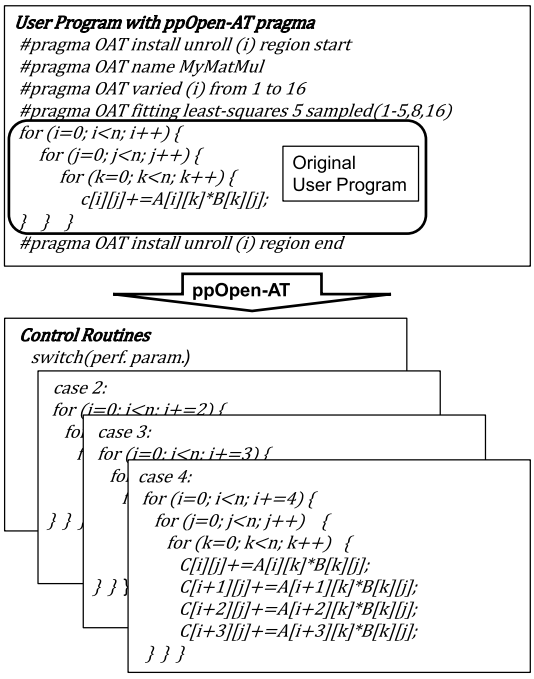

Fig. 7. Input and output codes of ppOpen-AT.

ample, the execution time from the first to the fifth, eighth, and sixteenth unrolling depths were measured. Subsequently, based on the measured data, the coefficients of the fifth-order linear polynomial were determined to estimate the values for the performance parameter. The least-squares method was used in this estimation.

5. Implementation of IPPE method with ppOpen-AT

We implemented the IPPE method with ppOpen-AT for the install-time and runtime phases. In ppOpen-AT, sampling points are selected by the pragma cooperator **sampled**. For each sampling point, the execution time for the mathematical library is measured. In contrast, in the IPPE method, sampling points are automatically selected.

To implement the IPPE method with ppOpen-AT, we entered two new pragmas:

OAT fitting dspline and

OAT fitting dynamicdspline.

The first pragma is used for the install-time phase AT, while the second is used for the runtime phase AT. As illustrated in Fig. 8, we replaced the pragma

OAT fitting least-squares 5 sampled (1–5, 8, 16)

with the pragma *OAT fitting dspline*.

For AT in the install-time phase, we used the procedure for the IPPE method described in Section 3. The AT for the runtime phase is described as follows. The left side of Fig. 9 shows the loop structure of the user program. Several pragmas are placed in the user's target mathematical library for AT. To add the functionality of the IPPE method to ppOpen-AT, the pragma that was added is as follows: *OAT fitting dynamicdspline*. By executing ppOpen-AT as a preprocessor, a user program with AT features is generated. The right side of Fig. 9 shows the pseudo-code of the generated user program.

In the sentence *do mathematical library*(P), P refers to the value of the performance parameter. To calculate the execution time ($t3$), the function *wall clock*() is executed at the beginning and at the end of the mathematical library. The function *dynamicdspline*() is executed in two phases: P is selected in the first phase and then a new P, which is used in the next iteration, is selected based on the result for $t3$ in the second phase.

User Program with original ppOpen-AT pragma

```
#pragma OAT install unroll (i) region start
#pragma OAT name MyMatMul
#pragma OAT varied (i) from 1 to 16
#pragma OAT fitting least-squares
                        5 sampled(1-5,8,16)
for (i=0; i<n; i++) {
   for (j=0; j<n; j++) {
      for (k=0; k<n; k++) {
         c[i][j]+=A[i][k]*B[k][j];
}  }  }
#pragma OAT install unroll (i) region end
```

User Program with ppOpen-AT pragma using *IPPE method*

```
#pragma OAT install unroll (i) region start
#pragma OAT name MyMatMul
#pragma OAT varied (i) from 1 to 16
#pragma OAT fitting dspline

for (i=0; i<n; i++) {
   for (j=0; j<n; j++) {
      for (k=0; k<n; k++) {
         c[i][j]+=A[i][k]*B[k][j];
}  }  }
#pragma OAT install unroll (i) region end
```

Fig. 8. New pragma *OAT fitting dspline* for d-Spline in AT.

```
for (iter=0; iter<itermax; iter++) {
  ...
  #pragma OAT dynamic unroll (i) region start
  #pragma OAT name MyMatMul
  #pragma OAT varied (i) from 1 to 16
  #pragma OAT fitting dynamicdspline

  do mathematical library(P)

  #pragma OAT dynamic unroll (i) region end

  ...
}
```

ppOpen-AT

```
for (iter=0; iter<itermax; iter++) {
   ...

   do dynamicdspline(get P)

   t1 = wall clock()
   do mathematical library(P)
   t2 = wall clock()
   t3 = t2-t1

   do dynamicdspline(use t3)
   ...
}
```

Fig. 9. Pseudo-code generated by ppOpenAT for the runtime phase.

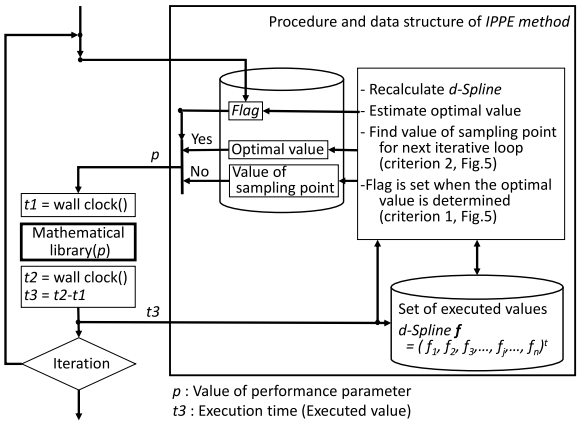

Fig. 10. Implementation of IPPE method for the runtime phase.

Figure 10 illustrates the procedure of the IPPE method for the runtime phase AT. In the first phase, *flag* is checked, which indicates whether the optimal value has been determined. If the optimal value has been determined, the optimal value is set to P; otherwise, it is set to the value of the sampling point. In the second phase, d-Spline is recalculated by adding the execution time $t3$ to select a next sampling point. According to the IPPE procedure described in Section 3, if the optimal value is determined, *flag* is set: otherwise, the value of the sampling point is selected and used in the mathematical library in the next iteration.

The first and second phases are called in each iteration. Consequently, the following data should be registered:

- d-Spline $\mathbf{f} = (f_1, f_2, f_3, \ldots, f_j, \ldots, f_n)^t$,
- *flag:* determination of optimal value,
- value of the sampling point for the next iteration,
- estimated optimal value,
- t times: number of successions for the identical optimal value,
- list of sampling points not yet measured.

6. Application to sparse matrix–vector multiplication

To confirm the effectiveness of the IPPE method for the runtime phase AT, we applied the method to sparse matrix–vector multiplication (SpMV). For a sparse matrix structure, compressed row storage (CRS) is usually used; however, in some cases, depending on the structure of the sparse matrix, blocked compressed row storage (BCRS) is used rather than CRS to obtain better performance. Figure 11 shows the structures

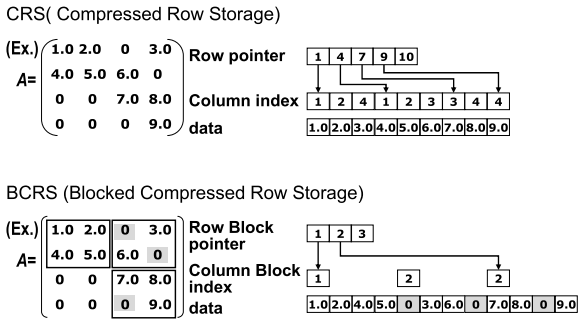

Fig. 11. Structures of CRS and BCRS for compressing sparse matrices.

of CRS and BCRS. CRS keeps only nonzero data elements; in contrast, BCRS comprises square blocks, where each block is a dense matrix, which may have some zeroes.

For our experiment, the range of the performance parameter was in block sizes from 1 to 14. For CRS, a block size of 1 was used, and for BCRS, block sizes from 2 to 14 were used. We used the sparse matrix *raefsky3* obtained from the University of Florida Sparse Matrix Collection [1].

Figure 12(a)–(d) show the changes in the shape of d-Spline. The x-axis is the value of the performance parameter (block size), and the y-axis is the execution time for the target mathematical library. The dotted lines represent execution times for all values of the performance parameter. The dotted lines show various tops and bottoms, where data fitting was not easy.

Figure 12(a) shows the shape of d-Spline at the fourth iteration of the user program. To obtain these initial four sampling points, the loop structure was iterated four times. In each iteration, the mathematical library was executed. The initial four sampling points were at regular intervals, including both ends. The numbers in the squares on d-Spline indicate the number of iteration times. The minimum point of d-Spline was found to be 5, which was already included among the sampling points. The next value selected for the sampling point was 6, where the difference equation of second order has the largest value (Section 3, criterion 2). The selected sampling point 6 was used for the calculation of the mathematical library in the next iteration.

Figure 12(b) shows the shape of d-Spline at the fifth iteration of the user program. The d-Spline was reshaped to add the execution time for sampling point 6. The minimum sampling point of the reshaped d-Spline was found to be 6, which was already among the sampling points. Following the same procedure described

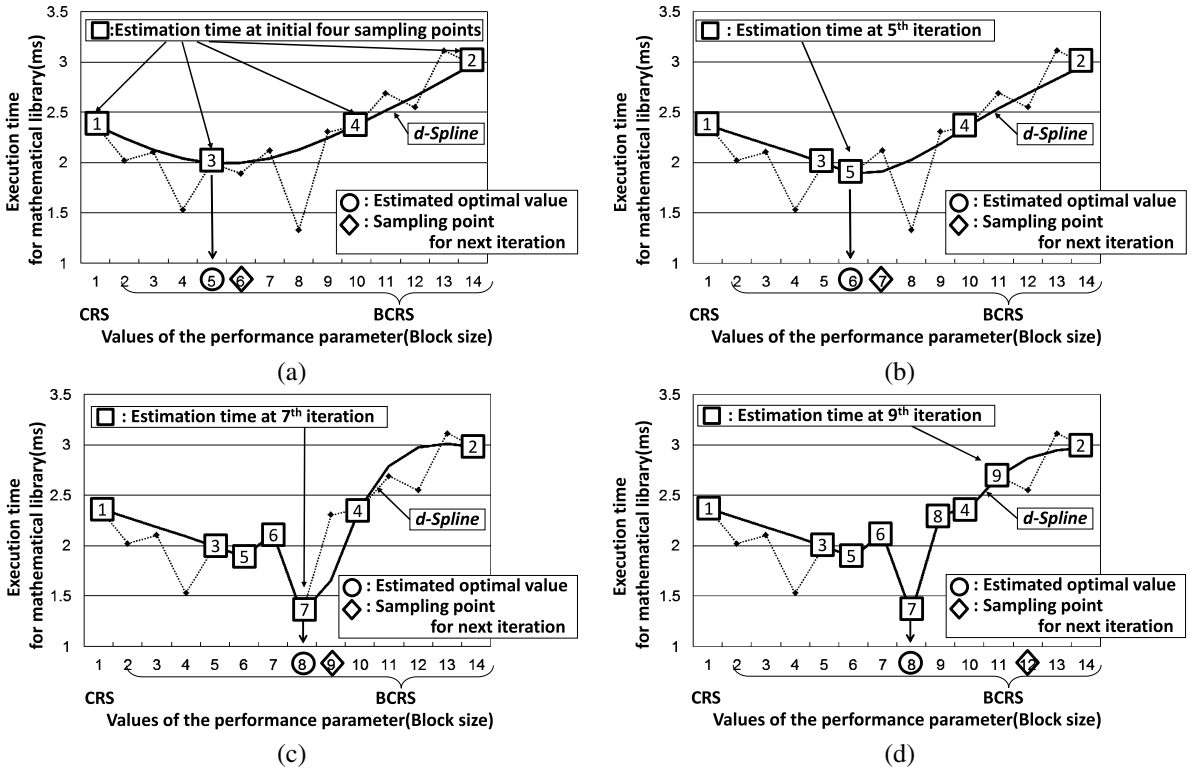

Fig. 12. Changes in the shape of d-Spline as optimization develops.

for Fig. 12(a), the next value selected for the sampling point was 7, where the difference equation of second order has the largest value (Section 3, criterion 2).

Figure 12(c) shows the shape of d-Spline at the seventh iteration of the user program, after the addition of execution times for sampling points 7 and 8. The minimum sampling point of the reshaped d-Spline was found to be 8. According to the same procedure described above, the next sampling point that was found by d-Spline was 9.

Figure 12(d) shows the shape of d-Spline at the ninth iteration of the user program. The minimum point of d-Spline was found to be 8; the value of sampling point 8 has now been selected for three successive times, which is considered to be a sufficient number of successions. Therefore, 8 was determined to be the optimal value, and *flag*, as shown in Fig. 10, was set.

Figure 12(a)–(d) illustrate the non-smooth behavior of the performance, which has been reported previously [7,8]. These figures also show that d-Spline defined by the expression (1) in Section 2, adapts well to the execution time (executed values).

Figure 13 shows the execution time for the mathematical library for each iteration. A mathematical library that uses different values of sampling points is calculated once in each iteration until the optimal value is determined. The broken line in the figure indicates the execution time for the mathematical library, including the estimation time using the IPPE method. This shows that the cost of AT for the runtime phase using the IPPE method was negligibly small. Moreover, the execution time for the mathematical library was reduced by 44% on comparing CRS and BCRS (block size 8) using the obtained optimal value. After the tenth iteration of the user program, the mathematical library will use 8 as the determined optimal value.

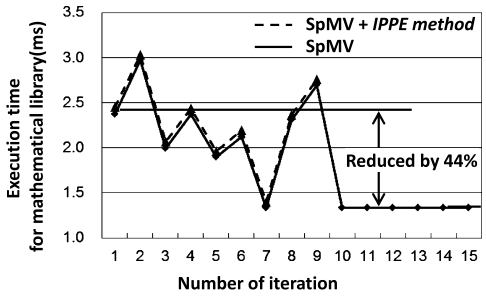

Fig. 13. Execution time for each iteration.

7. Related work

Reference [11] introduces Active Harmony, which is incorporated into system software and optimizes the runtime phase. Enhancement of Active Harmony has been discussed in [2,3] for obtaining two-dimensional performance parameters using the simplex method. Active Harmony generates a tuned code and executes just-in-time compilation multiple times during the runtime phase.

In contrast, AT with IPPE method is applied at the user level. The code to be tune is already complied, which is executed only for searching performance parameter. The function d-Spline is efficient because its overhead for runtime performance tuning is negligible (computational complexity of d-Spline is $O(n)$), while Active Harmony requires generating tuned code and executes just-in-time compilation at the system level with considerable overhead.

Reference [8] discusses multidimensional performance parameters. Comparisons have been reported among various search techniques, such as simplex, genetic algorithms, simulated annealing, and particle swarm optimization. These comparisons were applied only to install-time AT, and the target for AT was only matrix multiplication. These search techniques are totally different in viewpoint from the mathematics used in the IPPE method.

The target for AT in references [2,3] is kernels from explicit solvers, and in reference [8] the target is only matrix multiplication, while in this paper, we treat SpMV.

In the above references, two-dimensional and multiple dimensional performance parameters are examined. One of our future goals is to enhance d-Spline to handle multidimensional tuning parameter space. The smoothness of d-Spline in one-dimensional parameter space is represented by $|f_{i-1} - 2f_{i,j} + f_{i+1}|$. In two-dimensional parameter space, the smoothness of d-Spline can be represented by $|f_{i-1,j}+f_{i,j-1}-4f_{i,j}+f_{i+1,j}+f_{i,j+1}|$. The structure of the half bandwidth of the matrix Z in Fig. 4 can be reshaped from 3 to $2n-1$. Therefore, the computational complexity of d-Spline will increase to $O(n^3)$, which will reduce the efficiency of d-Spline. We are now considering some new approaches to reduce the computational complexity.

8. Conclusions

In this study, we proposed the IPPE method for estimating performance parameters. The IPPE method estimates optimal performance parameters by automatically inserting suitable sampling points that refer to computational results for a fitting function d-Spline.

The IPPE method based on d-Spline improved the two major problems in parameter search: (1) non-smooth behavior of the performance with respect to the tuning parameters and (2) a lack of efficient runtime of performance tuning.

We have integrated the methodology in the ppOpen-AT framework. Using the IPPE method with ppOpen-AT is effective for both the install-time and runtime phases because sampling points are selected automatically, i.e., the user does not have to specify the sampling points. Especially for the runtime phase AT, the cost in terms of execution time is low because the IPPE method does not have to perform additional executions of the mathematical library to measure the execution time for AT. The mathematical library is executed using different performance parameter values that are automatically selected in each iteration until the optimal value is determined.

Figure 12(a)–(d) show that d-Spline adapts well to the non-smooth behavior of the value of the performance parameter. The results from our experiment in Fig. 13 show that the cost of AT for the runtime phase using the IPPE method was negligibly small. Moreover, the execution time for the mathematical library SpMV was reduced by 44% on comparing CRS and BCRS (block size 8) using the obtained optimal value.

In future, we plan to enhance the IPPE method and d-Spline for simultaneous evaluation of multiple performance parameters. In the long run we aim to apply this methodology in a larger scope, such as more kinds of performance parameters, more matrices/data sets and parallelization.

Acknowledgement

This work was partially supported by JSPS KAKENHI, Grant-in-Aid for Scientific Research (B), "Exascale Adaptation to Sparse Iterative Library with Runtime Auto-tuning Facility", Grant Number 24300004.

References

[1] T. Davis, UF sparse matrix collection, www.cisu.ufi.e.,du/research/sparse/matrices.

[2] J.K. Hollingsworth, M. Hall, J. Chame, C. Chen and A. Tiwari, A scalable autotuning framework for compiler optimization, in: *IPDPS 2009*, May 2009, Rome.

[3] J.K. Hollingsworth and A. Tiwari, Online adaptive code generation and tuning, in: *IPDPS*, May 2011.

[4] T. Karagiri, S. Itoh and S. Ohshima, Adaptation of ppOpen-AT to numerical kernels on explicit method, in: *SIAM Annual Meeting (SIAM AN12)*, 2012.

[5] T. Katagiri, S. Ito and S. Ohshima, Early experiences for adaptation of ppOpen-AT to numerical kernels on explicit method, in: *Special Session: Auto-Tuning for Multicore and GPU (ATMG), in conjunction with the IEEE 7th International Symposium on Embedded Multicore/Manycore System-on-Chip (MCSoC-13)*, National Institute of Informatics, Tokyo, Japan, September, 2013, Proceedings of MCSoC2013, pp. 153–158, DOI: 10.1109/MCSoC.2013.15 2013.

[6] T. Katagiri, K. Kise, H. Honda and T. Yuba, ABCLibScript: A directive to support specification of an auto-tuning facility for numerical software, *Parallel Computing* **32**(1) (2006), 92–112.

[7] T. Katagiri, K. Kise, H. Honda and T. Yuba, ABCLib_DRSSED: A parallel eigensolver with an auto-tuning facility, *Parallel Computing* **32**(3) (2006), 231–250.

[8] K. Seymour, H. You and J.J. Dongarra, A comparison of search heuristics for empirical code optimization, in: *The 3rd International Workshop on Automatic Performance Tuning*, Tsukuba, Japan, October 1st, 2008.

[9] T. Tanaka, T. Katagiri and T. Yuba, *d-Spline* based incremental parameter estimation in automatic performance tuning, in: *Proceedings of the 8th International Conference on Applied Parallel Computing: State of the Art in Scientific Computing*, LNCS, Vol. 4699, Springer, 2007, pp. 986–995.

[10] T. Tanaka, R. Otsuka, A. Fujii and T. Katagiri, An incremental parameter estimation method applying *d-Spline* for software automatic tuning, in: *The 8th East Asia Section of SIAM Conference (EASIAM2012)*.

[11] C. Tapus, I.-H. Chung and J.K. Hollingsworth, Active harmony: Towards automated performance tuning, in: *Supercomputing 02, Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, 2002, pp. 1–11.

[12] R.C. Whaley, A. Petitet and J.J. Dongarra, Automated empirical optimizations of software and the ATLAS project, *Parallel Computing* **27** (2001), 3–35.