EigenKernel

A middleware for parallel generalized eigenvalue solvers to attain high scalability and usability

**Kazuyuki Tanaka[1] · Hiroto Imachi[1,2] · Tomoya Fukumoto[1] ·
Akiyoshi Kuwata[1] · Yuki Harada[1] · Takeshi Fukaya[3] · Yusaku Yamamoto[4] ·
Takeo Hoshi[1]**

Abstract

An open-source middleware named EigenKernel was developed for use with parallel generalized eigenvalue solvers or large-scale electronic state calculation to attain high scalability and usability. The middleware enables the users to choose the optimal solver, among the three parallel eigenvalue libraries of ScaLAPACK, ELPA, EigenExa and hybrid solvers constructed from them, according to the problem specification and the target architecture. The benchmark was carried out on the Oakforest-PACS supercomputer and reveals that ELPA, EigenExa and their hybrid solvers show better performance, when compared with pure ScaLAPACK solvers. The benchmark on the K computer is also used for discussion. In addition, a preliminary research for the performance prediction was investigated, so as to predict the elapsed time T as the function of the number of used nodes P ($T = T(P)$). The prediction is based on Bayesian inference in the Markov Chain Monte Carlo (MCMC) method and the test calculation indicates that the method is applicable not only to performance interpolation but also to extrapolation. Such a middleware is of crucial importance for application-algorithm-architecture co-design among the current, next-generation (exascale), and future-generation (post-Moore era) supercomputers.

Extended author information available on the last page of the article

1 Introduction

Efficient computation with the current and upcoming (both exascale and post-Moore era) supercomputers can be realized by application-algorithm-architecture co-design [1–5], in which various numerical algorithms should be prepared and the optimal one should be chosen according to the target application, architecture and problem. For example, an algorithm designed to minimize the floating-point operation count can be the fastest for some combination of application and architecture, while another algorithm designed to minimize communications (e.g. the number of communications or the amount of data moved) can be the fastest in another situation.

The present paper proposes a middleware approach, so as to choose the optimal set of numerical routines for the target application and architecture. The approach is shown schematically in Fig. 1 and the crucial concept is called 'hybrid solver'. In general, a numerical problem solver in simulations, is complicated and consists of sequential stages, as Stages I, II, III, . . . in Fig. 1. Here the routines of P, Q, R are considered for Stage I and those of S, T, U are for Stage II. The routines in a stage are equivalent in their input and output quantities but use different algorithms. The routines are assumed to be included in ScaLAPACK and other parallel libraries. Consequently, they show different performance characteristics and the optimal routine depends not only on the applications denoted as A, B, C but also on the architectures denoted as X, Y, Z. Our middleware assists the user to choose the optimal routine among
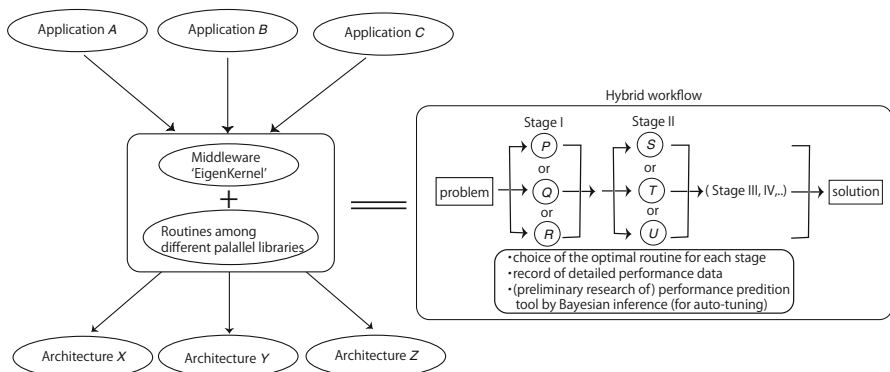
Fig. 1 Schematic figure of the present middleware that realizes a hybrid workflow. The applications, such as electronic state calculation codes, are denoted as A, B, C and the architectures (supercomputers) are denoted as X, Y, Z. A hybrid workflow for a numerical problem, such as generalized eigenvalue problem, consists of Stages I, II, III, . . . and one can choose the optimal one for each stage among the routines P, Q, R, S, T, U in different numerical libraries

different libraries for each stage and such a workflow is called 'hybrid workflow'. The present approach for hybrid workflow is realized by the following functions; First, it provides a unified interface to the solver routines. In general, different solvers have different user interface, such as the matrix distribution scheme, so the user is often required to rewrite the application program to switch from one solver to another. Our middleware absorbs this difference and frees the user from this troublesome task. Second, it outputs detailed performance data such as the elapsed time of each routine composing the solver. Such data will be useful for detecting the performance bottleneck and finding causes of it, as will be illustrated in this paper. In addition, we also focus on a performance prediction function, which predicts the elapsed time of the solver routines from existing benchmark data prior to actual computations. As a preliminary research, such a prediction method is constructed with Bayesian inference in this paper. Performance prediction will be valuable for choosing an appropriate job class in the case of batch execution, or choosing an optimal number of computational nodes that can be used efficiently without performance saturation. Moreover, performance prediction will form the basis of an auto-tuning function planned for the future version, which obviates the need to care about the job class and detailed calculation conditions. In this way, our middleware is expected to enhance the usability of existing solver routines and allows the users to concentrate on computational science itself.

Here we focus on a middleware for the generalized eigenvalue problem (GEP) with real-symmetric coefficient matrices, since GEP forms the numerical foundation of electronic state calculations. Some of the authors developed a prototype of such middleware on the K computer in 2015–2016 [6,7]. After that, the code appeared at GITHUB as EigenKernel ver. 2017 [8] under the MIT license. It was confirmed that EigenKernel ver. 2017 works well also on Oakleaf-FX10 and Xeon-based supercomputers [6]. In 2018, a new version of EigenKernel was developed and appeared on the developer branch at GITHUB. This version can run on the Oakforest-PACS supercomputer, a new supercomputer equipping Intel Xeon Phi many-core processors. A performance prediction tool using Python is also being developed to be built into a future version of EigenKernel. A related project is ELSI (ELectronic Structure Infrastructure) that provides interfaces to various numerical libraries to solve or circumvent GEP in electronic structure calculations [9,10]. However, ELSI allows the user to choose a library only *as a whole*. In contrast, the present project enables the user to construct a hybrid workflow of GEP solver, which combines routines from different libraries, as shown in Fig. 1. This will add more flexibility and increase the chance to get higher performance.

In this paper, we take up this version and discuss two topics. First, we show the performance data of various GEP solvers on Oakforest-PACS obtained using EigenKernel. Such data will be of interest on its own since Oakforest-PACS is a new machine and few performance results of dense matrix solvers on it have been reported; stencil-based application [11] and communication-avoiding iterative solver for a sparse linear system [12] were evaluated on Oakforest-PACS, but their characteristics are totally different from those of dense matrix solvers such as GEP solvers. Furthermore, we point out that one of the solvers has a severe scalability problem and investigate the cause of it with the help of the detailed performance data output by EigenKernel. This illustrates how EigenKernel can be used effectively for performance analysis. Second,

we describe the new performance prediction function implemented as a Python program. It uses Bayesian inference and predicts the execution time of a specified GEP solver as a function of the number of computational nodes. We present the details of the mathematical performance models used in it and give several examples of performance prediction results. It is to be noted that our performance prediction function can be used not only for interpolation but also for extrapolation, that is, for predicting the execution time at a larger number of nodes from the results at a smaller number of nodes. There is a strong need for such prediction among application users.

This paper is organized as follows. The algorithm for the GEP, the GEP solvers adopted in EigenKernel, and the features of EigenKernel are described in Sect. 2. Sect. 3 is devoted to the scalability analysis of various GEP solvers on Oakforest-PACS, which was made possible with the use of EigenKernel. Section 4 explains our new performance prediction function, focusing on the performance models used in it and the performance prediction results in the case of extrapolation. Section 5 discusses the advantages and limitations of our performance prediction method, comparing it with some existing studies. Finally Sect. 6 provides summary of this study and some future outlook.

2 EigenKernel

EigenKernel is a middleware for GEP that enables the user to use optimal solver routines according to the problem specification (matrix size, etc.) and the target architecture. In this section, we first review the algorithm for solving GEP and describe the solver routines adopted by EigenKernel. Features of EigenKernel are also discussed.

2.1 Generalized eigenvalue problem and its solution

We consider the generalized eigenvalue problem

$$A\mathbf{y}_k = \lambda_k B\mathbf{y}_k, \tag{1}$$

where the matrices A and B are $M \times M$ real symmetric ones and B is positive definite ($B \neq I$). The k-th eigenvalue or eigenvector is denoted as λ_k or $\mathbf{y}_k$, respectively ($k = 1, 2, \ldots, M$). The algorithm to solve Eq. (1) proceeds as follows. First, the Cholesky decomposition of B is computed, producing an upper triangle matrix U that satisfies

$$B = U^{\mathrm{T}}U. \tag{2}$$

Then the problem is reduced into a standard eigenvalue problem (SEP)

$$A'\mathbf{z}_k = \lambda_k \mathbf{z}_k \tag{3}$$

with the real-symmetric matrix of

$$A' = U^{-\mathrm{T}} A U^{-1}. \tag{4}$$

When the SEP of Eq. (3) is solved, the eigenvector of the GEP is obtained by

$$\boldsymbol{y}_k = U^{-1} \boldsymbol{z}_k. \tag{5}$$

The above explanation indicates that the whole solver procedure can be decomposed into the two parts of (I) the SEP solver of Eq. (3) and (II) the 'reducer' or the reduction procedure between GEP and SEP by Eqs. (2), (4), (5).

2.2 GEP solvers and hybrid workflows

EigenKernel builds upon three parallel libraries for GEP: ScaLAPACK [13], ELPA [14] and EigenExa [15]. Reflecting the structure of the GEP algorithm stated above, all of the GEP solvers from these libraries consist of two routines, namely, the SEP solver and the reducer. EigenKernel allows the user to select the SEP solver from one library and the reducer from another library, by providing appropriate data format/distribution conversion routines. We call the combination of an SEP solver and a reducer a *hybrid workflow*, or simply *workflow*. Hybrid workflows enable the user to attain maximum performance by choosing the optimal SEP solver and reducer independently.

Among the three libraries adopted by EigenKernel, ScaLAPACK is the *de facto* standard parallel numerical library. However, it was developed mainly in 1990's and thus some of its routines show severe bottlenecks on current supercomputers. Novel solver libraries of ELPA and EigenExa were proposed, so as to overcome the bottlenecks in eigenvalue problems. EigenKernel v.2017 were developed mainly in 2015–2016, so as to realize hybrid workflows among the three libraries. The ELPA code was developed in Europe under the tight-collaboration between computer scientists and material science researchers and its main target application is FHI-aims (Fritz Haber Institute *ab initio* molecular simulations package) [16], a famous electronic-state calculation code. The EigenExa code, on the other hand, was developed at RIKEN in Japan. It is an important fact that the ELPA code has routines optimized for X86, IBM BlueGene and AMD architectures [18], while the EigenExa code was developed so as to be optimal mainly on the K computer. The above fact motivates us to develop a hybrid solver workflow so that we can achieve optimal performance for any problem on any architecture. EigenKernel supports only limited versions of ELPA and EigenExa, since the update of ELPA or EigenExa requires us, sometimes, to modify the interface routine without backward compatibility. EigenKernel v.2017 supports ELPA 2014.06.001 and EigenExa 2.3c. In 2018, EigenKernel was updated in the developer branch on GITHUB and can run on Oakforest-PACS. The benchmark on Oakforest-PACS in this paper was carried out by the code with the commit ID of 373fb83 that appeared at Feb 28, 2018 on GITHUB, except where indicated. The code is called the 'current' code hereafter and supports ELPA v. 2017.05.003 and EigenExa 2.4p1.
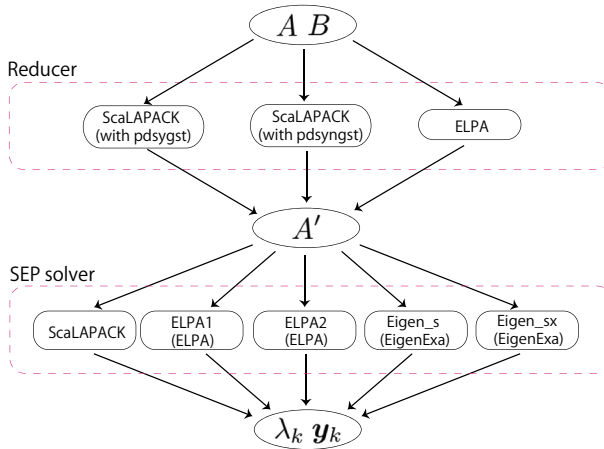
Fig. 2 Possible workflows for GEP solver

Figure 2 shows the possible workflows in EigenKernel. The reducer can be chosen from the ScaLAPACK routine and the ELPA-style routine , and the difference between them is discussed later in this paper. The SEP solver for Eq. (3) can be chosen from the five routines: the ScaLAPACK routine denoted as ScaLAPACK, two ELPA routines denoted as ELPA1 and ELPA2 and two EigenExa routines denoted as Eigen_s and Eigen_sx. The ELPA1 and Eigen_s routines are based on the conventional tridiagonalization algorithm like the ScaLAPACK routine but are different in their implementations. The ELPA2 and Eigen_sx routines are based on non-conventional algorithms for modern architectures. Details of these algorithms can be found in the references (see [14,17,18] for ELPA and [15,19–21] for EigenExa).

EigenKernel focuses on the eight solver workflows for GEP, which are listed as A, $A2$, B, C, D, E, F, G in Table 1. The algorithms of the workflows in Table 1 are explained in our previous paper [6], except the workflow $A2$. The workflow $A2$ is quite similar to $A1$ and the difference between them is only the point that the ScaLAPACK routine pdsyngst, one of the reducer routines, is used in the workflow $A2$, instead of pdsygst in the workflow A. The pdsygst routine is a distributed parallel version of the dsygst routine in LAPACK. This routine repeatedly calls the triangular solver, namely pdtrsm, with a few right-hand sides, and this part often becomes a serious performance bottleneck due to its difficulty of parallelization. The pdsyngst routine is an improved routine that employs the rank 2k update, instead of pdtrsm in pdsygst. Since rank 2k update is more suitable for parallelization, pdsyngst is expected to outperform pdsygst. We note that pdsyngst requires more working space (memory) than pdsygst and that pdsyngst only supports lower triangular matrices; if these requirements are not satisfied, pdsygst is called in pdsyngst. For more details of differences between pdsygst and pdsyngst, refer Refs. [22,23]. All the workflows except the workflow $A2$ are supported in the 'current' code. The workflow $A2$ was added to EigenKernel very recently in a developer version. The workflow $A2$ and other workflows with pdsyngst will appear in a future version. It should be noted that all the 3×5 combinations in

Table 1 Available workflows for GEP in EigenKernel

Workflow	SEP solver	Reducer
A	ScaLAPACK	ScaLAPACK (pdsygst)
A2	ScaLAPACK	ScaLAPACK (pdsyngst)
B	Eigen_sx	ScaLAPACK (pdsygst)
C	ScaLAPACK	ELPA
D	ELPA2	ELPA
E	ELPA1	ELPA
F	Eigen_s	ELPA
G	Eigen_sx	ELPA

Fig. 2 are possible in principle but have not yet been implemented in the code, owing to the limited human resource for programming.

2.3 Features of EigenKernel

As stated in Introduction, EigenKernel prepares basic functions to assist the user to use the optimal workflow for GEP. First, it provides a unified interface to the GEP solvers. When the SEP solver and the reducer are chosen from different libraries, the conversion of data format and distribution are also performed automatically. Second, it outputs detailed performance data such as the elapsed times of internal routines of the SEP solver and reducer for performance analysis. The data file is written in JSON (JavaScript Object Notation) format. This data file is used by the performance prediction tool to be discussed in Sect. 4.

In addition to these, EigenKernel has additional features so as to satisfy the needs among application researchers: (I) It is possible to build EigenKernel only with ScaLA-PACK. This is because there are supercomputer systems in which ELPA or EigenExa are not installed. (II) The package contains a mini-application called EigenKernel-app, a stand-alone application. This mini-application can be used for real researches, as in Ref. [7], if the matrix data are prepared as files in the Matrix Market format.

It is noted that there is another reducer routine called EigenExa-style reducer that appears in our previous paper [6] but is no longer supported by EigenKernel. This is mainly because the code (KMATH_EIGEN_GEV) [24] requires EigenExa but is not compatible with EigenExa 2.4p1. Since this reducer uses the eigendecomposition of the matrix B, instead of the Cholesky decomposition of Eq. (2), its elapsed time is always larger than that of the SEP solver. Such a reducer routine is not efficient, at least judging from the benchmark data of the SEP solvers reported in the previous paper [6] and in this paper.

3 Scalability analysis on Oakforest-PACS

In this section, we demonstrate how EigenKernel can be used for performance analysis. We first show the benchmark data of various GEP workflows on Oakforest-PACS

obtained using EigenKernel. Then we analyze the performance bottleneck found in one of the workflows with the help of the detailed performance data output by EigenKernel.

3.1 Benchmarks data for different workflows

The benchmark test was carried out on Oakforest-PACS, so as to compare the elapsed time among the workflows. Oakforest-PACS is a massively parallel supercomputer operated by Joint Center for Advanced High Performance Computing (JCAHPC) [25]. It consists of $P_{\text{max}} \equiv 8,208$ computational nodes connected by the Intel Omni-Path network. Each node has an Intel Xeon Phi 7250 many-core processor with 3TFLOPS of peak performance. Thus, the aggregate peak performance of the system is more than 25PFLOPS. The MPI/OpenMP hybrid parallelism was used and the number of the used nodes is denoted as P. The numbers of MPI processes and OMP threads per node is denoted as $n_{\text{MPI/node}}$ and $n_{\text{OMP/node}}$, respectively. The present benchmark test was carried out with $(n_{\text{MPI/node}}, n_{\text{OMP/node}}) = (1, 64)$. The present benchmark is limited to those within the regular job classes and the maximum number of nodes in the benchmark is $P = P_{\text{quarter}} \equiv 2,048$, a quarter of the whole system, because a job with P_{quarter} nodes is the largest resource available for the regular job classes. A benchmark with up to the full system ($P_{\text{quarter}} < P \leq P_{\text{max}}$) is beyond the regular job classes and is planed in a near future. The test numerical problem is 'VCNT90000' , which appears in ELSES matrix library [26]. The matrix size of the problem is $M = 90,000$. The problem comes from the simulation of a vibrating carbon nanotube (VCNT) calculated by ELSES [27,28], a quantum nanomaterial simulator. The matrices of A and B in Eq. (1) were generated with an *ab initio*-based modeled (tight-binding) electronic-state theory [29].

The calculations were carried out by the workflows of A, $A2$, D, E, F and G. The results of the workflows B and C can be estimated from those of the other workflows, since the SEP solver and the reducer in the two workflows appear among other workflows. The above discussion implies that the two workflows are not efficient.

The benchmark data is summarized in Fig. 3. The total elapsed time $T(P)$ is plotted in Fig. 3a and that of the SEP solver $T_{\text{SEP}}(P)$ or the reducer $T_{\text{red}}(P) \equiv T(P) - T_{\text{SEP}}(P)$ is plotted in Fig. 3b, c, respectively. Several points are discussed here; (I) The optimal workflow seems to be D or F as far as among the benchmark data ($16 \leq P \leq P_{\text{quarter}} = 2048$). (II) All the workflows except the workflow of A show a strong scaling property in Fig. 3a, because the elapsed time decreases with the number of nodes P. Figure 3b and c indicates that the bottleneck of the workflow A stems not the SEP solver but the reducer. The bottleneck disappears in the workflow $A2$, in which the routine of pdsygst is replaced by pdsyngst. (III) The ELPA-style reducer is used in the workflows of C, D, E, F, G. Among them, the workflows F and G, hybrid workflows between ELPA and EigenExa, require the conversion process of distributed data, since the distributed data format is different between ELPA and EigenExa [6]. The data conversion process does not dominate the elapsed time, as discussed in Ref. [6]. (IV) We found that the same routine gave different elapsed times in the present benchmark. For example, the ELPA-style reducer is used both in the workflows of D and E but the elapsed time T_{red} with $P = 256$ nodes is significantly
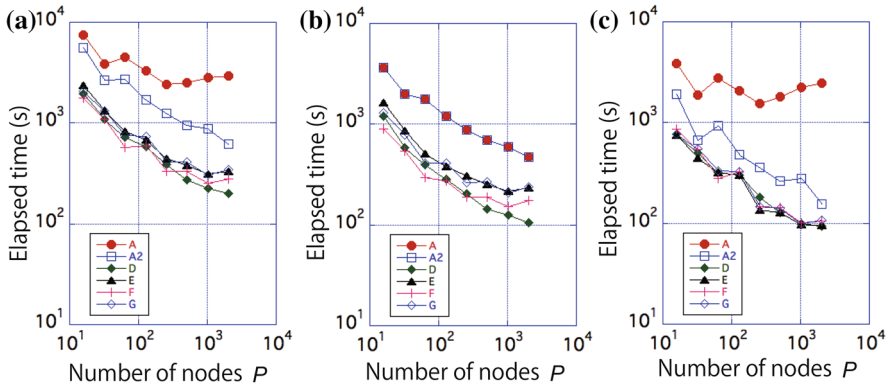
Fig. 3 Benchmark on Oakforest-PACS. The matrix size of the problem is $M = 90,000$. The computation was carried out with $P = 16, 32, 64, 128, 256, 512, 1024, 2048$ nodes in the workflows of A(circle), $A2$(square), D(filled diamond), E(triangle), F(cross), G(open diamond). The elapsed times of **a** for the whole GEP solver, **b** for the SEP solver and **c** for the reducer are plotted

different; The time is $T_{\text{red}} = 182$ s or 137 s, in the workflow of D or E, respectively. The difference stems from the time for the transformation of eigenvectors by Eq. (5), since the time is $T_{\text{trans}-\text{vec}} = 69$ s or 23 s in the workflow of D or E, respectively. The same phenomenon was observed also in the workflow of F and G with $P=64$ nodes, since $(T_{\text{red}}, T_{\text{trans}-\text{vec}})=(277 \text{ s}, 57 \text{ s})$ or $(334 \text{ s}, 114 \text{ s})$ in the workflow of F or G, respectively. Here we should remember that even if we use the same number of nodes P, the parallel computation time $T = T(P)$ can differ from one run to another since the geometry of the used nodes may not be equivalent. Therefore, the benchmark test for multiple runs with the same number of used nodes should be carried out in a near future. (V) The algorithm has several tuning parameters, such as $n_{\text{MPI/node}}, n_{\text{OMP/node}}$, though these parameters are fixed in the present benchmark. A more extensive benchmark with different values of the tuning parameters is one of possible area of investigation in the future for faster computations.

3.2 Detailed performance analysis of the pure ScaLAPACK workflows

Detailed performance data are shown in Fig. 4a for the two pure ScaLAPACK workflows A and $A2$. In the workflow A, the total elapsed time, denoted as T, is decomposed into six terms; the five terms are those for the ScaLAPACK routines of pdsytrd, pdsygst, pdstedc, pdormtr and pdotrf. The elapsed times for these routines are denoted as $T^{(\text{pdsygst})}(P)$, $T^{(\text{pdsytrd})}(P)$, $T^{(\text{pdstedc})}(P)$, $T^{(\text{pdotrf})}(P)$ and $T^{(\text{pdormtr})}(P)$, respectively. The elapsed time for the rest part is defined as $T^{(\text{rest})} \equiv T - T^{(\text{pdsygst})} - T^{(\text{pdsytrd})} - T^{(\text{pdstedc})} - T^{(\text{pdotrf})} - T^{(\text{pdormtr})}$. In the workflow $A2$, the same definitions are used, except the point that pdsygst is replaced by pdsyngst. These timing data are output by EigenKernel automatically in JSON format.

Figure 4 indicates that the performance bottleneck of the workflow A is caused by the routine of pdsygst, in which the reduced matrix A' is generated by Eq. (4). A possible cause for the low scalability of pdsygst is the algorithm used in it, which
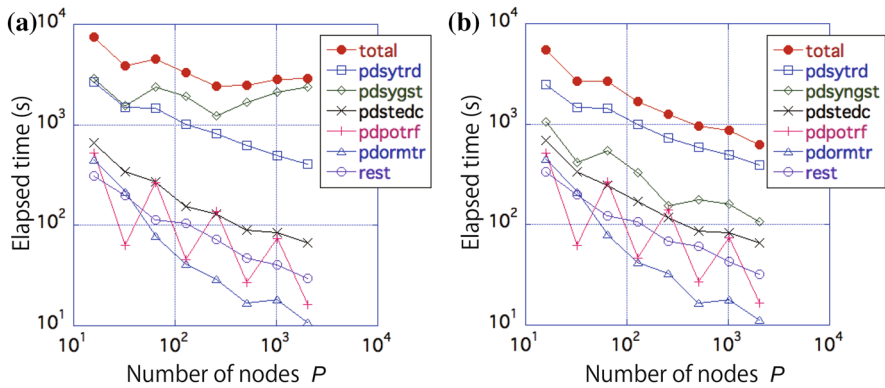
Fig. 4 Benchmark on Oakforest-PACS with the workflows **a** A and **b** $A2$. The matrix size of the problem is $M = 90,000$. The computation was carried out with $P = 16, 32, 64, 128, 256, 512, 1024, 2048$ nodes. The graph shows the elapsed times for the total GEP solver (filled circle), pdsytrd (square), pdsygst in the workflow A or pdsyngst in the workflow $A2$ (diamond), pdstedc (cross), pdpotrf (plus), pdormtr (triangle) and the rest part (open circle)

exploits the symmetry of the resulting matrix A' to reduce the computational cost [30]. Although this is optimal on sequential machines, it brings about some data dependency and can cause a scalability problem on massively parallel machines. The workflow $A2$ uses pdsyngst, instead of pdsygst and improves the scalability, as expected in the last paragraph of Sect. 2.2. We should remember that the ELPA-style reducer is best among the benchmark in Fig. 3c, since the ELPA-style reducer forms the inverse matrix U^{-1} explicitly and computes Eq. (4) directly using matrix multiplication [18]. While this algorithm is computationally more expensive, it has a larger degree of parallelism and can be more suited for massively parallel machines.

In principle, the performance analysis such as given here could be done manually. However, it requires the user to insert a timer into every internal routine and output the measured data in some organized format. Since EigenKernel takes care of these kinds of troublesome tasks, it makes performance analysis easier for non-expert users. In addition, since performance data obtained in practical computation is sometimes valuable for finding performance issues that rarely appears in development process, this feature can contribute to the co-design of software.

4 Performance prediction

4.1 The concept

This section proposes to use Bayesian inference as a tool for performance prediction, in which the elapsed time is predicted from teacher data or existing benchmark data. The importance of performance modeling and prediction has long been recognized by library developers. In fact, in a classical paper published in 1996 [31], Dackland and Kågström write, "we suggest that any library of routines for scalable high performance computer systems should also include a corresponding library of

Fig. 5 Schematic figure of performance prediction, in which the elapsed time of a routine T is written as the function of the number of processor nodes P ($T \equiv T(P)$). The figure illustrates the performance extrapolation that gives a typical behavior with a minimum
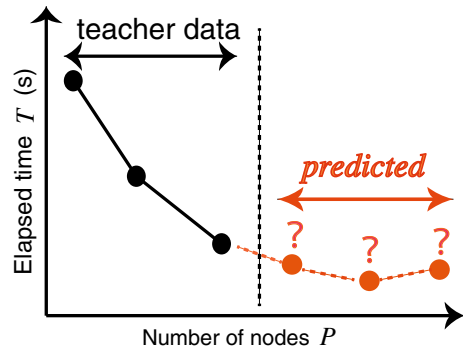
performance models". However, there have been few parallel matrix libraries equipped with performance models so far. The performance prediction function to be described in this section will be incorporated in the future version of EigenKernel and will form one of the distinctive features of the middleware.

Performance prediction can be used in a variety of ways. Supercomputer users are required to prepare a computation plan that requires estimated elapsed time, but it is difficult to predict the elapsed time from hardware specifications, such as peak performance, memory and network bandwidths and communication latency. The performance prediction realizes high usability since it can predict the elapsed time without requiring huge benchmark data. Moreover, the performance prediction enables an auto-optimization (autotuning) function, which selects the optimal workflow in EigenKernel automatically given the target machine and the problem size. Such high usability is crucial, for example, in electronic state calculation codes, because the codes are used not only among theorists but also among experimentalists and industrial researchers who are not familiar with HPC techniques.

The present paper focuses not only on performance interpolation but also on extrapolation, which predicts the elapsed time at a larger number of nodes from the data at a smaller number of nodes. This is shown schematically in Fig. 5. An important issue in the extrapolation is to predict the speed-up 'saturation', or the phenomenon that the elapsed time may have a minimum, as shown in Fig. 5. The extrapolation technique is important, since we have only few opportunities to use the ultimately large computer resources, like the whole system of the K computer or Oakforest-PACS. A reliable extrapolation technique will encourage real researchers to use large resources.

4.2 Performance models

The performance prediction will be realized, when a reliable performance model is constructed for each routine, so as to reflect the algorithm and architecture properly. The present paper, as an early-stage research, proposes three simple models for the elapsed time of the j-th routine $T^{(j)}$ as the function of the number of nodes (the degrees of freedom in MPI parallelism) P; the first proposed model is called generic three-parameter model and is expressed as

$$T^{(j)}(P) = T_1^{(j)}(P) + T_2^{(j)}(P) + T_3^{(j)}(P) \tag{6}$$

$$T_1^{(j)}(P) \equiv \frac{c_1^{(j)}}{P} \tag{7}$$

$$T_2^{(j)}(P) \equiv c_2^{(j)} \tag{8}$$

$$T_3^{(j)}(P) \equiv c_3^{(j)} \log P \tag{9}$$

with the three fitting parameters of $\{c_i^{(j)}\}_{i=1,2,3}$. The terms of $T_1^{(j)}$ or $T_2^{(j)}$ stand for the time in ideal strong scaling or in non-parallel computations, respectively. The model of $T^{(j)} = T_1^{(j)} + T_2^{(j)}$ is known as Amdahl's relation [32]. The term of $T_3^{(j)}$ stand for the time of MPI communications. The logarithmic function was chosen as a reasonable one, since the main communication pattern required in dense matrix computations is categorized into *collective* communication (e.g. `MPI_Allreduce` for calculating inner product of a vector), and such communication routine is often implemented as a sequence of point-to-point communications along with a binary tree, whose total cost is proportional to $\log_2 P$ [33].

The generic three-parameter model in Eq. (6) can give, unlike Amdahl's relation, the minimum schematically shown in Fig. 5. It is noted that the real MPI communication time is not measured to determine the parameter $c_3^{(j)}$, since it would require detailed modification of the source code or the use of special profilers. Rather, all the parameters $\{c_i^{(j)}\}_{i=1,2,3}$ are estimated simultaneously from the total elapsed time $T^{(j)}$ using Bayesian inference, as will be explained later.

The second proposed model is called generic five-parameter model and is expressed as

$$T^{(j)}(P) = T_1^{(j)}(P) + T_2^{(j)}(P) + T_3^{(j)}(P) + T_4^{(j)}(P) + T_5^{(j)}(P) \tag{10}$$

$$T_4^{(j)}(P) \equiv \frac{c_4^{(j)}}{P^2} \tag{11}$$

$$T_5^{(j)}(P) \equiv c_5^{(j)} \frac{\log P}{\sqrt{P}}, \tag{12}$$

with the five fitting parameters of $\{c_i^{(j)}\}_{i=1,2,3,4,5}$. The term of $T_4^{(j)} (\propto P^{-2})$ is responsible for the 'super-linear' behavior in which the time decays faster than $T_1^{(j)} (\propto P^{-1})$. The super-linear behavior can be seen in several benchmarks data [34]. The term of $T_5^{(j)} (\propto \log P/\sqrt{P})$ expresses the time of MPI communications for matrix computation; when performing matrix operations on a 2-D scattered $M \times M$ matrix, the size of the submatrix allocated to each node is $(M/\sqrt{P}) \times (M/\sqrt{P})$. Thus, the communication volume to send a row or column of the submatrix is $M/\sqrt{P}$. By taking into account the binary tree based collective communication and multiplying $\log P$, we obtain the term of $T_5^{(j)}(P)$. The term decays slower than $T_1^{(j)} (\propto P^{-1})$.

The third proposed model results when the MPI communication term of $T_3^{(j)}$ in Eq. (6) is replaced by a linear function;

$$T^{(j)}(P) = T_1^{(j)}(P) + T_2^{(j)}(P) + \tilde{T}_3^{(j)}(P) \qquad (13)$$

$$\tilde{T}_3^{(j)}(P) \equiv \tilde{c}_3^{(j)} P. \qquad (14)$$

The model is called linear-communication-term model. We should say that this model is fictitious, because no architecture or algorithm used in real research gives a linear term in MPI communication, as far as we know. The fictitious three-parameter model of Eq. (13) was proposed so as to be compared with the other two models. Other models for MPI routines are proposed [35,36], and comparison with these models will be also informative, which is one of our future works.

4.3 Parameter estimation by Bayesian inference

In this paper, Bayesian inference is carried out by the Markov Chain Monte Carlo (MCMC) iterative procedure and the uncertainty is included in predicted values. Here the uncertainty is formulated by the normal distribution, as usual. The result appears as the posterior probability distribution of the elapsed time T. Hereafter, the median is denoted as T_{med} and the upper and lower limits of 95% Highest Posterior Density (HPD) interval are denoted as $T_{\mathrm{up-lim}}$ and $T_{\mathrm{low-lim}}$, respectively. The predicted value appears both in the single (median) value of T_{med} and the interval of $[T_{\mathrm{low-lim}}, T_{\mathrm{up-lim}}]$.

In the present paper, Bayesian inference was carried out under the preknowledge that each term of $\{T_i^{(j)}\}_i$ is a fraction of the elapsed time and therefore each parameter of $\{c_i^{(j)}\}_i$ should be non-negative ($T_i^{(j)} \geq 0$, $c_i^{(j)} \geq 0$). The procedure was realized by Python with the Bayesian inference module of PyMC ver. 2.36, which employs the MCMC iterative procedure. The method is standard and the use of PyMC is not crucial. The details of the method are explained briefly here; (I) The parameters of $\{c_i^{(j)}\}_i$ are considered to have uncertainty and are expressed as probability distributions. The prior distribution should be chosen and the posterior distribution will be obtained by Bayesian inference. The prior distribution of the parameters of $c_i^{(j)}$ is set to the uniform distribution in the interval of $[0, c_{i(\mathrm{lim})}^{(j)}]$, where $c_{i(\mathrm{lim})}^{(j)}$ is an input. The upper limit of $c_{i(\mathrm{lim})}^{(j)}$ should be chosen so that the posterior probability distribution is so localized in the region of $c_i^{(j)} \ll c_{i(\mathrm{lim})}^{(j)}$. The values of $c_{i(\mathrm{lim})}^{(j)}$ depend on problem and will appear in the next subsection with results. (II) The present Bayesian inference was carried out for the logscale variables $(x, y) \equiv (\log P, \log T)$, instead of the original variable of (P, T). The prediction on the logscale variables means that the uncertainty in the normal distribution appears on the logscale variable y. When the original variables are used, the width of the HPD interval ($|T_{\mathrm{up-lim}} - T_{\mathrm{low-lim}}|$) is on the same order among different nodes and is much larger than the median value T_{med} for data with a large number of nodes ($|T_{\mathrm{up-lim}} - T_{\mathrm{low-lim}}| \gg T_{\mathrm{med}}$). This phenomenon is related to the uncertainty formulation in the normal distribution. We thought of the use of the logscale variables, since we discuss the benchmark data on the logscale variables as Fig. 3. Another choice of the transformed variables may be a possible future issue. (III) The uncertainty in the normal distribution is characterized by the standard deviation $\sigma^{(j)}$. The parameter $\sigma^{(j)}$ is also treated as a probability distribution and its prior

Table 2 Measured elapsed times in seconds for the matrix problem of 'VCNT22500' solved by the workflow *A* on the K computer

# Nodes	Total	pdsytrd	pdsygst	pdstedc	pdormtr	pdotrf	Rest
4	1872.7	1562.2	61.589	58.132	122.91	20.679	47.190
16	240.82	129.09	37.012	21.341	24.624	8.0851	20.670
64	103.18	44.494	24.584	9.9665	7.1271	3.3122	13.692
256	63.029	21.325	20.509	5.8159	3.4131	2.2474	9.7189
1024	55.592	17.524	17.242	6.1105	2.6946	3.1462	8.8753
4096	70.459	20.479	21.169	6.7494	3.9326	7.9400	10.189
10000	140.89	29.003	49.870	17.714	9.9534	19.817	14.536

The elapsed times are measured as a function of the number of nodes P for the total solver time $T(P)$ and the six routines of $T_{\text{pdsygst}}(P)$, $T_{\text{pdsytrd}}(P)$, $T_{\text{pdstedc}}(P)$, $T_{\text{pdotrf}}(P)$, $T_{\text{pdormtr}}(P)$, $T_{\text{rest}}(P)$

distribution is set to be the uniform one in the interval of $[0, \sigma_{\text{limit}}]$ with a given value of the upper bound $\sigma_{\text{limit}}^{(j)} = 0.5$.

The MCMC procedure consumed only one or a couple of minute(s) by note PC with the teacher data of existing benchmarks. The histogram of Monte Carlo sample data are obtained for the parameters of $\{c_i^{(j)}\}_i$, $\sigma^{(j)}$ and the elapsed time of $T^{(j)}(P)$ and form approximate probability distributions for each quantity. The MCMC iterative procedure was carried out for each routine independently and the iteration number is set to be $n_{\text{MC}} = 10^5$. In the prediction procedure of the j-th routine, each iterative step gives the set of parameter values of $\{c_i^{(j)}\}_i$, $\sigma^{(j)}$ and the elapsed time of $T^{(j)}(P)$, according to the selected model. We discarded the data in the first $n_{\text{MC}}^{(\text{early})} = n_{\text{MC}}/2$ steps, since the Markov chain has not converged to the stationary distribution during such early steps. After that, the sampling data were picked out with an interval of $n_{\text{interval}} = 10$ steps. The number of the sampling data, therefore, is $n_{\text{sample}} = (n_{\text{MC}} - n_{\text{MC}}^{(\text{early})})/n_{\text{interval}} = 5000$. Hereafter the index among the sample data is denoted by k ($\equiv 1, 2, \ldots, n_{\text{sample}}$). The k-th sample data consist of the set of the values of $\{c_i^{(j)}\}_i$, $\sigma^{(j)}$ and $T^{(j)}(P)$ and these values are denoted as $\{c_i^{(j)[k]}\}_i$, $\sigma^{(j)[k]}$ and $T^{(j)[k]}(P)$, respectively. The sampling data set of $\{T^{(j)[k]}(P)\}_{k=1,\ldots,n_{\text{sample}}}$ form the histogram or the probability distribution for $T^{(j)}(P)$. The probability distributions for the model parameters of $\{c_i^{(j)}\}_i$ are obtained in the same manner and will appear later in this section. The sample data for the total elapsed time is given by the sum of those over the routines;

$$T^{[k]}(P) = \sum_j T^{(j)[k]}(P). \tag{15}$$

Finally, the median T_{med} and the upper and lower limits of 95% HPD interval, $(T_{\text{up-lim}}, T_{\text{low-lim}})$, are obtained from the histogram of $\{T^{[k]}(P)\}_k$ as the predicted values.

4.4 Results

The prediction was carried out on the K computer for the matrix problem of 'VCNT22500', which appears in ELSES matrix library [26]. The matrix size is $M = 22,500$. The workflow A, pure ScaLAPACK workflow, was used with the numbers of nodes for $P = 4, 16, 64, 256, 1024, 4096, 10,000$. The elapsed times were measured for the total elapsed time of $T(P)$ and the six routines of $T^{(\text{pdsygst})}(P)$, $T^{(\text{pdsytrd})}(P)$, $T^{(\text{pdstedc})}(P)$, $T^{(\text{pdotrf})}(P)$, $T^{(\text{pdormtr})}(P)$ and $T^{(\text{rest})}(P)$. The values are shown in Table 2. The measured data of the total elapsed time of $T(P)$ shows the speed-up 'saturation' or the phenomenon that the elapsed time shows a minimum at $P = 1024$. The saturation is reasonable from HPC knowledge, since the matrix size is on the order of $M = 10^4$ and efficient parallelization can not be expected for $P \geq 10^3$.

Figure 6a–c shows the result of Bayesian inference, in which the teacher data is the measured elapsed times of the six routines of $T^{(\text{pdsygst})}(P)$, $T^{(\text{pdsytrd})}(P)$, $T^{(\text{pdstedc})}(P)$, $T^{(\text{pdotrf})}(P)$, $T^{(\text{pdormtr})}(P)$ and $T^{(\text{rest})}(P)$ at $P = 4, 16, 64$. In the MCMC procedure, the value of $c_{i(\text{lim})}^{(j)}$ is set to $c_{i(\text{lim})}^{(j)} = 10^5$ among all the routines and the posterior probability distribution satisfies the locality of $c_i^{(j)} \ll c_{i(\text{lim})}^{(j)}$. One can find that the generic three-parameter model of Eq. (6) and the generic five-parameter model of Eq. (10) commonly predict the speed-up saturation successfully at $P = 256 \sim 1024$, while the linear-communication-term model does not. Examples of the posterior probability distribution are shown in Fig. 6d, for $c_1^{(\text{pdsytrd})}$ and $c_4^{(\text{pdsytrd})}$ in the five-parameter model.

Figure 7 shows the detailed analysis of the performance prediction in Fig. 6. Here the performance prediction of $T^{(\text{pdsytrd})}$ and $T^{(\text{pdsygst})}$ is focused on, since the total time is dominated by these two routines, as shown in Table 2. The elapsed time of $T^{(\text{pdsytrd})}$ is predicted in Fig. 7a, b by the generic three-parameter model and the generic five-parameter model, respectively. The importance of the five-parameter model can be understood, when one see the probability distribution of $c_1^{(\text{pdsytrd})}$ and $c_4^{(\text{pdsytrd})}$ in Fig. 6d. Since the probability distribution of $c_4^{(\text{pdsytrd})}$ has a peak at $c_4^{(\text{pdsytrd})} \approx 2.3 \times 10^4$, the term of $T_4^{(j)}(P)$ in Eq. (11), the super-linear term, should be important. The contribution at $P = 10$, for example, is estimated to be $T_4(P = 10) = c_4/P^2 \approx (2.3 \times 10^4)/10^2 \approx 2 \times 10^2$. The above observation is consistent with the fact that the measured elapsed time shows the super-linear behavior between $P = 4, 16$ ($T(P = 4)/T(P = 16)$=(1872.7 s)/(240.82 s) ≈ 7.78) and the generic five-parameter model reproduces the teacher data, the data at P=4, 16, 64, better than the generic three-parameter model. The elapsed time of $T^{(\text{pdsygst})}$ is predicted in Fig. 7c, d by the generic three-parameter model and the generic five-parameter model, respectively. The prediction by the generic five-parameter model seems to be better than that by the three-parameter model. Unlike $T^{(\text{pdsytrd})}$, $T^{(\text{pdsygst})}$ is a case in a poor strong scaling, since the ideal scaling behavior ($T \propto 1/P$) or the super-linear behavior ($T \propto 1/P^2$) can not be seen even at the small numbers of used nodes ($P = 4, 16$).
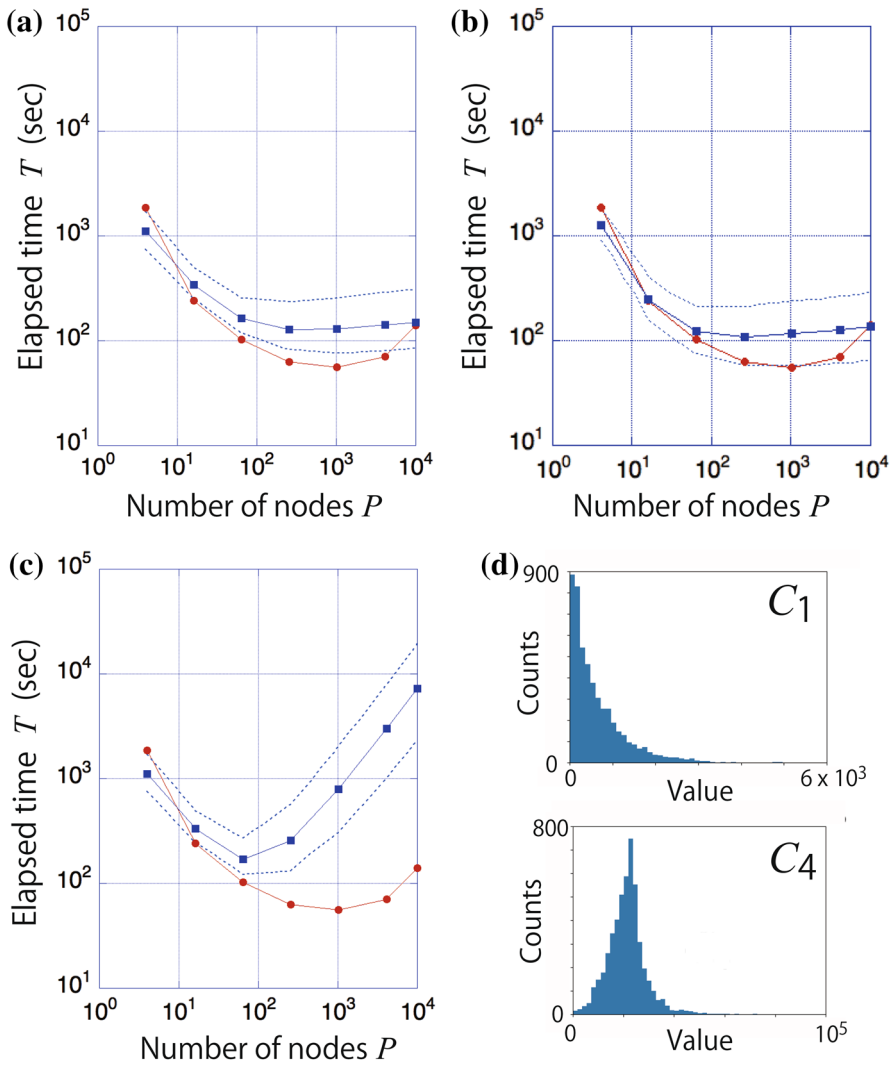
Fig. 6 Performance prediction with the three models of **a** the generic three-parameter model, **b** the generic five-parameter model, and **c** the linear-communication-term model. The workflow A, pure ScaLAPACK workflow, was used with the numbers of nodes for $P = 4, 16, 64, 256, 1024, 4096, 10,000$. The data is generated for the generalized eigenvalue problem of 'VCNT22500' on the K computer. The measured elapsed time for the total solver is drawn by circles. The predicted elapsed time is drawn by square for the median and by the dashed lines for the upper and lower limits of 95% Highest Posterior Density (HPD) interval. The used teacher data are the elapsed times of the six routines of $T^{(\text{pdsytrd})}$, $T^{(\text{pdsygst})}$, $T^{(\text{pdstedc})}$, $T^{(\text{pdotrf})}$, $T^{(\text{pdormtr})}$ and $T^{(\text{rest})}$ at $P = 4, 16, 64$. **d** The posterior probability distribution of $c_1^{(\text{pdsytrd})}$ (upper panel) and $c_4^{(\text{pdsytrd})}$ (lower panel) within the five-parameter model
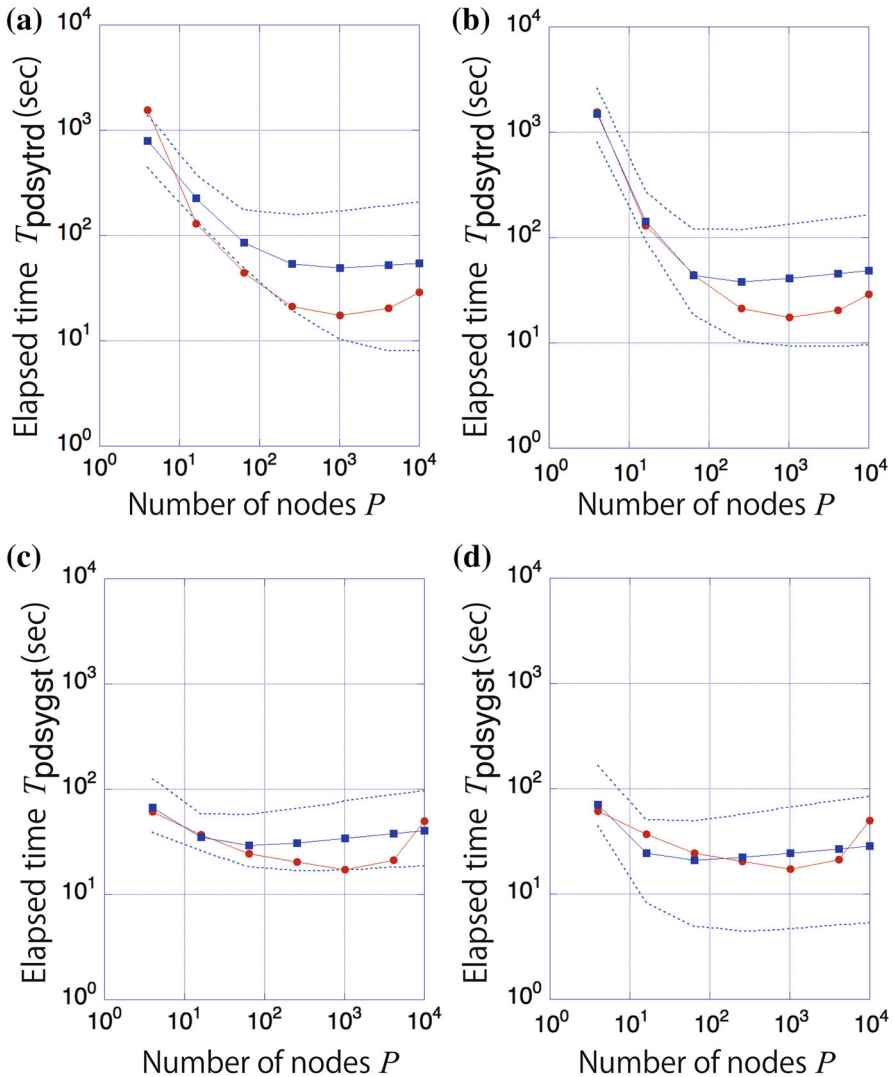
Fig. 7 Detailed analysis of the performance prediction in Fig. 6. The performance prediction for the elapsed time of pdsytrd ($T^{(\text{pdsytrd})}$) by **a** the generic three-parameter model and **b** the generic five-parameter model. The performance prediction for the elapsed time of pdsygst ($T^{(\text{pdsygst})}$) by **c** the generic three-parameter model and **d** the generic five-parameter model

5 Discussion on performance prediction

This subsection is devoted to discussions on the present performance prediction method.
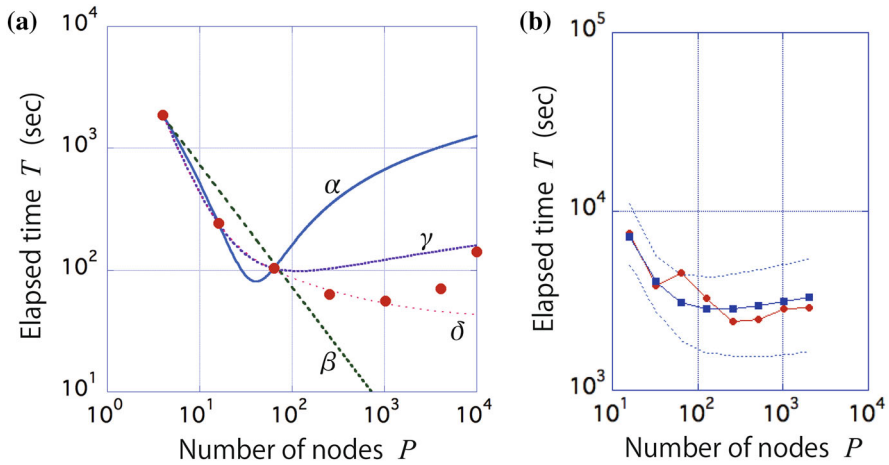
Fig. 8 a Performance prediction by least squares methods of 'VCNT22500' on the K computer. The circles indicate the measured elapsed times for $P = 4, 16, 64, 256, 1024, 4096, 10,000$, while the other curves are determined by the least squares methods. The line labelled α or β is determined without or with non-negative constraint, respectively, by the three-parameter model with the three teacher data at $P = 4, 16, 64$. The line labelled γ is determined with non-negative constraint by the five-parameter model with the three teacher data. The line labelled δ is determined with non-negative constraint by the five-parameter model with the five teacher data at $P = 4, 16, 64, 256, 1024$. **b** Performance prediction for the generalized eigenvalue problem of 'VCNT90000' on Oakforest-PACS. The generic five-parameter model is used. The red circles indicate the measured elapsed times for $P=16, 32, 64, 128, 256, 512, 1024$ and 2048. The predicted elapsed time is drawn by square for the median and by dashed lines for the upper and lower limits of 95% Highest Posterior Density (HPD) interval. The teacher data are the data at $P =16, 32, 64, 128$

5.1 Comparison with least squares method

In this subsection, the present MCMC method is compared with the least squares methods, so as to clarify the properties of the present method. The least squares methods have been applied to the performance modeling of numerical libraries. In fact, most of the recent studies on performance modeling [37–40] rely on the least squares methods to fit the model parameters.

In Fig. 8a, the total elapsed time of $T(P)$ for VCNT22500 on the K computer is fitted, as in Fig. 6. The line labelled α is the fitted curve, without any constraint, by the three-parameter model with the teacher data at $P = 4, 16, 64$. The fitting procedure determines the parameters as $(c_1, c_2, c_3) = (c_1^{(LSQ0)}, c_2^{(LSQ0)}, c_3^{(LSQ0)}) \equiv (1.06 \times 10^4, -1.14 \times 10^3, 2.60 \times 10^2)$. The fitted curve reproduces the teacher data, the data at $P = 4, 16, 64$, exactly, but deviates severely from the measured values at $P \geq 256$. The fitted value of c_2 is negative, since the method ignores the preknowledge of the non-negative constraint ($c_2 \geq 0$). A non-negative least-squares curve fitting was also carried out by the module lsqnonneg in MATLAB for the linear problem on (P, T). A non-linear problem may appear as an extension of performance model, as discussed in the next subsection. The present fitting procedures consume typically 1 s, which is smaller than that of the MCMC method. The line labelled β is fitted by the three-parameter model with the three teacher data at $P = 4, 16, 64$. The fitted values are $(c_1, c_2, c_3) = (c_1^{(LSQ1)}, c_2^{(LSQ1)}, c_3^{(LSQ1)}) \equiv (7.27 \times 10^3, 0.00, 0.00)$.

The line labelled γ is fitted by the five-parameter model with the three teacher data. The fitted values are $(c_1, c_2, c_3, c_4, c_5) = (c_1^{(\text{LSQ2})}, c_2^{(\text{LSQ2})}, c_3^{(\text{LSQ2})}, c_4^{(\text{LSQ2})}, c_5^{(\text{LSQ2})})$ $\equiv (1.65 \times 10^3, 0.00, 1.73 \times 10, 2.30 \times 10^4, 0.00)$. The fitted curve is comparable to the median values in the MCMC method of Fig. 6b. It is noted that the fitting problem is underdetermined, having smaller number of teacher data ($n_{\text{teacher}} = 3$) than that of fitting parameters ($n_{\text{param}} = 5$). The line labelled δ is fitted by the five-parameter model with the five teacher data. The fitted values are $(c_1, c_2, c_3, c_4, c_5) = (c_1^{(\text{LSQ3})}, c_2^{(\text{LSQ3})}, c_3^{(\text{LSQ3})}, c_4^{(\text{LSQ3})}, c_5^{(\text{LSQ3})}) \equiv (5.81 \times 10^2, 0.00, 3.34, 2.61 \times 10^4, 1.36 \times 10^2)$. The line labelled δ was drawn just for reference, because the present purpose is the prediction or extrapolation of the performance saturation at $P \approx 10^3$ from the data with smaller numbers of nodes ($P \ll 10^3$) and the fitting with the teacher data at $P = 1024$ is outside the scope of our interest.

In addition, the least squares method by the three-parameter model without constraint can be realized in the framework the MCMC method, if the parameters of $\{c_i\}_i$ are set to be the interval of $[-c_{\text{lim}}, c_{\text{lim}}]$ with $c_{\text{lim}} = 10^5$ and that for σ is set to be the interval of $[0, \sigma_{\text{lim}}]$ with $\sigma_{\text{lim}} = 10^{-5}$. The above prior distribution means that the non-negative condition ($c_i \geq 0$) is ignored and the method is required to reproduce the teacher data exactly ($\sigma_{\text{lim}} \approx 0$). In this case, the MCMC procedure was carried out for the variables $(x, T) \equiv (\log P, T)$, unlike in Sect. 4.3, because the non-negative condition is not imposed on T and we cannot use $y \equiv \log T$ as a variable. We confirmed the above statement by the MCMC procedure. As results, the median is located at $(c_1, c_2, c_3) = (c_1^{(\text{LSQ0})}, c_2^{(\text{LSQ0})}, c_3^{(\text{LSQ0})})$ and the width of the 95 % Highest Posterior Density (HPD) interval is 10^{-3} or less for each coefficient.

The above numerical experiments reflect the similarity and difference between the MCMC method and the least square methods. In general, the MCMC method has the following three properties; (i) The non-negative constraint on the elapsed time ($T > 0$) can be imposed as preknowledge. (ii) The uncertainty or error can be taken into account both for the teacher data and the predicted data. The uncertainty of the predicted data appears as the HDR interval in Fig. 6a, b, while the fittings in Fig. 8a do not contain the uncertainty. (iii) The iterative procedure is guaranteed to converge to the unique posterior distribution.

5.2 Possible extension to a non-linear modeling

Although the generic five-parameter model seems to be the best among the three proposed models, the model is still limited in applicability. Figure 8b shows the performance prediction by the five-parameter model for the benchmark data by the workflow A in Fig. 4, a data for the problem of 'VCNT90000' on Oakforest-PACS. The data at $P = 16, 32, 64, 128$ are the teacher data. In the present case, the upper limit is set to $c_{i(\text{lim})}^{(j)} = 10^5$ for $i = 1, 2, 3, 5$ and $c_{4(\text{lim})}^{(j)} = 10^7$ and the posterior probability distribution satisfies the locality of $c_i^{(j)} \ll c_{i(\text{lim})}^{(j)}$. The predicted data fails to reproduce the local maximum at $P = 64$ in the measured data, since the model can have only one (local) minimum and no (local) maximum. If one would like to overcome the above limitation, a candidate for a flexible model may be one with case classification;

$$T(P) = \begin{cases} T_{\text{model}}^{(\alpha)}(P) & P < P_{\text{c}} \\ T_{\text{model}}^{(\beta)}(P) & P > P_{\text{c}} \end{cases}, \tag{16}$$

where $T_{\text{model}}^{(\alpha)}(P)$ and $T_{\text{model}}^{(\beta)}(P)$ are considered to be independent models and the threshold number of P_{c} is also a fitting parameter. The model in Eq. (16) will give a non-linear problem on (T, P) for fitting parameters, unlike that in Eq. (10). A model with case classification will be fruitful from the algorithm and architecture viewpoints. An example is the case where the target numerical routine switches the algorithms according to the number of used nodes. Another example is the case where the nodes in a rack are tightly connected and parallel computation within these nodes is quite efficient. From the application viewpoint, however, the prediction in Fig. 8b is still meaningful, since the extrapolation implies that an efficient parallel computation cannot be expected at $P \geq 128$.

5.3 Discussions on methodologies and future aspect

This subsection is devoted to discussion on methodologies and future aspects.

(I) The proper values of $c_{i(\text{lim})}^{(j)}$ should be chosen in each problem and here we propose a way to set the values automatically. The possible maximum value of $c_i^{(j)}$ appears, when the elapsed time of the j-th routine is governed only by the i-th term of the given model ($T^{(j)}(P) \approx T_i(P)^{(j)}$). We consider, for example, the case in which the first term is dominant ($T^{(j)}(P) \approx c_1^{(j)}/P$) and the possible maximum value of $c_1^{(j)}$ is given by $c_1^{(j)} \approx PT^{(j)}(P)$. Therefore the limit of $c_{1(\text{lim})}^{(j)}$ can be chosen to be $c_{1(\text{lim})}^{(j)} \geq PT^{(j)}(P)$ among the teacher data of $(P, T^{(j)}(P))$. The locality of $c_1^{(j)} \ll c_{1(\text{lim})}^{(j)}$ should be checked for the posterior probability distribution. We plan to use the method in a future version of our code.

(II) In parallel computations, the elapsed times may be different among multiple runs, because the geometries of the used nodes are in general not equivalent. Such uncertainty of the measured elapsed time can be treated in Bayesian inference by, for example, setting the parameter $\sigma_{\text{lim}}^{(j)}$ appropriately based on the sample variance of the multiple measured data or other preknowledge. In the present paper, however, the optimal choice of $\sigma_{\text{lim}}^{(j)}$ is not discussed and is left as a future work.

(III) It is noted that ATMathCoreLib, an autotuning tool [41,42], also uses Bayesian inference for performance prediction. However, it is different from our approach in two aspects. First, it uses Bayesian inference to construct a reliable performance model from noisy observations [43]. So, more emphasis is put on interpolation than on extrapolation. Second, it assumes normal distribution both for the prior and posterior distributions. This enables the posterior distribution to be calculated analytically without MCMC, but makes it impossible to impose non-negative condition of $c_i \geq 0$.

(IV) The present method uses the same generic performance model for all the routines. A possible next step is to develop a proper model for each routine. Another possible theoretical extension is performance prediction among different problems. The elapsed time of a dense matrix solver depends on the matrix size M, as well as

on P, so it would be desirable to develop a model to express the elapsed time as a function of both the number of nodes and the matrix size ($T = T(P, M)$). For example, the elapsed time of the tridiagonalization routine in EigenExa on the K computer is modeled as a function of both the matrix size and the number of nodes [40]. In this study, several approaches to performance modeling are compared depending on the information available for modeling, and some of them accurately estimate the elapsed time for a give condition (i.e. matrix size and node count). The use of such a model will given more fruitful prediction, in particular, for the extrapolation.

6 Summary and future outlook

We developed an open-source named middleware EigenKernel for the generalized eigenvalue problem that realizes high scalability and usability, responding to the solid need in large-scale electronic state calculations. Benchmark results on Oakforest-PACS shows the middleware enables us to choose the optimal scalable solver from ScaLAPACK, ELPA and EigenExa routines. We presented an example that the detailed performance data provided by EigenKernel helps users to detect performance issues without additional effort such as code modification. For high usability, a performance prediction function was proposed based on Bayesian inference and the Markov Chain Monte Carlo procedure. We found that the function is applicable not only to performance interpolation but also to extrapolation. For a future look, we can consider a system that gathers performance data automatically every time users call a library routine. Such a system could be realized through a possible collaboration with the supercomputer administrator and will give a greater prediction power to our performance prediction tool, by providing huge set of teacher data. The present approach is general and applicable to any numerical procedure, if a proper performance model is prepared for each routine. The middleware thus developed will form a foundation of the application-algorithm-architecture co-design.

References

1. Shalf, J., Quinlan, D., Janssen, C.: Rethinking hardware–software codesign for exascale systems. Computer **44**, 22–30 (2011)
2. Dosanjh, S., Barrett, R., Doerfler, D., Hammond, S., Hemmert, K., Heroux, M., Lin, P., Pedretti, K., Rodrigues, A., Trucano, T., Luitjens, J.: Exascale design space exploration and co-design. Fut. Gen. Comput. Syst. **30**, 46–58 (2014)
3. FLAGSHIP 2020 Project: Post-K Supercomputer Project. http://www.r-ccs.riken.jp/fs2020p/en/. Accessed 25 Apr 2019
4. CoDEx: Co-Design for Exascale. http://www.codexhpc.org/. Accessed 25 Apr 2019

5. EuroEXA: European Co-Design for Exascale Applications. https://euroexa.eu/. Accessed 25 Apr 2019

6. Imachi, H., Hoshi, T.: Hybrid numerical solvers for massively parallel eigenvalue computation and their benchmark with electronic structure calculations. J. Inf. Process. **24**, 164–172 (2016)

7. Hoshi, T., Imachi, H., Kumahata, K., Terai, M., Miyamoto, K., Minami, K., Shoji, F.: Extremely scalable algorithm for 10^8-atom quantum material simulation on the full system of the K computer. In: Proceeding of 7th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA'16), held in conjunction with SC16: The International Conference for High Performance Computing, Networking, Storage and Analysis Salt Lake City, Utah, pp. 33–40, 13–18 Nov 2016 (2016)

8. EigenKernel: https://github.com/eigenkernel/. Accessed 25 Apr 2019

9. ELSI: https://wordpress.elsi-interchange.org/. Accessed 25 Apr 2019

10. Yu, V.W.-Z., Corsetti, F., Garcia, A., Huhn, W.P., Jacquelin, M., Jia, W., Lange, B., Lin, L., Lu, J., Mi, W., Seifitokaldani, A., Vazquez-Mayagoitia, Á., Yang, C., Yang, H., Blum, V.: ELSI: a unified software interface for Kohn-Sham electronic structure solvers. Comput. Phys. Commun. **222**, 267 (2018)

11. Hirokawa, Y., Boku, T., Sato, S., Yabana, K.: Performance Evaluation of Large Scale Electron Dynamics Simulation under Many-core Cluster based on Knights Landing. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPS Asia 2018), pp. 183–191 (2018)

12. Idomura, Y., Ina, T., Mayumi, A., Yamada, S., Matsumoto, K., Asahi, Y., Imamura, T.: Application of a communication-avoiding generalized minimal residual method to a Gyrokinetic five dimensional Eulerian code on many core platforms. In: Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA'17), held in conjunction with SC17: The International Conference for High Performance Computing, Networking, Storage and Analysis Salt Lake City, Utah, pp. 7:1–7:8 (2017)

13. ScaLAPACK: http://www.netlib.org/scalapack/. Accessed 25 Apr 2019

14. ELPA: Eigenvalue SoLvers for Petaflop-Application. http://elpa.mpcdf.mpg.de/. Accessed 25 Apr 2019

15. EigenExa: High Performance Eigen-Solver. http://www.r-ccs.riken.jp/labs/lpnctrt/en/projects/eigenexa/. Accessed 25 Apr 2019

16. Blum, V., Gehrke, R., Hanke, F., Havu, P., Havu, V., Ren, X., Reuter, K., Scheffler, M.: Ab initio molecular simulations withnumeric atom-centered orbitals. Comput. Phys. Commun. **180**, 2175–2196 (2009). https://aimsclub.fhi-berlin.mpg.de/. Accessed 25 Apr 2019

17. Auckenthaler, T., Blum, V., Bungartz, J., Huckle, T., Johanni, R., Kramer, L., Lang, B., Lederer, H., Willems, P.: Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations. Parallel Comput. **27**, 783–794 (2011)

18. Marek, A., Blum, V., Johanni, R., Havu, V., Lang, B., Auckenthaler, T., Heinecke, A., Bungartz, H.J., Lederer, H.: The ELPA library—scalable parallel eigenvalue solutions for electronic structure theory and computational science. J. Phys. Condens. Mater. **26**, 213201 (2014)

19. Imamura, T., Yamada, S., Machida, M.: Development of a high performance eigensolver on the peta-scale next generation supercomputer system. Progress Nucl. Sci. Technol. **2**, 643–650 (2011)

20. Imamura, T.: The EigenExa Library–High Performance and Scalable Direct Eigensolver for Large-Scale Computational Science, ISC 2014, Leipzig (2014)

21. Fukaya, T., Imamura, T.: Performance evaluation of the EigenExa eigensolver on Oakleaf-FX: Tridiagonalization versus pentadiagonalization. In: Proceedings of 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, pp. 960–969 (2015)

22. Sears, M.P., Stanley, K., Henry, G.: Application of a high performance parallel eigensolver to electronic structure calculations. In: Proceedings of the ACM/IEEE Conference on Supercomputing, IEEE Computer Society, pp. 1–1 (1998)

23. Poulson, J., Marker, B., van de Geijn, R.A., Hammond, J.R., Romero, N.A.: Elemental: A new framework for distributed memory dense matrix computations. ACM Trans. Math. Softw. **39**(13), 1–24 (2013)

24. KMATH_EIGEN_GEV: high-performance generalized eigen solver. http://www.r-ccs.riken.jp/labs/lpnctrt/en/projects/kmath-eigen-gev/. Accessed 25 Apr 2019

25. JCAHPC: Joint Center for Advanced High Performance Computing. http://jcahpc.jp/eng/index.html. Accessed 25 Apr 2019

26. ELSES matrix library. http://www.elses.jp/matrix/. Accessed 25 Apr 2019

27. Hoshi, T., Yamamoto, S., Fujiwara, T., Sogabe, T., Zhang, S.-L.: An order-N electronic structure theory with generalized eigenvalue equations and its application to a ten-million-atom system. J. Phys. Condens. Mater. **21**, 165502 (2012)
28. ELSES: Extra large Scale Electronic Structure calculation. http://www.elses.jp/index_e.html. Accessed 25 Apr 2019
29. Cerda, J., Soria, F.: Accurate and transferable extended Hückel-type tight-binding parameters. Phys. Rev. B **61**, 7965–7971 (2000)
30. Wilkinson, J.H., Reinsch, C.: Handbook for Automatic Computation. Linear Algebra, vol. II. Springer, New York (1971)
31. Dackland, K., Kågström, B.: A Hierarchical Approach for Performance Analysis of ScaLAPACK-Based Routines Using the Distributed Linear Algebra Machine. In: PARA '96 Proceedings of the 3rd International Workshop on Applied Parallel Computing, Industrial Computation and Optimization, pp. 186–195 (1996)
32. Amdahl, G.: Validity of the single processor approach to achieving large-scale computing capabilities. AFIPS Conf. Proc. **30**, 483–485 (1967)
33. Pacheco, P.: Parallel Programming with MPI. Morgan Kaufmann, Massachusetts (1996)
34. Ristov, S., Prodan, R., Gusev, M., Skala, K.: Superlinear Speedup in HPC Systems: Why and When? In: Proceedings of the Federated Conference on Computer Science and Information Systems, pp. 889–898 (2016)
35. Pješivac-Grbović, J., Angskun, T., Bosilca, G., Fagg, E., Gabriel, E., Dongarra, J.: Performance analysis of MPI collective operation. Clust. Comput. **10**, 127–143 (2007)
36. Hoefler, T., Gropp, W., Thakur, R., Träff, L.: Toward performance models of MPI implementations for understanding application scaling issues. Proceeding of the 17th European MPI Users' Group Meeting Conference on Recent Advances in the Message Passing Interface, pp. 21–30 (2010)
37. Peise, E., Bientinesi, B.: Performance Modeling for Dense Linear Algebra. In: Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, pp. 406–416 (2012)
38. Reisert, P., Calotoiu, A., Shudler, S., Wolf, F.: Following the blind seer-creating better performance models using less information. In: Proceedings of Euro-Par 2017: Parallel Processing. Lecture Notes in Computer Science 10417, Springer, New York, pp. 106–118 (2017)
39. Fukaya, T., Imamura, T., Yamamoto, Y.: Performance analysis of the Householder-type parallel tall-skinny QR factorizations toward automatic algorithm selection. In: Proceedings of VECPAR 2014: High Performance Computing for Computational Science - VECPAR 2014, Lecture Notes in Computer Science 8969, Springer, New York, pp. 269–283 (2015)
40. Fukaya, T., Imamura, T., Yamamoto, Y.: A case study on modeling the performance of dense matrix computation: Tridiagonalization in the EigenExa eigensolver on the K computer. In: Proceedings of 2018 IEEE International Parallel and Distributed Processing Symposium Workshop, pp. 1113–1122 (2018)
41. Suda, R.: ATMathCoreLib: Mathematical Core Library for Automatic Tuning, IPSJ SIG Technical Report, 2011-HPC-129(14), 1–12 (2011) (in Japanese)
42. Nagashima, S., Fukaya, T., Yamamoto, Y.: On Constructing Cost Models for Online Automatic Tuning Using ATMathCoreLib: Case Studies through the SVD Computation on a Multicore Processor. Proceedings of the IEEE 10th International Symposium on Embedded Multicore/Many-core Systtems-on-Chip (MCSoC-16), pp. 345–352 (2016)
43. Suda, R.: A Bayesian Method of Online Automatic Tuning, Software Automatic Tuning: From Concept to State-of-the-Art Results, 275–293. Springer, New York (2010)

Affiliations

Kazuyuki Tanaka[1] · Hiroto Imachi[1,2] · Tomoya Fukumoto[1] · Akiyoshi Kuwata[1] · Yuki Harada[1] · Takeshi Fukaya[3] · Yusaku Yamamoto[4] · Takeo Hoshi[1]

✉ Takeo Hoshi
  hoshi@damp.tottori-u.ac.jp

[1] Department of Applied Mathematics and Physics, Tottori University, 4-101 Koyama-Minami, Tottori 680-8552, Japan

[2] Present Address: Preferred Networks, Inc., Tokyo, Japan

[3] Information Initiative Center, Hokkaido University, W5, N11, Kita-ku, Sapporo 060-0811, Japan

[4] Department of Communication Engineering and Informatics, The University of Electro-Communications, 1-5-1 Chofugaoka,, Chofu Tokyo 182-8585, Japan