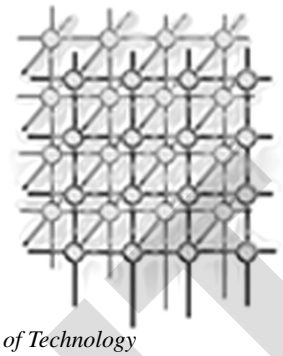
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2000; **00**:1–6 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*



Optimizing process allocation of parallel programs for heterogeneous clusters

Shuichi Ichikawa[1,2,*,†], Sho Takahashi[1,‡], Yuu Kawai[1,§]

[1] *Department of Knowledge-based Information Engineering, Toyohashi University of Technology*
[2] *Intelligent Sensing System Research Center, Toyohashi University of Technology*

SUMMARY

The performance of a conventional parallel application is often degraded by load-imbalance on heterogeneous clusters. Though it is simple to invoke multiple processes on fast PEs to alleviate load-imbalance, the optimal process allocation is not obvious. Kishimoto and Ichikawa presented performance models for HPL (High Performance Linpack), with which the sub-optimal configurations of heterogeneous clusters were actually estimated. Their results on HPL are encouraging, whereas their approach is not yet verified with other applications. This study presents some enhancements of Kishimoto's scheme, which are evaluated with four typical scientific applications: CFD (computational fluid dynamics), FEM (finite element method), HPL (linear algebraic system), and FFT (fast Fourier transform). According to our experiments, our new models (NP-T models) are superior to Kishimoto's models, particularly when non-negative least squares (NNLS) method is used for parameter extraction. The average errors of the derived models were 0.2% for CFD benchmark, 2% for FEM benchmark, 1% for HPL, and 28% for FFT benchmark. This study also emphasizes the importance of predictability in clusters, listing practical examples derived from our work.

KEY WORDS: heterogeneous cluster, high performance computing, performance evaluation, multiprocessing, optimization

*Correspondence to: Department of Knowledge-based Information Engineering, Toyohashi University of Technology, Aichi 441-8580, Japan.
†E-mail: ichikawa@ieee.org
‡Presently, the author is affiliated with NEC Engineering, Ltd.
§Presently, the author is affiliated with NEC Software Chubu, Ltd.

1. Introduction

PC clusters have been widely adopted for scientific and technological computation, from small-scale parallel computing up to supercomputing. Quick and drastic advances in microprocessor technology have supported the advantages of PC clusters, while it simultaneously means that the nodes of PC clusters become obsolete in a very short period of time.

It is thus necessary to continually upgrade the nodes of the PC cluster to keep the cluster competitive and up-to-date, though it requires a substantial budget each time. On the other hand, there are certain demands to maintain the same platform for a certain period. There are always some users who prefer to stay with the existing cluster to avoid the cost of migration, even if others are eager to move to a new cluster for more performance. A practical solution is to add a new cluster that consists of new processors, while keeping the old cluster as long as it is worthwhile.

The resulting cluster becomes heterogeneous. It consists of two or more generations of clusters (sub-clusters), each of which consists of homogeneous processing elements (PEs). Users may use a homogeneous sub-cluster as before, or the whole heterogeneous cluster for maximal computational resources.

Many existing applications are written for distributed memory parallel computers or clusters, which consist of homogeneous processing elements. Since these applications distribute computational workloads equally among PEs, their performances are degraded on a heterogeneous cluster by load-imbalance.

Invoking multiple processes on fast PEs (*multiprocessing scheme*) is a simple and straightforward way to alleviate load-imbalance of parallel applications on heterogeneous clusters. The multiprocessing scheme basically requires no modification of source codes, while providing a reasonable speedup by configuring cluster middleware appropriately.

Despite all these advantages, it is not an easy task to find the optimal process allocation for a multiprocessing scheme. The first problem is that the performance ratio between PEs is not always an integer, while the number of processes is always an integer. The multiprocessing overhead in each PE makes things even more complicated. Another serious problem is communication time. It is not always preferable to use all available PEs, because superfluous communications can prolong the total execution time. Particularly in a heterogeneous cluster, the use of the slowest PEs is not always reasonable. It is thus very important to select the optimal subset of PEs for a given problem size, together with finding the optimal number of processes on each PEs.

Kishimoto and Ichikawa [1][2] presented a scheme to estimate the optimal configuration of a heterogeneous cluster, i.e., the optimal subset of PEs and the optimal process allocation. They constructed the execution-time estimation models from the measurement results of HPL (High Performance Linpack) [3], and showed that the optimal or sub-optimal configurations were actually estimated for various sizes of HPL. Although these studies [1][2] presented some promising results, they are specific to an application. More parallel applications have to be examined to verify the feasibility of the multiprocessing scheme.

The present study examines the multiprocessing scheme with four conventional parallel applications: a CFD (computational fluid dynamics) benchmark, an FEM (finite element method) benchmark, a HPL (linear algebraic system) benchmark, and an FFT (fast Fourier transform) benchmark. We use popular benchmark programs, because (1) they are designed to be fair measures of system performance, (2)

typical examples of important scientific applications, (3) portable and thoroughly verified, and (4) readily available for public use.

In this study, we apply Kishimoto's models (N-T and P-T models) [1][2] to these four applications, and further propose some improvements, including a new model (NP-T model) that is superior to Kishimoto's.

The rest of this paper is organized as follows. Section 2 outlines the background and related studies of this work. Section 3 introduces the target applications that are examined in this study. Section 4 then outlines Kishimoto's models and our new model. The evaluation results are summarized in Section 5, and some discussions are added in Section 6. Section 7 emphasizes the importance of predictability issues, listing the checkpoints to build a well-behaved cluster with practical examples.

2. Background and Related Studies

There have been many attempts to rewrite existing parallel applications for heterogeneous clusters. Typically, the applications for heterogeneous clusters are designed to distribute one process to each PE, where each process handles heterogeneous computational workload according to PE performance. This strategy is called HoHe (*Homogeneous distribution of processes of parallel program over processors with Heterogeneous distribution of data over processes*) in Kalinov's terminology [4].

Some examples of HoHe strategy are found in the following studies. Kalinov and Lastovetsky [5] presented a "heterogeneous block cyclic distribution" for the Cholesky factorization of square dense matrices. Beaumont et al. [6] reported a "2D heterogeneous grid allocation" for the heterogeneous cluster ScaLAPACK [7]. Legrand et al. [8] investigated the mapping of various iterative algorithms onto heterogeneous clusters. Ohtaki et al. [9] proposed a parallel Strassen's matrix multiplication algorithm for heterogeneous clusters.

A serious problem of HoHe strategy is that it is very costly to redesign the existing application. The derived application tends to be more complex than the original, which implies that much effort would be required to establish high performance and reliability. Moreover, such effort must be repeated for each application. Though the HoHe strategy might be suited to derive the maximum performance possible from a heterogeneous cluster, it is sometimes very difficult to optimize the total execution time considering communication time, since the optimal distribution of computational workload does not immediately mean that the total execution time is minimal.

Meanwhile, a multiprocessing approach basically requires no modification of the application program; just by modifying the configuration file of cluster/communication middleware, we can derive a reasonable speedup. This scheme might be applicable to commercial applications, which are usually distributed without source codes. The multiprocessing approach does not aim to extract the maximum performance from a heterogeneous cluster, but seeks rather to provide an easy and simple way to accelerate a wide range of conventional parallel applications in heterogeneous clusters.

The multiprocessing approach is called HeHo strategy in Kalinov's terminology [4], which stands for *Heterogeneous distribution of processes of parallel program over processors with Homogeneous distribution of data over processes*. Besides Kishimoto and Ichikawa's work [1][2], there are some recent studies on this strategy. Kalinov [4], for example, presented an algorithm that computes the optimal number of processes and their distribution over processors minimizing execution time. Though Kalinov's algorithm is based on idealized time functions, an actual application (3D modeling of

supernova explosion) was examined. Cuenca et al. [10] discussed heuristics for load-balancing based on heterogeneous allocation of processes, each of which deals with the same amount of data. Though their scheme is general, their experimental results involve a single application (linear algebra routines). Despite these preceding studies, the advantages and limitations of HeHo strategy are not yet fully explored. This study throws light on about some of these issues by exhibiting the quantitative evaluation results with four popular benchmark programs.

Many studies have focused on the execution time modeling of parallel computation. Culler et al. [11] presented the LogP model, which characterizes a parallel machine by the number of processors (P), the communication bandwidth (g), the communication delay (L), and the communication overhead (o). Alexandrov et al. [12] presented the LogGP model, which extends the LogP model by incorporating long messages. LoPC [13] and LoGPC [14] by Frank et al. further extended the LogP and LogGP models with a model of network contention delay. All these models are constructed for conventional parallel computers, where all processing elements are equivalent. The present investigation, on the other hand, deals with heterogeneous clusters, which consist of different processing elements.

The models for heterogeneous clusters are still not fully explored. Lastovetsky et al. [15] recently presented a communication model for heterogeneous clusters, which can handle fundamental communication patterns. However, the behavior of an actual application is complicated, and its execution time is largely affected by various overheads such as blocking and synchronization. Process scheduling makes things even more complicated, particularly in the HeHo strategy in which multiple processes interact with each other. Modeling the execution time of such an environment is an unsolved problem, which seems very difficult.

In this study, we thus start with the simple models shown in Section 4, and concentrate on baseline evaluations from an empirical and practical standpoint. Simpler models are effective, if they can provide estimations good enough for practical applications. Actually, as shown in Section 5, even our simple models can estimate sub-optimal configurations to a certain extent. Though further improvements of models are desirable, they are beyond the scope of the current study.

3. Benchmark Programs

Although many benchmark programs are written for a fixed and small size computation, such benchmarks do not reflect the practical performance of the system. Benchmark programs must be scalable, if they are going to be used for performance prediction [16]. In this study, benchmark programs that can handle arbitrary problem sizes were selected for experiments.

The Himeno benchmark [17] was originally developed to evaluate the performance of incompressible Navier-Stokes solver in fluid analysis code. It measures the performance of a kernel, which solves a pressure Poisson equation with Jacobi iteration. Recently, the Himeno benchmark has been widely used as an HPC benchmark for supercomputers and clusters.

The FEM benchmark [18][19] was originally developed to measure the performance of 3D linear elastic finite-element application for various types of architectures, e.g., Earth Simulator [20][21], IBM SP3, Hitachi SR8000, and PC clusters. The benchmark kernel is based on a parallel preconditioned Conjugate Gradient (CG) iterative solver with incomplete Cholesky (IC) factorization. Though the FEM benchmark has many features for heterogeneous architectures (e.g., vector parallel supercomputers), we adopt the simplest configuration for general PC clusters in this study.

HPL (High Performance Linpack) [3] is a portable implementation of the Linpack benchmark for distributed-memory computers. HPL solves a random dense linear system of equations in double precision floating-point arithmetic. HPL is widely acknowledged as the performance measure of the TOP500 project [22], which periodically releases the ranking of supercomputers. HPL is also included in the HPC Challenge benchmark suite [23]. HPL is included here to replicate Kishimoto's results in a new environment in contrast to other applications.

Fast Fourier Transform (FFT) is another important application in parallel computing. FFTE [24][25] is a package to compute Discrete Fourier Transforms of 1-, 2-, and 3-dimensional sequences of length $2^p \times 3^q \times 5^r$. The 1-dimensional version of FFTE is included in the HPC Challenge benchmark suite [23]. The FFTE benchmark measures the floating-point arithmetic rate of double precision complex one-dimensional DFT.

With these benchmarks, we can generate binary codes for various sizes by changing the definitions of sizes in header files and then recompiling their source codes. The execution time can be measured without modifying the code, because these benchmark programs are self-timed. In other cases, a simple wrapper program would be required to measure the elapsed time.

4. Methodology

4.1. Kishimoto's models

This section outlines Kishimoto's models, which are applied in the following experiments. More details may be found in previous papers [1][2].

Let N be the size of the problem. A sub-cluster G_i is a group of PEs comprised of equivalent PEs in a heterogeneous cluster, while Γ represents the number of sub-clusters that compose the whole heterogeneous cluster. We construct models from the measurement results of each G_i; i.e., a *homogeneous* sub-cluster is used to construct a model, as in the previous studies [1][2].

Let P_i be the number of PEs actually used for the job in G_i ($0 \leq P_i \leq |G_i|$). M_i is the number of processes on each PE in G_i, if $P_i \neq 0$; M_i is zero, if $P_i = 0$. A **configuration** of a heterogeneous cluster is defined by the set of P_i and M_i for all sub-clusters: i.e., $(P_1, M_1, ..., P_\Gamma, M_\Gamma)$. P represents the total number of processes in the cluster. When a configuration is given, P is calculated by $P = \sum_i P_i M_i$.

Our goal is to build the execution time estimation model for each possible configuration. In other words, it is to build models that estimate the execution time T_i of G_i from parameters N, P, and M_i. The total execution time T is estimated by $T = \max_i T_i$.

Each T_i can be estimated as a function of P and N. In case of HPL, the estimation function of T_i is given by the following equation [1][2]:

$$T_i = \frac{1}{P} \cdot O(N^3) + P \cdot O(N^2) + O(N^2) \tag{1}$$

T_i is thus represented by a cubic function of N for specific P and M_i:

$$T_i(N)|_{P,M_i} \quad = \quad k_0 N^3 + k_1 N^2 + k_2 N + k_3 \tag{2}$$

for each possible configuration $C = (P_1, M_1, ..., P_\Gamma, M_\Gamma)$
  Calculate $P = \Sigma_i P_i M_i$
  for each sub-cluster G_i
    Calculate T_i for given N, P, and M_i
  Calculate $T = \max_i T_i$
Choose the configuration C_{opt} that minimize T
return C_{opt} as the estimated optimal configuration

Figure 1. A fundamental procedure to find the estimated optimal configuration for a given N.

This kind of model is designated as *N-T model* in the following discussions. The constant factors $k_0, ..., k_3$ are determined from the measurement results by the least squares method.[†]

Each N-T model is constructed specific to a configuration (P and M_i). Though N-T models can interpolate or extrapolate T from the size N, they are incapable of estimating T for other configurations. Since it is not practical to manage many N-T models for every combination of parameters, we integrate N-T models of the same M_i into a new model. Since this model includes P as a variable, it is called a *P-T model*. For example, P-T models of HPL are represented by the following equation, considering Eq. (1).

$$T_i(N,P)|_{M_i} \quad = \quad k_4 P \cdot T_i(N)|_{P,M_i} + \frac{k_5}{P} \cdot T_i(N)|_{P,M_i} + k_6 \tag{3}$$

Here, the constant factors $k_4, ..., k_6$ have to be extracted from the corresponding N-T models by the least squares method. With P-T models, we can interpolate or extrapolate T from P, thus substantially reducing the model construction time.

P-T models are adopted only for the case $P > M_i$, which also follows [1][2]. The case $P = \exists M_i$ is special in that the application is executed with a single processor, where no communication over a network is involved. Therefore, N-T model is adopted when $P = \exists M_i$ holds.

Though the above discussions were made for HPL, it should be noted that the estimation functions of N-T and P-T models are dependent on the target application. The estimation functions for other benchmarks will be found in Table V of Section 5.

Once we have constructed the models for every possible configuration, we can estimate the optimal configuration for a given N by solving a combinatorial optimization problem that minimizes the estimated execution time. Figure 1 outlines the fundamental procedure to find the estimated optimal configuration. Although Figure 1 implements an exhaustive search, any pruning techniques for combinatorial optimization can be applied to reduce the search time. Such techniques as branch-and-bound or various heuristics would be particularly effective, when the number of configurations is very large.

[†]In this study, we used GSL (GNU Scientific Library) [26] to extract parameters.

Table I. Details of four benchmark programs.

Benchmark	Version	Compile options
Himeno	HimenoBMTxp C + MPI (static allocate version)	-O2
FEM	hpcmw-solver-test 1.00	-O
HPL	1.0a	-O3 -funroll-loops -fomit-frame-pointer
FFTE	4.0	-O3 -fomit-frame-pointer

This is no more than an estimation based on models; therefore, the estimated optimal configuration is not necessarily an actual optimal configuration. The quality of the estimated optimal configuration may be evaluated by the estimation error ε, which is defined by $\varepsilon = (\hat{\tau} - \hat{T})/\hat{T}$. Here, $\hat{\tau}$ is the actual execution time of the estimated optimal configuration, and $\hat{T}$ is the actual execution time of the actual optimal configuration. Since an estimation error ε is dependent on a specific problem size, we examine the average estimation error $\bar{\varepsilon}$ in the following discussion, where $\bar{\varepsilon}$ is the average value of ε for various sizes (N).

4.2. NP-T model

Directly inducing from Eq. (1), the estimation equation of T_i for parameters N and P would be as follows:

$$T_i(N,P)|_{Mi} = \frac{1}{P}(k_0N^3 + k_1N^2 + k_2N + k_3) + P(k_4N^2 + k_5N + k_6) + k_7N^2 + k_8N + k_9 \quad (4)$$

This equation includes 10 coefficients ($k_0, ..., k_9$), which can be determined from 10 or more measurements using the least squares method. This new model is designated as *NP-T model* in this study. NP-T models are adopted only for the case $P > M_i$ as P-T models.

Though the usage of the NP-T model is the same as for the P-T model, the two models are not equivalent. Kishimoto's P-T model is an empirical approximation to integrate several N-T models into one P-T model, while the NP-T model is naturally induced from Eq. (1).

NP-T model, as well as P-T model, can be constructed by measuring various configurations of G_i. Though a P-T model contains only three coefficients, typically $\nu\,(|G_i| - 1)$ measurements are required to construct a P-T model, where ν stands for the number of sizes (N) to be measured for model construction [1][2]. On the other hand, an NP-T model can be constructed by measuring 10 or more configurations, which is smaller than $\nu(|G_i| - 1)$ in usual situations.

Since an NP-T model includes more coefficients than a P-T model, an NP-T model is intrinsically more flexible than a P-T model. Thus, an NP-T model could be more precise than a P-T model, if it is extracted from a sufficient number of measurements. On the other hand, an NP-T model might become inaccurate, if the extracted parameters are not exact enough. Such pros and cons must be examined quantitatively, based on the measurement results of actual applications on actual systems.

Table II. Problem sizes N for model construction and model evaluation.

Benchmark	Model construction	Model evaluation
Himeno	$32 \leq N \leq 192$ (9 points)	$32 \leq N \leq 256$ (10 points)
FEM	$70 \leq N \leq 504$ (7 points)	$70 \leq N \leq 660$ (20 points)
HPL	$400 \leq N \leq 6400$ (9 points)	$1600 \leq N \leq 9600$ (7 points)
FFTE	$2^{12} \leq N \leq 2^{20}$ (9 points)	$2^{16} \leq N \leq 2^{23}$ (8 points)

Table III. Times [second] for measurements, model construction, and estimation of optimal configuration for four benchmark programs using N-T, P-T, and NP-T models.

	N-T model			P-T model			NP-T model		
	Meas.	Const.	Estim.	Meas.	Const.	Estim.	Meas.	Const.	Estim.
Himeno	1.19×10^{2}	1.24×10^{0}	1.40×10^{-1}	9.88×10^{0}	1.12×10^{1}	4.00×10^{-2}	9.88×10^{0}	4.26×10^{0}	9.00×10^{-2}
FEM	2.08×10^{5}	7.58×10^{0}	1.00×10^{-1}	2.22×10^{4}	3.85×10^{0}	2.00×10^{-2}	2.22×10^{4}	3.44×10^{0}	3.00×10^{-2}
HPL	3.34×10^{4}	4.95×10^{0}	3.20×10^{-1}	2.38×10^{3}	1.33×10^{0}	5.00×10^{-2}	2.38×10^{3}	3.43×10^{0}	2.00×10^{-2}
FFTE	1.83×10^{2}	3.86×10^{0}	1.70×10^{-1}	2.21×10^{1}	1.73×10^{1}	2.03×10^{0}	2.21×10^{1}	2.15×10^{1}	2.30×10^{-2}

Table IV. Specifications of our heterogeneous cluster.

	Subcluster G_1	Subcluster G_2	Subcluster G_3
Processor	Pentium 4 3.6 GHz	Xeon 2.8 GHz	Celeron M 1.5 GHz
$\lvert G_i \rvert$	8	16	8
OS	FedoraCore 4	Redhat Linux 9	FedoraCore 3
Network	1000BASE-TX		

5. Evaluation

In this section, the estimation models are constructed and evaluated for the four benchmarks stated in Section 3. Their specifications are summarized in Table I, and the problem sizes N for model construction and evaluation are summarized in Table II for each benchmark. Measurement time, model construction time, and estimation time of optimal configurations are listed in Table III.

Table IV lists the specifications of the simple heterogeneous cluster used for the following evaluations. For compilation, GNU GCC 3.2.2 and Intel Fortran Compiler 8.1 were adopted with MPICH 1.2.6 as the MPI library.[‡] In this cluster, all processing elements are connected by a single wire-speed switch to exclude the effects of network topology and contention. Although it is important

[‡]MPICH 1.2.6 was compiled with the buffer size set to 8 KB, which is detailed in Section 7.4.

Table V. N-T, P-T, and NP-T models for four benchmark programs.

Benchmark	Model	Estimation equation		
Himeno	N-T	$T_i(N)\|_{P,Mi}$	$=$	$k_0N^3 + k_1N^2 + k_2N + k_3$
	P-T	$T_i(N,P)\|_{Mi}$	$=$	$\frac{k_0}{P} \cdot T_i(N)\|_{P,Mi} + k_1 + k_2 \log P$
	NP-T	$T_i(N,P)\|_{Mi}$	$=$	$\frac{1}{P}(k_0N^3 + k_1N^2 + k_2N + k_3) + k_4N^2 + k_5N + k_6 + k_7 \log P$
FEM	N-T	$T_i(N)\|_{P,Mi}$	$=$	$k_0N^3 + k_1N^2 + k_2N + k_3$
	P-T	$T_i(N,P)\|_{Mi}$	$=$	$\frac{k_0}{P} \cdot T_i(N)\|_{P,Mi} + k_1 + k_2 \log P$
	NP-T	$T_i(N,P)\|_{Mi}$	$=$	$\frac{1}{P}(k_0N^3 + k_1N^2 + k_2N + k_3) + k_4N^2 + k_5N + k_6 + k_7 \log P$
HPL	N-T	$T_i(N)\|_{P,Mi}$	$=$	$k_0N^3 + k_1N^2 + k_2N + k_3$
	P-T	$T_i(N,P)\|_{Mi}$	$=$	$k_4P \cdot T_i(N)\|_{P,Mi} + \frac{k_5}{P} \cdot T_i(N)\|_{P,Mi} + k_6$
	NP-T	$T_i(N,P)\|_{Mi}$	$=$	$\frac{1}{P}(k_0N^3 + k_1N^2 + k_2N + k_3) + P(k_4N^2 + k_5N + k_6) + k_7N^2 + k_8N + k_9$
FFTE	N-T	$T_i(N)\|_{P,Mi}$	$=$	$k_0N \log N + k_1N + k_2N^{\frac{1}{3}} + k_3$
	P-T	$T_i(N,P)\|_{Mi}$	$=$	$\frac{k_0}{P} \cdot T_i(N)\|_{P,Mi} + k_1P \cdot T_i(N)\|_{P,Mi} + k_2$
	NP-T	$T_i(N,P)\|_{Mi}$	$=$	$\frac{1}{P}(k_0N \log N + k_1N + k_2) + k_3P + k_4N + k_5N^{\frac{1}{3}} + k_6$

to examine the clusters of heterogeneous network configurations, they will be left for the following studies.

5.1. Himeno benchmark

Himeno benchmark [17] deals with a domain of $N \times N \times N$. Its execution time mainly consists of Jacobi iterations of $\frac{1}{P} \cdot O(N^3)$, data exchanges of $O(N^2)$, and collective communications (AllReduce) of $\log P \cdot O(1)$. Thus, the execution time of the Himeno benchmark is estimated by the following equation:

$$T(N,P) = \frac{1}{P} \cdot O(N^3) + O(N^2) + \log P \cdot O(1) \tag{5}$$

N-T, P-T, and NP-T models are derived from the above estimation equation, and are summarized in Table V.

Here, it should be noted that the communication time depends on both the algorithm and implementation. For example, AllReduce was estimated to be $\log P \cdot O(1)$ in Eq. (5), assuming that a balanced-tree algorithm is adopted and the network does not form a bottleneck. However, it might become $P \cdot O(1)$, if the communication is sequentialized by the algorithm, network contention, or bandwidth limitation. Though this issue does not occur in our case (with a single-stage wire-speed switch), all such issues have to be considered in modeling.

Measurements were conducted for 9 sizes ($32 \leq N \leq 192$) of Himeno benchmark with the 404 possible configurations listed in Table VI. The models were then constructed from these measurements, and were applied to estimate the execution times of every possible configuration. The estimated optimal execution times for $N = 32, ..., 256$ are plotted in Figure 2, where **Optimal** designates the execution time of the actual optimal configuration of each size, which corresponds to $\hat{T}$ defined in Section 4.1. **N-**

Table VI. Configurations examined for Himeno, FEM, and HPL.

G_1	G_2	G_3	Total
$0 \leq P_1 \leq 4$	$0 \leq P_2 \leq 4$	$0 \leq P_3 \leq 4$	404
$0 \leq M_1 \leq 2$	$0 \leq M_2 \leq 2$	$0 \leq M_3 \leq 1$	

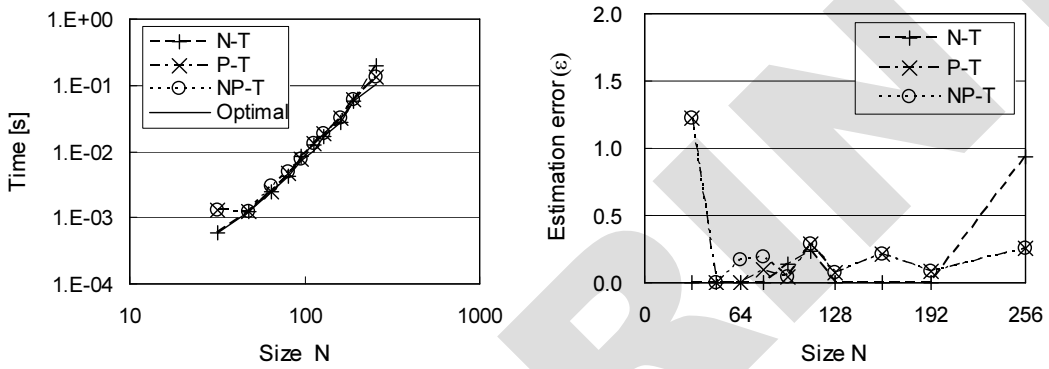


Figure 2. Evaluation results of Himeno benchmark. Left graph displays the execution time of actual optimal configuration (Optimal) and those of optimal configurations estimated by N-T, P-T, and NP-T models. Right graph displays the corresponding estimation error (ε).

T, P-T and **NP-T** in Fig. 2 designate the actual execution times of the estimated optimal configurations derived with N-T, P-T, and NP-T models, which correspond to $\hat{\tau}$ in Sect. 4.1.

As readily seen from Fig. 2, sub-optimal configurations were actually estimated using the N-T, P-T and NP-T models. The respective average errors ($\bar{\varepsilon}$) were 13%, 23%, and 25% for the N-T, P-T, and NP-T models. Although $\bar{\varepsilon}$ of P-T and NP-T are inferior to that of N-T, the difference is not large. This difference is mainly caused by the error at $N = 32$, which is 0% for N-T, 123% for P-T, and 122% for NP-T. This error does not matter practically, because the execution time is accordingly small for small sizes. If the case $N = 32$ is excluded, respective average errors would be 15%, 12%, and 15% for the N-T, P-T, and NP-T models. These values seem good enough for practical applications.

5.2. FEM benchmark

The FEM benchmark [18][19] originally deals with a three-dimensional domain of $N \times N \times N$, where N must be a multiple of P. A two-dimensional domain, $N \times N \times 1$, is measured in this study for the restrictions of execution time and memory space.

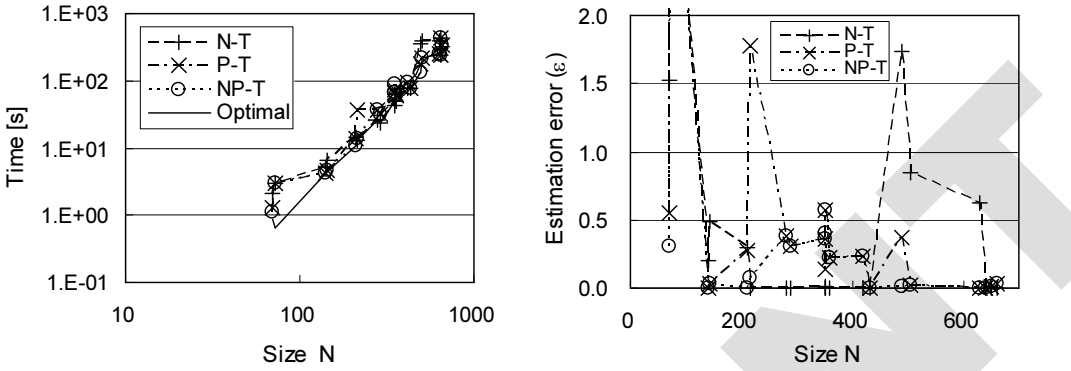


Figure 3. Evaluation results of FEM benchmark. Left graph displays the execution time of actual optimal configuration (Optimal) and those of optimal configurations estimated by N-T, P-T, and NP-T models. Right graph displays the corresponding estimation error (ε).

The execution time of the FEM benchmark consists of matrix-vector multiplications of $\frac{1}{P} \cdot O(N^3)$, vector inner-products of $\frac{1}{P} \cdot O(N^2)$, vector addition/subtractions of $\frac{1}{P} \cdot O(N^2)$, data exchange of $O(N^2)$, and collective communications (AllReduce) of $\log P \cdot O(1)$. Thus, the execution time of FEM benchmark is estimated by the following equation:

$$T(N, P) = \frac{1}{P} \cdot O(N^3) + O(N^2) + \log P \cdot O(1) \tag{6}$$

The models derived for the FEM benchmark are summarized in Table V.

As stated above, N must be a multiple of P in the FEM benchmark. Though this restriction is very natural from a programmer's standpoint, it is crucial to both model construction and optimization. In constructing a model for a specific set of P and M_i, we cannot use arbitrary sizes but must choose the multiples of the specified P. This means that we generally have to use a different set of problem sizes for the models of different P. In estimating the optimal configuration for a given size N, we cannot examine arbitrary configurations but can only use the models whose P is a factor of the specified N. This means that the search space of optimization is greatly restricted. All these factors may affect the accuracy of optimization.

Figure 3 plots the evaluation results of the FEM benchmark. Measured problem sizes and configurations are summarized in Tables II and VI. As readily seen, sub-optimal configurations were successfully estimated using the N-T, P-T and NP-T models. The respective average errors ($\bar{\varepsilon}$) were 47%, 34%, and 33% for the N-T, P-T, and NP-T models. Though these values are not quite satisfactory, most of the errors are caused by the estimations for two small sizes ($N = 70$ and 72) as readily seen in Fig. 3. Excluding these two sizes, the respective average errors become 23%, 18%, and 15% for the N-T, P-T, and NP-T models, which is practical and on a par with the Himeno benchmark. Despite the restriction between N and P, our models were shown to be practical for the FEM benchmark.

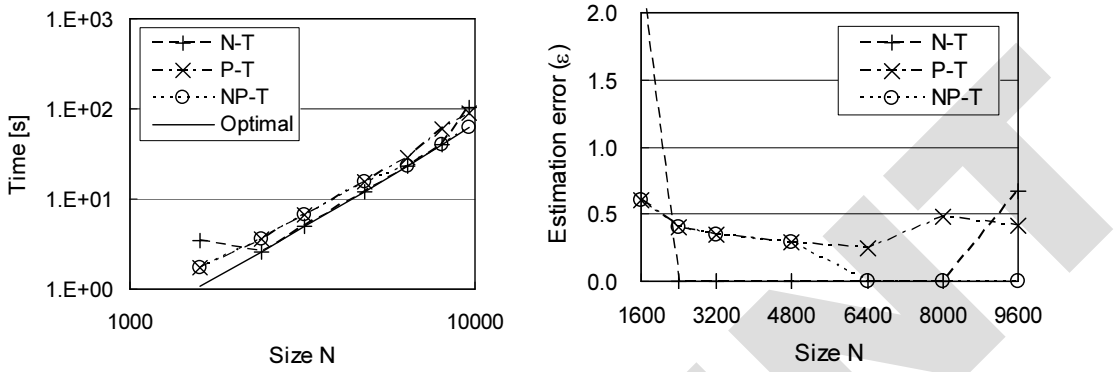


Figure 4. Evaluation results of HPL benchmark. Left graph displays the execution time of actual optimal configuration (Optimal) and those of optimal configurations estimated by N-T, P-T, and NP-T models. Right graph displays the corresponding estimation error (ε).

Although the errors in small sizes do not matter practically, it is desirable to construct more precise models for arbitrary sizes. Section 5.7 will describe an application-specific technique, which can be used to construct NP-T models whose $\bar{\varepsilon}$ is 19% for all sizes.

5.3. HPL

HPL solves a dense linear system of N variables with the following algorithms [3]:

- Two-dimensional block-cyclic data distribution,
- Right-looking variant of the LU factorization with row partial pivoting featuring multiple look-ahead depths,
- Recursive panel factorization with pivot search and column broadcast combined,
- Various virtual panel broadcast topologies,
- Bandwidth reducing swap-broadcast algorithm,
- Backward substitution with look-ahead of depth 1.

The estimation equation for HPL is shown as Eq. (1) in Section 4, and the derived models are summarized in Table V. In this study, ATLAS-3.6.0 [27] was adopted as the BLAS library. The process grid was fixed to one-dimensional block cyclic as in the previous study [2].

Figure 4 displays the evaluation results of the HPL benchmark. Measured problem sizes and configurations are summarized in Tables II and VI. As seen from Fig. 4, sub-optimal configurations were estimated with N-T, P-T and NP-T models, where the respective average errors ($\bar{\varepsilon}$) were 42%, 40%, and 23% for N-T, P-T, and NP-T models. It is readily seen that errors are larger in smaller sizes in HPL, as well as in the Himeno and FEM benchmarks. Excluding the small sizes with execution times

Table VII. Configurations examined for FFTE.

G_1	G_2	G_3	Total
$0 \leq P_1 \leq 8$ $0 \leq M_1 \leq 2$	$0 \leq P_2 \leq 8$ $0 \leq M_2 \leq 2$	$0 \leq P_3 \leq 8$ $0 \leq M_3 \leq 1$	285

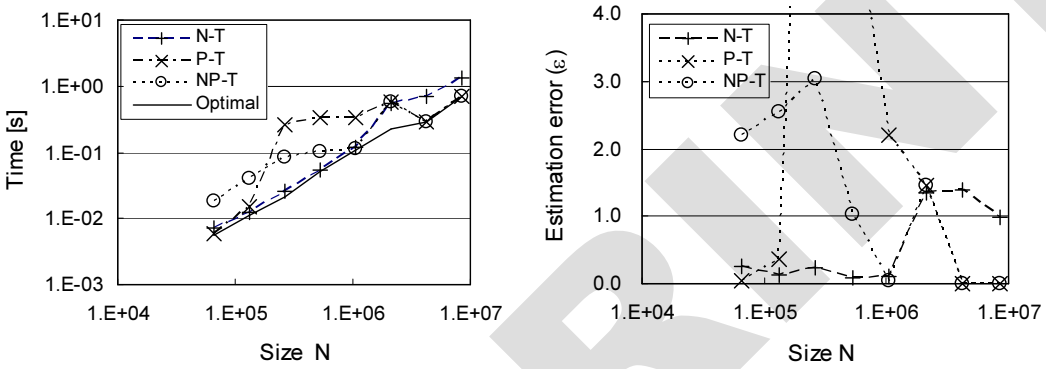


Figure 5. Evaluation results on FFTE benchmark. Left graph displays the execution time of actual optimal configuration (Optimal) and those of optimal configurations estimated by N-T, P-T, and NP-T models. Right graph displays the corresponding estimation error (ε).

of less than 10 seconds, the average errors are reduced to as small as 17%, 36%, and 7% for N-T, P-T, and NP-T models ($4800 \leq N \leq 9600$). In either case, the NP-T model looks reasonably accurate and practical.

Although our evaluation environment is different from that of the previous studies [1][2], our results basically confirmed and reproduced their results. Furthermore, our NP-T model was shown to be superior to N-T and P-T models presented in previous studies.

5.4. FFTE benchmark

FFTE benchmark [24][25] measures the execution time of one-dimensional DFT of size N, where N must be a multiple of P^2. In this study, we examine the cases where P is a power of 2, because FFTE is written on this implicit assumption.

The execution time of FFTE consists of a fast Fourier transform of $\frac{1}{P} \cdot O(N \log N)$, factoring of $O(N)$, table preparations of $O(N^{\frac{1}{3}})$, and all-to-all communications of $\frac{1}{P} \cdot O(N) + P \cdot O(1)$. Thus, the

Table VIII. Problem sizes N for P-T6, NP-T6, ..., P-T9, and NP-T9 models.

Model	Model construction	Model evaluation
P-T9, NP-T9	$32 \leq N \leq 192$ (9 points)	$32 \leq N \leq 256$ (10 points)
P-T8, NP-T8	$32 \leq N \leq 160$ (8 points)	
P-T7, NP-T7	$32 \leq N \leq 128$ (7 points)	
P-T6, NP-T6	$32 \leq N \leq 112$ (6 points)	

execution time of FFTE is estimated by the following equation:

$$T(N,P) = \frac{1}{P} \cdot O(N \log N) + O(N) + O(N^{\frac{1}{3}}) + \frac{1}{P} \cdot O(N) + P \cdot O(1) \tag{7}$$

The N-T, P-T, and NP-T models are derived from the above estimation equation, and are summarized in Table V.

Figure 5 plots the evaluation results of the FFTE benchmark. Measured problem sizes are listed in Table II, while possible configurations (P_i and M_i) are summarized in Table VII. Since the execution time of FFTE benchmark is shorter than for the other three benchmarks, a twice larger cluster ($|G_i| = 8$) was examined for FFTE than for the other three applications ($|G_i| = 4$). This makes the estimation of optimal configuration much harder. Another difficulty of FFTE is that the all-to-all communication is a dominant factor in the execution time. Considering these two factors, the result shown in Fig. 5 can be interpreted as a worst-case evaluation of our scheme.

As readily seen from Fig. 5, the estimation errors for FFTE are larger than the other three benchmarks. The average errors were 56%, 267%, and 129% for the N-T, P-T, and NP-T models, respectively. As in the other three benchmarks, the errors are larger for smaller sizes. Excluding the sizes whose optimal execution times are less than 0.1 second, the average errors are reduced to 96%, 92%, and 38% for N-T, P-T, and NP-T models ($2^{20} \leq N \leq 2^{23}$). For large sizes, the NP-T models are superior to the N-T and P-T models and looks reasonably accurate.

The residual errors of FFTE are still larger than other benchmarks, and are not quite satisfactory. Further improvements are attempted in the following sections.

5.5. Robustness of models

As shown in the above sections, NP-T models have various advantages over P-T models. One of the virtues of an NP-T model is that it can be constructed from fewer measurements than a P-T model. This may lead to another advantage of NP-T models; an NP-T model might be more accurate than a P-T when the number of measurements is not large enough. In other words, NP-T models are expected to be more robust against the shortage of measurement results.

To verify this conjecture, the estimation errors of the Himeno benchmark were evaluated with various P-T and NP-T models that were constructed from the measurements of various sizes, which are listed in Table VIII. In the following discussion, P-T6 designates the P-T model that was constructed with 6 sizes of measurements, while NP-T9 designates the NP-T model constructed from 9 sizes.

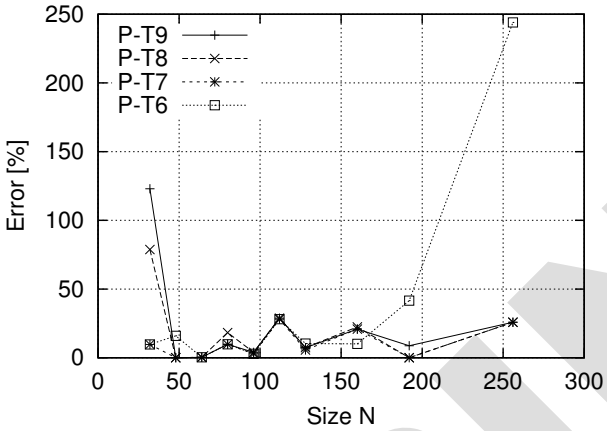


Figure 6. Estimation errors of P-T6, P-T7, P-T8, and P-T9 models in Himeno benchmark.

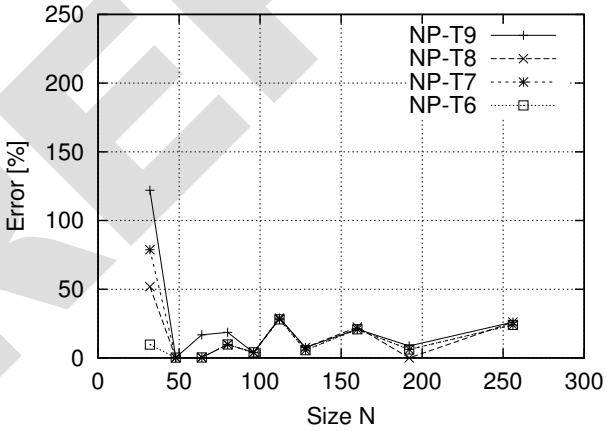


Figure 7. Estimation errors of NP-T6, NP-T7, NP-T8, and NP-T9 models in Himeno benchmark.

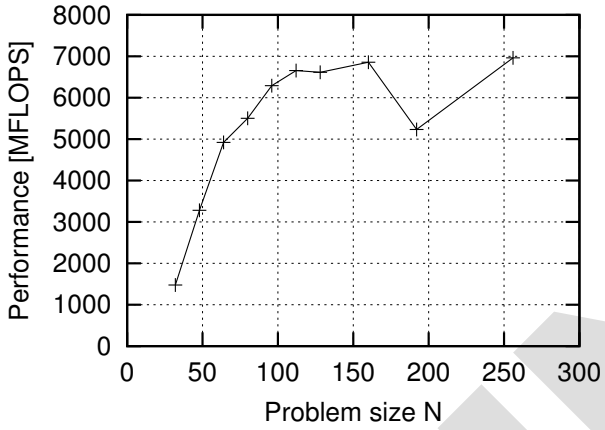


Figure 8. An example of measured performance of Himeno benchmark, where a performance anomaly exists at $N = 192$.

The estimation errors of P-T and NP-T models are summarized in Figures 6 and 7. As readily seen, P-T6 models failed in large sizes and yielded large errors. NP-T6, on the other hand, maintained good accuracy in all sizes. All these results seem to support our conjecture.

In the smallest size ($N = 32$), the errors of P-T6 and NP-T6 are smaller than those of P-T9 and NP-T9. However, this fact does *not* mean that P-T6 and NP-T6 are more accurate than P-T9 and NP-T9. Rather, it should be understood that the weight of the size $N = 32$ is heavier in P-T6 and NP-T6 (one of six) than in P-T9 and NP-T9 (one of nine). Thus, the derived models better fit the data at $N = 32$ in P-T6 and NP-T6 than in P-T9 and NP-T9.

5.6. Performance anomalies

As stated in Section 5.1, the models for the Himeno benchmark were not quite satisfactory, in particular for $N = 32$. Investigating the reasons for such estimation errors, many performance anomalies (glitches) were found in the measurement results, which resulted in imprecise models.

Figure 8 displays an example of a performance glitch. Though the performance is expected to increase monotonically as the size N increases, the performance of $N = 192$ is substantially lower than expected. Figure 9 shows the measured execution times and the derived N-T model of the same configuration as Fig. 8. Model (1) was extracted from all measurement results, while Model (2) was extracted from the measurements except for the glitch at $N = 192$. As readily seen, Model (1) suffers from a large error at $N = 256$, whereas Model (2) only incurs a modest error at $N = 192$. Thus, the accuracy of models is much degraded by performance anomalies in the measurement results.

Such performance anomalies have various causes (e.g., cache thrashing), while they should be avoided by programming techniques or compilation techniques (e.g., padding) [28][29][30]. However,

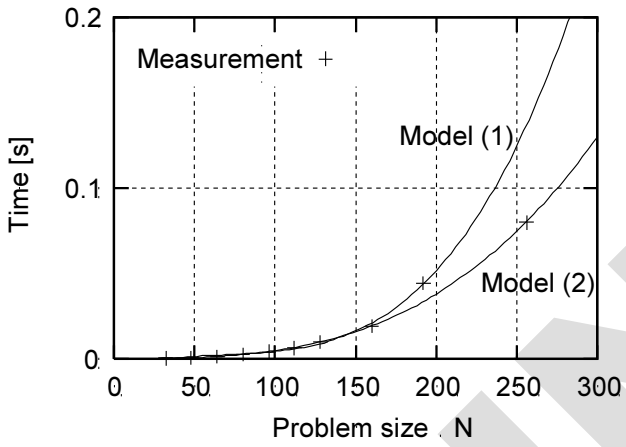


Figure 9. Performance anomalies affect the model parameters extracted. Model (1) was extracted from all measurement results, while Model (2) was extracted from the measurements excluding $N = 192$.

in this study, we accept the fact that such anomalies often exist in measurement results, and attempt to reduce the average errors of models.

As the first step, we here examine a simple heuristic to evade glitches. The datum of size n is regarded as a glitch and excluded from model construction, if the following condition holds: $p(n) \leq k \cdot p(m)$. Here, $p(n)$ represents the performance (e.g., Mflops) of size n. The size m is the point of measurement, which is nearest to and smaller than n. Constant factor k has to be determined empirically.

The average error of the model is expected to be improved by excluding glitches with an appropriate k. On the other hand, the model might become imprecise or it might be impossible to construct the corresponding model, if too many sizes are excluded from model construction. The appropriate value of k is dependent on application and platform, and thus must be determined empirically.

With the Himeno benchmark, the average error of the NP-T models was improved from 25% to 13% by excluding glitches with $k = 0.9$. If the size $N = 32$ is additionally excluded, $\bar{\varepsilon}$ of NP-T is improved from 15% to 10% with this heuristic. Though exclusion of glitches worked very well for the Himeno benchmark, it did not work with the HPL and FEM benchmarks. We observed that the Himeno benchmark involves many glitches, while glitches scarcely appear in the results of the HPL and FEM benchmarks. In case of FFTE, exclusion of glitches yielded no improvement, despite the fact that several performance glitches are observed.

Considering all these facts, we concluded that exclusion of glitches is undoubtedly useful for certain kinds of applications. Though it is not for all applications, it can be used optionally to improve the models. Exclusion of glitches is solely a partial solution, and models should be improved for more precise estimation. Such refinements will be made in the following sections.

5.7. Simplification of estimation equation

As stated in Section 5.2, the average estimation error of FEM benchmark was 33% with the NP-T model. Though the error for $N \geq 140$ is no more than 15%, it is desirable to further reduce the error for all sizes.

In the investigation of the estimation errors, a problem in estimation equation was identified. In our evaluation environment, the collective communication takes only 100 ms, which is negligible in the FEM benchmark. Though it is possible to extract parameters with the term of $\log P$, such a model often incurs a large estimation error caused by the erroneous coefficient in the $\log P$ term. This was the cause of the large errors at small sizes in Fig. 3.

We thus examined a model whose $\log P$ term is omitted. The estimation equation is given by the following equation:

$$T(N,P) \approx \frac{1}{P} \cdot O(N^3) + O(N^2) \tag{8}$$

P-T and NP-T models were constructed and evaluated without $\log P$ term, and the derived average errors were 43% with the P-T model and 19% with the NP-T model. The average error of P-T slightly increased from 34%, whereas the average error of NP-T was successfully improved from 33% to 19%. Though this is an application-specific adjustment, it is worth considering excluding negligible terms to check whether the models become more accurate.

5.8. Robust parameter extraction

Although NP-T models were shown to be more robust than other models, they are still sensitive to the fluctuations and anomalies in measurement results.

Figure 10 displays two NP-T models, both of which are derived from the same set of measurements. The left model is the NP-T model, whose parameters are extracted (or fitted) by using a standard least squares method (LS). It is readily seen that this model is wrecked and estimates a negative execution time for large sizes (e.g., $N = 256$). This problem was caused by the failure of parameter extraction, where the coefficient of N^3 was fit to a negative value. Once this kind of wrecked model is constructed, it will be estimated as the optimal configuration, since reasonable models never return negative values as execution time. Consequently, the quality of estimation is easily degraded to impractical levels.

The simplest way to avoid such failures is to restrict the coefficients to non-negative values in model parameter extraction. Finding the coefficients $k_i \geq 0$ by least squares method is known as *non-negative least squares* (NNLS) method. The formal definition, algorithm, and example implementation of NNLS method are found in the reference [31]. NNLS induces nothing but monotonically increasing models, which assures the estimated execution time is always non-negative. The right graph of Figure 10 displays the NP-T model, whose parameters are extracted by NNLS method.

Here, it should be noted that this condition ($k_i \geq 0$) is sufficient but over-kill to assure that the estimated execution time is non-negative. This strong restriction might have a negative impact on the accuracy of models, because it will reduce the freedom of models significantly. On the other hand, NNLS is expected to be more robust and resistant to the anomalies and fluctuations in measurement; this may lead to more accurate estimation. NNLS naturally leaves some of the coefficients to zero in parameter fitting, which can be interpreted as a generalization of the simplification of estimation

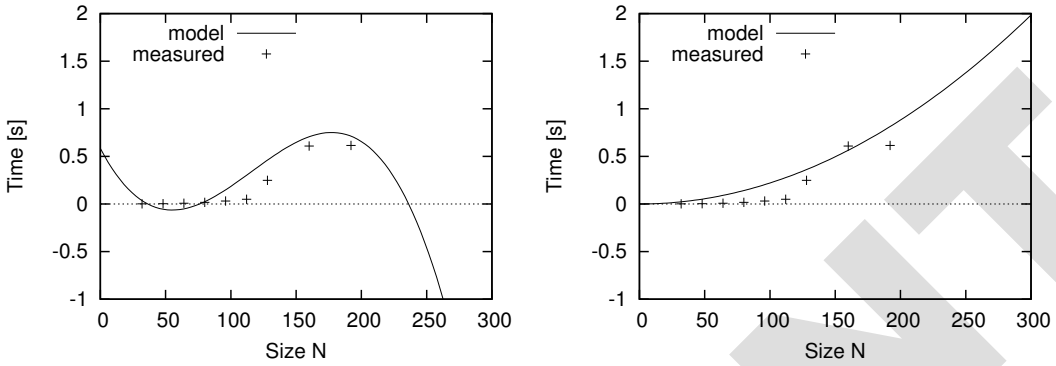


Figure 10. NP-T models for Himeno benchmark, where the model parameters were extracted by two different algorithms for the same set of measurement results. Least squares method was used for parameter extraction in the left graph, while non-negative least squares method was used in the right graph.

equation (Sect. 5.7). This may also improve the accuracy of estimation. The pros and cons of two algorithms, a standard least squares method (LS) and a non-negative least squares method (NNLS), must be evaluated experimentally for model parameter extraction.

The behaviors of execution time are heavily dependent on the software environment, even for the same hardware platform. In the following discussions, Intel C/C++ compiler 9.0 and Intel Fortran compiler 9.0 were used with MPICH 1.2.7-p1. The differences between LS and NNLS are exhibited more definitely with this combination of software, since far more anomalies and fluctuations are observed with them compared to the software used in the previous sections (GCC 3.2.2 and Intel Fortran 8.1 with MPICH 1.2.6).

Figures 11, 12, 13, and 14 summarize the evaluation results of the models, whose parameters are extracted by LS and NNLS. As readily seen, the effect of NNLS is outstanding for all four benchmarks.

In case of the Himeno benchmark, the average error $\bar{\varepsilon}$ of the NP-T model whose parameters are extracted by LS was 487% for the sizes between 32 and 256, while that by NNLS remains as low as 0.2% (Fig. 11). The large errors of LS are due to wrecked models; for 7 sizes out of 11, negative execution times were estimated as optimal, which is definitely far from reality. By the way, with the software of Section 5.1, only small number of wrecked models were derived with LS method. In case of Fig. 2, negative execution time was not estimated as optimal except for one size.

In case of FEM benchmark, the respective average errors $\bar{\varepsilon}$ between 60 and 600 were 41% for LS and 2% for NNLS (Fig. 12). For HPL, the respective average errors $\bar{\varepsilon}$ between 1600 and 9600 were 94% for LS and 1% for NNLS (Fig. 13). In both cases, the improvements are drastic.

Meanwhile, in case of FFTE, the average errors $\bar{\varepsilon}$ between 2^{16} and 2^{23} was 194% for LS, while that of NNLS remained approximately 28% (Fig. 14). The large errors of LS are, yet again, due to wrecked models; for 5 sizes out of 8, negative execution times were estimated as optimal. This problem is fixed by using NNLS for parameter extraction. Although the average error with NNLS is not yet quite

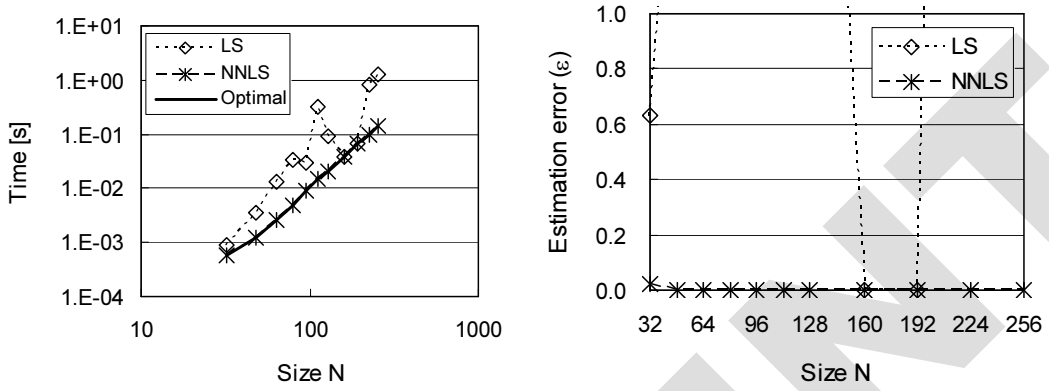


Figure 11. Parameter extraction for NP-T model by least squares method (LS) and non-negative least squares method (NNLS) for Himeno benchmark. Left graph displays the execution time of actual optimal configuration (Optimal) and those of estimated optimal configurations with the parameters extracted by LS method and NNLS method. Right graph displays the corresponding estimation error (ε).

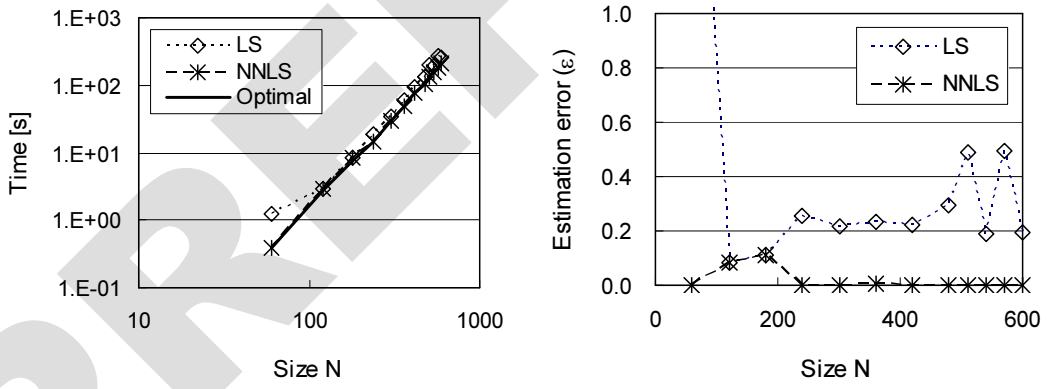


Figure 12. Parameter extraction for NP-T model by least squares method (LS) and non-negative least squares method (NNLS) for FEM benchmark. Left graph displays the execution time of actual optimal configuration (Optimal) and those of estimated optimal configurations with the parameters extracted by LS method and NNLS method. Right graph displays the corresponding estimation error (ε).

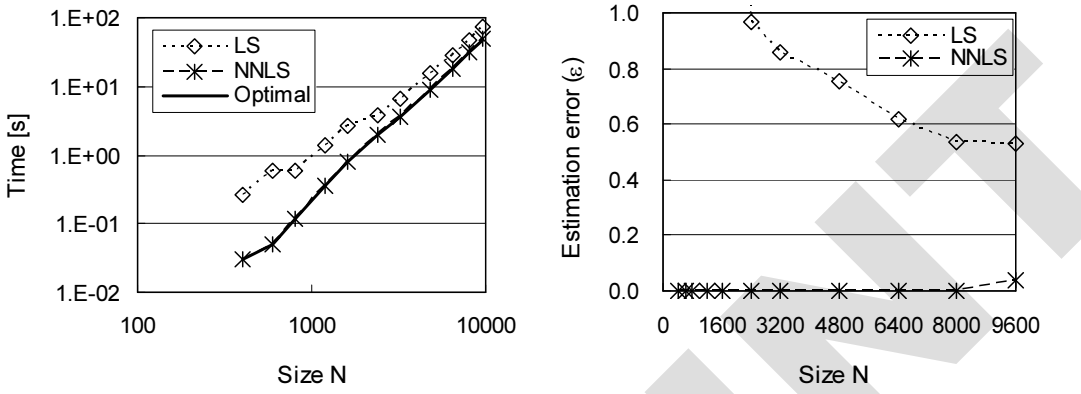


Figure 13. Parameter extraction for NP-T model by least squares method (LS) and non-negative least squares method (NNLS) for HPL benchmark. Left graph displays the execution time of actual optimal configuration (Optimal) and those of estimated optimal configurations with the parameters extracted by LS method and NNLS method. Right graph displays the corresponding estimation error (ε).

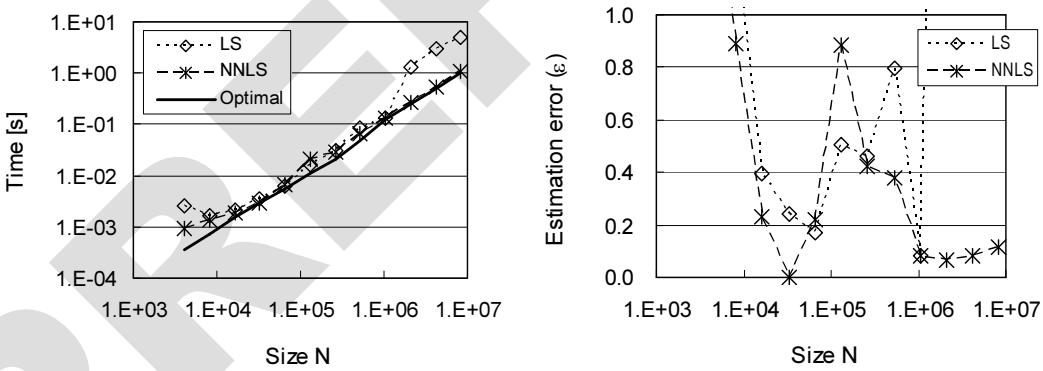


Figure 14. Parameter extraction for NP-T model by least squares method (LS) and non-negative least squares method (NNLS) for FFTE benchmark. Left graph displays the execution time of actual optimal configuration (Optimal) and those of estimated optimal configurations with the parameters extracted by LS method and NNLS method. Right graph displays the corresponding estimation error (ε).

satisfactory, 28% is not far from the practical level. As stated in Section 5.4, the FFTE benchmark spends much time for the all-to-all communication, which is not very precisely modeled in our current model. Given this fact, the average error of 28% seems not too bad. Though more improvement on this aspect is desirable, it is left for future studies.

6. Discussion

6.1. Advantages over static process allocation schemes

In this study, we have presented a method to estimate the optimal configuration from all possible configurations for a given problem size, where the estimation error of our method is reportedly modest in many cases. Meanwhile, there are simpler alternatives for practitioners; they can use a simple static allocation scheme, where the process allocation is fixed regardless of problem sizes. The simplest example of static allocation scheme is to use the whole heterogeneous cluster, invoking one process for each PE, regardless of the problem size.

Our scheme is conceptually better than a simple static allocation, since it selects the estimated best configuration for a given size from all possible configurations. However, this single fact is not enough to convince us that our method is practically superior to static process allocation schemes. The anomalies and fluctuations in measurements might lead to estimation errors, which may erode the performance advantage of our scheme. If the performance advantage is not large enough, our scheme might not be justified given all the efforts required to construct the models for all possible configurations.

Thus, it is necessary to show the quantitative performance advantage of our scheme over simple static allocation schemes. This section is devoted to this topic.

Figure 15 summarizes the comparison results of various process allocation schemes for the HPL benchmark. In Figure 15, "Optimal" stands for the results of the actual optimal configurations, and "NP-T/NNLS" stands for that of the estimated optimal configurations with the NP-T model whose parameters were extracted with the NNLS method. The numbers in parentheses stand for the results of static allocation schemes, each of which corresponds to the configuration of $(P_1, M_1, P_2, M_2, P_3, M_3)$. The left graph plots the execution times, while the right graph plots the estimation error ε of each configuration. In this figure, four trivial static allocation schemes are examined: a single Pentium 4 node (1, 1, 0, 0, 0, 0), the whole Pentium 4 sub-cluster (4, 1, 0, 0, 0, 0), Pentium 4 and Xeon sub-clusters (4, 1, 4, 1, 0, 0), and the whole heterogeneous cluster (4, 1, 4, 1, 4, 1).

HPL shows the clear advantage of our scheme. The optimal configurations of HPL were (1, 1, 0, 0, 0, 0) between the size 400 and 1600, (4, 1, 0, 0, 0, 0) between 2400 and 8000, and (4, 1, 4, 1, 0, 0) for 9600. None of the trivial static allocation schemes gives the optimal execution time for all sizes, whereas our scheme successfully estimates the optimal or sub-optimal configuration for each size. Size $N = 9600$ was the only case in which NP-T/NNLS failed to select the actual optimal configuration (4, 1, 4, 1, 0, 0), where the estimated optimal configuration was (4, 1, 0, 0, 0, 0) and its estimation error was as small as 3.9%.

NP-T/NNLS displays an excellent score in Himeno benchmark as well. Figure 16 summarizes the results of the Himeno benchmark for NP-T/NNLS and three static allocations. The Himeno benchmark is interesting in that the configuration (4, 1, 0, 0, 0, 0) is optimal in most sizes. Using more sub-clusters, the execution time increases because of the increase of communication time. If a single Pentium 4

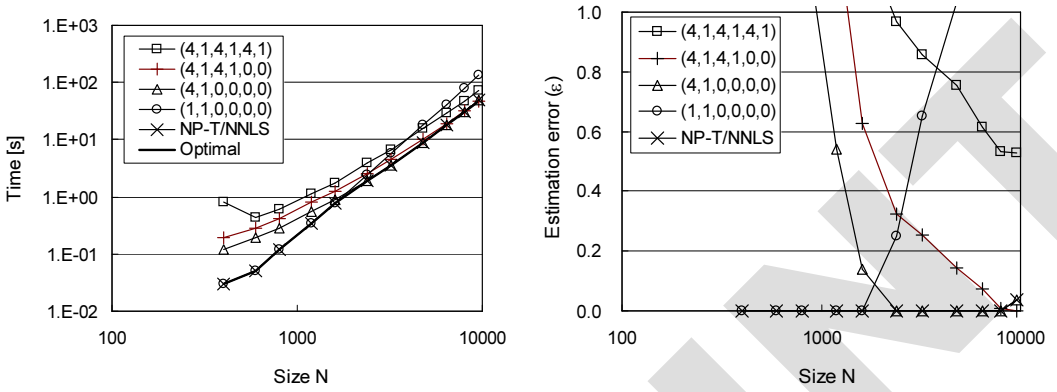


Figure 15. Comparison of various process allocation schemes for HPL benchmark. Left graph displays the execution time of the actual optimal configuration (Optimal) and those of estimated optimal configurations by various allocation schemes. Right graph displays the corresponding estimation error (ε).

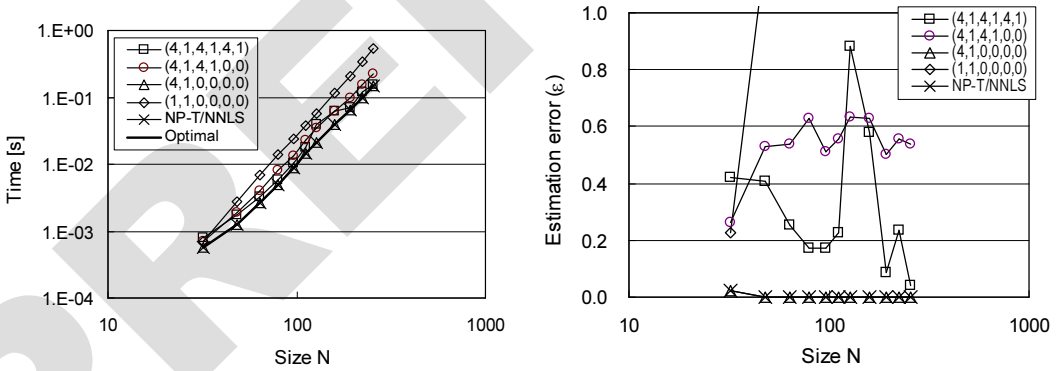


Figure 16. Comparison of various process allocation schemes for Himeno benchmark. Left graph displays the execution time of the actual optimal configuration (Optimal) and those of estimated optimal configurations by various allocation schemes. Right graph displays the corresponding estimation error (ε).

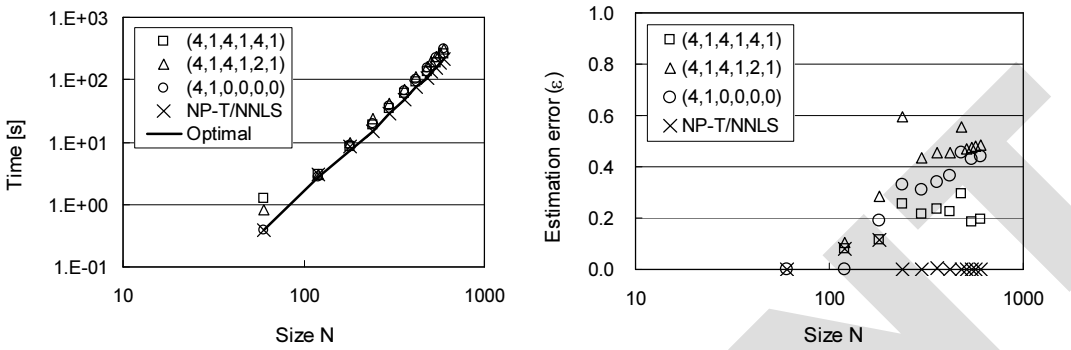


Figure 17. Comparison of various process allocation schemes for FEM benchmark. Left graph displays the execution time of the actual optimal configuration (Optimal) and those of estimated optimal configurations by various allocation schemes. Right graph displays the corresponding estimation error (ε).

node is used, its execution time becomes much larger than the optimal configuration because of the low processing power. NP-T/NNLS estimates the optimal configuration in most cases except for size $N = 32$, where the optimal configuration was (3, 1, 0, 0, 0, 0).

Figure 17 displays the results of the FEM benchmark, for which the size (N) must be a multiple of the total number of processes (P), as stated in Section 5.2. Thus, the lines between labels were omitted in this figure, because the size N is discrete. In Figure 17, we plotted the sizes that are the multiples of 60, to examine three trivial configurations of $P = 4$, 10, and 12 that satisfy the above restriction: Pentium 4 sub-cluster (4, 1, 0, 0, 0, 0), 10 fastest nodes (4, 1, 4, 1, 2, 1), and the whole heterogeneous cluster (4, 1, 4, 1, 4, 1). NP-T/NNLS shows obviously good estimations, whereas three trivial configurations are far from optimal.

The FEM benchmark is remarkable in that multiprocessing is effective to improve the overall performance. Figure 18 displays the results of the FEM benchmark, where three simple multiprocessing configurations for $P = 12$, 15, and 20 are examined. Though the configuration (4, 2, 4, 1, 3, 1) seems relatively good in static allocation, it is still inferior to our NP-T/NNLS. The configuration (4, 2, 4, 1, 3, 1) involves large error for small sizes ($N = 60, 120$), where the Pentium 4 sub-cluster (4, 1, 0, 0, 0, 0) is optimal. The configuration (4, 2, 4, 1, 3, 1) also involves slight errors in $N = 240$ and 480, both of which are the multiples of 80. In these cases, the configuration of $P = 16$, that is (4, 2, 4, 1, 4, 1), was actually optimal. NP-T/NNLS succeeded in estimating this actual optimal configuration for these sizes.

Although the configuration (4, 2, 4, 1, 3, 1) is relatively good for the case of the multiples of 60, the optimal configuration is strongly dependent on the size and no trivial static allocation is expected to be optimal for various sizes. NP-T/NNLS is particularly advantageous in such cases, since it automatically estimates the optimal or sub-optimal configuration for various sizes.

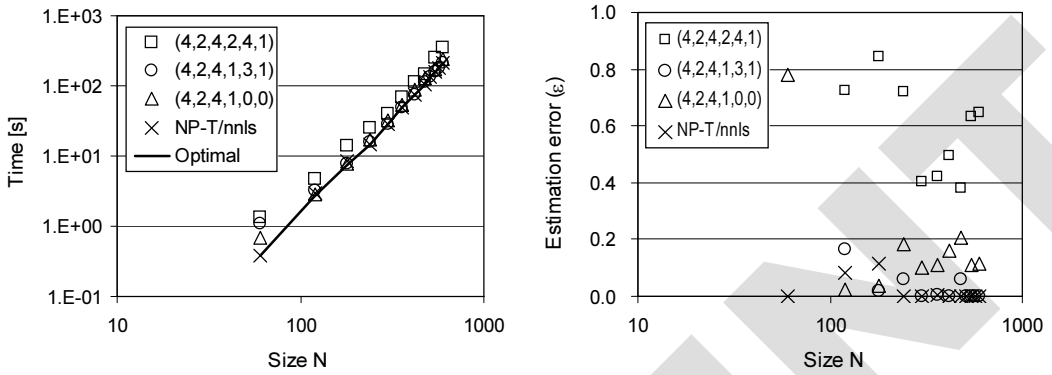


Figure 18. The effect of multiprocessing schemes for FEM benchmark. Left graph displays the execution time of the actual optimal configuration (Optimal) and those of estimated optimal configurations by various allocation schemes. Right graph displays the corresponding estimation error (ε).

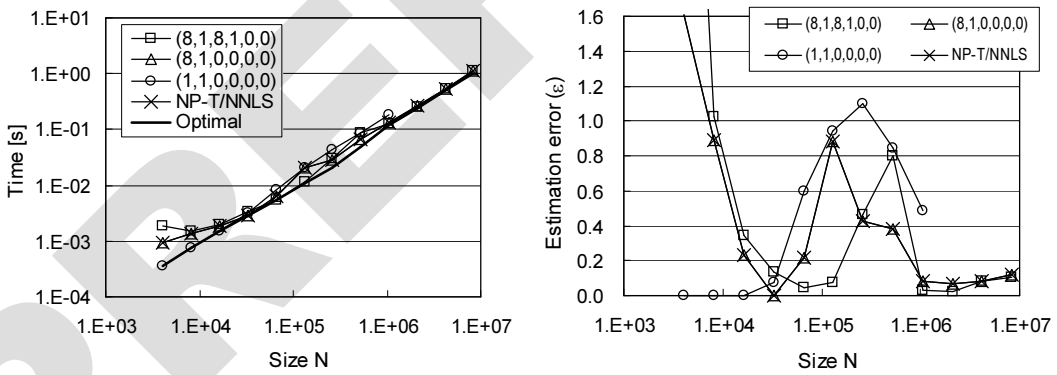


Figure 19. Comparison of various process allocation schemes for FFTE benchmark. Left graph displays the execution time of the actual optimal configuration (Optimal) and those of estimated optimal configurations by various allocation schemes. Right graph displays the corresponding estimation error (ε).

Figure 19 displays the results of the FFTE benchmark. As stated in Section 5.4, both the size (N) and the total number of processes (P) have to be a power of 2 in this benchmark. Here, three trivial configurations ($P = 1, 8, 16$) are summarized in Figure 19; two configurations (2, 1, 0, 0, 0, 0) and (4, 1, 0, 0, 0, 0) were omitted, since their errors (ε) are mostly larger than 1.6.

Although our scheme is not very good at handling FFTE, it rather depends on the tricky behavior of FFTE. For example, the actual optimal configuration for $N = 2^{20}, ..., 2^{23}$ was (8, 1, 0, 0, 8, 1), instead of (8, 1, 8, 1, 0, 0). The respective actual optimal configurations for $N = 2^{16}, ..., 2^{19}$ were (7, 1, 7, 1, 2, 1), (7, 1, 5, 1, 4, 1), (8, 2, 8, 2, 0, 0), and (8, 2, 7, 2, 2, 1), neither of which is trivial. It would be very difficult to find such configurations optimal, without a detailed timing model of communication. We expect that precise modeling of communication might lead to more accurate estimations for FFTE. This point is also left for future studies.

6.2. Estimation of actual execution time

In previous sections, the quality of the estimated optimal configuration was evaluated by the estimation error $\varepsilon = (\hat{\tau} - \hat{T})/\hat{T}$, where $\hat{\tau}$ is the actual execution time of the estimated optimal configuration, and $\hat{T}$ is the actual execution time of the actual optimal configuration. This ε is a measure to evaluate the quality of the estimated optimal configuration.

Meanwhile, it is equally important to evaluate the accuracy of the estimated execution time. Although there are as many models as possible configurations, the accuracy of the execution time of the estimated optimal configuration is of practical importance. Thus, in this section, we examine the error δ of the estimated execution time, which is defined by $\delta = (\tau - \hat{\tau})/\hat{\tau}$. Here, τ is the estimated execution time of the estimated optimal configuration.

Roughly speaking, it is more difficult to make δ small than to make ε small, since accurate execution time is not necessary to estimate the optimal configuration. If the order of execution time is kept accordingly among models, the optimal configuration can be correctly estimated, even if the execution time itself is not accurate. This is obvious from considering an example, where each model estimates the execution time as half of its actual execution time. It is still possible to estimate the optimal configuration correctly with such models, while the estimated optimal execution time (τ) would always be half of the actual optimal execution time ($\hat{\tau}$). Thus, such models yield the respective errors $\varepsilon = 0$ and $\delta = -0.5$. An actual example of this kind of difficulty was shown with the HPL benchmark by Kishimoto and Ichikawa [2].

Figure 20 summarizes the error δ of NP-T/NNLS for four benchmarks. It is readily seen that $-0.2 \leq \delta \leq 0.2$ holds in most cases. This error amount looks acceptable for practical applications. Though there is a tendency for $|\delta|$ to be large in small sizes, it is not practically a disadvantage. Since the execution time is small for small-sized problems, the absolute differences of execution time are kept accordingly small.

7. Building a predictable cluster

This study presented a method to estimate optimal or sub-optimal configurations for heterogeneous clusters. Although it is an old idea to estimate execution time based on measurements, it has not been

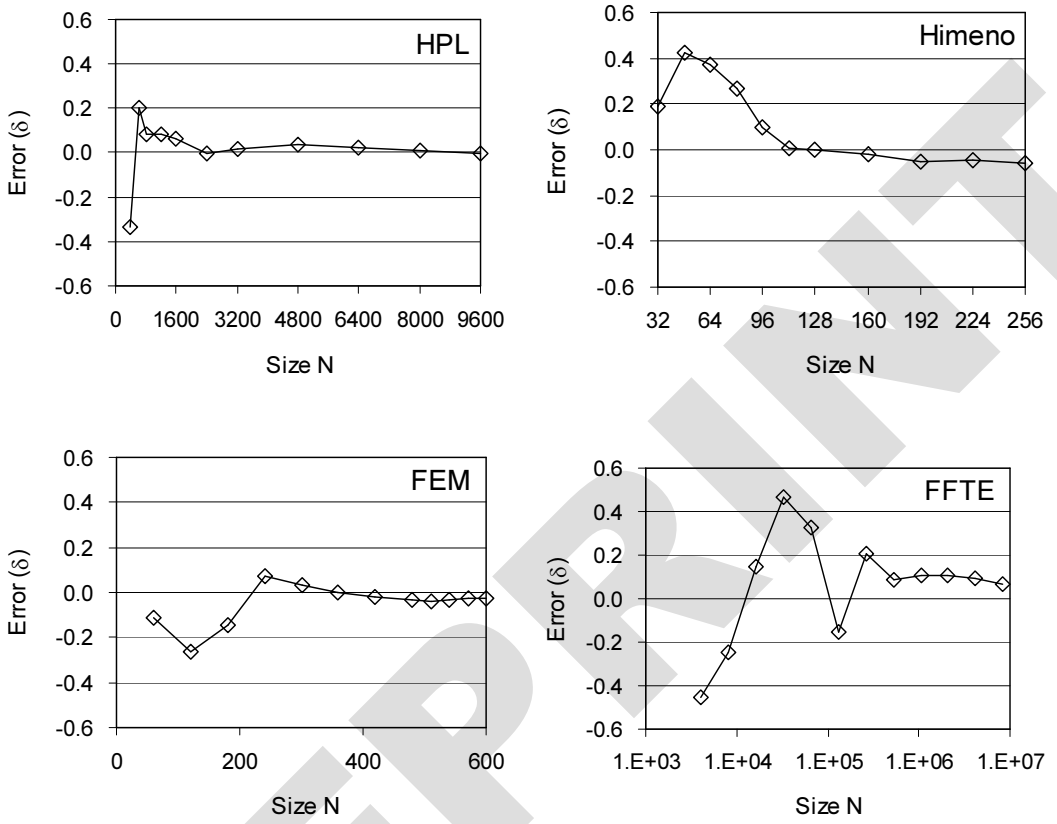


Figure 20. The error δ of NP-T/NNLS for four benchmark programs.

fully accepted in the parallel computing community. Below are some examples of the objections often raised against estimation models including ours.

- Although the results shown here are good, it is a very special case. It may not be applicable to general cases.
- An actual system is too complex to be represented by a simple model.
- Estimation models can only provide rough estimations. The errors are too large for practical purposes.
- Exceptional performance anomalies are often observed, but they are difficult to model.
- Even a homogeneous cluster is difficult to predict. A heterogeneous cluster is a much more difficult target.

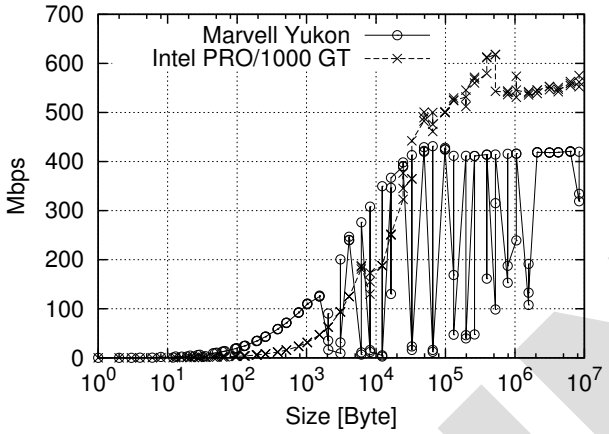


Figure 21. Network transfer rates of Marvell Yukon and Intel Pro/1000GT network interfaces.

Although these objections focus on the quality of models, we would like to point out another aspect of the problem here. A system must show a simple and predictable behavior to keep the corresponding model accurate. In other words, we have to build well-behaved clusters. Performance and reliability issues are often discussed in cluster computing, whereas not enough attention is paid to predictability. This section emphasizes predictability issues, since it is practically very important to make a cluster predictable.

The heterogeneous cluster used in this study was built carefully in order to be predictable. Its components are not special, but they were chosen and coordinated carefully. Under such conditions, the results shown in Section 5 were derived. Though there are many pitfalls to be avoided in building a predictable cluster, some applications (e.g., HPL) are well-behaved and resistant to most of these pitfalls. Thus, the pitfalls shown below did not cause problems in previous studies [1][2] which examined HPL as an example application.

This section examines some checkpoints that we newly confronted and overcame in the process of this study. Hopefully, these might help researchers to build a well-behaved and predictable cluster. They are not specific to heterogeneous clusters, but should be more seriously considered in heterogeneous clusters which consist of a wide variety of components.

7.1. Network interface

Usually, one or more network interfaces are integrated in recent motherboards. However, it does not always yield good results to use an on-board network interface for communication. For example, each Celeron node of our cluster has an on-board Marvell Yukon Gigabit Ethernet interface, whose transfer rates are summarized in Figure 21 for various communication sizes. The results were derived with the

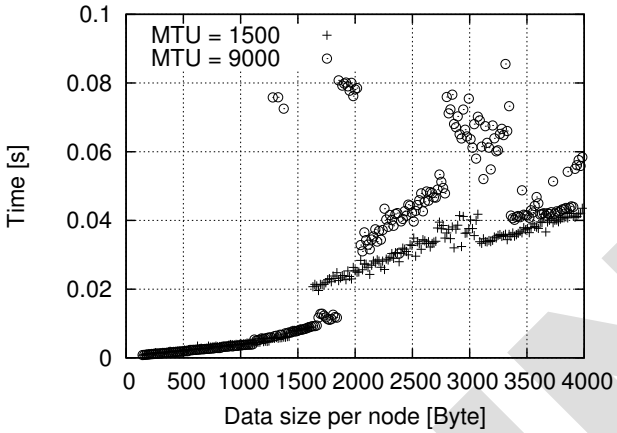


Figure 22. Latencies of all-to-all communication among 16 Xeon processors: MTU=1500 and MTU=9000.

TCP version of NetPIPE [32]. As readily seen, the transfer rates of Marvell Yukon fluctuate and are highly unpredictable. This actually results in unpredictable behavior of application performance.

Therefore, we decided to add an Intel PRO/1000 GT Gigabit Ethernet card for each Celeron node via PCI bus. The derived transfer rates are also shown in Figure 21. Although a PRO/1000 is slower than a Yukon for small communication sizes, it is faster for large sizes and behaves well for any size. By replacing a Yukon with a PRO/1000, the estimation errors were much improved, especially for communication intensive applications (e.g., FFTE).

It should also be noted that device drivers greatly affect the transfer rates. Although it did not make any difference in our case, the device driver for network interface should first be updated before replacing the network interface hardware.

7.2. MTU

Recent Ethernet interfaces and switches can support various MTU (maximum transmission unit) sizes. Though the default MTU value is around 1500 in many cases, recent hardware often supports a larger MTU (i.e., jumbo frame). The peak transfer rate might be improved by using a larger MTU, since the overhead is reduced in communication.

Figure 22 displays the latencies of all-to-all communication for MTU=1500 and 9000. The communication time obviously becomes longer and more unpredictable in the case of MTU=9000. Figure 23 summarizes the execution times of FFTE, in which all-to-all communication occupies a substantial part. It is evident that the FFTE performances of MTU=9000 are actually affected by the anomalies of all-to-all communication time. Contrary to expectations, a large MTU did not improve performance in our applications.

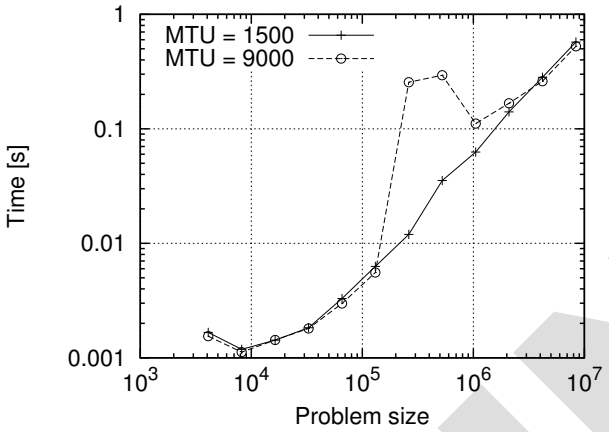


Figure 23. The execution time of FFTE among 16 Xeon processors: MTU=1500 and MTU=9000.

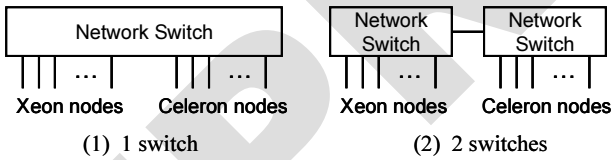


Figure 24. Two network topologies of a simple heterogeneous cluster.

Optimizing MTU for performance is not an easy task. It is not enough to use the hardware that supports jumbo frames. Appropriate parameters must also be set for both hardware and software. Moreover, the best MTU value might depend on the application program. Thus, the default MTU value (1500) was adopted in this study, since it did not lead to unpredictable behaviors of application programs. Although we could not be sure this value is best for performance, we could not find a better one for our cluster.

7.3. Network topology

Needless to say, network topology affects the performance of a parallel program in a cluster. Figure 25 summarizes the latencies of all-to-all communication among 8 Xeons and 8 Celerons for the two topologies shown in Figure 24. If a single wire-speed switch connects all 16 processors, the latencies are kept very low. If two switches are used for Xeons and Celerons, the latencies become much larger

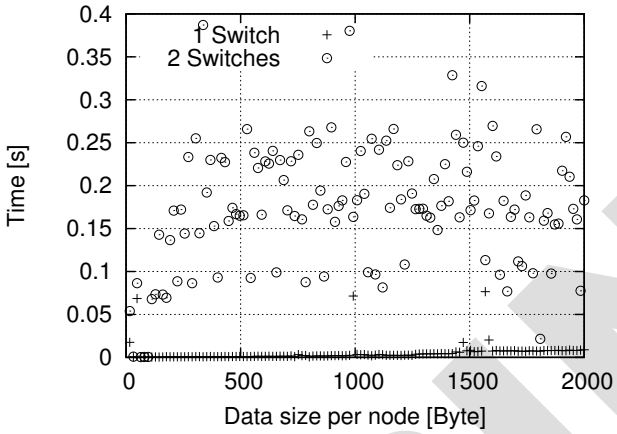


Figure 25. Latencies of all-to-all communication among 8 Xeon and 8 Celeron processors of two network topologies.

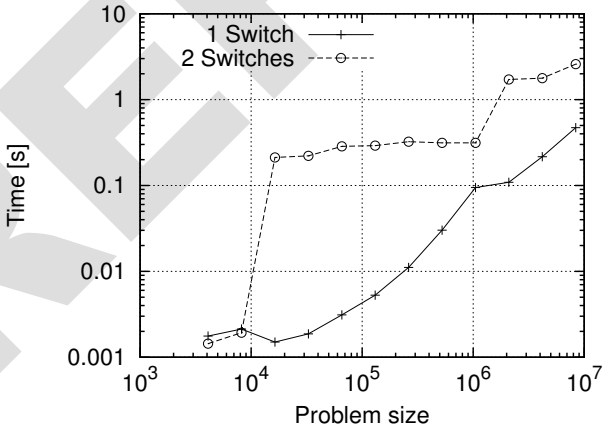


Figure 26. The execution time of FFTE for two network topologies.

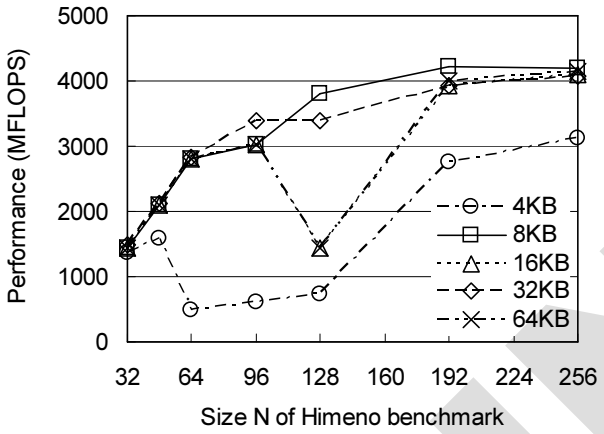


Figure 27. Sensitivity of Himeno benchmark performance to the buffer size of MPI library. Measurements made with 16 Xeon processors.

and more unpredictable due to the heavy contention on the connection between two switches. As seen in Figure 26, this difference heavily affects the execution time of FFTE, where all-to-all communication is a dominant factor.

A heterogeneous network is a challenging target for modeling. Lastovetsky [33] has presented some interesting results on this subject.

7.4. Library

In this study, MPICH [34] [35] [36] was adopted as the MPI library [37] [38] [39]. Since MPICH is an open-source library, it can be configured with various buffer sizes. Though the default value is 16 KB, it does not always yield the best results.

Figure 27 summarizes the performances of the Himeno benchmark for various buffer sizes of MPICH. Performances are obviously sensitive to the buffer size. It is not surprising to see different performance curves for various sizes, but it should be noted that performance anomalies are observed if the buffer size is set to 4 KB, 16 KB, or 64 KB. Such anomalies result in poor estimation quality in modeling. Therefore, in this study, we compiled MPICH with an 8 KB buffer instead of the default 16 KB.

7.5. Power reduction features

Many recent microprocessors are incorporated with power reduction features, which also affect the performances of parallel applications. Figure 28 displays the network transfer rate between Pentium

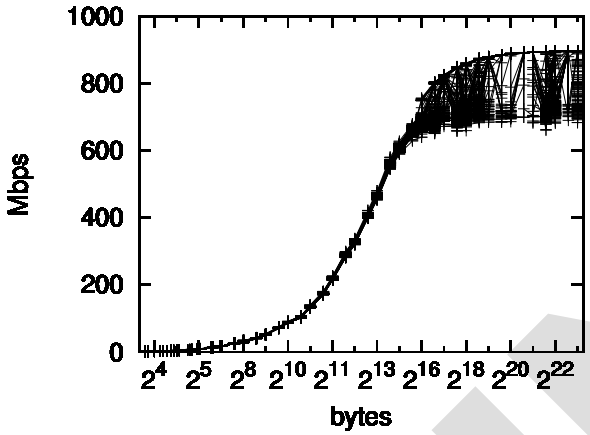


Figure 28. Network transfer rates between Pentium 4 and Athlon 64 X2 with Cool'n'Quiet feature enabled. Figure displays the results of 144 trials, which are quite unstable and unpredictable.

4 and Athlon 64 X2 processors.[§] Measurement results exhibit highly unpredictable and diverse behaviors, particularly for large communication sizes. Such behaviors actually affect the performance of parallel applications.

Such behaviors are caused by Cool'n'Quiet technology of AMD Athlon 64 X2, which dynamically changes the clock frequency to reduce power consumption and heat dissipation. The transfer rate becomes unpredictable by Cool'n'Quiet, since the performance of the microprocessor limits the peak transfer rate. This effect is clearly observed in large communication sizes, where a substantial amount of work is required of the processor. By disabling Cool'n'Quiet in the BIOS menu, the transfer rate becomes very stable, predictable, and reproducible.

8. Conclusions

Invoking multiple processes on fast PEs (multiprocessing scheme) is a simple and straightforward way to alleviate load-imbalance of parallel applications on heterogeneous clusters. Kishimoto and Ichikawa [1][2] constructed performance models of HPL, and actually estimated sub-optimal configurations of heterogeneous clusters. The present study offered some improvements of Kishimoto's scheme, which were evaluated with four typical scientific applications: CFD (computational fluid dynamics), FEM (finite element method), HPL (linear algebraic system), and FFT (fast Fourier transform).

[§] These results were derived with the TCP version of NetPIPE [32].

Our experiments showed that our NP-T/NNLS models are superior to Kishimoto's models, and practically applicable to four benchmarks; the average errors of NP-T models were 0.2% for the Himeno benchmark, 2% for the FEM benchmark, 1% for the HPL, and 28% for the FFTE benchmark. Our study also emphasized the importance of predictability in clusters, listing practical examples derived from our work.

Our models for execution time estimation are extensively applicable for other optimization purposes. Though the estimated execution time itself was adopted as an objective function for optimization in this study, other measures may be adopted instead. An obvious option is to define a cost value for each node, which might be the CPU charge rate or the power consumption rate of that node. With these cost values, we can estimate the total cost for the task to find the best configuration to complete the task at the lowest cost. Another practically useful option is to adopt a heuristic function. For example, it is possible to select the most cost-effective configuration under the condition that its estimated execution time is no more than 110% of that of the estimated optimal configuration. This kind of heuristics is especially useful when the estimated execution time is known to include maximally 10% error; i.e., the 10% difference of the estimated execution time does not matter in such a case.

Though our scheme was originally developed to minimize the execution time, it can be further utilized for the effective management of computational resources. With our scheme, users can estimate the optimal subset of PEs, according to the sizes of their own computation. The remaining nodes generally consist of heterogeneous PEs, which can be utilized for the following computation by recursively applying our scheme. When a job is finished, the corresponding PEs are returned to the pool of available PEs for the next job. Thus, the pool of available PEs is inevitably kept heterogeneous.

Our scheme is not designed for resource scheduling as it is. It is the mission of scheduling algorithms to manage computational resources effectively, while our scheme provides a subordinate function for scheduling algorithms to estimate the execution time of each configuration. However, all these issues are beyond the scope of the present study.

More issues to address in future studies include investigations of more applications, refinement of models for more precise estimations, model construction techniques for more heterogeneous clusters, modeling of heterogeneous network and topology, and process mapping to PEs considering the network topology.

ACKNOWLEDGEMENTS

This study was partially supported by a Grant-in-Aid for Scientific Research from the Japan Society for the Promotion of Science (JSPS). Support for this work was also provided by the 21st Century COE Program "Intelligent Human Sensing" from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

The authors are grateful to the anonymous reviewers for their valuable comments to improve the manuscript.

REFERENCES

1. Y. Kishimoto and S. Ichikawa, "An execution-time estimation model for heterogeneous clusters," in *Proc. 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, IEEE Computer Society, 2004. (CD-ROM).
2. Y. Kishimoto and S. Ichikawa, "Optimizing the configuration of a heterogeneous cluster with multiprocessing and execution-time estimation," *Parallel Computing*, vol. 31, no. 7, pp. 691–710, 2005.

3. A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary, "HPL – a portable implementation of the high-performance Linpack benchmark for distributed-memory computers." http://www.netlib.org/benchmark/hpl/.
4. A. Kalinov and S. Klimov, "Optimal mapping of a parallel application processes onto heterogeneous platform," in *Proc. 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005)*, IEEE Computer Society, 2005. (CD-ROM).
5. A. Kalinov and A. Lastovetsky, "Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers," *Journal of Parallel and Distributed Computing*, vol. 61, pp. 520–535, 2001.
6. O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert, "A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers)," *IEEE Transaction on Computers*, vol. 50, no. 10, pp. 1052–1070, 2001.
7. L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1997.
8. A. Legrand, R. Renard, Y. Robert, and V. Vivien, "Mapping and load-balancing iterative computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 6, pp. 546–558, 2004.
9. Y. Ohtaki, D. Takahashi, T. Boku, and M. Sato, "Parallel implementation of Strassen's matrix multiplication algorithm for heterogeneous clusters," in *Proc. 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, IEEE Computer Society, 2004. (CD-ROM).
10. J. Cuenca, D. Gimenez, and J.-P. Martinez, "Heuristics for work distribution of a homogeneous parallel dynamic programming scheme on heterogeneous systems," *Parallel Computing*, vol. 31, pp. 711–735, 2005.
11. D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: towards a realistic model of parallel computation," in *Proc. Fourth ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPOPP '93)*, pp. 1–12, ACM Press, 1993.
12. A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: incorporating long messages into the LogP model - one step closer towards a realistic model for parallel computation," in *Proc. Seventh annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '95)*, pp. 95–105, ACM Press, 1995.
13. M. I. Frank, A. Agarwal, and M. K. Vernon, "LoPC: modeling contention in parallel algorithms," in *Proc. Sixth ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPOPP '97)*, pp. 276–287, ACM Press, 1997.
14. C. A. Moritz and M. I. Frank, "LoGPC: Modeling network contention in message-passing programs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 4, pp. 404–415, 2001.
15. A. Lastovetsky, I.-H. Mkwawa, and M. O'Flynn, "An accurate communication model of a heterogeneous cluster based on a switch-enabled Ethernet network," in *12th Int'l Conf. Parallel and Distributed Systems (ICPADS 2006)*, vol. 2, pp. 15–20, 2006.
16. J. L. Gustafson and R. Todi, "Conventional benchmarks as a sample of the performance spectrum," *The Journal of Supercomputing*, vol. 13, no. 3, pp. 321–342, 1999.
17. R. Himeno, "Himeno benchmark," 2005. http://accc.riken.jp/E/HPC_e/HimenoBMT_e/index_e.html.
18. K. Nakajima, "Test programs for parallel iterative methods on various types of architectures (hpc-mw-solver-test ver.1.0)," 2003. http://www.fsis.iis.u-tokyo.ac.jp/en/result/software/.
19. K. Nakajima, "OpenMP/MPI hybrid vs. flat MPI on the Earth Simulator: Parallel iterative solvers for finite element method," in *Proc. 5th International Symposium on High Performance Computing (ISHPC 2003)*, LNCS 2858, pp. 486–499, Springer, 2003.
20. JAMSTEC, "The Earth Simulator," 2005. http://www.es.jamstec.go.jp/.
21. M. Yokokawa, "Present status of development of the Earth Simulator," in *Proc. Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'01)*, pp. 93–99, IEEE Computer Society, 2001.
22. Top500.Org, "TOP500 supercomputer sites," 2005. http://www.top500.org/.
23. "HPC challenge benchmark," 2005. http://icl.cs.utk.edu/hpcc/.
24. D. Takahashi, "FFTE: A fast Fourier transform package," 2005. http://www.ffte.jp/.
25. D. Takahashi, "Efficient implementation of parallel three-dimensional FFT on clusters of PCs," *Computer Physics Communications*, vol. 152, pp. 144–150, 2003.
26. Free Software Foundation, "GSL – GNU scientific library," 2005. http://www.gnu.org/software/gsl/.
27. "Automatically tuned linear algebra software (ATLAS)," 2006. http://math-atlas.sourceforge.net/.
28. S. Goedecker and A. Hoisie, *Performance Optimization of Numerically Intensive Codes*. SIAM, 2001.
29. K. R. Wadleigh and I. L. Crawford, *Software Optimization for High Performance Computing*. Prentice Hall PTR, 2000.
30. D. F. Bacon, S. L. Graham, and O. J. Sharp, "Compiler transformations for high-performance computing," *ACM Computing Surveys*, vol. 26, no. 4, pp. 345–420, 1994.
31. C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*. Classics in Applied Mathematics 15, Society for Industrial Mathematics, 1987.
32. "NetPIPE: A Network Protocol Independent Performance Evaluator," 2007. http://www.scl.ameslab.gov/Projects/NetPIPE/.
33. A. Lastovetsky, *Parallel Computing on Heterogeneous Networks*. Wiley, 2003.

34. W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "High-performance, portable implementation of the MPI Message Passing Interface Standard," *Parallel Computing*, vol. 22, no. 6, pp. 789–828, 1996.
35. W. Gropp *et al.*, *MPICH2 User's Guide (Ver. 1.0.5)*. Mathematics and Computer Science Division, Argonne National Laboratory, 13 Dec. 2006.
36. "MPICH2," 2006. http://www-unix.mcs.anl.gov/mpi/mpich/.
37. M. Snir and W. Gropp, *MPI: The Complete Reference*. MIT Press, 1998.
38. Message Passing Interface Forum, "MPI-2: Extensions to the Message Passing Interface," July 1997.
39. "Message Passing Interface Forum," 2007. http://www.mpi-forum.org/.