

# Performance Prediction of Large-scale Parallel System and Application using Macro-level Simulation

Ryutaro Susukita

Institute of Systems, Information Technologies &  
Nanotechnologies  
Fukuoka, Japan  
susukita@isit.or.jp

Hisashige Ando

Fujitsu  
Tokyo, Japan  
hando@jp.fujitsu.co.jp

Mutsumi Aoyagi

Research Institute for Information Technology  
Kyushu University  
Fukuoka, Japan  
aoyagi@cc.kyushu-u.ac.jp

Hiroaki Honda

Institute of Systems, Information Technologies &  
Nanotechnologies  
Fukuoka, Japan  
dahon@c.csce.kyushu-u.ac.jp

Yuichi Inadomi

Institute of Systems, Information Technologies &  
Nanotechnologies  
Fukuoka, Japan  
inadomi@isit.or.jp

Koji Inoue

Department of Informatics  
Kyushu University  
Fukuoka, Japan  
inoue@i.kyushu-u.ac.jp

Shigeru Ishizuki

Fujitsu  
Tokyo, Japan  
sishi@jp.fujitsu.com

Yasunori Kimura

Fujitsu  
Tokyo, Japan  
ykimura@jp.fujitsu.com

Hidemi Komatsu

Fujitsu  
Tokyo, Japan  
komatsu@strad.ssg.fujitsu.com

Motoyoshi Kurokawa

RIKEN (The Institute of Physical & Chemical Research)  
Wako, Japan  
motoyosi@riken.jp

Kazuaki J. Murakami

Department of Informatics  
Kyushu University  
Fukuoka, Japan  
murakami@i.kyushu-u.ac.jp

Hidetomo Shibamura

Institute of Systems, Information Technologies &  
Nanotechnologies  
Fukuoka, Japan  
shibamura@isit.or.jp

Shuji Yamamura

Fujitsu  
Tokyo, Japan  
yamamura.shuji@jp.fujitsu.com

Yunqing Yu

Department of Informatics  
Kyushu University  
Fukuoka, Japan  
yu@c.csce.kyushu-u.ac.jp

**Abstract**—To predict application performance on an HPC system is an important technology for designing the computing system and developing applications. However, accurate prediction is a

challenge, particularly, in the case of a future coming system with higher performance.

In this paper, we present a new method for predicting application performance on HPC systems. This method combines modeling of sequential performance on a single processor and macro-level simulations of applications for parallel performance on the entire system. In the simulation, the execution flow is traced but kernel computations are omitted for reducing the execution time. Validation on a real terascale system showed that the predicted and measured performance agreed within 10% to 20 %. We employed the method in designing a hypothetical petascale system of 32768 SIMD-extended processor cores. For predicting application performance on the petascale system, the macro-level simulation required several hours.

*Keywords-component; performance prediction; large-scale system; large-scale application*

## I. INTRODUCTION

It is important to predict application performance on a large-scale system for designing such computer systems. Application performance is one of key benchmarks to determine various components of the system, e.g., processor architecture, the number of processors, communication performance and selection between homogeneous and heterogeneous systems. Performance prediction is useful also for developing applications executed on large-scale systems. If application performance is predicted for any given problem size, selected algorithm, the number of parallel processes etc, the developer can begin to optimize the application even before the new computer system is available. Indeed, application development and system design are inseparable from each other. When we attempt to predict application performance on a specifically designed system, we often need applications optimized for the system.

There are two well-known approaches for performance prediction. One approach is to develop a system simulator to execute applications on it for performance prediction. This approach has the advantage of accurate prediction using a simulator with the detailed system information. The difficulty is that such kind of simulations require a large amount of computational cost because an accurate system simulator is much slower and more memory-consuming than the real system. In particular, when we predict for a non-existing, future system with higher performance than existing systems, the computational cost is too much to simulate the entire system. There are a few studies on simulation-based prediction for recent large-scale systems (e.g., [7, 12, 13]). However, problem sizes are relatively small as recent large-scale computations and the time scale of simulations remains one second in the target system.

The second approach is to model application performance on the target system. In this approach, the computational cost of the application is estimated for any given input parameters, e.g., problem size and algorithm selection. The performance is modeled as a function of input parameters and processor performance. The model-based prediction can be less accurate than the simulation-based prediction because complicated system and application behaviors are simplified in performance modeling. Instead, the performance is easily predicted without computational cost once the model is made.

However, in large-scale parallel applications, the execution time is determined by complicated convolution of sequential calculation times in each process, load balance and communication times (e.g., [6]), this leads to a new difficulty in performance modeling.

There are numerous studies also on others approaches. MPI-SIM [8, 9] was a simulator for predicting application performance on large-scale systems. It simulated only the communication library but directly executed applications on real processors. MPI-SIM was integrated with compiler optimization technique for reducing computational cost [1]. However, direct execution is possible only when the target system consists of the same processors as an existing one. The Performance Modeling and Characterization framework [2, 10, 11] predicted application performance from a function of system profiles (e.g., memory performance, communication performance) and application signatures (e.g., memory access pattern, communication pattern). Basically the application signatures were determined by executing the application in the framework, which required that the application scale was sufficiently small for framework user's system. ChronosMix [3] was a performance prediction tool by using static analysis of applications. The application performance was predicted from the analysis result and measured processor and network benchmarks. ChronosMix aimed at performance prediction for cluster systems consisting of existing, real processors and network. Also the combination of a simple algebraic model and a performance prediction tool was used for parallel applications [14]. In the model, it was assumed that the execution time of an application was expressed as the sum of calculation and communication times separated by synchronization points. Those times were modeled as simple algebraic expressions. The tool predicted the application performance from the expressions.

In this paper, we present a new prediction method for large-scale parallel systems and applications. This method combines performance modeling for sequential calculation blocks and macro-level simulations for parallel performance. The sequential performance is modeled using processor simulations or real executions. To execute calculation blocks on the processor simulator requires much smaller computational cost than that to simulate the entire system. The performance models are embedded in the application. In the macro-level simulation, only calculation times predicted from the models are counted, considering load balance and communication times. Since real calculations are not performed, the macro-level simulation does not require large computational cost either.

The paper is organized as follows. In Section 2, we describe the prediction method in detail. Then we validated the method on a real large-scale system. The results are presented in Section 3. We employed our method in designing a petascale system. Section 4 shows the performance prediction of the system. Finally, in Section 5, we give conclusions of this paper.

## II. METHODOLOGY

Our method predicts the execution time of an application in the following steps (Fig. 1):

- Predict and model sequential performance for calculation blocks in computational kernels of the application
- replace calculation blocks with sequential performance (skeletonize)
- model communication performance of the system
- execute the skeleton in an environment (BSIM) for predicting parallel performance

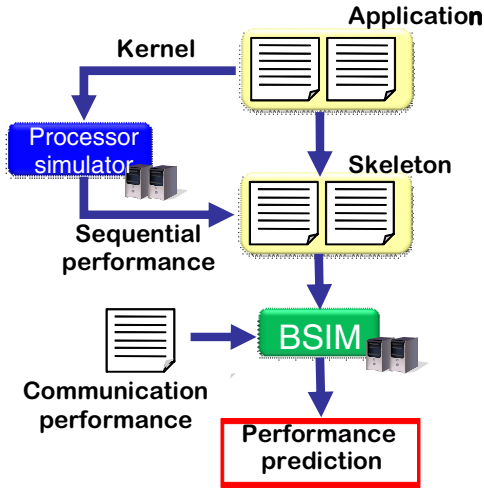


Figure 1. Prediction flow

### A. Skeletonizing applications

First the target application is analyzed for finding computationally intensive kernels. Scientific applications generally consume most of the execution time only in its small portion. Therefore, we approximate the application performance as the kernel performance. Then we predict the execution times of calculation blocks in the kernels using processor simulations or real executions. The predicted execution time may be modeled as a function of parameters such as problem size, loop counter or MPI rank.

After we modeled the predicted execution time, we replace the calculation block with a `BSIM_Add_time(t)` call, where  $t$  is the execution time predicted by the model. The `BSIM_Add_time` function passes the predicted execution time to BSIM. BSIM is an environment we developed for predicting parallel performance as described in Section 2.C. After the replacement, the application does not produce meaningful results any longer. This skeleton is executed only for performance prediction. If a calculation block is necessary to preserve the execution flow of the application, the calculation

block is not removed. For example, if a calculation block determines the number of iterations for a loop, the calculation block is not removed. In some applications, most calculation blocks are necessary to reserve the execution flow. Consequently, few calculation blocks are removed. Our method is not suitable for such applications. A similar problem is discussed in compiler-optimized simulation [1]. However, in many kernels, even after most of calculations are replaced with `BSIM_Add_time` calls, the execution flow is preserved. Thus, skeletonization dramatically reduces the calculation cost of the application. Note that unremoved calculation blocks do not affect the predicted application performance.

In addition, some of MPI calls are replaced with LMPI functions. An LMPI function is a BSIM API with the same interface as the corresponding MPI function has. The LMPI function does not communicate real messages but only passes the function arguments to BSIM for predicting the communication time. If an MPI call is necessary only to correct calculations but not necessary to preserve the execution flow of the application, the MPI call can be replaced with an LMPI call. Since LMPI functions do not communicate real messages, the replacement with LMPI calls reduces the communication cost of the application.

Skeletonization enables us to reduce the memory usage of the application as well. The target system may have much larger amount of memory than the system where the skeleton is executed (hereafter the host system) has. Target applications often require memory close to all available memory in the target system. Therefore, we need to reduce memory usage for executing the application on the host system. In skeletonization, after calculation blocks are removed, the application does not need to allocate memory accessed in the calculation blocks. Similarly, after MPI calls are replaced with LMPI calls, the application does not need to allocate data arrays the MPI functions communicate. Since scientific applications generally require a large amount of memory and communicate large data arrays only in computational kernels, most of the memory usage is reduced in skeletonization.

From our experience, the cost of skeletonization is relatively low for an application developer. A considerable portion of skeletonization is necessary for developing applications executed on large-scale systems even if the application is not skeletonized. Finding computational kernels is generally indispensable for optimizing such applications. To remove calculation blocks in the kernel without the change of the execution flow, the knowledge of the execution flow is required, but the application developer may be obliged to know it in any case when he develops the application. Also replacement of MPI calls with LMPI calls needs how to communicate in the kernel, but it is important information for optimization the application. Performance modeling of calculation blocks is probably the major additional cost in the skeletonization.

### B. Communication time

For taking account of communication into prediction of parallel performance, we model communication time on the target system. The models are made for send operation, receive

operation and collective communications based on the communication performance of the system. These models are used when BSIM predicts communication time. In each model, communication time is expressed as a function of the communication attributes. For example, in the model for receive operation, the communication time is expressed as a function of the message size, the source rank and the destination rank in `MPI_COMM_WORLD`. The function includes process-to-node mapping for converting the ranks to node numbers. The communication time is predicted from the message size and communication performance between the source and the destination nodes. Similarly, for collective communications, communication time is expressed as a function of the message size, the number of processes involved in the communication etc.

These models of communication time can include not full but some information on delay due to interactions between communications, for example, the conflict of access to node resources that occurs when two communications are performed simultaneously in the same node, and contention on the interconnect. These interactions are divided into two types. The first type is an interaction between MPI communications, e.g., between MPI point-to-point communications and between collective communications. The second type is an interaction between primitive point-to-point communications for performing a collective communication. The models of communication time can include only information on delay due to the second type of interaction. If a simulator of the target system is available and it supports system nodes and the interconnect, it is possible to model communication times of collective communications including such delay by simulating the communications. If we model communication times by measuring on a real system, the measured times of collective communications may include the delay.

### C. Executing skeleton

After calculation blocks are replaced with `BSIM_Add_time` calls, i.e., skeletonized, the skeleton is executed with BSIM on the host system. BSIM is an environment we developed for predicting the execution time from a skeleton and communication time models. BSIM supposes that the execution time of each process is the sum of calculation times, wait times and communication times. A calculation time is the execution time of a calculation block in a computational kernel. The predicted calculation time is passed from the skeleton to BSIM by the `BSIM_Add_time` function.

Wait time is the time which an MPI function waits until the corresponding MPI function being called in a different process. One example is the time that `MPI_Recv` waits until the corresponding `MPI_Send` is called. A wait time expresses the load imbalance between the two processes before the communication. BSIM accumulates predicted calculation times, wait times and communication times in a simulation clock for each process. When two processes communicate in the skeleton, BSIM compares the simulation clocks of the two processes. If the simulation clock of send process is advanced beyond that of receive process, the wait time of the receive process is predicted to be the time difference between the two

clocks. BSIM assumes send operations to be always asynchronous, so that wait time is predicted to be zero if the simulation clock of the receive process has advanced beyond that of the send process. In the case where this assumption is too simple, BSIM can be easily extended to support synchronous send operations for long messages. Wait times are predicted for collective communications in a similar way. When the skeleton calls `MPI_Bcast`, BSIM compares simulation clocks between the send process and each receive process that calls `MPI_Bcast`. If the simulation clock of the send process is advanced beyond that of a receive process, the wait time of the receive process is predicted to be the time difference between the two clocks.

A communication time is the execution time of an MPI function. BSIM predicts it from the communication time model. For example, the execution time of `MPI_Recv` is predicted from the model for receive operation. The model parameters are message size, source rank and destination rank. The message size and the source rank are passed from `MPI_Recv` or `LMPI_Recv` arguments. The destination rank is obtained from the process itself that calls `MPI_Recv`.

The skeleton is compiled and executed as an MPI application on the host system. When the skeleton calls an MPI function, BSIM intercepts the call and calculates the wait time and the communication time. One problem in executing skeleton is the number of process. Since the number of application processes executed on the target system can be much larger than the number of processor cores of the host system, we need to execute a large number of application processes per processor core. The operating system or the MPI library may not support such circumstance. The number of processes is limited by several reasons. For example, when a large number of application processes are executed on a single core, each process is executed very slowly. As a result, the response time of the communication may exceed the timeout of the MPI library. Furthermore, if the MPI library consumes file descriptors with the increase of the number of processes, the number of files descriptors may exceed the default limit of the system. To increase the process limit, we extended the timeout of the MPI library or increased the file limit. After this improvement, we confirmed that at least 64 or 128 application processes were executed per processor core on the systems where we executed skeletons.

## III. EXPERIMENTS ON EXISTING SYSTEM

For validating the prediction accuracy of our method, we made experiments on a real large-scale system. We skeletonized two applications by replacing calculation blocks with their predicted execution times. Skeletons were executed with BSIM for predicting parallel performance. Communication times were modeled by measuring communication performance on the system. Finally we compared the prediction results to real execution times.

### A. Target system

We chose Fujitsu PRIMEQUEST 580 as target system for validating the prediction accuracy of our method. The system is a fat-node cluster for large-scale applications. A user job can

use up to 1024 processor cores. The hardware specification is given in Table 1. Each node is connected via 4X DDR InfiniBand crossbar network. Each connection uses four ports

and data is transferred in 8B/10B encoding, so that the useful rate is 4 GB/s in each direction.

TABLE I. SYSTEM SPECIFICATION

System	Max nodes / user job	# of processors / node	Processor	Memory / node	Interconnect	Peak performance / user job
PRIMEQUEST 580	16	32	Dual-core 1.6 GHz Itanium 2	128 GB	InfiniBand	6.6 TFlop/s
PRIMERGY RX200 S2	16	2	3 GHz Xeon	7 GB	Gigabit Ethernet	192 GFlop/s
RSCC Linux cluster	128	2	3.06 GHz Xeon	4 GB	Gigabit Ethernet	1.6 TFlop/s

### B. Target application

We chose HPL (High Performance LINPACK, e.g., [5]) and PHASE as application programs for the verification of the proposed performance evaluation methodology. HPL is a well known HPC benchmark for solving linear system of equations using LU decomposition with two dimensional block cyclic data distribution. Most of the computation in HPL is in matrix updates; double precision matrix multiply using dgemm subroutine.

PHASE is a first principles MD (Molecular Dynamics) analysis program based on the DFT(Density Functional Theory) and pseudo potential method. It uses Car-Parrinello method [4] for solving Schrödinger equation. It contains three computationally intensive portions; pseudo potential product, Gram-Schmidt orthogonalization and 3D FFT.

PHASE as it was developed is MPI parallelized, but the parallelization is done with one dimension only and is not suitable for petascale computation. We have modified PHASE for parallelizing it with both wave numbers and energy bands. This modification not only allows it to use more MPI nodes in parallel, but also reduces the amount of data transfer between the three computation intensive kernels.

Majority of the computation, done in pseudo potential product and Gram-Schmidt orthogonalization, are double precision complex matrix multiply using zgemm subroutine. Significant amount of inter-node communication is done by using MPI\_Broadcast and MPI\_Allreduce in both pseudo potential product and Gram-Schmidt orthogonalization. 3D FFT is done with three 1D FFTs with global matrix transpose between the 1D FFTs.

### C. Skeletonizing application

For the performance evaluation of a large scale target system using a small host system of 1/100 size, application programs need to be modified to run 100x faster and also memory requirement need to be reduced to 1/100.

For HPL, dgemm, dgemv and dtrsm calls were replaced with BSIM\_Add\_time calls with the separately estimated execution times. This removes all the computationally intensive codes and reduces execution time of the skeleton code. All the pointer references to the matrix to be solved were changed to the reference to a small work array, and then, memory allocation of the matrix is removed. This change reduced required main memory by NB(block size)/N(matrix size) and

allows to run more than 100 copies of the skeleton code on one host system node.

For the PHASE, zgemm, zgemv and 1D fft calls were replaced with the BSIM\_Add\_time calls to reduce execution time on the small host system. Input parameters for small scale problem which requires less than 200KB of main memory were used as input, then separately specified scale factors (Nfscale and Nescale) scale the time value given to BSIM\_Add\_time calls for representing the computation time for the large scale problem. Similarly, data size for MPI calls were scaled with these scale factors. Fig. 2 shows a portion of original code and Fig. 3 shows the corresponding portion of the resulted skeleton code.

```

nnn=Nfp*Nep
call MPI_BCAST(PSI2,nnn,MPI_DOUBLE_COMPLEX,lrnk,
& NCOM2(1),ierr)
do 42 nb=1,Nblk
  nbss=(nb-1)*Ni+myir+1
  if(nbss.le.(Ne-1)/Nep+1 .and. nbss.ge. nbs+1) then
    call zgemm('C','N',Nep,Nep,Nfp,z1,PSI2(1,1),Nfp,
    & PST(1,4,nb),Nfp,z0,CC(1,1,nb),Nep)
  endif
42 continue
nnn=Nep*Nep*(Nblk-nbsn+1)
call MPI_ALLREDUCE(CC(1,1,nbsn),CCW(1,1,nbsn),nnn,
& MPI_DOUBLE_COMPLEX,
& MPI_SUM,NCOM(1),ierr)

```

Changes to LMPI call

Changes to BSIM\_Add\_time call

Changes to LMPI call

Figure 2. Original Code

```

nnn=Nfp*Nep*Nfscale*Nescale
call LMPI_BCAST(ZPSI2,nnn,MPI_DOUBLE_COMPLEX,lrnk,
& NCOM2(1),ierr)
mx=(Nep*Nescale-1)/32+1
nx=((Nep*Nescale-1)/256+1)*8
jx=(Nfp*Nfscale-1)/32+1
time_val=7.9d-8+mx*(0.512d-6*jx+1.141d-6*nx
& +0.666d-6*jx*nx)
time_val=time_val/2.0d+0*Nblk
call BSIM_Add_time(time_val)
nnn=Nep*Nep*(Nblk-nbsn+1)*Nescale*Nescale
call LMPI_ALLREDUCE(ZCC,ZCCW,nnn,
& MPI_DOUBLE_COMPLEX,
& MPI_SUM,NCOM(1),ierr)

```

Scaled size given to LMPI call

zgemm call replaced with BSIM\_Add\_time call

Scaled size given to LMPI call

Figure 3. Skeletonized Code

Coefficients of the execution time equation were determined from the measurements using the small number of computation nodes of the target system. Each computation subroutine was run with variety of input parameters and the coefficient for each input parameter was determined by curve fitting.

Fig. 4 shows an example of the curve fitting for dgemm subroutine for 1.6GHz Itanium 2 node. Matrix size and block size were chosen to match with the size equivalent to the HPL panel size for each node to calculate. Horizontal axis of Fig. 4 is number of floating point operations. Time equation has the slope of 0.173ns/Flop for the region up to 10M Flop and 0.165ns/Flop for higher Flop region.

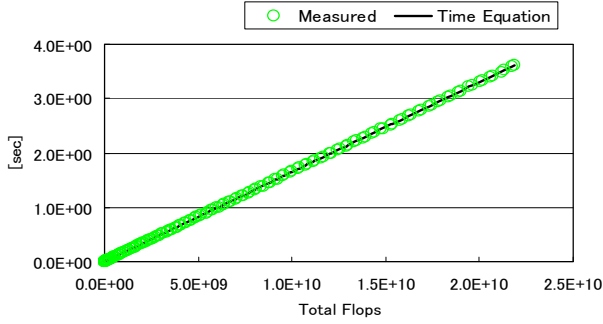


Figure 4. Execution time curve fitting for dgemm

#### D. Communication time on PRIMEQUEST

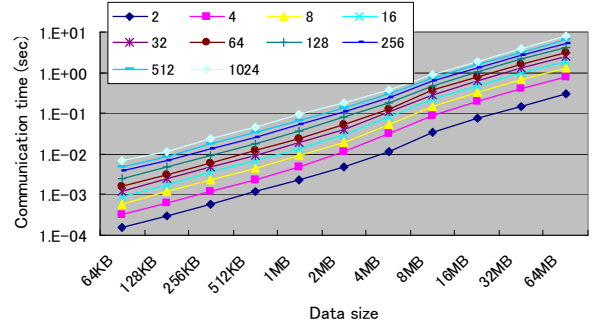
We modeled communication time by using measured communication performance of PRIMEQUEST. For send operation, we approximated the communication time as zero. It is good approximation for measured values unless the message is too long to send asynchronously. We modeled the communication time of receive operation as  $l + s/w$ , where  $l$ ,  $s$  and  $w$  are the minimum latency, the message size and the bandwidth, respectively. We chose  $l = 40$  microseconds and  $b = 100$  MB/s for roughly fitting measured communication times in round-trip communication and HPL with a small problem size and a small number of processes. This  $b$  value is much poorer than expected from the interconnect performance. Measured communication times were not largely different between inter-node and intra-node communications. Therefore, we believe the communication times are governed by data transfer inside the PRIMEQUEST node rather than on the interconnect.

For collective communications, we modeled communication time by interpolating measured values. We measured communication times for each collective communication, varying the data size and the number of processes involving the collective communication and then linearly interpolated measurement values. For example, we show measured communication times of MPI\_Allreduce, which is the most time-consuming communication in PHASE, with data type of double precision and the reduce operation of sum (Fig. 5). In the case of global reduce operation such as MPI\_Allreduce, communication time possibly depends on the data type and the reduce operation. We also measured with other data types and reduce operations but found no significant difference in communication times. Therefore, we used the same model for all data types and all reduce operations.

Figure 5. Communication time of MPI\_Allreduce measured on PRIMEQUEST

#### E. Executing skeletons and prediction results

We executed HPL and PHASE skeletons with BSIM for



predicting execution times. The skeletons were executed on Fujitsu PRIMERGY RX200 S2 (Table 1). We executed HPL skeleton for six sets of parameter values and PHASE skeleton for three parameter values. Each execution was completed in one hour at the maximum. We would like to emphasize that some execution times of skeletons were shorter than real execution times on PRIMEQUEST.

Comparisons of predicted and real execution times are shown in Total columns of Tables 2 and 3. Calculation columns list the sum of calculation times excluding wait and communication times. A positive (or negative) error indicates that the predicted time is longer (or shorter) than the real time.

TABLE II. HPL PREDICTION ON PRIMEQUEST

Problem size	# of processes	Block size	Total (sec)			Calculation (sec)		
			Prediction	Real	Error	Prediction	Real	Error
160000	1024	120	707	770	-8%	468	481	-3%
240000	1024	120	2047	2155	-5%	1548	1580	-2%
320000	1024	120	4484	4348	3%	3630	3624	< 1%
160000	256	120	2168	2043	6%	1814	1780	2%
160000	512	120	1156	1119	3%	915	905	1%
320000	1024	180	4543	4562	< 1%	3684	3764	-2%

TABLE III. PHASE PREDICTION ON PRIMEQUEST

# of bands	Total (sec)			Calculation (sec)		
	Prediction	Real	Error	Prediction	Real	Error
4096	23.5	28.8	-19%	11.6	12.3	-5%
8192	84.0	97.8	-14%	39.8	39.2	2%
16384	301	336	-10%	146	148	-1%

For HPL, predicted and real execution times agreed within 10%. Predictions of calculation times were more accurate. Although predictions were less accurate (errors of 10 to 30 %) for communication times, the errors were moderated in total execution times because the ratio of calculation times was small in HPL. Wait times were included in total execution times but very little. PHASE performance was predicted with errors of 10 to 20%. Since the ratio of communication times was larger than that in HPL, the overall accuracy was more affected by the prediction accuracy for communication times. However, the prediction accuracy of communication times was not so much different between HPL and PHASE.

#### IV. PREDICTION OF PETASCALE PERFORMANCE

We employed our prediction method for a hypothetical petascale system we designed. First we simulated a processor of the system for predicting execution times of calculation

blocks in applications. We replaced the calculation blocks with the prediction results for skeletonizing the applications. By executing the skeletons with BSIM, we obtained performance prediction for the petascale system.

##### A. Target system

For the petascale target system, we have assumed an 8 core computation node with each core having 16 SIMD FMA (Floating-point Multiply and Add) units. It was also assumed that this computation node runs at 2.0GHz clock and having 512GFlop/s peak performance. Assumed target petascale system includes 4096 computation nodes connected in 16x16x16 3D torus. An aggregated peak performance of the target system is 2.1PFlops.

For the 3D torus, link bandwidth is assumed to be 20GB/s (10GB/s for each direction). This link bandwidth resulted in 0.24B/Flop communication-computation ratio. Table 4 summarizes the specifications of the petascale target system.

TABLE IV. SPECIFICATIONS OF PETASCALE SYSTEM

	Specification	Performance
Processor core	4 Issue Super-Scalar Core + 16 SIMD FMA units	2GHz Clock 64GFlop/s
Compute Node	8 Processors + 64GB memory 6 interconnect ports	512GFlop/s 6 x 10GB/s x 2 = 60GB/s
System	4096 Compute Nodes 16 x 16 x 16 3D Torus	2.1PFlop/s BiSec BW=10.24TB/s

For the SIMD processor core, we have developed a detailed microarchitecture and a series of tools; SIMD compiler, SIMD emulator/tracer, cycle accurate architecture simulator for performance evaluation and energy calculator based on simulated pipeline activities. Also the SIMD optimized version of each computation subroutines was developed.

##### B. Target applications

The same HPL and PHASE were used for petascale target system performance predictions. For the HPL, problem size of N=1310720 was chosen and the NB=512, P=64, Q=64 were used. The amount of floating point operation for solving this problem is about 1.5Exa Flops.

For the PHASE, a set of input parameters for representing a system of 10 thousand atoms class was chosen. The number of floating point operations for each iteration of the self consistent convergence loop was about 108PFlops.

##### C. Skeletoning applications

Execution times of the SIMD core were evaluated with the architecture simulator by running the SIMD optimized subroutines. As shown in Table 5, SIMD extended core achieves 2.6x performance for 1024 point fft and 6.9x performance for N=1024 dgemm compared with the 4-issue superscalar processor core without 16 SIMD units.

TABLE V. PERFORMANCE IMPROVEMENT WITH SIMD EXTENSION

	Base Scalar Processor	16 SIMD extended	
	Execution time (ms)		Perf.
dgemm N=1024	368.9	53.12	6.9x
zgemm N=1024	1114.6	185.82	6.0x
fft 1024 point x8	0.1764	0.0668	2.6x

The SIMD core performance for N=1024 dgemm is 20.2 Flop/cycle which is 63% of the peak Flop/s of SIMD unit. Although the ratio of this performance to the peak Flop/s is low compared to the 73% of the scalar processor, SIMD extended processor is more efficient in performance per silicon area and it is suitable for the petascale system.

For each SIMD optimized subroutines, the number of execution cycles for each loop and loop component were obtained through single core architecture simulation. These execution cycle numbers were modified to include the effect of memory access delay due to 8 core access conflict for the portions where delay is expected. Then, the equations for the

execution time for each subroutine were formulated. These equations were used for calculating BSIM\_Add\_time values for skeleton codes.

#### D. Executing skeletons and prediction results

We executed HPL and PHASE skeletons with BSIM for predicting performance on the petascale system. For HPL, we modeled communication times according to the communication performance specified in the text of Section 4.C and Table 4. As in the case of PRIMEQUEST (Section 3.C), we assumed zero communication times for send operation. Communication times of receive operation were expressed as  $l + s/w$ . The minimum latency  $l$  was calculated along the shortest route between the two nodes in the 3D torus. For PHASE, we focused on effects of load balance. Both skeletons were executed on RIKEN Super Combined Cluster (RSCC) Linux Cluster (Table 1) and PRIMERGY. The first predicted performance of each application was remarkably lower than the peak performance. After we reanalyzed and improved the parallelization, we executed the new skeleton.

As shown in Table 6, we obtained prediction of petascale performance for HPL and PHASE. The predicted elements column shows which elements of calculation, wait and communication are included in the prediction. HPL performance was predicted to be over 1 PFlop/s. It corresponds

to approximately 50% of the peak performance. But, the original HPL skeleton predicted un-expectedly low performance. We found that this is due to the low performance of SIMD dgemm and dgemv for smaller matrices ( $N \ll 1024$ ) which we overlooked their contributions. With these findings, SIMD dgemm and dgemv codes were improved for better handling of small matrix to achieve over 1.0 PFlop/s performance.

Also the PHASE skeleton performance was lower due to load imbalance among the compute nodes. Input parameters to PHASE were modified to reduce the granularity of the processing. This resulted in longer execution time for the first part of the PHASE, but the overall execution time was reduced by the larger reduction in execution time in the second part with improved load balance. But, still the predicted PHASE performance with wait is 30 % lower than the one without wait due to the remaining load imbalance. From these experiences, we believe our method not only predicts application performance efficiently but, also is helpful for optimizing applications.

These predictions were obtained from skeleton executions on systems with lower performance by a factor of thousand or ten thousands. This fact shows extremely high scalability of the prediction method.

TABLE VI. PERFORMANCE PREDICTION OF PETASCALE SYSTEM

Application	Predicted elements	Predicted time (sec)	Performance (PFlop/s)
HPL	calc, wait	1397	1.07
HPL	calc, wait, comm	1477	1.02
PHASE	calc	124	0.87
PHASE	calc, wait	165	0.65

Finally we compared the execution time of each skeleton and the real execution time of the original application on the host system. As mentioned in Section 2.A, it was generally impossible to execute the same computation on the host system as on the target system because memory usage was too large. Instead of real executions, we estimated the execution time from the calculation cost and the peak performance of the host system.

As shown in Table 7, the execution times of skeletons were roughly 40 times shorter than estimated execution times of the original applications. The numbers in the original execution column were calculated assuming 100% effective peak Flop/s execution. Since the real performance is expected to be considerably lower than the peak performance, it is concluded that applications were apparently accelerated by a factor of more than 40 in skeletonization and execution with BSIM.

TABLE VII. BSIM EXECUTION PERFORMANCE

Application	Host system	Skeleton execution (hr)	Original execution (hr)	Speedup
HPL	RSCC	~6	> 261	> 44
PHASE	PRIMERGY	~ 4	> 156	> 39

performance, load balance and the given communication performance.

#### V. CONCLUSION

In this paper, we presented a new method for predicting application performance on large-scale systems. First we model the sequential performance of each calculation block in application kernels. The performance is predicted using processor simulations or real executions. Then we replace calculation blocks with these models in the application. The skeletonized application is executed with BSIM, an environment we developed for predicting parallel performance. BSIM predicts the performance from the sequential

Our method predicted application performance on a real terascale system with errors of 10 to 20%. The predictions were obtained from skeleton executions on a system with performance of only 1/30. While the prediction accuracy is similar to those of simulation-based prediction, the computational cost is much smaller. In addition, we were able to predict the performance of a future petascale system we designed. The prediction was performed on a terascale or sub-terascale system. Therefore, this method has prediction ability for non-existing, higher performance systems by three or four orders of magnitude than existing systems. We conclude that



our method provides a powerful technology option for performance prediction of large-scale systems and applications.

#### ACKNOWLEDGMENTS

This work was supported by Petascale System Interconnect Project in Japan. The computation was mainly carried out using the computer facilities at Research Institute for Information Technology, Kyushu University. A part of computations was performed by using RIKEN Super Combined Cluster.

#### REFERENCES

- [1] V. S. Adve, R. Bagrodia, E. Deelman and R. Sakellariou, "Compiler-optimized simulation of large-scale applications on high performance architectures," *Journal of Parallel and Distributed Computing*, 62(3), pp. 393-426, 2002.
- [2] D. H. Bailey and A. Snaveley, "Performance modeling: understanding the past and predicting the future," In *Proceedings of Euro-Par 2005, Parallel Processing*, 11th International Euro-Par Conference, pp. 185-195.
- [3] J. Bourgeois and F. Spies, "Performance prediction of an nas benchmark program with ChronosMix environment," In *Proceedings of Euro-Par 2000, Parallel Processing*, 6th International Euro-Par Conference, pp. 208-216.
- [4] R. Car and M. Parrinello, "Unified approach for molecular dynamics and density-functional approach," *Phys. Rev. Lett.* 55(22), pp. 2471-2474, 1985..
- [5] J. J. Dongarra and G. W. Stewart, "LINPACK - A package for solving linear systems," *Sources and Development of Mathematical Software*, (W. R. Cowell, Editor), Prentice-Hall Inc., New Jersey, pp. 20-48, 1984
- [6] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman and M. Gittings, "Predictive performance and scalability modeling of a large-scale application," In *Proceedings of SC2001*.
- [7] T. L. Wilmarth, G. Zheng, E. J. Bohm, Y. Mehta, N. Choudhury, P. Jagadishprasad and L. V. Kale, "Performance prediction using simulation of large-scale interconnection networks in POSE," In *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation*, pp. 109-118, 2005.
- [8] S. Prakash and R. L. Bagrodia, "MPI-SIM: using parallel simulation to evaluate MPI programs," In *Proceedings of the 30th conference on Winter simulation*, pp. 467-474, 1998.
- [9] S. Prakash, E. Deelman, R. Bagrodia, "Asynchronous parallel simulation of parallel programs," *IEEE Transactions on Software Engineering*, 26(5), pp. 385-400, 2000.
- [10] A. Snaveley, L. Carrington, N. Wolter, J. Labarta, R. Badia and A. Purkayastha, "A framework for application performance modeling and prediction," In *Proceedings of SC2002*.
- [11] A. Snaveley, X. Gao, C. Lee, L. Carrington, N. Wolter, J. Labarta, J. Gimenez and P. Jones, "Performance modeling of HPC applications," In *Proceedings of Parallel Computing 2003*.
- [12] G. Zheng, G. Kakulapati and L. V. Kale, "BigSim: A parallel simulator for performance prediction of extremely large parallel machines," In *Proceedings of the International Parallel and Distributed Processing Symposium*, 2004.
- [13] G. Zheng, T. Wilmarth, P. Jagadishprasad and L. V. Kale, "Simulation-based performance prediction for large parallel machines," *International Journal of Parallel Programming*, 33(2-3), pp. 183-207, 2005.
- [14] J. Brehm, P. H. Worley and M. Madhukar, "Performance modeling for SPMD message-passing programs," *Concurrency: Practice and Experience*, 10(5), pp. 333-357, 1998