IEICE TRANS. INF. & SYST., VOL.E96–D, NO.7 JULY 2013

PAPER

VACED-SIM: A Simulator for Scalability Prediction in Large-Scale Parallel Computing

Yufei LIN[†,††a)], ***Student Member***, **Xuejun YANG**[†,††], ***Nonmember***, **Xinhai XU**[†,††], ***Member***, ***and*** **Xiaowei GUO**[†,††], ***Nonmember***

SUMMARY Scaling up the system size has been the common approach to achieving high performance in parallel computing. However, designing and implementing a large-scale parallel system can be very costly in terms of money and time. When building a target system, it is desirable to initially build a smaller version by using the processing nodes with the same architecture as those in the target system. This allows us to achieve efficient and scalable prediction by using the smaller system to predict the performance of the target system. Such scalability prediction is critical because it enables system designers to evaluate different design alternatives so that a certain performance goal can be successfully achieved. As the de facto standard for writing parallel applications, MPI is widely used in large-scale parallel computing. By categorizing the discrete event simulation methods for MPI programs and analyzing the characteristics of scalability prediction, we propose a novel simulation method, called *virtual-actual combined execution-driven* (VACED) simulation, to achieve scalable prediction for MPI programs. The basic idea behind is to predict the execution time of an MPI program on a target machine by running it on a smaller system so that we can predict its communication time by virtual simulation and obtain its sequential computation time by actual execution. We introduce a model for the VACED simulation as well as the design and implementation of VACED-SIM, a lightweight simulator based on fine-grained activity and event definitions. We have validated our approach on a sub-system of Tianhe-1A. Our experimental results show that VACED-SIM exhibits higher accuracy and efficiency than MPI-SIM. In particular, for a target system with 1024 cores, the relative errors of VACED-SIM are less than 10% and the slowdowns are close to 1.

key words: *parallel computing, scalability prediction, MPI, communication primitive*

1. Introduction

In the past few decades, scaling up the system size has been the common approach to achieving high performance in parallel computing. The Top500 list [1] shows that the number of cores in a Petascale system has already reached 10^5. Some experts forecast that an Exascale system will consist of more than 10^8 cores in the future [2]. Such large-scale parallel systems each cost millions of dollars and take years to design and implement [3]. When building a target system, system designers usually start with a smaller system with the processing nodes whose architecture is the same as those in the target system in order to predict the performance of the target system [4], [5]. This kind of performance prediction is called *scalability prediction* [6]. Scalability prediction is desirable because it enables system designers to evaluate design alternatives to meet a certain performance goal.

Scalability prediction is a kind of performance prediction. Discrete event simulation is a popular performance prediction method. According to the driving events used, discrete event simulation can be categorized into *trace-driven simulation* and *execution-driven simulation*. In trace-driven simulation [5], [7]–[9], the trace of a program is obtained with an instrumentation tool by running the program first. Then the simulator uses the trace to drive the events and predict the execution time of the program. However, when predicting the performance of a large-scale parallel computing system, the time spent on extracting and the space taken for storing the trace can become intolerable. In addition, due to some uncertain factors in a program (e.g., condition branches, dynamic instruction generations and non-deterministic communications), trace-driven simulation may be inaccurate when an incorrect trace is acquired.

In contrast, execution-driven simulation drives events through executing a program [10]–[13]. In addition to being able to simulate more complex program characteristics such as branch prediction and dynamic instruction generation, execution-driven simulation can also make use of available system resources to directly execute portions of the program and simulate the features that are of specific interest or are unavailable [14]. Therefore, compared to the trace-driven simulation, the execution-driven simulation is closer to the program execution reality and has higher efficiency. However, existing execution-driven simulation methods cannot deal with the scalability prediction problem well, because they do not utilize the characteristics of scalability prediction to gain high accuracy and efficiency.

Nowadays, most large-scale parallel applications are written using MPI, which has become the de facto standard for parallel computing [15]. In this paper, we present the design and implementation of an execution-driven simulator by combining virtual and actual simulation to achieve scalable prediction for MPI programs in large-scale parallel computing. The contributions are summarized as follows:

- We categorize the existing discrete event simulation methods for MPI programs, according to how they predict the performance and drive the events.
- We propose for the first time the idea of conducting

Manuscript received May 18, 2012.
Manuscript revised January 10, 2013.

[†]The authors are with State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha, China.
[††]The authors are with School of Computer Science, National University of Defense Technology, Changsha, China.
a) E-mail: linyufei@nudt.edu.cn

DOI: 10.1587/transinf.E96.D.1430

virtual-actual combined execution-driven (VACED) simulation to achieve scalable prediction for MPI programs and build a model for the VACED simulation.

- We propose for the first time a fine-grained activity and event definition approach, which allows us to design and implement a lightweight VACED simulator, called VACED-SIM, for scalability prediction.
- We demonstrate the effectiveness of VACED-SIM by experimentation using NPB3.3-MPI benchmarks on a sub-system of Tianhe-1A [16]. For a target system with 16 cores, the relative errors of VACED-SIM are less than 6% and the performance speedups are close to linear. Compared to a classic execution-driven simulator (named MPI-SIM) developed by UCLA [12], VACED-SIM exhibits better prediction accuracy and efficiency. For a target system with 1024 cores, the relative errors of VACED-SIM are less than 10% and the slowdowns are close to 1.

The rest of the paper is organized as follows. Section 2 categorizes the discrete event simulation methods for MPI programs and presents the basic idea behind the VACED simulation. Section 3 introduces our VACED-SIM simulator, which is designed and implemented based on the idea of VACED simulation. Section 4 describes our approach for simulating communication activities and calculating the timestamps of the communication end events. Our experimental results are presented and analyzed in Sect. 5. The related work is reviewed in Sect. 6. Section 7 concludes.

2. VACED Simulation for Scalability Prediction

This section describes our VACED simulation approach. Section 2.1 analyzes the characteristics of scalability prediction. Section 2.2 introduces some concepts of discrete event simulation-based performance prediction methods for MPI programs and then categorizes these methods. Section 2.3 proposes the basic idea of VACED simulation for scalability prediction of MPI programs and then builds its simulation model, based on the characteristics of scalability prediction analyzed in Sect. 2.1 and the the performance prediction methods introduced in Sect. 2.2.

2.1 Scalability Prediction

Let us explain a few terms used. By a *target machine*, we mean a large-scale parallel machine that designers plans to build. By a *host machine*, we mean a smaller machine built with the processing nodes whose architecture is the same as those in the target machine. By a *target program*, we mean a program whose performance is to be predicted. Scalability prediction is to predict the performance of a target program on a target machine by using the resources of a host machine, with the following two characteristics:

- First, the architectures of the processing nodes in the host and target machines are the same.
- Second, the host machine has fewer processing nodes than the target machine, which implies that they have different network sizes and configurations.

We are thus motivated to develop a novel prediction method to solve the scalability prediction problem.

2.2 Categorization of the Discrete Event Simulation-based Performance Prediction Methods for MPI Programs

Discrete event simulation [17] is a popular way to solve the performance prediction problem. As a standard library interface for message-passing applications, the Message Passing Interface (MPI) [18] is widely used in high-performance computing. For an MPI program, at any point in execution, a process is either in *sequential computation state* or in *communication state*. So there are four kinds of events during the simulation of MPI program:

- Simulation start event: starts the simulation of a process.
- Simulation end event: ends the simulation of a process.
- Communication start event: changes the state of a process from the sequential computation state to the communication state.
- Communication end event: changes the state of a process from the communication state to the sequential computation state.

In a process, the operations between two adjacent events are referred to as *activities*. The activities in an MPI program can be categorized into *sequential computation activities* and *communication activities*. A sequential computation activity, which performs sequential computation operations that are irrelevant to communication, corresponds to the sequential computation state. A communication activity, which performs operations that are relevant to communication, corresponds to the communication state. A process alternately executes the sequential computation activities and communication activities. Figure 1 shows the relationships among the events and activities in a process.

If we can predict the execution time of all the activities in a process, we can calculate the occurrence time of each event and predict the execution time of the process. When simulating an activity, we call the simulation of this activity an *actual* simulation if the following conditions are satisfied. First, the way in which the activity executes on the host machine is exactly the same as that on the target machine. Second, the execution time is obtained by measurement during its execution on the host machine. Otherwise, the simulation of this activity is referred to as a *virtual* simulation.

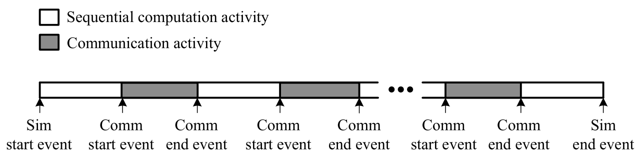

Fig. 1 The relationship between events and activities.

Table 1 Categorization of the discrete event simulation-based performance prediction methods for MPI programs.

	Pure-virtual	Virtual-actual combined
Trace-driven	Dip [7] PERC [8]	PHANTOM [5]
Execution-driven	BigSim [13] MPI-SIM [12]	—

Generally speaking, actual simulation is more accurate than virtual simulation. A simulation method is said to be *pure-virtual* if it only contains virtual simulation and *virtual-actual combined* if it also contains actual simulation. Note that pure-actual simulation does not exist because the target machine has not been completely built yet.

As stated in Sect. 1, there are two ways to drive the events: trace-driven and execution-driven. Therefore, we can categorize the simulation-based performance prediction methods for MPI programs into pure-virtual trace-driven (PVTD), pure-virtual execution-driven (PVED), virtual-actual combined trace-driven (VACTD) and virtual-actual combined execution-driven (VACED) simulations. Table 1 lists some typical simulation methods for MPI programs, which will be discussed in Sect. 6. To the best of our knowledge, predicting the performance of MPI programs by VACED simulation has not been proposed yet.

2.3 The VACED Simulation

The basic ideas behind the VACED simulation are:

Execution-Driven Execute a target program on the host machine so that events are invoked during execution;

Actual Simulation for Sequential Computation Measure the execution time of each sequential computation activity when the target program executes on the host machine, since the first characteristic of scalability prediction ensures that the actual simulation is accurate;

Virtual Simulation for Communication Calculate the execution time of each communication activity on the target machine based on the network information of the target machine, according to the second characteristic of scalability prediction.

Below we present a model for the VACED simulation to achieve scalable prediction for MPI programs.

Without loss of generality, we assume that the number of processes in the target program is equal to the number of cores in the target machine, denoted as P. The simulation for the i^{th} ($i = 0, 1, \dots, P-1$) process of the target program is associated with $n_i + 1$ events. We use the term $E_{i,j}$ ($j = 1, 2, \dots, n_i + 1$) to refer to the j^{th} event of the i^{th} process, where $E_{i,1}$ and E_{i,n_i+1} are the simulation start event and simulation end event, respectively, and we use $A_{i,j}$ ($j = 1, 2, \dots, n_i$) to refer to the j^{th} activity of the i^{th} process.

Let $t(E_{i,j})$ be the timestamp [17] of event $E_{i,j}$, i.e., the time when $E_{i,j}$ happens on the target machine, where $t(E_{i,1}) = 0$. $T_T(A_{i,j})$ and $T_H(A_{i,j})$ represent the execution time of activity $A_{i,j}$ on the target and host machines, respectively. Since $A_{i,j}$ occurs between $E_{i,j}$ and $E_{i,j+1}$, we have:

$$t(E_{i,j+1}) = t(E_{i,j}) + T_T(A_{i,j}) \tag{1}$$

In the VACED simulation for scalability prediction of MPI programs, when a target MPI program is executed on the host machine, the events of each process are invoked and the execution time $T_T(A_{i,j})$ of each activity $A_{i,j}$ is acquired by measurement (actual simulation) or virtual simulation:

- If $A_{i,j}$ is a sequential computation activity, then

  $$T_T(A_{i,j}) = T_H(A_{i,j}) \tag{2}$$

  where $T_H(A_{i,j})$ is measured when the target program is executed on the host machine.
- If $A_{i,j}$ is a communication activity, $T_T(A_{i,j})$ is derived by the virtual simulation of $A_{i,j}$, and the details of virtual simulation will be discussed in Sect. 4.

Due to $t(E_{i,1}) = 0$ and Eq. (1), we can obtain the timestamps of all the events in process i, including $t(E_{i,n_i+1})$, which is the timestamp of the simulation end event and represents the execution time of process i on the target machine. Then $\max_{i=0}^{P-1} t(E_{i,n_i+1})$ is the execution time of the target program on the target machine, i.e., the prediction result.

3. The VACED-SIM Simulator

This section introduces our VACED-SIM simulator, which is fine-grained (Sect. 3.1) and lightweight (Sect. 3.2). We deal with the events by modifying an MPI library and implement the execution-driven simulation by executing a target program on the host machine with the modified MPI library.

3.1 Fine-Grained Activities and Events

To design a simulator using discrete event simulation, the activities of a process and its corresponding events should be defined first.

An MPI library provides the programmers with point-to-point and collective communication primitives. The existing simulators, such as MPI-SIM [12], SIM-MPI [9], define the communication activities at the granularity of MPI communication primitives; they choose some typical point-to-point communication primitives from the MPI library and use them to re-implement the other communication primitives in the library. Then by simulating those chosen primitives, the time overheads of all the communication primitives can be predicted [9], [19]. However, in existing MPI libraries, communication primitives are not implemented by point-to-point communication primitives. Therefore, those existing prediction methods, which need to re-implement the communication primitives, require lots of modifications to an MPI library. In addition, changing the ways in which the communication primitives are implemented in an MPI library also reduces prediction accuracy.

Therefore, in this section, we first analyze how the communication primitives are implemented in existing MPI libraries (Sect. 3.1.1), and then propose a method to define the activities and events at a finer granularity, which respects the ways in which the communication primitives are implemented in existing MPI libraries (Sect. 3.1.2).

3.1.1 Analyzing Point-to-Point Communication

In existing MPI libraries, both point-to-point and collective communication primitives are implemented by point-to-point communications. Point-to-point communications switch communication protocols according to the size of the data to be transferred. There are two major protocols, *eager* and *rendezvous* [20]. In the eager protocol, the data message is immediately sent to the buffer of the receiver directly, as shown in Fig. 2 (a). In the rendezvous protocol, a handshake happens between the sender (the process sending data) and the receiver (the process receiving data) via REQ and ACK messages before the data message is sent to the receiver, as shown in Fig. 2 (b). Therefore, there are three types of messages implemented in MPI libraries, REQ, ACK and DATA.

For a point-to-point communication, denoted as c, both the sender and receiver complete the sending and receiving operations in two phases: *starting phase* and *waiting phase*. It is noteworthy that the waiting phase of c does not always follow the starting phase of c immediately, and either the sequential computation or some other point-to-point communication's starting/waiting phase can appear between c's starting phase and waiting phase. The detailed operations of the starting and waiting phases are as follows:

- The starting phase is used to start c. The sender starts c by sending the DATA message in the eager protocol or sending the REQ message in the rendezvous protocol. For the receiver, there are two cases. If the receiver can find the message related to c in an *unexpected queue*, which is used to store the messages that arrive before their corresponding starting phases begin, it will start c by receiving the DATA message in the eager protocol or sending the ACK message in the rendezvous protocol. Otherwise, the receiver starts c by registering some information without sending or receiving any message.
- The waiting phase is used to guarantee the completion of c. Both the sender and receiver must wait until c is completed (for the sender, c is completed when the DATA message has been sent; for the receiver, c is completed when the DATA message has been received). Furthermore, in the MPI implementation, a message will only be stored in a *message buffer*, which is maintained by an MPI library and directly accessed by hardware, when arriving at its destination. So when c has not been completed, the waiting phase takes the initiative to progress c by accessing and dealing with the messages in the message buffer. The operations in the waiting phase are described in Algorithm 1.

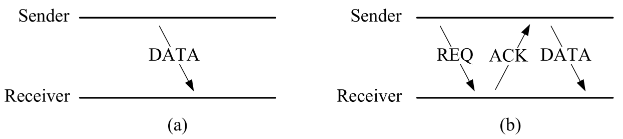

Fig. 2 Communication protocols: (a) eager protocol; (b) rendezvous protocol.

Algorithm 1 Operations of waiting phase
```
1: while c is not completed do // c is the point-to-point communication,
   which this waiting phase belongs to
2:   Access the message buffer in arrival order and get the first message
     which has not been accessed, denoted as m;
3:   if c′ has began its starting phase then // c′ is the point-to-point com-
     munication, which m is related to
4:     if m.type = REQ then
5:       Send the ACK message;
6:     else if m.type = ACK then
7:       Send the DATA message;
8:     else if m.type = DATA then
9:       Receive the DATA message; // Copy the data from the message
         buffer to the buffer specified by MPI primitive
10:    end if
11:  else
12:    Add m into the unexpected queue;
13:  end if
14: end while
```

3.1.2 Defining Fine-Grained Activities and Events

Based on the analysis described in Sect. 3.1.1, we divide all the communication operations in an MPI library into five fine-grained activities: DATA-sending activity and REQ-sending activity for the starting phase of the sender; DATA-receiving activity and ACK-sending activity for the starting phase of the receiver; and waiting activity for the waiting phase. All the activities involved in VACED-SIM are listed in Table 2, with the five items in the middle row being communication activities.

It is noteworthy that the communication activities listed in Table 2 are implemented in the MPI library code and invoked by MPI primitives. However, in an MPI library, apart from the code used to implement these communication activities, there is also a great deal of code used to perform some prerequisite operations. These prerequisite operations do not influence the execution times of communication activities, and the execution times of these prerequisite operations are only related to the architecture of a local processing node. Therefore, VACED-SIM simulates these operations as sequential computation activities by actual simulation, thereby improving the simulation accuracy.

Figure 3 gives an example of standard non-blocking point-to-point communication in the rendezvous protocol. When the sender encounters the MPI_Isend, the starting phase of the sender begins and invokes A_{RS} to send a REQ message. When the REQ message arrives at the receiver, it will be stored in the message buffer. In this example, no waiting phase takes the initiative to access the REQ message

Table 2 Activities simulated in VACED-SIM.

Symbol	Activity	Operations
A_{DS}	DATA-sending activity	Send the DATA message in eager protocol in starting phase
A_{RS}	REQ-sending activity	Send the REQ message in rendezvous protocol in starting phase
A_{DR}	DATA-receiving activity	Receive the DATA message in eager protocol in starting phase
A_{AS}	ACK-sending activity	Send the ACK message in rendezvous protocol in starting phase
A_{WT}	Waiting activity	Operations in waiting phase, as shown in Algorithm 1
A_{SC}	Sequential computation activity	Compute the sequential codes

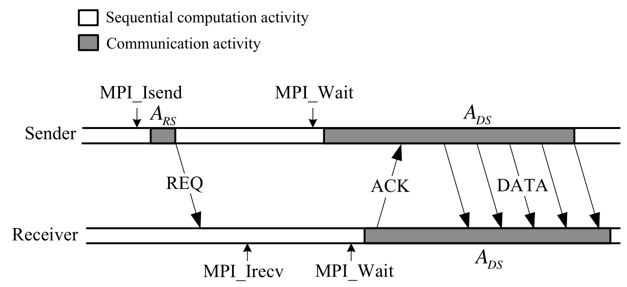

Fig. 3 An example of standard non-blocking communication.

from the message buffer before the MPI_Irecv. So when the receiver encounters the MPI_Irecv, the REQ message has not been added into the unexpected queue and then the starting phase of the receiver just registers some information. When the sender encounters the MPI_Wait, the waiting phase of the sender begins. Because the DATA message has not been sent, it takes the initiative to progress c by accessing and dealing with the messages in the message buffer. Concretely, the sender sends the DATA message after accessing the ACK message in the message buffer. When the receiver encounters the MPI_Wait, the waiting phase of the receiver begins. Because the DATA message has not been received, it also takes the initiative to progress c by accessing and dealing with the messages in the message buffer. Concretely, the receiver sends the ACK message after accessing the REQ message in the message buffer and receives the DATA message from the message buffer after having accessed the DATA message.

Based on the activities listed in Table 2, VACED-SIM includes 12 types of events listed as follows:

- Simulation start event E_{Start}
- Simulation end event E_{End}
- 5 types of communication start events:
  - DATA-sending start event E^{DS}_{start}
  - REQ-sending start event E^{RS}_{start}
  - DATA-receiving start event E^{DR}_{start}
  - ACK-sending start event E^{AS}_{start}
  - Waiting start event E^{WT}_{start}
- 5 types of communication end events:
  - DATA-sending end event E^{DS}_{end}
  - REQ-sending end event E^{RS}_{end}
  - DATA-receiving end event E^{DR}_{end}
  - ACK-sending end event E^{AS}_{end}
  - Waiting end event E^{WT}_{end}

The first executable statement in an MPI process is MPI_Init and the last one is MPI_Finalize. So the execution time of a process that we are concerned with is the time period spanning between MPI_Init and MPI_Finalize. Therefore, the simulation start event E_{Start} (simulation end event E_{End}) happen after the MPI_Init (before the MPI_Finalize) statement. The communication start (end) event happens before (after) the corresponding communication activity.

3.2 Lightweight Driving and Simulation

Based on the fine-grained activities and events defined in Sect. 3.1, VASED-SIM is used to perform a lightweight execution-driven simulation, by simply modifying an existing MPI library and integrating all the functions of VACED-SIM into the modified MPI library. The guiding principles used when modifying the MPI library are as follows:

- Do not modify the code corresponding to sequential computation activities.
- Make modifications as few as possible to the codes corresponding to the 5 kinds of communication activities in an MPI library to implement the virtual simulations.
- Add codes at the places, where the 12 types of events occur, to compute their timestamps.

In order to calculate the timestamp of each event, VACED-SIM adds three private variables for each process: t_V for the simulation time; t'_1 for the process CPU time when the latest sequential computation activity starts on the host machine; t'_2 for the process CPU time when the latest sequential computation activity finishes on the host machine.

The core idea of VACED-SIM lies in updating t_V when a process executes. When a process encounters event e, t_V will be updated with the timestamp of e, denoted as $t(e)$, which refers to the time when e happens on the target machine. For different events, t_V is updated in different ways.

Simulation start event E_{Start} Initialize t_V to 0 and then update t'_1 with the current process CPU time.

Communication start event E^*_{start} Update t'_2 with the current process CPU time and then calculate the value of t_V, based on Eq. (1) and Eq. (2), as follows:

$$t_V = t(E^*_{start}) = t_V + (t'_2 - t'_1)$$

Communication end event E^*_{end} Calculate the value of $t(E^*_{end})$ by simulating the corresponding communication activity virtually, and update t_V with $t(E^*_{end})$ and update t'_1 with the current process CPU time. Note that the way that $t(E^*_{end})$ is calculated is related to the virtual simulation of the corresponding communication activity, which will be discussed in Sect. 4.

Simulation end event E_{End} Update t'_2 with the current process CPU time and then $t_V = t_V + t'_2 - t'_1$.

Based on the modifications to an MPI library, VACED-SIM works by executing a target program on the host machine with the modified MPI library. The code added into the library can drive the simulator to simulate sequential computation activities actually and communication activities virtually. After E_{End} for a process is finished, t_V represents the execution time of the process on the target machine. When all the processes of the target program finish their E_{End}s, the maximum of all the t_Vs represents the execution time of the target program on the target machine, i.e., the prediction result obtained by VACED-SIM.

3.3 Synchronization Mechanisms

There is a common problem shared by all discrete event simulation methods for MPI programs. For the communications in non-deterministic mode (e.g., a receive request contains MPI_ANY_SOURCE as the source), the simulator relies on a synchronization mechanism to make sure that the right messages (i.e., the messages that are received when the program executes on the target machine) are accepted during the simulation on the host machine.

There are two kinds of synchronization mechanisms: *conservative* and *optimistic*. If a conservative mechanism is used, all events are strictly processed in the chronological order. Two protocols have been commonly used: *synchronous* and *asynchronous*. With the synchronous protocol (also called the quantum protocol) [21], after a previously determined interval, every process executes a global barrier. With the asynchronous protocol, when a process encounters a non-deterministic receive, EIT [22], which represents a lower bound on the receive timestamps of future messages, will be calculated. A receive timestamp for a message is defined as the time when the message arrives at the receiver on the target machine. A process only accepts a message if its receive timestamp is smaller than EIT.

An optimistic mechanism [23], [24] allows the earliest available event to be processed with no regard for safety. When an older message arrives, a rollback mechanism is needed to undo earlier out-of-order executed events and re-execute these events in order to ensure correctness.

In VACED-SIM, an asynchronous conservative mechanism is preferred for two reasons. First, the overheads incurred by an optimistic mechanism in saving the program state, rollback and re-execution are unacceptable for large-scale parallel computing. Second, the asynchronous conservative protocol, which only gets switched on when a non-deterministic receive is encountered, is more efficient than the synchronous conservative protocol. Note that synchronization mechanisms are not the focus of this work. The interested readers are referred to [19], [25] for more details.

4. Virtual Simulation of Communication Activities

In this section, we introduce the virtual simulations of communication activities and explain how to calculate the timestamps of communication end events.

4.1 Terminology and Assumptions

When studying communications, researchers always make assumptions on whether there are message buffers at the sending and receiving ends and whether DMA (direct memory access) and asynchronous message transfer using network interface hardware are supported or not [26], [27]. According to the state-of-the-art parallel machines, we assume that both the sending and receiving ends have message buffers, both DMA and asynchronous message transfer using network interface hardware are supported, and the MPI library we modify is a standard MPI library in which the rendezvous protocol does not support zero-copy data transfer [28]. In order to clearly describe the virtual simulations of communication activities in VACED-SIM, we use a *user buffer* to refer to the buffer specified by an MPI primitive. Both the message buffer and user buffer are in the memory.

For a message m, $m.size$ stands for the size of m and $m.sID$ and $m.rID$ stand for the process IDs of the sender and receiver, respectively. Because both the sending and receiving ends have message buffers, the procedure of sending a message is divided into two steps. In the first step, the message is sent from the user buffer to the message buffer at the sending end. In the second step, the message is sent from the message buffer at the sending end to the message buffer at the receiving end. Note that after the first step, the sender can move on to perform computations. Consequently, for a message m, we define three time parameters: $t_s(m)$ represents the time when the sender starts to send m; $t_r(m)$ represents the time when the first step of sending m finishes; and $t_a(m)$ represents the arrival time, i.e., the time when m completely arrives at the receiver's message buffer. $t_s(m)$, $t_r(m)$ and $t_a(m)$ satisfy the following equation:

$$\begin{aligned} t_r(m) &= t_s(m) + m.size/B_{mem} \\ t_a(m) &= t_r(m) + L(m.size, hop(m.sID, m.rID)) \end{aligned} \tag{3}$$

where B_{mem} is the memory bandwidth, $hop(x, y)$ is the number of hops traversed by a message from process x to process y on the target machine, and $L(s, h)$ is the time overhead of transferring the message of size s from the message buffer at the sending end to the message buffer at the receiving end across h hops. $hop(x, y)$ depends on the network topology of the target machine and how a target program is mapped onto the target machine. $L(s, h)$ is defined to be:

$$L(s,h) = \begin{cases} s/B_{net} + l(h), & h > 0 \\ s/B_{mem}, & h = 0 \end{cases} \tag{4}$$

where B_{net} is the network bandwidth and $l(h)$ is the network latency of an h-hop transmission on the target machine. VACED-SIM does not presently model communication congestion, but it can be easily extended to do so by integrating with it an existing congestion model.

4.2 Virtual Simulation of A_{DS}, A_{RS} and A_{AS}

As shown in Table 2, the operations of A_{DS}, A_{RS} and A_{AS} are concerned with sending the corresponding messages, denoted as m. However, the time overheads of transferring m on the target machine and on the host machine may be different. In order to make sure that the receiver knows $t_a(m)$ during the simulation, VACED-SIM simulates A_{DS}, A_{RS} and A_{AS} by inserting code to calculate $t_a(m)$ and sending $t_a(m)$ with m together. Since t_V has been updated by E^{DS}_{start}, E^{RS}_{start} and E^{AS}_{start}, respectively, before simulating A_{DS}, A_{RS} and A_{AS}, the equation for calculating $t_a(m)$ is defined by:

$$t_a(m) = t_V + m.size/B_{mem} + L(m.size, hop(m.sIDr, m.rID))$$

The sender can proceed to perform its own computations after it has sent a message in the first step. Therefore, t_V has been updated by E^{DS}_{end}, E^{RS}_{end} and E^{AS}_{end} at the end of A_{DS}, A_{RS} and A_{AS}, respectively:

$$t_V = t_r(m) = t_V + m.size/B_{mem}$$

4.3 Virtual Simulation of A_{DR}

The operations of A_{DR} are concerned with receiving the DATA message m; they are implemented by transferring the data from the message buffer to the user buffer. VACED-SIM simulates A_{DR} without modifying any code in the original MPI library, i.e., A_{DR}s in the original MPI library and in the modified MPI library are the same. Because both the message buffer and user buffer are in the memory, E^{DR}_{end}, which is at the end of A_{DR}, updates t_V by:

$$t_V = t_V + m.size/B_{mem}$$

4.4 Virtual Simulation of A_{WT}

There are some difficulties in performing virtual simulation for A_{WT}. First, the time when a message arrives on the target machine may be different from that on the host machine. Second, A_{WT} may take the initiative to progress the point-to-point communication by handling the messages in the message buffer. For example, Fig. 4 (a) shows the arrival of message m on the target machine when m is processed by the second A_{WT}. When the target program runs on the host machine, m may arrive before the first A_{WT} (as shown in Fig. 4 (b)) or after the second A_{WT} (as shown in Fig. 4 (c)). These two cases may induce an incorrect A_{WT} (i.e., the first or the third rather than the second A_{WT}) to deal with m.

It should be clarified that the other existing simulation methods, which define the activities and events at the granularity of MPI primitives, cannot even reveal, let alone solve the above problems. But our fine-grained activity and

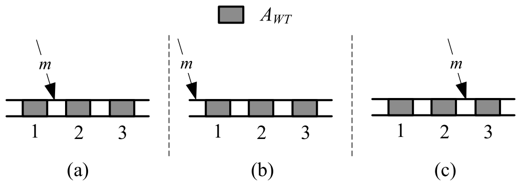

Fig. 4 Problems in simulating A_{WT}: (a) message arrival on the target machine; (b) the first problem of message arrival on the host machine; (c) the second problem of message arrival on the host machine.

event definitions provide us with opportunities to discover and address these problems effectively. If simulation costs are ignored, we can use the synchronization mechanisms described in Sect. 3.3 to solve the problems successfully.

However, we aim to develop a lightweight simulator. To this end, we have designed an *early queue* for each process in VACED-SIM to store the messages to deal with the problem illustrated in Fig. 4 (b); In addition, in order to address the problem illustrated in Fig. 4 (c), we also define *num* to count the number of A_{WT}s that have been simulated, and record the start time and end time of all the A_{WT}s in $AWT_{start}[]$ and $AWT_{end}[]$, respectively, for each process.

Our algorithm for performing virtual simulation of A_{WT} is given in Algorithm 2. If c has not completed when the waiting phase begins, we progress the communication by accessing the messages in the early queue (lines 5-10). We then progress the communication by accessing the mes-

Algorithm 2 Virtual simulation of A_{WT}

```
1: $AWT_{start}[num] \leftarrow t_V$; // Currently, $t_V = t(E^{WT}_{start})$
2: if $c$ is completed then // $c$ is the point-to-point communication, which
   this waiting phase belongs to
3:     $AWT_{end}[num] \leftarrow t_V$;
4: else
5:     for all $m \in$ early queue do
6:         if $t_a(m) \le t_V$ or $m$ is related to $c$ then
7:             DealMessage(c,m);
8:             Delete $m$ from the early queue;
9:         end if
10:    end for
11:    while $c$ is not completed do
12:        Access the message buffer in arrival order and get the first mes-
           sage which has not been accessed, denoted as $m$;
13:        if $t_a(m) > t_V$ and $m$ is not related to $c$ then
14:            Add $m$ into the early queue;
15:        else if $c'$ has not began its starting phase then // $c'$ is the point-to-
           point communication, which $m$ is related to
16:            Add $m$ into the unexpected queue;
17:        else
18:            DealMessage(c,m);
19:        end if
20:    end while
21: end if
22: for all $m \in$ early queue do
23:     if $t_a(m) \le AWT_{end}[num]$ then
24:         DealMessage(c,m);
25:         Delete $m$ from the early queue;
26:     end if
27: end for
28: $num \leftarrow num + 1$;
```

sages in the message buffer (lines 11-20). Unlike Algorithm 1, after having retrieved message m from the message buffer, we must decide whether m exhibits the problem as shown in Fig. 4 (b) (lines 13-14). Furthermore, there may be messages that arrive between $AWT_{start}[num]$ and $AWT_{end}[num]$ on the target machine in the early queue after calculating the time of $AWT_{end}[num]$ when c completes. Therefore, we re-traverse the early queue to find out these messages and deal with them (lines 22-27).

DealMessage(c,m), the function used in Algorithm 2, is given in Algorithm 3. We first calculate t_{temp}, the time when m is dealt with on the target machine (lines 1-13) and then deal with m. Unlike Algorithm 1, we send the corresponding message with its arrival time together if m is a REQ message or ACK message (lines 14-19). When dealing with an ACK or DATA message, we set $AWT_{end}[num]$ as desired (lines 20-22 and 25-27) if m is related to c.

Algorithm 3 Dealing with the message: *DealMessage(c,m)*

```
1: if $t_a(m) \geq AWT_{start}[num]$ then
2:     $t_{temp} \leftarrow t_a(m)$;
3: else
4:     $i \leftarrow num - 1$;
5:     while $AWT_{start}[i] > t_a(m)$ do
6:         $i \leftarrow i - 1$;
7:     end while
8:     if $AWT_{end}[i] > t_a(m)$ then
9:         $t_{temp} \leftarrow t_a(m)$;
10:    else
11:        $t_{temp} \leftarrow AWT_{start}[i+1]$;
12:    end if
13: end if
14: if $m.type = REQ$ then
15:     Calculate $t_a(m_{ACK})$ by Eq. (3) with $t_s(m_{ACK}) = t_{temp}$;
16:     Send the ACK message $m_{ACK}$ with $t_a(m_{ACK})$ together;
17: else if $m.type = ACK$ then
18:     Calculate $t_a(m_{DATA})$ by Eq. (3) with $t_s(m_{DATA}) = t_{temp}$;
19:     Send the DATA message $m_{DATA}$ with $t_a(m_{DATA})$ together;
20:     if $m$ is related to $c$ then
21:         $AWT_{end}[num] \leftarrow max\{AWT_{start}[num], t_{temp}+m_{DATA}.size/B_{mem}\}$;
22:     end if
23: else if $m.type = DATA$ then
24:     Receive the DATA message;
25:     if $m$ is related to $c$ then
26:         $AWT_{end}[num] \leftarrow max\{AWT_{start}[num], t_{temp}+m_{DATA}.size/B_{mem}\}$;
27:     end if
28: end if
```

Finally, E^{WT}_{end}, which is at the end of A_{WT}, updates t_V by $AWT_{end}[num-1]$, which is obtained by Algorithm 2.

It should be noted that in Algorithms 2 and 3, we assume that there is no interaction between the messages that are dealt with in the same A_{WT} on the target machine, for the convenience of dealing with the problem as shown in Fig. 4 (c). Otherwise, we must use checkpointing methods [29] to rollback the simulation and re-calculate the time overheads and timestamps of the corresponding A_{WT}. Our experiments discussed below show that this assumption is acceptable in the sense that VACED-SIM can still achieve high accuracy even under this assumption.

5. Experiments

We demonstrate the accuracy and efficiency of VACED-SIM in this section. Section 5.1 introduces the benchmarks and experiment platform used. Section 5.2 describes the methodology used for evaluating this work. Section 5.3 presents and analyzes our results.

To facilitate introducing the experiments, we introduce the following notations. $T^{Real}_{x,y}$ denotes the real execution time of an x-process benchmark on a y-core machine by using the original MPI library. $T^{Sim}_{x,y}$ represents the prediction result obtained by VACED-SIM for an x-process benchmark on a y-core host machine. $T^{Exe}_{x,y}$ represents the execution time that VACED-SIM takes to predict the performance of an x-process benchmark on a y-core host machine.

5.1 Benchmarks and Platform

We have selected the NPB3.3-MPI benchmarks to evaluate VACED-SIM. There are five parallel kernels, EP, MG, CG, FT and IS, and three parallel applications, LU, SP and BT. All the benchmarks are executed on a Red Hat Enterprise Linux Server (Release 5.5) with MPICH2-1.3.1 being used.

Our platform is a sub-system of Tianhe-1A, used to be the fastest supercomputer (designed by the National University of Defense Technology) in the world [1]. In TianHe-1A, each processing node is equipped with two 2.93G 6-core Intel Xeon X5670 CPUs with 24 GB RAM. The interconnection is the same as described in [30]: 16 processing nodes are connected with a switching board, and switching boards compose a fat-tree through routers.

5.2 Evaluation Methodology

We have implemented VACED-SIM by modifying MPICH2-1.3.1 according to the techniques described in Sect. 3 and Sect. 4. The parameters discussed in Sect. 4.1 are determined by using a ping-pong program: $B_{mem} = 6.38\,GB/s$, $B_{net} = 4.16\,GB/s$ and $l(h) = 15us + h \times 0.6us$.

We compare the accuracy and performance of VACED-SIM with MPI-SIM [12], which also requires the same processor configurations in the host and target machines. A detailed description of MPI-SIM is given in Sect. 6. The CPU used in MPI-SIM is a single-core processor. So we have used Xeon X5670 CPUs during these experiments: the processes allocated in the same CPU are bound to the same core. In our experiments, we use 16 CPUs connected by the same switching board as the target machine with $hop(x,y) = 3$. We vary the core count N for the host machine and measure the relative error $RE(16,N) = T^{Sim}_{16,N}/T^{Real}_{16,16} - 1$ and the performance speedup $S(16,N) = T^{Exe}_{16,N}/T^{Exe}_{16,1}$ of VACED-SIM. We then compare VACED-SIM with MPI-SIM.

In order to validate the accuracy and performance of VACED-SIM for large-scale parallel computing, we fix the core count N of the target machine to be 1024 and the problem size of the target program to be class C and then vary N to measure the relative error $RE(1024, N) = T^{Sim}_{1024,N}/T^{Real}_{1024,1024} - 1$ and the slowdown [31] $SD(1024, N) = T^{Exe}_{1024,N}/T^{Real}_{1024,N}$ of VACED-SIM. In these tests, all the six cores in a Xeon X5670 CPU are used. In addition, we have:

$$hop(x,y) = \begin{cases} 0, & if(x/12 = y/12) \\ 3, & if(x/12 \neq y/12, x/192 = y/192) \\ 5, & otherwise \end{cases} \tag{5}$$

5.3 Results and Analysis

5.3.1 VACED-SIM vs. MPI-SIM

To compare VACED-SIM with MPI-SIM, we choose the same benchmarks with those used in MPI-SIM [12]: LU (Class A), MG (Class S), BT (Class S) and SP (Class S). Figure 5 shows the relative errors of VACED-SIM when predicting the execution time of 16-process target programs. The absolute values of the relative errors are less than 6% by using VACED-SIM. In VACED-SIM, the errors of prediction results are mainly affected by two communication-related aspects. First, communication activities cause the process switching in the cores of the host machine, leading to the inaccurate simulation of sequential computation activities. For example, the sequential computation times may be over-estimated due to the cache thrashing. Second, the problem discussed in Sect. 4.4 will arise during the virtual simulation of communication. Algorithm 2 and 3 have solved well the problem illustrated in Fig. 4 (b). However, for the problem shown in Fig. 4 (c), we assume that there is no interaction between the messages dealt with in the same A_{WT} on the target machine rather than using checkpointing to totally solve it. While critical in making our simulator lightweight, this assumption will cause the predicted execution time to be smaller than the real one. Which of these two aspects affects prediction accuracy more varies from program to program. This is why the four benchmarks in Fig. 5 show different relative errors. Generally speaking, $RE(16, 16)$ is small, because the host machine has the same scale as the target machine. $RE(16, 16)$ for MG, shown in Fig. 5, is –5%, because its execution time (in Class S) on 16 cores is small. However, compared to MPI-SIM [12], whose relative errors are 5%–20% for the above benchmarks, VACED-SIM is much more accurate.

Figure 6 shows the performance speedups of VACED-SIM, by varying the core counts of the target machine. It is easy to observe that for all the four benchmarks, the performance speedups of VACED-SIM are close to linear, which are better than MPI-SIM. When $N = 16$, the speedup $S(16, 16)$ is as large as 39.13. That is because when there are as many cores in the host machine as the number of processes in the target program, VACED-SIM runs with one process per core, with no process switching.

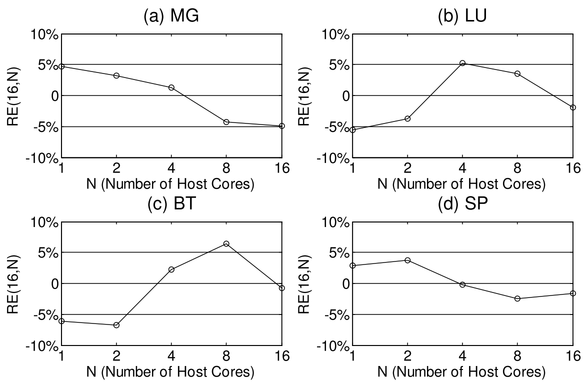

Fig. 5 The relative errors of VACED-SIM for 16-process target programs: $RE(16, N) = T^{Sim}_{16,N}/T^{Real}_{16,16} - 1$.

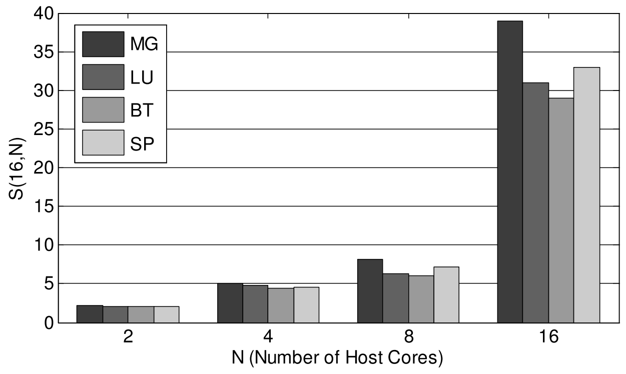

Fig. 6 The performance speedups of VACED-SIM: $S(16, N) = T^{Exe}_{16,N}/T^{Exe}_{16,1}$.

Table 3 The real execution time of 1024-process target programs on different machine configurations (s).

	$T^{Real}_{1024,1024}$	$T^{Real}_{1024,512}$	$T^{Real}_{1024,256}$	$T^{Real}_{1024,128}$
CG	62.09	476.68	2343.10	8932.46
IS	272.13	52.97	173.30	625.63
EP	0.48	1.51	7.57	26.25
MG	9.45	52.30	262.28	1034.15
FT	16.64	73.18	346.26	1327.51
SP	34.87	900.56	4694.97	16995.70
BT	20.87	453.88	2357.30	8553.26
LU	47.19	599.30	3255.13	11853.90
Avg.	57.96	326.30	1679.99	6168.61

5.3.2 Prediction for Large-Scale Computing

To validate the accuracy and efficiency of VACED-SIM for large-scale parallel computing, we choose all the NPB benchmarks (Class C) with 1024 processes in the target programs. The real execution times of these target programs on different machines are listed in Table 3. It is easy to find that $T^{Real}_{1024,N}$ increases greatly as N decreases. There are two reasons. First, as the hardware parallelism decreases, each core's computation time increases. Second, binding several

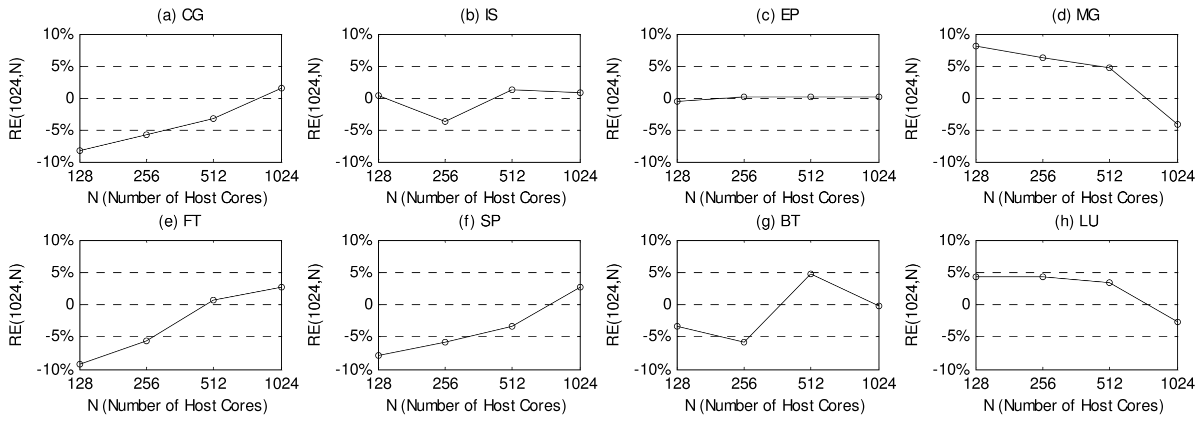

Fig. 7 The relative errors of VACED-SIM for 1024-process target programs: $RE(1024, N) = T^{Sim}_{1024,N}/T^{Real}_{1024,1024} - 1$.

Table 4 Slowdowns of VACED-SIM: $SD(1024, N) = T^{Exe}_{1024,N}/T^{Real}_{1024,N}$.

	CG	IS	EP	MG	FT	SP	BT	LU	Avg.
$SD(1024, 128)$	1.00	1.00	1.02	1.00	1.00	0.99	1.00	0.99	1.00
$SD(1024, 256)$	1.01	1.00	1.01	1.01	1.00	1.00	0.99	0.98	1.00
$SD(1024, 512)$	1.00	1.03	0.99	1.01	1.01	0.99	1.00	1.02	1.01
$SD(1024, 1024)$	1.02	1.01	1.05	0.99	1.02	1.02	0.98	1.02	1.02

processes onto the same core increases the communication and synchronization costs. It should be pointed out that for IS, $T^{Real}_{1024,1024} > T^{Real}_{1024,512}$. This is because IS makes use of lots of collective communications, with their costs increasing rapidly as the interconnection size scales up.

From Table 3, we observe that for the NPB benchmarks, the average values of $T^{Real}_{1024,128}/T^{Real}_{1024,1024}$ are more than 100. The large difference between the sizes of the target and host machines leads to extremely long execution times for the target programs on the host machine. Therefore, the trace-driven simulation method, which relies on extracting the trace of a target program by executing it several times, may result in intolerable time overhead.

Figure 7 illustrates the results of the scalability prediction for the above benchmarks on the 1024-core target machine by using the host machines with different sizes. The accuracy of VACED-SIM for the scalability prediction of large-scale parallel computing system, whose parallelism degree reaches 1024, increases as the size of the host machine increases. VACED-SIM has high accuracy with the average values of relative errors being less than 10%, even when the size of target machine is 8 times larger than the host machine (i.e., when the host machine only contains 128 cores). For the same reasons illustrated in Sect. 5.3.1, the prediction errors of the eight benchmarks show different characteristics. From Fig. 7, we can find that as the scale of the host machine (i.e. N) goes up, the prediction becomes more and more accurate (i.e., $|RE(1024, N)|$ decreases). Furthermore, for the benchmarks with little communication, such as EP, VACED-SIM shows higher accuracy, which is close to 0. That is because when predicting the performance of EP, all simulations performed in VACED-SIM are actual.

The slowdowns of VACED-SIM are shown in Table 4. All the functions of VACED-SIM are integrated into the modified MPI library and the additional codes are used to calculate the timestamps of events. Therefore, the slowdowns of VACED-SIM are close to 1, as listed in Table 4.

6. Related Work

MPI is widely used as the de facto standard for writing parallel applications. Performance prediction simulators for MPI programs are highly needed for finding the bottlenecks of a system at its design phase.

Dip [7] is a PVTD simulator according to our categorization. For MPI programs, Dip uses a set of trace files generated by linking a program with the instrumented MPI library to reconstruct the behavior of a parallel program. Sequential computation time is acquired by using the architecture model of the target machine. Communication time is computed using a simple latency-bandwidth model, which ignores the influence of distance between processors.

PERC [8] is a performance prediction framework using PVTD simulation. It is conducted based on a simplifying hypothesis that a parallel application's performance is often dominated by its single processor performance and its use of the network. To model single-processor performance, it separates various performance factors by measuring each in isolation and integrates them for a model of overall performance, and then combines the information derived by the performance model with a existing network simulator.

BigSim [13] is a PVED simulator. Based on the CHARM++ [32] parallel programming system, BigSim defines a set of application interfaces, such as addMessage and sendPacket, which are used to implement the MPI interfaces. All the application interfaces are executed by the simulator, and the other codes are directly executed on the host machine. In BigSim, the sequential computation time is calculated by heuristic approaches, which cannot obtain high accuracy, and the communication time is derived by a no-contention model or a network simulator known as BigNetSim [33].

MPI-SIM [12] is a PVED simulator developed by UCLA. Just as in VACED-SIM, the processor configurations in the host and target machines must be similar. MPI-SIM uses a portable library MPI-LITE to translate a target program into a multithreaded program and measures the execution time of sequential computation codes when the multithreaded program is executing. As cache replacement policy and execution pattern (among others) are different between threads and processes, sequential computation time cannot be measured accurately. For the communication simulation, the developers re-implement all the collective communication functions and point-to-point communication functions by using a set of core functions, MPI_Issend, MPI_Ibsend, MPI_Irecv and MPI_Wait. The simulation is performed at a coarse granularity according to our analysis in Sect. 3.1 and the re-implementation changes the behaviors of the communications implemented in the MPI library, both of which reduce prediction accuracy.

It is worth mentioning that Susukita et al. proposed a macro-level simulation method [34], which executes a skeleton in an environment (BSIM) for predicting parallel performance. The skeleton contains the execution time of each sequential computation activity, i.e., sequential performance, and the execution trace of the program. After the target application is analyzed, the sequential calculation blocks are abstracted as macros, whose execution times are predicted independently by processor simulations or real executions, so that the simulation efficiency is improved. However, BSIM simulates the communication at the granularity of MPI primitives, such as MPI_Send and MPI_Recv and assumes that send operations are always asynchronous. This kind of simulation cannot take the influence of MPI implementation details into consideration, including, for example, the communication protocol, the implementation of point-to-point communication. Therefore, it will reduce the prediction accuracies of the communication time and the convolution of the computation and communication, and consequently, the prediction accuracy of the whole program.

Furthermore, Zhai et al. [5] proposed a new method, named Phantom to acquire the sequential computation time, which is used in a trace-driven simulator SIM-MPI [9]. The main contribution of Phantom lies in obtaining the sequential computation time more accurately. Phantom uses a deterministic replay technique and thus must execute a target program twice. During the first execution, communication traces of parallel applications are generated by intercepting all communication operations for each process and the computation between communication operations is marked as a sequential computation unit. During the second execution, the real sequential computation time is measured on a target processing node for each process one by one. This technique eliminates the influence of process switching, such as cache thrashing, resulting in sometimes higher accuracy than VACED-SIM. However, executing a target program twice is time-consuming. In addition, it cannot deal with uncertain programs (the programs with condition branches, dynamic instruction generations, non-deterministic communication, etc.) because an inaccurate trace is usually generated.

7. Conclusion

Scalability prediction for MPI programs in large-scale parallel computing is greatly desired by system designers. According to the characteristics of scalability prediction, we propose a novel simulation method, called VACED simulation, to predict the execution time of communication activities by virtual simulation and sequential computation activities by actual simulation. Based on VACED simulation, we have designed and implemented VACED-SIM, a fine-grained and lightweight simulator for scalability prediction of MPI programs. Our experimental results show that VACED-SIM exhibits high accuracy and efficiency. For a target system with 1024 cores, the relative errors of VACED-SIM are less than 10% and the slowdowns are close to 1. In future work, we will further improve the efficiency of VACED-SIM, for example, by using (with extensions) the method proposed by Susukita et al. [34] to reduce the times of simulating sequential computation activities while maintaining and further improving our prediction accuracy.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (NSFC) No.60921062.

References

[1] Website. http://www.top500.org

[2] A. Geist, "Paving the roadmap to exascale," SciDAC Review, NUMBER 16 Special Issue, 2010.

[3] J. Dongarra, P. Beckman, T. Moore, and P. Aerts, "The international exascale software project roadmap," Int. J. High Perform. Comput. Appl., vol.25, no.1, pp.3–60, Feb. 2011.

[4] 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," Proc. 2001 ACM/IEEE Conference on Supercomputing (CDROM), Supercomputing '01, p.37, New York, NY, USA, 2001.

[5] J. Zhai, W. Chen, and W. Zheng, "Phantom: Predicting performance of parallel applications on large-scale parallel machines using a single node," Proc. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP, pp.305–314, Bangalore, India, 2010.

[6] B.J. Barnes, J. Reeves, B. Rountree, B. De Supinski, D.K. Lowenthal, and M. Schulz, "A regression-based approach to scalability prediction," Proc. International Conference on Supercomputing, pp.368–377, Island of Kos, Greece, 2008.

[7] J. Labarta, S. Girona, V. Pillet, T. Cortes, and L. Gregoris, "Dip: A

parallel program development environment," Lect. Notes Comput. Sci., vol.1124, p.665, 1996.
[8] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, "A framework for performance modeling and prediction," Proc. 2002 ACM/IEEE conference on Supercomputing, Supercomputing '02, pp.1–17, Los Alamitos, CA, USA, 2002.
[9] Website. http://sourceforge.net/projects/sim-mpi/
[10] S.K. Reinhardt, M.D. Hill, J.R. Larus, A.R. Lebeck, J.C. Lewis, and D.A. Wood, "The wisconsin wind tunnel: virtual prototyping of parallel computers," Proc. 1993 ACM SIGMETRICS conference on Measurement and modeling of computer systems, SIGMETRICS '93, pp.48–60, New York, NY, USA, 1993.
[11] S.S. Mukherjee, S.K. Reinhardt, B. Falsafi, M. Litzkow, M.D. Hill, D.A. Wood, S. Huss-Lederman, and J.R. Larus, "Wisconsin wind tunnel ii: A fast, portable parallel architecture simulator," IEEE Concurrency, vol.8, pp.12–20, Oct. 2000.
[12] S. Prakash and R.L. Bagrodia, "Mpi-sim: using parallel simulation to evaluate mpi programs," Proc. 30th conference on Winter simulation, WSC '98, pp.467–474, Los Alamitos, CA, USA, 1998.
[13] G. Zheng, G. Kakulapati, and L. Kale, "Bigsim: a parallel simulator for performance prediction of extremely large parallel machines," 2004, Proc. 18th International, Parallel and Distributed Processing Symposium, p.78, April 2004.
[14] S. Prakash, E. Deelman, and R. Bagrodia, "Asynchronous parallel simulation of parallel programs," IEEE Trans. Softw. Eng., vol.26, no.5, pp.385–400, May 2000.
[15] C. Coarfa, Portable High Performance and Scalability of Partitioned Global Address Space Languages, Ph.D. thesis, RICE University, 2007.
[16] X.J. Yang, X.K. Liao, K. Lu, Q.F. Hu, J.Q. Song, and J.S. Su, "The tianhe-1a supercomputer: Its hardware and software," J. Computer Science and Technology, vol.26, no.3, pp.344–351, 2011.
[17] R.M. Fujimoto, "Parallel discrete event simulation," Commun. ACM, vol.33, pp.30–53, Oct. 1990.
[18] Website. http://www-unix.mcs.anl.gov/mpi
[19] S. Prakash, Performance Prediction of Parallel Programs, Ph.D. thesis, University of California, 1996.
[20] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the mpi message passing interface standard," Parallel Comput., vol.22, pp.789–828, Sept. 1996.
[21] J. Misra, "Distributed discrete-event simulation," ACM Comput. Surv., vol.18, pp.39–65, March 1986.
[22] V. Jha and R.L. Bagrodia, "Transparent implementation of conservative algorithms in parallel simulation languages," Proc. 25th Conference on Winter Simulation, WSC '93, pp.677–686, New York, NY, USA, 1993.
[23] D. Jefferson, B. Beckman, F. Wieland, B.L., and M. Diloreto, "Time warp operating system," SIGOPS Oper. Syst. Rev., vol.21, pp.77–93, Nov. 1987.
[24] J.S. Steinman, "Breathing time warp," SIGSIM Simul. Dig., vol.23, pp.109–118, July 1993.
[25] K. Chandy and J. Misra, "Distributed simulation: A case study in design and verification of distributed programs," IEEE Trans. Softw. Eng., vol.SE-5, no.5, pp.440–452, Sept. 1979.
[26] G.K.A. Grama, A. Gupta, and V. Kumar, Introduction to parallel computing (Second Edition), Pearson Education, 2003.
[27] C.A. Moritz and M.I. Frank, "Logpc: Modeling network contention in message-passing programs," IEEE Trans. Parallel Distrib. Syst., vol.12, pp.404–415, April 2001.
[28] A. Afsahi, Design and Evaluation of Communication Latency Hiding/Reduction Techniques for Message-Passing Environments, Ph.D. thesis, University of Victoria, 2000.
[29] X. Xu, X. Yang, and Y. Lin, "WBC-ALC: A weak blocking coordinated application-level checkpointing for MPI programs," IEICE Trans. Inf. & Syst., vol.E95-D, no.3, pp.786–796, March 2012.
[30] M. Xie, Y. Lu, L. Liu, H. Cao, and X. Yang, "Implementation and evaluation of network interface and message passing services for tianhe-1a supercomputer," Proc. 2011 IEEE 19th Annual Symposium on High Performance Interconnects, HOTI '11, pp.78–86, Washington, DC, USA, 2011.
[31] P. Dickens, P. Heidelberger, and D. Nicol, "Parallelized direct execution simulation of message-passing parallel programs," IEEE Trans. Parallel Distrib. Syst., vol.7, no.10, pp.1090–1105, Oct. 1996.
[32] L.V. Kale and S. Krishnan, "Charm++: A portable concurrent object oriented system based on c++," SIGPLAN Not., vol.28, no.10, pp.91–108, Oct. 1993.
[33] G. Zheng, T. Wilmarth, O. Lawlor, L. Kale, S. Adve, D. Padua, and P. Guebelle, "Performance modeling and programming environments for petaflops computers and the blue gene machine," Proc. 18th International Parallel and Distributed Processing Symposium, 2004, p.197, April 2004.
[34] R. Susukita, H. Ando, M. Aoyagi, H. Honda, Y. Inadomi, K. Inoue, S. Ishizuki, Y. Kimura, H. Komatsu, M. Kurokawa, K.J. Murakami, H. Shibamura, S. Yamamura, and Y. Yu, "Performance prediction of large-scale parallell system and application using macro-level simulation," Proc. 2008 ACM/IEEE Conference on Supercomputing, SC '08, pp.20:1–20:9, Piscataway, NJ, USA, 2008.

Yufei Lin born in 1985. Received her B.S. degree in automation from the University of Science and Technology of China (USTC) in 2006 and M.S. degree in computer science from the National University of Defense Technology (NUDT) in 2008. She is now a Ph.D. candidate in Computer Science from School of Computer Science at NUDT. Her research interest lies in high performance computing and evaluation. Her email address is linyufei@nudt.edu.cn

Xuejun Yang born in 1963. Received his M.S. and Ph.D. degree in computer science from the National University of Defense Technology (NUDT) in 1986 and 1991. He is the chief designer of Tianhe-1A, an Academician of Chinese Academy of Science (CAS) and a professor in the State Key Laboratory of High Performance Computing, NUDT. His research interests include supercomputer architecture, parallel and distributed operating systems, parallel language, and compilers. His email address is xjyang@nudt.edu.cn

Xinhai Xu born in 1984. Received his B.S., M.S. and Ph.D. degrees in computer science from NUDT in 2006, 2008 and 2012. He is now an assistant professor from School of Computer Science at NUDT. His research interest lies in high performance computing and fault tolerance. His email address is xuxinhai@nudt.edu.cn

Xiaowei Guo born in 1986. Received his B.S. and M.S. degree in computer science from the National University of Defense Technology (NUDT) in 2009 and 2011. He is now a Ph.D. candidate in Computer Science from School of Computer Science at NUDT. His research interest lies in high performance computing and its applications. His email address is guoxiaowei@nudt.edu.cn