CCF Transactions on High Performance Computing (2023) 5:442–464
https://doi.org/10.1007/s42514-023-00173-9

REGULAR PAPER

Leveraging simulation of high performance computing systems with node simulation using architecture simulator

Fang Lin[1] · Yi Liu[1] · Xin Wang[1] · Xueyan Gai[1]

Received: 25 July 2023 / Accepted: 25 October 2023 / Published online: 13 November 2023

Abstract
With the scaling-up of high-performance computing (HPC) systems, their simulation becomes more challenging. Realizing that simulation of HPC systems plays an important role in system evaluation and software development, this paper proposes an approach to simulate HPC systems by simulating one node using traditional execution-driven full-system simulators. Our approach incorporates an off-the-shelf architecture simulator with a message emulation environment and an interconnection network simulation module. The architecture simulator is used to model the hardware architecture of the node of the target HPC system, and then to simulate one specified node of the target system by executing MPI processes as well as the operating system in the node simulation instance. The message emulation environment is used to emulate message transportation between the processes in the simulated node and other processes, which is essential to drive the execution of those processes in the node. The interconnection network simulation module assists the message emulation environment in computing the transport latency of messages based on the model of the target interconnection network. By utilizing the powerful modeling and simulation capabilities of architecture simulators, researchers can not only perform detailed and accurate simulation of the node of the target HPC system, but also can simulate the execution of large-scale parallel programs in the target HPC system. Our simulation system is implemented based on the GEM5, and experimental results demonstrate the effectiveness & performance of our approach.

Keywords High-performance computing · Full-system simulator · Execution-driven simulation · Performance evaluation

1 Introduction

Simulator plays a very important role in the design of high-performance computing (HPC) systems as well as the development of HPC software. In the past decades, with the increasing performance of HPC systems, the system scale has grown continuously. In addition to the rapid growth of the system scale, various heterogeneous processors and accelerators (e.g., GPU, FPGA, AI accelerator) have been widely used in HPC systems, which also increases the complexity of the system. The huge-scale heterogeneous HPC systems bring challenges to traditional simulators, including both execution-driven and trace-driven simulation.

The execution-driven methods simulate the target HPC system by executing parallel programs on the modeled target system. Furthermore, a simulator is full-system if it supports the execution of both operating system and applications. The current execution-driven HPC simulators mainly have the following limitations. (1) The scalability is insufficient, and the simulator will eventually become unusable when the scale of the target system and parallel programs reaches a certain level. And the larger the scale of the target system and the parallel program, the more local hosts are needed for simulation, which is a huge consumption of resources. (2) To improve the scalability, some simulators only simulate the workload of the parallel program and do not simulate the execution of the operating system, which cannot reflect the interaction between the parallel program and the software environment(e.g., operating system and MPI library, etc.).

✉ Fang Lin
fanglinjsi@buaa.edu.cn

Yi Liu
yi.liu@buaa.edu.cn

Xin Wang
qaqxwang@buaa.edu.cn

Xueyan Gai
gaixueyan@buaa.edu.cn

[1] School of Computer Science and Engineering, Beihang University, Xueyuan Road, Haidian 100191, Beijing, China

(3) To improve the simulation performance of HPC systems and simplify the modeling of the microarchitecture of the computing node, some simulators use the host nodes that are the same as the target HPC system, which can only simulate the HPC system when the target node is available.

The trace-driven methods firstly collect traces of program executions in available platforms and then simulate the target system using the traces. Most component-level simulators use this kind of method due to its simplicity. When using the trace-driven approach to simulate the HPC system, the researchers run the parallel program with the desired scale to collect specified event traces (e.g., communication traces) firstly; then the trace-driven simulator simulates the target HPC system by modeling the traces based on the target HPC system. Since there are various kinds of traces for parallel programs, trace-driven simulators are often used to evaluate the performance of components or sub-systems, e.g., simulating the interconnection network with message traces, simulating the storage system with file-access traces, etc. The trace-driven methods are very effective in simulating some specific characteristics of the HPC system. However, the trace-driven methods are difficult to simulate the dynamic procedure of the execution of parallel programs. Therefore, it is complex to comprehensively evaluate the interaction and integration between the application, the software environment and the hardware architecture through the trace-driven methods.

Traditional execution-driven full-system simulators (e.g., Gem5 (Binkert et al. 2011)) have powerful modeling and simulation capabilities for processors as well as systems, and provide multiple levels of simulation precisions. They also can run unmodified operating systems to simulate the execution of applications in the operating system. Although this kind of simulator cannot simulate large-scale parallel systems due to its performance and scalability, it is very suitable for modeling and simulating computing node of HPC systems, especially when the processor / accelerator / node is newly designed and therefore is unavailable in the stage of system design. However, the modeling and simulation of one single node cannot support large-scale execution of parallel programs, and therefore is not sufficient to evaluate the performance of the target HPC system.

This paper proposes an execution-driven HPC simulation system based on traditional execution-driven full-system simulators, more specifically, the Gem5 in this paper. Our simulation system incorporates an off-the-shelf execution-driven full-system simulator with a message emulation environment and an interconnection network simulation module. The off-the-shelf execution-driven full-system simulator, called node-simulator, is used to model the hardware architecture of the node of the target HPC system and then simulate one specified node of the target system by executing MPI processes as well as the operating system in the node simulation instance. The message emulation environment is used to emulate message transportation between the processes in the simulated node and other processes, which is essential to drive the execution of those processes in the node. The interconnection network simulation module assists the message emulation environment in computing the transport latency of messages by using the off-the-shelf networking simulator, more specifically, the Omnet++(Varga and Hornig 2008) in this paper. The messages among processes are collected in a pre-execution of the program with the desired number of processes in an available HPC system.

Compared to existing simulators of HPC systems, our simulation system leverages traditional architecture simulators to simulate large-scale HPC systems. By utilizing the powerful modeling & simulation capabilities of architecture simulators, researchers can not only perform detailed and accurate simulation of the target node of the HPC system, but also can evaluate the overall performance of the target system by running large-scale parallel programs in the simulation system. Our simulation system mainly has the following advantages:

(1) Our simulation system is the first work that can simulate HPC systems and the execution of large-scale parallel programs based on execution-driven using only one local host. And our simulation system has sufficient scalability.
(2) Our simulation system extends classic architectural simulators to enable them to simulate HPC systems and the execution of large-scale parallel programs. Researchers can comprehensively evaluate the interaction and integration between the application, the software environment and the hardware architecture through our simulation system.

The rest of this paper is organized as follows. Section 2 analyzes and presents our simulation approach of HPC systems. Section 3 introduces the architecture and implementation of the HPC simulation system. Section 4 gives experimental results. Section 5 discusses related work. Finally, we conclude the paper in Sect. 6.

2 Approach

2.1 Analysis

Execution-driven simulation of HPC systems requires large-scale execution of parallel programs on the modeled target system, which is very challenging, especially when the target node is unavailable.

Some architecture simulators (e.g., Gem5) support execution-driven simulation of distributed systems. The set of components of gem5 that simulates distributed systems is named dist-gem5. The approach dist-gem5 simulates a distributed system is to use enough node simulation instances to simulate all nodes of the target system, and use a switch simulation instance to connect all simulated nodes. The simulated switch is used to transport the data from different nodes.

Although the approach of dist-gem5 is theoretically able to simulate HPC systems, it is difficult to use in practice. There are three main reasons, as follows: (1) Each node simulation instance is a process running on the host. The scheduling of the processes by the host operating system will cause the running speed of each node simulation instance to be inconsistent. In order to balance the running speed of each simulation instance, each simulation instance stops the simulation procedure at certain intervals for synchronization. When the number of nodes reaches a certain scale, the simulation speed will be quite slow. (2) The scale of HPC systems far exceeds the scale of distributed systems. Simulating HPC systems requires enough local hosts, resulting in huge resource consumption. (3) When the node's hardware is relatively complex, such as having a large number of cores or having acceleration components, it will cause a single node instance to run slowly. Many node instances can make the simulation of the system slower.

The MPI programs use the parallel paradigm of single-program multiple-data (SPMD), in which each process executes the same program and computes on different data. Therefore, the behavior of nodes are often the same in the HPC system that executes an MPI program. Suppose an MPI program running with 4096 processes that are allocated to the node 0–255 of an HPC system, there will be only one or several kinds of node-behavior according to the number of process groups. Therefore, we can simulate one node of the target system under large-scale parallel execution of programs. Considering that processes always communicate with each other, when simulating the execution of the processes in one node, necessary messages must be simulated to drive those processes to continue. In addition, the latency of message transportation must also be simulated.

For a process running in one node, its running behavior can be mainly divided into two aspects: the computing behavior and the communication behavior. The computing behavior depends on the hardware and software architecture of the target node, which can be simulated by the execution-driven full-system simulators, e.g., Gem5. The communication behavior mainly depends on the implementation of communication functions as well as the target interconnection network. In this case, we can use the off-the-shelf network simulators to model the target interconnection network and compute the transport latency of messages.

2.2 Simulation of HPC systems based on node-simulation

Through the analysis, we propose an execution-driven simulation approach for HPC systems based on node simulation. Figure 1 shows the overview of our approach. We simulate the target HPC system by incorporating an off-the-shelf architecture simulator with a message emulation environment and an interconnection network simulation module in a local host. The researcher specifies one node of the target system to be simulated. Our simulation system simulates the node by using a node simulation instance of the *node-simulator*. Then the researcher run the parallel program with the desired process-node distribution file. The MPI processes that are distributed in the chosen node are simulation executed in the simulated node. During the simulation, the message emulation environment emulates message transportation between the processes in the simulated node and the other processes that are not started. The messages among processes are collected in a pre-execution of the program with the desired number of processes in an existing available HPC system. The interconnection network simulation module is based on an off-the-shelf network simulator Omnet++. The network simulator is used to model and simulate the target interconnection network. The interconnection network simulation module computes the transport latency of messages by simulating message transportation in the simulated interconnection network.

2.2.1 Modeling of the target node

The researcher models the computing node of the target HPC system using the execution-driven full-system simulator, including the instruction system, processor micro-architecture, memory system, etc. The researcher creates the system disk image deployed with the same software architecture as the target system, including the operating system, kernel, parallel environment (MPI library), etc. The modeling and configuring of hardware and software architecture makes the node full-system simulation instance the same as the node of the target system. When researchers run the parallel program in the node full-system simulation instance, the researchers can flexibly switch the simulation precision through the various precisions provided by the full-system simulator to explore the target computing node at different levels. The researchers can also use the functions provided by the simulator, such as checkpoints, fast forwarding, etc., to flexibly locate the program segments of interest and better pay attention to the behavior and performance of the system and the program.

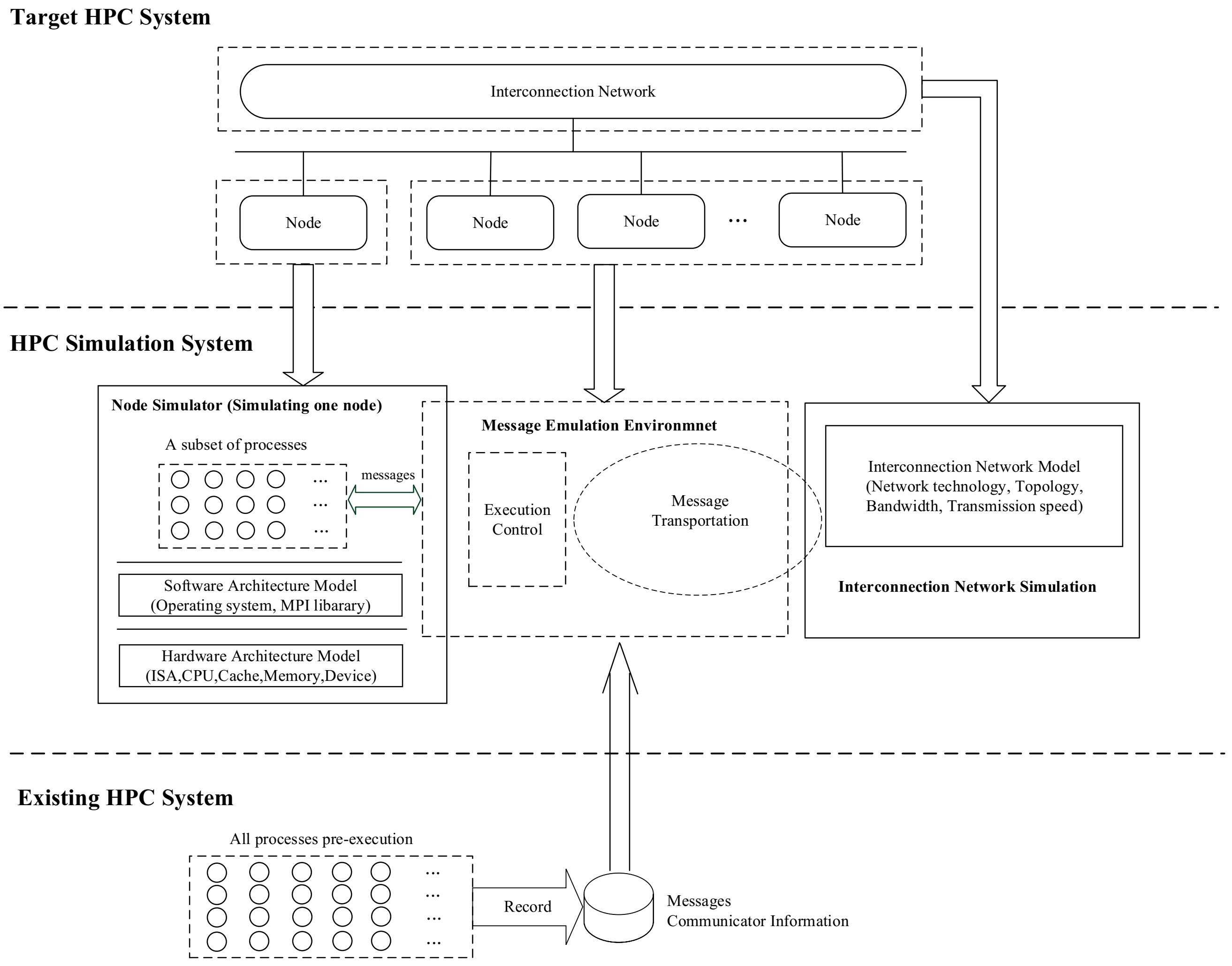

Target HPC System
Interconnection Network
Node
Node
Node
Node
HPC Simulation System
Node Simulator (Simulating one node)
A subset of processes
messages
Software Architecture Model (Operating system, MPI libarary)
Hardware Architecture Model (ISA,CPU,Cache,Memory,Device)
Message Emulation Environmnet
Execution Control
Message Transportation
Interconnection Network Model (Network technology, Topology, Bandwidth, Transmission speed)
Interconnection Network Simulation
Existing HPC System
All processes pre-execution
Record
Messages
Communicator Information

Fig. 1 Illustration of the simulation approach

2.2.2 Message emulation environment

When starting the node-simulator, the researcher specifies the node to be simulated, then runs the processes distributed on the node in the simulation instance according to the desired process-node distribution. We use the message emulation environment to ensure that those processes execute correctly as expected. That is, the message emulation environment controls the execution of the processes, and simulates the messages from/to all other nodes of the target system. When the program runs to the specified MPI functions, the functions are not executed by the native MPI library but re-directed to the message emulation environment to accomplish essential processing. When the processes execute to communicator-related functions and communication-group-related functions, the message emulation environment controls the execution of the process that makes the process behave with the specified rank and the specified number of processes. When the process executes the communication functions, the message emulation environment simulates the message transportation between the process and the processes in other nodes, to drive the execution of the process. In addition to making the process send and receive the correct message content, the message emulation also computes the execution time of the communication function, to make the execution behavior of the process closer to the execution behavior in target system.

2.2.3 Simulation of interconnection network

The researchers model the interconnection network of the target system using an off-the-shelf network simulator Omnet++, mainly including the network technology (e.g., Infiniband), topology, bandwidth and transmission speed, etc. An interconnection network simulation instance maintains the virtual target interconnection network. The message emulation environment requests the interconnection network simulation instance to simulate the message transportation to compute the transport latency of messages. Since the processes outside the simulated node are not executed, we only need to model the time of message transmissions in the interconnection network, regardless of the message content. It simplifies the design of interconnection network modeling. The interconnection network simulates the transmission of the message of a specified size from the source node to the destination node to compute the transport latency of the message.

Through the cooperative work of the node-simulator, the interconnect network modeling, and message emulation, we can simulate the complete execution of the processes in the specified node of the target system. The researcher can arbitrarily choose the node to simulate and specify the process-node distribution.

3 Implementation

Figure 2 shows the architecture of our simulation system. The system mainly consists of an off-the-shelf full-system simulator (Gem5 in this paper), the message emulation environment, and the interconnection network simulation module. The full-system simulator is used to model the hardware and software architecture of the node of the target HPC system and then starts a node simulation instance to simulate the specified node and runs the processes in the simulated node according to the desired process-node distribution. The message emulation environment is used to emulate message transportation between the processes in the simulated node and other processes, which is essential to drive the execution of those processes in the node. The interconnection network simulation module assists the message emulation environment in computing the transport latency of messages based on modeling the target interconnection network through an off-the-shelf network simulator (Omnet++ in this paper).

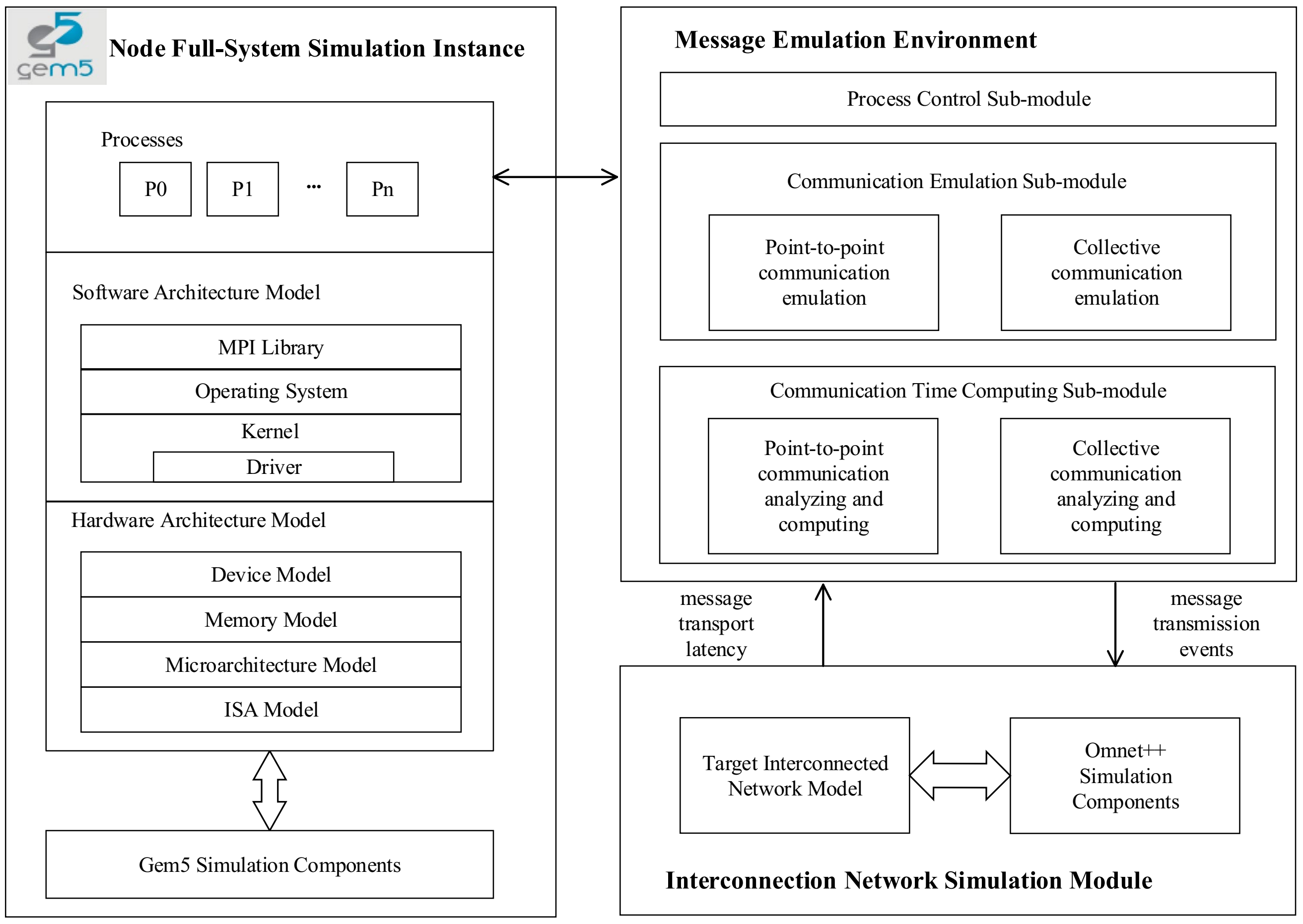

gem5
Node Full-System Simulation Instance
Processes
P0
P1
···
Pn
Software Architecture Model
MPI Library
Operating System
Kernel
Driver
Hardware Architecture Model
Device Model
Memory Model
Microarchitecture Model
ISA Model
Gem5 Simulation Components
Message Emulation Environment
Process Control Sub-module
Communication Emulation Sub-module
Point-to-point communication emulation
Collective communication emulation
Communication Time Computing Sub-module
Point-to-point communication analyzing and computing
Collective communication analyzing and computing
message transport latency
message transmission events
Target Interconnected Network Model
Omnet++ Simulation Components
Interconnection Network Simulation Module

Fig. 2 Architecture of the simulation system

3.1 Node full-system simulation instance

We use the Gem5 to realize our HPC simulator. Gem5 is one of the most classic computer architecture simulation frameworks which integrates the advantages of the M5(Binkert et al. 2006) and Gems(Martin et al. 2005) simulator. Gem5 supports full-system simulation and can run a completely unmodified operating system. Researchers can flexibly deploy the operating system kernel, MPI library, and other software environments in the node simulation instance.

For heterogeneous nodes with accelerators, we can use the corresponding accelerator simulator according to the type of the accelerator and combine it with Gem5 to model the heterogeneous node system. For example, the modeling and simulating of GPUs can utilize the classic GPU simulator gpgpu-sim (Bakhoda et al. 2009). Gem5-gpu (Power et al. 2014) is a classic CPU-GPU heterogeneous system architecture simulator which integrates gem5 with gpgpu-sim and can simulate the heterogeneous system with the full-system mode.

It should be noted that this paper is not focusing on how to model the microarchitecture in a simulation framework. Instead, we aim to propose and verify the approach to simulate the HPC system using an architecture simulator. The implementation of the node-simulator, such as the simulation of the heterogeneous architecture of the node, the simulation of accelerators, etc., is beyond this paper's scope.

3.2 Interconnection network simulation module

We model the interconnection network of the target HPC system using the classic network simulation framework Omnet++. The current working mode of Omnet++ is as follows. The user firstly models the target network architecture and defines the network transmission events to be simulated in the C++ files; then the user starts an Omnet++ simulation instance to simulate the defined network transmission events. Omnet++ will exit directly after the simulation is complete. We modified the working mode of Omnet++ to make the simulation instance stay in the memory. We add a management component in Omnet++ that listens for the requests of message transmission events. The management component drives a data transmission in the network simulation instance according to the request.

We developed the interconnection network simulation module based on the modified Omnet++, to compute the latency of a message transmission from the source node to the destination node in the target system. Figure 3 shows the structure of the interconnection network simulation module. After modeling the interconnection network of the target system, a software simulation instance of the target interconnection network is started and stays in the memory. The message emulation environment analyzes the communication-related functions and generates message transmission events to enter the simulation instance. The simulation instance simulates each event and returns the simulated transport latency back to the message emulation environment.

3.3 Message emulation environment

After modeling the hardware and software architecture of the target node, the researcher specifies the node to simulate and starts a node simulation instance. The researcher runs the processes in the simulated node according to the expected process-node distribution. We developed the message emulation environment which cooperates with the native MPI environment to simulate the execution of the communicator-related and communication-related MPI functions to ensure that the simulated processes run with the expected behavior. The message emulation environment consists of the sub-modules of the process control, the communication emulation, and the communication time computation. The process control sub-module emulates the communicator-related functions. The communication emulation sub-module and the communication time computing sub-module simulate the execution of the communication-related functions. They mainly fill the output of the functions with expected values, analyze the communication behavior, and model the execution time of the function based on the simulation result of the interconnection network simulation module.

3.3.1 Process control sub-module

After the simulated process is started in the simulation instance, each process has a global rank in the native MPI environment. However, the rank is not the expected rank to be simulated. The process control sub-module assigns each process the target rank to be simulated. It maintains the mapping between the global rank in the native MPI environment and the target rank in the message emulation environment. When a process runs to the communicator-related and communication-group-related function, the process control sub-module will assign the function the expected output according to the mapping so that the process can behave as the target process it simulates. For example, in Fig. 4, we simulate the node 16 of the target system and we run the process 512–543 in the node. The global ranks of the processes in the native MPI environment are from 0 to 31, and the global ranks in the message emulation environment are their target ranks from 512 to 543. In many applications, there are not only the global communicator but also some other communicators created by some communicator-related functions. Therefore, in addition to maintaining the corresponding relationship between the global ranks in the message emulation environment and the native MPI environment, it

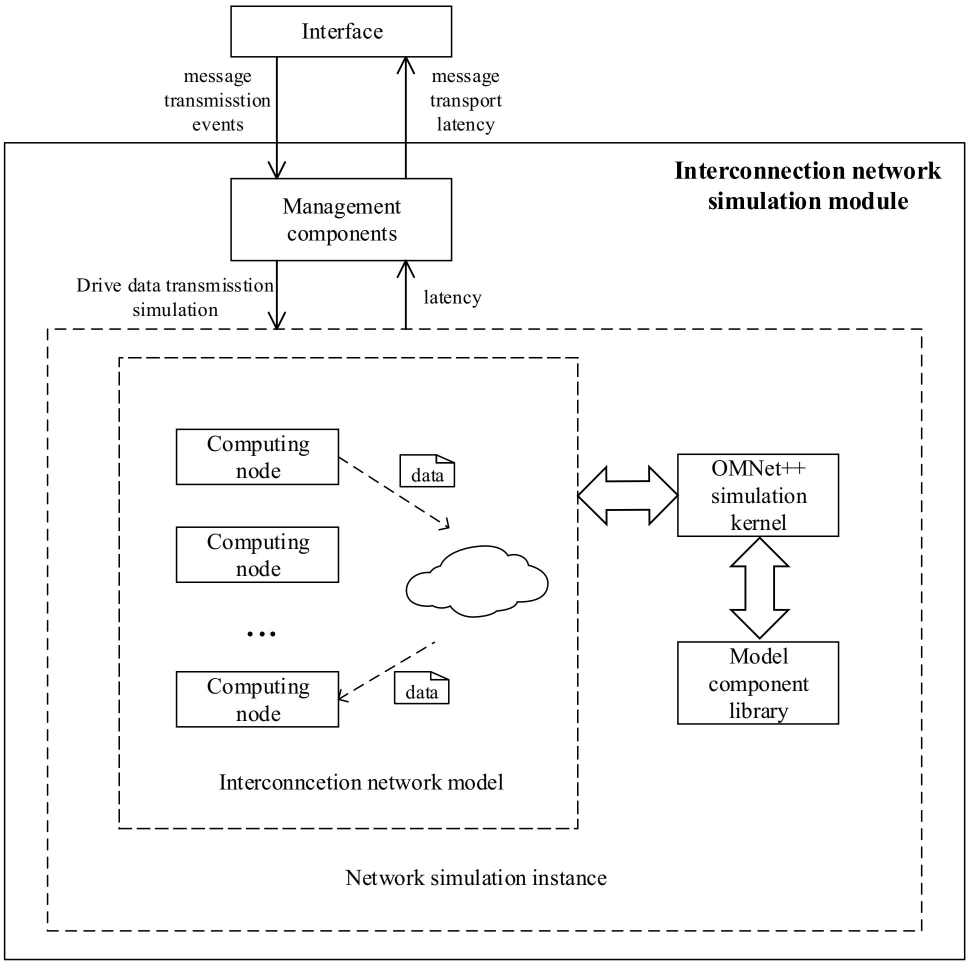

Interface
message transmisstion events
message transport latency
Interconnection network simulation module
Management components
Drive data transmisstion simulation
latency
Computing node
Computing node
...
Computing node
data
data
OMNet++ simulation kernel
Model component library
Interconncetion network model
Network simulation instance

Fig. 3 Structure of the interconnection network simulation module

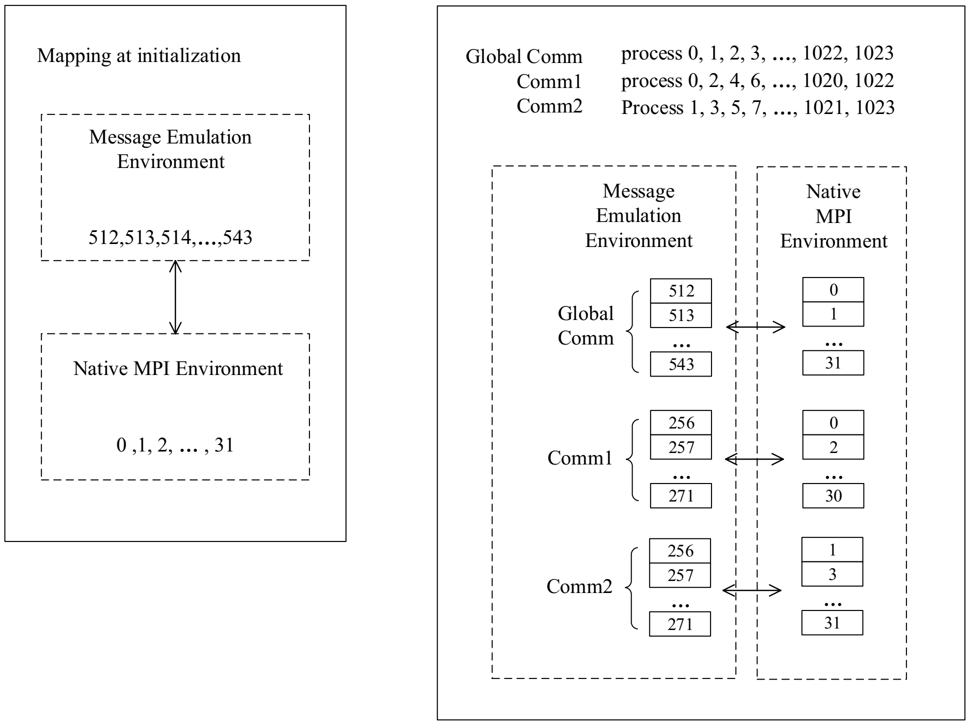

Mapping at initialization
Message Emulation Environment
512,513,514,...,543
Native MPI Environment
0 ,1, 2, ... , 31
Global Comm process 0, 1, 2, 3, ..., 1022, 1023
Comm1 process 0, 2, 4, 6, ..., 1020, 1022
Comm2 Process 1, 3, 5, 7, ..., 1021, 1023
Message Emulation Environment
Native MPI Environment
Global Comm
512
513
...
543
0
1
...
31
Comm1
256
257
...
271
0
2
...
30
Comm2
256
257
...
271
1
3
...
31

Fig. 4 An example of mapping the ranks

is also necessary to maintain the corresponding relationship between the ranks in other communicators in the message emulation environment and the global rank in the native MPI environment. For example, in Fig. 4, the comm1 and comm2 are split from the global communicator according to the result of *"global rank % 2"*. The process control sub-module maintains the mapping between the ranks in the comm1/comm2 and the global rank in the native MPI environment for each process.

3.3.2 Communication time computing sub-module

In the pre-execution of the parallel program, we record all uncertain factors in the communication functions. In this case, if the process scale of the program is specified, the sequence of communication functions for each process is fixed. To simplify the simulation procedure of the communication function, we developed the communication time computing sub-module. We record the sequences of communication functions of the processes which will be simulated during the pre-execution. Before simulating, the communication time computing sub-module traverses the sequences of communication functions, and analyzes each function to generate the message transmission events. The message transmission event mainly indicates the source node, destination node, and data length without modeling specific message content. The communication time computing sub-module enters the events into the interconnection simulation instance, obtains the simulated message transport latency, and records them. During the simulation, the message emulation sub-module will use the recorded time to model the execution time of the communication-related functions. Figure 5 shows the main procedure of

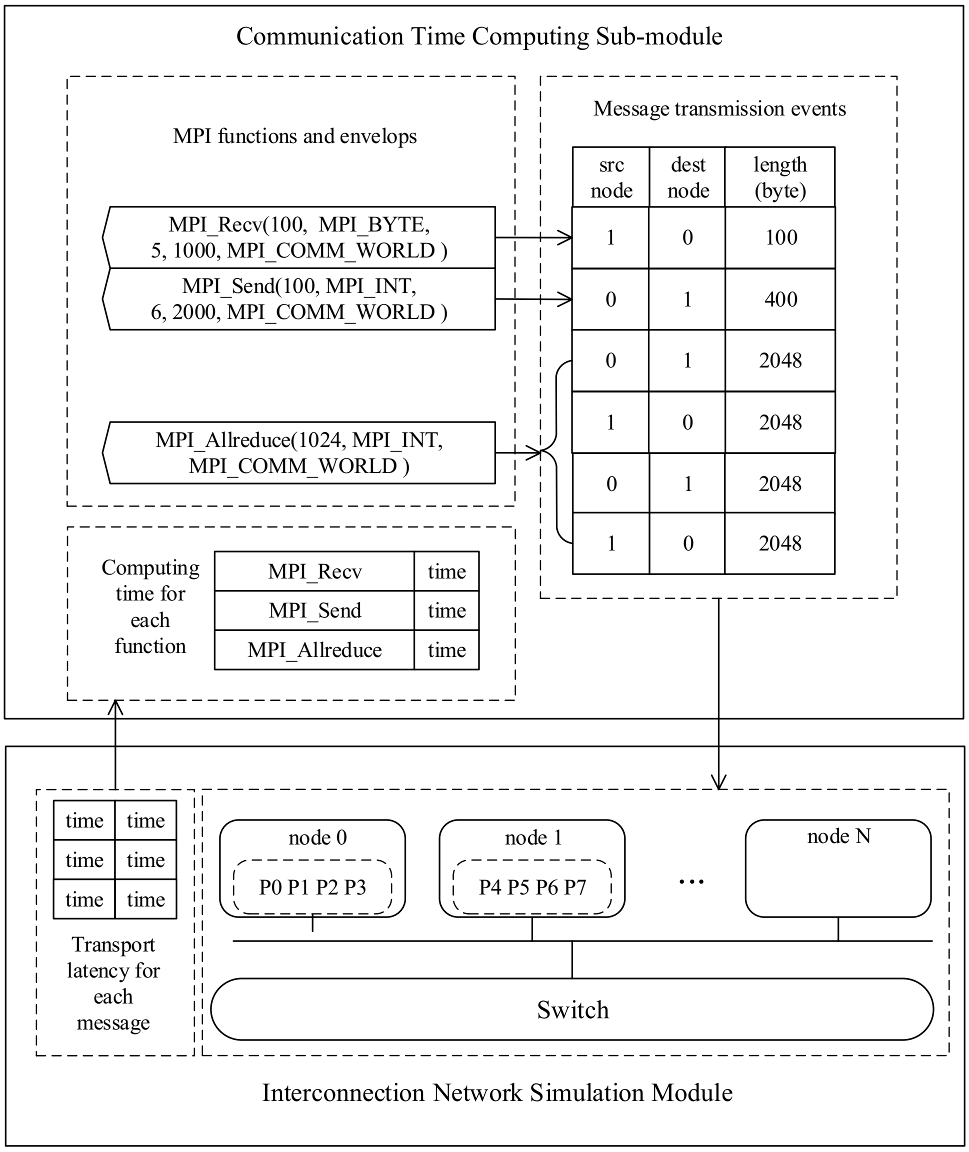

Communication Time Computing Sub-module
MPI functions and envelops
MPI_Recv(100, MPI_BYTE, 5, 1000, MPI_COMM_WORLD)
MPI_Send(100, MPI_INT, 6, 2000, MPI_COMM_WORLD)
MPI_Allreduce(1024, MPI_INT, MPI_COMM_WORLD)
Message transmission events
src node
dest node
length (byte)
1 0 100
0 1 400
0 1 2048
1 0 2048
0 1 2048
1 0 2048
Computing time for each function
MPI_Recv time
MPI_Send time
MPI_Allreduce time
time time
time time
time time
Transport latency for each message
node 0
P0 P1 P2 P3
node 1
P4 P5 P6 P7
…
node N
Switch
Interconnection Network Simulation Module

Fig. 5 The procedure of computing the time of the communication function

how the communication time computing sub-module computes the time of each communication function.

For the point-to-point communication functions, the communication time computing sub-module generates the message transmission event according to the envelope and the process-node distribution. For the collective communication functions, the communication time computing sub-module generates multiple message transmission events based on the algorithm implementation of the collective communication functions in the native MPI library. The implementation of collective communication function in the native MPI library is not directly converted into multiple point-to-point communications according to the semantics of the function but according to a certain algorithm. For example, the implementation of the broadcast communication function *MPI_Bcast()* is based on the broadcast tree algorithm, instead of the root process doing point-to-point sending operations to all other processes in turn. According to the function's algorithm, the communication time computing sub-module parses out the point-to-point communication events that the process participates in the algorithm. Then the communication time computing sub-module analyzes whether each message transmission event is intra-node communication or inter-node communication and enters the inter-node message transmission events into the interconnection simulation instance.

The example in Fig. 6 shows the parsing procedure of the function *MPI_Allreduce()*, which is based on the

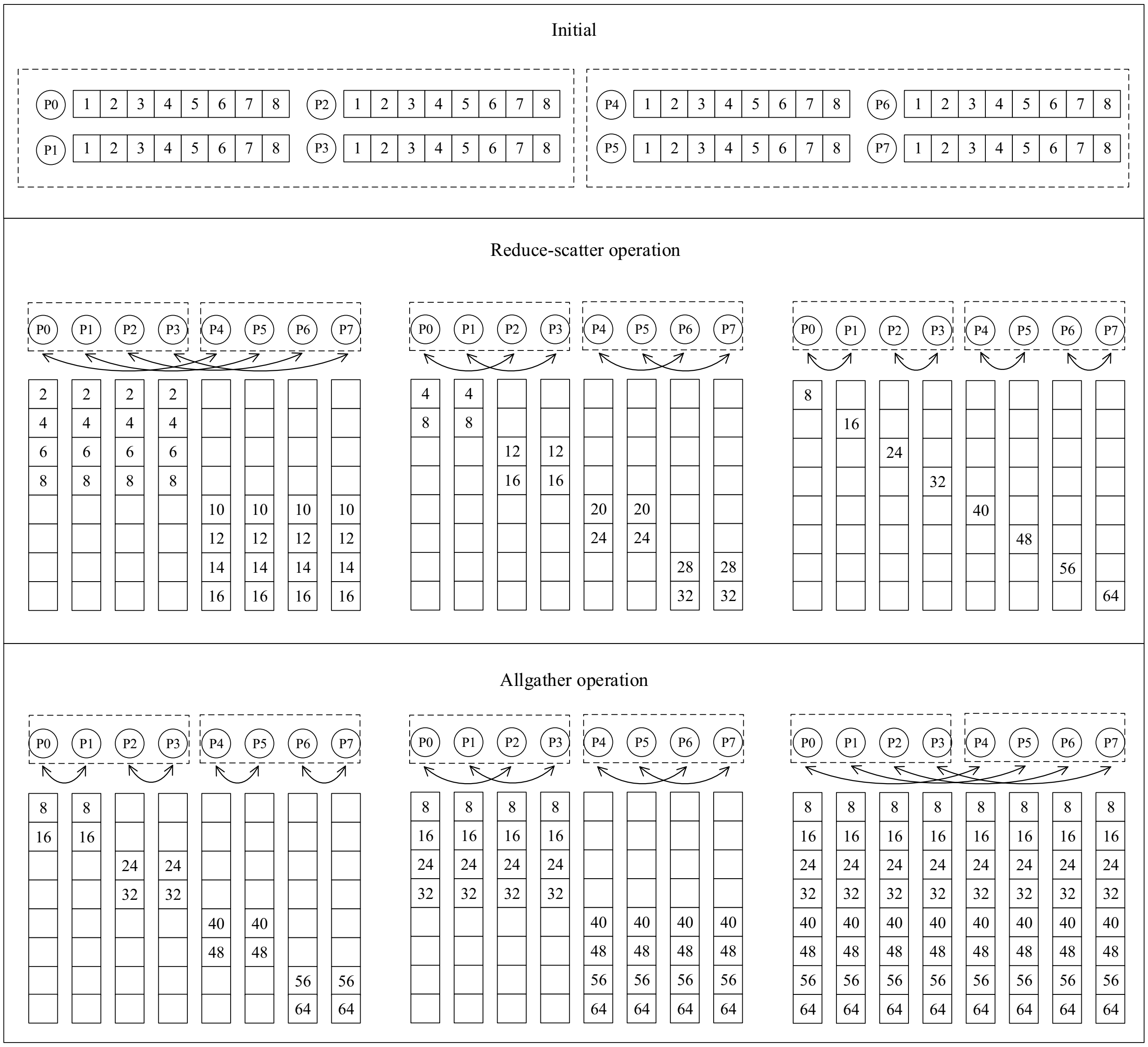


Fig. 6 The parsing procedure of *MPI_Allreduce()*

implementation in the MPICH3(MPICH 2023). For the function *MPI_Allreduce()*, when the data length is greater than 2KB, Rabenseifner's algorithm is used. This algorithm is divided into two steps. The first step is a *reduce-scatter* operation, and the second step is an *allgather* operation. The *reduce-scatter* operation uses the recursive-halving algorithm, and the *allgather* operation uses the recursive doubling algorithm. We further describe these two steps with an 8-process example in Fig. 6. In the *reduce-scatter* operation, each process exchanges data with the process that is a distance p/2 away in the first step. The process sends the data needed by the processes in the other half and receives the data needed by the processes in its own half. Then the process performs a *reduce* operation. In the second step, each process exchanges data with the process that is a distance p/4 away, and finally exchanges data with the process that is a distance 1 away. In the *allgather* operation, in the first step, each process exchanges data with the process that is a distance of 1 away, and so on until that exchanges data with the process that is a distance p/2 away. Assuming that these 8 processes are distributed in 2 nodes, process 0 performs 8 intra-node sending and receiving and 4 inter-node sending and receiving when running the *MPI_Allreduce()* function.

3.3.3 Communication emulation sub-module

(1) Point-to-point communication

When the process executes the point-to-point communication, the message emulation sub-module analyzes if the target process is on the same node as the process, according to the message envelope and the process-node distribution. If the two processes are on the same node, the message emulation sub-module translates the ranks of the two processes in the message emulation environment into the ranks in the native MPI environment. Then the process calls the native MPI functions with the translated ranks to complete the communication. If the target process is not on the same node, the message emulation sub-module analyzes the communication semantics of this function and then simulates the execution of the function. Figure 7 shows an example of point-to-point communication emulation, it illustrates how the communication emulation sub-module analyzes whether the function is intra-node or inter-node communication. Table 1 shows the emulation procedure of some commonly used point-to-point communication function types in the case of inter-node communication.

(2) Collection communication

The simulation of collective communication functions is based on the semantics of the function and the recorded data comprehensively. For example, *MPI_Allreduce()* is used for all processes performing a reduce operation. Figure 6 shows the result of the function. The data in the receive buffers of all processes is the same after the function completes. For these functions that all processes have the same content of receive buffers, the message emulation sub-module fills the receive buffers of processes and models the execution time of the function. Some functions are used to exchange data between every two processes in the same communicator, such as *MPI_Alltoall()* and *MPI_Alltoallv()*. For these functions, processes call the native *MPI_Isend()* and *MPI_Irecv()* to exchange data with the processes in the same node and receive the data from the processes in other nodes from the message emulation sub-module.Table 2 shows the emulation procedure of some commonly used collective functions.

(3) Model the execution timeline for each process

During the simulation execution of the processes, the message emulation sub-module models the execution timeline for each process based on the target HPC system. The total execution time mainly consists of computation time and communication time. The computation time refers to the time it takes for a process to execute the computation statements. The communication time refers to the time of the execution of the communication functions. Figure 8

Table 1 The emulation procedure of the point-to-point communication function

Function type	Emulation procedure
Blocking standard send function	The process gets the expected return value from the message emulation sub-module and then returns. The message emulation sub-module models the execution time of the send function.
Blocking standard receive function	The process receives the expected message buffer and return value from the message emulation sub-module and then returns. The message emulation sub-module models the execution time of the receive function.
Non-blocking standard send function	The process gets the expected return value from the message emulation sub-module. The message emulation sub-module sets the value to the request object and puts the request object and the envelope into a table. Then the process returns.When the process runs to the corresponding wait function, the message emulation sub-module gets the envelope from the table and models the execution time of the wait function.
Non-blocking standard receive function	Similar to the simulation procedure of the non-blocking standard send function. The difference is that the message emulation sub-module also puts the head address of the receive buffer into the table and fills the receive buffer with expected content in the wait function.

Table 2 The emulation procedure of the collective communication function

Function	Emulation of the function
MPI_Bcast	If the process is not the root, the process receives the expected message buffer and return value from the message emulation sub-module. If the process is the root, the process receives the expected return value from the message emulation sub-module.The message emulation sub-module models the execution time of the function.
MPI_Gather MPI_Gatherv MPI_Reduce	If the process is the root, the process receives the expected message buffer and return value from the message emulation sub-module. If the process is not the root, the process receives the expected return value from the message emulation sub-module.The message emulation sub-module models the execution time of the function.
MPI_Allgather MPI_Allgatherv MPI_Allreduce	The process receives the expected message buffer and return value from the message emulation sub-module. The message emulation sub-module models the execution time of the function.
MPI_Alltoall MPI_Alltoallv	The process calls multiple pairs of *MPI_Irecv()* and *MPI_Isend()* to exchange intra-node messages with the processes on the same node. The process receives the expected inter-node messages and return value from the message emulation sub-module. The message emulation sub-module models the execution time of the function.

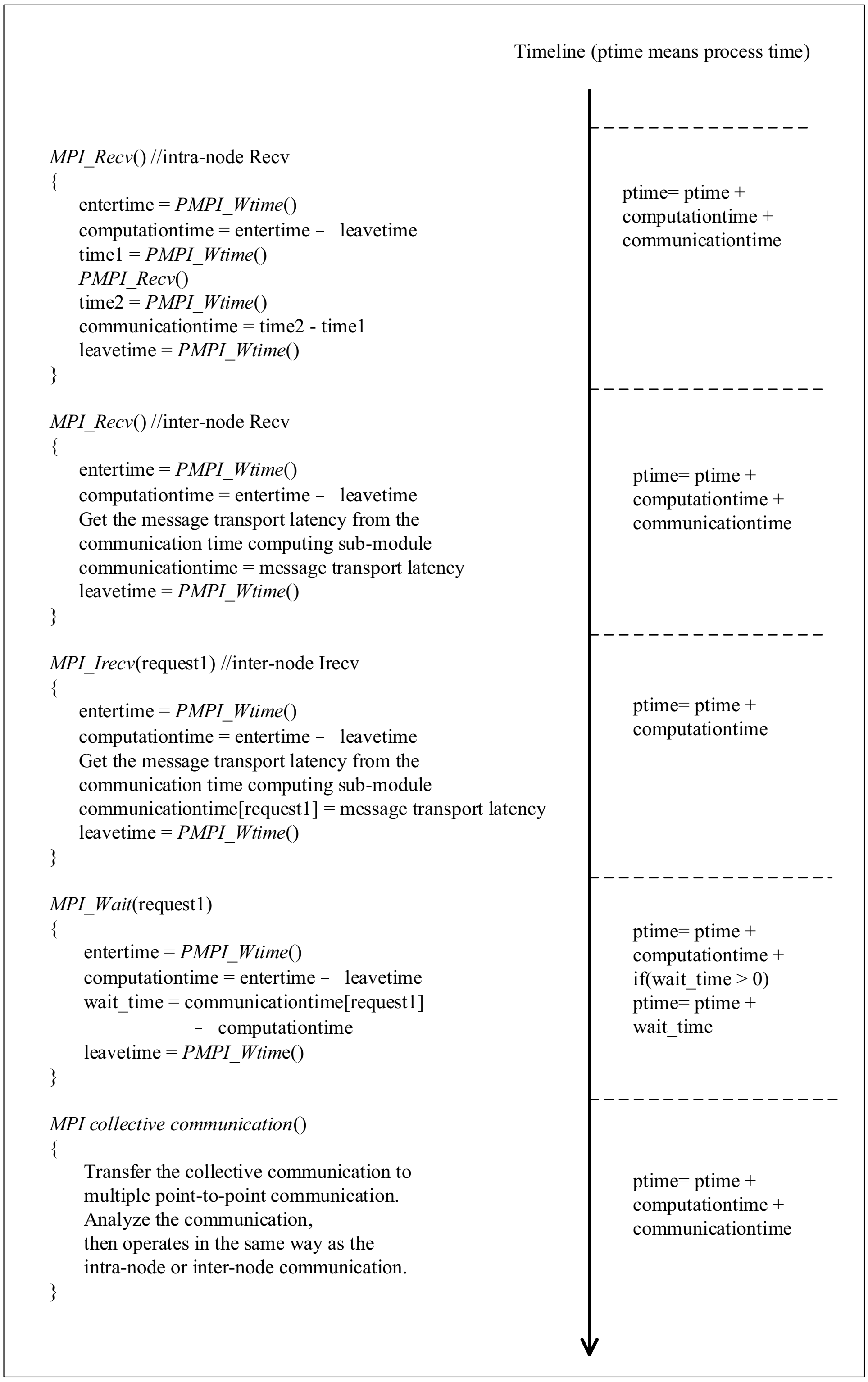
Fig. 7 An example of point-to-point communication emulation

```
// members of comm2 is from process 512 to 1023
// simulating the node 20, in which running process 608 to process 639
// analyzing the point-to-point communication of process 608

MPI_Recv(buf, count, datatype, 50, 1000, comm2 ,status)
{
    The process control sub-module analyze the envelop, the dest 50 in comm2 indicates
the  process 562. The porcess 562 is not simulated, so the process gets buf and status from
the  communication emulation  submodule.
}

MPI_Irecv(buf, count, datatype, 100, 2000, comm2 , request)
{

    The process control sub-module analyze the envelop, the dest 100  in comm2
indicates the  process 612. The porcess 612 is simulated in the node, and the global rank
in the native MPI evironment is 5, so the process call  native MPI function as follows,
   call PMPI_Irecv(buf, count, datatype, 5, 2000, MPI_COMM_WORLD , request)
}

MPI_Wait(request, status)
{
    call PMPI_Wait(request, status)
    The process control sub-module modifies the status as expected.
}
```

shows the build method of the execution timeline. The message emulation sub-module starts with the *MPI_Init()* function, stores the time the process leaves, and enters each communication function. The execution time of the computation statements between two adjacent communication functions can be computed by subtracting the time of leaving the former function from the time of entering the latter function. Since the hardware and software architecture of the node simulation instance is consistent with the target node, this time can be directly used to establish the timeline without any handling. The establishment of the execution time of the communication function is based on the communication type, semantics, and implementation algorithm of the function. Figure 8 shows the procedure of establishing some basic communication functions.

3.4 Pre-execution of the program

During the pre-execution of the parallel program, the parallel program is completely run on an existing HPC system


Timeline (ptime means process time)
ptime= ptime + computationtime + communicationtime
ptime= ptime + computationtime + communicationtime
ptime= ptime + computationtime
ptime= ptime + computationtime + if(wait_time > 0) ptime= ptime + wait_time
ptime= ptime + computationtime + communicationtime

```
MPI_Recv() //intra-node Recv
{
    entertime = PMPI_Wtime()
    computationtime = entertime –  leavetime
    time1 = PMPI_Wtime()
    PMPI_Recv()
    time2 = PMPI_Wtime()
    communicationtime = time2 - time1
    leavetime = PMPI_Wtime()
}

MPI_Recv() //inter-node Recv
{
    entertime = PMPI_Wtime()
    computationtime = entertime –  leavetime
    Get the message transport latency from the
    communication time computing sub-module
    communicationtime = message transport latency
    leavetime = PMPI_Wtime()
}

MPI_Irecv(request1) //inter-node Irecv
{
    entertime = PMPI_Wtime()
    computationtime = entertime –  leavetime
    Get the message transport latency from the
    communication time computing sub-module
    communicationtime[request1] = message transport latency
    leavetime = PMPI_Wtime()
}

MPI_Wait(request1)
{
    entertime = PMPI_Wtime()
    computationtime = entertime –  leavetime
    wait_time = communicationtime[request1]
                  –  computationtime
    leavetime = PMPI_Wtime()
}

MPI collective communication()
{
    Transfer the collective communication to
    multiple point-to-point communication.
    Analyze the communication,
    then operates in the same way as the
    intra-node or inter-node communication.
}
```

Fig. 8 Procedure of building the execution timeline

with the desired scale to collect the necessary messages and information. The researcher specifies the processes that need to record. We instrument all the functions that need to be recorded.

The data volume of the recorded log is quite large because of the need to log the contents of all messages received from other processes. Therefore, our system mainly reduces volume of the log in two ways:

(1) Our system only records the message content of inter-node communication. Processes on the same node are always simulation executed together. When the process communicates with processes on the same node, it can directly send/receive the message to/from them. We only record the messages received from the processes on other nodes. For point-to-point communication functions, if the message is received from a process of other nodes, the contents of the entire receive buffer are recorded. For some collective communication functions used to exchange data between every two processes, we only record the data of the received buffers, which corresponds to the processes in other nodes.

(2) As mentioned in section 3.3.2, a notable feature of some collective communications is that all of the processes in the communicator have the same content of the receive buffer after completing the collective communication, such as the function *MPI_Allreduce()*, *MPI_Allgather()*, and *MPI_Bcast()*. For this type of functions, our system only records one copy of data instead of multiple data copies for each process.

4 Experiments

4.1 Methodology

We evaluate our simulation system by simulating a 300-nodes target HPC system, in which each node has an FT2000+ processor based on the ARM V8 ISA. The HPC system is part of the prototype system of a supercomputer. The 300 nodes are connected to one switch. Table 3 shows the detailed configuration of the target HPC system.

We run the simulator in the local host equipped with two 12-core Intel Xeon processors and 256GB memory. We model the target node according to the node configuration (same number of cores and memory size, etc.), using the ARM V8 model of Gem5. We create a disk image and deploy the same operating system and MPI library in it. The MPI library deployed in the target node is a customized version and is not open source, so we use the original version instead.

We model the target interconnection network using Omnet++. The interconnection network of the target prototype system is self-developed and technical details are not in the open. After investigation, we find that the speed of this network is similar to that of the 100Gbps Infiniband network. So we model the target interconnection network based on the 100Gbps Infiniband network instead.

We use the NPB (NAS Parallel Benchmarks)(NPB 2023) as the workload to evaluate our system. The NPB has 8 classic programs, including IS (Integer Sort), CG (Conjugate Gradient), EP (Embarrassingly Parallel), MG (Multi-Grid), LU (Lower-Upper GaussSeidel solver), FT (discrete 3D fast Fourier Transform), BT (Block Tri-diagonal solver) and SP (Scalar Penta-diagonal solver). We run the 8 programs on the target system with different scales according to specified process-node distribution, respectively. Then we simulate one node of the target HPC system and run the corresponding processes of the programs in the simulated node.

Table 3 Configuration of the target HPC system

Parameter	Specifications
Node configuration	CPU: FT2000+ (customized ARMv8 architecture) @2.3GHz 64 processor cores L1dcache 2MB L1icache 2MB L2cache 256MB shared by 64 cores 128GB memory
Operating system	Ubuntu 18.04
MPI	MPICH3
Interconnection connection	Autonomous high speed interconnection network (100Gbps)

4.2 Validation

This experiment is to evaluate the accuracy of our simulation system by comparing the execution of processes of each program in the target system with that of our simulation system. The number of processes of CG, EP, IS, FT, LU, and MG is 256, 512 and 1024, respectively, while the number of processes of SP and BT is 256, 576 and 1024, respectively. The problem size of MG,EP,and IS is CLASS A, the problem size of LU, SP and BT is CLASS B, and the problem size of CG and FT is CLASS C. For the seven programs except for EP, each program consists of multiple iterations. The behavior of each iteration is close, so we just selectively simulate the procedure of the first three iterations of these programs. For EP, we simulate the whole procedure. It should be pointed out that in the main procedure of EP, there is only computation but no communication. After the main procedure, each process will call MPI communication functions to exchange some statistical information. To better verify the simulation of the communication, we simulate the whole procedure that refers to the execution from *MPI_Init()* to *MPI_Finalize()*.

We firstly run these programs with a certain number of processes on the target system. We distribute the processes in order of node number starting from node 0. The node of the target system has 64 cores, we also distribute the processes in the order of the process ranks, and there are 64 processes running in each node. Then we simulate node 0 of the target system with the AtomicCPU of gem5 and run process 0 to process 63 of each program to evaluate the accuracy of our simulation system.

To further demonstrate the simulation accuracy of the program's computation and communication behaviors in our simulation system, we define the time it takes for a process to run the computation statements of the program as the computation time, and define the time of running the communication functions as the communication time. Obviously, the total time of the execution of the process is the sum of the computation time and the communication time.

Figure 9 shows the simulation result of the 8 programs with different number of processes running in the simulation system. In Fig. 9, we list the average total time of the simulated execution of the processes in the simulation system and the average total time of the real execution in the target system, respectively. We also list the average computation time of the processes in the simulation system and the target system. The communication time can be obtained by subtracting the computation time from the total time.

Table 4 shows the error rate of our system. For each program, we separately show the error rate in total time, error rate in computation time, and error rate in communication time. The error rate is calculated by Eq. 1.

$$rate = \left(1 - \frac{simulated\_time}{real\_time}\right) \times 100\% \tag{1}$$

From Table 4, the error rate of the average total execution time of the processes ranges from 0.5% to 10.5%, and the errors are mainly composed of computation time errors and communication time errors. Table 4 shows that the error rate of computation time ranges from 4.6% to 14.5%. From Table 4, it can be concluded that EP and FT have the relatively largest errors of the computation behavior. This is because the computation behavior is the main behavior in EP and FT, and it is easier to accumulate errors of computation behavior. For instance, all processes of EP run computation statements from the beginning, and only call several communication functions to transport results to other processes until the whole computation procedure is completed. The error of computation time is mainly composed of two aspects. One is that we cannot accurately model the microarchitecture of the target node, and the other is the error of the computer architecture simulator (Gem5) itself. Researchers can improve the accuracy of computation behavior by modeling the hardware architecture (e.g., microarchitecture, memory, etc.) of the node in more detail.

From Table 4, the error rate of the communication time ranges from 4% to 11%, and the simulated communication time is always less than the real communication time. It can be seen from Table 4 that the errors of communication behavior of LU, SP, and BT are relatively larger. This is because these programs call a large number of communication functions, which leads to the accumulation of errors of communication behavior. Although there are only a few communication function calls in IS, the communication functions called are full exchange collective communications, which will generate a vast number of messages, therefore the error of communication behavior of IS is not relatively small. The error of communication time is mainly composed of three aspects. The first one is that we cannot accurately model the target interconnection network. The second one is that we lack the simulation of the data dependency when simulating some communication functions. For example, when the process runs to *MPI_Recv()*, we directly compute the transmission time of the message, but do not model the possible synchronization time for the target process to call the corresponding *MPI_Send()*. This causes the simulated communication time is always less than the real communication time. The third one is the error of the network simulator itself. Researchers can improve the accuracy of communication behavior by modeling the interconnection network and simulating the procedure of message transportation in more detail.

4.3 Performance of the simulation system

This experiment is to evaluate the performance of the simulation system. The simulation speed is the most important metric to evaluate the performance of an architecture simulator. Table 5 and Table 6 show the performance of our system by listing the simulation time and the slowdown of each program.

There are two main reasons that affect the simulation speed of our HPC simulation system. One is the speed of the architecture simulator (Gem5) that we use, and the other is the speed of the components that we extend the architecture simulator (e.g., message emulation environment). To verify the main reasons that affect the simulation speed, we do a set of comparative experiments.

We assume that the target system has only one computing node. The configuration of the computing node is the same as that in Table 3. We simulate the execution of the eight programs with the scale of 64 processes in the target computing node, respectively. Since all the 64 processes of the programs can run in the node, we can only use the Gem5 to simulate the target system and simulate the execution of the programs.

The problem size of all programs is CLASS A. For the seven programs except for EP, we just selectively simulate the process of the first three iterations of these programs. For EP, we simulate the whole procedure. We run the 8 programs with the scale of 64 processes in a Gem5 simulation instance, respectively. Table 7 shows the simulated execution time of the programs versus the simulation time that the

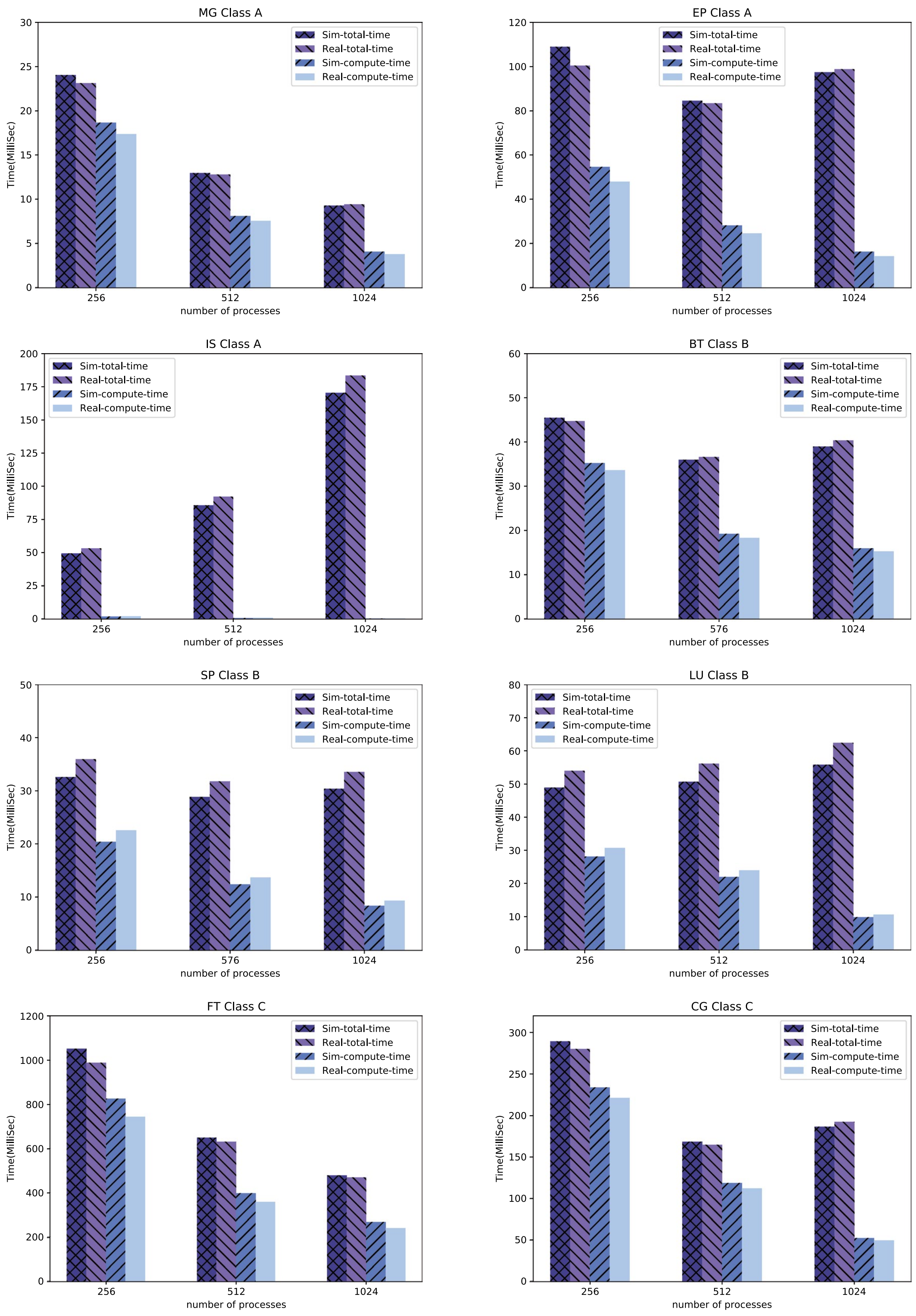


Fig. 9 The result of simulation. (Sim-total-time indicates the average total time of the simulated execution of the processes, Real-total-time indicates the average total time of the real execution of the processes in the target system, Sim-compute-time indicates the average computation time of the simulated execution, Real-compute-time indicates the average computation time of the real execution.)

Table 4 The error rate of the simulation system (%) (The Comp time indicates the computation time, the Comm time indicates the communication time)

Program	Num of proc	Total time	Comp time	Comm time
MG	256	−4.02%	−7.58%	7.18%
	512	−1.37%	−7.38%	7.26%
	1024	1.37%	−7.65%	7.44%
EP	256	−4.28%	−14.05%	3.98%
	512	−1.32%	−13.96%	3.98%
	1024	1.39%	−14.11%	4.01%
IS	256	7.01%	8.04%	6.96%
	512	6.99%	7.78%	6.98%
	1024	7.11%	7.86%	7.1%
BT	256	−1.65%	−4.86%	8.03%
	576	1.55%	−5.07%	8.21%
	1024	3.41%	−4.66%	8.32%
SP	256	9.38%	9.53%	9.13%
	576	9.27%	9.37%	9.19%
	1024	9.5%	10%	9.31%
LU	256	9.51%	8.56%	9.51%
	512	9.81%	8.32%	9.81%
	1024	10.41%	7.51%	11.01%
FT	256	−6.26%	−11.06%	7.81%
	512	−2.82%	−10.88%	7.83%
	1024	−1.72%	−11.41%	7.89%
CG	256	−1.01%	−5.74%	5.58%
	512	−2.06%	−5.67%	5.62%
	1024	2.93%	−5.33%	5.82%

Table 5 The simulation time of programs (seconds)

Program	Number of processes		
	256	512/576	1024
MG	3.080×10^3	1.702×10^3	1.180×10^3
EP	11.365×10^3	8.134×10^3	6.761×10^3
IS	1.678×10^3	1.420×10^3	1.255×10^3
BT	4.648×10^3	2.892×10^3	2.528×10^3
SP	3.014×10^3	2.134×10^3	1.694×10^3
LU	5.292×10^3	5.038×10^3	3.284×10^3
FT	107.507×10^3	57.092×10^3	40.740×10^3
CG	30.146×10^3	17.449×10^3	10.180×10^3

Gem5 simulates the programs. The slowdown is calculated by comparing the simulation time and the simulated execution time of each program. From Table 7, it can be concluded that the simulation speed of our simulation system is close to the Gem5. Therefore, it can be concluded that the main factor affecting the speed of our simulation system is the simulation speed of the off-the-shelf architecture simulator we use.

Table 6 The slowdown of programs

Program	Number of processes		
	256	512/576	1024
MG	1.281×10^5	1.313×10^5	1.271×10^5
EP	1.043×10^5	0.961×10^5	0.694×10^5
IS	0.339×10^5	0.166×10^5	0.074×10^5
BT	1.021×10^5	0.803×10^5	0.648×10^5
SP	0.925×10^5	0.740×10^5	0.558×10^5
LU	1.082×10^5	0.994×10^5	0.588×10^5
FT	1.022×10^5	0.879×10^5	0.849×10^5
CG	1.041×10^5	1.036×10^5	0.545×10^5

Table 7 The slowdown of Gem5 simulator in our system

Program	Simulated execution time(seconds)	Simulation time (seconds)	Slowdown
MG	94.686×10^{-3}	10.480×10^3	1.107×10^5
EP	271.880×10^{-3}	30.588×10^3	1.125×10^5
IS	18.314×10^{-3}	2.123×10^3	1.159×10^5
BT	46.303×10^{-3}	5.128×10^3	1.107×10^5
SP	29.650×10^{-3}	3.288×10^3	1.109×10^5
LU	44.240×10^{-3}	5.010×10^3	1.132×10^5
FT	255.220×10^{-3}	28.564×10^3	1.119×10^5
CG	28.651×10^{-3}	3.250×10^3	1.134×10^5

Note: When Gem5 is simulating, no matter how many cores it simulates, it all works with one thread. Although gem5 has a multi-threading mode, this mode can only be used for fast forwarding, not for simulation

4.4 Comparison with dist-gem5

This subsection conducts a set of comparative experiments to compare the accuracy and performance of our system with dist-gem5. We try to use dist-gem5 to simulate HPC systems. The number of simulation instances launched by dist-gem5 is the number of nodes of the target system plus one (one switch instance), each simulation instance occupies one processor core. For HPC systems with a large number of nodes, using dist-gem5 to simulate HPC systems requires a considerable number of hosts. Therefore, using dist-gem5 to simulate HPC systems is theoretically possible, but difficult to implement in practice.

In order to be able to complete the comparative experiment between our system and dist-gem5, we assume that the target HPC system has only four nodes, and the four nodes are connected through an Ethernet switch. Each node has an ARMv8 processor with 8 cores and 8 G memory. We run eight programs of NPB in dist-gem5 and our system respectively. The number of processes for MG, CG, IS, EP, LU and FT is 32. The number of processes for BT and SP is 25. Processes are distributed in order of process number and node number. The program scale of all programs is CLASS A. We also selectively simulate the procedure of the first three iterations of the seven programs except EP. For EP, we simulate the whole procedure. In our system, we select node 0 of the target system and simulate the running of processes 0 to 7.

Table 8 shows the average simulated execution time of processes 0 to 7 of programs in our system and dist-gem5. Figure 10 shows the ratio of the simulated time in our system to the simulated time in dist-gem5, which is calculated by comparing the simulated time in our system and the simulated time in dist-gem5. From Table 8 and Fig. 10 we can conclude that the simulated time of programs in our system is approximate to the simulated time in dist-gem5.

Table 9 shows the simulation time of programs in our system and dist-gem5. Table 10 shows the slowdown of programs in our system and dist-gem5. Figure 11 shows the speedup of the simulation speed of our system to dist-gem5, which is calculated by comparing the simulation time in dist-gem5 and the simulation time in our system. From Table 9, Table 10 and Fig. 11 we can conclude that the simulation

Table 8 Simulated execution time of programs: our system vs. dist-gem5 (seconds)

Program	Simulated execution time (our system)	Simulated execution time (dist-gem5)
MG	245.955×10^{-3}	248.514×10^{-3}
EP	788.965×10^{-3}	796.030×10^{-3}
IS	766.846×10^{-3}	818.402×10^{-3}
BT	137.358×10^{-3}	139.459×10^{-3}
SP	73.283×10^{-3}	74.784×10^{-3}
LU	84.629×10^{-3}	87.859×10^{-3}
FT	360.568×10^{-3}	372.495×10^{-3}
CG	54.009×10^{-3}	55.270×10^{-3}

Table 9 Simulation time of programs: our system vs. dist-gem5 (seconds)

Program	Simulation time (our system)	Simulation time (dist-gem5)
MG	9.126×10^{3}	12.925×10^{3}
EP	23.225×10^{3}	56.885×10^{3}
IS	2.453×10^{3}	34.742×10^{3}
BT	5.346×10^{3}	13.668×10^{3}
SP	2.394×10^{3}	5.904×10^{3}
LU	4.122×10^{3}	7.110×10^{3}
FT	10.214×10^{3}	20.826×10^{3}
CG	1.355×10^{3}	3.468×10^{3}

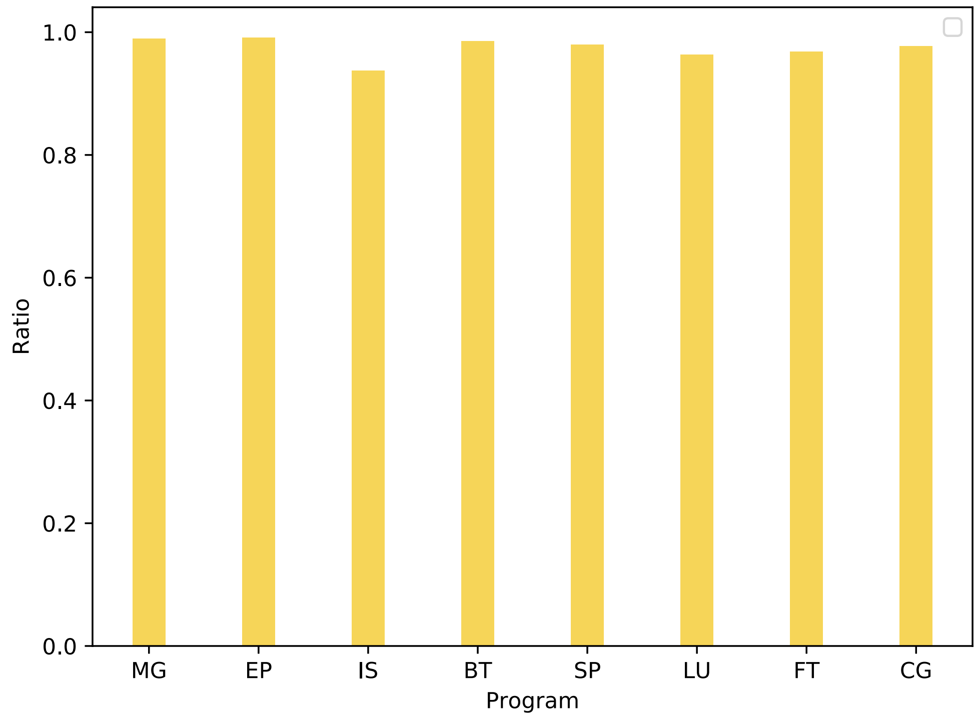


Fig. 10 The ratio of the simulated time in our system to dist-gem5

Table 10 Slowdown of programs: our system vs. dist-gem5

Program	Slowdown (our system)	Slowdown (dist-gem5)
MG	0.371×10^5	0.520×10^5
EP	0.294×10^5	0.715×10^5
IS	0.032×10^5	0.425×10^5
BT	0.389×10^5	0.980×10^5
SP	0.327×10^5	0.789×10^5
LU	0.487×10^5	0.809×10^5
FT	0.283×10^5	0.559×10^5
CG	0.251×10^5	0.627×10^5

speed of our system is significantly improved compared to dist-gem5. The biggest improvement is the program IS. This is because the communication behavior of IS mainly consists of full-exchange collective communication, which will generate a vast number of messages. The speed of the execution of this type of message function in our message emulation environment will be faster than the execution of these functions in the native MPI environment. In addition, the synchronization mechanism of dist-gem5 will further slow down the simulation speed.

We further compare the scalability of our simulation system and dist-gem5. We assume that the target system has 2048 nodes. The interconnection network is InfiniBand EDR and the topology is Fat-Tree. Each node has an ARMv8 processor with 8 cores and 8 G memory. Since the maximum number of processes supported by IS is 1024, we simulate seven programs of NPB except IS. The number of processes for MG, CG, EP, LU and FT is 2048. The number of processes for BT and SP is 2025. The program scale of CG, FT, EP, SP and BT is CLASS C. The program scale of LU and MG is CLASS D. Processes are distributed in order of process number and node number. We also selectively simulate the procedure of the first three iterations of the six programs except EP. For EP, we simulate the whole procedure.

We first run the programs in our simulation system. We select node 0 of the target system and simulate the running of processes 0 to 7. Table 11 shows the average simulated execution time of the processes and the simulation time. The

Table 11 The result and performance of simulating a target system with 2048 nodes

Program	Simulated execution time(seconds)	Simulation time (seconds)	Slowdown
MG	236.660×10^{-3}	3.213×10^3	1.358×10^5
EP	357.998×10^{-3}	7.956×10^3	0.222×10^5
BT	98.358×10^{-3}	2.772×10^3	0.282×10^5
SP	75.652×10^{-3}	1.416×10^3	0.187×10^5
LU	496.838×10^{-3}	8.766×10^3	0.176×10^5
FT	285.006×10^{-3}	4.344×10^3	0.152×10^5
CG	117.402×10^{-3}	9.894×10^3	0.843×10^5

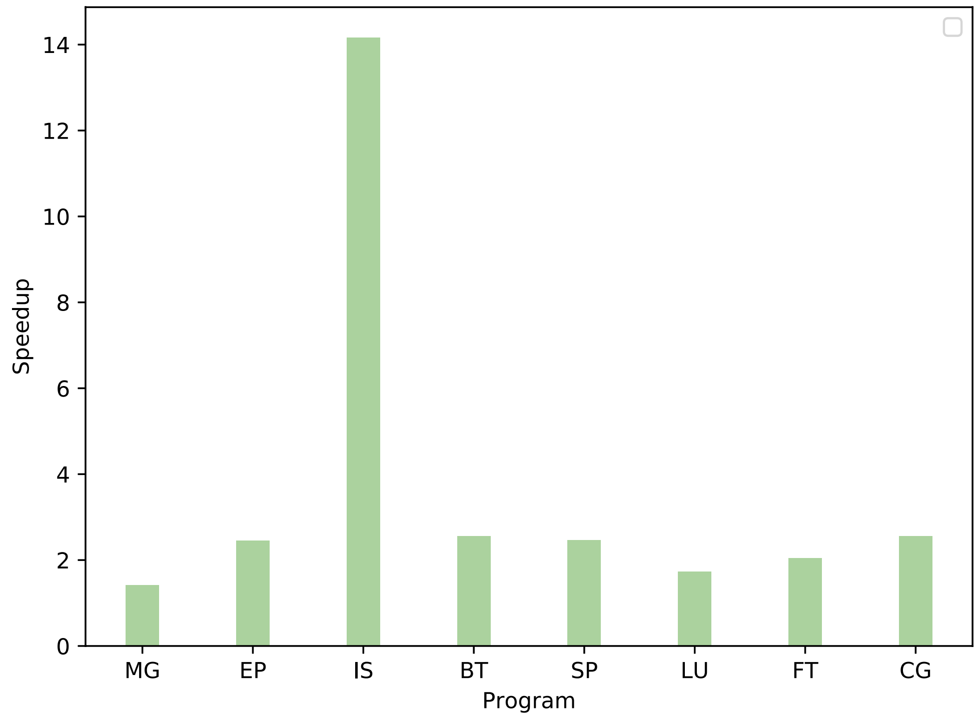


Fig. 11 The speedup of the simulation speed of our system to dist-gem5

slowdown is calculated by comparing the simulation time and the simulated execution time of each program.

From Tables 6, 10 and 11 we can conclude that as the number of nodes of the simulated target system increases, the slowdown of our system does not corresponding increase. The system's slowdown is more related to the complexity of the computing nodes and the behavior of the program. This demonstrates that our system has sufficient scalability and is capable of simulating large-scale HPC systems.

We then test dist-gem5. As mentioned above, the number of simulation instances launched by dist-gem5 is the number of nodes of the target system plus one (one switch instance), and each simulation instance occupies one processor core. Since each local host has 24 cores, 86 local hosts are required to simulate the target system using dist-gem5. Since simulating the complete target system requires a large number of hosts, we only simulate a subset of nodes of the target system. We simulate 64 nodes of the target system in three local hosts, running the program CG with 512 processes with the problem size CLASS C. Experimental results show that the simulation time of the first three iterations of the program exceeds 12 h. From the experimental result we can conclude that if we simulate all the 2048 nodes and execute the CG with 2048 processes, the simulation time will be significantly more than 12 h.

The above experiments show that the scalability of our system is suitable for simulating HPC systems. Obviously, from the perspective of resource usage and simulation speed, dist-gem5 is not suitable for simulating HPC systems.

4.5 Validation of heterogeneous systems

There are only a few execution-driven simulators for heterogeneous architectures. One of the most classic off-the-shelf heterogeneous node full-system simulators is Gem5-gpu. However, the Gem5-gpu can only support simulation of NVIDIA GTX580, and support CUDA version up to version 3.2. It is difficult for us to find a target system to verify the accuracy of our system for heterogeneous CPU-GPU nodes, so in this sub-section we only verify the functionality of our system for heterogeneous nodes.

We assume a target heterogeneous HPC system that the node has a dual-core X86 processor and an NVIDIA GTX580 GPU. The memory of the node is 4GB. The target HPC system has 300 nodes connected via a 100Gb Infiniband switch.

We use the SHOC(Scalable HeterOgeneous Computing benchmark) (Danalis et al. 2010) as the workload to evaluate our simulation system for heterogeneous architecture. There are 4 MPI+CUDA programs in SHOC, namely scan (parallel prefix sum algorithm on a large array of floating point data), reduction (a sum reduction operation using single-precision floating-point data), qtc (Quality Threshold Clustering) and stencil2d (a standard two-dimensional nine-point stencil calculation) and QTC (Quality Threshold Clustering). Since the CUDA version supported by SHOC is above 4.0, while the highest CUDA version supported by Gem5-gpu is 3.2, we have made minimal modifications to these four programs. However, only the program scan and reduction can run correctly without affecting their core procedure after removing some high version functions. Some high version functions of qtc and stencil2d are functions that implement their core procedures and cannot be removed. Ultimately, we choose scan and reduction as the workload.

Gem5-gpu only supports one process to use GPU. Therefore, we assume scan and reduction run on 128 nodes of the target system with 128 processes. Processes are assigned to node 0 to node 127 in sequence. We simulate node 0 and run process 0 in the node.

Table 12 shows the simulation result and the performance of our simulation system.

5 Related work

There is a lot of work related to simulation and simulators of computer architecture, and we selectively introduce some work closely related to our work. We introduce related work from three aspects: (1) the classic computer architecture simulator, (2) the HPC system simulator, and (3) the extensions of the Gem5 simulator.

(1) The classic computer architecture simulator

Computer architecture simulators can be classified in many ways according to different perspectives. For instance, it can be divided into full-system simulators and application-level simulators based on the scope of the target. The full-system simulator can simulate the complete operating system, and the application level simulator can only run target applications. Some classic computer architecture simulators support both full-system simulation and application-level simulation.

Several simulators support full-system simulation. Gem5 is a simulation framework that integrates the advantages of both the simulator M5, which models networked systems, and the Gems, which is a simulation toolset for multiprocessors. It covers the simulation of system-level architectures

Table 12 The result and performance of simulating the heterogeneous system

Program	Simulated execution time (seconds)	Simulation time (seconds)	Slowdown
Scan	9.831	34.560×10^3	3.515×10^3
Reduction	9.056	51.120×10^3	5.644×10^3

as well as processor microarchitectures. Gem5 provides four interpretation-based CPU models: a simple one-CPI CPU, a detailed model of an in-order CPU, and a detailed model of an out-of-order CPU. Gem5 supports mainstream instruction set architectures such as X86, Alpha, ARM, SPARC, MIPS, etc. Gem5 is one of the most classic and commonly used computer architecture simulators. It is developed in C++ and Python. C++ code implements the core simulation components, and Python implements user interfaces, so that users can flexibly create and configure target simulation objects.

SimOS (Rosenblum et al. 1997) is an early commonly used full-system simulator. It is enabled to run IRIX on MIPS, and Unix on Alpha. Simics (Magnusson et al. 2002) is a full-system simulation platform; it supports numerous instruction sets and can directly run unmodified operating systems. What is special is that Simics can implement execution in a backward direction to help analyze how program errors and exceptions occur.

SimpleScalar (Austin et al. 2002) is a classic simulation toolset and provides multiple simulation modes, of which ss-os is a full-system simulator. PTLsim (Yourst 2007) is a full-system simulator that models a modern superscalar out-of-order x86-64 processor core and only targets the real commercially available x86 ISA. MARSSx86 (Patel et al. 2011) is an x86 full-system simulator that is based on PTLsim. It adds various optimizations for better performance and flexibility than PTLsim, and adds supporting in-order (IO) pipeline models.

Gem5-gpu is a classic CPU-GPU heterogeneous system architecture simulator. It integrates gem5 with gpgpu-sim and can simulate the heterogeneous system with the full-system mode. Gem5-gpu adds a memory interface, which is implemented by modeling CU memory accesses through Ruby, between the Gem5 and gpgpu-sim.

If the researcher does not care about the interaction of the application with the operating system, the application-level simulator is also useful. Some simulation platforms also provide application-level simulators, such as Gem5, SimpleScalar, Gem5-gpu and SimpleScalar. Multi2Sim (Ubal et al. 2012) is an application-level simulator that targets CPU-GPU architectures. It supports multi-threaded or single-threaded processor cores with an out-of-order pipeline. Sniper is an application-level parallel multi-core simulator that supports both out-of-order and in-order pipeline simulation.

For the application-level simulators, in addition to simulating the entire system, there are many simulators that simulate a component, such as memory simulator, accelerator simulator, etc. For example, DRAMSim2 (Rosenfeld et al. 2011) is a memory system simulator that can obtain various static and dynamic parameters of the memory system during the program operation, including memory access delay, memory bandwidth, memory power consumption, and memory controller scheduling. GPGPU-Sim is a GPGPU simulator. The GPU model of GPGPU-Sim consists of a series of SIMT cores. These SIMT cores are connected to the memory partition through an interconnection network to communicate with the GDDR memory.

(2) The HPC system simulator

Bigsim (Zheng et al. 2004) is a trace-driven simulator that simulates processes through threads. The target application runs on an emulator and generates communication traces, and a trace-based simulator takes these traces as input to predict the overall performance of the target system. Bigsim relies on the CHARM++ (Kale and Krishnan 1993) environment.

LogGOPSim (Hoefler et al. 2010) and SIM-MPI (Zhai et al. 2010, 2015) are trace-driven HPC simulators. They model the traces and predict the overall performance of the target system. The input of LogGOPSim is expressed by Group Operation Assembly Language (GOAL), which is used to define the three different types of tasks: send, receive, and computation. LogGOPSim uses the LogGOP model to simulate the send and receive traces to predict the communication time. The input of SIM-MPI is composed of the computation time sequence and the MPI operation traces. SIM-MPI uses the LogGPO model to simulate the computation traces. To obtain more accurate traces related to the computation time, the two simulators usually collect the computation trace on local nodes that are the same as the target system.

Denzel et al. (2010) presented a trace-driven HPC simulation framework. Similar to LogGOPSim and SIM-MPI, the simulation framework models the computation and communication traces to predict the overall performance of the target system. It consists of the computing node model and the interconnection network model. The computing node model simulates the computation traces. The interconnection network model that is based on Omnet++ simulates the communication traces.

The Structural Simulation Toolkit (SST) (Rodrigues et al. 2011) is a parallel simulation framework that was developed to explore innovations in highly concurrent systems. It supports integration of different simulation components by providing a set of common services and interfaces. The implementation of SST is based on MPI. The components run as processes on different physic processor cores to simulate concurrently and communicate with other components through MPI communication. This provides a high level of performance. BE-SST (Ramaswamy et al. 2018) is an execution-driven simulator for behavioral emulation of extreme-scale systems. It incorporates BE (behavioral emulation models) into the SST. It mainly utilizes the interfaces provided by SST for defining processor BE objects and communication BE objects, resulting in a parallel and scalable coarse-grained simulator. Hsieh et al. (2012) presented a scalable execution-driven simulation infrastructure for HPC system by integrating the Gem5 into the SST. The simulation infrastructure makes all Gem5 SimObjects live inside an SST component and modifies the Gem5 event queue to be driven by the SST event queue.

SystemC (Panda 2001) is a library that provides an event-driven simulation interface. It provides a set of C++ classes and macros to model systems composed of both hardware and software components. Some works (Galiano et al. 2009; Ziyu et al. 2009) implement parallel simulation based on systemC and MPI. They use systemC to model processors and run one logical process on one processor core. The synchronization and communication between the logical processes are implemented based on MPI.

MPI-SIM (Prakash and Bagrodia 1998) is an execution-driven simulator that provides a capability for multithreaded execution of MPI programs. It reprocesses the program to convert each MPI call to an MPI-SIM call. MPI-SIM is mainly used to predict the performance of MPI programs. The researchers can set the number of processors and communication latencies of the target system. MPI-SIM does not support modeling and simulation of the processor microarchitecture and the interconnection network.

SimHPC (Liu et al. 2013), PS-SIM (Guo et al. 2011) and VACED-SIM (Lin et al. 2013) are execution-driven HPC simulators. These simulators run the MPI program with the desired scale on a local cluster with fewer processors than the target system. They use local hosts and an interconnection network model to simulate the target system. During the execution of the program, they simulate the computation and communication time of the processes on target system. They have different methods to simulate the computation and communication time. SimHPC captures the process scheduling events to simulate the running time of the process and uses a mathematical model to simulate the message transport latency. PS-SIM uses a logical process mapping mechanism to simulate the computation time and a communication model based on LogP (Culler et al. 1993) model to simulate the communication time. VACED-SIM assumes that the number of processor cores of the local hosts is greater than or equal to the number of processes in the target program. Under this assumption, each process can be allocated a processor core, so that the computation time of processes is acquired by measurement directly. VACED-SIM also uses a communication model to simulate the communication time of the processes. To make the modeling of the microarchitecture more accurate and to simplify the design of the microarchitecture simulation, these simulators usually use the nodes that are the same as the target system as local hosts.

Riesen (2006) presented a hybrid MPI simulator. The program runs on computing nodes and its MPI communication calls are re-direct to a trace-driven network simulator. The network simulator analyzes the MPI communication traces to simulate network delay. Then the network simulator feeds back the network delay to the running application. Similar to VACED-SIM, the simulator uses the nodes are the same as the target system, and assumes the number of processor cores of the local hosts is greater than or equal to the number of processes in the target program.

Compared with the work introduced above, our system mainly has the following advantages: (1) Our system only requires one local host to simulate an HPC system and simulate the execution of large-scale parallel programs. And our simulation system has sufficient scalability. (2) Our system can reproduce the scenario of large-scale parallel programs running on any computing node of an HPC system, including the same hardware architecture of the computing node, the same operating system and MPI library, and the approximate behavior of processes. (3) Our system can support the simulation of large-scale heterogeneous parallel systems. More specifically, in this paper our system demonstrates support for large-scale CPU-GPU parallel systems and can run MPI+CUDA parallel programs.

(3) The extensions of Gem5

Gem5 is one of the most classic and commonly used simulators, there is much work to extend Gem5. Dist-gem5 (Mohammad et al. 2017) is a simulator of the distributed system based on Gem5. It connects an Ethernet switch simulation instance with multiple gem5 node full-system simulation instances by the linking objects. Each simulation node transmits data through the Ethernet switch simulation. COSSIM (Tampouratzis et al. 2020) is also a simulator of the distributed system with a similar design idea to Dist-gem5. COSSIM connects the gem5 node instance with the Omnet++ simulation instance by developing an interface between the gem5 and Omnet++. gem5-X (Qureshi et al. 2021) is a simulation framework for architectural exploration of heterogeneous many-core systems. It supports heterogeneous architecture simulation allowing both in-order and out-of-cores cores to be simulated simultaneously.Gem5v (Nikounia and Mohammadi 2015) is a modified gem5 that can simulate simulates the behavior of the hypervisor and can simulate virtual machines. It modifies Ruby to support the translation of physical addresses to real addresses. Each VM is simulated by an instance. All VM instances are connected to one instance of Ruby.

6 Conclusion and future work

This paper demonstrates a novel approach for simulating the HPC system with node simulation using the architecture simulator and realizes a simulation system based on the classic simulator Gem5. In our system, researchers can flexibly model and configure the hardware and software architecture of the target node as well as the target interconnection network. Compared to existing HPC simulators, researchers can not only perform detailed and accurate simulation of the node of target HPC system, but also can evaluate the overall performance of the target system by running large-scale parallel programs in our simulation system. In addition, researchers can comprehensively evaluate the interaction and integration between the application, the software environment and the hardware architecture through our simulation system.

In the future, we will improve the simulation system mainly by three aspects. (1) We will simulate the procedure of message transportation in more detail, mainly including researching the approach to simulate the data dependency among the processes, which will make the simulation system more accurate. (2) We will enhance the support for heterogeneous architecture. (3) We will research how to simulate the parallel file I/O (e.g. *MPI_File_read_at()* and *MPI_File_write_at())* in our simulation system.

Data availability All relevant data are within the manuscript.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no confict of interest.

References

Austin, T., Larson, E., Ernst, D.: Simplescalar: An infrastructure for computer system modeling. Computer **35**(2), 59–67 (2002)

Bakhoda, A., Yuan, G.L., Fung, W.W., et al.: Analyzing cuda workloads using a detailed gpu simulator. IEEE, 2009 IEEE international symposium on performance analysis of systems and software, pp 163–174 (2009)

Binkert, N., Beckmann, B., Black, G., et al.: The gem5 simulator. ACM Sigarch Comput Archit News **39**(2), 1–7 (2011)

Binkert, N.L., Dreslinski, R.G., Hsu, L.R., et al.: The m5 simulator: Modeling networked systems. IEEE Micro **26**(4), 52–60 (2006)

Culler, D., Karp, R., Patterson, D., et al: Logp: Towards a realistic model of parallel computation. Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming, pp 1–12 (1993)

Danalis, A., Marin, G., McCurdy, C., et al: The scalable heterogeneous computing (shoc) benchmark suite. Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp 63–74 (2010)

Denzel, W.E., Li, J., Walker, P., et al.: A framework for end-to-end simulation of high-performance computing systems. SIMULATION **86**(5–6), 331–350 (2010)

Galiano, V., Migallón, H., Pérez-Caparrós, D., et al: Distributing systemc structures in parallel simulations. Proceedings of the 2009 Spring Simulation Multiconference, pp 1–8 (2009)

Guo, X., Lin, Y., Xu, X., et al: Ps-sim: An execution-driven performance simulation technology based on process-switch. Springer, International Conference on Computer Science, Environment, Ecoinformatics, and Education, pp 15–22 (2011)

Hoefler, T., Schneider, T., Lumsdaine, A.: Loggopsim: simulating large-scale applications in the loggops model. Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pp 597–604 (2010)

Hsieh, M., Pedretti, K., Meng, J., et al: Sst+ gem5= a scalable simulation infrastructure for high performance computing. Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques, pp 196–201 (2012)

Kale, L.V., Krishnan, S.: Charm++ a portable concurrent object oriented system based on c++. Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications, pp 91–108 (1993)

Lin, Y., Yang, X., Xu, X., et al.: Vaced-sim: A simulator for scalability prediction in large-scale parallel computing. IEICE T Inf Syst **96**(7), 1430–1442 (2013)

Liu, Y., Zhi, Y.Z., Zhang, X., et al: Simhpc: An execution-driven simulator for high-performance computers. Jisuanji Xuebao(Chinese Journal of Computers) 36(4):738–746 (2013)

Magnusson, P.S., Christensson, M., Eskilson, J., et al.: Simics: A full system simulation platform. Computer **35**(2), 50–58 (2002)

Martin, M.M., Sorin, D.J., Beckmann, B.M., et al.: Multifacet's general execution-driven multiprocessor simulator (gems) toolset. ACM Sigarch Comput Archit News **33**(4), 92–99 (2005)

Mohammad, A., Darbaz, U., Dozsa, G., et al: dist-gem5: Distributed simulation of computer clusters. IEEE, 2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp 153–162 (2017)

MPICH (2023) MPICH homepage, [online]. https://www.mpich.org/

Nikounia, S.H., Mohammadi, S.: Gem5v: a modified gem5 for simulating virtualized systems. J Supercomput **71**(4), 1484–1504 (2015)

NPB (2023) THE NAS PARALLEL BENCHMARKS, [online]. https://www.nas.nasa.gov/publications/npb.html

Panda, P.R.: Systemc: a modeling platform supporting multiple design abstractions. Proceedings of the 14th international symposium on Systems synthesis, pp 75–80 (2001)

Patel, A., Afram, F., Ghose, K.: Marss-x86: A qemu-based microarchitectural and systems simulator for x86 multicore processors. Citeseer, 1st International Qemu Users' Forum, pp 29–30 (2011)

Power, J., Hestness, J., Orr, M.S., et al.: gem5-gpu: A heterogeneous cpu-gpu simulator. IEEE Comput Archit L **14**(1), 34–36 (2014)

Prakash, S., Bagrodia, R.L.: Mpi-sim: using parallel simulation to evaluate mpi programs. IEEE, pp 467–474 (1998)

Qureshi, Y.M., Simon, W.A., Zapater, M., et al.: Gem5-x: A many-core heterogeneous simulation platform for architectural exploration and optimization. ACM T Archit Code Op (TACO) **18**(4), 1–27 (2021)

Ramaswamy, A., Kumar, N., Neelakantan, A., et al: Scalable behavioral emulation of extreme-scale systems using structural simulation toolkit. Proceedings of the 47th International Conference on Parallel Processing, pp 1–11 (2018)

Riesen, R.:A hybrid mpi simulator. IEEE, 2006 IEEE international conference on cluster computing, pp 1–9 (2006)

Rodrigues, A.F., Hemmert, K.S., Barrett, B.W., et al.: The structural simulation toolkit. ACM SIGMETRICS Performance Evaluation Review **38**(4), 37–42 (2011)

Rosenblum, M., Bugnion, E., Devine, S., et al.: Using the simos machine simulator to study complex computer systems. ACM T Model Comput S (TOMACS) **7**(1), 78–103 (1997)

Rosenfeld, P., Cooper-Balis, E., Jacob, B.: Dramsim2: A cycle accurate memory system simulator. IEEE Comput Archit L **10**(1), 16–19 (2011)

Tampouratzis, N., Papaefstathiou, I., Nikitakis, A., et al.: A novel, highly integrated simulator for parallel and distributed systems. ACM T Archit Code Op (TACO) **17**(1), 1–28 (2020)

Ubal, R., Jang, B., Mistry, P., et al: Multi2sim: A simulation framework for cpu-gpu computing. IEEE, 2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT), pp 335–344 (2012)

Varga, A., Hornig, R.: An overview of the omnet++ simulation environment. Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, pp 1–10 (2008)

Yourst, M.T.: Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator. IEEE, 2007 IEEE International Symposium on Performance Analysis of Systems & Software, pp 23–34 (2007)

Zhai, J., Chen, W., Zheng, W.: Phantom: predicting performance of parallel applications on large-scale parallel machines using a single node. Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp 305–314 (2010)

Zhai, J., Chen, W., Zheng, W., et al.: Performance prediction for large-scale parallel applications using representative replay. IEEE Trans. Comput. **65**(7), 2184–2198 (2015)

Zheng, G., Kakulapati, G., Kalé, L.V.: Bigsim: A parallel simulator for performance prediction of extremely large parallel machines. IEEE, 18th International Parallel and Distributed Processing Symposium, 2004. Proceedings., p 78 (2004)

Ziyu, H., Lei, Q., Hongliang, L., et al: A parallel systemc environment: Archsc. IEEE, 2009 15th International Conference on Parallel and Distributed Systems, pp 617–623 (2009)

Fang Lin received the master's degree in computer science and technology from the Beijing University of Posts and Telecommunications (BUPT), in 2013. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Beihang University(BUAA). His research interests include high-performance computing, distributed computing, and computer architecture.

Yi Liu is a professor in School of Computer Science and Engineering, and Director of the Sino-German Joint Software Institute (JSI) at Beihang University, China. In 2000, he completed Ph.D in Department of Computer Science of Xi'an Jiaotong University. His research interests include computer architecture, HPC and new generation of network technology.

Xin Wang received the bachelor's degree from Chongqing University in 2018 and the master's degree from Beihang University in 2021. His research interest is high-performance computing and computer architecture.

Xueyan Gai received the bachelor's degree from Jilin University in 2020 and the master's degree from Beihang University in 2023. Her research interest is high-performance computing and computer architecture.