IEEE Transactions on Quantum Engineering

Received 27 August 2025; revised 9 January 2026; accepted 11 January 2026; date of publication 15 January 2026; date of current version 5 February 2026.

Digital Object Identifier 10.1109/TQE.2026.3654543

Improving Decision Diagram-Based Quantum Circuit Simulation Using Static Variable Ordering and Multinode Ring Communication

YUSUKE KIMURA[1,2], SHAOWEN LI[3], HIROYUKI SATO[3] (Member, IEEE), MASAHIRO FUJITA[3,4] (Life Member, IEEE), AND ROBERT WILLE[1,5] (Senior Member, IEEE)

[1]Technical University of Munich, 80333 Munich, Germany
[2]Fujitsu Limited, Kawasaki 211-8588, Japan
[3]The University of Tokyo, Tokyo 113-8658, Japan
[4]National Institute of Advanced Industrial Science and Technology, Tokyo 113-8658, Japan
[5]Munich Quantum Software Company, Garching near Munich 85748 Munich, Germany

Corresponding author: Yusuke Kimura (e-mail: yusuke.kimura@tum.de).

ABSTRACT Currently, the development of quantum computers is active; however, large-scale machines remain limited and noisy. Furthermore, such quantum computers do not allow direct access to state vectors, posing challenges for quantum algorithm development. Quantum circuit simulators on classical computers offer a solution, with decision diagram (DD)-based simulators being particularly memory-efficient for representing quantum states. However, DD-based simulation still requires optimization, especially concerning variable ordering and multinode parallelization. Processing time for DDs heavily depends on variable order, but existing static variable ordering methods did not have general applicability. The prior multithreaded DD simulations showed slowdowns. This study proposes two techniques to address these issues. The first is a scoring-based heuristic static variable ordering method that analyzes an input quantum circuit, such as the distribution of parameterized rotations and multibit gates, to suppress the growth of graph nodes and amount of computation. The second is a ring communication approach that reduces communication overhead when performing parallel simulations across multiple computing nodes. Parallel computation requires data exchange between nodes, which results in additional communication time. The proposed ring communication method can eliminate broadcast communication, thereby reducing the waiting time until communication is completed. Evaluations using various benchmark circuits demonstrate that the proposed ordering achieves up to a 4.5 speedup in a single-node environment. Furthermore, the ring communication exhibits superior scalability, achieving up to an 11× speedup with 16 computing nodes. The proposed variable ordering method generally reduces the overall simulation time in multinode environments.

INDEX TERMS Decision diagram (DD), parallel computation, quantum circuit simulation, ring communication, static variable order.

I. INTRODUCTION

Recent research and development in quantum computing has been remarkable, with processors exceeding hundreds to thousands of qubits [2]. However, these devices have not yet implemented large-scale quantum error correction and are susceptible to noise. To correctly execute practical quantum algorithms, a single logical qubit may require thousands or tens of thousands of physical qubits [3]. Thus, a situation in which many people can freely use large-scale, ideal quantum computers remains far from reality. Moreover, it is impossible to directly access the state vectors (SV) in real quantum computers, which is inconvenient for the development of algorithms.

Given this background, quantum circuit simulators that mimic the behavior of quantum computers on classical computers play a crucial role in the research and development of quantum computing algorithms. Several types of quantum circuit simulators exist, the most common and intuitive of

which is the explicit SV-based simulator. It allocates a 2^N-length complex vector in memory for N qubits [4], [5]. Although this method is relatively easy to implement and can be accelerated by leveraging graphics processing units (GPUs) for parallelization, the exponentially increasing memory usage with the number of qubits poses a severe constraint. Even with supercomputers, the number of simulatable qubits is at most approximately 40–50 [6], [7]. To mitigate this memory problem, simulators based on tensor networks [8], [9] and matrix product states (MPS) [10], [11], [12] have been proposed, which perform well under specific conditions, such as circuits with low entanglement levels.

This study focuses on a different approach: decision diagram (DD)-based quantum circuit simulation. DD-based simulators represent quantum states and gates using a graph structure called a DD [13], [14], [15], [16]. This representation method reduces memory usage by sharing graph nodes when common subvectors or sparsity (many zero-value elements) exist within the SV. Consequently, for certain types of quantum algorithms, such as Shor's and Grover's algorithms, DD is known to outperform other types of simulators [17], [18].

However, DD-based simulators have not yet fully utilized the methods used in similar studies, and further optimization is possible. The first problem is the variable order. Historically, the size and processing time of DDs depend heavily on variable ordering, that is, which variable (qubit) is assigned to which level of the graph [19]. In quantum circuit simulations, dynamic reordering has been reported to worsen the simulation time and numerical accuracy [20]. Although an initial study observed the effects of static variable orders in small circuits [21], no method has been proposed to determine a static variable order that is applicable to a wide range of quantum algorithms. The second limitation is the difficulty of parallelization. Existing multithreading research [22] found that DD-based simulators became even slower for Shor's and Grover's circuits. Moreover, little research has been conducted on multinode environments in which multiple computing nodes are connected to high-speed wires.

Therefore, this study proposes methods for significantly improving the performance of DD-based quantum circuit simulators to address the two challenges mentioned above. To address the first challenge, we propose a heuristic static variable ordering method that analyzes the characteristics of quantum circuits and determines a good variable ordering based on scoring. For the second challenge, we proposed a multinode parallelization method that introduces ring communication to minimize the communication overhead between computing nodes.

Experiments have been conducted using multiple benchmark circuits to evaluate the effectiveness of the proposed method. As a result, we have confirmed that the proposed static variable ordering method can achieve speedups of a 4.5× speedup compared to the given original orders. Furthermore, ring communication achieves 11× faster simulation with 16 computing nodes compared with 1-node, and achieves a simulation speed approximately 2.8 times faster than that of the existing broadcast communication. We have also experimentally demonstrate that our variable ordering method, which does not directly consider the multinode setting, reduces the overall simulation time even in a multinode environment.

The rest of this article is organized as follows. Section II introduces background knowledge on quantum simulations and DDs. Section III summarizes the issues with DD-based simulators and explains the motivation behind the proposed methods. Section IV describes the proposed static variable ordering, and Section V describes ring communication. In the next section, we prepare multiple benchmark circuits to compare our proposed methods with existing methods and present the results and their interpretations. Finally, Section VII concludes this article.

II. BACKGROUND AND EXISTING RESEARCH

This section details the background knowledge essential for understanding the proposed methods and the related existing research. First, we outline the basic concepts of quantum computation and compare quantum circuit simulation methods. Finally, we focus on a DD-based simulation, which is the subject of this study.

A. QUANTUM COMPUTING, CIRCUIT, STATE, AND GATE

Quantum computation uses "qubits," which can have a superposition of "0 state" and "1 state." The N-qubits quantum state is described by a complex vector of length 2^N. Each element of this vector corresponds to a state from $|00\ldots0\rangle$ to $|11\ldots1\rangle$, and the square of the absolute value of each element represents the probability of observing the corresponding state.

Operations in a quantum computer are represented by "quantum gates." Quantum gates are unitary matrices that act on quantum states by transforming vectors into other vectors. A "quantum circuit" is a sequence of these gates.

Quantum circuit simulators mimic the behavior of quantum circuits on classical computers. Specifically, an initial state (typically an SV corresponding to state $|00\ldots0\rangle$ where all qubits are zero) is prepared. Next, the product of the unitary matrix corresponding to the first gate and initial SV is calculated. The resulting new SV is then applied by the unitary matrix corresponding to the second gate, and the matrix–vector product is calculated again. This process is repeated sequentially until the final gate.

Therefore, a quantum circuit simulation is a series of matrix–vector products used to obtain the final quantum state.

B. TYPES OF QUANTUM CIRCUIT SIMULATION

Quantum circuit simulators can be broadly categorized into several types based on their internal data representation and

computational methods. Each has its advantages and disadvantages and should be selected according to the characteristics of the simulated circuit.

1) EXPLICIT SV-BASED SIMULATOR

As described in the introduction, this type of simulator explicitly allocates a 2^N-length complex vector to the memory for N qubits [4], [5]. The main advantages are the simplicity of implementation and ease of parallelization. Matrix–vector multiplication can easily be accelerated by GPUs or by distributing data across multiple computing nodes [6], [7]. However, it suffers from the problem of exponentially increasing memory as the number of qubits increases.

In quantum circuit simulations, to maintain accuracy, a complex number is typically represented by double-precision floating-point numbers (8 bytes) for its real and imaginary parts. Therefore, $8 \times 2 = 16$ bytes of memory are required per complex number. An N-qubit SV has 2^N complex elements; therefore, the total memory required for the simulation is 16×2^N bytes.

1) 30 qubits: 16×2^{30} bytes $= 16$ GB.
2) 40 qubits: 16×2^{40} bytes $= 16$ TB.
3) 50 qubits: 16×2^{50} bytes $= 16$ PB.

Although a typical PC can easily handle 30 qubits, 16 TB of memory for 40 qubits can no longer be installed on a single computer. This exponential increase in memory requirements limits the scalability of SV simulators.

2) MPS-BASED SIMULATOR

An MPS-based simulator was proposed to overcome the aforementioned memory problem [10], [11], [12]. In this method, an SV of length 2^N is represented as the product of smaller matrices.

The advantage of MPS is that it can efficiently represent quantum states with very few parameters when there is little entanglement between qubits. This allows for significant memory reduction and faster computation, especially in simulating systems with 1-D structures or shallow circuits, such as variational quantum algorithms. However, when entanglement is spread throughout the system or as the circuits deepen, the size of the matrices (bond dimension) increases, and the computational cost increases.

Although MPS simulators are potent tools for specific types of problems, this study primarily focuses on DDs; therefore, we do not discuss MPS simulators in further detail.

3) DD-BASED SIMULATOR

DD simulators are the central focus of this study [13], [14], [15], [16]. These simulators use DDs to represent quantum states (vectors) and gates (matrices). When common substructures exist within the SV (e.g., a subvector is a constant multiple of another subvector) or when many elements are zero (sparse), the DDs can share graph nodes.

This node-sharing mechanism allows the DD to reduce memory usage and achieve faster simulations than SV simulators for specific quantum circuits (e.g., Shor's factoring algorithm) [17], [18].

Section II-C explains the DD-based quantum circuit simulation details.

The following are recent studies related to DD and quantum circuit simulations. Zulehner and Wille [23] focused on the fact that DD-based simulators can efficiently perform matrix–matrix multiplication and showed that simulations can be accelerated by combining some quantum gates. FlatDD [24] is a simulator that combines DD and SV techniques. DD is used to represent SV when the number of graph nodes is small, and SV is used when the number of nodes increases and DD computation becomes slow. BQSim [25] is a quantum circuit simulator that uses GPUs. The computation was accelerated using DD only for the quantum gates. Jaques and Häner [26] accelerates simulations by focusing on the sparsity of the SV, although this is not a DD-based simulation. It successfully and rapidly simulates the subproblems of factorization, allowing for the handling of 20-bit integers.

C. DD AND QUANTUM CIRCUIT SIMULATION

1) DECISION DIAGRAM

DDs were originally conceived as data structures to efficiently represent and manipulate Boolean functions [19], [27]. DDs provide a unique (canonical) representation of a given function when the variable order is fixed; thus, they have been widely used for logical circuit equivalence checking and analysis. It is also known that the size of a DD, that is, the number of graph nodes, can change significantly depending on the order of the variables [28], [29], [30], [31].

2) QUANTUM DD (QDD)

The type of DD used in quantum circuit simulations is called a QDD [13], [14], [15], [16]. The main feature of QDDs is that each edge of a graph has complex weights. The use of edge weights allows complex vectors and matrices to be represented using fewer graph nodes.

a) SV representation

QDD uses a binary graph to represent the SV. The values in the SV are determined by traversing the branches according to the binary index of the SV. Starting from the root node, if the first bit of the index is zero, it proceeds to the left child node; if it is one, it proceeds to the right child node. This process was repeated until the final index bit was obtained. The single path determined in this manner corresponds to each index of the SV, and the product of all edge weights on that path is the value of the corresponding vector element. If the edge weights are not explicitly stated in the diagram, they are assumed to be equal to one.

Example: Consider the SV shown in Fig. 1. To find the fourth value from DD, we used 3 (counting from 0, binary representation: 011) as the index. In this case, we followed

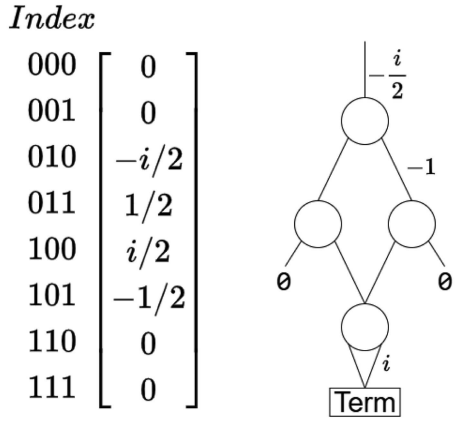

FIGURE 1. SV in the DD.

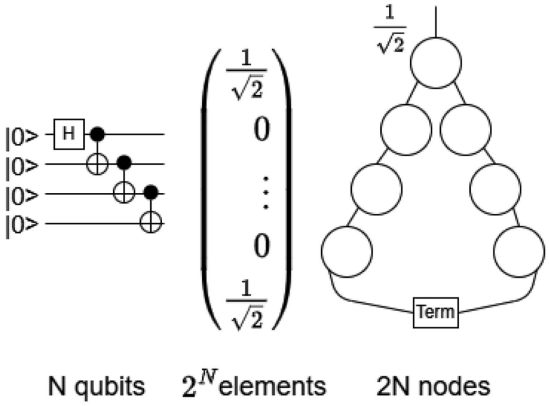

FIGURE 2. DD memory compression.

the edges from the top in the order of left, right, and right. The edge values are $(-\frac{i}{2}, 1, 1, i)$; thus, the product is $\frac{1}{2}$. Note that one bottom node is pointed to by two middle nodes that correspond to the sharing of subgraphs.

Fig. 2 shows an example of an N-qubit GHZ circuit, where an SV simulator requires a complex vector of length 2^N, whereas a DD simulator can represent the quantum state with $2N$ graph nodes. Thus, DDs can significantly reduce memory usage.

b) Quantum gate representation

When representing a quantum gate (matrix), the QDD becomes a graph with four child nodes, as shown in Fig. 3. A $2^N \times 2^N$ matrix acting on N qubits can be decomposed into four $2^{N-1} \times 2^{N-1}$ submatrices as follows:

$$U = \begin{pmatrix} U_{00} & U_{01} \\ U_{10} & U_{11} \end{pmatrix}. \tag{1}$$

Because the child nodes in the decision graph correspond to the four submatrices U_{00}, U_{01}, U_{10}, and U_{11}, respectively, one node has branches to four child nodes. This structure allows the entire matrix to be represented in a recursive manner.

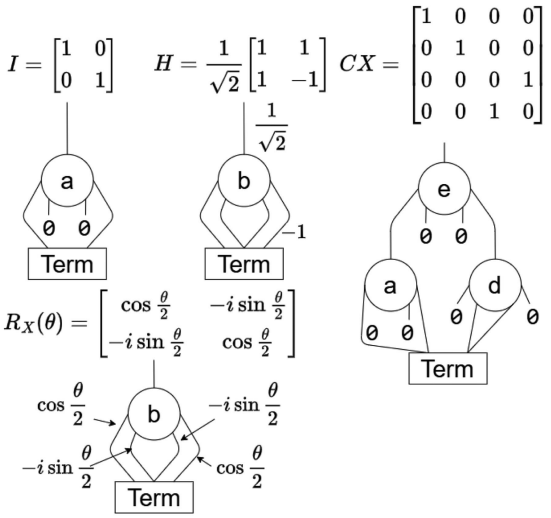

FIGURE 3. Quantum gates in the DD.

c) Matrix–vector multiplication

The matrix–vector product is obtained by recursively traversing graphs. The calculation began with the root nodes of both DDs. The definition of the matrix–vector product $v' = Uv$ can be expanded using submatrices and subvectors as follows:

$$\begin{aligned} \begin{pmatrix} v'_0 \\ v'_1 \end{pmatrix} &= \begin{pmatrix} U_{00}\ U_{01} \\ U_{10}\ U_{11} \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \end{pmatrix} \\ &= \begin{pmatrix} U_{00}v_0 + U_{01}v_1 \\ U_{10}v_0 + U_{11}v_1 \end{pmatrix}. \end{aligned} \tag{2}$$

This equation corresponds to the recursive structure of the DD. Specifically, to construct the left child node v'_0, Multiply($U00, v_0$) and Multiply(U_{01}, v_1) are calculated recursively, and the two resulting DDs are added (Add). The right child node v'_1 is calculated in a similar manner. This recursive process continues until the end of the graph (terminal nodes) is reached. Finally, a DD representing the new SV is constructed from the bottom up. Because the same calculations (products of the same gate node and state node pairs) can appear repeatedly during the calculation (resulting in a shared graph), it is essential to cache the results to eliminate computational redundancy and significantly improve efficiency. The details are shown in Code 1.

Thus far, we have explained how to represent vectors and matrices using DDs and how to calculate matrix–vector products. Quantum-circuit simulations can be performed using these methods. Fig. 4 illustrates the flow of DD-based simulations.

Example: Fig. 4 shows an example of a quantum circuit simulation. The changes in the DD are depicted in (1)–(3).

d) Unique table

The memory efficiency and canonicity of DDs are supported by a hash table called "Unique Table." The role of this table is to prevent the generation of multiple nodes with the same

Code 1: DD Multiplcation.

```
Input:
  m: A DD representing the matrix
  v: A DD representing the {vector}
Output:
  result: The resulting {vector} DD

Function Multiply(m, v):
  // If both are terminal, multiply
  weights
  if isTerminal(m.node) and
  isTerminal(v.node) then
    return DD(m.weight * v.weight,
    Terminal)
  end if

  //Check if the result is in cache or
  not
  if Cache contains key (m.node, v.node)
  then
    node = retrieve (m.node, v.node)
    from Cache
  else
    //Recursion: Calculate subproblems
    l=Add(Multiply(m.child[0],
       v.child[0]),
      Multiply(m.child[1], v.child[1]))
    r=Add(Multiply(m.child[2],
    v.child[0]),
      Multiply(m.child[3], v.child[1]))

    //Search the same node or create a
    new
    node= search_or_create(left=l,
    right=r)
    //Store the result in the cache
    store node in Cache (m.node, v.node)
  end if

  return DD(m.weight * v.weight, node)
```

information, thereby ensuring that all nodes are unique. The behavior of a unique table when creating a new node is described as follows.

1) *Hash value calculation:* When a new node needs to be generated during simulation, a hash value is first calculated using a hash function from the information that the node should possess, i.e., "pointers to child nodes" and "edge weights."
2) *Identification of the corresponding bucket:* Access a specific location (bucket) in the unique table using the calculated hash value as a key. Because hash functions can output the same hash value for different inputs, this location may contain multiple nodes with different information.
3) *Search for a matching node:* Compare each node's information in the list with the information of the node to be created.

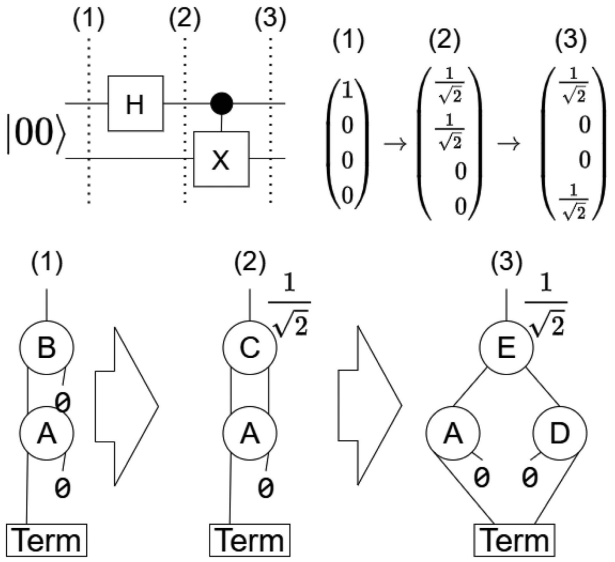

FIGURE 4. DD-based quantum simulation.

1) *If the same node is found:* No new node is created, and the pointer to that existing node is reused.
2) *If not found:* The new node information is added to the corresponding bucket in the hash table.

This strict comparison and registration process ensures that the same subgraph corresponds to one node. Therefore, DD can achieve memory reduction.

III. MOTIVATION

A. VARIABLE ORDERING

DDs have long been used to represent classical circuits, and it has been noted that the variable order can significantly change the size of a graph, which can affect the operating speed [19]. Variable ordering can be categorized as dynamic [28], [32] or static [29], [30], [31]. Combining both types can reduce execution time and memory usage.

In terms of quantum circuit simulations, our initial experiments showed that the number of DD nodes varied depending on the variable order. Fig. 5 shows DDs of the SV using two different static variable orders. Although the same SV was represented, the upper figure used 95 nodes, whereas the lower figure used 67 nodes, indicating a reduction of about 30% .

Variable order methods in quantum circuit simulations can also be classified as dynamic or static. Hillmich et al. [20] reported the effect of dynamic reordering on quantum circuit simulations. The simulation time and numerical errors increased with dynamic reordering. There are few studies on static variable orders; however, Van Schaick and Kent [21] simulated and discussed all variable orders for ten sample four-qubit circuits. The problem with this research is that the experiments were conducted with only small circuits, and

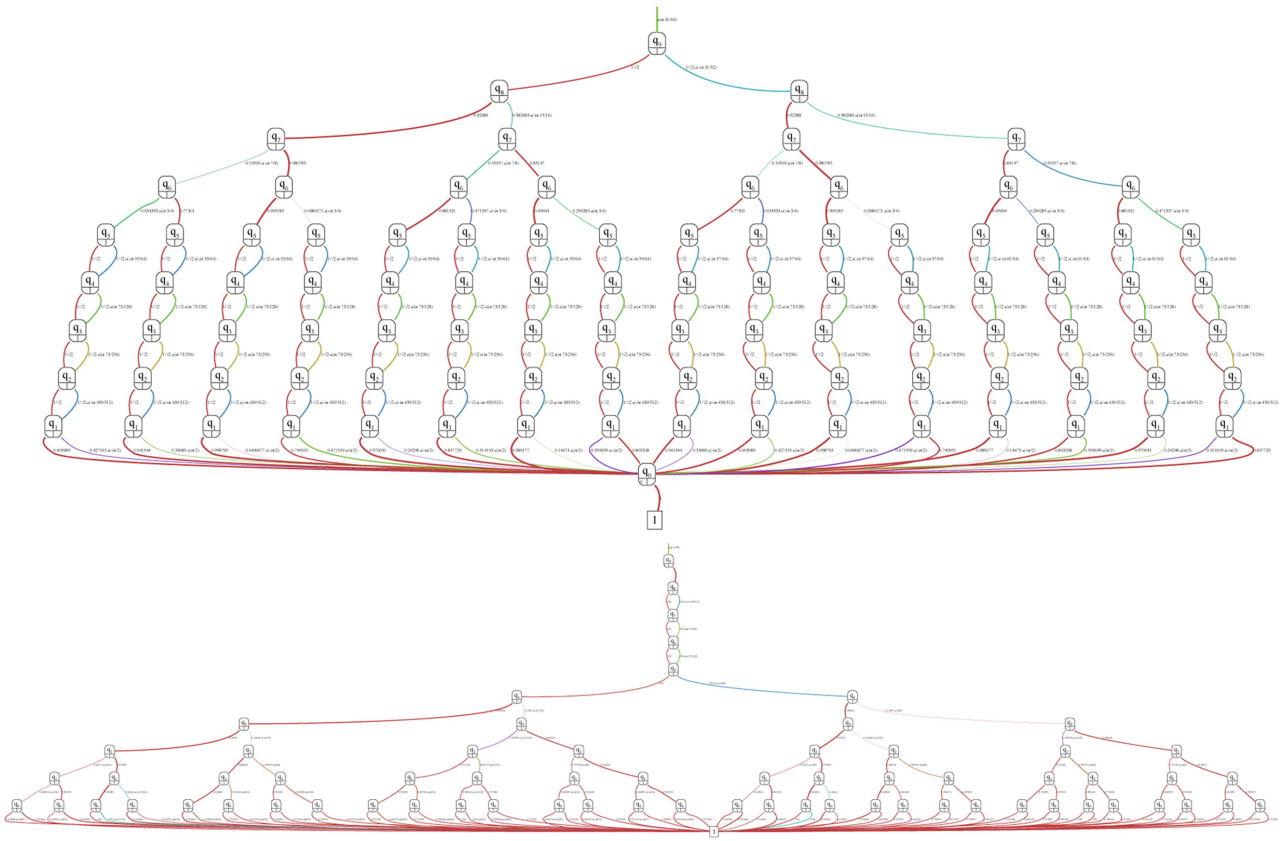

FIGURE 5. DDs representing the SV with different static variable orders (Intermediate result of 10-qubit QPE circuit).

researchers have not yet discovered rules applicable to all circuits in these experiments.

As described above, although we confirmed that the static variable order can change the size of the QDD, research in this area is lacking. Moreover, existing research has reported that dynamic reordering worsens the runtime of quantum circuit simulations. Therefore, this study proposes a static ordering method that can be applied to various quantum circuits to reduce the simulation time. The details of this method are presented in Section IV.

B. MULTINODE PARALLELIZATION

Parallelization can be divided into two categories: multithreading and multinode. Hillmich et al. [22] demonstrated that multithreaded DD simulation can be accelerated or slowed down depending on the input quantum circuit. DD-based simulators can simulate Grover and Shor circuits faster than other circuits, but these circuits become slower when multithreaded. Our previous studies [33], [34] achieved speedup through multithreading for various circuits; however, program optimization and refactoring after the publications eliminated the speedup effect, resulting in the same outcome as [22]. Therefore, this study focuses on multinode parallelization.

A multinode environment refers to a computing system, typically represented by a supercomputer high performance computing (HPC), in which multiple independent computing nodes are connected by a high-bandwidth, low-latency interconnect.

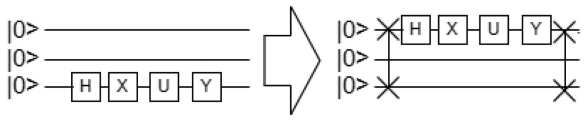

FIGURE 6. Auto SWAP insertion.

When executing an SV simulator in a multinode environment, the common method is to divide the SV evenly into each node [7]. For example, when simulating an N-qubit SV (length 2^N) with 2^M nodes, the SV is divided into 2^M chunks, and each node is responsible for 2^{N-M} elements. The first $N-M$ qubits are called local qubits, and the last M qubits are global. This is because gates acting on local qubits do not require communication between computing nodes, whereas gates acting on global qubits require internode communication.

Because internode communication is time-consuming, minimizing the number of communications as much as possible is important. Auto SWAP insertion [7] is a technique that automatically inserts SWAP gates before and after the calculations when many gates are applied to a specific global qubit. This temporarily swaps a frequently used global qubit with one of the local qubits (changing the qubit order), as shown in Fig. 6. Although the

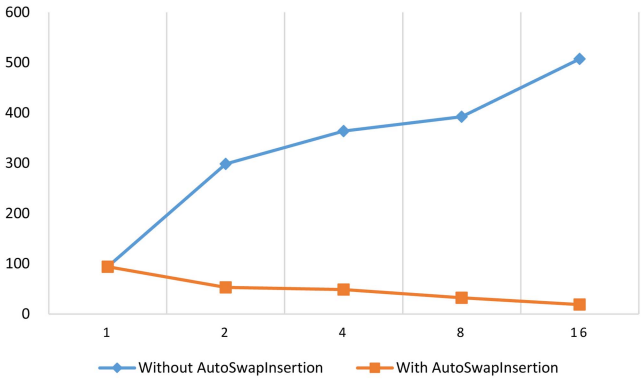

FIGURE 7. Simulation time for Shor57 circuits with or without Auto SWAP insertion (horizontal: number of computing nodes; vertical: simulation time).

SWAP operation requires internode communication, it can consolidate multiple global gate communications into two SWAP communications, thereby significantly reducing the overall simulation time. Fig. 7 shows a graph comparing the Shor circuit simulation time with and without auto-swap insertion. The simulation time is reduced with autoswap insertion as the number of computing nodes increases, whereas the time increases without autoswap insertion. Thus, it is an important technique for reducing runtime in multinode simulations.

The above technique can be regarded as dynamic reordering because inserting SWAP gates is equivalent to changing the variable (qubit) order in DD-based simulations. However, as mentioned in the previous section, dynamic reordering negatively affects DD simulators. Therefore, another acceleration method may be necessary to speedup multinode DD simulators without dynamic reordering. In this study, we propose ring communication, which reduces the waiting time in broadcast communication by devising a communication order. The approach of dynamic reordering and rind communication is different: dynamic reordering reduces the amount of communication, but ring communication does not. Ring communication changes the order of communication only. The details of this method are presented in Section V.

IV. SCORING-BASED HEURISTIC METHOD FOR STATIC VARIABLE ORDERING

The performance of a DD-based quantum circuit simulator depends heavily on variable ordering, that is, which qubit is assigned to which level of the DD. Therefore, this section proposes a heuristic method for determining the static variable order that shortens the execution time of a DD-based simulation. Static variable order determination is applied only once to the quantum circuit before the simulation.

This method focuses on multibit and parameterized rotation gates. Multibit gates generate entanglement between qubits, which can increase the size of a DD, as shown in Fig. 4. In addition, as shown in Fig. 3, the matrix values of the parameterized rotation gates (e.g., RX, RY, and RZ) tend to

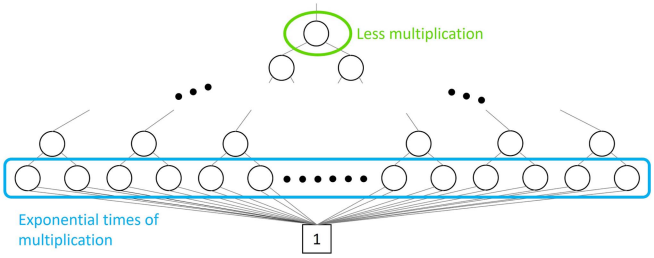

FIGURE 8. Relationship between the amount of computation and the qubit position.

be randomly distributed. When many rotation gates with different parameters are included in a circuit, the edge weights in the DD become diverse and the number of nodes increases. Our previous study [18] also confirmed that the performance of DD-based simulators degrades when the number of parameterized rotation gates is large. From the above discussion, special attention is required for qubits in which many multibit or parameterized rotation gates are applied.

In DD simulation, the computational complexity varies significantly depending on which qubit the quantum gate acts on. In Fig. 8, the qubit at the beginning of the variable order corresponds to the root node, and the calculations for the quantum gates acting on such a qubit only require changes around the top position. In contrast, the last qubit corresponds to a leaf node and may require updating up to $O(2^{N-1})$ graph nodes if DD is fully populated. This suggests that qubits with numerous parameterized rotations and multibit gates should be placed as close to the beginning as possible.

Based on the above observations, the proposed method determines the static variable order using a quantum circuit as the input. The number of multibit and parameterized rotation gates applied to each qubit was counted, and the variable order was determined in descending order. This process is shown in Code 2. Note that when the rotation parameter is $\frac{\pi}{2}$ or its multiple, the corresponding unitary matrix may contain many 0 or 1. In such cases, the size of the DD does not increase; therefore, such a gate is excluded from the count.

This method can calculate the scores in linear time relative to the number of gates in a quantum circuit, making it highly scalable. However, it should be noted that it cannot accurately predict the size of the DD and is a heuristic approach. In Section VI-A, we apply this method to various benchmark circuits and verify its usefulness.

V. RING COMMUNICATION FOR MULTINODE PARALLEL COMPUTATION

This section describes the "Ring Communication" method, which optimizes internode communication when performing DD-based quantum circuit simulations on multiple computing nodes.

Similar to the other multinode simulators described in Section III-B, the quantum state is assumed to be evenly divided and distributed among the computing nodes without overlapping. For implementation simplicity, the number of

Code 2: Proposed Static Variable Ordering.

```
Input:
  circuit: A quantum circuit, represented as
  a list of gates.
Output:
  sorted_qubits: A list of qubits, sorted in
  descending order of their scores.

Function SortVariables(circuit):
  scores = new Map()
  //Initialize all qubit scores to 0
  all_qubits = GetAllQubits(circuit)
  for each qubit in all_qubits do
    scores[qubit] = 0
  end for

  //Assign scores
  if IsMultiQubitGate(gate)
      or HasParams(gate) then
    for each qubit in gate.qubits do
      scores[qubit] = scores[qubit] + 1
    end for
  end if

  //Sort qubits by descending score
  sorted_qubits = SortKeysDescending(scores)
  return sorted_qubits
```

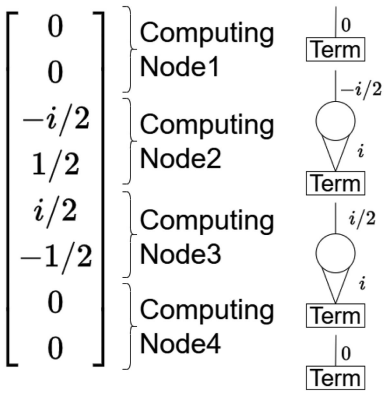

FIGURE 9. SV distribution in a multinode environment.

computing nodes was set to $2^M = 1, 2, 4, 8, \ldots$. As shown in Fig. 9, each computing node retains its responsible subvector in the DD.

Matrix–vector multiplication can be described by (3). Here, the SV is divided and stored across four computing nodes, represented by subvectors v_1, v_2, v_3, and v_4. A unitary matrix represents the quantum gate, and its submatrices are expressed as $w_{11}, \ldots, w_{44}$. The unitary matrix and its submatrices can be easily obtained from the quantum circuit description (e.g., OpenQASM)

$$\begin{pmatrix} v_1' \\ v_2' \\ v_3' \\ v_4' \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}$$

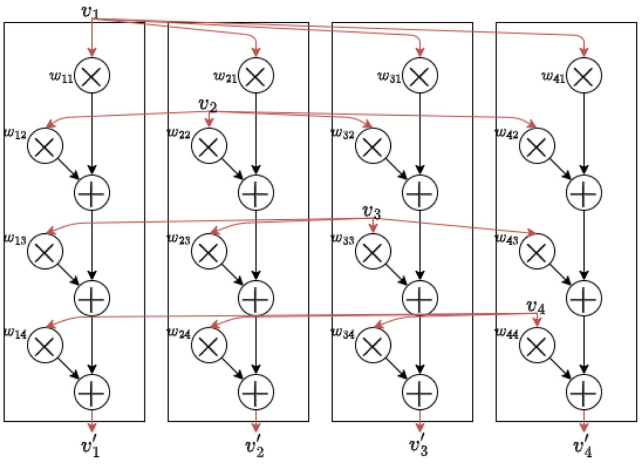

FIGURE 10. MPI communication with broadcast.

$$= \begin{pmatrix} w_{11} \times v_1 + w_{12} \times v_2 + w_{13} \times v_3 + w_{14} \times v_4 \\ w_{21} \times v_1 + w_{22} \times v_2 + w_{23} \times v_3 + w_{24} \times v_4 \\ w_{31} \times v_1 + w_{32} \times v_2 + w_{33} \times v_3 + w_{34} \times v_4 \\ w_{41} \times v_1 + w_{42} \times v_2 + w_{43} \times v_3 + w_{44} \times v_4 \end{pmatrix}. \quad (3)$$

As discussed in Section III-B, when a quantum gate acts on a global qubit, internode communication is required to access each subvector to compute the next SV. Fig. 10 shows a broadcast method in which v_1 from the first computing node is broadcast to all other nodes, and v_2, v_3, and v_4 are also broadcast. After each broadcast is completed, the matrix–vector product with the submatrix is calculated, and the sum of the intermediate results obtained thus far is determined. This method calculates the formula sequentially from left to right and is an intuitive implementation adopted by multinode SV simulators. However, nodes experience a waiting time until all communications are completed, which poses the problem of a large communication overhead.

To reduce the communication overhead, this study proposes ring communication, as shown in Fig. 11. The pseudocode is given in Code 3. Here, communication occurs only in one direction in a ring shape, between adjacent computing nodes. For example, v_1 in the first computing node is gradually transferred to the neighboring nodes. Similarly, v_2, v_3, and v_4 are progressively transferred to the adjacent nodes. Each computing node only needs to wait to complete communication with its adjoining node, and calculations and communications can be performed independently of the nonadjacent nodes. Therefore, it is expected to reduce the waiting time (communication overhead) compared with that of broadcast communication.

It should be noted that both broadcast communication and ring communication are calculated using (3). The difference between the two is the order of the calculation; although the values may differ slightly owing to floating-point errors, the final calculation results should be approximately

Code 3: "Ring Communication".

```
Input:
  M: DD representing the matrix
  v: DD representing the {vector}
  total_qubits: Total number of qubits
  target_qubits: Qubits that M applied to
Output:
  v_result: The resulting {vector} DD

Function ParallelMatrixVectorMultiply(M, v,
 total_qubits, target_qubits):
  rank = Get the rank of the process
  num_procs = Get total number of processes

  //Step 1: Perform local computation
  M_local = GetMatrixBlock(M, rank, rank)
  v_result = MatrixVectorMultiply(M_local, v)

  //Step 2: Is communication necessary?
  comm_threshold = total_qubits - log2
  (num_procs)
  if max(target_qubits) < comm_threshold then
    return v_result // No comm. needed
  end if

  //Step 3: Initialization for comm.
  send_buffer = Serialize(v)

  //Step 4: Ring communication loop
  for i from 1 to comm_size - 1 do
    //Send buffer to right node
    AsyncSend(send_buffer to right_node)
    //Receive buffer from left node
    recv_buffer = Receive from left_node

    //Get corresponding sub-matrix
    M_sub = GetSubMat(M, num_procs, rank,i)

    //Computation
    v_received = Deserialize(recv_buffer)
    v_partial = Multiply(M_sub, v_received)
    v_result = Add(v_result, v_partial)
    //Reuse buffer for the next loop
    MPI_Wait(recv_buffer)
    send_buffer = recv_buffer
  end for

  return v_result
```

equal. Fig. 12 schematically illustrates the communication between nodes. There were 12 arrows in both cases; therefore, the communication volume did not change. However, the presence or absence of broadcast communication causes differences in waiting time (overhead), which may also affect the overall simulation time. In ring communication, all four communication can be performed independently and in parallel.

Internode communication is unnecessary if a quantum gate acts only on local qubits. This is because all the submatrices, except w_{11}, w_{22}, w_{33}, and w_{44} are 0. To shorten the simulation time, it is important to avoid communication between nodes. This optimization is

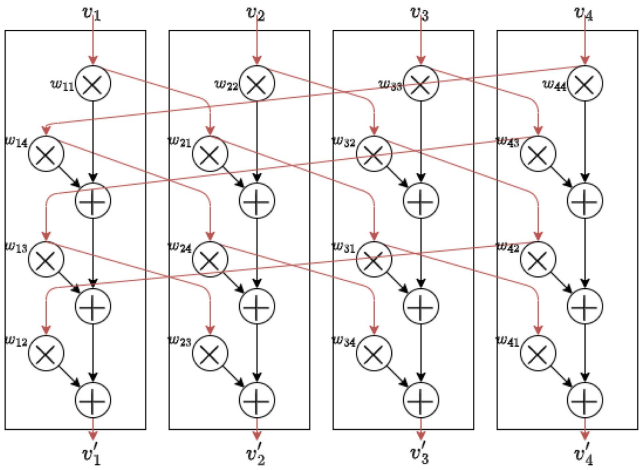

FIGURE 11. MPI communication with only neighbors (ring).

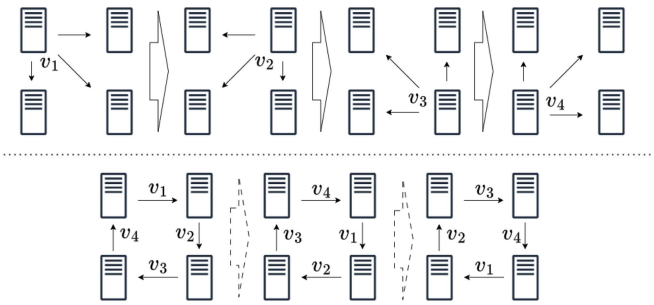

FIGURE 12. Difference between (upper) broadcast and (lower) ring communication.

implemented in our implementation, as shown in Step 2 of Code 3.

In Section VI-B, we measured and evaluated the difference in calculation time between the two communication methods in a multinode HPC environment. We also investigated the difference in the execution time when combined with the static variable order proposed in Section IV.

VI. EXPERIMENTAL RESULTS AND DISCUSSION

A. EX. 1: SINGLE-NODE EXPERIMENTS WITH THE PROPOSED STATIC VARIABLE ORDERING

1) ENVIRONMENT

The experiments were conducted in a Linux environment (Ubuntu 24.04, Kernel: 6.6.87.2-microsoft-standard-WSL2, GCC 13.3) built on a Windows laptop (CPU: Core i7 1370P, memory: 32 GB). The memory allocated to the Linux environment was 16 GB. The simulator used in this experiment is our DD-based simulator: QDD.[1] It is single-thread software written in C++ and Python. The implementation was disclosed as an open-source software. All experiments were conducted three times, and the shortest runtime was adopted, although the difference was insignificant (less than 3% or the

[1] https://github.com/Fujitsu-UTokyo-QDD/QDD/tree/ordering

TABLE 1. Ex. 1 Experimental Results: Runtime Difference Between the Original and Proposed Orders

Name	nQubits	nGates	Original order			Proposed order		
			#Node	max #Node	Time (s)	#Node	max #Node	Time (s)
GHZ	50	101	99	99	0.001	99	99	0.001
Deutsch Jozsa	50	197	50	99	0.004	50	52	0.004
WState	50	248	99	99	0.008	99	99	0.006
QFT	50	1351	50	50	0.029	50	50	0.012
Grover vchain	17	54 829	62	1000	7.5	26	345	1.6
QPE exact	28	474	28	9 608 851	139	28	9 516 247	122
VQE	16	95	62 828	65 535	0.31	55 882	61 995	0.43
Portofolio QAOA	17	494	131 071	131 071	13	131 071	131 071	13
Portofolio VQE	18	550	262 143	262 143	15	262 143	262 143	15
Two Local	18	550	262 143	262 143	48	262 143	262 143	50
Real Amplitude	18	550	262 143	262 143	50	262 143	262 143	50
SU2	18	550	262 143	262 143	51	262 143	262 143	54
QNN	18	1026	262 143	262 143	54	262 143	262 143	54
Graph State	40	121	121 357	121 357	62	121 357	121 357	61
Random	18	820	262 143	262 143	85	262 143	262 143	80
Amplitude Estimation	20	307	1 048 575	1 048 575	185	1 048 575	1 048 575	186
QFT entangled	21	284	2 097 151	2 097 151	51	2 097 151	2 097 151	22
QPE inexact	21	283	1 048 576	1 048 575	186	1 048 576	1 048 575	99

runtime was shorter than 1 s). This is similar to the setting of the previous simulation study [5]. The detailed experimental results can be found in the public repository [35]. It also includes cases where the allocated memory amount was 8 and 24 GB, although there were no significant difference.

The quantum circuits used in this experiment were obtained from MQT Bench [36]. This benchmark suite provides circuits with 2–130 qubits for each quantum algorithm. In this experiment, we used target-independent level circuits for Qiskit. The circuit size was chosen such that the simulation time was approximately 5 min. Algorithms that provided circuits with 15 or fewer qubits, in which the experiment time would be extremely short, were excluded; therefore, 18 types of circuits were used in this experiment. In this experiment, the measurement gates were ignored, and only the unitary quantum gates were simulated.

2) RESULTS AND EVALUATION

Table 1 presents the experimental results. The results of the original and proposed scoring-based static variable orders for each circuit are presented. #Node is the number of graph nodes used to represent the final quantum state of the simulation. "max #Node" is the maximum number of nodes of the SV during the simulation.

Figs. 13–16 show the number of graph nodes and elapsed time during the simulation. The horizontal axis represents the number of simulated gates, the left vertical axis represents the elapsed time, and the right vertical axis represents the number of graph nodes. Owing to space constraints, only the important results for the four quantum circuits [Grover vchain, Quantum Phase Estimation (QPE) exact, Random, and QPE inexact] are shown. The graphs of the VQE to amplitude estimation circuits (7–16th rows in Table 1) are similar to those of the random circuit in Fig. 15. The simulation times shown in these graphs are longer than those

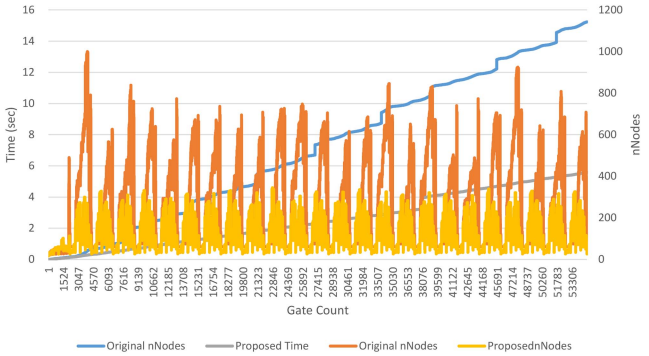

FIGURE 13. Ex. 1: Runtime and #Nodes during simulation: Grover vchain (horizontal: number of gates simulated).

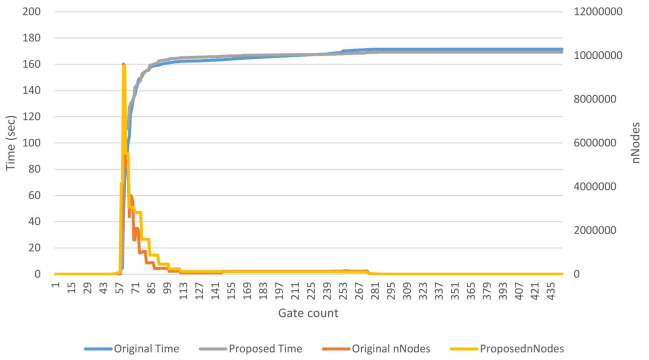

FIGURE 14. Ex. 1: Runtime and #Nodes during simulation: QPE exact (horizontal: number of gates simulated).

in Table 1 because additional calculations were performed to count the number of graph nodes at every gate.

For the first six quantum circuits in the table, the number of DD nodes is very small compared to the vector length 2^N of the N-bit quantum state. Therefore, these are good examples of DD-based compression of quantum states. For most

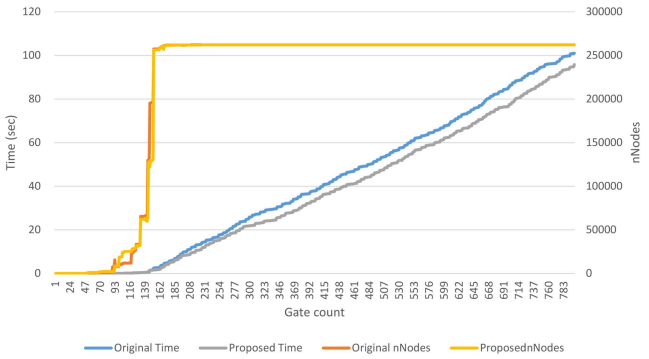

FIGURE 15. Ex. 1: Runtime and #Nodes during simulation: Random (horizontal: number of gates simulated).

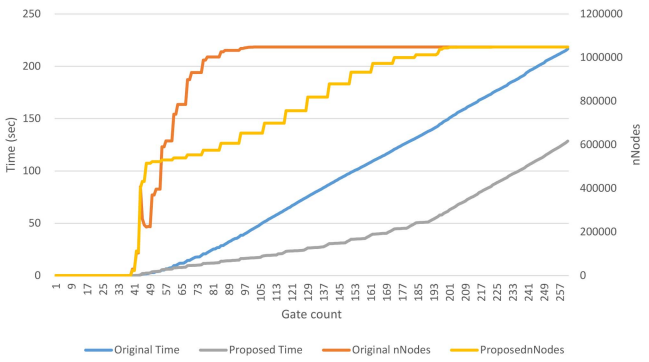

FIGURE 16. Ex. 1: Runtime and #Nodes during simulation: QPE inexact (horizontal: number of gates simulated).

circuits, the simulation was completed in a very short time, regardless of the variable order. However, in the Grover-vchain circuit, the proposed method reduced the number of nodes and accelerated the simulation by a factor of approximately 100. Furthermore, Fig. 13 shows that the number of graph nodes frequently increased and decreased during the simulation. The final number of nodes differed by more than ten times from that of the maximum number of nodes during the simulation. Therefore, both the final number of nodes and the number of nodes used during the simulation are important. The QPE exact circuit is another example in which the number of nodes during the simulation is important. In this case, there was no significant difference in the simulation time regardless of the variable order; however, there was a difference of approximately 300 000 times between the final number of nodes and maximum number of nodes.

The number of nodes was close to 2^N for the remaining circuits. The values of the SV in these quantum circuits are widely distributed, and the final DD is almost fully populated. In such cases, even if the variable order was changed, the graph structure itself did not change, and the experimental results showed no difference in the number of nodes or the simulation time. The "QPE inexact" and "QFT entangled" circuits were exceptions, showing a difference in the simulation time. Fig. 16 shows the statistics of the simulation of QPE inexact circuits. The number of nodes was suppressed during the simulation using the proposed variable order. As described above, the simulation time can be shortened by adopting the proposed method, even when the DD finally becomes fully populated.

While DD-based simulators can operate rapidly for algorithms, such as Shor and Grover, they are known to be slower than explicit SV-based simulators for circuits like random where the DD becomes fully populated [17], [18]. The experimental results confirm speedups for QPE, Grover, and QFT (used in Shor), indicating the proposed method is effective for DD-based simulators. On the other hand, the circuits not accelerated by the proposed method were not well suited for DD-based simulators and should have been simulated using other types of simulators, or took less than 1 s for simulation.

Summary of Example 1: The proposed static variable ordering method accelerated the simulation of several circuits. The number of nodes in these circuits differed. Differences in node counts may only occur during the simulation.

The proposed method was effective mainly for the QPE, QFT, and Grover circuits. In other algorithms where the number of graph nodes is extremely small (e.g., wstate) or approximately 2^N (e.g., random), the simulation time is the same, regardless of the variable order.

B. EX. 2: MULTINODE EXPERIMENTS WITH THE RING COMMUNICATION AND THE PROPOSED ORDERING

1) ENVIRONMENT

This experiment used Miyabi-C system,[2] which is an HPC provided by the University of Tokyo. This system has 190 computing nodes equipped with Xeon MAX and 128 GB of memory, and the nodes are connected using an InfiniBand NDR (200 Gbps). A maximum of 16 computing nodes were used in the experiment. Each node was a Linux environment (RHEL9.4, Kernel:5.14.0) with GCC11.4 and OpenMPI4.1.6. The DD simulator is the same one (QDD) as that used in Ex. 1, but with multinode functionality. Experiments were conducted under four conditions to evaluate the proposed method by combining the two communication methods and two variable orders.

1) Ring communication with original variable order.
2) Broadcast communication with original variable order.
3) Ring communication with proposed variable order.
4) Broadcast communication with proposed variable order.

In addition, a multinode SV-based quantum circuit simulator [7] was prepared for comparison. This simulator does not support ring communication and uses broadcasts. It also has an automatic swap-insertion function, as described in Section III-B. This is equivalent to dynamically changing the variable order. Experiments were conducted with this function enabled and disabled.

[2] https://www.cc.u-tokyo.ac.jp/en/supercomputer/miyabi/system.php

TABLE 2. Ex. 2: Experimental Results: Multinode Simulation: Random

Simulator	nQubits	nGates	Communication	Qubit Order	1	2	4	8	16
DD	22	1875	Ring	Original	2658	1288	726	343	235
			Bcast	Original	2658	1345	889	646	662
			Ring	Proposed	2622	1284	771	428	326
			Bcast	Proposed	2622	1411	1018	713	815
SV	31	3702	Bcast	Original	246	472	368	227	154
			Bcast	AutoSwap (Dynamic)	246	114	133	71	51

TABLE 3. Ex. 2: Experimental Results: Multinode Simulation: QPE Inexact

Simulator	nQubits	nGates	Communication	Qubit Order	1	2	4	8	16
DD	25	386	Ring	Original	3290	3316	1690	1116	849
			Bcast	Original	3336	3216	2021	1710	1744
			Ring	Proposed	2179	1022	589	436	349
			Bcast	Proposed	2179	1070	791	712	776
SV	31	572	Bcast	Original	99	129	115	79	54
			Bcast	Dynamic	99	41	47	28	13

TABLE 4. Ex. 2: Experimental Results: Multinode Simulation: Shor

Simulator	Composite Number	nQubits	nGates	Communication	Qubit Order	1	2	4	8	16
DD	533	42	479 497	Ring	Original	2543	2094	1845	1472	1138
				Bcast	Original	2543	2100	1788	1465	1190
				Ring	Proposed	1178	1023	822	755	887
				Bcast	Proposed	1178	1080	906	853	897
SV	57	26	51 312	Bcast	Original	94	299	364	392	507
				Bcast	Dynamic	94	53	49	32	19

1) SV: Broadcast communication with fixed original order.
2) SV: Broadcast communication with Auto SWAP insertion (dynamic reordering).

Using all the benchmark circuits from the previous experiment required excessive computation time; therefore, we decided to use two circuits: random and QPE inexact circuits. The QPE circuit is an important circuit used in various FTQC algorithms and was adopted because its execution time varies depending on the order of the variables in the previous section. A random circuit was adopted because its execution time was similar regardless of the variable order, unlike that of the QPE circuit. Furthermore, experiments were conducted using Shor's algorithm circuit, for which DD-based quantum circuit simulators can perform better than the others [17], [18]. The circuit uses $4N + 2$ qubits as N-bit integers [1].

Ideally, the same quantum circuits should be used for experiments with both SV- and DD-based quantum circuit simulators. However, this would lead to problems in which the simulation time becomes extremely short or long. In Table 4, the SV simulator does not finish within a practical time for the number of qubits in the DD simulator, and vice versa in Tables 2 and 3. Therefore, we present the results with different numbers of qubits between the two quantum circuit simulators in this experiment. This makes it easier to understand the effects of the multinode parallelization and variable order.

2) RESULTS AND EVALUATION

The experimental results are listed in Tables 2–4. In addition, Fig. 17 shows the relative simulation times of the multinode settings, where the time for a single computing node is set to 1.

Ring communication achieved the same or faster simulation as broadcast communication in all the experiments. Although previous research, such as [22], failed to speedup Shor simulations with parallelization, our proposed multinode method makes this possible. The effect was greater with more computing nodes, achieving a simulation speedup of up to 2.8 times compared to broadcast communication with 16 nodes. In addition, while the difference was small for Shor circuits, the effect was significant for the random circuits. The 16-node QDD was approximately 11 times faster than the single node.

The proposed static variable ordering method reduces the simulation time for a single node. For Shor and QPE, it also accelerates simulations in a multinode environment. However, it should be noted that the difference between the two became smaller because the original variable order accelerated more.

When the random circuit was tested with one node, there was no significant difference in time between the original and

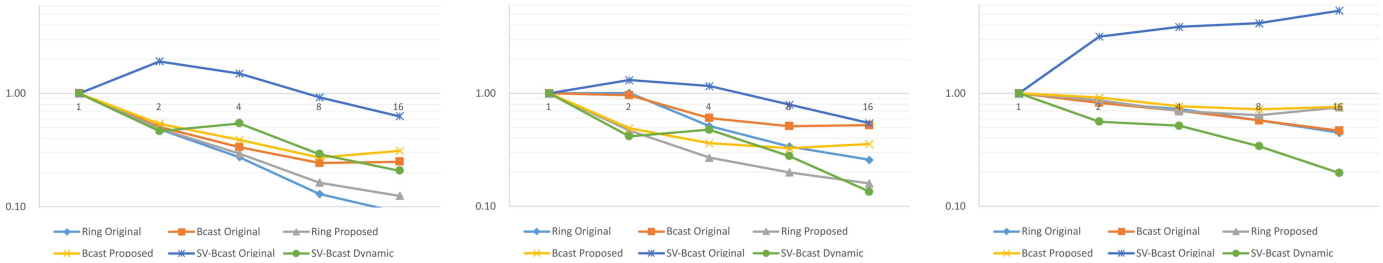

FIGURE 17. Ex. 2: Multinode runtime speedup (Left) Random, (middle) QPE inexact, and (right) Shor [horizontal: number of computing nodes; vertical. time (single-node runtime = 1)].

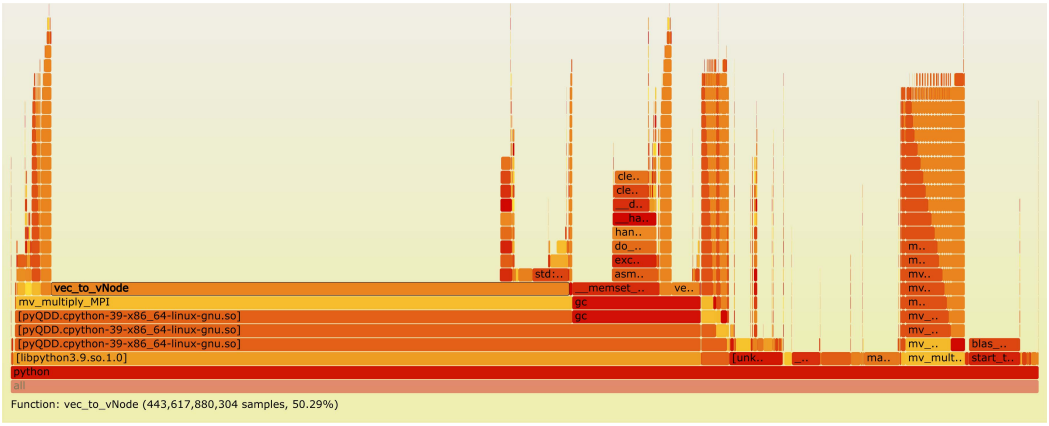

FIGURE 18. Ex. 2: Flame graph of the multinode computation [horizontal: length of the stack trace (CPU time); vertical: depth of the stack].

proposed orders. However, when the number of computing nodes increased, the original ordering had a greater acceleration effect, resulting in shorter simulation times. Upon investigation, it was found that the original order had fewer global gates than the proposed order, which resulted in less overhead owing to internode communication. This is because the proposed method does not consider the communication volume in a multinode environment. Fig. 18 shows a flame graph that summarizes the results of profiling a computing node while simulating a random circuit with 16 nodes. It displays where CPU time is consumed by stacking up function calls. The horizontal axis represents the length of the stack trace (CPU time), and the vertical axis represents the depth of the stack, with wider blocks indicating greater time consumption. This indicates that the stack (vec_to_vNode) corresponding to line 37 in Code 3 is the dominant one. Here, the DD data from adjacent computing nodes are searched and stored in a unique table, as described in Section II-C2d. When creating a new node, an operation is performed to determine whether the same node already exists in a unique table. However, cache misses always occur because the same node cannot appear in a random circuit, resulting in a long search duration. Thus, although QPE and Shor were not affected, the number of global gates significantly affected the overall simulation time of the random circuit. Such completely random circuits are rarely used in quantum algorithm research, and SV can simulate them faster; therefore, this problem is not considered serious.

Next, we compared it with SV-based simulators. When the SV simulator was parallelized with multiple nodes, a 5–7 times speedup was observed with 16 nodes in combination with dynamic reordering. In contrast, DD achieved up to 11 times speedup with 16 nodes, successfully utilizing the multinode environment more effectively. However, the SV simulator was faster for the random and QPE circuits because both contain many rotation gates, and 2^N graph nodes are required for N qubits. Although Shor's algorithm also includes QPE, the DD simulator operates faster than SV.

Summary of Example 2: Our multinode simulation achieved a speedup for various circuits, including Shor's algorithm. The proposed ring communication realizes faster simulations than the broadcast communication. The proposed static variable ordering contributed to acceleration even in a multinode environment, except in extreme cases, such as random circuits.

VII. CONCLUSION

In this study, we proposed a static variable-ordering method and multinode parallelization method using ring communication to improve the performance of DD-based quantum-circuit simulators.

For static variable ordering, we proposed a heuristic method that orders qubits based on a score derived from the number of parameter rotation gates and multibit gates in the quantum circuit. In a single-node environmental evaluation, we showed that the proposed method can achieve

up to a 4.5× speedup compared with the original variable orders.

For parallelization in a multinode environment, we proposed a ring communication method that reduces the communication overhead (waiting time) between computing nodes. Ring communication showed comparable or superior performance to existing broadcast communication, and its effect became more pronounced as the number of nodes increased, achieving up to 11 times speedup with 16 computing nodes. This indicates that ring communication reduced the waiting time for synchronous communication and improved communication efficiency. Our multinode simulator accelerated the Shor's circuit simulation, which was not achieved in previous multithreading studies [22].

We also evaluated the combination of the proposed static variable-ordering method and ring communication in a multinode environment. Consequently, it was confirmed that further acceleration could be achieved by combining the two methods. However, in specific extreme cases, such as random circuits, it was also revealed that the proposed variable order increased the number of global gates, leading to longer simulation times than other variable orders. However, such circuits are simulated faster using other types of simulators, and this problem is not considered serious.

Future work will include the proposal of a static variable order that considers multinode communication. Moreover, ring communication can be applied to other types of quantum simulators, e.g., explicit SV-based simulators in which the SV can be equally divided among nodes as in Section V. In this study, the data exchanged between the computing nodes were DDs; however, they can also assume different formats. Using a different representation changes the amount of communication required. Therefore, further investigation is required to confirm the usefulness of ring communication.

ACKNOWLEDGMENT

This research builds upon the foundational work initially presented at QSW2024 [1], where the concept of ring communication for multinode simulation was first introduced. This journal paper represents a substantial advancement of that work, integrating a newly developed variable ordering method and presenting more extensive experimental evaluation.

REFERENCES

[1] Y. Kimura, S. Li, H. Sato, and M. Fujita, "Accelerating decision diagram-based multi-node quantum simulation with ring communication and automatic swap insertion," in *Proc. 2024 IEEE Int. Conf. Quantum Softw. (QSW)*, 2024, pp. 107–115, doi: 10.1109/QSW62656.2024.00025.

[2] J. Gambetta, "IBM quantum system two: The era of quantum utility is here| IBM Quantum Computing Blog — ibm.com," 2023, Accessed: Feb. 14, 2025. [Online]. Available: https://www.ibm.com/quantum/blog/quantum-roadmap-2033

[3] S. Bravyi, D. Browne, P. Calpin, E. Campbell, D. Gosset, and M. Howard, "Simulation of quantum circuits by low-rank stabilizer decompositions," *Quantum*, vol. 3, Sep. 2019, Art. no. 181, doi: 10.22331/q-2019-09-02-181.

[4] Q. contributors, "Qiskit: An open-source framework for quantum computing," 2023. [Online]. Available: https://github.com/Qiskit/qiskit

[5] Y. Suzuki et al., "Qulacs: A fast and versatile quantum circuit simulator for research purpose," *Quantum*, vol. 5, Oct. 2021, Art. no. 559, doi: 10.22331/q-2021-10-06-559.

[6] H. De Raedt et al., "Massively parallel quantum computer simulator, eleven years later," *Comput. Phys. Commun.*, vol. 237, pp. 47–61, 2019, doi: 10.1016/j.cpc.2018.11.005.

[7] S. Imamura et al., "mpiqulacs: A distributed quantum computer simulator for A64FX-based cluster systems," 2022, *arXiv:2203.16044*, doi: 10.48550/arXiv.2203.16044.

[8] R. Orús, "A practical introduction to tensor networks: Matrix product states and projected entangled pair states," *Ann. Phys.*, vol. 349, pp. 117–158, 2014, doi: 10.1016/j.aop.2014.06.013.

[9] J. Gray and S. Kourtis, "Hyper-optimized tensor network contraction," *Quantum*, vol. 5, Mar. 2021, Art. no. 410, doi: 10.22331/q-2021-03-15-410.

[10] M. Fannes, B. Nachtergaele, and R. F. Werner, "Finitely correlated states on quantum spin chains," *Commun. Math. Phys.*, vol. 144, no. 3, pp. 443–490, Mar. 1992, doi: 10.1007/BF02099178.

[11] G. Vidal, "Efficient classical simulation of slightly entangled quantum computations," *Phys. Rev. Lett.*, vol. 91, no. 14, Oct. 2003, Art. no. 147902, doi: 10.1103% 2Fphysrevlett/91.147902.

[12] J. C. Bridgeman and C. T. Chubb, "Hand-waving and interpretive dance: An introductory course on tensor networks," *J. Phys. A: Math. Theor.*, vol. 50, no. 22, May 2017, Art. no. 223001, doi: 10.1088/1751-8121/aa6dc3.

[13] D. Miller and M. Thornton, "QMDD: A decision diagram structure for reversible and quantum circuits," in *Proc. 36th Int. Symp. Mult.-Valued Log.*, 2006, pp. 30–30, doi: 10.1109/ISMVL.2006.35.

[14] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler, "QMDDs: Efficient quantum function representation and manipulation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, no. 1, pp. 86–99, Jan. 2016, doi: 10.1109/TCAD.2015.2459034.

[15] A. Zulehner and R. Wille, "Advanced simulation of quantum computations," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 5, pp. 848–859, 2019, doi: 10.1109/TCAD.2018.2834427.

[16] A. Zulehner, S. Hillmich, and R. Wille, "How to efficiently handle complex values? implementing decision diagrams for quantum computing," in *Proc. 2019 IEEE/ACM Int. Conf. Computer-Aided Des.*, 2019, pp. 1–7, doi: 10.1109/ICCAD45719.2019.8942057.

[17] T. Grurl, J. Fuß, S. Hillmich, L. Burgholzer, and R. Wille, "Arrays vs. decision diagrams: A case study on quantum circuit simulators," in *Proc. IEEE 50th Int. Symp. Mult.-Valued Log.*, 2020, pp. 176–181, doi: 10.1109/ISMVL49045.2020.000-9.

[18] Y. Kimura, S. Li, H. Sato, and M. Fujita, "Decision diagram vs. state vector: A comparative study on quantum computing simulation efficiency," in *Proc. 2024 IEEE Int. Conf. Quantum Comput. Eng.*, vol. 01, 2024, pp. 757–763, doi: 10.1109/QCE60285.2024.00095.

[19] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. 35, no. 8, pp. 677–691, Aug. 1986, doi: 10.1109/TC.1986.1676819.

[20] S. Hillmich, L. Burgholzer, F. Stögmüller, and R. Wille, "Reordering decision diagrams for quantum computing is harder than you might think," in *Reversible Computation*, C. A. Mezzina and K. Podlaski, Eds., Cham, Switzerland: Springer, 2022, pp. 93–107, doi: 10.1007/978-3-031-09005-9_7.

[21] S. Van Schaick and K. B. Kent, "Analysis of variable reordering on the QMDD representation of quantum circuits," in *Proc. 10th Euromicro Conf. Digit. System Des. Architectures, Methods Tools (DSD 2007)*, 2007, pp. 347–352, doi: 10.1109/DSD.2007.4341491.

[22] S. Hillmich, A. Zulehner, and R. Wille, "Concurrency in DD-based quantum circuit simulation," in *Proc. 25th Asia South Pacific Des. Automat. Conf.*, 2020, pp. 115–120, doi: 10.1109/ASP-DAC47756.2020.9045711.

[23] A. Zulehner and R. Wille, "Matrix-vector vs. matrix-matrix multiplication: Potential in DD-based simulation of quantum computations," in *Proc. 2019 Des., Autom. Test Eur. Conf. Exhibit.*, 2019, pp. 90–95, doi: 10.23919/DATE.2019.8714836.

[24] S. Jiang, R. Fu, L. Burgholzer, R. Wille, T.-Y. Ho, and T.-W. Huang, "FlatDD: A high-performance quantum circuit simulator using decision diagram and flat array," in *Proc. 53rd Int. Conf. Parallel Process.*, 2024, pp. 388–399, doi: 10.1145/3673038.3673073.

[25] S. Jiang, Y.-H. Chung, C.-C. Chang, T.-Y. Ho, and T.-W. Huang, "BQSim: GPU-accelerated batch quantum circuit simulation using decision diagram," in *Proc. 30th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, vol. 2, 2025, pp. 79–94, doi: 10.1145/3676641.3715984.

[26] S. Jaques and T. Häner, "Leveraging state sparsity for more efficient quantum simulations," *ACM Trans. Quantum Comput.*, vol. 3, no. 3, pp. 1–17, Jun. 2022, doi: 10.1145/3491248.

[27] M. Fujita, P. C. McGeer, and J. C.-Y. Yang, "Multi-terminal binary decision diagrams: An efficient data structure for matrix representation," *Formal Methods System Des.*, vol. 10, no. 2, pp. 149–169, Apr. 1997, doi: 10.1023/A:1008647823331.

[28] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Proc. 1993 Int. Conf. Comput. Aided Des.*, 1993, pp. 42–47, doi: 10.1109/ICCAD.1993.580029.

[29] M. Fujita, H. Fujisawa, and N. Kawato, "Evaluation and improvement of Boolean comparison method based on binary decision diagrams," in *Proc. IEEE Int. Conf. Comput.-Aided Des. ICCAD-89 Dig. Tech. Papers*, 1988, pp. 2–5, doi: 10.1109/ICCAD.1988.122450.

[30] K. Qayyum, A. Mahzoon, and R. Drechsler, *Start Small But Dream Big: On Choosing a Static Variable Order for Multiplier BDDs*. Cham, Switzerland: Springer, 2023, pp. 155–169, doi: 10.1007/978-3-031-28916-3_11.

[31] K. Qayyum et al., "Monitoring the effects of static variable orders on the construction of BDDs," in *Proc. Int. Interdiscipl. Conf. Math., Eng. Sci.*, 2022, pp. 1–6, doi: 10.1109/MESIICON55227.2022.10093493.

[32] S. Shirinzadeh, M. Soeken, and R. Drechsler, "Multi-objective BDD optimization with evolutionary algorithms," in *Proc. 2015 Annu. Conf. Genet. Evol. Computation*, 2015, pp. 751–758, doi: 10.1145/2739480.2754718.

[33] S. Li, Y. Kimura, H. Sato, J. Yu, and M. Fujita, "Parallelizing quantum simulation with decision diagrams," *IEEE Trans. Quantum Eng.*, vol. 5, 2024, Art. no. 2500212, doi: 10.1109/TQE.2024.3364546.

[34] S. Li, Y. Kimura, H. Sato, and M. Fujita, "Parallelizing quantum simulation with decision diagrams," *IEEE Trans. Quantum Eng.*, vol. 5, 2024, Art. no. 2500212, doi: 10.1109/TQE.2024.3364546.

[35] "Improving decision diagram-based quantum circuit simulation using static variable ordering and multi-node ring communication," 2025. [Online]. Available: https://doi.org/10.5281/zenodo.17972524

[36] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT bench: Benchmarking software and design automation tools for quantum computing," *Quantum*, vol. 7, 2023, Art. no. 1062, doi: 10.22331/q-2023-07-20-1062.

Yusuke Kimura received the B.S., M.S., and Ph.D. degrees in electronic engineering from the University of Tokyo, Tokyo, Japan, in 2015, 2017, and 2020, respectively.

He is currently a Researcher with Quantum Laboratory, Fujitsu Research, and a Visiting Researcher with the Technical University of Munich, Munich, Germany. His research interests include the verification of digital systems, program synthesis and quantum computing simulations.

Shaowen Li received the B.S. degree in computer engineering from the University of British Columbia, Vancouver, BC, Canada, in 2019, and the M.S. degree in electrical engineering and information systems in 2021 from the University of Tokyo, Tokyo, Japan, where he is currently working toward the Ph.D. degree in computer science.

His research interests include operating system, blockchain, and quantum computing.

Hiroyuki Sato (Member, IEEE) received the B.Sc., M.Sc., and D.Sc. degrees in computer science from the Department of Information Science, University of Tokyo, Tokyo, Japan, in 1985, 1987, and 1990, respectively.

In 1990, he was an Assistant Professor with Kyushu University, Fukuoka, Japan. He is currently a Project Professor with the Information Technology Center, University of Tokyo. His research interests include security and program semantics.

Masahiro Fujita (Life Member, IEEE) received the Ph.D. degree in information engineering from the University of Tokyo, Tokyo, Japan, in 1985.

In 1985, he joined Fujitsu, Tokyo. From 1993 to 2000, he was assigned to Fujitsu Laboratories of America, Santa Clara, CA, USA, where he was the VLSI CAD research group Director. Since 2000, he has been a Professor with the VLSI Design and Education Center, University of Tokyo, until he retired in 2022. He is currently a Researcher with the University of Tokyo and National Institute of Advanced Industrial Science and Technology. His research interests include hardware/software codesigns targeting embedded systems and formal analysis, verification, and synthesis of cyber-physical systems and quantum computing systems. Recently, he has been also working on hardware implementation of neural network-based information processing systems, such as learned image compression.

Robert Wille (Senior Member, IEEE) is a Full and Distinguished Professor with the Technical University of Munich, CEO of the Munich Quantum Software Company, and Scientific Director with the Software Competence Center Hagenberg, Hagenberg, Germany. He has authored or coauthored more than 400 papers and serves on editorial and advisory boards of major journals and conferences. His research focuses on the design of circuits and systems for both conventional and emerging technologies, with more than 15 years of contributions to quantum computing—particularly in foundational software and design automation .

Mr. Wille was the recipient of numerous accolades, including Best Paper Awards, the DAC Under-40 Innovator Award, a Google Research Award, and an ERC Consolidator Grant. He collaborates with leading academic and industrial partners, plays a key role in initiatives, such as the Munich Quantum Valley, and actively supports technology transfer through his roles in industry.