2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO) | 978-1-6654-6272-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/MICRO56248.2022.00074

AutoComm: A Framework for Enabling Efficient Communication in Distributed Quantum Programs

Anbang Wu*, Hezi Zhang*, Gushu Li†, Alireza Shabani‡, Yuan Xie† and Yufei Ding*
*CS Department, †ECE Department
Uuniversity of California, Santa Barbara
Santa Barbara, USA
{anbang, hezi, yufeiding}*@ucsb.edu, {gushuli, yuanxie}†@ece.ucsb.edu
‡Cisco Research
Los Angeles, USA
ashabani@cisco.com

***Keywords*-quantum computing, quantum compiler**

***Abstract*—Distributed quantum computing (DQC) is a promising approach to extending the computational power of near-term quantum hardware. However, the non-local quantum communication between quantum nodes is much more expensive and error-prone than the local quantum operation within each quantum device. Previous DQC compilers focus on optimizing the implementation of each non-local gate and adopt similar compilation designs to single-node quantum compilers. The communication patterns in distributed quantum programs remain unexplored, leading to a far-from-optimal communication cost. In this paper, we identify *burst communication*, a specific qubit-node communication pattern that widely exists in various distributed quantum programs and can be leveraged to guide communication overhead optimization. We then propose AutoComm, an automatic compiler framework to extract burst communication patterns from input programs and then optimize the communication steps of burst communication discovered. Compared to state-of-the-art DQC compilers, experimental results show that our proposed AutoComm can reduce the communication resource consumption and the program latency by 72.9% and 69.2% on average, respectively.**

I. Introduction

Quantum computing is promising with its great potential of providing significant speedup to many problems, such as large-number factorization with an exponential speedup [1] and unordered database search with a quadratic speedup [2]. A large number of qubits is required in order to solve practical problems with quantum advantage and the qubit count requirement is even higher after taking quantum error correction [3] into consideration. However, it has turned out that extending the number of qubits on a single quantum processor is exceedingly difficult due to various hardware-level challenges such as crosstalk errors [4], [5], qubit addressability [6], fabrication difficulty [7], etc. Those challenges usually increase with the size of quantum hardware and may limit the number of qubits in a single quantum processor.

Rather than relying on the advancement of a single quantum processor, an alternative way to scale up quantum computing is to explore distributed quantum computing (DQC) [8], [9], which integrates the computing resources of multiple quantum processors. In DQC, remote quantum communication involving qubits in different compute nodes is essential yet far more expensive than the local communication between same-node qubits (e.g., 5-100x time consumption and up to 40x fidelity degradation [10], [11]). There are two major schemes for inter-node communication: one built upon cat-entangler and -disentangler protocols [12], and the other based on quantum teleportation [3]. In this paper, we refer to the former scheme as *Cat-Comm* and the latter one as *TP-Comm*. Both schemes consume remote EPR pairs [13], which are pre-distributed entangled qubit pairs, as a resource to establish quantum communication. Cat-Comm can implement a remote CX gate [3] with only one EPR pair, but for general two-qubit gates such as SWAP gate [3], Cat-Comm requires up to three EPR pairs [14]. In contrast, TP-Comm can execute any remote two-qubit gate with two EPR pairs [13] and is thus more efficient for the SWAP gate. For a distributed quantum program, more complex remote operations or more quantum information transferred per EPR pair would lead to less communication cost.

The overall compiling flow for DQC is similar to that of single-node quantum programs, except with more emphasis on remote communication overhead. One compiler design proposed by Ferrari et al. [14] adopts a similar compilation strategy to single-node compilers [15]–[19], using Cat-Comm for implementing the remote CX. This strategy has a low communication throughput due to the low information of the remote CX gate. The compiler by Baker et al. [10] and another design by Ferrari et al. [14] instead eliminate all remote CX gates by using remote SWAP, which only requires two EPR pairs for implementation but contains the information of three CX gates. Unfortunately, bounded by the information of a single two-qubit gate, these compilers cannot achieve higher throughput of information per EPR pair. Eisert et al. [13] suggest that a higher throughput could be achieved by considering multi-qubit gates. Diadamo et al. [20]

978-1-6654-6272-3/22/$31.00 ©2022 IEEE
DOI 10.1109/MICRO56248.2022.00074

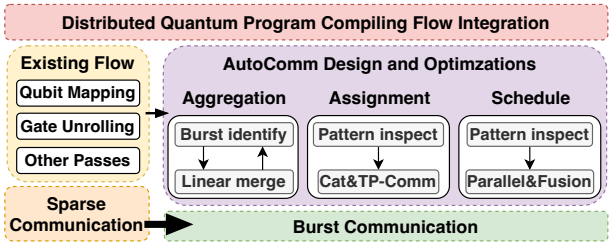


Figure 1. AutoComm Overview.

compiles distributed VQE by using Cat-Comm to implement controlled-unitary-unitary and controlled-controlled-unitary gates. However, their work can only optimize gates written in the controlled-unitary form, not applicable to the decomposed circuit that consists of only quantum basis gates (e.g., CX+U3 [16]). Besides, their work cannot optimize programs lacking controlled-unitary blocks.

Besides increasing the 'height' (number of qubits) of remote operations, we observe that the throughput of information per EPR pair can also be significantly boosted up by expanding the 'width' (number of gates) of each remote communication. Specifically, we discover that it is possible to implement a group of remote two-qubit gates collectively through one or two EPR pairs. On top of the observation, we propose optimizing the communication overhead of distributed quantum programs based on the *burst communication*, which denotes a group of continuous remote two-qubit gates between one qubit and one node. Burst communication is powerful as it is more information-intensive than a single two-qubit gate and contains but is not limited to controlled-unitary blocks. Burst communication is also flexible for optimization as it does not require specialized circuit representation and is available in decomposed circuits.

To this end, we develop the first burst-communication-centric optimization framework, *AutoComm*. Our framework focuses on the optimization of remote communication and leverages existing compiling flows [10], [14]–[19] on other program optimization aspects (e.g., qubit mapping to assign qubits over different DQC nodes), as shown in Figure 1. This two-step design not only makes our work extensible but also outstands our significant novel contributions on communication optimization. In existing compiling flows, each CX is implemented independently (i.e., sparse communication); while with *AutoComm*, sparse communication is converted into burst communication and specifically optimized for higher communication throughput.

Overall, our framework consists of three key stages. Firstly, we perform a communication aggregation pass to group remote gates and extract burst communication blocks. Due to the wide existence of burst communication in distributed quantum programs, this pass could generate a large amount of burst communication blocks for the following optimizations. Secondly, we propose a hybrid communication scheme that examines the patterns of each burst communication block and assigns the optimal communication scheme for each block. The insight is that TP-Comm and Cat-Comm are more resource-efficient for different types of burst communication, thus considering only one scheme would incur extra resource consumption. Finally, we adopt an adaptive schedule for burst communication blocks of different patterns to squeeze out the parallelism between them and thus reduce the overall program latency. We observe that it is possible to execute burst communication with shared qubits or nodes in parallel, and we can fuse some burst communication blocks to cut down the communication footprint.

Our contributions are summarized as follows:

- We identify the burst communication feature in DQC and promote its importance in optimizing distributed quantum programs. we further propose the first communication optimization framework based on it.
- We propose a communication aggregation pass to expose burst communications and design a hybrid communication scheme, using both Cat- and TP-Comm to accommodate different communication patterns.
- We propose an efficient communication scheduling method to optimize the program latency adaptively, squeezing out the parallelism of burst communication.
- Compared to state-of-the-art baselines, AutoComm reduces the EPR pair consumption and the program latency by 72.9% and 69.2% on average, respectively.

II. Background

In this section, we would introduce the necessary background to understand distributed quantum computing and inter-node quantum communication. We do not cover the basic quantum computing concepts (e.g., qubit, gate, measurement) and recommend [3] for more details.

(a) Quantum communication

Like in classical distributed computing, remote/inter-node quantum communication is the bedrock of distributed quantum computing (DQC) but is also the bottleneck of DQC. Different from classical distributed computing, quantum data cannot be easily shared across quantum nodes due to the quantum no-cloning theorem [3]. The workaround is to exploit inter-node quantum entanglement. If two qubits are in the entangled quantum state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, we say they form an *EPR pair* [3]. The two qubits of an EPR pair can be distributed to different quantum nodes, formulating a remote EPR pair [13]. The remote EPR pair is the most widely-used quantum communication resource, providing the necessary quantum entanglement for transferring quantum data between nodes. Actually, based on EPR pairs, two communication protocols emerge, named *Cat-Comm* and *TP-Comm* respectively. Figure 2 illustrates how to use these two schemes to implement one *remote CX gate*, with the control qubit q_0 residing in quantum node A and the target qubit q_0' in node B. Qubits in Figure 2 fall into two categories. The first

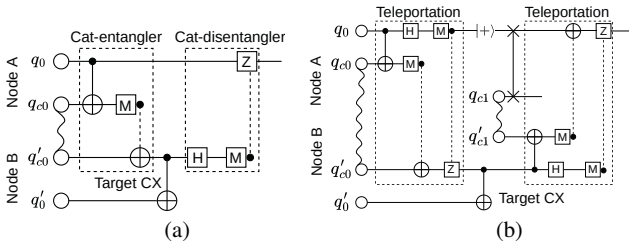


Figure 2. The implementation of one remote CX. (a) The Cat-Comm version. (b) The TP-Comm version. Wavy lines and dashed lines denote EPR pairs and classical communication bits respectively. M denotes measurement.

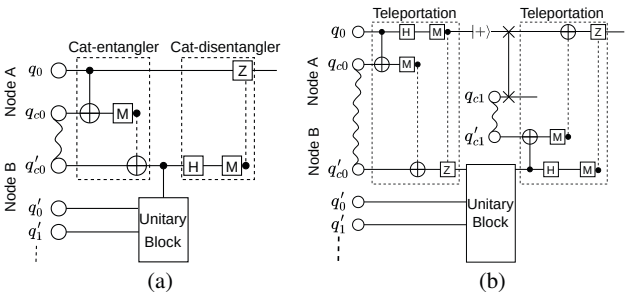


Figure 3. The optimized implementation of complex remote interactions. (a) *Controlled-unitary* block by one call of Cat-Comm. (b) *Unitary* block by two calls of TP-Comm.

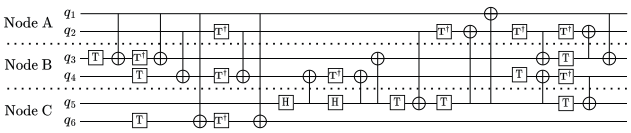


Figure 4. Program snippet extracted from quantum arithmetic circuits [22].

category of qubits is used to store program information and is called *data qubits*, e.g., q_0 and q'_0. The second category of qubits, called *communication qubits*, is used to hold remote EPR pairs, e.g. q_{c0} and q'_{c0}.

(b) Cat-Comm and TP-Comm

As shown in Figure 2(a), Cat-Comm utilizes cat-entangler to transfer the state of the control qubit q_0 to node B, execute the target CX gate, and then use cat-disentangler to transfer the state back to node A. TP-Comm in Figure 2(b) employs quantum teleportation [3] to transfer the state of q_0 to node B, and then execute the target CX gate. Note that in Figure 2(b), we also include another teleportation to move the state teleported to q'_{c0} back to q_0. Essentially, the second teleportation is used to handle the side effect of TP-Comm: the communication qubit q'_{c0} will be occupied by the teleported state of q_0 and thus cannot be used to establish new EPR pairs. We need the second teleportation to set q'_{c0} free. For the second teleportation, besides moving the teleported state of q_0 back to its original location as in Figure 2(b), we can also move the teleported state to any other location as long as the occupation on the current communication qubit q'_{c0} is released. Overall, two EPR pairs are required to implement one remote CX gate by TP-Comm, with one of them handling the side effect. To avoid ambiguity, when we say *one invocation of TP-Comm*, we actually refer to one invocation of quantum teleportation. That's to say, one invocation of TP-Comm consumes one EPR pair, just like one call of Cat-Comm. Thus, to implement one remote CX, we would need two invocations of TP-Comm.

Figure 2 only shows how to implement one remote CX gate. To implement a complex remote interaction between quantum nodes, one simple strategy is to first decompose the remote interaction into several remote CX gates and implement each remote CX as in Figure 2. However, this strategy may incur heavy communication costs. Eisert et al. observe that optimized implementations exist for specific inter-node interactions, as shown in Figure 3. Our framework will cleverly take advantage of the implementations in Figure 3 based on our observation of burst communication, to optimize the communication cost of distributed quantum programs, as we can see in later sections.

III. Problem and Motivation

In this section, we first introduce the communication problem in DQC and then identify optimization opportunities by considering burst communication. For the rest of the discussion, we assume that quantum communication can be established between any two quantum nodes, a typical assumption in data-center distributed computing [21]. We also assume that each quantum node has only two communication qubits, which is realistic for near-term DQC [14].

A. Communication Problem

The example distributed program in Figure 4 is modified from quantum arithmetic circuits [22]. This program contains many remote CX gates, e.g., $CX\,q_1, q_3$. Remote CX gates are inevitable in DQC especially when the program's qubit number is substantially larger than that of each quantum node. To execute a distributed program, we need to invoke either Cat-Comm or TP-Comm to implement remote operations, as shown in Figures 2 and 3. Due to the noisy nature of quantum communication, remote operations are far more error-prone than local quantum gates. The long runtime of quantum communication would also lead to the decoherence of quantum states. As a result, to produce high fidelity outcome, we hope the number of remote communication to be as small as possible, and so is the latency induced.

As indicated in Section II, one remote CX gate requires at least one remote EPR pair. While there is little room for optimizing the communication cost of one remote CX, there is a large optimization space when considering burst communication, which involves a group of remote CX gates. For example in Figure 4, we can execute the first two CX gates on q_1, q_3 collectively, with only one EPR pair by using the circuit in Figure 3(a). From the perspective of information theory, burst communication is more informative

than communication that carries only one remote CX. The overall communication overhead would be considerably lowered if handling all remote CX gates in this burst manner.

Fortunately, as we can see in the next section, burst communication is prevalent in diverse distributed quantum programs.

B. Burst Communication in DQC

Aside from the arithmetic program shown in Figure 4, we also see burst communication in a variety of quantum programs. As examples, we examine the burst communication of the Quantum Fourier Transform (QFT) program [3] and the Quantum Approximate Optimization Algorithm (QAOA) [23] by hand. These two represent different categories of quantum programs: QAOA is one of the most important applications in near-term quantum computing whereas QFT is the building block circuit of quantum algorithms.

We first give a formal definition of burst communication in DQC. In this paper, we refer to a group of continuous remote two-qubit gates between one qubit and one node as *burst communication*. For two remote two-qubit gates g_1 and g_2, the continuity of these two gates means there are no other remote gates between g_1 and g_2.

To characterize the burst communication of a distributed program $dprog$, for a remote gate g in $dprog$, we define function $\varepsilon(g)$ to be the largest burst communication block that contains g. The gate order of $dprog$ may affect the burst communication block found. $\varepsilon(g)$ is defined to be the largest over all functional-equivalent gate order of $dprog$. We then define $len(\varepsilon(g))$ to be the number of remote CX gates in $\varepsilon(g)$ if compiled to the CX+U3 basis [16]. Finally, we are ready to define the *inverse-burst distribution* as follows:

$$P(x) = \frac{|\{g | len(\varepsilon(g)) < x\}|}{\#g}. \tag{1}$$

A lower $P(x)$ suggests more burst communication. Specifically, for a given x, the lower $P(x)$ is, the larger $1-P(x)$ is and the more remote CX gates belong to burst communication blocks that each possesses more than x remote CX gates, indicating more burst communication opportunities. On the other hand, for the given probability $P_0 = P(x)$, we hope the corresponding x to be as large as possible since it means there are $1-P_0$ of remote CX gates belonging to burst communication blocks that each possesses more than x remote CX gates. The distribution $1-P(x)$ actually provides an ideal upper bound for the burst communication existing in distributed quantum programs and can serve as a metric to evaluate the communication efficiency of various distributed quantum algorithms.

We begin by examining the QFT program using the aforementioned definition. We assume the total qubit number is n, the quantum node number is k, and qubits are evenly distributed across all nodes, with $t = \frac{n}{k}$ qubits per node. Figure 5 shows the QFT program with $k = 2$ and $t = 2$. For the QFT program, each q_i is controlled by all qubits

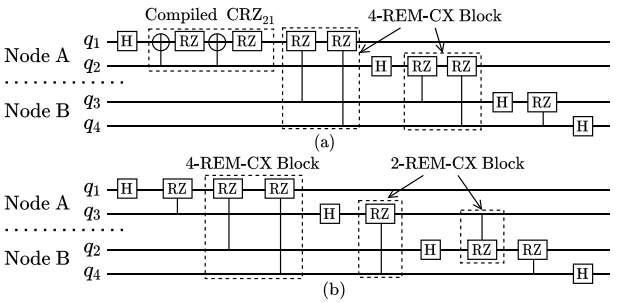


Figure 5. (a) QFT program with two nodes and two qubits per node. (b) The layout for the maximal P_4. Parameters of CRZ gates are omitted here for simplicity. For the purpose of demonstration, we do not combine CRZ_{43} and CRZ_{32} to form a 4-REM-CX block. The 4-REM-CX (or 2-REM-CX) block denotes the gate block which contains four (resp. two) remote CX gates when compiled to the CX+U3 basis.

q_j (through the CRZ gate) that satisfies $j > i$ [3], as shown in Figure 5. Then, we have $P(2) = 0$ because each CRZ gate in QFT is compiled into two CX gates, as illustrated in Figure 5(a). Now, we consider $P(4)$. For the i-th qubit satisfies $i \le n-k$, the number of j s.t. $\varepsilon(\mathrm{CRZ}_{ji}) < 4$ is at most $\lfloor \frac{i-1}{t-1} \rfloor$ because for one node, if at least two of its qubits have subscripts $> i$, this node would have at least two qubits interacting with qubit i. Since CRZ gates are commutable with each other, we could then form a communication block with at least 4 CX gates. It's easy to see that there are at most $\lfloor \frac{i-1}{t-1} \rfloor$ nodes, each containing no more than one qubit with subscript $> i$. On the other hand, if $i > n-k$, then the i-th qubit is at most controlled by $n-i$ qubits, thus the number of j s.t. $\varepsilon(\mathrm{CRZ}_{ji}) < 4$ is at most $n-i$. Therefore, we have

$$P(4) \le \frac{\sum_{i=1}^{n-k} \lfloor \frac{i-1}{t-1} \rfloor + \sum_{i=n-k+1}^{n} (n-i)}{\sum_{i=1}^{n} (n-i) - k\sum_{l=1}^{t} (t-l)} = \frac{1}{t}.$$

This indicates there are $1-P(4) \ge 1-\frac{1}{t}$ remote gates within burst communication blocks that each possesses more than 4 remote CX gates. Generally, we can prove that $P(x) \le \frac{x/2-1}{t}$, for $x > 2$. This upper bound is quite promising when t is large and it is actually loose. For Figure 5(b) which approximates the upper bound of $P(4)$, there may be $\frac{1}{t}$ of remote CRZ gates, i.e., CRZ_{43} and CRZ_{32} not in a block with 4 remote CX gates at the first glance. But as shown in Figure 5(b), we can actually combine CRZ_{43} and CRZ_{32} to form a 4-REM-CX block (the block that contains 4 remote CX gates when compiled to the CX+U3 basis). More 4-REM-CX blocks or n-REX-CX blocks ($n > 4$) can enable more burst communication opportunities. This indicates that the distributed QFT program has more abundant burst communication than the upper bound of $P(x)$ (thus the lower bound of $1-P(x)$) suggests.

Similarly, for the QAOA program, we assume k nodes and t qubits per node. We also suppose r remote ZZ interactions between any two nodes. Figure 6 shows the QAOA program with $k = 2$ and $t = 3$. Likewise, $P(2) = 0$ since each ZZ

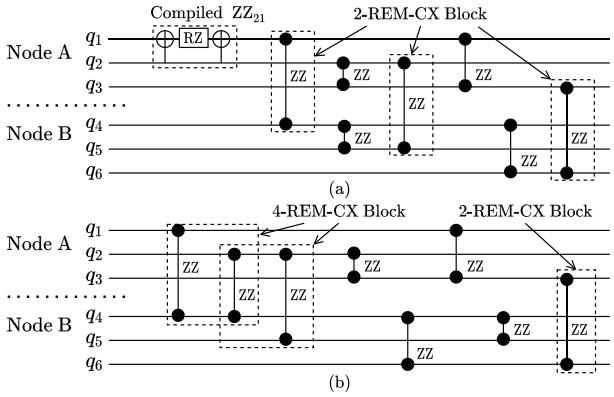


Figure 6. QAOA program with two nodes and three qubits per node. Parameters of ZZ interactions are omitted here for simplicity. (a) inter-node communication number $r = 3$. (b) $r = 4$.

interaction is compiled into two CX gates, as shown in Figure 6(a). For every two nodes, the qubit layout to minimize $len(\varepsilon(ZZ))$ for each ZZ interaction is to make every two ZZ interactions have no shared qubits, i.e., not adjacent. However, this layout at most accommodates t ZZ interactions. For $r > t$, considering $m^2 + (t-m) < r \leq (m+1)^2 + (t-m-1)$, there are at most $t-m-1$ ZZ interactions that are not adjacent to any other ZZ interactions. Thus, we have $P(4) = \frac{t-m-1}{r} < \frac{t-m-1}{m^2+(t-m)}$. For example in Figure 6(b), we have $1^2 + (t-1) = 3 < r < 2^2 + (t-2) = 5$, thus we predict $P(4) < \frac{t-1-1}{1^2+(t-1)} = \frac{1}{3}$. This bound is correct because in Figure 6(b), only $\frac{1}{4}$ of remote ZZ interactions are not in a 4-REM-CX block. It is easy to see that $P(4)$ would quickly decrease if m becomes large. For the general $P(x)$, a similar conclusion can be reached. Therefore, burst communication is also broadly available in the distributed QAOA program.

We could derive a similar analysis for other distributed quantum programs. Further numerical evidence for the richness of burst communication in various distributed quantum programs is shown in Figure 15. The next step is to figure out how to utilize abundant burst communication to optimize the communication cost of distributed quantum programs, as discussed in the next section.

C. Optimization Opportunities

To exploit burst communication in distributed quantum programs, we need to answer three key questions:

How to unveil burst communication?

Burst communication is a type of high-level program information and cannot be easily deduced from the low-level circuit language, especially when remote interactions between multiple nodes are all mixed together. For example in Figure 4, gate $CX\,q_2;q_4$ between node A and node B is followed by $CX\,q_1;q_6$, which is the interaction between node A and node C. Such a disorder in distributed quantum programs causes great difficulty in utilizing the benefits of burst communication.

How to select the best communication scheme?

Burst communication comes in various forms. While being more efficient for implementing one remote CX gate, Cat-Comm is not always better than TP-Comm for burst communication. For example in Figure 4, if we use Cat-Comm to implement the last three remote CX gates between q_3 and node A, three EPR pairs are needed. However, with TP-Comm to teleport q_3 to node A, at most two EPR pairs are needed. Thus, to reduce the communication cost, we should examine the pattern of burst communication and choose the communication scheme wisely.

How to schedule burst communication?

Finally, we need to schedule the execution of burst communication blocks. If we arrange all burst communication in a sequential way, the large time overhead would impose non-negligible decoherence errors on quantum states. As a result, we should maximize the parallelism in burst communication to generate high-fidelity output. To achieve this goal, we should first identify the relationships between communication blocks and then reduce the time gaps caused by communication blocks adaptively.

IV. AutoComm Framework

In this section, we first give an overview of the AutoComm framework and then introduce each component in detail.

A. Design overview

We propose the *AutoComm* framework as shown in Figure 1. AutoComm focuses on the communication optimization of distributed quantum programs and serves as the back-end of front compiling flows (e.g., mapping qubits to quantum nodes). We would adopt existing technologies for these front compiling stages, as we would see in Section V.

To reduce the communication overhead in distributed quantum programs, AutoComm comes with three stages to utilize burst communication. Firstly, it aggregates remote two-qubit gates by gate commutation. Gate commutation is common in quantum programs [24]. Commutable gates, on the one hand, may be ordered arbitrarily and hide the burst communication. On the other hand, we could also utilize gate commutation to uncover burst communication blocks. In this stage, we first identify potential burst communication and then employ a linear merge step to combine isolated burst communication blocks.

Secondly, it assigns an optimal communication scheme for each burst communication block. We observe that the pattern of burst communication impacts the efficiency of communication schemes. Cat-Comm is less expensive for some patterns, while TP-Comm may be more cost-effective for others. It is thus important to examine the patterns of burst communication and consider both Cat-Comm and TP-Comm for implementing them, rather than only focusing on one scheme.

Thirdly, it performs a block-level schedule of burst communication. It is possible to execute communication blocks with shared nodes or qubits concurrently or shorten the

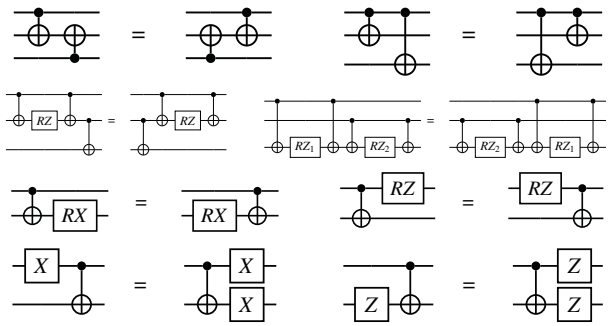


Figure 7. Some representative gate commutation rules used in AutoComm.

quantum state transfer path across quantum nodes. Combined with these optimizations, a greedy schedule is effective for burst communication.

Algorithm 1: Linear merge procedure

Input: An array of communication blocks *blk_list*
Output: Merged communication blocks *blk_list_merge*

```
blk_list_merge = [];
blk = blk_list[0] ;
while there are blocks in blk_list not visited do
    non_commute_gates = [];
    for blk_next in unvisited blocks of blk_list do
        // Attempt merge blk to blk_next
        for gate between blk and blk_next do
            if gate is single-qubit and not commutes with blk
              then
                non_commute_gates.append(gate);
            if gate is two-qubit then
                check if gate is commutable with
                  non_commute_gates and blk;
                if not commutable then
                    if gate is in-node two-qubit then
                        non_commute_gates.append(gate);
                    else
                        break;
        end
        blk =merge blk, non_commute_gates and blk_next;
    end
    if the above merge failed then
        Try to merge blk_next to blk similarly;
        if succeeds then
            blk =merge blk, non_commute_gates and blk_next;
        else
            blk = blk_next;
end
output the merged blocks and adjust the order of commutable gates;
```

B. Communication Aggregation

Burst communication is prevalent in distributed programs, but may not be immediately available due to two factors: CX gates may be scattered across the program, and whether CX gates are remote or not depends on the qubit mapping to quantum nodes. To make our framework able to uncover hidden burst communication regardless of qubit mappings, we need to rewrite the circuit and aggregate remote CX gates.

Figure 7 shows a fraction of circuit rewriting rules for remote gate aggregation. The first two rows of Figure 7 contain gate commutation rules for two-qubit gates, from simple ones to complex ones. These rules enable flexible two-qubit gate relocation so that burst communication can be automatically exposed and utilized. The remaining two rows of Figure 7 are about exchanging single-qubit gates with the CX gate and affect the pattern of the aggregated burst communication (more details in Section IV-C). Based on these circuit rewriting rules, we design the following steps to aggregate remote gates.

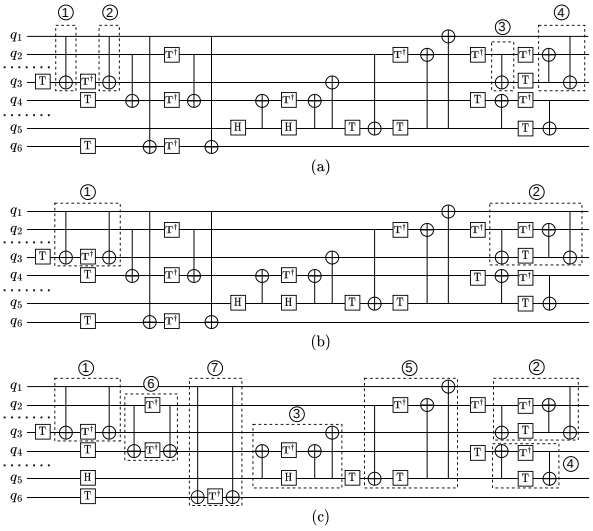


Figure 8. Communication aggregation for the example program in Figure 4. (a) Identifying potential burst communication. (b) Linear merge. (c) Iterative refinement.

(a) Identifying potential burst communication

As burst communication is defined between one qubit and one node, the first step of communication aggregation is thus to identify the qubit-node pair of potential burst communication. We start with the qubit-node pair associated with most remote gates as it would likely lead to a large burst communication block. For example in Figure 4, the chosen qubit-node pair is (q_3, node A) as it is associated with 5 remote CX gates. We then search for consecutive remote CX gates related to this qubit-node pair. In this step, circuit rewriting is not applied yet and the search would result in many isolated communication blocks. For example in Figure 8(a), we obtain four small blocks.

(b) Linear merge

The next step is to merge isolated small communication blocks obtained in step (a). As illustrated in Algorithm 1, we merge related communication blocks in a linear and greedy manner. For communication blocks ①, ②, ③, ④ in Figure 8(a), we can easily merge block ① and ② since only single-qubit gates exist between those two blocks. We call denote the merged block of ① and ② by *blk_new*. Unfortunately, we can not merge block *blk_new* and block ③. On the one hand, $CX\,q_5, q_3$ is not commutable with *blk_new*,

so we cannot move *blk_new* close to ③. On the other hand, $CX\,q_5, q_2$ is not commutable with ③, making it impossible to move ③ close to *blk_new*. We then skip *blk_new* and start from block ③ to find other merge opportunities. The linear merge procedure will visit all blocks at least once. Finally, we obtain two larger communication blocks after the linear merge, as shown in Figure 8(b).

(c) Iterative refinement

Then we repeat steps (a) and (b) for other qubit-node pairs until no more merge opportunities exist. The final result of communication aggregation is shown in Figure 8(c). Identified burst communication blocks are ordered by the time being discovered.

C. Communication Assignment

With burst communication blocks, the next optimization is to find the best way to execute them. We address this problem by first examining the pros and cons of Cat-Comm and TP-Comm, and then assigning the optimal communication scheme based on the pattern analysis of burst communication blocks. Since we assume only two communication qubits in each quantum node, the communication patterns discussed here center on interactions between one qubit and one node. Extending burst communication to the node-to-node situation is promising when communication qubits are plentiful. We leave it for future work.

Cat-Comm vs. TP-Comm:

Suppose we have a burst communication block between q_1 in node A and several qubits in node B, with n remote CX gates totally. If the block can be executed by one invocation of Cat-Comm, the savings on EPR pairs would be up to n times, compared to executing each remote CX gate individually. However, Cat-Comm only supports controlled-unitary blocks and needs many (≥ 2) EPR pairs to implement communication blocks not being controlled-unitary. Compared to Cat-Comm, TP-Comm can implement any burst communication block with two EPR pairs: one to teleport q_1 to node B, the other to teleport q_1 back to node A, in order to handle the side effect of TP-Comm. There are cases we would like to teleport q_1 to another node instead of simply moving it back. We postpone the details to Section IV-D. Compared to Cat-Comm, the disadvantage of TP-Comm is that its EPR pair saving is at most $\frac{n}{2}$ times. Overall, Cat-Comm provides higher EPR pair savings for specific burst communication blocks, while TP-Comm can handle any burst communication block with up to two EPR pairs.

Pattern analysis:

Figure 9(a)(b) shows two primitive patterns of burst communication. For the unidirectional communication pattern in Figure 9(a) where one qubit, i.e., q_1 always serves as the control qubit, the communication block can be implemented by Cat-Comm with only one EPR pair if no single-qubit gate on q_1 separates two-qubit gates [12]. For example, one call of Cat-Comm can handle the gate sequence $CX\,q_1, q_1'; CX\,q_1, q_2'$

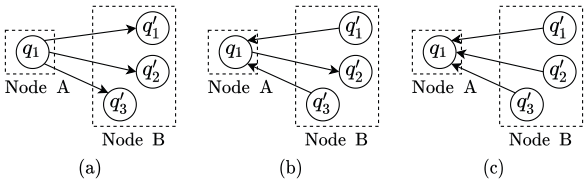


Figure 9. Two primitive communication patterns (a)(b) and the variant (c).

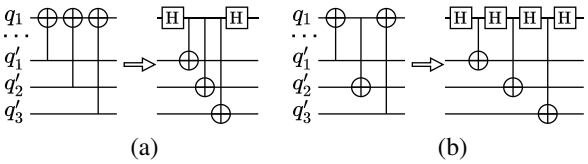


Figure 10. The transformation between communication patterns by using Hadamard gates.

, but cannot address $CX\,q_1, q_1'; RZ\,q_1; CX\,q_1, q_2'$ due to the middle RZ gate. However, by moving RZ behind $CX\,q_1, q_2'$, the resulted gate sequence $CX\,q_1, q_1'; CX\,q_1, q_2'; RZ\,q_1$ only requires one invocation of Cat-Comm. Thus, to execute a communication block of unidirectional pattern with Cat-Comm, we should move single-qubit gates on the control qubit outside the communication block. Otherwise, we resort to TP-Comm. For example, to implement $CX\,q_1, q_1'; H\,q_1; CX\,q_1, q_2'; H\,q_1; CX\,q_1, q_3'$, it is better to use TP-Comm.

A varied unidirectional pattern in which q_1 always serves as the target qubit, as shown in Figure 9(c), also occurs frequently in distributed quantum programs. This pattern can be transformed into the pattern in Figure 9(a) by applying a series of Hadamard gates, as shown in Figure 10(a).

Figure 9(b) shows a bidirectional pattern in which q_1 serves as both control qubit and target qubit. Although we can transform a bidirectional communication block to be unidirectional as in Figure 10(b) with Hadamard gates, single-qubit gates on the control qubit may still prevent a cheap implementation by Cat-Comm. In fact, for the block in Figure 10(b), TP-Comm is more efficient as it only requires two EPR pairs, while Cat-Comm requires three EPR pairs.

To summarize, for unidirectional communication patterns in Figure 9(a)(c), we will try Cat-Comm first, while for the bidirectional pattern in Figure 9(b), TP-Comm is mostly preferred. The insight behind the conclusion is that, analogous to classical distributed computing, Cat-Comm only shares its read-only copy to another node, thus it is not a natural fit for bidirectional communications which involve read and write on the shared qubit. TP-Comm, in contrast, migrates data to another node, allowing read and write operations on the migrated qubit.

Communication scheme assignment:

Now, we are ready to assign an optimal communication scheme to each burst communication block. Taking Fig-

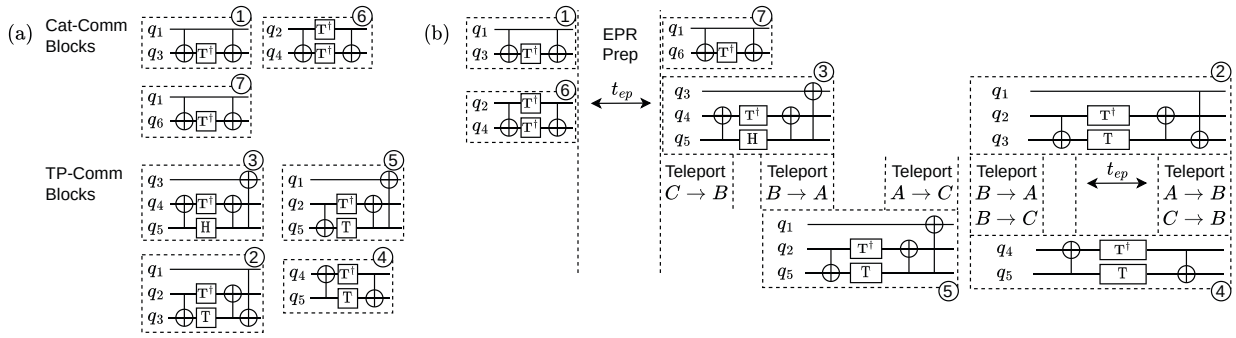


Figure 11. (a) The result of the communication assignment pass. (b) The result of the communication scheduling pass.

ure 8(c) as an example, we assign Cat-Comm to unidirectional blocks ①, ⑥ (we can move the $T^\dagger$ gate on q_2 outside the communication block) and ⑦. We call these blocks *Cat-Comm blocks* for simplicity. We then assign TP-Comm to bidirectional blocks ②, ④ and ⑤. Likewise, we call them *TP-Comm blocks*. For ③, although being unidirectional, it cannot be executed by one invocation of Cat-Comm due to the H gate on q_5. Since executing it with either Cat-Comm or TP-Comm requires two EPR pairs, we set the TP-Comm assignment as default. The finalized communication assignment is shown in Figure 11(a).

D. Communication Scheduling

After optimizing the EPR pair consumption, we then schedule the execution of burst communication blocks to reduce the overall program latency and mitigate the effect of decoherence. Based on the quantitative data shown in Table I, the preparation of remote EPR pairs is the most time-consuming one among various operations and hence should be carefully optimized to hide its latency. While the quantitative data may vary across quantum computing platforms, the schedule design in this section should be also effective.

Operation	Variable Name	Latency
Single-qubit gates	t_{1q}	$\sim$ 0.1 CX
CX and CZ gates	t_{2q}	1 CX
Measure	t_{ms}	5 CX
EPR preparation	t_{ep}	$\sim$ 12 CX
One-bit classical comm	t_{cb}	$\sim$ 1 CX

Table I
THE QUANTITATIVE DATA OF OPERATIONS IN DQC, EXTRACTED FROM [25], [26]. LATENCIES ARE NORMALIZED TO CX COUNTS.

The designs here aim to maximize the parallelism of communication blocks and shorten the latency of sequential communication by fusion.

More block-level parallelism:

The essence of scheduling is to maximize the parallelism of a circuit. For burst communication blocks without nodes or qubits in common, they can be concurrently executed in nature. For blocks with shared nodes or qubits, their parallelism is limited by their commutability, as well as the communication resource each node holds. With the constraint that each node can establish only two communications in parallel, there is little room for lazy operations. We adopt a greedy strategy to execute commutable blocks, i.e., execute as many blocks as possible simultaneously, as soon as EPR pairs are prepared.

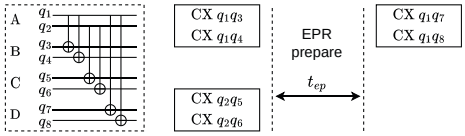


Figure 12. The schedule optimization for commutable Cat-Comm blocks, with shared qubit or node.

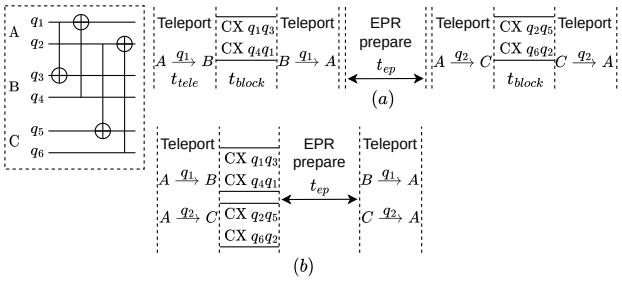


Figure 13. The schedule optimization for TP-Comm blocks. Aligned qubit teleportation in (b) is better than the independent qubit teleportation in (a).

For Cat-Comm blocks, we can execute two commutable blocks in parallel at most if they share nodes, as shown in Figure 12. For TP-Comm blocks, the situation is complex as each TP-Comm block requires two EPR pairs. For two commutable TP-Comm blocks, rather than prioritizing the completion of one TP-comm block as in Figure 13(a), we observe that parallelism can be enabled by communication alignment, as shown in Figure 13(b). Compared to Figure 13(a), Figure 13(b) aligns the qubit teleportation of two TP-Comm blocks, leading to a latency saving of

$t_{block}+2t_{tele}$. This TP-Comm alignment technique can be generalized to the case of n commutable TP-Comm blocks. With TP-Comm alignment, the total latency saving can be up to $(n-1)(t_{block}+2t_{tele})$ (e.g., if those TP-Comm blocks are on nodes $\{A_1,A_2\},\{A_2,A_3\},\cdots,\{A_n,A_{n+1}\}$ respectively).

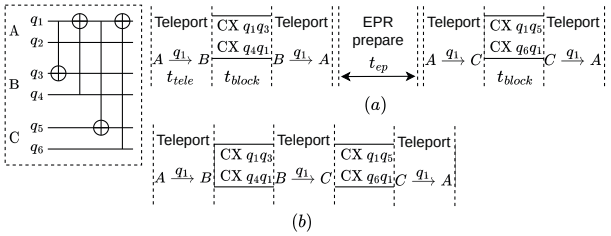


Figure 14. The schedule optimization for TP-Comm blocks. Cyclic qubit teleportation in (b) is better than the SWAP-style qubit teleportation in (a).

Fusion of sequential blocks:

Sometimes communication blocks have to be executed in sequence. However, if one qubit is teleported across many TP-Comm blocks, we can shorten the latency of executing those TP-Comm blocks by fusing the teleportations, as shown in Figure 14. Figure 14(a) shows a simple schedule where each TP-Comm block is executed independently. As each node has only two communication qubits, we need to wait for t_{ep} before executing the next TP-Comm block. In contrast, Figure 14(b) fuses the teleportations between quantum nodes, forming a cycle: $A\rightarrow B\rightarrow C\rightarrow A$. With TP-Comm fusion, the number of teleportations is reduced by one and the overall execution time is reduced by $t_{ep}+t_{tele}$. Generally, if we have n TP-Comm blocks with the same teleported qubit, the total number of teleportation would be reduced by $n-2$, and the overall latency saving would be $(n-2)(t_{ep}+t_{tele})$. From another view, the fusion also optimizes the token passing problem in classical distributed computing [21], which also appears in Section IV-C, about whether to move the teleported qubit back or to another node, in order to handle the side effect of TP-Comm.

With the designs above, the communication scheduling pass should apply block-level commutation analysis to unveil the patterns discussed above and then apply corresponding optimizations. We omit the details since this procedure is very similar to the communication aggregation except working at the block level. With all those optimizations applied, Figure 11(b) shows the optimized communication schedule for the example program in Figure 4. In total, 58.3% latency saving is achieved compared to executing each remote CX gate independently.

V. Evaluation

In this section, we first compare the performance of AutoComm to two baselines and then evaluate the effect of optimization passes in AutoComm. We finally perform a sensitivity analysis on AutoComm to study how its performance evolves as the experiment configuration changes.

A. Experiment Setup

(a) Platforms

We perform all experiments on a Ubuntu 18.04 server with a 6-core Intel E5-2603v4 CPU and 32GB RAM. Other software includes Python 3.8.3 and Qiskit 0.18.3 [16].

(b) Benchmark programs

We consider two categories of benchmark programs, as shown in Table II. The first category of benchmarks focuses on implementing elementary functions, e.g., arithmetic operations and Fourier transformation. These quantum programs are often used as building blocks of large quantum applications. The second category of benchmarks aims to solve real-world problems, including Bernstein-Vazirani (BV) algorithm, Quantum Approximate Optimization algorithm (QAOA), and Unitary Coupled Cluster ansatzes (UCCSD). Specifically, for BV, we choose 1000 randomized secret strings which on average contain $\frac{2}{3}\#qubit$ nonzeros. For QAOA, we choose the graph maxcut problem (over 1000 randomized graphs), and for UCCSD, we select molecules LiH, BeH_2, and CH_4 which correspond to programs with 8, 12 and 16 qubits, respectively. All benchmark programs used in the evaluation are collected from IBM Qiskit [16] and RevLib [22].

(c) Baseline

We implement two state-of-the-art compilers, GP-Cat and GP-TP, which, to the best of our knowledge, represent the best efforts for distributed quantum compilation. GP-Cat implements one of the compiler designs proposed by [14] which exploits the Cat-Comm scheme for remote CX gates but does not consider burst communication. We did not extend GP-Cat to use TP-Comm as TP-Comm is not efficient at implementing a single remote CX gate. For the GP-TP baseline, we adopt a similar compiler design to [10], [14] where nonlocal operations are turned into local operations by swapping qubits between nodes. In GP-TP, we use TP-Comm to implement nonlocal qubit swapping operations as TP-Comm is better at implementing the remote SWAP gate than Cat-Comm. For both baselines and AutoComm, we map qubits to compute nodes by the 'Static Overall Extreme Exchange' (Abbrev. *SOEE*) strategy [10], which aims to reduce inter-node communication. To reduce the program latency, the baselines adopt a greedy scheduling method, i.e., executing operations as soon as possible.

(d) Distributed quantum computing model

We assume a uniform DQC system. Each node in the DQC system has the same number of qubits. The fidelity of EPR pairs between any two nodes is the same, and so is the latency of preparing them. We also assume that the EPR pair can be established between any two nodes and each node has *two communication qubits* as in [10], [20]. Since our framework focuses on communication optimization, we assume a trapped-ion style device [6] for each compute node that any two local qubits can communicate with each other. Our work can also be easily applied to superconducting

devices [27] with sparse two-qubit connections.

For Table II, we assume each node has 10 data qubits for benchmark programs except UCCSD. For UCCSD, we assume each node has 2 data qubits. For all experiments, we assume that qubits of the test program are evenly distributed over all nodes.

(e) Metric

The first metric is the total number of consumed EPR pairs for executing a distributed quantum program. Each invocation of Cat-Comm or TP-Comm requires one EPR pair. Note that without TP-Comm fusion, two EPR pairs are needed to execute one burst communication block with the TP-Comm, with one of the EPR pairs moving the teleported qubit back to its original node. The number of consumed EPR pairs models the resource overhead of executing distributed quantum programs and a lower value is favored.

The second metric is the maximum number of inter-node two-qubit gates got executed with one EPR pair. We denote this metric by 'Peak # REM CX'. To give a concrete example, assuming a distributed quantum program where the largest Cat-Comm block contains 10 remote CX gates and the largest TP-Comm block contains 18 remote CX gates, if without TP-Comm fusion, then for this program, 'Peak # REM CX' is $10 = \max(10, 18/2)$. If we assume the largest TP-Comm block is fused with the next TP-Comm block, then 'Peak # REM CX' is $18 = \max(10, 18)$ because TP-Comm fusion reduces the EPR pair consumption of a TP-Comm block. The metric 'Peak # REM CX' characterizes the communication throughput and a higher value is preferred.

Finally, we consider two metrics that model the relative performance of AutoComm to baselines, with respect to EPR consumption and program latency. The first one is the 'improv. factor', which is defined to be '# total EPR pairs by baseline/# total EPR pairs by AutoComm'. The second one is the 'LAT-DEC factor' that is defined to be 'program latency by baseline/program latency by AutoComm'. The target of AutoComm is to make these two metrics as large as possible.

B. Compared to Baselines

We first analyze the ability of AutoComm in exposing burst communications, with communication statistics shown in Figure 15. We then evaluate AutoComm and two baselines on benchmark programs in Table II. The results of AutoComm and its relative performance to GP-Cat and GP-TP are shown in Table III. When we say *on average* in this section, we refer to the *geometric mean*.

Burst communication statistics:

Figure 15 shows the distribution of burst communications assembled by AutoComm. This distribution is closely related to the inverse-burst distribution discussed in Section III-B but is easier to compute. We can see that burst communications exist widely, no matter in building-block circuits (Figure 15(a)) or in real-world applications (Figure 15(b)). Moreover, Figure 15 demonstrates the effectiveness of AutoComm in unveiling burst communications. In Figure 15, the EPR pairs that each support ≥ 2 remote CX gates account for 76.8% of the overall consumed EPR pairs, on average.

Type	Name	# qubit	# node	# gate	# CX	# REM CX by SOEE
Building Blocks	Multi-Controlled Gate (MCTR)	100	10	10640	4560	1680
		200	20	21840	9360	3568
		300	30	33040	14160	5632
	Ripple-Carry Adder (RCA)	100	10	1569	785	99
		200	20	3169	1585	209
		300	30	4769	2385	319
	Quantum Fourier Transform (QFT)	100	10	19800	9900	9000
		200	20	79600	39800	38000
		300	30	179400	89700	87000
Real World Applications	Bernstein Vazirani (BV)	100	10	265	65	56
		200	20	535	135	126
		300	30	803	203	194
	QAOA	100	10	6000	4000	3144
		200	20	24000	16000	14076
		300	30	54000	36000	32896
	UCCSD	8	4	3129	1420	900
		12	6	40659	19142	15136
		16	8	129829	64956	53426

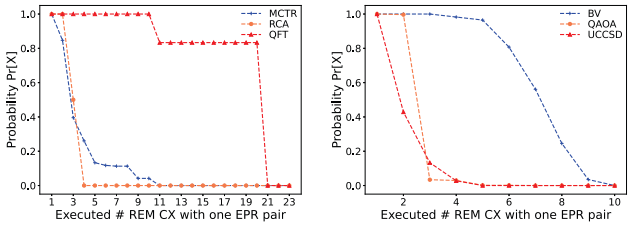
Table II
BENCHMARK PROGRAMS. # QUBIT IS THE TOTAL NUMBER OF QUBITS AND EACH NODE HAS EXACTLY '# QUBIT/# NODE' DATA QUBITS.

Compared to GP-Cat:

As shown in Table III, AutoComm achieves significant reduction in both EPR pair consumption and program latency, compared to GP-Cat. Specifically, AutoComm reduces the number of consumed EPR pairs by a factor of 3.9x on average, up to 18.8x (ref. 'Improv. factor'). AutoComm also reduces the program latency by a factor of 3.1x on average, up to 9.4x (ref. 'LAT-DEC factor'). These significant improvements come from the high communication throughput enabled by AutoComm. In GP-Cat, each EPR pair, i.e., each invocation of Cat-Comm is used to implement only one remote CX gate. In contrast, the peak communication throughput (ref. 'Peak # REM CX') by AutoComm is 7.2x on average and up to 20x of that by GP-Cat. Those results indicate that AutoComm can efficiently discover and utilize burst communications, transferring more information with each EPR pair.

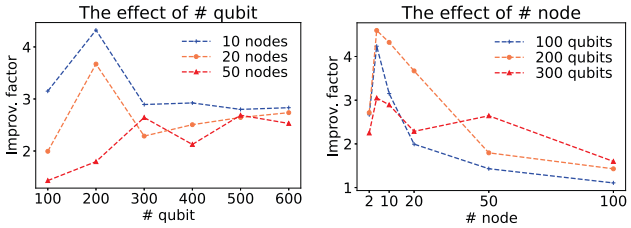
Compared to GP-TP:

As shown in Table III, AutoComm achieves significant reduction in both EPR pair consumption and program latency, compared to GP-TP. Specifically, AutoComm reduces the number of consumed EPR pairs by a factor of 3.5x on average, up to 13.3x. AutoComm also reduces the program latency by a factor of 3.4x on average, up to 10.7x. On the side of information theory, AutoComm enables a higher throughput of information. Each EPR pair in GP-TP carries 3/2 remote CX gates (i.e., a remote SWAP gate over two EPR pairs), much smaller than the throughput by AutoComm. On the algorithmic side, AutoComm avoids unnecessary qubit movement. For example, consider the gate sequence CX q_1, q_2; CX q_1, q_3; CX q_1, q_4; CX q_2, q_1 where q_1 is in



(a) (b)

Figure 15. Burst communications by AutoComm: Pr[X] = Pr[one EPR pair supports $\geq$ X REM-CXs].



(a) (b)

Figure 16. The effects of (a) #qubit and (b) #node on the 'improv. factor' of AutoComm when compared to GP-Cat. The test program is MCTR.

node A, q_2 is in node B, and q_3, q_4 are in node C. To execute these remote gates, GP-TP needs to swap q_1 into node B first, then to node C, and back to node B again. However, with AutoComm, we only need to first move q_1 to node C and then to node B, since CX q_1, q_2 is commutable with CX q_1, q_3 and CX q_1, q_4.

C. Optimization Analysis

In this section, we further analyze the effectiveness of each optimization pass in AutoComm. Again, when we say *on average* in this section, we refer to the *geometric mean*.

For simplicity, we denote the communication aggregation pass by *P1*, the assignment pass by *P2*, and the scheduling pass by *P3*. We first study how P1 and P2 affect the 'improv. factor' of AutoComm to GP-Cat, then evaluate how P3 affects the 'LAT-DEC factor'. The results are shown in Table IV. We do not compare P2 to GP-Cat directly as P2 cannot work properly without communication aggregation.

The effect of communication aggregation:

As shown in Table IV, compared to GP-Cat, 'P1+Cat-Comm' reduces the EPR pair consumption by a factor of 2.6x, on average. The result indicates the effectiveness of the communication aggregation pass in reducing the communication cost by grouping remote CX gates into a burst communication block. On the other hand, this analysis also shows that burst communication may not be readily available in distributed quantum programs and we need the communication aggregation pass to unveil them.

The effect of communication assignment:

As shown in Table IV, compared to 'P1+Cat-Comm', 'P1+P2' further reduces the EPR pair consumption by a factor of 1.4x, on average. The result demonstrates the importance of considering both Cat-Comm and TP-Comm for burst communication. The benefit of P2 is even more significant for programs where bidirectional communication patterns appear frequently, e.g., RCA and QFT. This is because Cat-Comm is not as efficient as TP-Comm for implementing bidirectional burst communication.

The effect of communication scheduling:

As shown in Table IV, compared to 'P1+P2', 'P1+P2+P3' further reduces the program latency by a factor of 1.1x, on average. The result illustrates the effectiveness of P3 in reducing communication-induced latency. The effectiveness of P3 for scheduling burst communication stems from its smart utilization of communication qubits, especially for TP-Comm blocks, as discussed in Section IV-D. As for programs comprised of Cat-Comm blocks, e.g., BV and UCCSD, P3 behaves as efficiently as the default as-soon-as-possible scheduling method.

D. Sensitivity Analysis

The performance of AutoComm may be affected by factors like the number of program qubits, the number of DQC nodes, the qubit mapping, and the heterogeneity of compute nodes. In this section, we study how the performance of AutoComm changes as those factors varies.

When evaluating the effect of #qubit and #node (ref. Figure 16), we assume program qubits are evenly distributed over all nodes: each node has exactly '#qubit/#node' data qubits. We also assume two communication qubits per node.

The effect of #qubit:

As shown in Figure 16(a), the 'improv. factor' of AutoComm converges when #qubit increases (i.e., #qubit/#node becomes large). The reason may be that the number of burst communication blocks also increases when the total number of remote multi-qubit gates grows with the number of program qubits. Such behavior is preferable because it illustrates that AutoComm can provide a consistent reduction of the communication overhead as the number of program qubits grows.

The effect of #node:

As shown in Figure 16(b), the 'improv. factor' of AutoComm deteriorates when #node increases (i.e., #qubit/#node becomes small). On the one hand, the remote multi-qubit gate would proliferate when #node increases, potentially providing more chances for burst communication. On the other hand, it is harder to find large burst communication blocks when #qubit/#node becomes small, instead increasing the communication overhead. Overall, we should not use too many nodes for distributing programs.

The effect of qubit mapping:

When evaluating the sensitivity to qubit mappings, we adapt two widely used algorithms, NoiseAdaptive [28] and SABRE [15] to benchmark programs in Table II. Such adaptations are straightforward as the DQC backend can also be described by the coupling graph. As shown in Figure 17(a),

Name: abbrev. – # qubit – # node	# Tot. EPR pairs consumed		Peak # REM CX	Compared to GP-Cat		Compared to GP-TP	
	By Cat-Comm	By TP-Comm		Improv. factor	LAT-DEC factor	Improv. factor	LAT-DEC factor
MCTR-100-10	313	220	10	3.15	3.27	2.81	3.90
MCTR-200-20	554	418	10	3.67	3.83	3.26	4.51
MCTR-300-30	932	1112	10	2.76	2.88	2.45	3.39
RCA-100-10	0	36	3	2.75	2.22	2.00	1.37
RCA-200-20	0	76	3	2.75	2.26	2.00	1.38
RCA-300-30	0	116	3	2.75	2.27	2.00	1.38
QFT-100-10	0	540	20	16.67	9.35	4.67	3.24
QFT-200-20	0	2090	20	18.18	9.40	5.27	3.26
QFT-300-30	0	4640	20	18.75	9.41	5.50	3.26
BV-100-10	9	0	8	6.22	4.33	12.22	9.68
BV-200-20	19	0	8	6.63	4.63	13.16	10.47
BV-300-30	29	0	8	6.69	4.69	13.31	10.65
QAOA-100-10	1182	266	6	2.17	1.83	1.56	2.09
QAOA-200-20	6059	728	8	2.07	1.79	1.57	2.52
QAOA-300-30	14915	1138	6	2.05	1.69	1.62	2.68
UCCSD-8-4	464	0	4	1.94	1.74	3.97	4.08
UCCSD-12-6	8973	0	4	1.69	1.55	3.10	3.31
UCCSD-16-8	33303	0	5	1.60	1.50	3.02	3.29

Table III
RESULTS OF AUTOCOMM AND ITS COMPARISON TO BASELINES. THE FIRST COLUMN CONTAINS ACRONYMS OF PROGRAMS IN TABLE II.

Name	Improv. factor compared to GP-Cat		LAT-DEC factor compared to GP-Cat	
	P1+Cat-Comm	P1+P2	P1+P2	P1+P2+P3
MCTR	3.05	3.17	2.76	3.30
RCA	1.88	2.75	2.25	2.25
QFT	2.22	10.00	7.14	9.39
BV	6.51	6.51	4.55	4.55
QAOA	2.08	2.10	1.65	1.77
UCCSD	1.74	1.74	1.59	1.59

Table IV
OPTIMIZATION ANALYSIS FOR AUTOCOMM. RESULTS ARE AVERAGED OVER PROGRAMS IN TABLE II. 'P1+P2+P3' IS JUST AUTOCOMM.

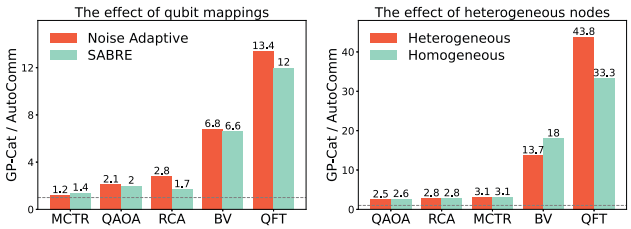


Figure 17. The effects of (a) qubit mappings and (b) heterogeneous nodes. Numbers in (a)(b) are averaged (geometric mean) 'improv. factor' of AutoComm to GP-Cat.

our framework still achieves significant communication cost reduction with NoiseAdaptive and SABRE. This indicates the practicality of AutoComm's two-step compilation design (ref. Figure 1), which enables us to focus on communication optimization while leveraging tons of existing efforts on qubit mapping.

The effect of heterogeneous nodes:

For this analysis, we consider two settings: the heterogeneous setting distributes each 100-qubit program over 4 nodes with 10, 20, 30, and 40 data qubits, respectively; the homogeneous setting evenly distributes each program over 4 nodes with 25 data qubits per node. As shown in Figure 17(b), our framework still achieves significant communication cost reduction on heterogeneous nodes. In the heterogeneous setting, nodes with few qubits limit the benefits of burst communication while nodes with many qubits boost them. These two effects cancel out each other and guarantee the performance of AutoComm.

VI. DISCUSSION AND FUTURE WORK

To the best of our knowledge, this paper is the first attempt that formalizes and optimizes burst communication in distributed quantum programs. Although our framework significantly surpasses existing works in optimizing communication, there is still much space left for potential improvements.

Extending to general collective communication:

This paper only considers the near-term DQC where communication qubits are supposed to be limited. In such a case, we are restricted to studying the qubit-to-node burst communication, which is a special case of general collective communication that involves a group of nodes. Assuming the availability of more communication qubits in the future, we could consider node-to-node collective communication which offers a potential optimization opportunity as we can now aggregate small qubit-to-node burst communication blocks into a large collective communication block.

Adapting to higher-level program abstraction:

This paper works with the low-level circuit language to maintain compatibility with existing compiling flows. However, if higher-level program information is provided, more aggressive communication optimization could be enabled. For example, if we know one inter-node circuit block is related to

a controlled-controlled unitary, we could implement it with at most two EPR pairs as at most two control qubits need to be shared by Cat-Comm. It is also promising to extend existing quantum programming languages to provide burst communication primitives which could expose extra burst communication that is difficult to uncover at a low level.

Combining with quantum error correction:

Since DQC involves quantum communication which is far noisier than local quantum gates, reinforcing the whole distributed quantum system with quantum error correction (QEC) becomes vital for future DQC. No matter encoding logical qubits in each node independently or forming a logical qubit with several nodes [29], [30], the implementation of logical gates (e.g., the logical CX) would involve a large number of physical qubits (possibly across several nodes) and provide great opportunities for burst communication optimization.

VII. Related Work

Most existing quantum compilers [15]–[19] focus on the compilation of programs within a single quantum computer. These works do not consider inter-node communication. Extending them to DQC cannot achieve high communication throughput in distributed quantum programs.

Unfortunately, existing compilers for DQC adopt similar methodologies to single-node quantum compilers. One compiler design proposed by Ferrari et al. [14] exploits Cat-Comm to implement each remote CX gate independently, treating the remote CX like the local CX. Another compiler design by Ferrari et al. [14] and the compiler by Baker et al. [10] use remote SWAP gates to transform remote operations into local operations, resembling SWAP-based routing (e.g., SABRE [15]) for single-node quantum programs. Diadamo et al. [20] consider specific optimizations of inter-node controlled-unitary blocks. However, their work requires specialized circuit representation and cannot optimize general quantum programs. All these works do not consider the burst communication proposed in this paper and thus cannot achieve high communication throughput.

Another line of work executes distributed quantum programs without using inter-node quantum communication protocols [31], [32]. These works run large quantum circuits in a divide-and-conquer way. To overcome the expressibility loss due to no inter-node communication, these works rely heavily on classical post-processing techniques and cannot be extended to large-scale quantum programs.

There are also works trying to reduce the communication overhead of distributed quantum programs by exploring various circuit partition/qubit mapping techniques [8], [33]–[36]. These works are orthogonal to our work and can be easily merged into our framework.

VIII. Conclusion

As in classical distributed computing, the inter-node communication overhead bottlenecks distributed quantum computing. Existing compilers [10], [14], [20] for distributed programs either treat the inter-node communication like the local communication or only provide optimization for gates in the controlled-unitary form. These works fail to utilize the hidden communication patterns in distributed quantum programs. To overcome the shortcomings of existing DQC compilers, this paper explores various distributed quantum programs and identifies burst communication for the first time. Burst communication is a qubit-node communication pattern that widely exists in many distributed quantum programs. Based on burst communication, we propose the framework, AutoComm, which is demonstrated to be efficient in cutting down inter-node communication overhead. The proposed framework can be easily integrated into existing compiling flows of quantum programs and would benefit near-term distributed quantum computing.

Acknowledgment

We thank the anonymous reviewers for their constructive feedback. This work was supported in part by NSF 2048144 and Cisco Research. G. L. was in part funded by NSF QISE-NET fellowship under the award DMR1747426.

References

[1] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[2] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[3] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[4] Kenneth R Brown, Jungsang Kim, and Christopher Monroe. Co-designing a scalable quantum computer with trapped atomic ions. *npj Quantum Information*, 2(1):1–10, 2016.

[5] Colin D Bruzewicz, John Chiaverini, Robert McConnell, and Jeremy M Sage. Trapped-ion quantum computing: Progress and challenges. *Applied Physics Reviews*, 6(2):021314, 2019.

[6] Colin D Bruzewicz, John Chiaverini, Robert McConnell, and Jeremy M Sage. Trapped-ion quantum computing: Progress and challenges. *Applied Physics Reviews*, 6(2):021314, 2019.

[7] Markus Brink, Jerry M Chow, Jared Hertzberg, Easwar Magesan, and Sami Rosenblatt. Device challenges for near term superconducting quantum processors: frequency collisions. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 6–1. IEEE, 2018.

[8] Pablo Andr'es-Mart'inez and Chris Heunen. Automated distribution of quantum circuits via hypergraph partitioning. *Physical Review A*, 2019.

[9] Nicholas Laracuente, Kaitlin N. Smith, Poolad Imany, Kevin L. Silverman, and Fred Chong. Short-range microwave networks to scale superconducting quantum computation. *ArXiv*, abs/2201.08825, 2022.

[10] Jonathan M. Baker, Casey Duckering, Alexander Hoover, and Frederic T. Chong. Time-sliced quantum circuit partitioning for modular architectures. *Proceedings of the 17th ACM International Conference on Computing Frontiers*, 2020.

[11] Christopher Young, Akbar Safari, Preston Huft, J. Zhang, Eun Oh, Ravikumar Chinnarasu, and Mark Saffman. An architecture for quantum networking of neutral atom processors. 2022.

[12] Anocha Yimsiriwattana and Samuel J Lomonaco Jr. Generalized ghz states and distributed quantum computing. *arXiv preprint quant-ph/0402148*, 2004.

[13] Jens Eisert, Kurt Jacobs, Polykarpos Papadopoulos, and Martin B Plenio. Optimal local implementation of nonlocal quantum gates. *Physical Review A*, 62(5):052317, 2000.

[14] Davide Ferrari, Angela Sara Cacciapuoti, Michele Amoretti, and Marcello Caleffi. Compiler design for distributed quantum computing. *IEEE Transactions on Quantum Engineering*, 2:1–20, 2021.

[15] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[16] MD SAJID ANIS, Héctor Abraham, AduOffei, Rochisha Agarwal, Gabriele Agliardi, Merav Aharoni, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Matthew Amy, Sashwat Anagolum, Eli Arbel, Abraham Asfaw, Anish Athalye, Artur Avkhadiev, Carlos Azaustre, PRATHAMESH BHOLE, Abhik Banerjee, Santanu Banerjee, Will Bang, Aman Bansal, Panagiotis Barkoutsos, Ashish Barnawal, George Barron, George S. Barron, Luciano Bello, Yael Ben-Haim, M. Chandler Bennett, Daniel Bevenius, Dhruv Bhatnagar, Arjun Bhobe, Paolo Bianchini, Lev S. Bishop, Carsten Blank, Sorin Bolos, Soham Bopardikar, Samuel Bosch, Sebastian Brandhofer, Brandon, Sergey Bravyi, Nick Bronn, Bryce-Fuller, David Bucher, Artemiy Burov, Fran Cabrera, Padraic Calpin, Lauren Capelluto, Jorge Carballo, Ginés Carrascal, Adam Carriker, Ivan Carvalho, Adrian Chen, Chun-Fu Chen, Edward Chen, Jielun (Chris) Chen, Richard Chen, Franck Chevallier, Kartik Chinda, Rathish Cholarajan, Jerry M. Chow, Spencer Churchill, CisterMoke, Christian Claus, Christian Clauss, Caleb Clothier, Romilly Cocking, Ryan Cocuzzo, Jordan Connor, Filipe Correa, Abigail J. Cross, Andrew W. Cross, Simon Cross, Juan Cruz-Benito, Chris Culver, Antonio D. Córcoles-Gonzales, Navaneeth D, Sean Dague, Tareq El Dandachi, Animesh N Dangwal, Jonathan Daniel, Marcus Daniels, Matthieu Dartiailh, Abdón Rodríguez Davila, Faisal Debouni, Anton Dekusar, Amol Deshmukh, Mohit Deshpande, Delton Ding, Jun Doi, Eli M. Dow, Eric Drechsler, Eugene Dumitrescu, Karel Dumon, Ivan Duran, Kareem EL-Safty, Eric Eastman, Grant Eberle, Amir Ebrahimi, Pieter Eendebak, Daniel Egger, ElePT, Emilio, Alberto Espiricueta, Mark Everitt, Davide Facoetti, Farida, Paco Martín Fernández, Samuele Ferracin, Davide Ferrari, Axel Hernández Ferrera, Romain Fouilland, Albert Frisch, Andreas Fuhrer, Bryce Fuller, MELVIN GEORGE, Julien Gacon, Borja Godoy Gago, Claudio Gambella, Jay M. Gambetta, Adhisha Gammanpila, Luis Garcia, Tanya Garg, Shelly Garion, James R. Garrison, Tim Gates, Leron Gil, Austin Gilliam, Aditya Giridharan, Juan Gomez-Mosquera, Gonzalo, Salvador de la Puente González, Jesse Gorzinski, Ian Gould, Donny Greenberg, Dmitry Grinko, Wen Guan, Dani Guijo, John A. Gunnels, Harshit Gupta, Naman Gupta, Jakob M. Günther, Mikael Haglund, Isabel Haide, Ikko Hamamura, Omar Costa Hamido, Frank Harkins, Kevin Hartman, Areeq Hasan, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Stefan Hillmich, Hiroshi Horii, Connor Howington, Shaohan Hu, Wei Hu, Junye Huang, Rolf Huisman, Haruki Imai, Takashi Imamichi, Kazuaki Ishizaki, Ishwor, Raban Iten, Toshinari Itoko, Alexander Ivrii, Ali Javadi, Ali Javadi-Abhari, Wahaj Javed, Qian Jianhua, Madhav Jivrajani, Kiran Johns, Scott Johnstun, Jonathan-Shoemaker, JosDenmark, JoshDumo, John Judge, Tal Kachmann, Akshay Kale, Naoki Kanazawa, Jessica Kane, Kang-Bae, Annanay Kapila, Anton Karazeev, Paul Kassebaum, Josh Kelso, Scott Kelso, Vismai Khanderao, Spencer King, Yuri Kobayashi, Kovi11Day, Arseny Kovyrshin, Rajiv Krishnakumar, Vivek Krishnan, Kevin Krsulich, Prasad Kumkar, Gawel Kus, Ryan LaRose, Enrique Lacal, Raphaël Lambert, Haggai Landa, John Lapeyre, Joe Latone, Scott Lawrence, Christina Lee, Gushu Li, Jake Lishman, Dennis Liu, Peng Liu, Abhishek K M, Liam Madden, Yunho Maeng, Saurav Maheshkar, Kahan Majmudar, Aleksei Malyshev, Mohamed El Mandouh, Joshua Manela, Manjula, Jakub Marecek, Manoel Marques, Kunal Marwaha, Dmitri Maslov, Paweł Maszota, Dolph Mathews, Atsushi Matsuo, Farai Mazhandu, Doug McClure, Maureen McElaney, Cameron McGarry, David McKay, Dan McPherson, Srujan Meesala, Dekel Meirom, Corey Mendell, Thomas Metcalfe, Martin Mevissen, Andrew Meyer, Antonio Mezzacapo, Rohit Midha, Daniel Miller, Zlatko Minev, Abby Mitchell, Nikolaj Moll, Alejandro Montanez, Gabriel Monteiro, Michael Duane Mooring, Renier Morales, Niall Moran, David Morcuende, Seif Mostafa, Mario Motta, Romain Moyard, Prakash Murali, Jan Müggenburg, Tristan NEMOZ, David Nadlinger, Ken Nakanishi, Giacomo Nannicini, Paul Nation, Edwin Navarro, Yehuda Naveh, Scott Wyman Neagle, Patrick Neuweiler, Aziz Ngoueya, Johan Nicander, Nick-Singstock, Pradeep Niroula, Hassi Norlen, NuoWenLei, Lee James O'Riordan, Oluwatobi Ogunbayo, Pauline Ollitrault, Tamiya Onodera, Raul Otaolea, Steven Oud, Dan Padilha, Hanhee Paik, Soham Pal, Yuchen Pang, Ashish Panigrahi, Vincent R. Pascuzzi, Simone Perriello, Eric Peterson, Anna Phan, Kuba Pilch, Francesco Piro, Marco Pistoia, Christophe Piveteau, Julia Plewa, Pierre Pocreau, Alejandro Pozas-Kerstjens, Rafał Pracht, Milos Prokop, Viktor Prutyanov, Sumit Puri, Daniel Puzzuoli, Jesús Pérez, Quant02, Quintiii, Rafey Iqbal Rahman, Arun Raja, Roshan Rajeev, Isha Rajput, Nipun Ramagiri, Anirudh Rao, Rudy Raymond, Oliver Reardon-Smith, Rafael Martín-Cuevas Redondo, Max Reuter, Julia Rice, Matt Riedemann, Rietesh, Drew Risinger, Marcello La Rocca, Diego M. Rodríguez, RohithKarur, Ben Rosand, Max Rossmannek, Mingi Ryu, Tharrmashastha SAPV, Nahum Rosa Cruz Sa, Arijit Saha, Abdullah Ash-Saki, Sankalp Sanand, Martin Sandberg, Hirmay Sandesara, Ritvik Sapra, Hayk Sargsyan, Aniruddha Sarkar, Ninad Sathaye, Bruno Schmitt, Chris Schnabel, Zachary Schoenfeld, Travis L. Scholten, Eddie Schoute, Mark Schulterbrandt, Joachim Schwarm, James Seaward, Sergi, Ismael Faro Sertage, Kanav Setia, Freya Shah, Nathan Shammah, Rohan Sharma, Yunong Shi, Jonathan Shoemaker, Adenilton Silva, Andrea Simonetto, Deeksha Singh, Divyanshu Singh, Parmeet Singh, Phattharaporn Singkanipa, Yukio Siraichi, Siri, Jesús Sistos, Iskandar Sitdikov, Seyon Sivarajah, Magnus Berg Sletfjerding, John A. Smolin, Mathias Soeken, Igor Olegovich Sokolov, Igor Sokolov, Vicente P. Soloviev, SooluThomas, Starfish,

Dominik Steenken, Matt Stypulkoski, Adrien Suau, Shaojun Sun, Kevin J. Sung, Makoto Suwama, Oskar Słowik, Hitomi Takahashi, Tanvesh Takawale, Ivano Tavernelli, Charles Taylor, Pete Taylour, Soolu Thomas, Kevin Tian, Mathieu Tillet, Maddy Tod, Miroslav Tomasik, Caroline Tornow, Enrique de la Torre, Juan Luis Sánchez Toural, Kenso Trabing, Matthew Treinish, Dimitar Trenev, TrishaPe, Felix Truger, Georgios Tsilimigkounakis, Davindra Tulsi, Wes Turner, Yotam Vaknin, Carmen Recio Valcarce, Francois Varchon, Adish Vartak, Almudena Carrera Vazquez, Prajjwal Vijaywargiya, Victor Villar, Bhargav Vishnu, Desiree Vogt-Lee, Christophe Vuillot, James Weaver, Johannes Weidenfeller, Rafal Wieczorek, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, WinterSoldier, Jack J. Woehr, Stefan Woerner, Ryan Woo, Christopher J. Wood, Ryan Wood, Steve Wood, James Wootton, Matt Wright, Lucy Xing, Jintao YU, Bo Yang, Unchun Yang, Daniyar Yeralin, Ryota Yonekura, David Yonge-Mallo, Ryuhei Yoshida, Richard Young, Jessie Yu, Lebin Yu, Christopher Zachow, Laura Zdanski, Helena Zhang, Iulia Zidaru, and Christa Zoufal. Qiskit: An open-source framework for quantum computing, 2021.

[17] Matthew Amy and Vlad Gheorghiu. staq—a full-stack quantum processing toolkit. *arXiv: Quantum Physics*, 2019.

[18] Nader Khammassi, Imran Ashraf, J. van Someren, Răzvan Nane, A. M. Krol, M. A. Rol, Lingling Lao, Koen Bertels, and Carmen Garcia Almudever. Openql : A portable quantum programming framework for quantum accelerators. *ACM J. Emerg. Technol. Comput. Syst.*, 18:13:1–13:24, 2022.

[19] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. t—ket⟩: a retargetable compiler for nisq devices. *Quantum Science and Technology*, 2020.

[20] Stephen Diadamo, Marco Ghibaudi, and James R. Cruise. Distributed quantum computing and network control for accelerated vqe. *IEEE Transactions on Quantum Engineering*, 2:1–21, 2021.

[21] Maarten Van Steen and A Tanenbaum. Distributed systems principles and paradigms. *Network*, 2:28, 2002.

[22] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: An online resource for reversible functions and reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.

[23] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv: Quantum Physics*, 2014.

[24] Yun Seong Nam, Neil J. Ross, Yuan Su, Andrew M. Childs, and Dmitrii L. Maslov. Automated optimization of large

[26] Roberto Sanchez Correa and Jean Pierre David. Ultra-low latency communication channels for fpga-based hpc cluster. *Integration*, 63:41–55, 2018.

quantum circuits with continuous parameters. *npj Quantum Information*, 4:1–12, 2017.

[25] Nemanja Isailovic, Yatish Patel, Mark Whitney, and John Kubiatowicz. Interconnection networks for scalable quantum computers. In *33rd International Symposium on Computer Architecture (ISCA'06)*, pages 366–377. IEEE, 2006.

[27] Philip Krantz, Morten Kjaergaard, Fei Yan, Terry P Orlando, Simon Gustavsson, and William D Oliver. A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews*, 6(2):021318, 2019.

[28] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[29] Hamza Jnane, Brennan Undseth, Zhenyu Cai, Simon C. Benjamin, and Bálint Koczor. Multicore quantum computing. 2022.

[30] Ying Li and Simon C. Benjamin. Hierarchical surface code for network quantum computing with modules of arbitrary size. *Physical Review A*, 94:042303, 2016.

[31] Wei Tang, Teague Tomesh, Martin Suchara, Jeffrey Larson, and Margaret Martonosi. Cutqc: using small quantum computers for large quantum circuit evaluations. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 473–486, 2021.

[32] Tianyi Peng, Aram Wettroth Harrow, Maris A. Ozols, and Xiaodi Wu. Simulating large quantum circuits on a small quantum computer. *Physical review letters*, 125 15:150504, 2020.

[33] Davood Dadkhah, Mariam Zomorodi, Seyed Ebrahim Hosseini, Pawel Plawiak, and Xujuan Zhou. Reordering and partitioning of distributed quantum circuits. *IEEE Access*, 10:70329–70341, 2022.

[34] Omid Daei, Keivan Navi, and Mariam Zomorodi-Moghadam. Optimized quantum circuit partitioning. *International Journal of Theoretical Physics*, 59(12):3804–3820, 2020.

[35] Mariam Zomorodi Moghadam, Monireh Houshmand, and Mahboobeh Houshmand. Optimizing teleportation cost in distributed quantum circuits. *International Journal of Theoretical Physics*, 57:848–861, 2016.

[36] Zohreh Davarzani, Mariam Zomorodi Moghadam, Mahboobeh Houshmand, and Mostafa Nouri. A dynamic programming approach for distributing quantum circuits by bipartite graphs. *Quantum Inf. Process.*, 19:360, 2020.