UNIVERSITY OF CALIFORNIA SAN DIEGO

Toward Scalable and Efficient Quantum Computing Systems

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Hezi Zhang

Committee in charge:

Professor Yufei Ding, Chair
Professor Daniel Grier
Professor Dean Tullsen
Professor Yi-Zhuang You

2026

The Dissertation of Hezi Zhang is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2026

DEDICATION

To my family, for their unwavering support and encouragement throughout my Ph.D. journey.

This dissertation would not have been possible without their patience and understanding during the many years of graduate study.

I am especially grateful for their constant belief in me through both progress and setbacks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Yufei Ding for her support as the chair of my committee. Through multiple drafts and many long nights, her guidance has proved to be invaluable.

Chapter 1, in full, is based on the paper from Hezi Zhang, Keyi Yin, Anbang Wu, Hassan Shapourian, Alireza Shabani, and Yufei Ding, "MECH: Multi-Entry Communication Highway for Superconducting Quantum Chiplets" published in Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. The dissertation author is the primary investigator and author of this paper.

Chapter 2, in full, is based on the paper from Hezi Zhang, Yiran Xu, Haotian Hu, Keyi Yin, Hassan Shapourian, Jiapeng Zhao, Ramana Rao Kompella, Reza Nejabati, and Yufei Ding, "SwitchQNet: Optimizing Distributed Quantum Computing for Quantum Data Centers with Switch Networks" published in the 49th lEEE/ACM International Symposium on Computer Architecture (ISCA). The dissertation author is the primary investigator and author of this paper.

Chapter 3, in full, is based on the paper from Hezi Zhang, Anbang Wu, Yuke Wang, Gushu Li, Hassan Shapourian, Alireza Shabani, and Yufei Ding, "OneQ: A Compilation Framework for Photonic One-Way Quantum Computation" published in Proceedings of the 50th Annual International Symposium on Computer Architecture. The dissertation author is the primary investigator and author of this paper.

Chapter 4, in full, is based on the paper from Hezi Zhang, Jixuan Ruan, Hassan Shapourian, Ramana Rao Kompella, and Yufei Ding, "OnePerc: A Randomness-aware Compiler for Photonic Quantum Computing" published in Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. The dissertation author is the primary investigator and author of this paper.

Chapter 5, in full, is based on the paper from Hezi Zhang, Jixuan Ruan, Dean Tullsen, Yufei Ding, Ang Li, and Travis Humble, "OneAdapt: Resource-Adaptive Compilation of Measurement-Based Quantum Computing for Photonic Hardware" published in Proceedings of

the 58th IEEE/ACM International Symposium on Microarchitecture. The dissertation author is the primary investigator and author of this paper.

VITA

2016	Bachelor of Science, University of Science and Technology of China
2019	Master of Science, Georgia institute of Technology
2026	Doctor of Philosophy, University of California San Diego

PUBLICATIONS

Hezi Zhang, Jixuan Ruan, Dean Tullsen, Yufei Ding, Ang Li, Travis Humble. *OneAdapt: Resource-Adaptive Compilation of Measurement-Based Quantum Computing for Photonic Hardware*. Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture, 733–748 (2025).

Hezi Zhang, Yiran Xu, Haotian Hu, Keyi Yin, Hassan Shapourian, Jiapeng Zhao, Ramana Rao Kompella, Reza Nejabati, Yufei Ding. *SwitchQNet: Optimizing Distributed Quantum Computing for Quantum Data Centers with Switch Networks*. Proceedings of the 52nd Annual International Symposium on Computer Architecture, 1449–1463 (2025).

Hezi Zhang, Jixuan Ruan, Hassan Shapourian, Ramana Rao Kompella, Yufei Ding. *OnePerc: A Randomness-Aware Compiler for Photonic Quantum Computing*. Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, 738–754 (2024).

Hezi Zhang, Keyi Yin, Anbang Wu, Hassan Shapourian, Alireza Shabani, Yufei Ding. *MECH: Multi-Entry Communication Highway for Superconducting Quantum Chiplets*. Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, 699–714 (2024).

Hezi Zhang, Anbang Wu, Yuke Wang, Gushu Li, Hassan Shapourian, Alireza Shabani, Yufei Ding, *OneQ: A Compilation Framework for Photonic One-Way Quantum Computation*. Proceedings of the 50th Annual International Symposium on Computer Architecture, 1–14 (2023).

ABSTRACT OF THE DISSERTATION

Toward Scalable and Efficient Quantum Computing Systems

by

Hezi Zhang

Doctor of Philosophy in Computer Science

University of California San Diego, 2026

Professor Yufei Ding, Chair

Quantum computing has emerged as a transformative frontier of computation. In recent years, quantum hardware has scaled at an unprecedented rate. As this momentum continues, the central challenge is shifting upward in the stack—from hardware-level feasibility toward system-level scalability. This thesis will focus on quantum computer architecture and compiler, introducing the challenges and opportunities to efficiently harness device capabilities and lower the demands on hardware technology, thereby accelerating timelines for practical quantum advantage.

Introduction

The concept of quantum computing has been around for about 40 years. Recent years have seen rapid progress across various qubit technologies, and now many of these machines are available through the cloud, so we can experiment with hundreds or thousands of qubits remotely, just using our laptops. With this rapid progress, it is envisioned that over the next ten years, we will be able to reach 100,000 or even a million qubits, which would enable us to solve practical problems.

In the 1980s, theorists proposed this fundamentally new way of computing. As it is based on the laws of quantum physics rather than classical physics, it opens up the possibility of computing as efficiently as nature does. When they looked for practical applications, what they found was exciting: for problems such as simulating physical systems and integer factorization, quantum computing can provide exponential speedups over classical computing. With this motivation, researchers started demonstrating the experimental feasibility of quantum computing by building quantum hardware with more and more quantum bits, or qubits. Today we have various qubit technologies, such as superconducting, neutral atom, trapped ion, and photonic platforms. These technologies are also being actively pursued by industry, from tech giants like Google and IBM to a growing number of startups.

But having qubits alone doesn't give us a full-stack computer. It is essential to move from small prototypes to scalable and efficient computing platforms through proper system design. Looking back on how we got from the first digital computers to the machines we have today, it wasn't just about adding more transistors. It requires a full-stack system design to reach the level of performance and scale we take for granted today.

So what are the roles of system design in quantum computing? First, it supports efficient scaling of quantum computing. In the early days we only had around ten qubits, we didn't really need systems or software because everything could be designed manually. But now we have thousands of qubits, and the hardware is also becoming more and more complicated as the scale increases. Software stack — especially the optimizing compiler — is the key to fully harnessing that complexity without losing performance. Second, it lowers the requirements for quantum hardware performance. If we only look at the hardware, it is not always clear how much hardware performance is good enough and how much we could instead rely on software. In fact, among the qubit technologies we mentioned earlier, each comes with its own strengths and limitations. Cross-stack co-design is important for mitigating these limitations and isolating truly critical hardware challenges. Third, it guides future investment in quantum technologies. Today, quantum computing is still in an early stage. There are many directions we could take and many technologies we could invest in. But in reality we always have limited budget, and it is important to know where to put our money and effort. As hardware development is usually much more expensive than software, cost-effective future hardware development must be guided by its impact on the overall system performance.

Today, the quantum computing stack is still far from mature. But if we make an analogy to the classical computing stack, we can organize the layers between devices and applications into quantum architecture, quantum compilation, and quantum programming. This thesis focuses on the architecture and compiler layers of quantum computing systems. On the compiler layer, it discusses the challenges of scaling up quantum computing through modular (Chapter 1) and distributed (Chapter 2) systems and how they could be addressed by compiler optimization. On the architecture layer, it explores quantum computing platforms with different native operations and how they could be addressed by different quantum computing paradigms, such as measurement-based quantum computing (Chapter 3,4,5).

Chapter 1

MECH: Multi-Entry Communication Highway for Superconducting Quantum Chiplets

1.1 Introduction

The past decade has witnessed exciting breakthroughs of quantum computing technologies on various hardware platforms [16, 154, 226, 236, 80, 241, 14], with the superconducting platform being one of the leading candidates. However, the complexity of building increasingly larger devices [111, 76] and the capacity limitation of individual cryogenic dilution refrigerators [127, 13] bring extreme challenges for eventually realizing a giant superconducting quantum chip. Moreover, the fabrication yields will also decline as the number of qubits on the chip increases, leading to a significant increase in manufacturing costs. To realize large-scale quantum computers, a modular system linking processors together, either with long-range or short-range links [18, 196, 59, 217, 109], is required.

Recently, there has been significant research interest [44, 60, 131, 195] in the modular architecture that connects nearby small quantum processors, known as chiplets, due to advancements in short-range inter-chip connections on superconducting platforms [129, 242, 146, 243, 98]. These chiplets form a multi-chip processor that offers a middle ground between monolithic computing and long-range distributed quantum networks. This approach is expected to enable

1-10 thousand qubits in the near term [93], with physical gate error rates ranging from 1e-4 (e.g. single-qubit gates [175]) to 1e-2 (e.g. cross-chip links [98]). Indeed, it is essential to acknowledge that this computing scale remains a considerable distance from reaching the real-world fault-tolerant quantum computing, as the 1-10 thousand physical qubits could accommodate only several long-lived logical qubits [157] and the gate error rates are still above the fault-tolerant threshold [90]. However, it already has the potential to significantly advance the capabilities of near-term quantum computing and empower a range of applications in the NISQ era.

Despite the significant leap in scale, this emerging chiplet architecture presents a new set of challenges which requires novel techniques in compilation of quantum programs. First, with the increased computing scale comes longer execution time of quantum programs, which places a higher demand on the coherence time of the qubits. Second, the different technologies used for on-chip and cross-chip connections introduce heterogeneity in their characteristics, with the fidelity of cross-chip connections typically lower than that of on-chip connections. Third, in contrast to the possibility of all-to-all connectivity between processors in long-range distributed quantum computing, cross-chip connections in the chiplet architecture are subject to constraints that limit connectivity to neighboring chiplets only, and these cross-chip connections may be sparser than on-chip ones. These challenges cannot be addressed directly by existing compilers which are designed and tailored for either monolithic or (long-range) distributed quantum computing for the following reasons.

On one hand, compilation techniques for monolithic quantum computing is not efficient at the scale of computing on chiplets. Monolithic compilers rely on insertion of SWAP gates to route qubits of each gate toward each other [173], thus enabling the multi-qubit gate execution between distant qubits. However, the routing paths of qubits increase with the scale of computing, and the qubits may route back and forth if they are involved in multiple gates. Hence this approach would result in significant latency that soon becomes intolerable for the chiplet architecture. Moreover, these compilers do not take into account the discrepancy between fidelities of on-chip and cross-chip connections.

On the other hand, compilation techniques for distributed quantum computing cannot be effectively applied to the chiplet architecture either. Distributed compilers focus on minimizing the long-range communications, with the motivation that remote gates between different processors are much more expensive than the local ones within each processor. However, this should not be the only objective for compilation on the chiplet architecture, as the disparity between on-chip connections and cross-chip connections is not as significant as that in the distributed quantum computing. Furthermore, these compilers usually make some assumptions that are not applicable to the chiplet architecture, such as an all-to-all connectivity among different processors or easy access to the dedicated communication qubits from any data qubit.

In this paper, we investigate the optimization of quantum program compilation for the chiplet architecture to reveal the sweet spot between monolithic quantum computing and distributed quantum computing. The insight is that the aforementioned challenges can be overcome by innovative compilation techniques being novel at three levels. First, the computing paradigm of the circuit model can be extended to trade additional ancillary qubits for more concurrency of gate execution, thus reducing the execution time and mitigating the increasing demand for coherence time. Second, the qubit communication can be boosted through an efficient communication mechanism over the ancillary qubits, which allocates ancillary qubits in a resource-efficient and architecture-aware manner to enable easy access to the ancillary resource while taking into account the discrepancy between on-chip and cross-chip links. Third, gates in a program can be scheduled by a compiler to execute with or without the ancillary qubits based on their patterns in the program, with corresponding qubits routed accordingly by the compiler to respect the conncetivity constraints within or among the chiplets.

To this end, we propose a compilation framework that incorporates optimizations at the three levels above. **At the level of computing paradigm**, we incorporate ancillary qubits by adopting a hybrid of gate-based computing and measurement-based computing. In particular, the ancillary qubits operate in a measurement-based manner, taking advantage of the recent advancement of dynamic circuit [66, 3], while the regular data qubits remain in their purely

gate-based manner. This hybrid paradigm enables a tradeoff between additional qubit resources and program concurrency by leveraging the outstanding concurrency of the protocol in [229] (Fig. 1.3). Besides the incorporation of measurements in the protocol itself, we further propose a measurement-based scheme for the entanglement preparation among the ancillary qubits, which facilitates the efficient realization of the protocol.

At the level of communication mechanism, we abstract the ancillary qubits into an adjustable *multi-entry communication highway (MECH)*, providing a mechanism to efficiently utilize the ancillary resources with minimized qubit overhead. In particular, the highway is composed by consecutive paths of ancillary qubits, with the paths spanning across the chiplets to give easy access to all qubits. The density of ancillary qubits along the paths are adaptive to the coupling structures of the chiplets, with sparse patterns allowed to reduce the qubit overhead and dense patterns enforced around critical positions to reduce the communication latency and address the architectural heterogeneity. In this way, the highway forms a communication channel on the software level without the need of hardware modification, enabling concurrent gate execution regardless of the distances among the involved qubits.

At the level of program compilation, we fully exploit the highway channels with a dynamic scheduling of gates and an efficient routing of relevant qubits that maximizes the sharing of highway resources. In particular, the scheduling dynamically determines the periods of communication protocols based on the communication demand to accommodate as many gates as possible in each round, allowing a temporal sharing of highway resource among gates. The routing addresses the hardware connectivity constraints with optimizations from two aspects. First, it routes necessary qubits towards nearby highway in a way that ensures the earliest execution of their gates. Second, it routes those gates through the highway in a way that minimizes routing path lengths by allowing spatial sharing of highway resources among qubits. To summarize, our contributions in this paper are listed as follows:

- We trade ancillary qubit resources for program concurrency by proposing a hybrid com-

puting paradigm of gate-based and measurement-based computing, thus mitigating the enhanced demand for coherence time by larger scale of quantum computing.

- We abstract the ancillary qubits into an adjustable multi-entry communication highway at the software level, and propose a communication mechanism for allocation of highway with small overhead and efficient realization of the communication protocol.
- We perform compilation optimizations through a dynamic scheduling of gates onto the highway and an efficient routing of relevant qubits under the connectivity constraints, allowing for temporal and spatial sharing of highway to fully exploit the resources.
- We demonstrate a compilation framework that significantly outperforms the baseline in both the circuit depth and the number of operations. The evaluation shows a trend of reduced qubit overhead and increased outperformance as the computing scale increases, suggesting the scalibility of our framework.

1.2 Background and Related Work

1.2.1 Quantum Computing Basics

Entangled States

Entangled states can serve as valuable resources for quantum computational tasks. For example, the *GHZ states* can be used to facilitate qubit communication over long distances. To prepare an N-qubit GHZ state, one can start with N qubits in the state $|0\rangle$, and apply a chain of CNOT gates as illustrated in Fig. 1.1(a), with the dark blue dots connected by wavy lines standing for the GHZ state. Another example is the utilization of *cluster states* [176], a state of qubits arranged in a lattice $G = (V, E)$, enables universal computation in the measurement-based quantum computing (MBQC) model. To prepare a cluster state, one can initialize all qubits in G to the $|+\rangle$ state and then apply CZ gates between each pair of neighboring qubits in graph G. A 1-dimensional example is illustrated in Fig. 1.1(b), with the light blue dots connected by straight lines standing for the cluster state.

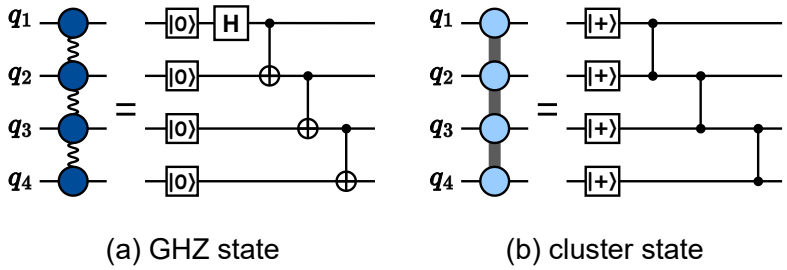


Figure 1.1. GHZ state and cluster state preparation.

Quantum Gates

In the circuit model of quantum computing, qubits are manipulated by quantum gates such as the 1-qubit Hadamard gate and 2-qubit CNOT gate. By arranging these basic gates appropriately, it is possible to abstract higher-level gates for communicating quantum data between qubits. For instance, a SWAP gate can exchange the states of two qubits, which can be achieved by 3 CNOT gates as illustrated in Fig. 1.2(a). A bridge gate can perform an effective CNOT between two qubits that cannot interact with each other directly, through the use of a third qubit. This can be accomplished using 4 CNOT gates as depicted in Fig. 1.2(b). These gates play an important role in addressing the connectivity constraints between qubits on the hardware, and are widely used in various compilation frameworks.

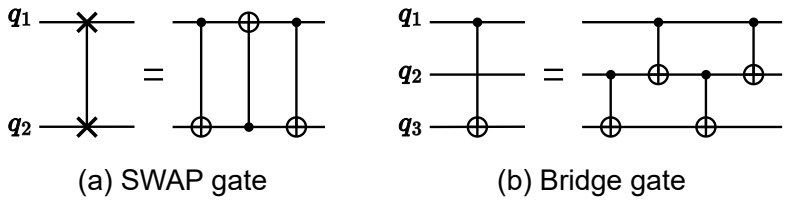


Figure 1.2. SWAP and bridge gates.

Quantum Measurement

Quantum measurement is a fundamental operation that allows us to extract information about the state of a quantum system. While measurements of quantum circuits are traditionally performed in a delay-to-the-end manner, recent advancement of hardware technologies [66, 3] has

enabled dynamic circuits with mid-circuit measurements. This allows a seamless incorporation of real-time classical communication into quantum circuits, and will greatly increase the variety of circuits that can be executed on near-term quantum hardware, forming an important part of many quantum applications in the future.

1.2.2 Quantum Architecture and Compiler

Monolithic Quantum Computing

Monolithic architecture is the simplest architecture for quantum computing that presents a self-contained system on a single chip. However, it faces extreme challenges in its scalability [131], as the increasing number of qubits would increase the noise, reduce the yield rate of manufacturing, and eventually push the capability of cooling and control technologies to their limit.

Compilation for monolithic quantum computing is responsible for transforming quantum circuits to a form that complies with the constraints of the underlying hardware. In particular, it addresses the connectivity constraints among physical qubits by routing logical qubits involved in the same gate to physical qubits that are directly connected on the hardware. On state-of-the-art compilers [173, 9, 119, 193], this is achieved by insertion of SWAP or bridge gates into the circuit. However, as the scale of computing increases, this compilation approach will become inefficient, because the increasingly long routing paths will result in significant latency and harm the overall fidelity.

Distributed Quantum Networks

A distributed architecture for quantum computing, also known as a distributed quantum network, comprises multiple processing nodes physically separated across a considerable distance. While quantum channels across the nodes can be realized by microwave-to-optical (M2O) internode links [103, 128, 130, 134, 61, 152], these links are typically much noisier and slower than the connections within each node [18, 131]. Moreover, internode communications rely on complex protocols [172, 69], usually including the preparation, distribution and measurement of

EPR pairs. Consequently, remote operations between different nodes are considered much more expensive than the local ones.

Compilers for the distributed architecture primarily focuses on minimizing the number of these expensive remote operations. This objective can be achieved by an efficient utilization of entangled states among different nodes. For example, [229] proposes a protocol that enables simultaneous data control on multiple nodes with the help of a pre-established GHZ state. This protocol is depicted in Fig. 1.3, with the dark blue nodes connected by wavy lines standing for the GHZ state. The three entangled qubits q_0, q_4, q_7 in the GHZ state are distributed to three different nodes, with the first node containing qubits $\{q_0, q_1, q_2, q_3\}$, the second node containing $\{q_4, q_5, q_6\}$ and the third one containing $\{q_7, q_8, q_9\}$. By entangling control qubit q_1 with a qubit q_0 in the GHZ state, conducting a measurement on q_0 with Pauli corrections on all the other qubits in the GHZ state, q_1 involves its data into the entangled state, and enables simultaneous control gates over qubits on all the three nodes. However, in distributed compilers [21, 85, 77, 224], certain assumptions are usually made, such as that each node may communicate with any other node in the network, and the data qubits on each node can easily access the dedicated communication qubits on the same node.

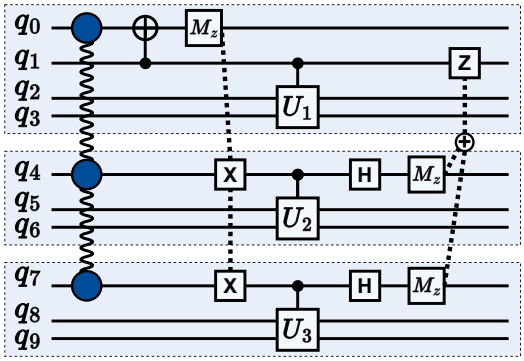


Figure 1.3. Communication protocol.

Chiplet Architecture

The chiplet architecture for quantum computing integrates multiple small, locally-connected chips, known as chiplets, to form a multi-chip module (MCM). This can be achieved by linking separate physical modules to a larger carrier chip or interposer using flip-chip bonds [98], or connecting devices in different refrigerators using cryogenic links [146]. The chiplet architecture offers greater scalability than the monolithic architecture, as the smaller size of the chiplets results in higher yield and lower fabrication costs, and reduces the complexity of design verification and post-fabrication testing. Furthermore, it has better performance than the distributed architecture, as the short-range cross-chip connections can be made with lower latency and higher fidelity compared to long-range internode links.

Recent research on the architecture side has shown the potential of chiplet architecture to exceed the performance of monolithic architecture on near-term hardware [131] and demonstrated the feasibility of scaling quantum computing through modularity by analysis of yield and gate performance [195]. However, there is still much work to be done on the compilation side.

1.2.3 Communication between Qubits

Quantum communication takes place via the movement of qubits or the transfer of their quantum states. [163] introduces a quantum wire architecture based upon quantum teleportation to transport data over long distance, comparing it with the swapping channels by analyzing their latency, bandwidth, fabrication challenges with solid-state silicon technology and associated classical control logic. [114] investigates the construction of reliable long-distance quantum communication channels with ion trap technology by combining ballistic ion movement and quantum teleportation, addressing the routing problem for EPR pair distribution. [144] demonstrates on-chip genuine multipartite entanglement and quantum teleportation in silicon by coherently controlling an integrated network of microresonator nonlinear single-photon sources and linear optic multiqubit entangling circuits. [186] proposes a multi-core architecture that utilizes dedicated communication qubits and specialized physical channels to perform inter-core quantum

communication via quantum teleportation. [112] incorporates the teleportation protocol in qubit mapping. But due to the lack of optimization, their compiler [2] can only handle small-scale circuits, thus not applicable for the chiplet architecture.

Although quantum communication schemes over long distances have been described in the literature above, what has been missing is an efficient management of communication resources that allows different operations to share and utilize the available resources to their full potential. Our work fills this gap by conceptualizing the communication resources to an adjustable multi-entry communication highway (MECH). In this model, different qubits and gates can simultaneously access and share the resources, entering and exiting the highway at various points along it based on their needs. The efficient preparation and consumption of entanglement on the highway, and the precise management of the timing and trajectories of accessing the highway, all handled by our compiler, enhance the overall communication efficiency. Furthermore, as this highway channel is purely software-driven, it can be adjusted conveniently in response to the available qubits and to accommodate architectural heterogeneity.

1.3 Overview: Building Highways on Chiplets

As quantum computing scales up with the help of the emerging chiplet architecture, efficient communication between qubits would become a critical challenge. The increasing routing distances between qubits can lead to significant overhead in executing quantum programs, limiting the scale of programs that could be run within the coherence time. To overcome this challenge, a high-speed communication channel needs to be built on the chiplets to facilitate efficient communication between distant qubits, akin to building highways across cities to enable long-range transportation.

Our insight is that this high-speed communication channel can be built on the software side, by enabling concurrency between gate execution regardless of the distances among the involved qubits. Specifically, the concurrency is achieved by resolving the following two problems. First, gate execution in the superconducting quantum computing system can not reach its

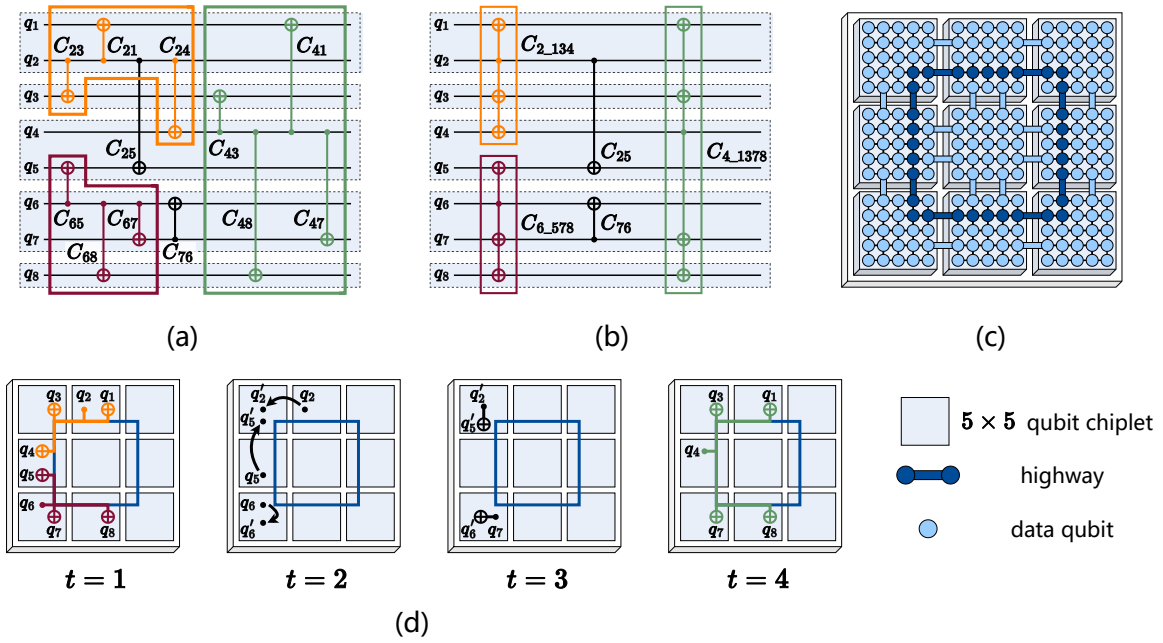


Figure 1.4. Computation process on chiplets in the presence of highway (dark blue paths in (c)). Commutable gates in circuit (a) are aggregated to multi-target control gates (b) and executed simultaneously on the highway (d).

maximum concurrency due to the hardware constraints. While in principle, execution of commutable gates in a circuit do not have dependency on each other, on the superconductive platform they have to be executed sequentially if they share common qubits. Second, communications between distant qubits are usually achieved through insertion of sequential SWAP gates between them. When qubits are involved in multiple gates, they often have to be routed back and forth, preventing the concurrent execution among those gates.

To build the communication channel, we provide an approach that allocates some of the qubits as ancillary qubits and establishes entanglements among them. These entanglements can then be utilized as a computing resource to connect distant qubits and enhance the concurrency of their operations. In particular, we utilize the communication protocol as shown in Fig. 1.3 which allows simultaneous execution of controlled gates sharing the same control qubit. Controlled gates sharing the same target qubit can also utilize this protocol by transforming themselves to controlled gates sharing the same control with Hadamard gates. The entangled states required

by the protocol are formed on the ancillary qubits, which are allocated close to each other on the hardware, forming a computational highway across chiplets. By sacrificing a small portion of qubits, which are not as scarce as in monolithic computing with the highly scalable chiplet architecture, we can trade for more concurrency using this protocol originally proposed for distributed computing.

To illustrate the computation process in the presence of highway, Fig. 1.4 shows an example of executing the circuit in Fig. 1.4(a) on a chiplet architecture in Fig. 1.4(c). This chiplet architecture consists of 3x3 chiplets each containing 5x5 qubits, with the highlighted qubits in dark blue being the ancillary qubits that form a highway across the chiplets. In the circuit (1.4(a)), gates $\{C_{21}, C_{23}, C_{24}\}$, gates $\{C_{65}, C_{67}, C_{68}\}$ and gates $\{C_{41}, C_{43}, C_{47}, C_{48}\}$ are three sets of commutable gates that can be executed concurrently via the utilization of highway. Thus we aggregate them into multi-target controlled gates C_{2_134}, C_{6_578} and C_{4_1378} (shown with corresponding colors in 1.4(b)) while keeping the other gates (i.e., gates C_{25} and C_{76} in black) as their original forms. At $t = 1$, the CNOT components in C_{2_134} and C_{6_578} are executed concurrently on two segments of GHZ states (orange and deep magenta segments in 1.4(d), respectively), while at $t = 4$, the CNOT components in C_{4_1378} are executed concurrently on a new GHZ state prepared between $t = 1$ and $t = 4$ (green segment in 1.4(d)). In contrast, the other gates are executed individually by bringing their qubits together using SWAP gates ($t = 2$ and $t = 3$ in 1.4(d)). We omit the details of scheduling and routing here and will discuss them in later sections.

1.4 Hybrid Computing Paradigm

Chiplet architecture is a highly scalable architecture that can provide massive qubit resources for NISQ devices. This not only leads to a leap of computing scale, but can also revolutionize the quantum computing model by making ancillary qubits more affordable. When combined with the mid-circuit measurement technology, these ancillary qubits can be utilized to increase the concurrency of circuit execution, which is particularly crucial when operating

at the scale enabled by the chiplet architecture. This is because as the scale increases, program execution time naturally increases, thereby posing a considerable challenge to program fidelity due to qubit decoherence. While this can be mitigated by the hardware advancements aimed at extending coherence time, we emphasize the vital role of software optimization, which addresses the problem in a more cost-effective manner.

To take advantage of the ancillary qubits, we extend the circuit model to a hybrid computing paradigm that allows different manners of computation on different types of qubits. Specifically, computation on the ancillary qubits is performed in a measurement-based manner, while computation on the regular qubits, which we call *data qubits*, remains in the purely gate-based manner. This hybrid paradigm allows the ancillary qubits to facilitate gate execution between data qubits through a communication protocol depicted in Fig. 1.3. This protocol enables simultaneous execution of control gates that share the same control qubit by consuming pre-established GHZ states among the ancillary qubits through 1-qubit measurements. Given the prevalence of these controlled gates in quantum programs, the integration of additional ancillary qubits can significantly enhance program concurrency.

An efficient implementation of the protocol in Fig. 1.3 calls for a fast preparation of GHZ states. This can be realized by manipulating the ancillary qubits in a measurement-based manner. Traditionally, a straightforward preparation via chaining the qubits with CNOTs would result in a significant latency which is proportional to the number of entangled ancillary qubits. To mitigate this issue, we propose an efficient preparation strategy that allows a constant-time preparation by leveraging the mid-circuit measurement technology. In particular, this is achieved by three steps, as illustrated in Fig. 1.5. First, we prepare an $n-$qubit cluster state in a highly parallel manner. Second, we perform measurements on half of the entangled qubits, forcing the other half of the qubits to form a $\frac{n}{2}-$GHZ state by applying necessary Pauli corrections. Third, if necessary, the measured ancillary qubits can be re-entangled by applying parallel CNOT gates controlled by the $\frac{n}{2}-$GHZ state, thus forming a larger GHZ state.

The reason behind is that a GHZ state can be extended by the circuit in Fig. 1.6. Specif-

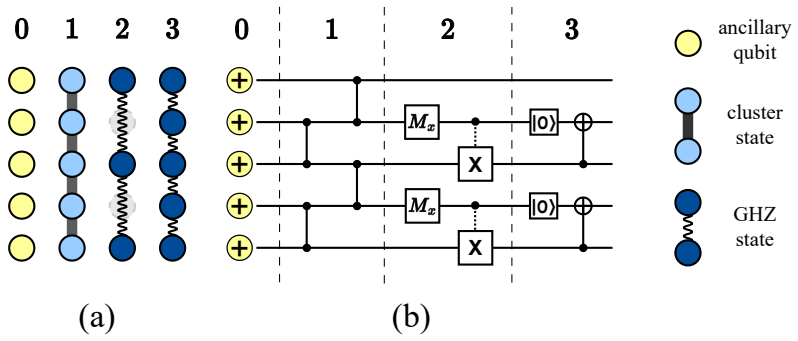


Figure 1.5. Efficient preparation of GHZ states.

ically, the n-qubit GHZ state $|+\rangle_n \equiv |0\rangle^{\otimes n} + |1\rangle^{\otimes n}$ and the 1-qubit $|+\rangle$ state apply two CNOT gates on another $|0\rangle$ state, leading to a state

$$|0\rangle^{\otimes n}|0\rangle|0\rangle + |1\rangle^{\otimes n}|0\rangle|1\rangle + |1\rangle^{\otimes n}|1\rangle|0\rangle + |0\rangle^{\otimes n}|1\rangle|1\rangle .$$

Then measuring the another qubit results in either $|0\rangle^{\otimes n}|0\rangle + |1\rangle^{\otimes n}|1\rangle$ with an outcome 0, or $|0\rangle^{\otimes n}|1\rangle + |1\rangle^{\otimes n}|0\rangle$ with an outcome 1. The former state is an (n+1)-GHZ state as shown in Fig. 1.6(c), and the latter one can also be transformed to an (n+1)-GHZ state with a Pauli correction. Since the circuit in Fig. 1.6(a) is equivalent to that in Fig. 1.6(b). This recursively generates the circuit in Fig. 1.5, which can be regarded as the generation of a cluster state followed by subsequent measurements and Pauli corrections. When substituting the 1-qubit $|+\rangle$ state with a multi-qubit GHZ state, this circuit can also be used to merge two GHZ states.

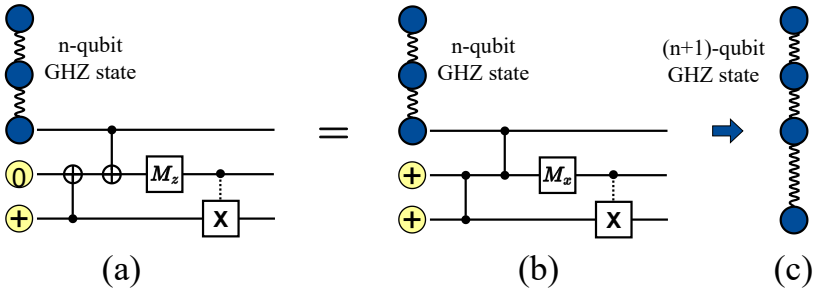


Figure 1.6. Extension of GHZ states.

This method is applicable for generic qubit layouts, not limited to linear cluster states [73].

For example, a cluster state among the qubits lying on a cross-shaped graph can be transformed to a GHZ state through the process shown in Fig. 1.7. This lays a foundation for the formation of highway in subsequent sections.

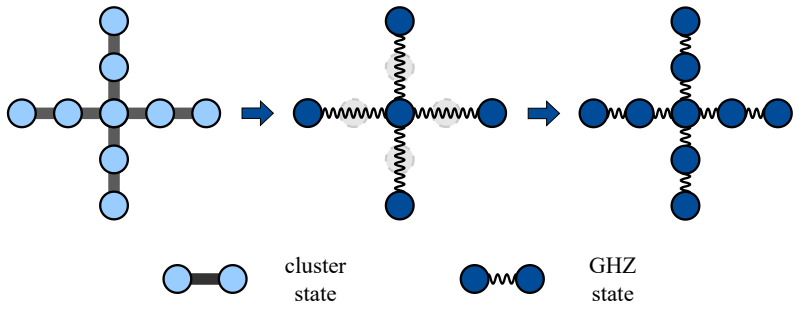


Figure 1.7. Beyond linear cluster state.

1.5 Efficient Highway Mechanism

When it comes to the allocation of ancillary qubits, there are three challenges we need to address. First, the implementation of the fast GHZ state preparation is constrained by the hardware connectivity. The 2-qubit gates in Fig. 1.5 can be conducted only between qubits that are adjacent to each other in the coupling graph of chiplets. Second, the presence of ancillary qubits inevitably reduces the number of qubits that can be used as data qubits since the total number of qubits is limited. Hence we need to minimize this qubit overhead so that it does not significantly affects the scale up of computing on the chiplets. Third, as cross-chip connections have lower fidelity than the on-chip ones, the layout of ancillary qubits should be designed to take this heterogeneity into account and minimize the use of cross-chip connections.

To resolve the first challenge, ancillary qubits should be allocated close to each other on the hardware. For example, a series of adjacent ancillary qubits can be allocated to form consecutive paths as illustrated in Fig. 1.7, so that 2-qubit gates can be directly applied among them. We refer to this layout as a *highway* to indicate its speedup for gate execution, and refer to the ancillary qubits forming the highway as *highway qubits*. The highway should span across chiplets to make sure that it can be accessed easily by all data qubits and facilitate

communications between distant qubits. To maintain the proximity between highway qubits, we allocate the highway qubits prior to the computation and fix their layout throughout the computation.

The second challenge can be tackled by making the highway structure sparser by interleaving highway qubits with data qubits, as shown in Fig. 1.8(a). The efficient entanglement generation among them can be achieved with bridge gates, with the concrete circuit shown in Fig. 1.8(b). The latency of this process is determined by the maximum number of bridge gates each qubit is involved. Consequently, a potential bottleneck arises at the crossroads where different paths intersect. To mitigate this, we maintain a dense arrangement of highway qubits around the crossroads, only employing an interleaving pattern in the non-crossroad areas.

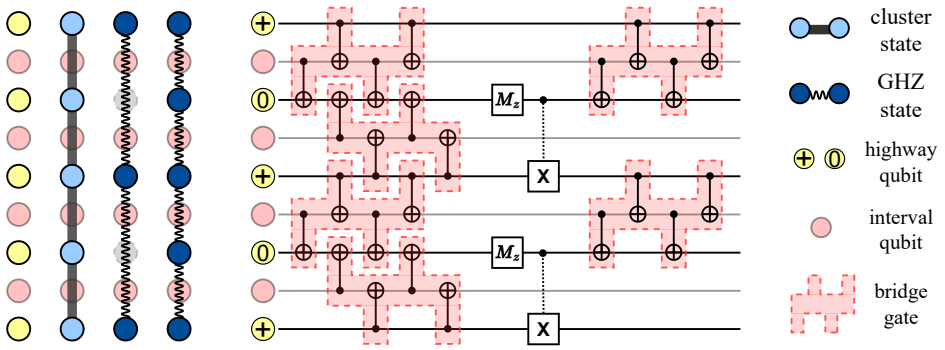
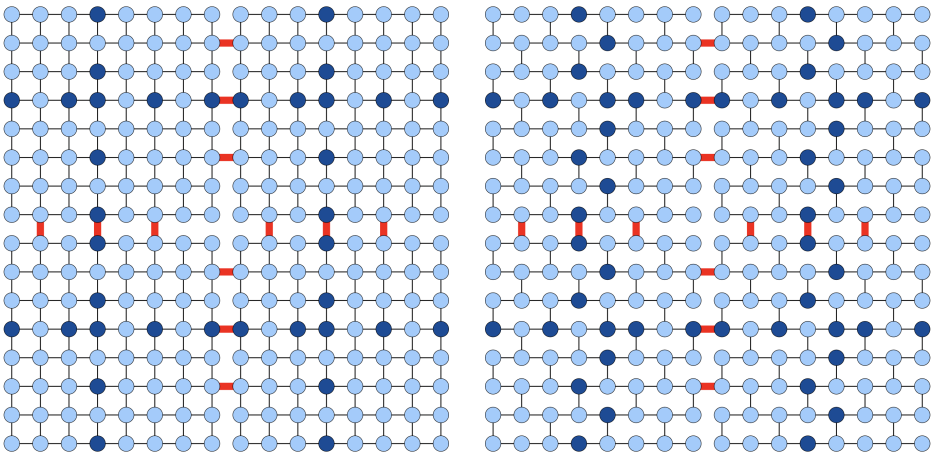


Figure 1.8. Interleaving highway structure (a) with its efficient GHZ state preparation (b).

As for the third challenge, since each cross-chip gate can introduce errors equivalent to several on-chip gates, it is imperative to minimize their usage during the preparation of the highway entanglement. Similar to that around the crossroads, we can alleviate this concern by maintaining dense qubit arrangements on the edges of each chiplet, so that the entanglement among highway qubits on different chiplets can be created by a direct CNOT instead of a bridge gate. This tradeoff between the qubit overhead and highway performance can be adjusted flexibily based on the performance disparity between on-chip and cross-chip links.

In compliance with the conditions set forth by the insights above, we can construct the



(a) (b)

Figure 1.9. Highway configuration on square and hexagon chiplets.

highway in a roughly mesh-like configuration, with the selection of horizontal and vertical paths ensuring that they encounter the cross-chip links at the boundaries. As an example, Fig. 1.9(a)(b) illustrate the highway configurations when the chiplets are in a square and hexagon structure. In the example, the highways on each chiplet have the same configuration, since the periodicity of the mesh aligns with the the width of each chiplet. However, this is not mandatory. In general cases, we can consider connections within and beween all chiplets collectively, allowing the mesh's periodicity to be flexibly determined based on the affordability of ancillary qubtis.

The GHZ state on the highway is prepared by identifying a set of critical highway qubits, i.e., those at crossroads and those at the ends of a path, then entangling them in the way of Fig. 1.8. For each pair of critical qubits, we first count the number of highway qubits between them. If the number is odd, we connect them with the GHZ preparation circuit (Fig. 1.8) with either bridge gates or direct CNOT gates. If the number is even, we turn it into the odd case by first conducting a CNOT (preferred) or bridge gate between one of the critical and the highway qubit next to it. After that, we measure the necessary qubits and do the Pauli corrections. Finally, for the measured qubits, if they serve as a highway entrance (will be explained in the next section), they will be re-entangled by the nearest unmeasured highway qubits which so far has the least number of gates during the GHZ state preparation process.

1.6 Routing and Scheduling Strategies

To optimize the utilization of highway resources, it is crucial to reduce the communication latency with an efficient implementation of the *highway gates*, including an efficient routing and scheduling. By highway gates, we refer to the gates executed via the highway protocol. Since each highway gate consists of multiple 2-qubit controlled gates, we call each of these 2-qubit controlled gates a *gate component* of the highway gate.

1.6.1 Routing

There are two major challenges for routing, one arising from the hardware constraint and the other from the highway mechanism. First, due to the connectivity constraints, the execution of a highway gate requires each of its control and target qubits to be adjacent to a highway qubit. We call that highway qubit a *highway entrance* for that control or target. Second, the efficient preparation of each GHZ state involves CNOTs along a path of qubits, and the paths of GHZ states required by different highway gates should not overlap. We refer to these paths occupied by corresponding highway gates as their *highway paths*. For example, in Figure 1.4, the highway paths of C_{2_134} (orange gate in 1.4(b)) and C_{6_578} (deep magenta gate in 1.4(b)) are disjoint to each other.

The first challenge necessitates an efficient *local routing* that brings qubtis of highway gates towards the highway. Our strategy is that each qubit involved in the highway gates is routed to a nearby highway entrance that enables its earliest execution, which is implemented by inserting SWAP gates along a path that connects the qubit to the highway entrance. To find an optimal highway entrance, each data qubit in the highway gates searches for candidate entrances from its nearby highway qubits. Given the current accumulated depth on the data qubit, its arrival time to each candidate entrance can be obtained by finding the shortest paths to the candidate, denoted as t_{arr}. Compared with the next available time of each candidate, denoted as t_{ava}, we can get the data qubit's earliest execution time on each candidate entrance, which is $t_{exe} = \max\{t_{arr}, t_{ava}\}$. Then the candidate with the earliest t_{exe} is selected as the entrance for the

data qubit. This entrance assignment is performed on data qubits in an ascending order of their shortest distance to the highway to minimize the conflicts between the local routing paths.

The second challenge can be addressed by an efficient *highway routing* that finds the optimal highway path for each highway gate. Our insight is that the length of the highway path for a highway gate can be minimized by maximizing the reuse of highway paths among its gate components. Specifically, each component of the highway gate is assigned a highway path that occupies the least number of additional highway qubtis. This can be achieved by a shortest path search between the component's control entrance and target entrance, with the edge weights along the paths occupied by other components set as 0. In this process, if the highway path of a component has to occupy a highway qubit that is already occupied by components of a different highway gate, then the component is temporarily not executable and has to wait for those occupied highway qubits to be released. This highway path assignment is performed for all components of each highway gate, in descending order of the number of components in the highway gates, resulting in a set of disjoint highway paths occupied by different highway gates.

1.6.2 Scheduling

During the highway communication, GHZ states are periodically prepared and consumed, with each round referred to as a *highway shuttle*. To reduce the latency of highway protocols, we allow a dynamic period for each shuttle based on the traffic demand so that more gate components can share the highway in the same shuttle. Each component of the highway gates is scheduled to the earliest possible highway shuttle given the available highway paths and entrances of each shuttle. The period of each shuttle is determined by waiting on more gates until a new shuttle is needed by gates requiring conflict highway paths. After a new shuttle is scheduled, subsequent highway gates can still be added to this shuttle if they arrive at the highway within the shuttle period, but they can no longer increase the period.

Figure 1.10 illustrates the dynamic period with an example of 5 data qubits contending for 3 highway entrances. Although only 3 of the qubits can be executed at $t = 1$, this shuttle

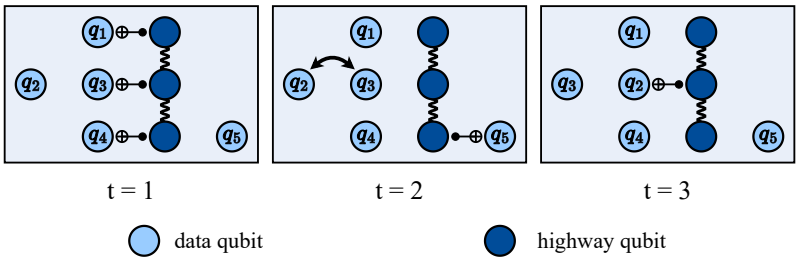


Figure 1.10. Dynamic period of highway communication according to traffic demand.

waits until $t = 3$ so that the other 2 qubits can be executed at $t = 2$ and $t = 3$. Then this shuttle will be ended by destroying the entanglement with measurements on the highway qubits, and the highway qubits become free to prepare for the next shuttle. This dynamic period allows different target qubits of a highway gate to share the same highway entrance at different times, thus reducing the total number of shuttles required.

To maximize the number of gate components in the highway gates, we allow rewriting of the circuit to aggregate the controlled gates sharing the same control qubit. The aggregated multi-target gates among all executable gates are ranked by their numbers of component. Those with the most gate components will be scheduled as highway gates, while the others are scheduled as regular 2-qubit gates off the highway.

1.7 Evaluation

1.7.1 Experiment Setup

Baseline

We implement the baseline by applying state-of-the-art compiler in Qiskit [173] with the highest optimization level 3. The coupling graph passed to the Qiskit compiler contains both the on-chip and cross-chip links. We execute the same circuits on the same chiplet settings (chiplet size, chiplet array size, coupling structure, etc.) for both the baseline and our framework, with the circuit sizes determined by the numbers of data qubits in our framework.

Benchmark Programs

We select quantum Fourier transform (QFT), quantum approximate optimization algorithm (QAOA), variational quantum eigensolver (VQE) and Bernstein Vazirani (BV) algorithm as our benchmarks. For QAOA, we choose the graph maxcut problem on randomly generated graphs. Specifically, the graphs are generated by randomly connecting half of all its possible edges. For VQE, we follow the commonly used full-entanglement ansatz, which proves to be an expressive ansatz [8, 232]. For BV, we select the secret strings randomly, with approximately half of the digits being 0 and half being 1. In table 1.1, we list the number of data qubits, the number of total qubits, the size of each chiplet and the size of the chiplet array for these benchmarks in subsequent experiments including those in sensitivity analysis.

Table 1.1. Architecture Settings.

name-#data qubits	coupling structure	#total qubits	chiplet size	chiplet array
program-261	square	324	6x6	3x3
program-360	square	441	7x7	3x3
program-495	square	576	8x8	3x3
program-630	square	729	9x9	3x3
program-160	square	196	7x7	2x2
program-240	square	294	7x7	2x3
program-480	square	588	7x7	3x4
program-420/366/288	square	486	9x9	2x3
program-312	hexagon	384	8x8	2x3
program-351	heavy square	432	8x8	3x3
program-336	heavy hexagon	480	8x8	3x4

Metric

The first metric we consider is the depth of the compiled circuit, as it represents the latency of program execution and is essential for the mitigation of demand for qubit decoherence time. When couting the circuit depth, we only focus on 2-qubit CNOT gates and measurements, while ignoring the 1-qubit gates and classical operations as they are much faster. We count each measurement operation as multiple depths to manifest its longer latency than gate operations.

The second metric we consider is the number of the most error-prone operations, including on-chip CNOT gates, cross-chip CNOT gates and measurements. The overall error rate is as below when these error rates are small

$$\text{error\_rate} = \ 1 - \prod_i (1-p_i)^{n_i} = \ \sum_i n_i p_i + O(p_i^2)$$

where n_i is the number of the i$^{\text{th}}$ operation, and p_i is its error rate. Hence we address the disparity in the error rates of different operations by defining an *effective number of CNOT gates* as below, abbreviated as '#eff_CNOT',

$$\begin{aligned} \#\text{eff\_CNOTs} = \ & \#\text{on\_chip\_CNOTs} \\ & + \frac{p_{cross}}{p_{on}} \times \#\text{cross\_chip\_CNOTs} \\ & + \frac{p_{meas}}{p_{on}} \times \#\text{measurements} \end{aligned}$$

where p_{on}, p_{cross} and p_{meas} are the error rates of on-chip CNOTs, cross-chip CNOTs and measurements.

Coupling Structure

Our experiments are performed for chiplets of various structures, including square, hexagon, heavy-square and heavy-hexagon. The concrete coupling structures of them are shown in Fig. 1.11, with the black edges indicating on-chip connections and the red edges indicating cross-chip connections.

1.7.2 Experiment Result

In this subsection, we demonstrate the performance and scalability of our framework. Each measurement is counted as a depth of 2 based on the calibration data of IBM [1]. We adopt a rate 7.4 for p_{cross}/p_{on} according to the on-chip fidelity reported recently by IBM [175] and the cross-chip fidelity of flip-chip bonds [98]. We obtain a ratio $p_{meas}/p_{on} = 2.2$ using the same

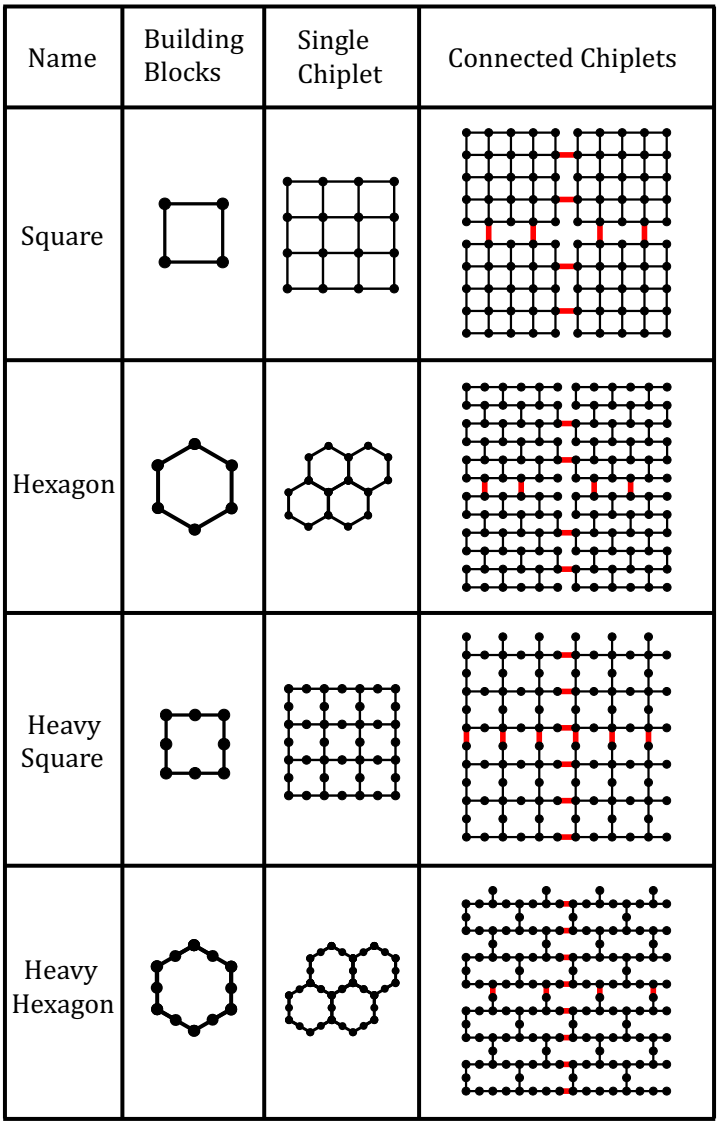


Figure 1.11. Different structures for chiplet architecture. Black lines indicate on-chip connections, red lines indicate cross-chip connections.

on-chip fidelity and the measurement fidelity reported recently [55]. Since these parameters vary across different platforms, we will vary them in the sensitivity analysis later. Moreover, this subsection focuses on the square coupling structure with dense cross-chip links. Different coupling structures and cross-chip sparsity levels will be explored in sensitivity analysis too.

Table 1.2. The results of our framework and its relative performance to the baseline.

Program	Baseline Depth	Framework Depth	Depth Improvement	Baseline eff_CNOTs	Framework eff_CNOTs	eff_CNOTs Improvement	Highway Qubit %
QFT-261	19,282	7,504	61.1%	325,236	216,771	33.3%	19.4%
QAOA-261	14,837	6,586	55.6%	201,637	151,120	25.1%	19.4%
VQE-261	15,725	6,784	56.9%	261,286	180,044	31.1%	19.4%
BV-261	418	31	92.6%	1,179	960	18.6%	19.4%
QFT-360	32,086	11,189	65.1%	582,500	451,553	22.5%	18.4%
QAOA-360	22,757	9,735	57.2%	389,773	300,847	22.8%	18.4%
VQE-360	26,277	10,181	61.3%	471,148	385,647	18.1%	18.4%
BV-360	597	34	94.3%	1,711	1,415	17.3%	18.4%
QFT-495	57,143	18,028	68.5%	1,048,824	827,653	21.1%	14.1%
QAOA-495	43,478	14,175	67.4%	716,324	507,897	29.1%	14.1%
VQE-495	47,193	16,512	65.0%	854,935	690,826	19.2%	14.1%
BV-495	823	37	95.5%	2,297	1,784	22.3%	14.1%
QFT-630	90,535	24,138	73.3%	1,673,337	1,511,568	9.7%	13.6%
QAOA-630	66,342	19,115	71.2%	1,171,597	914,800	21.9%	13.6%
VQE-630	75,178	21,687	71.2%	1,370,750	1,296,846	5.4%	13.6%
BV-630	1,063	40	96.2%	2,772	2,612	5.8%	13.6%

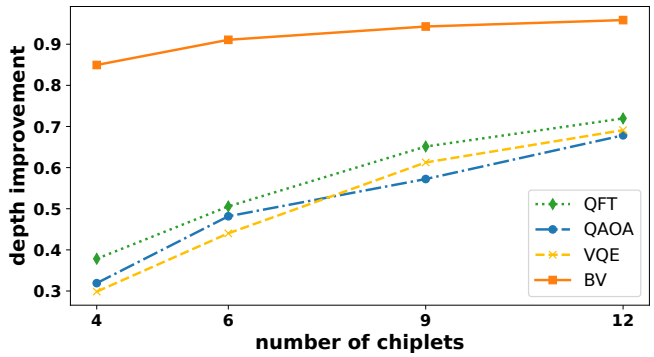
Performance

Table 1.2 shows the results of the baseline and our framework on a 3x3 array of chiplets, with the size of each chiplet varying from 6x6 to 9x9. It can be seen that our framework significantly reduces both the circuit depth and the effective number of CNOT gates compared to the baseline. On average (geomean), the circuit depth is reduced by 70.8%, and the effective number of CNOTs is reduced by 18.1%. Furthermore, as the chiplet size increases, the qubit overhead decreases from 19.4% to 13.6%, as can be obtained from Table 1.1. The outperformance in circuit depth increases with the chiplet size, because the program benefits more from the increased concurrency as the circuit grows larger. The outperformance in the number of effective CNOT decreases with the chiplet size, because the lower percentage of highway qubits result in more demand for local routing of data qubits to access the highway.

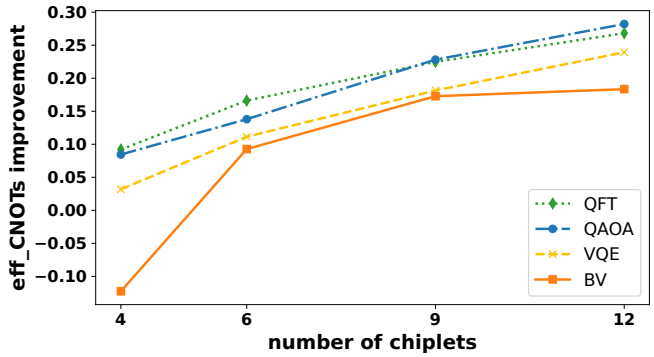
Scalability

Besides the demonstration on varying sizes of chiplets, we demonstrate the scalability of our framework by evaluating on varying numbers of chiplets. Fig. 1.12 shows the improvements in the circuit depth (1.12(a)) and the effective number of CNOTs (1.12(b)) by our framework on

2x2, 2x3, 3x3 and 3x4 chiplet arrays, with the size of each chiplet fixed as 7x7. It can be seen that our framework not only achieves significant improvement in performance, but the improvement also increases as the number of chiplets increases. This trend suggests that the highway model can potentially play an important role in the way of achieving tens of thousands of qubits with the chiplet architecture.



(a)



(b)

Figure 1.12. Improvement in performance for increasing number of chiplets.

1.7.3 Sensitivity Analysis

In this section, we further analyze the effects of the various parameters and configurations, including the latency of measurements, the fidelities of measurements and cross-chip gates, the sparsity of cross-chip links, the highway qubit percentage and the coupling structure of chiplets.

Measurement Latency

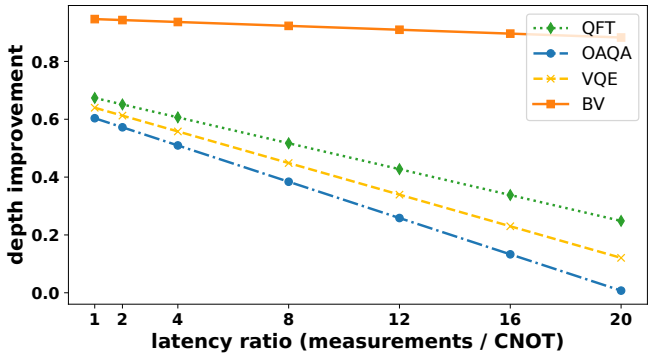
We illustrate the effect of measurement latency with a 3x3 array of 7x7 chiplets. As can be seen from Fig. 1.13(a), the improvement in circuit depth exhibits a linear decrease as the measurement latency increases, remaining positive up to a depth of 20. This implies that the outperformance of our framework in circuit depth is not sensitive to the measurement latency. The linearity is because the measurement latency affects the overall circuit depth only by introducing a constant overhead to each highway protocol, as measurements are conducted highly concurrently.

Operation Fidelity

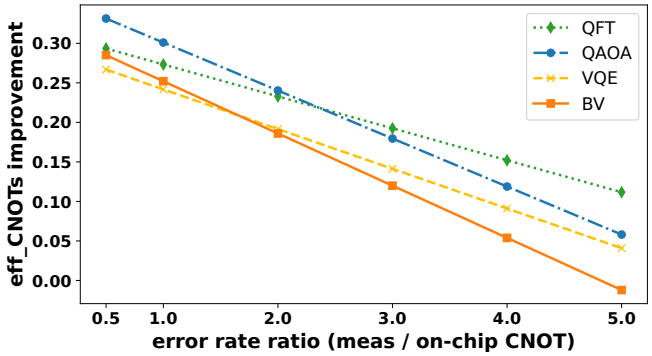
To evaluate the effect of fidelity disparity among different operations, we vary the fidelity ratios of measurements and cross-chip CNOTs to on-chip CNOTs for a 3x3 array of 7x7 chiplets. Fig. 1.13(b)(c) shows the impact of measurement fidelity and the impact of cross-chip CNOT fidelity on the number of effective CNOT gates, respectively. The improvement of effective CNOTs decreases with an increased error rate of measurements, remaining positive up to a ratio of 5, while it increases with an increased error rate of cross-chip CNOTs, being positive for ratios larger than 4. These indicate that our framework can outperform the baseline within a wide range of fidelities of different operations.

Cross-chip Sparsity

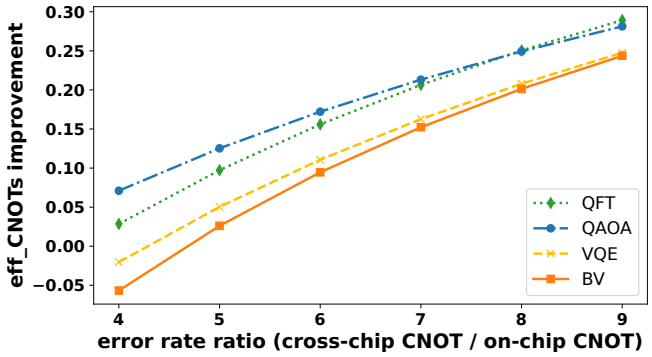
To investigate the impact of sparse cross-chip connections in the chiplet architecture, we evaluate on a 3x3 array of 7x7 chiplets with cross-chip structures at varying sparsity levels. The sparsity was manipulated by keeping 7, 3 and 1 out of the 7 possible cross-chip connections at each edge of a chiplet, denoted as sparsity$= 7/7,\ 3/7$ and $1/7$, respectively. Fig. 1.14 shows the performance of our framework, with the circuit depth (1.14(a)) and the number of effective CNOTs (1.14(b)) normalized by those of the baseline. It can be seen that as the structures become sparser, the normalized circuit depth decreases, indicating an increased improvement in the circuit depth, while the normalized effective number of CNOTs increases, indicating a



(a)



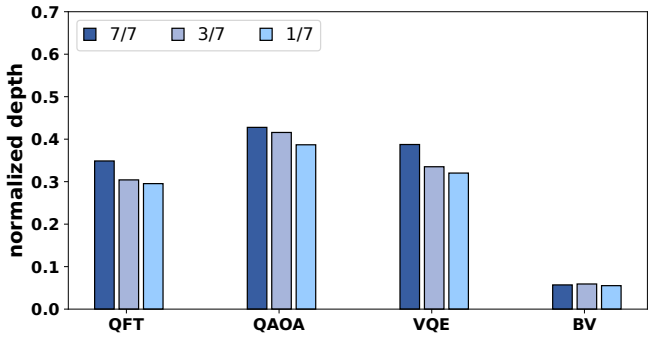
(b)



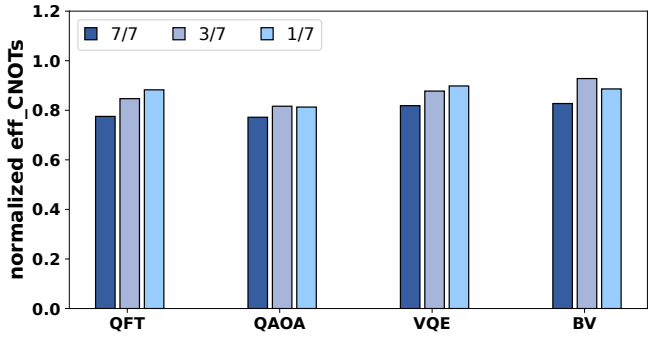
(c)

Figure 1.13. Improvement in performance for varying latencies and fidelities of measurements and cross-chip CNOTs.

decreased improvement in the effective number of CNOTs. These trends primarily stem from the baseline approach, since our framework is relatively stable as the sparsity varies. This is because in our framework, cross-chip communications among qubits are achieved primarily by the utilization of highway, whose formation only requires the connectivity among chiplets, but



(a)



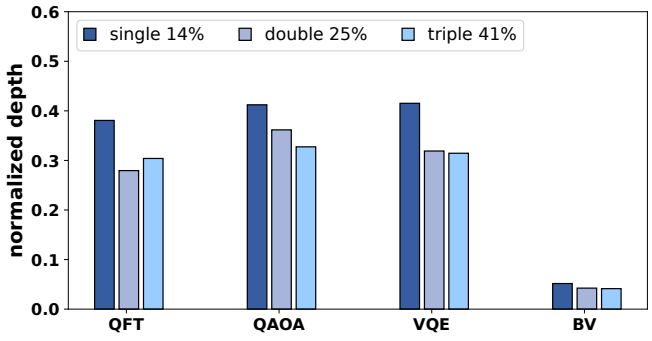
(b)

Figure 1.14. Performance of compiled circuits normalized by that of the baseline approach for chiplets with different sparsity levels of cross-chip connections

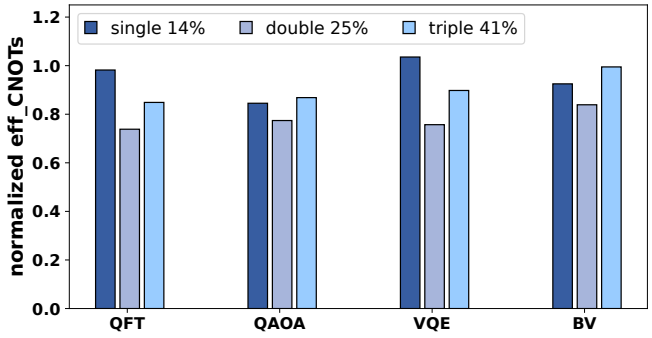
not sensitive to the density of those connections.

Highway Qubit Percentage

We explore the effect of highway qubit percentage with a 2x3 array of 9x9 chiplets when the highway paths are similar to Fig. 1.9 (percentage 14%), doubled (percentage 25%) and tripled (percentage 41%), keeping the circuit sizes of the baseline the same as the number of data qubtis. For comparison, the circuit depth (Fig. 1.15(a)) and the number of effective CNOTs (Fig. 1.15(b)) are normalized by those of the baseline. It can be seen that the normalized depth is reduced when the highway qubit percentage increases from 14% to 25%, but is not affected much as the percentage increases further. The normalized effective CNOT also drops as the highway qubit percentage increases to 25%, but then slightly increases when it further increases to 41%. This is because the increased highway qubits incurs a larger overhead of entanglement



(a)



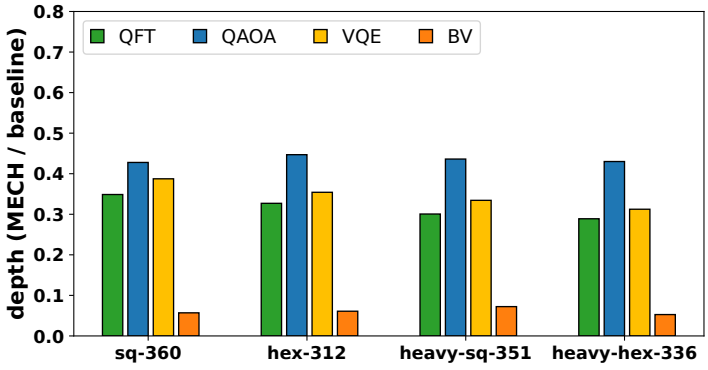
(b)

Figure 1.15. Performance of compiled circuits normalized by that of the baseline approach for highways with different percentages of highway qubits.

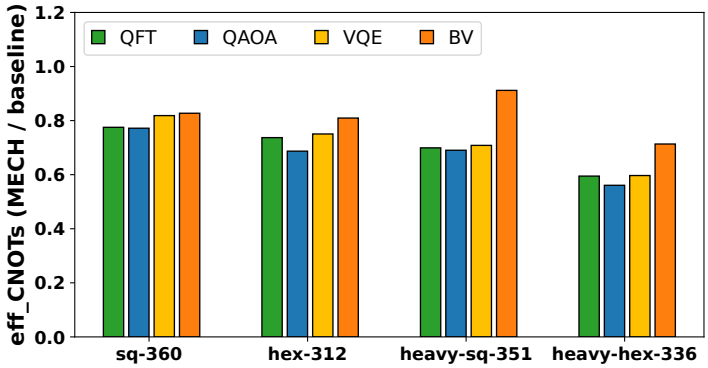
generation.

Coupling Structure

To demonstrate the generality of our framework, Fig. 1.16 shows the performance of our framework on chiplets when the coupling structures are square, hexagon, heavy square and heavy hexagon. For comparison, the circuit depth (1.16(a)) and the number of effective CNOTs (1.16(b)) are normalized by those of the baseline. It can be seen that our framework achieves similar levels of improvements in performance for all of these structures. This implies the appilcability of our framework for general coupling structures.



(a)



(b)

Figure 1.16. Performance of compiled circuits normalized by that of the baseline approach for chiplets with various coupling structures.

1.8 Conclusion

In this work, we provide in-depth analysis and discussion of the compilation challenges of scaling up quantum computing with superconducting chiplet architecture. We propose a hybrid model of gate-based and measurement-based computing to increase program concurrency with ancillary qubits, and propose a compilation framework of multi-entry communication highway to manage the ancillary resources, facilitating the quantum computing at a larger scale. We hope that our work could attract more effort from the computer architecture and compiler community to further explore the chiplet architecture and overcome the challenges in the scaling up.

1.9 Acknowledgments

This chapter is a full reprint of material published as: Hezi Zhang, Keyi Yin, Anbang Wu, Hassan Shapourian, Alireza Shabani, and Yufei Ding, "MECH: Multi-Entry Communication Highway for Superconducting Quantum Chiplets" published in Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. The dissertation author is the primary investigator and author of this paper.

Chapter 2

SwitchQNet: Optimizing Distributed Quantum Computing for Quantum Data Centers with Switch Networks

2.1 Introduction

Distributed quantum computing (DQC) [51, 10, 65, 52, 74] has emerged as a promising approach to address scalability challenges in quantum computing. By enabling quantum communication between multiple quantum processing units (QPUs), DQC extends the capabilities of single-chip systems to multi-node architectures. Among various implementations of DQC, quantum data centers (QDCs) [143, 142, 12, 19, 132, 153] have gained significant attention from industry as a practical near-term solution. Unlike early visions of DQC that focused on long-distance quantum communication across cities, QDCs aim to connect multiple QPUs within a local range of a single room or facility [189, 79, 94, 149, 150]. Their relatively short communication distances and controlled environment mitigate photon loss and various noise, making them well-suited for near-term computational needs.

Recently, an emerging scalable architecture has been proposed [189] for local QDCs, aligning with near-term hardware capabilities. As shown in Fig. 2.1, it connects multiple racks of QPUs via a dynamic optical network with reconfigurable optical switches, which, unlike long-range DQC settings, requires no memory on each switch. Cross-rack switches (core switches),

depicted in orange in Fig. 2.1, can leverage existing classical optical switch technology, while only a small number of specialized quantum switches, depicted in blue, need to be deployed on top of racks (ToR switches) to enable EPR pair generation for quantum communication. This significantly enhances the practicality and cost-efficiency of QDCs, making them a more viable DQC infrastructure in the near future.

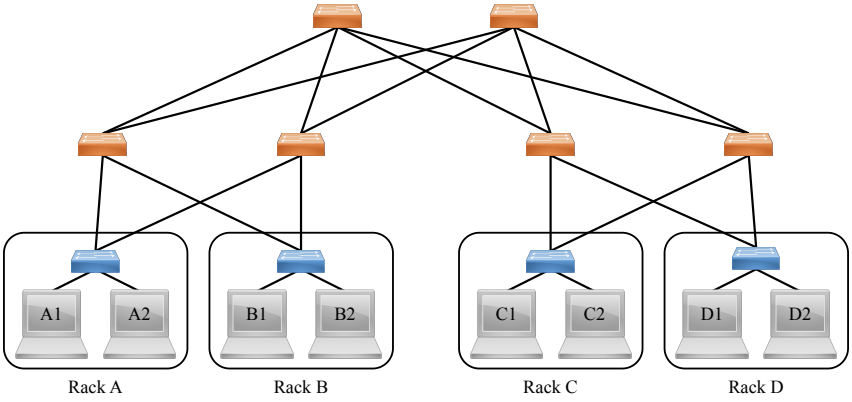
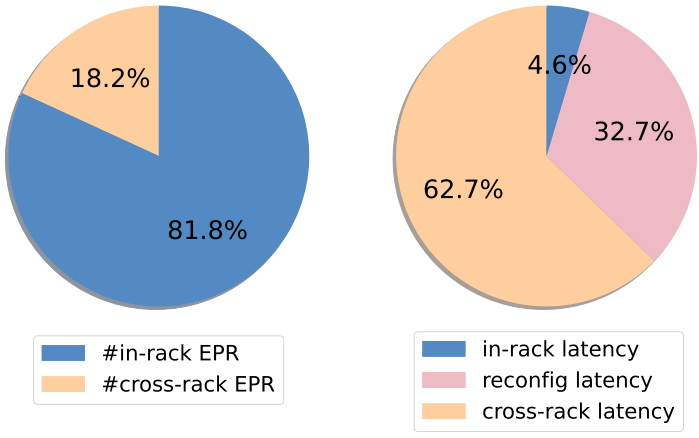


Figure 2.1. QDC with QPUs connected by a switch network.

The primary bottleneck in DQC lies in inter-QPU communication, which is significantly slower and more error-prone than QPU computation. Existing software solutions [78, 86, 104, 225, 223] aim to reduce latency and improve fidelity by minimizing the number of EPR pairs required by communication. However, these approaches show substantial inefficiencies when applied to the QDC architecture due to two key factors. **First**, the distinction between classical core switches and quantum ToR switches requires converters to facilitate their interaction. This added complexity results in a higher latency ($\sim$10 ms) for cross-rack EPR pair generation than in-rack EPR pair generation ($\sim$0.1 ms). **Second**, each switch reconfiguration introduces an additional latency of $\sim$ 1 ms, due to the need to suppress photon loss. Therefore, frequent switch reconfigurations can significantly extend communication latency, especially for in-rack communications.

In Fig. 2.2, we illustrate the communication budget by profiling the quantum workload across different operations. Specifically, we first count the required number of in-rack and



(a) (b)

Figure 2.2. (a) Number percentages of in-rack and cross-rack EPR pairs. (b) Latency percentages of in-rack EPR pair generation, reconfiguration and cross-rack EPR pair generation.

cross-rack EPR pairs, then isolate the contributions of different operations to the overall latency by: (1) setting the latency of both in-rack communication and reconfiguration to 0, attributing all latency to cross-rack communication; (2) setting only the latency of in-rack communication to 0, attributing the latency difference between the two cases to reconfiguration. As shown in Fig. 2.2, while cross-rack EPR pairs constitute only 18.2% of the total required EPR pairs, they account for 62.7% of the overall latency, with reconfigurations contributing an additional 32.7%. This implies that cross-rack communication and frequent switch reconfigurations substantially increase latency, which would also degrade the overall fidelity due to the increased qubit decoherence over time.

To address these challenges in QDCs, a co-optimization across the program layer and the network layer is required in terms of EPR pair scheduling. Existing DQC systems, such as long-range repeater networks [45, 156, 20], decouple the scheduling of EPR pairs and the generation of them into a program layer and a network layer, with the scheduler merely considering the program-layer communication demands and the network layer being responsible for generating the scheduled EPR pairs according to current bandwidth and resource availability. However, given the known computation tasks for QDCs, the compiler can schedule EPR pair

generations by considering both the upcoming communication demands of the quantum program and the bandwidth and resource availability on switches and QPUs. In this way, a larger optimization space can be enabled for hiding the latencies of cross-rack communication and switch reconfiguration.

At a high level, this co-optimization is achievable by leveraging two key quantum features. First, in quantum systems, preparation of EPR pairs can be decoupled from actual communications, as they do not carry any program data. Even if the switches in QDCs are memoryless, these EPR pairs can be stored temporarily on QPUs by allocating a portion of computation qubits as buffer [223]. This storage allows for a flexible look-ahead scheduling of EPR pairs, which allows for optimized timing of EPR generation when combining an analysis of program patterns with a management of buffer resources. Second, an EPR pair between source s and destination d can be generated by merging multiple pre-existing EPR pairs along a path between s and d, referred to as *entanglement swapping*. Unlike that in repeater networks, entanglement swapping on QPUs can be performed fast and deterministically by gates and measurements. With an awareness of bandwidth and buffer availability, this can enable a more efficient network exploitation through a flexible strategy that generates EPR pairs by part.

However, leveraging these features in a compiler is highly non-trivial. **First**, they can introduce fidelity overheads in two ways. (1) The storage of EPR pairs in buffer can bring a risk of decoherence, causing the fidelity of EPR pairs to decrease over time. (2) The strategy of generating EPR pairs by part necessitates generation of additional EPR pairs, which can compromise the overall fidelity as EPR generations are more error-prone than gates on QPUs. These fidelity overheads impose stricter constraints on QDCs than classical data centers (DCs), preventing a straightforward application of pre-fetching techniques in classical DCs. **Second**, the flexibilities in scheduling can lead to complications such as deadlocks (i.e., generations of EPR pairs need to wait for each other's consumption) and buffer congestion (i.e., required buffer occupied by data qubits or other EPR pairs). These need to be eliminated in the compilation stage to ensure successful program execution.

To this end, we present an efficient compiler to strategically leverage those quantum features, striking a balance between latency reduction and fidelity overhead, while eliminating the risks of deadlock and buffer congestion. The compiler analyzes the programs by identifying gate dependencies, looking ahead into near-future gates and extracting their requirements for EPR pairs. With an awareness of the network heterogeneity and the availability of bandwidth and buffer resources, these EPR pairs are then scheduled with the following two optimizations.

First, it **parallelizes cross-rack EPR pair generations** by splitting them into parts, with each split resulting in a new cross-rack pair and some additional in-rack pairs (referred to as *post-split EPR pairs*). In this way, even if the source or destination QPU of an EPR pair is busy, a substitute cross-rack EPR pair can be generated between the source and destination racks through a same-rack QPU with the busy QPU. Once the busy QPU becomes available, it can communicate with that same-rack QPU by generating an additional in-rack EPR pair promptly, thereby completing the generation of the original cross-rack EPR pair through an entanglement swapping. **To mitigate the fidelity overhead** brought by additional in-rack EPR pairs, we prepare multiple copies of each post-split in-rack EPR pair and enhance its fidelity by entanglement distillation [28, 116] using the extra copies.

Second, it further **hides the latency of the additional in-rack EPR pairs**, either from split or for distillation, by a collective generation of them, so that they are prevented from becoming a new communication bottleneck. That is, given that the latency of reconfiguration is much longer than that of in-rack EPR pair generation, we can collect in-rack EPR pairs between the same QPUs and generate them together when the channel between the QPUs is available. In this way, we reduce the average latency of in-rack EPR pair generation by avoiding frequent reconfiguration. The collection of in-rack EPR pairs is guided by the look-ahead program analysis of our compiler. This program-aware guidance reduces the wait time of both in-rack and cross-rack EPR pairs before their usage, thereby **minimizing the fidelity overhead** brought by the storage of prepared EPR pairs.

To **prevent potential deadlocks and buffer congestion** caused by the flexible scheduling,

we propose a resource managing approach that is aware of bandwidth and buffer status. It consists of several rules for scheduling and splitting EPR pairs, along with a retry mechanism for EPR scheduling. These rules mitigate the risk of deadlock and buffer congestion by adapting to current bandwidth availability and ensuring enough buffer size for future EPR pairs, leaving only rare cases that are hard to avoid. When those rare cases happen, we resolve them by incorporating an efficient retry mechanism that downgrades the scheduling strategy to a more conservative one which ensures a deadlock-free and congestion-free scheduling.

Our contributions in this paper can be summarized as the following:

- We propose a compiler to overcome communication challenges in QDCs through a flexible scheduling that is aware of both program patterns and network resources, while preventing potential deadlock and buffer congestion brought by the flexibility.
- We improve the communication efficiency by overlapping the latencies of cross-rack communications, at the cost of incurring additional in-rack communications, with the latency of in-rack communications further reduced by a collective generation.
- We minimize the fidelity overhead by incorporating entanglement distillation for the additional in-rack communications and reducing the wait time of prepared EPR pairs through a program-aware guidance.
- Through a comprehensive simulation with practical hardware parameters, we demonstrate an $8.02\times$ reduction in communication latency over previous approaches while maintaining a low fidelity overhead.

2.2 Background and Related Work

2.2.1 Quantum Communication Protocols

Communication between different QPUs can be achieved through EPR pairs (wavy lines in Fig. 2.3) established between the QPUs with different protocols. The Cat protocol [230]

(Fig. 2.3(a)) can realize a block of remote control gates sharing the same control qubit without transferring data from a QPU to another. In contrast, the teleportation (TP) protocol (Fig. 2.3(b)) allows any pattern of remote gates by transferring a data qubit from a QPU to another through teleportation (from q_0 to q'_c in Fig. 2.3(b)).

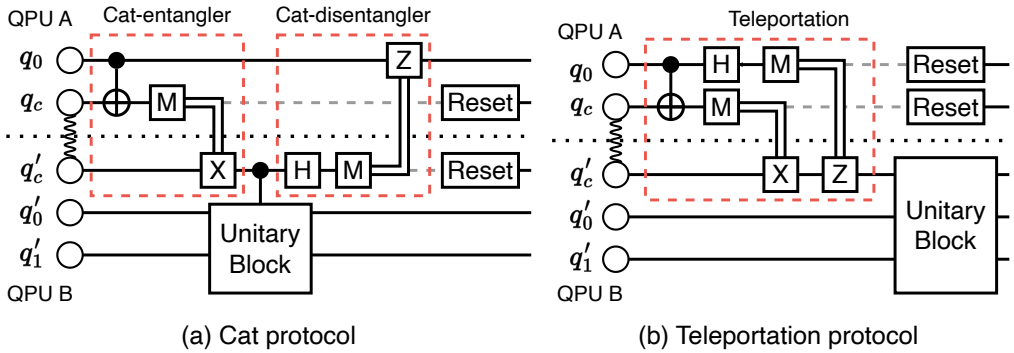


Figure 2.3. (a) Cat protocol and (b) TP protocol for realizing inter-QPU gates with inter-QPU EPR pairs.

2.2.2 Quantum Data Center (QDC)

QDC networks As illustrated by Fig. 2.4, a QDC connects QPUs through a local optical network consisting of optical fibers and reconfigurable optical switches. Edges in the network present possible physical channels connecting nodes (i.e., switches and QPUs) through optical fibers. Channel capacity can be increased via multiplexing, e.g., multiple optical fibers or multiple wavelengths through a single fiber, represented by an edge weight $w > 1$. For instance, Fig. 2.4(b)(c) demonstrate a case of edge weight 2 where each pair of nodes are connected by 2 fibers.

Optical switch The reconfiguration latency of switches ranges from multiple of 100 ns to 100 ms, with the latency increasing with the number of ports [198]. Moreover, a faster switch reconfiguration also leads to an increased photon loss [198]. For switches with about 16×16 ports (or 32 ports), the reconfiguration latency is typically ~ 1 ms with a photon loss rate around 1 dB [200], while the latency of larger switches 1024×1024 ports commercially

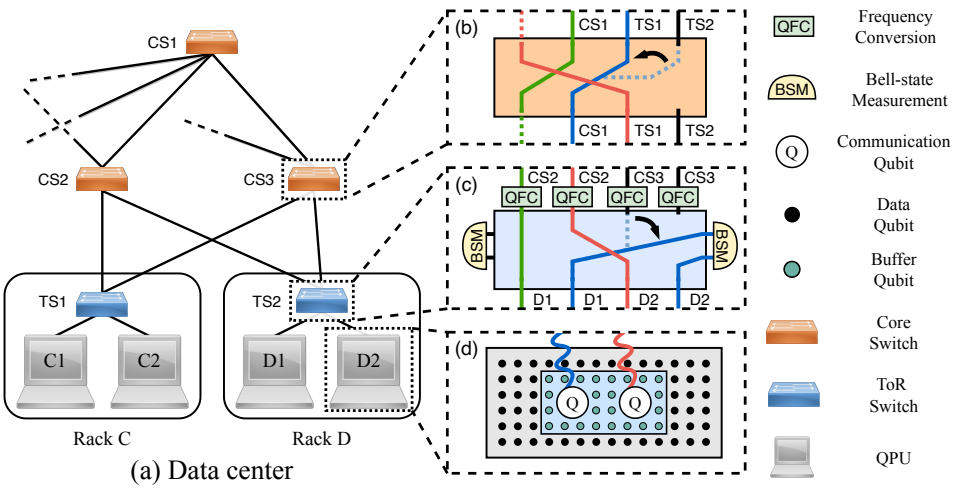


Figure 2.4. (a) A QDC network with (b) classical core switches and (c) quantum ToR switches equipped with a QFC on each outward port and some BSMs. (d) Each QPU has two dedicated communication qubits and many computation qubits, with the computation qubits in the light blue box indicating a buffer initially allocated on the QPU.

available are typically around 100 - 200ms. In this paper, we will adopt the value of 1 ms if not stated otherwise. Each ToR switch can be equipped with Bell-state measurement (BSM) devices to facilitate EPR pair generation. These BSM devices can be shared among QPUs in the same rack by ToR reconfiguration, as shown in Fig. 2.4(c).

EPR pair generation As depicted in Fig. 2.4(d), the generation of EPR pairs requires dedicated communication qubits on each QPU with spin-photon interface between the stationary communication qubits and flying photonic qubits. Fig. 2.5 illustrates the EPR generation protocol [153, 33], where optical switches are eliminated for simplicity.

For in-rack communication, as shown Fig. 2.5(a), communication qubits are prepared in the superposition state $\sqrt{\alpha}|\uparrow\rangle + \sqrt{1-\alpha}|\downarrow\rangle$ and then driven the spin to emit a photon via spontaneous emission, leading to an entangled spin-photon state. Next, the two photonic qubits are directed towards a beam splitter and are eventually measured by the two single-photon detectors, as depicted in the pink box. We only post-select the single detection events to project the spin-spin state into $|\phi_+\rangle = (|\uparrow\downarrow\rangle + |\uparrow\downarrow\rangle)/\sqrt{2}$. This effectively implements a BSM with $2\alpha(1-\alpha)$ success probability. Fig. 2.5(b) shows the equivalent quantum circuit, where the

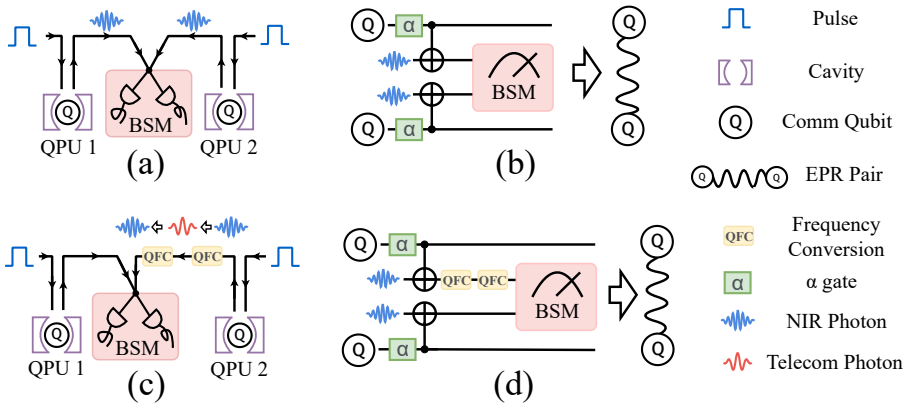


Figure 2.5. In-rack EPR pair generation: physical process (a) and corresponding circuit (b). Cross-rack EPR pair generation: physical process (c) and corresponding circuit (d).

emission process effectively acts as a CNOT gate between the spin and photonic qubit. For cross-rack communication, the protocol is similar but requires quantum frequency converters (QFCs), as shown in Fig. 2.5(c)(d). This is because our ToR switches and BSM devices operate at the near-infrared (NIR) where commercially available optical fiber or switches are not optimized (in terms of the photon loss rate). To extend the communication range to other racks, we convert the NIR photon into telecom regime and back via bidirectional QFCs [215, 183, 123, 30, 29].

EPR rate and fidelity We now estimate the rates and fidelity for in-rack and cross-rack EPR pair generation with some hardware parameters available from the literature. We note that our EPR generation process is probabilistic due to the probabilistic BSM and due to photon loss during transmission from spin to fiber and as they travel through the switches. Hence, the success probability is found to be $p = 2\alpha\eta$ where α is the initial state parameter and $\eta \ll 1$ denotes the overall photon transmission rate (i.e., overall photon loss rate is $1-\eta$). As a result, we consider a repeat-until-success protocol which keeps trying to generate an EPR pair until getting a positive signal in the BSM. Let τ_0 be the operation time for each attempt, then the average time for a successful EPR pair generation is $\tau = \tau_0/p$. Considering hardware parameters [153, 183, 245, 199] $\alpha = 0.05$, $\eta = 0.1$ (i.e., 10 dB loss), and $\tau_0^{-1} \sim 1$ MHz, we obtain $\tau_{\text{ToR}} = 0.1$ ms for in-rack EPR pair generation. For the cross-rack communication, the

EPR generation rate is reduced by a factor of 100 (i.e., 20 dB additional loss) as the transmission rate is further reduced due to the signal attenuation in the second NIR switch, and two QFC devices, leading to $\tau_{\text{inter}} = 10$ ms.

For EPR fidelity, because of false positive signals, we obtain a noisy Bell state in the form of $\hat{\rho} = (1-\alpha)\left|\phi_+\right\rangle\left\langle\phi_+\right| + \alpha\left|\uparrow\uparrow\right\rangle\left\langle\uparrow\uparrow\right|$, and the resulting fidelity is then given by $F = 1-\alpha$. With $\alpha = 0.05$, we obtain $F_{\text{ToR}} = 0.95$ for in-rack EPR pair generation. We further assume that the fidelity is dropped by another 10% due to conversion infidelity in QFCs [123, 245, 199], resulting in a lower fidelity $F_{\text{inter}} = 0.85$ for cross-rack EPR pair generation.

2.2.3 Related Work

DQC compilation Over the last few years, various compilers have been developed for DQC. Some previous work optimizes qubit placement [11, 22, 27, 67, 68, 72, 246] or optimizes communications when routing program qubits among QPUs [78, 86, 104, 225, 223]. These approaches are orthogonal to our compiler and can be combined with our optimizations. Some compilers [170, 203] have also been proposed to implement DQC without inter-QPU communications by cutting large circuits into smaller ones and running them on different devices, yet they require non-scalable classical post-processing. Furthermore, recent work [121] has proposed an efficient implementation of surface code for DQC. Our work can be considered as a higher-level optimization than it, as each of our qubits can be considered as either a physical qubit or a logical qubit.

Long-range quantum repeater networks Long-range quantum communication leverages quantum repeaters to mitigate significant photon loss over extended distances, requiring specialized memory qubits and quantum devices at each repeater for probabilistic entanglement swapping. Previous work have proposed architecture designs to ensure scalability of repeater networks for random workload (e.g., congestion-free on repeater memories, size-independent threshold for error-corrected entanglement, distance-independent communication rates [57, 168, 58, 165]). Moreover, recent work has proposed making quantum repeater networks

compatible with classical networks by designed protocols on the control plane [231]. However, these repeater networks are designed for long-range quantum communication rather than local-range efficient quantum computing. In contrast, QDCs operate within shorter distances, eliminating the need for repeaters, and can leverage off-the-shelf optical switches with minimal quantum devices. This makes them better suited for meeting computational needs with near-term hardware capabilities. For repeater networks, EPR distribution protocols [165, 190, 237, 238, 137] have been designed to maximize EPR throughput by strategically selecting successful links among memory qubits. However, these protocols are tailored for repeater networks and reduce to simple shortest-path searches in memoryless QDCs, thus cannot address the unique challenges of QDCs effectively.

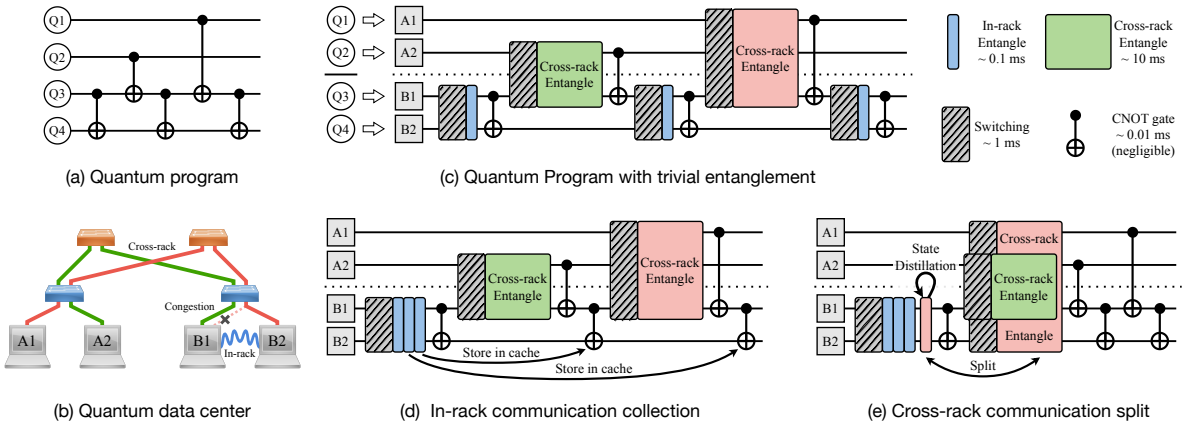


Figure 2.6. Deployment of a quantum program (a) on to a quantum data center (b). The in-rack communication time is reduced from 3.3 ms (c) to 1.3 ms (d) by a collection. The overall communication time is reduced from 23.3 ms (d) to 12.4 ms (e) by splitting a congested cross-rack communication into a non-congested cross-rack communication and an in-rack communication, where the in-rack one can be distilled at a low cost to enhance its fidelity.

Classical DC While classical and quantum data centers (DCs) [5, 206] both exhibit hierarchical network structures with higher cost for cross-rack communication than in-rack communication, their underlying communication models are fundamentally different in terms of data replicability and fidelity sensitivity. These differences impose stricter constraints on QDCs and necessitate more sophisticated compilation techniques.

In classical DCs, data can be freely replicated, stored indefinitely, and retransmitted without fidelity degradation. Consequently, techniques such as pre-transmission, dynamic rerouting, and flexible buffer management are highly effective. In contrast, communication in QDCs rely on EPR pairs that can be stored for only limited time, and consumed only once, rendering classical approaches ineffective. Furthermore, while classical DCs can reduce cross-rack communication by rerouting data through in-rack paths with negligible impact, QDCs require additional in-rack EPR pairs when splitting communication paths. Each additional in-rack EPR generation introduces fidelity overhead, creating a fundamental trade-off between latency reduction and fidelity maintenance.

Our compilation framework addresses these quantum-specific constraints by enabling collective and parallel EPR generation, applying entanglement distillation to mitigate fidelity degradation, ensuring deadlock- and congestion-free scheduling in the compilation stage to guarantee program progress. These techniques have no direct classical analogue and are essential to achieving low-latency, high-fidelity communication in QDCs.

2.3 Motivation

This section introduces the optimizations in our compiler with a motivating example, including collective generation of in-rack EPR pairs and parallelized generation of cross-rack EPR pairs. To facilitate these two optimizations, techniques for resolving deadlock and buffer congestion are also required, which will be introduced with more technical details in the next section.

Fig. 2.6(c) demonstrates an example of scheduling EPR pair generations without a look-ahead analysis of the program pattern. It deploys a circuit in Fig. 2.6(a) to the QDC in Fig. 2.6(b), scheduling EPR pair generations (depicted as rectangles in Fig. 2.6(c)) right before they are needed by inter-QPU CNOT gates, with switch reconfiguration represented by the shaded rectangles in Fig. 2.6(c). Specifically, each line in Fig. 2.6(c) represents a qubit on a different QPU, with the three gates between QPU B_1 and B_2 requiring in-rack EPR pairs and

the other two gates requiring cross-rack EPR pairs according to Fig. 2.6(b). Adopting latencies of in_rack = 0.1 ms, reconfig = 1 ms and cross_rack = 10 ms, neglecting the much shorter gate execution time, the execution of the 5 remote gates requires 25.3 ms in total (Fig. 2.6(c)). As will be seen, this can be reduced to 12.4 ms with the following optimizations (Fig. 2.6(d)(e)).

The first optimization is the collective generation of near-future in-rack EPR pairs, which reduces their average latency by avoiding frequent switch reconfigurations. Through a look-ahead into the program, the compiler finds that three in-rack pairs between QPU B_1 and B_2 are needed in the near future (blue rectangles), which takes 0.3 ms only. However, the switch reconfigurations for enabling the optical channel for three times takes 3 ms (shaded rectangles before the blue ones). To reduce this overhead, it schedules the generations of these three in-rack EPR pairs collectively, as shown by the consecutive blue rectangles in Fig. 2.6(d). These generated EPR pairs are then stored temporarily in the buffer before usage. With this collection, the time for generating the three in-rack EPR pairs is reduced from 3.3 ms to 1.3 ms, with the overall communication time reduced from 25.3 ms to 23.3 ms.

The second optimization is the parallelization of near-future cross-rack EPR pairs, which maximizes the utilization of network bandwidth by splitting congested EPR pairs into non-congested ones. In Fig. 2.6(d), generations of the two cross-rack EPR pair (A_2, B_1) and (A_1, B_1) are sequential, as QPU B_1 can communicate with only one other QPU at a time (assuming edge weight $= 1$). That is, in Fig. 2.6(b), the green path conflicts with the red dashed path. To parallelize them, we can split the congested EPR pair (A_1, B_1) by allowing B_1 to borrow bandwidth from B_2, a QPU in the same rack as B_1. Specifically, the cross-rack EPR pair (A_1, B_1) is split into a new cross-rack EPR pair (A_1, B_2) and an additional in-rack EPR pair (B_1, B_2), followed by an entanglement swapping between these two pairs once they are both prepared. In this way, we can generate the new cross-rack EPR pair (A_1, B_2) in parallel with the originally conflicted (A_2, B_1) and store it in the buffer, while scheduling an additional in-rack EPR pair (B_1, B_2) in the collective generation, as shown in Fig. 2.6(d). This further reduces the overall latency from 23.3 ms (Fig. 2.6(d)) to 12.4 ms (Fig. 2.6(e)).

As the split of cross-rack communication incurs additional in-rack EPR pairs, it poses a risk of reducing the overall fidelity of quantum computing. To mitigate this, we enhance the fidelity of these additional in-rack EPR pairs by scheduling multiple copies of each and implementing entanglement distillation among them, as shown in Fig. 2.6(e). This can be achieved with a low cost as the multiple copies can be scheduled collectively with other in-rack EPR pairs. Given that in-rack EPR pairs have a 95% fidelity, a distillation by two copies results in an EPR pair of $> 96.5\%$ fidelity (where we approximated our input states as Werner states with 93.6% success probability [28, 116]). This fidelity can be further enhanced by using more copies. Note that cross-rack EPR pairs and original (non-split) in-rack EPR pairs can also be distilled upon requests. This can be easily accommodated by our framework, which is equivalent to an increased latency of EPR pair generation.

2.4 Framework Design

2.4.1 Preprocessing

Our compiler can be used in combination with previous compilers [225, 223] that reduce the required EPR pairs. Considering the dynamic reconfigurability of QDCs, these compilers for static network topology should be applied by assuming a full connection between QPUs. In our experiments of Section 2.5, we will combine with the buffer-aware compilation technique in [223]. Its output is a list of required EPR pairs, with corresponding QPUs and communication protocols specified (i.e., Cat vs. TP, see Section 2.2).

To leverage its minimization of EPR pairs, we transform its output to serve as input to our compiler through preprocessing. Specifically, we convert this EPR list to a directed acyclic graph (DAG) by imposing dependencies among them, with each node in DAG representing an EPR pair, and each directed edge representing a dependency. Specifically, whenever the involved QPUs of two EPR pairs overlap, we add an edge from the earlier one in the output list to the later one.

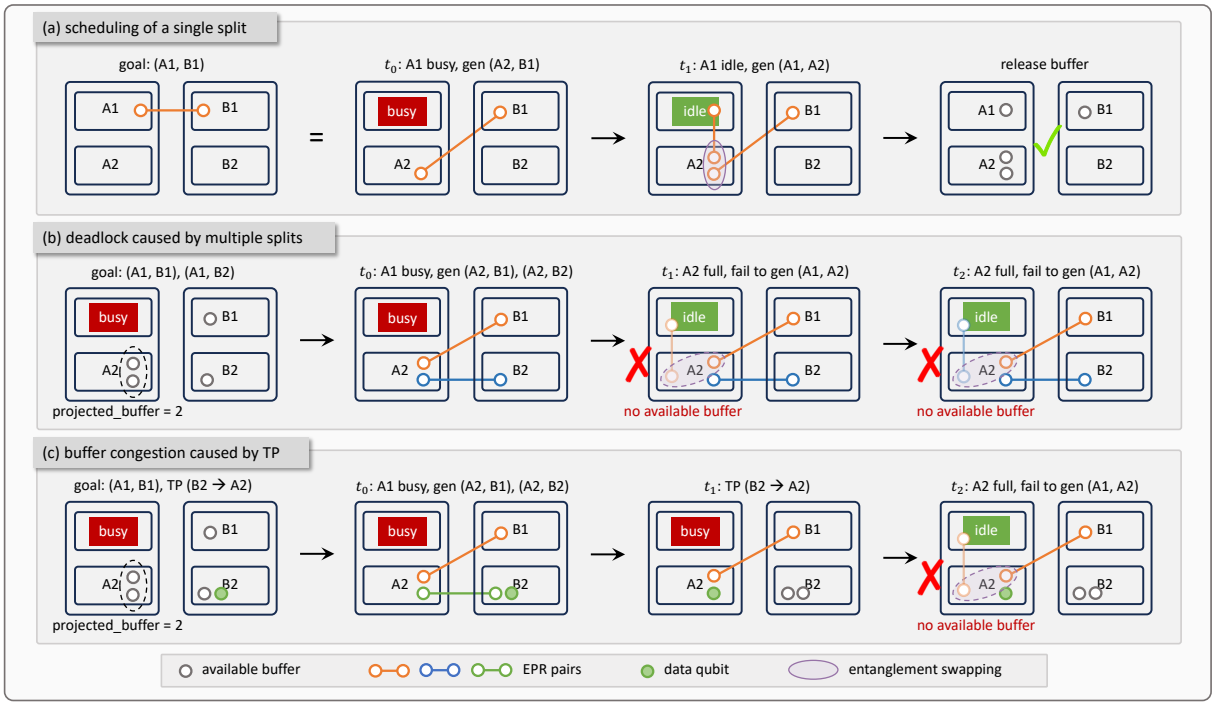


Figure 2.7. (a) A normal EPR split. (b) Deadlock caused by multiple EPR splits. (c) Buffer congestion caused by TP communication.

This DAG needs to be utilized appropriately by our compiler, as the imposed dependency can deviate from the real dependency due to the following reasons. First, even if two EPR pairs are independent according to the DAG, they may not be able to be scheduled concurrently due to bandwidth contention. Second, even if two EPR pairs are dependent according to the DAG, they may be able to be scheduled concurrently, as the overlapped QPUs between them may have multiple communication qubits to generate both pairs. However, these inaccurate dependencies can be used as a rough guidance in our scheduling process, with the above cases handled automatically by our compiler.

2.4.2 EPR Scheduling

This subsection introduces the conditions for EPR pair scheduling, with in-rack collection introduced in the explanation of in-rack vs. cross-rack EPR pairs. Due to the limited communication qubits and network bandwidth, at each time slice it depends on four conditions

to decide whether an EPR between two QPUs can be scheduled. We will first list them as below and then explain.

Scheduling Conditions:

1. **Hard:** available communication qubits on both QPUs
2. **Hard:** available BSMs on the rack of either QPU
3. **Hard:** available optical channel between the QPUs
4. **Soft:** buffer_size + avail_comm $\geq$ threshold $\cdot$ not_in_front_layer
   (threshold $\geq$ #comm_qubits per QPU) on both QPUs

Hard vs. soft conditions The first three conditions are mandatory resource requirements for EPR generation. By checking these conditions for a certain EPR pair, we ensure that this EPR pair can be supplied at the moment by the network. That is, an EPR pair between QPU A and B can be generated only if there are available communication qubits on both A and B, there is an available BSM on the ToR switch of either A's rack or B's rack, and there is an available path in the network between A and B.

The fourth condition is optional, depending on whether the EPR pair is in the front layer of the DAG (i.e., not_in_front_layer = True / False), as explained later. It aims to prioritize the buffer utilization of front-layer EPR pairs to mitigate buffer congestion, which not only improves the overall compilation performance, but also reduces the incurrence of retries that will be introduced in Section 2.4.5.

In-rack vs. cross-rack Due to our collective in-rack EPR pair generation strategy, the third condition can be different when applied to in-rack and cross-rack EPR pairs. For a cross-rack EPR pair between QPU A and B, this condition requires an available optical channel between A and B that is not occupied by any other communication. For an in-rack EPR pair, this condition is checked in two steps. We first check whether this EPR pair can share an occupied channel with other in-rack EPR pairs between A and B. If so, we combine it with those EPR pairs

by scheduling it right after them. If not, we then check if there is an available optical channel that is not occupied by any other communication.

Front layer vs. non-front layer The fourth condition only applies to EPR pairs that are not in the front layer of the DAG. For EPR pairs in the front layer, we allow their generations as long as the three mandatory conditions are satisfied, since they are required by the program urgently. However, for those not in the front layer, which are less urgent, we add this condition to mitigate buffer congestion, requiring the total number of available buffer qubits and available communication qubits on each involved QPU to exceed an adjustable threshold.

2.4.3 Cross-rack EPR Split

This subsection introduces the conditions for EPR pair split. While cross-rack split can help maximize the utilization of network bandwidth, its flexibility can bring a risk of deadlocks or buffer congestion when the involved QPUs do not have enough buffer available. As a result, we impose the following conditions to mitigate this risk, first listing them and then explain. For the rare cases that cannot be prevented by these conditions, they will be resolved by our retry mechanism that will be introduced in Section 2.4.5.

Split Conditions:

- **Hard:** available communication qubits on another QPU in the same rack as the busy QPU
- **Soft:** projected_buffer − reserved_buffer $\geq m_{QPU}$ for each QPU involved in the post-split EPR pairs

Hard Condition The hard condition ensures that there are available communication qubits on another QPU in the same rack as the busy QPU. We illustrate the hard condition by an example in Fig. 2.7(a). When a QPU A_1 is busy, the cross-rack EPR pair (A_1, B_1) can be split into a new cross-rack pair (A_2, B_1) and an additional in-rack pair (A_1, A_2). This requires that QPU A_2 in the same rack as A_1 has an available communication qubit. This split is followed by

an entanglement swapping that merges the two pairs into the required EPR pair (A_1, B_1) and a buffer release after (A_1, B_1) is consumed by communication.

Soft Condition The soft condition aims to ensure that there is enough buffer on the involved QPUs to generate all the post-split EPR pairs. In the soft condition, the projected_buffer of a QPU is defined as the buffer size it would have if all currently scheduled EPR pairs were consumed by the execution of corresponding communications. This reflects the maximum buffer size available in the near term. When calculating this variable, if an EPR pair is scheduled for a Cat protocol, the buffer size of each involved QPU should increase by 1 after the communication. In contrast, if the EPR pair is scheduled for a TP protocol with data teleported from QPU A to QPU B, then the buffer size of A should increase by 2 after the communication, while the buffer size of B should remain unchanged, as the released buffer qubit on B would be occupied by the teleported data qubit.

With this variable, a basic condition for EPR split is that the projected_buffer on each QPU should at least be able to accommodate all EPR pairs induced by an individual split, i.e., projected_buffer $\geq m_{QPU}$, with m_{QPU} being the number of post-split EPR pairs each involved QPU needs to store. For example, if an EPR pair (A, B) is split into (A, A') and (A', B), then there should be $m_A = m_B = 1$ and $m_{A'} = 2$, as each of A and B needs to store only one EPR pair, while A' needs to store two EPR pairs.

However, this basic condition is not sufficient to prevent deadlock caused by multiple EPR splits. We illustrate this deadlock risk with an example in Fig. 2.7(b). The two orange EPR pairs (A_1, A_2) and (A_2, B_1) are the post-split pairs of an EPR pair (A_1, B_1), while the two blue EPR pairs (A_1, A_2) and (A_2, B_2) are the post-split pairs of an EPR pair (A_1, B_2). Supposing QPU A_2 has projected_buffer $= 2$, while the two post-split cross-rack pairs are scheduled to be generated simultaneously, then the post-split in-rack EPR pairs would never have a chance to be scheduled due to a deadlock. This is because the orange in-rack pair will be waiting for buffer release from the blue cross-rack pair, which requires the blue in-rack pair to be generated first; while the blue in-rack pair will be waiting for buffer release from the orange cross-rack pair,

which in turn requires the orange in-rack pair to be generated first.

To accommodate simultaneous splits of multiple EPR pairs, we need to prevent such deadlock by reserving enough buffer size on QPUs. Specifically, for each EPR split, we reserve m_{QPU} buffer qubits from projected_buffer until the post-split EPR pairs are all scheduled, with m_{QPU} being the number of post-split EPR pairs on each involved QPU as explained earlier. This is implemented by tracking a variable reserved_buffer on each QPU, which is initiated as 0. For each split, we increase the reserved_buffer of each QPU by m_{QPU} when the first post-split EPR pair of this split is scheduled, then decrease them by m_{QPU} once all post-split pairs of this split are scheduled. With this reservation, the condition for an EPR split becomes projected_buffer $-$ reserved_buffer $\geq m_{QPU}$.

2.4.4 Post-split distillation

Since the split of cross-rack communications requires additional in-rack EPR pairs, it reduces the overall fidelity since the fidelity of EPR pairs is much lower than the local gates on each QPU. To improve the overall fidelity, we incorporate an entanglement distillation for the post-split in-rack EPR pairs. Given the 95% fidelity of in-rack EPR pairs, a distillation of two EPR pairs enhances the fidelity to $> 96.5\%$. However, the additional EPR pair required by the distillation also needs to occupy a buffer qubit on the QPU. As a result, if an EPR pair (A,B) is split into (A,A') and (A',B), then we should increase m_{QPU} to $m_A = 2$, $m_{A'} = 3$ and $m_B = 1$.

When k-pair distillation is considered, a feasible strategy is distilling with the $k-1$ sacrificed pairs sequentially [116, 138], as these k pairs are generated sequentially through an in-rack optical channel. This strategy reduces the total wait time of EPR pairs, preventing the sacrificed EPR pairs from error exposure, and is efficient in buffer utilization, i.e., the reserved m_{QPU} does not need to be further increased as the $k-1$ sacrificed EPR pairs can reuse the same buffer qubit sequentially. Depending on QPU performance, it may be worthy to wait for the generation of all the k pairs and conduct a parallel distillation [95]. In this case, m_{QPU} should be further increased to $m_A = k$, $m_{A'} = k+1$ and $m_B = 1$.

2.4.5 Algorithm

This subsection describes the overall algorithm of our compiler, which provides each QPU and switch with a deadlock-free and congestion-free communication schedule that dictates their operations at each time slice. Specifically, the compiler optimizes the communication time slice by time slice, looking ahead into near-future EPR demand of each time slice, performing in-rack collection and cross-rack split, scheduling EPR pairs through the proposed conditions to prevent deadlocks and buffer congestion. For the rare cases that could not be prevented, it retries the scheduling with a conservative strategy that ensures the elimination of any deadlock or buffer congestion.

Look-ahead scheduling At each time slice t_0, our compiler looks into the subgraph of the first l layers of the DAG, maximizing the number of scheduled EPR pairs greedily. This scheduling consists of two rounds, with the first round for the scheduling of regular EPR pairs and the second round for EPR split and the scheduling of post-split EPR pairs.

In the first round, it tries scheduling each node in the subgraph in the ascending order of their layers. If the EPR pair represented by a node satisfies the scheduling conditions, then we schedule it, removing the node and updating the dependencies between the remaining nodes in the subgraph. Otherwise, we continue with the next node until no EPR pair in the first l layers satisfies the conditions.

Then we conduct a second round of scheduling if network bandwidth remains. Specifically, we try splitting each node in the remaining subgraph in the ascending order of node layer. If an EPR pair satisfies the split conditions, we split the EPR pair, scheduling the post-split cross rack pair and adding the post-split in-rack pairs into the subgraph. Then we continue to the next node in the updated subgraph until all nodes in the subgraph are traversed. After that, we repeat these two rounds for time slice $t_0 + 1$.

Auto retry While the conditions for scheduling and EPR split can significantly mitigate deadlock and buffer congestion, it is still possible that they may occur occasionally. For example,

in Fig. 2.7(c), the green EPR pair is for a TP communication that teleports a qubit from B_2 to A_2. While a qubit on A_2 would be released after EPR pair (A_2, B_2) is consumed by this TP communication, this released qubit would be taken by the teleported data, making the remaining buffer size insufficient for generating (A_1, A_2) and (A_2, B_1), i.e., the post-split EPR pairs of (A_1, B_2).

To completely eliminate deadlocks and buffer congestion in the compilation, we implement an automatic retry mechanism that reverts to a saved state and retries scheduling with a gradually more conservative strategy if a deadlock or congestion occurs. The most conservative approach would be a strict on-demand EPR pair generation, which follows the exact order of EPR pair generations set by the pre-processing stage, scheduling the generation of each EPR pair right before it is required by an inter-QPU communication. This ensures a deadlock-free and congestion-free scheduling in the worst case.

While ensuring progress, the strict on-demand strategy significantly limits opportunities for parallelizing cross-rack communications. To improve this, we enhance the strategy into a buffer-assisted on-demand strategy that allows parallelization of communications between non-overlapping QPU pairs through buffer utilization. Specifically, this strategy checks the list of EPR pairs provided by the pre-processing in their strict order. If an EPR pair, say between A and B, satisfies the hard scheduling conditions, it then schedules and stores this EPR pair if either its predecessors do not involve A and B or if those predecessors are also successfully scheduled. Note that this strategy does not guarantee deadlock-free and congestion-free communication as the strict on-demand strategy does.

To summarize, whenever a failure occurs, our retry mechanism first reverts to a previously saved state and applies this buffer-assisted on-demand strategy. If the issue persists, it then falls back to the strict on-demand strategy to ensure progress.

2.5 Evaluation

2.5.1 Experiment Setup

Architecture setups In the primary experiment, we evaluate our framework on the CLOS network architecture [5, 206] as shown in Fig. 2.1, with 25% of the total computation qubits reserved as buffer, as listed in Table 2.1. Each QPU is assumed to have two communication qubits. To allow all communication qubits in a rack to work in parallel, we assume each ToR switch to have #BSMs = 2× #QPUs / rack and each switch to be a #BSMs × #BSMs switch (or 2× #BSMs ports), We take a look-ahead depth of 10, adopting 0.1ms, 1ms and 10ms as the latencies of in-rack EPR pair generation, switch configuration and cross-rack EPR pair generation respectively, according to Section 2.2. The settings of experiments are listed in Table 2.1. These parameters will be further varied in subsequent experiments.

Table 2.1. Program and architecture settings

Benchmark	#rack	#QPUs / rack	#Data Qubits / QPU	Buffer Size / QPU	Comm Qubits / QPU
program-480	4	4	30	10	2
program-608	4	4	38	12	2
program-720	4	4	45	15	2
program-360	4	3	30	10	2
program-480	4	4	30	10	2
program-600	4	5	30	10	2
program-720*	4	6	30	10	2
program-240	4	3	20	7	2
program-540	9	3	20	7	2
program-960	16	3	20	7	2
spine-leaf-720	6	4	30	10	2
fat-tree-960	8	4	30	10	2

Benchmark programs We select a set of benchmark programs including building blocks of quantum applications and practical quantum algorithms: multi-control target gate (MCT) [148], Quantum Fourier transform (QFT) [181], Grover's Algorithm (Grover) [56], Ripple-Carry Adder (RCA) [64]. Among them, MCT represents key non-Clifford operations essential for universality and benchmarks the efficiency of multi-qubit gate decomposition. QFT and

RCA [181] are crucial components of Shor's algorithm [191] as well as other applications in signal processing [147, 17] and cryptography [7]. Grover's Algorithm (Grover) [100] is also an important algorithm, which can provide quadratic quantum speed up for unstructured search problems. For Grover's algorithm, we consider the secret string with all ones and repeat the iteration by 100 times. Moreover, we increase the complexity of the RCA circuit by repeating the adder for 100 iterations, which effectively adapts it to a sum calculation.

Metrics We evaluate the compilation performance with four metrics. The first is the overall communication latency of compiled programs, abbreviated as 'latency', normalized by the latency of switch reconfiguration. We ignore the computation time within each QPU as it is much faster than inter-QPU communication.

The second and the third metrics together measure the fidelity overhead. Specifically, the second metric is the EPR overhead, indicating the number of additional distilled EPR pairs, arising from communication split, in a weighted manner. According to the 15% and 5% infidelity of cross-rack and in-rack EPR pairs (see Section 2.2), we count them by weights 1 and 0.33, respectively. For the distilled in-rack EPR pairs, we count each of them by a weight 0.23 based on their $< 3.5\%$ infidelity. The third metric is the average wait time of EPR pairs in buffer, which is also normalized by reconfiguration latency. We list these two overheads separately as the effect of wait time depends on the coherence time of computation qubits, which is not decided by the network but varies with different QPU technology.

The fourth metric is retry overhead, which indicates the compilation time overhead caused by the retry mechanism. Specifically, this is defined as the ratio between the total number of time steps tried in the compilation process and the number of time steps in the compilation result. Hence this value would be 1 if no retry occurs in the compilation.

Baseline Due to the lack of optimized compilers for the emerging hierarchical QDC architecture [189], we construct our baseline by combining state-of-the-art compilation techniques for general DQC [223] with shortest-path buffer-assisted on-demand EPR generation. First, similar to the preprocessing step in Section 2.4.1, we obtain a list of EPR pairs required

Table 2.2. Performance of our compiler and the baseline. Units of latency and wait time are the latency of switch reconfiguration.

Experiment	Benchmark	Baseline: Latency	Ours: Latency	Improv. Factor	#cross-rack EPR (15% infidelity)	#in-rack EPR (5% infidelity)	Ours: #distilled EPR (3.5% infidelity)	EPR Overhead	Baseline: Wait Time	Ours: Wait Time	Additional Wait Time	Retry Overhead
Increase #qubits/QPU	MCT-480	2,312	485	4.77×	15	240	13	3.09%	1.98	6.75	4.77	1.00
	MCT-608	2,312	454	5.09×	15	240	15	3.55%	1.98	6.36	4.39	1.00
	MCT-720	2,312	382	6.05×	15	240	16	3.78%	1.98	8.23	6.25	1.00
	QFT-480	121,728	16,693	7.29×	1,080	2,970	1,133	11.32%	1.30	6.87	5.57	1.00
	QFT-608	155,960	20,781	7.50×	1,368	3,762	1,452	11.44%	1.26	6.92	5.66	1.00
	QFT-720	194,526	24,670	7.89×	1,620	4,455	1,772	11.75%	1.00	7.77	6.77	1.00
	Grover-480	156,213	26,943	5.80×	1,800	7,200	1,927	9.67%	2.08	8.73	6.65	1.00
	Grover-608	150,702	27,412	5.50×	1,800	7,200	1,933	9.70%	1.87	9.54	7.67	1.00
	Grover-720	156,213	25,883	6.04×	1,800	7,200	2,122	10.55%	2.08	9.14	7.06	1.00
	RCA-480	92,259	9,169	10.06×	603	2,412	630	9.46%	0.03	8.49	8.46	1.00
	RCA-608	92,259	9,304	9.92×	603	2,412	650	9.73%	0.03	8.57	8.54	1.00
	RCA-720	92,226	9,395	9.82×	603	2,404	658	9.86%	0.01	9.78	9.77	1.00
Increase #QPUs/rack	MCT-360	1,476	468	3.15×	15	144	15	5.26%	3.03	5.81	2.78	1.00
	MCT-480	2,312	485	4.77×	15	240	13	3.09%	1.98	6.75	4.77	1.00
	MCT-600	3,214	634	5.07×	15	432	12	1.73%	1.17	5.30	4.13	1.00
	MCT-720*	4,413	921	4.79×	15	624	11	1.14%	0.87	5.27	4.40	1.00
	QFT-360	78,300	13,504	5.80×	810	1,500	715	11.30%	1.48	6.27	4.79	1.00
	QFT-480	121,728	16,693	7.29×	1,080	2,970	1,133	11.32%	1.30	6.87	5.57	1.00
	QFT-600	169,831	20,041	8.47×	1,350	4,920	1,559	10.85%	1.14	7.00	5.86	1.00
	QFT-720*	216,372	23,362	9.26×	1,620	7,350	1,915	9.89%	1.20	7.63	6.43	1.00
	Grover-360	140,813	29,717	4.74×	1,800	4,800	1,594	9.86%	2.03	9.96	7.93	1.00
	Grover-480	156,213	26,943	5.80×	1,800	7,200	1,927	9.67%	2.08	8.73	6.65	1.00
	Grover-600	171,613	25,438	6.75×	1,800	9,600	2,057	8.76%	2.08	8.77	6.68	1.00
	Grover-720*	187,013	24,580	7.61×	1,800	12,000	2,178	8.06%	2.07	8.85	6.77	1.00
	RCA-360	83,470	10,127	8.24×	603	1,608	531	9.81%	0.03	9.69	9.66	1.00
	RCA-480	92,259	9,169	10.06×	603	2,412	630	9.46%	0.03	8.49	8.46	1.00
	RCA-600	101,048	8,916	11.33×	603	3,216	683	8.69%	0.03	8.68	8.64	1.00
	RCA-720*	109,837	8,592	12.78×	603	4,020	710	7.86%	0.03	8.78	8.75	1.00
Increase #racks	MCT-240	1,575	490	3.21×	15	144	14	4.93%	2.74	4.66	1.92	1.00
	MCT-540	6,514	3,069	2.12×	99	900	52	2.95%	3.12	5.14	2.03	1.15
	MCT-960	15,369	5,415	2.84×	255	2,304	170	3.73%	3.21	5.44	2.23	1.22
	QFT-240	50,195	9,663	5.19×	540	1,000	389	9.41%	1.61	6.25	4.64	1.00
	QFT-540	212,522	28,694	7.41×	2,640	4,900	1,802	8.96%	2.36	6.28	3.93	1.00
	QFT-960	564,973	58,482	9.66×	8,100	15,400	4,250	6.97%	3.05	6.61	3.56	1.00
	Grover-240	147,468	34,265	4.30×	1,800	4,800	1,248	7.89%	1.00	9.54	8.54	1.01
	Grover-540	365,312	38,648	9.45×	4,800	10,800	3,071	7.86%	0.99	8.63	7.65	1.00
	Grover-960	665,612	39,969	16.65×	9,000	19,200	4,576	6.48%	0.98	7.99	7.01	1.00
	RCA-240	83,470	11,050	7.55×	603	1,608	432	8.13%	0.03	8.91	8.88	1.00
	RCA-540	213,523	12,666	16.86×	1,608	3,618	853	6.61%	0.08	7.10	7.02	1.00
	RCA-960	399,478	13,240	30.17×	3,015	6,432	924	4.01%	0.03	7.03	7.00	1.00
Spine-leaf topology	MCT-720	4,611	1,008	4.57×	39	600	33	3.12%	2.24	6.60	4.36	1.00
	QFT-720	249,317	34,657	7.19×	2,400	6,570	1,767	8.24%	1.77	9.89	8.12	1.00
	Grover-720	242,013	34,357	7.04×	3,000	10,800	2,001	6.61%	2.22	11.63	9.42	1.00
	RCA-720	149,316	11,418	13.08×	1,005	3,618	679	6.69%	0.03	10.36	10.33	1.00
Fat-tree topology	MCT-960	6,910	2,497	2.77×	63	960	30	1.79%	2.44	5.82	3.38	2.34
	QFT-960	421,181	49,568	8.50×	4,200	11,610	3,107	8.24%	1.99	9.40	7.41	1.00
	Grover-960	327,813	39,193	8.36×	4,200	14,400	2,420	5.90%	2.28	12.80	10.52	1.00
	RCA-960	206,373	12,639	16.33×	1,407	4,824	896	6.48%	0.03	10.52	10.49	1.00

by each benchmark program, with the number of those EPR pairs minimized through a buffer-aware compilation pass [223], where the buffer size is set to the same as our framework. Then, we schedule the generation of each EPR pair with the buffer-assisted on-demand strategy as explained in Section 2.4.5. Although in principle, this strategy does not ensure deadlock-free and congestion-free communication as the strict on-demand strategy does, we did not observe

fatal issues with this strategy during experiments.

2.5.2 Primary Experiment Result

Outperformance Table 2.2 presents the comparison of our framework with the baseline as the number of qubits per QPU increases, as the number of QPU per rack increases and as the number of racks increases. Besides CLOS, we also include two other commonly used network topologies, i.e., spine-leaf topology [206] and fat tree topology [57]. It can be seen that our compiler significantly reduces the overall latency, with an average improvement factor of 8.02.

The results of the baseline stay stable with #qubits per QPU, as the increase of #qubits per QPU of primarily affects QPU computation rather than inter-QPU communication. Only the QFT results of the baseline vary with #qubits per QPU, since it has a denser communication pattern than other benchmarks. In contrast, the latency of our compiler varies with #qubits per QPU, as the allocated buffer size increases with #qubits per QPU. This leads to a slightly increased improvement as #qubits per QPU increases.

The improvement of our compiler increases with #QPUs per rack, showing the ability of our compiler to mitigate the bandwidth contention caused by the increased #QPUs. It also increases with #racks, exhibiting the capability of our compiler of effectively utilizing the increased cross-rack bandwidth. These results demonstrate the scalability of our compiler. Furthermore, the improvement factors of the other two network topologies are at a similar level with CLOS network, which demonstrates the general applicability of our compiler to various network topologies.

Overhead It can also be seen from Table 2.2 that these improvements are achieved with a small overhead. First, our compiler requires the generation of 7.41% more (weighted) EPR pairs on average, which decreases as the #QPUs per rack or #racks increases. Second, our compiler increases the wait time of generated EPR pairs in the buffer by only 6.51$\times$ reconfiguration latency on average. This wait time is acceptable since it is even less than the latency of generating one cross-rack EPR pair. Third, the retry overhead is very close to 1 for most programs, indicating

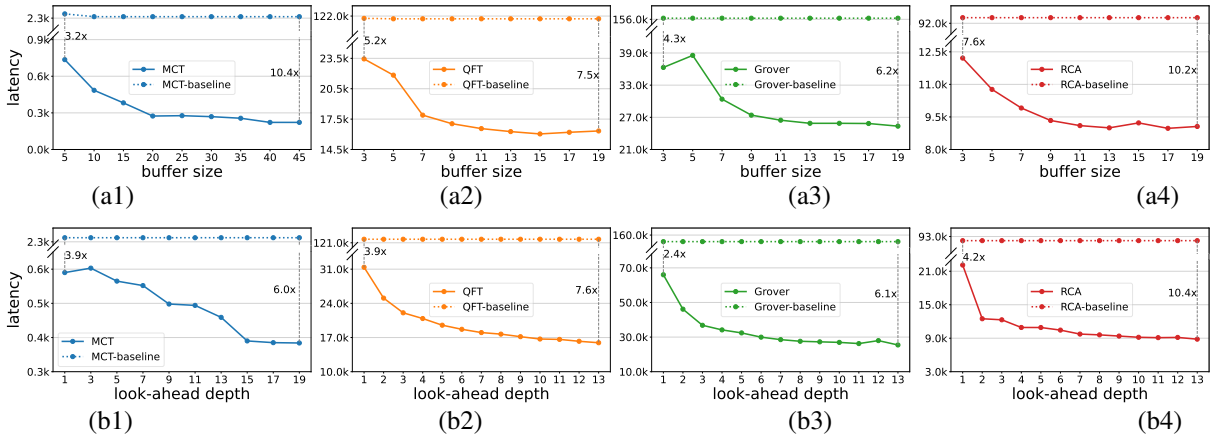


Figure 2.8. Performance improvement of our compiler varying with (a) buffer size and (b) look-ahead depth.

that the retry rarely occurs in the compilation. Only one of the programs (i.e., MCT-960 in fat-tree topology) presents a relative high retry overhead of 2.34. This is because a very early scheduled EPR pair generation caused a buffer congestion for very late EPR pairs, causing the compiler to retry multiple times.

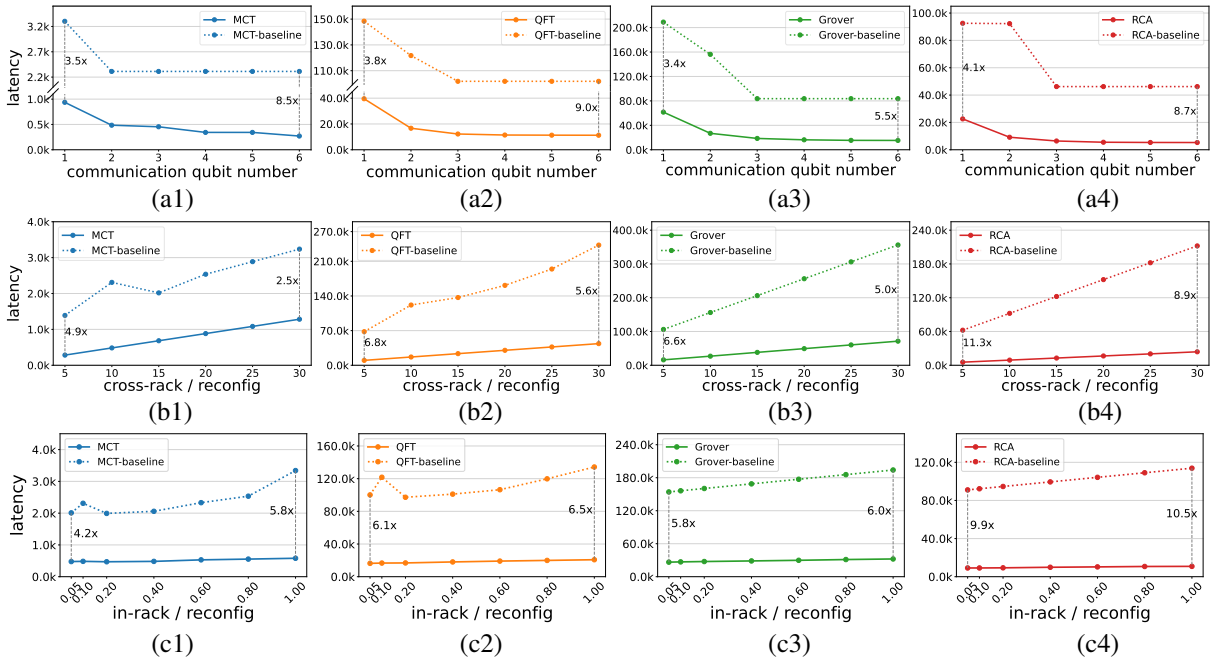


Figure 2.9. Performance improvement of our compiler varying with (a) #communication qubits per QPU, (b) cross-rack EPR latency and (c) in-rack EPR latency (both normalized by reconfiguration latency).

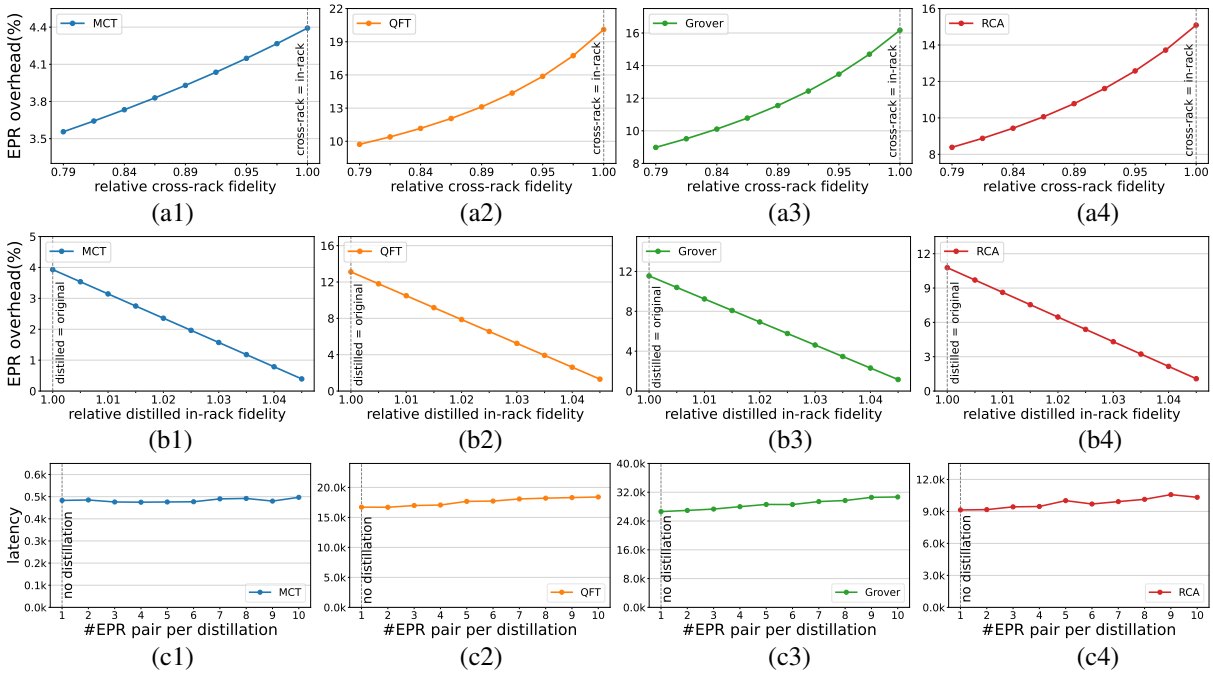


Figure 2.10. Fidelity overhead of our compiler varying with (a) cross-rack fidelity and (b) distilled in-rack fidelity (both relative to the original in-rack fidelity). (c) The overall latency varying with in-rack distillation through different numbers of EPR pairs.

2.5.3 Choice of Hyper-parameters

Buffer size We illustrate the effect of varying buffer size while keeping all other parameters the same as program-480 in Table 2.1. Fig. 2.8(a) shows the overall latency of baseline and our compiler as the buffer size increases. The latency of our compiled programs first decreases with the buffer size and then becomes stable, with the improvement factor first increasing and then becoming stable. The turning points of QFT, Grover and RCA are around 7, which takes $7/(30+7) = 18.9\%$ of the total #qubits per QPU. In contrast, the turning point of MCT is around 20, which is much larger than other benchmarks. This is because MCT is much more dominated by in-rack communications than other benchmarks. It can benefit from a larger buffer size, as in-rack EPR pairs can be collected and stored in the buffer with less restriction by network bandwidth.

Look-ahead depth We also illustrate the effect of varying look-ahead depth while

keeping all other parameters the same as program-480 in Table 2.1. Fig. 2.8(b) shows the overall latency of baseline and our compiler as the look-ahead depth increases. The latency of our compiler first decreases with the look-ahead depth and then becomes stable, with the improvement factor first increasing and then becoming stable. Similar to buffer size, the turning point of MCT is larger than other benchmarks. It can benefit from a larger look-ahead depth, as an increased look-ahead depth leads to an increased collection of its more dominant in-rack communications.

2.5.4 Sensitivity Analysis

This subsection provides a sensitivity analysis for various hardware parameters, demonstrating the adaptability of our compiler to a varying EPR supply (i.e., the number of communication qubits per QPU and the latencies of cross-rack and in-rack EPR generation) and a varying EPR quality (i.e., fidelity of cross-rack, in-rack and distilled EPR pairs). Since only the ratios between different latencies and fidelity matter, cross-rack and in-rack latency will be normalized by reconfiguration latency, while cross-rack and distilled in-rack fidelity will be normalized by the original in-rack fidelity.

Communication qubit number We illustrate the effect of varying #communication qubits per QPU by increasing it from 1 to 6, keeping all other parameters the same as program-480 in Table 2.1. As shown in Fig. 2.9(a), the overall latency of both baseline and our compiler decreases first and then becomes stable, which naturally arises from the increased bandwidth by communication qubits. The improvement factor of our compiler first increases and then becomes stable, which demonstrates its more effective utilization of network bandwidth than the baseline.

Cross-rack EPR latency We illustrate the effect of varying the latency of cross-rack EPR pair generation by increasing its ratio to reconfiguration latency from 5 to 30, keeping all other parameters the same as program-480 in Table 2.1. As shown in Fig. 2.9(b), the overall latency of both baseline and our compiler increases with cross-rack latency. These trends arise naturally from the fact that a longer cross-rack EPR latency leads to a longer overall latency. The

improvement factor of our compiler decreases with an increased cross-rack latency, but remains significant even if cross-rack latency is as large as 30× reconfiguration latency.

In-rack EPR latency We illustrate the effect of varying the latency of in-rack EPR pair generation by increasing its ratio to reconfiguration latency from 0.05 to 1, keeping all other parameters the same as program-480 in Table 2.1. As shown in Fig. 2.9(c), the overall latency of both baseline and our compiler increases with in-rack latency. These trends arise naturally from the fact that a longer in-rack EPR latency leads to a longer overall latency. In contrast to cross-rack latency, the improvement factor of our compiler slightly increases with the in-rack latency.

Relative cross-rack fidelity We analyze the effect of varying the fidelity of cross-rack EPR pairs. We fix the fidelity of in-rack EPR pairs to 95%, varying the fidelity of cross-rack EPR pairs from 75% to 95%, with the ratio between cross-rack and in-rack ones being from 0.79 to 1. All other parameters are kept the same as program-480 in Table 2.1. As shown in Fig. 2.10(a), the EPR overhead of our compiler slightly increases as this fidelity ratio becomes closer to 1. This is because our compiler adopts a tradeoff of incurring additional for hiding latencies of cross-rack communications. With a smaller fidelity distinction between cross-rack and in-rack pairs, the cost of additional in-rack EPR pairs would appear more significant.

Relative distilled in-rack fidelity We also analyze the effect of varying the fidelity of distilled in-rack EPR pairs, as the distillation can be improved by advancing distillation protocols or sacrificing more EPR pairs for the distillation. We fix the fidelity of in-rack EPR pairs to 95%, varying the fidelity of distilled in-rack EPR pairs form 95% to 99.5%, with the ratio between distilled in-rack and (non-distilled) in-rack ones being from 1 to 1.047. All other parameters are kept the same as program-480 in Table 2.1. As shown in Fig. 2.10(b), the EPR overhead of our compiler decreases rapidly with this fidelity ratio. That means if we can distill the additional in-rack EPR pairs to a high enough fidelity, the fidelity overhead brought by them will become negligible.

#EPR pairs per distillation While the fidelity of distilled in-rack EPR pairs can be

Table 2.3. QEC Integration: performance of our compiler and the baseline with surface code of distance 5.

Experiment	Benchmark-#alg_qubits	Baseline: Latency	Ours: Latency	Improv. Factor	#cross-rack EPR (15% infidelity)	#in-rack EPR (5% infidelity)	Ours: #distilled EPR (3.5% infidelity)	EPR Overhead	Baseline: Wait Time	Ours: Wait Time	Additional Wait Time	Retry Overhead
Surface Code ($d = 5$)	MCT-64	145,202	35,859	4.05×	1,920	7,680	2,621	12.01%	0.00	2.40	2.40	1.00
	QFT-64	1,239,922	277,850	4.46×	15,360	3,840	19,889	21.81%	0.00	5.02	5.02	1.00
	Grover-64	18,482	2,718	6.80×	120	480	180	13.04%	0.00	1.31	1.31	1.00
	RCA-64	16,777	3,965	4.23×	180	720	249	12.15%	0.43	3.02	2.59	1.00

enhanced by sacrificing more EPR pairs, the generation of these sacrificed EPR pairs can also increase the overall latency. However, our experiments show that this increase is not significant. This is because all the sacrificed EPR pairs are in-rack pairs, which can be generated collectively in our compiler. As shown in Fig. 2.10(c), the overall latency is increased by only 7.4% on average as the number of EPR pairs used in each distillation increases from 1 (i.e., no distillation) to 10.

2.5.5 Integration of QEC

We demonstrate the capability of our compiler to integrate QEC by an experiment with programs encoded in surface code, one of the most promising QEC codes [92]. We decompose programs into the Clifford + T basis [160], implementing logical operations with lattice surgery [141, 222], along with a magic state factory to facilitate logical T gates [91]. Therefore, the logical qubits include both algorithmic qubits required by the quantum program and additional qubits to facilitate quantum computation (e.g., magic states for implementing logical T gates [34]).

We adopt an architecture steup similar to the primary experiment, which consists of 4 racks, with each rack containing 4 QPUs, each QPU containing 2 communication qubits. Each benchmark contains 64 algorithmic qubtis. On each QPU, 4 algorithmic qubits of code distance $d = 5$ are arranged in a lattice, separated by ancilla qubits with a spacing of d [141], surrounded by a magic state factory at the periphery of the QPU. The EPR pairs can be stored with a low resource overhead [43, 197] to prevent from decoherence, until participating syndrome

measurements [121]. Specifically, each QPU has a buffer of 12 logical qubits with a code distance of 6, which are encoded in the [[72,12,6]] LDPC code [43]. This leads to a buffer qubit overhead that is much less than the number of qubits used for computation.

As shown in Table 2.3, our framework achieves a reduction in latency by an average factor of 4.89, with an average EPR pair overhead of 14.75%, an average additional wait time of 2.83, and an average retry overhead of 1.00 (i.e., no retry occurs). This demonstrates the applicability of our framework in fault tolerant quantum computing.

2.6 Conclusion

In this work, we provide in-depth analysis and discussion of the compilation challenges of scaling up quantum computing with QDCs based on reconfigurable optical switch network. We propose a compiler that optimizes the quantum communication in QDCs across both the program and network layers, which employs a look-ahead EPR scheduling along with two key optimizations: collective in-rack EPR pair generation and parallelized cross-rack EPR pairs generation. This compiler reduces the overall communication latency by a factor of 8.02, with an overhead of requiring 7.41% more EPR generation and increasing the wait time of EPR pairs by 6.51$\times$ switch reconfiguration latency. We also demonstrate its capability of integrating QEC by an evaluation on surface code. Moreover, we have open-sourced our codes to facilitate further research and collaboration within the community.

2.7 Acknowledgments

This chapter is a full reprint of material published as: Hezi Zhang, Yiran Xu, Haotian Hu, Keyi Yin, Hassan Shapourian, Jiapeng Zhao, Ramana Rao Kompella, Reza Nejabati, and Yufei Ding, "SwitchQNet: Optimizing Distributed Quantum Computing for Quantum Data Centers with Switch Networks" published in the 49th lEEE/ACM International Symposium on Computer Architecture (ISCA). The dissertation author is the primary investigator and author of this paper.

Chapter 3

OneQ: Compilation Framework for Photonic One-Way Quantum Computation

3.1 Introduction

Quantum computing (QC) is promising in providing significant speedup for many problems, such as large-number factorization [191], unordered database search [100] and simulation in quantum chemistry [53, 151]. Tremendous progress has been made in hardware technologies for quantum computing, including superconducting [75], ion trap [50], neutral atoms [182], and photonics [125], in both research lab settings and industrial production levels. With no clear winner among current platforms, it is expected that multiple technologies will continue to evolve together to meet diverse mission needs and application requirements.

In recent years, there has been a surge of interest in photonic quantum computing [201, 166], mainly due to the unique advantages of photonic qubits [162, 37], such as great scalability, long coherence time and easy integration with quantum networks. This interest is further fueled by the breakthroughs in photonic technologies [133, 221] and the successful demonstration of quantum supremacy on photonic quantum computers [145, 240, 239]. Moreover, PsiQuantum [24, 41, 26] recently announced the world's first manufacturing milestone for integrated quantum photonic chips, with quantum photonic and electronic chips being manufactured using advanced semiconductor tools.

However, photonic quantum computing also presents a unique set of features that cannot

be directly handled by conventional programming and compiler frameworks [173, 192] for solid-state qubits. First, unlike the permanent qubits in solid-state systems [75, 50, 182], photonic qubits are flying qubits at the speed of light, and are generated dynamically over time. Second, in contrast to the solid-state platforms which entangle qubits by multi-qubit gates such as CNOT, entanglement between photonic qubits are more natively performed by projective measurements onto entangled states, a process referred to as *fusion* (e.g., simultaneous measurements in XX- and ZZ-bases project the qubits onto Bell states). Third, measurements on photonic qubits, including single-qubit measurements and fusions, instantaneously destroy the photons, meaning that each photonic qubit can be measured or fused at most once and cannot be reinitialized.

At programming level, measurement-based quantum computing (MBQC) [176] has been suggested as a native programming paradigm for photonic quantum computing, because it does not require long-lasting qubits or quantum gates, but only one-time measurements on qubits. Unlike the circuit model which initializes qubits to a product state of all $|0\rangle$s and performs algorithms by a sequence of gates that gradually creates entanglements, MBQC initializes qubits to an entangled state and performs algorithms by a pattern of single-qubit projective measurements (called *measurement pattern*) that gradually consumes the entanglements. During the computation, each qubit is measured only once and is then free to detach from the computing system or even disappear. This is why MBQC is also called "one-way" quantum computing (1WQC). In addition to MBQC, another popular programming paradigm for photonic quantum computing [120, 145] is the special-purpose model for Gaussian boson sampling [102], but it is not a universal computing model like MBQC and thus has restricted applications.

Yet few attempts have been made on the compilation side. While previous work [176, 177, 46, 214] has shown that the utilization of a lattice-like entangled state, i.e., a *cluster state*, can enable a basic interpreter from the circuit model to MBQC by joining the measurement patterns of individual gates on it, this approach is not efficient for scaling up on hardware. Practical photonic hardware scales up by first generating small entangled states (called *resource states*) and then combining them into larger entangled states through fusions [96]. Attempting to

generate a large cluster state on such hardware would result in an unnecessarily significant fusion overhead, because a large amount of entanglements in the cluster state is actually redundant for the computation, but only to keep the lattice geometry. Instead of generating all entanglements of the cluster states, it is more efficient to only generate the entanglements required by the computation. Theses required entanglements can be represented by a non-lattice *graph state*, i.e. a generalized cluster state with arbitrary graph geometry. However, as the geometry of the graph state is irregular and varies depending on the specific program to be executed, it is not yet known how to efficiently map the irregular graph geometry onto an MBQC machine while complying with the hardware constraints.

In this paper, we investigate how to optimize the quantum program compilation for a photonic MBQC machine. As the first step, we abstract a *extendable space-time coupling graph* to formulate the hardware model. This coupling graph represents the hardware resources (e.g., resource state generators (RSG)) and constraints (e.g., available photon routing) in a way that suits the subsequent compilation designs. With this abstraction, we identify three key challenges in compiling graph states onto the hardware model. First, resource states are generated dynamically in layers over time by limited number of RSGs and thus cannot be available simultaneously. Second, there is no direct correspondence between a graph state node and a photonic qubit due to the lack of high-degree qubits in the resource states. Third, there is a mismatch between the irregular geometry of the graph state and the regular structure of fusion supports represented by the coupling graph. Additionally, for resource states of small size, there is a further constraint that non-planar graphs cannot be accommodated by resource states generated in the same layer since the number of possible fusions on each resource state is too limited.

To this end, we propose an optimizing compilation framework consisting of three sequential stages (Figure 3.1), with each stage handled by a key module of the framework. The **Graph Partition and Scheduling** module (Section 3.4) partitions the graph state into subgraphs and schedules them onto resource states generated at different times. It achieves a balance

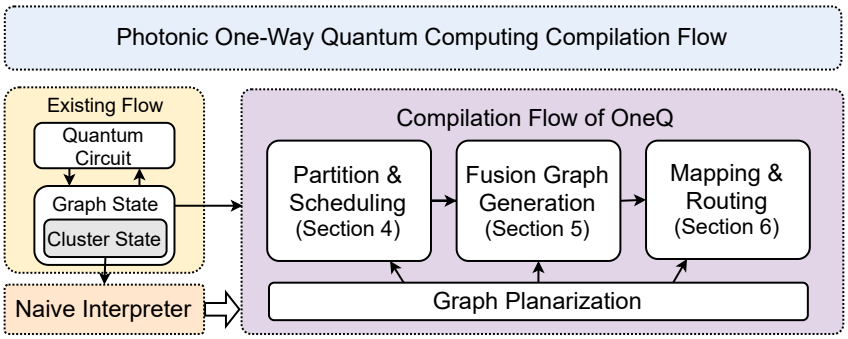


Figure 3.1. Framework Overview.

between reducing the wait time of photons and increasing the potential efficiency of mapping by analyzing the dependencies among operations and considering the overall geometry. The **Fusion Graph Generation** module (Section 3.5) synthesizes the scheduled subgraphs by fusing low-degree qubits in the resource states into high-degree nodes in the graph and connecting them appropriately. The produced fusion strategy is agnostic to the coupling constraints, and is represented by a *fusion graph*. The **Fusion Mapping and Routing** module (Section 3.6) tackles the coupling constraints by finding an embedding layout of the irregular fusion graph into the regular coupling graph. Specifically, it makes the fusion graph compatible with the coupling graph by extending and routing the edges of the fusion graph along the edges of the coupling graph. By maximizing the compactness of the layout, we can reduce the number of required fusions, thus reducing the computation overhead and enhancing the overall fidelity. Lastly, the additional constraint for small resource states can be addressed by a planarity enforcement for all modules. With these modules and optimizations, our framework maximizes the utilization of computing resources and efficiently accommodates the irregularity and adaptivity of the graph states.

To summarize, our contributions in this paper are listed as below:

- We abstract a extendable space-time coupling graph to formulate the hardware model of a realistic photonic MBQC architecture.

- We identify three critical compilation challenges in the gap between MBQC programs and realistic photonic MBQC hardware, as well as an additional constraint in the cases of small resource states.
- We propose an optimizing compilation framework to efficiently deploy quantum programs onto the hardware, which includes three key optimization modules to tackle the identified challenges, along with a planarity enforcement process to address the additional constraint.
- Our framework outperforms the basic MBQC interpreter by orders of magnitude in terms of execution time and resource consumption of the compiled program.

3.2 Background and Related Work

3.2.1 MBQC on Photonic Platforms

Photonic qubits are an exceptional carrier of quantum information and are well-suited for the one-way model of quantum computing [47, 158]. The feasibility of photonic one-way quantum computing has been shown by implementing various quantum algorithms on photonic qubits in cluster states [205]. For example, Grover's algorithm was implemented on a 4-photon 4-qubit cluster state in [216] with a fidelity of 0.9, Deutsch-Jozsa algorithm was realized on a six-qubit cluster state in [212] with a fidelity of 0.96, and Simon's algorithm was performed on a five-qubit cluster state in [202] for a period-finding problem. Notably, rapid progress is being made toward scaling up the photonic platform with integrated waveguides and optical chips [204, 185, 23, 87, 220, 88, 188], and a fully reconfigurable optical processor has been demonstrated in [54].

Practical photonic hardware is scaled up by generating small-size resource states (e.g., three-qubit [96], four-qubit graph states [25]) and cascading them via fusion [83]. Fusion is a native operation in linear optics that projects a two-qubit state to an entangled state through joint measurements on the two qubits. For example, joint measurements in XX- and ZZ-basis projects

two-qubit states onto Bell states. The joint measurements we use in this paper are in XZ- and ZX-basis, which merge two graph states to another graph state. As shown in Figure 3.2, the 3-qubit graph states ABC and DEF can be merged into a 4-qubit graph state $ABEF$ via a fusion on qubits C and D. The cost of each fusion is losing the two measured photons (qubit C and D in Fig. 3.2). In general, the fusion between an m-qubit graph state and an n-qubit graph state generates an $(m+n-2)$-qubit graph state, which is larger than the original graph states when $m,n>2$.

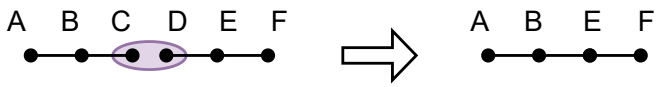


Figure 3.2. A fusion between graph states ABC and DEF. After the fusion between C and D, qubit C and D vanish while qubit A,B,E and F form a new graph state.

Despite facilitating scalability, fusions come with an overhead of losing two qubits each time and are the lowest-fidelity operations in the computing process. As a result, reducing the number of fusions is critical for reducing the required computing resources and enhancing the overall fidelity. Thus we consider it as one of essential optimization goals in our compilation framework, which is analogous to the goal of minimizing the number of CNOT gates in compilers for the circuit model.

3.2.2 Programming Paradigm of MBQC

MBQC Basics

MBQC is a universal but conceptually distinct computational model from the circuit model, as computation in MBQC is driven by single-qubit projective measurements instead of 1-qubit and 2-qubit gates. The initial state used as the computing resource in MBQC can be a *graph state*, which is an entangled state of qubits located on a graph $G=(V,E)$, formally defined

as the eigenstates of operator

$$s = X_i \bigotimes_{j \in n_i} Z_j, \quad \forall i \in V$$

where n_i is the set of neighboring qubits of $i \in V$. A graph state can be generated by preparing all qubits in $|+\rangle$ and applying a CZ gate on each pair of qubits connected by an edge $e \in E$. Given this graph state, computation can be driven by a pattern of Z-measurements and *equatorial measurements* $E(\alpha)$, i.e., measurements on the X-Y plane of Bloch sphere at an angle α. The graph G and the measurement basis of each qubit together form a *measurement pattern*.

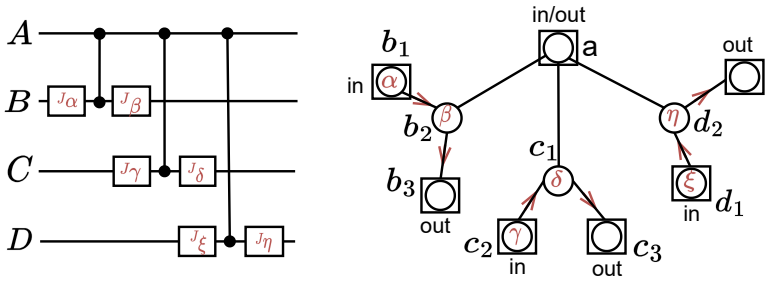
MBQC has a classical-quantum hybrid nature in that it relies on a classical feed-forward to address the non-determinism of quantum measurement. Specifically, the state of unmeasured qubits are subjected to Pauli X- or Z-corrections according to the outcomes of the measured qubits. In practice, this can be implemented equivalently by adaptively adjusting the measurement bases of the remaining qubits, such that all corrections can be postponed to the end of the computation. As a result, these *adaptive measurements* induce dependencies between measurements. In particular, an X^sZ^t-correction with $s,t \in \{0,1\}$ can be postponed by adjusting its measurement angle from α to $(-1)^s\alpha + t\pi$, i.e., $E(\alpha)X^sZ^t = E((-1)^s\alpha + t\pi)$. The dependencies in s and t are called X- and Z-dependency respectively.

There is a straightforward translation [48] from a circuit in the universal gate set $\{J(\alpha), \mathrm{CZ}\}$ into a measurement pattern, because a measurement $E(\alpha)$ yielding an outcome $m \in \{0,1\}$ is equivalent to a $J(\alpha)$ operator followed by a Z-measurement M_z yielding the same value, i.e., $E(\alpha)|\psi\rangle \to m$ corresponds to $M_zJ(\alpha)|\psi\rangle \to m$,

$$J(\alpha) = \begin{bmatrix} 1 & e^{i\alpha} \\ 1 & -e^{i\alpha} \end{bmatrix}$$

For example, Figure 3.3 shows the translation from 3.3(a) to 3.3(b), with the measurement angles of qubits and the roles of being input or output marked on each of them. The 3-degree node a in 3.3(b) corresponds to the qubit A with 3 CZ gates in 3.3(a), while nodes b_1, b_2 correspond to

single-qubit gates on qubit B (similarly for qubit C and D). Dependencies between measurements are represented by directed edges. For example, $b_1 \to b_2$ indicates that the measurement basis of b_2 may be adjusted according to the measurement outcome of b_1. This process can be rigorously described in ZX-calculus and optimized by available tools such as PyZX [122].



(a) (b)

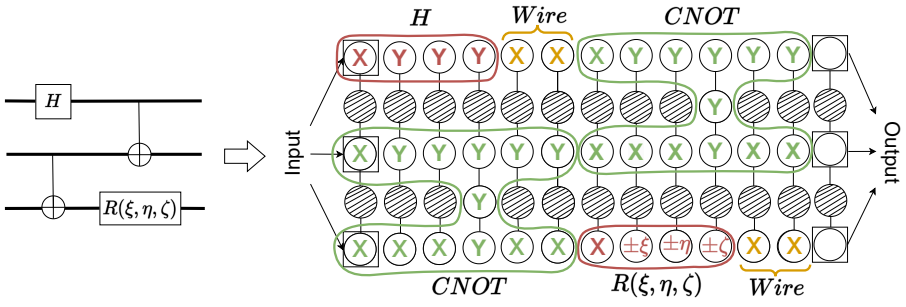
Figure 3.3. Translation from a circuit (a) to a graph state (b), with the measurement bases labeled on qubits. Qubits in (b) with 'in', 'out' or 'in/out' are input or output qubits. Arrows on the edges indicate the dependencies between measurements.

MBQC Interpreter

While there has not been a compiler for practical MBQC machines, previous work on cluster states enables a basic interpreter from the circuit model to MBQC, which we adopt as the baseline approach in our evaluation.

Cluster states are universal for MBQC as it allows the implementation of a universal gate set, such as single-qubit gates along with CNOT. In particular, an arbitrary single-qubit gate can be implemented by a measurement pattern on a line of qubits in the cluster state, such as the red circles in Figure 3.4(b), while a CNOT gate can be implemented on two lines of qubits joined by a qubit in between, such as the green circles in Figure 3.4(b). The X, Y labels stand for X-, Y-basis, while ξ, η, ζ stand for the angles of general equatorial measurements. Measurements in bases except X, Y and Z are adaptive measurements and can thus induce dependency. These adaptive measurements only arise in the measurement patterns of non-Clifford gates, meaning that all Clifford gates can be executed simultaneously on the cluster state even if they are dependent in

the circuit model. For more details about measurement patterns of different gates, please refer to [176].



(a) (b)

Figure 3.4. From a circuit (a) to an MBQC measurement pattern (b). Labels on the qubits stand for their measurement bases, with X, Y standing for X-,Y-basis, and ξ, η, ζ standing for the angles of equatorial measurements. Shaded qubits are redundant qubits which can be removed by Z-measurements.

A basic interpreter from the circuit model to MBQC can be constructed by simply joining the inputs and outputs of measurement patterns of those gates according to their positions in the circuit, as shown in Figure 3.4. Those measurement patterns can be aligned up by inserting identity gates, implemented by two adjacent X-measurements (or three adjacent Y-measurements), as shown by the yellow 'wires' in Figure 3.4(b). The qubits not used by measurement patterns of any gates are redundant to the computation, and can be removed by measurements in Z-basis (shaded qubits in Figure 3.4(b)). Similar to the circuit model compilation [136], CNOT gates on non-neighboring qubit strips can be implemented by inserting measurement patterns of SWAP gates whenever necessary. In this way, we can implement any quantum circuit in an MBQC manner. This interpreter greatly simplifies the mapping from qubits and gates in the circuit to the measurements on the cluster state. However, this approach is inefficient in the utilization of computing resources. Specifically, it makes many qubits redundant for computation and consumes their entanglements trivially by Z-measurements, without any regard for the huge overhead of generating those redundant entanglements on a practical MBQC machine.

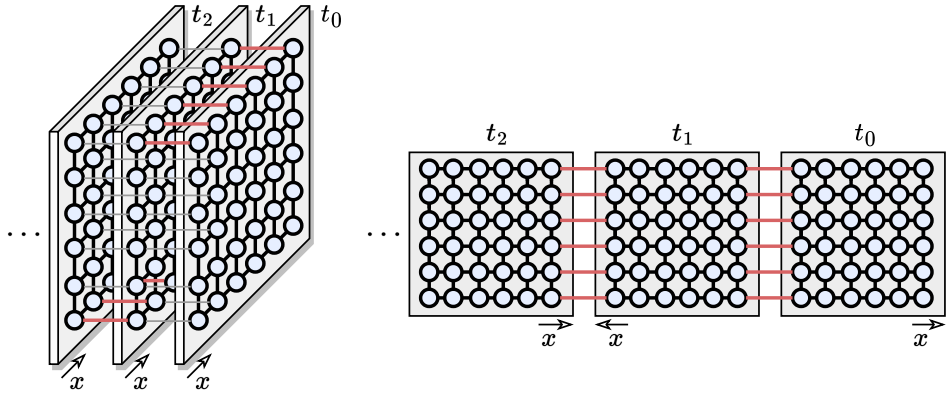
3.3 Problem Formulation

3.3.1 Hardware Abstraction

Photonic hardware for one-way quantum computing involves three key stages: generation, routing, and measurement. In the first stage, an array of *resource state generators (RSG)* periodically produces copies of resource states, with the period referred to as a *clock cycle*. In the second stage, routers channel the resource states to different measurement locations using *spatial* and *temporal routing*. Spatial routing allows fusions between resource states generated by different RSGs, while temporal routing allows fusions between resource states generated at different times using *delay lines*, which can delay the arrivals of photons at the measurement devices for some clock cycles. In the third stage, computation is performed by measuring photons in a predetermined pattern, with the outcomes fed into a classical computer to determine the bases of subsequent measurements.

It can be seen that photonic hardware has distinct features that set it apart from solid-state hardware. Specifically, photonic qubits are continuously generated over time and consumed by measurements. The use of identical resource states simplifies hardware manufacture and enhances efficiency, but they are typically in a small size [96, 25] and the number of resources states generated in each clock cycle is limited by the number of RSGs. Moreover, photonic hardware features two types of connections: spatial and temporal, both of which support fusions but are subject to different constraints. For example, spatial routing may only occur between resource states generated by neighboring RSGs, while temporal routing may only occur between resource states generated by the same RSG within a limited delay time.

To capture these features, we formulate a hardware model abstraction called the *extendable space-time coupling graph*, as illustrated in Figure 3.5. Each node in the coupling graph represents a resource state generated by the RSG in the corresponding location, with resource states in different layers generated at different times. Each edge of the coupling graph represents a support for possible fusion operations. There are three levels of coupling, which we describe in



(a) (b)

Figure 3.5. 3D structure of hardware connectivity (a) and its extendable space-time coupling graph (b). The physical layer t_1 in (b) is flipped in the horizontal direction so that the physical layers t_0, t_1 and t_2 and the their temporal connections in red form an extended physical layer. To keep the figure clear, other temporal connections are omitted in (b).

detail below.

Coupling in the resource state The first level of coupling comes from entanglements of qubits in the same *resource state*. These entanglements are generated directly by the RSGs and do not require fusion operations. As a result, this level of coupling is implicit in Figure 3.5.

Coupling within physical layers The second level of coupling comes from the resource states generated within the same clock cycle, which together form a layer of qubits referred to as a *physical layer*. As shown in Figure 3.5, the nodes in the physical layer at t_0 (or t_1, t_2) are connected into a 2D lattice structure. Such connections within a clock cycle represent the spatial connections, which allows a resource state to conduct a fusion with resource states generated at the same time by its neighboring RSGs.

Coupling across physical layers The third level of coupling is that a resource state can make a fusion with another resource state generated in a different clock cycle by the same RSG, up to the limit of the delay time, as shown by the red and grey vertical lines in Figure 3.5(a). Note that we should avoid using too long-range temporal connections because photons suffer more from the loss error as they stay longer in the delay lines.

In the coupling graph, the physical layers are extendable by combining consecutive physical layers and keeping their temporal connections at boundary. For example, in Figure 3.5, the three physical layers (3.5(a)) generated at time t_0, t_1 and t_2 can form an *extended physical layer* by keeping their spatial connections and the temporal connections in red (3.5(b)), with physical layer t_1 being flipped in the x direction. Temporal connections other than the red ones are ignored in this extension and thus are not explicitly shown in Figure 3.5(b). This extendability makes the area of physical layers flexible, and can allow mapping of more global structures onto the physical layers.

Despite the similarity to the quantum hardware coupling graph [136] prevailing used for today's superconducting and ion trap hardware architectures, we emphasize on the differences between them as following.

1) Different levels of coupling are not independent, and actually their interplay could create some additional constraints. For example, suppose the resource states generated by RSGs are 3-qubit resource states, then although the fusion device allows each location on the physical layers to maximally connect to 4 spatial neighbors (up, down, left, right) and the corresponding locations on previous and future physical layers up to the limit of delay lines, only up to 3 of them can be activated.

2) The temporal connections are convertable to spatial connections in the extended physical layers. The 3D connectivity structure of the coupling graph can be flattened into a series of 2D extended physical layers along the time dimension, with each extended layer consisting of several consecutive physical layers. Within each extended physical layer, the temporal connections (red lines in Figure 3.5) are treated in the same way as spatial connections.

3.3.2 Overview of our framework

As the first end-to-end compiler for MBQC, it is important to figure out a set of meaningful optimization goals. Our investigation on photonic one-way computing and realistic photonic quantum device development suggests the following two key optimization objectives. **1) Min-**

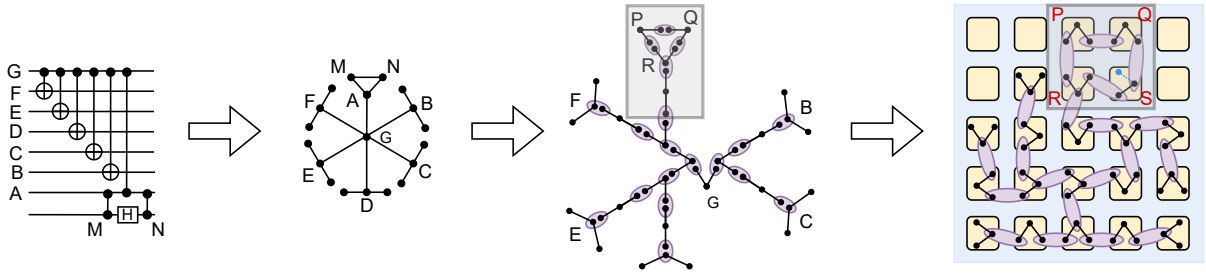


Figure 3.6. Compilation flow. (a) Quantum circuit. (b) Graph state. (c) Fusion graph generation. (d) Fusion mapping and routing.

imizing the physical depth. The physical depth is defined as the number of physical layers consumed in the computation, which represents the time of executing the compiled program. **2) Reducing the number of fusions**. Fusions are often the most low-fidelity and costly operations incurred in such an MBQC photonic quantum computer.

In the rest of this subsection, we use a motivating example (shown in Figure 3.6) to give high-level ideas about the key challenges introduced in Section 3.1 and the insights of how they can be addressed by our framework. We have attempted to maximize simplicity and ease of understanding, while possibly sacrificing some rigor. We often omit the dependency arrows and explicit measurement bases when they are not important for our optimization techniques, and they can always be derived from the graph state [48].

Challenge 1: Dynamic Computing Resources As resource states are generated dynamically in physical layers, a large number of qubits cannot be available simultaneously. In real-world quantum programs, graph states are typically too large (in terms of their number of nodes and edges) to be mapped onto a single or a few physical layers. Therefore, careful partition and scheduling schemes are needed to decompose large graph states into smaller components so that the graph state can be accommodated by the coordination of multiple physical layers. To minimize the delay time of photonic qubits and decrease their loss rates, dependent measurements can be scheduled according to their execution orders. However, this approach may compromise the compactness of graph state mapping onto the hardware by resulting in fragmented structures

that lose the overall geometry of the graph state. In our framework, this challenge is addressed by an approach that concurrently considers both the dependencies between measurements and the overall program geometry, allowing for a more efficient and effective allocation of resources. Section 3.4 gives detailed schemes of partition and scheduling.

Challenge 2: Lack of High-Degree Qubits High-degree nodes are a common occurrence in graph states, as qubits performing many two-qubit gates with other qubits (e.g. node G in Figure 3.6(a)) can result in high-degree qubit nodes in the corresponding graph state (e.g., node G in Figure 3.6(b)). However, such high-degree qubits may not be available in the resource states due to the limited inter-connections among the photonic qubits they contain (e.g., qubits in the 3-qubit resource states in Figure 3.6(c) have a maximum degree of 2). This prevents a direct correspondence between the graph state nodes and the photonic qubits. In our framework, those high-degree qubit nodes are synthesized by a proper strategy of fusions between low-degree qubits in resource states (e.g., node A to the shaded area in Figure 3.6(c), and node G to the area around G in the central part of Figure 3.6(c)). By matching with some basic fusion patterns, arbitrary high-degree input graph states (e.g., Figure 3.6(b)) can be synthesized to accommodate input adaptivity. Such a fusion strategy is agnostic to the routing constraints on the hardware, and aims to establish a viable fusion relation between resource states to synthesize the graph state eventually. The strategy can be abstracted to a *fusion graph* that indicate necessary fusions at the level of resource state. Section 3.5 gives detailed schemes for the generation of synthesis strategies and their fusion graphs.

Challenge 3: Mapping and Routing Constraints Due to the routing constraints, the irregular structure of fusions in Figure 3.6(b) may not be compatible with the hardware connectivity. In Figure 3.6(d), each yellow square represents a location of RGS, with the spatial routing allowing fusions between resource states from neighboring RGS's. It can be seen that the triangle in Figure 3.6(c) consisting of resource states P, Q and R (circled part in Figure 3.6(c)) cannot be mapped to the orthogonal grid directly because R can not be adjacent to both P and Q if P and Q are mapped to adjacent RSG locations. Such incompatibility exists widely in a

general fusion graph, which prevents a direct mapping from the fusion graph to the underlying hardware topology. In our framework this issue is resolved by extending some of the edges and winding them along the 2D lattice of individual or extended physical layers, which form paths of consecutive fusions. We refer to this procedure as fusion routing. For example, in Figure 3.6(d), P, Q and R can conduct fusions with each other with an auxiliary resource state S. In S, two qubits participate in the routing path RSQ, with the other qubit (in blue) removed by a Z-measurement. By maximizing the compactness of the mapping, we can minimize the overhead from extra fusions. Section 3.6 gives detailed searching schemes to map the edges onto orthogonal paths while minimizing the routing overhead.

Additional Challenge: Non-Planar Graph States The number of possible fusions on resource states is limited by the number of qubits they contain. Typically, for small resource states, each location in the coupling graph can be passed by at most one routing path. For example, in Figure 3.6(d), resource state S is passed by the routing path of edge QR, which destroys two of its qubits. This leaves only one qubit remaining, which is insufficient for use in another routing path. Therefore, the layout of the graph state cannot have edges intersecting each other unless the resource states are large enough to be splitted into multiple 2-qubit cluster states (e.g., as large as 5-qubit linear cluster state or 6-qubit ring-like cluster state). In other words, only *planar graphs*, i.e., graphs that can be embedded on a plane with no edges crossing each other, can be mapped to a single physical layer. In our framework, this is addressed by enforcing a planarization process in all modules to prohibit the graph non-planarity.

3.4 Graph Partition and Scheduling

Scheduling of operations for MBQC differs from that for the circuit model in two ways. First, the dependency relations in MBQC can be quite different from those in the circuit model. For example, dependencies between measurements occur only in the presence of non-Clifford gates, while all measurements corresponding to Clifford gates can be executed simultaneously. Second, the number of graph state nodes that can be accommodated by each physical layer is

flexible, which is hard to predict beforehand since it depends on the strategies of graph synthesis and routing. Specifically, the synthesis of a node in the graph state may result in multiple nodes in the fusion graph, while the mapping of an edge in the fusion graph may require a routing path through multiple locations in the coupling graph.

To handle these distinct features, we partition the graph state into subgraphs in a way that respects the measurement dependency, and allow a dynamic scheduling for each subgraph to accommodate the flexibility in the resource demand. The graph partition enables the qubits measurable earlier to be scheduled onto earlier physical layers in a coarse grain, which reduces the wait time of qubits in the delay lines and enhances the program fidelity. The dynamic scheduling schedules each partition of the graph state onto a flexible number of physical layers, with the physical layers allocated dynamically until the partition is all mapped. This enables the search for a compact layout of the scheduled partition, which increases the efficiency of resource utilization and reduces the total number of physical layers required by the program.

The partition of the graph state respects the measurement dependency through an analysis of the executability order of the measurement operations. Due to the non-deterministic nature of quantum measurements, MBQC requires a classical feed-forward that induces dependencies between measurements. The partial order between measurements can be described by a causal flow [48], which is determined by two types of dependencies. Specifically, based on the measurement outcome of a particular qubit, the measurement angles of its neighboring qubits may be adjusted from α to $(-1)^s\alpha + t\pi$ where $s,t \in \{0,1\}$, with the dependency on s called X-dependency and that on t called Z-dependency. Note that changing the angle from α to $\alpha+\pi$ only interchanges $|\pm_\alpha\rangle$ and $|\mp_\alpha\rangle$, which means the Z-dependency is equivalent to a re-interpretation of the measurement outcome of the affected qubit. Therefore the condition when a measurement becomes executable can be summarized as Lemma 1, with qubits executable at each time forming a *dependency layer*.

Lemma 1. *A measurement on a qubit in the graph state is executable if all its X-dependent*

qubits are measured and all the Z-dependent qubits of all its X-dependent qubits are measured.

This respect for measurement dependency is only in a coarse-grained manner. This means that a partition can contain multiple dependency layers, and dependency layers within the same partition do not have to be scheduled strictly according to their executability orders. The reason for this is two-fold. First, the presence of delay lines can allow the delay of qubits' arrival at the measurement devices for some time, making an exact match between the executability order and the physical layer order unnecessary. Second, allowing for this mismatch enables a preservation of the overall geometry structure among nearby dependency layers, which can be used to increase the potential compactness of the graph layout in the mapping process. Specifically, this coarse-grained manner can be achieved by first sorting the measurements according to their executability orders, and then forming each graph partition by including a number of consecutive dependency layers.

Partitioned subgraphs are then scheduled for fusion graph generation and layout search for the fusion graph in ascending order of their executability. The scheduling of these subgraphs is dynamic in two levels. First, for each partition, physical layers would be allocated dynamically for mapping and routing until all the nodes in the corresponding fusion graph are mapped to the hardware. Second, among partitions, there can be additional edges connecting nodes of fusion graphs for different partitions, which also need to be mapped to the hardware to form the whole graphs state. This can be achieved by dynamically allocating additional physical layers between the partitions, and routing on those physical layers until all cross-partition edges are mapped.

Graph Planarization To further respect the planarity constraint for resource states of small size, the planarity of the subgraphs can be used as an additional condition for graph partitioning. Specifically, in the process of grouping consecutive dependency layers into partitions in ascending order of their executability, a planarity check can be performed along the way. New dependency layers will not be included if either the limit number of dependency layers is exceeded or the accumulated subgraph is no longer a planar graph. If a single dependency layer

is already non-planar, it can be decomposed into subgraphs by repeatedly finding the maximal planar subgraph G_i from its remaining graph G, where a maximal planar subgraph G_i is defined as a subgraph such that the addition of any edge $e \in G - G_i$ breaks the planarity of G_i. This planarization can ensure in advance that each partition scheduled for fusion graph generation and layout searching is a planar graph, avoiding unnecessary conflicts when the graph is large and complex.

3.5 Fusion Graph Generation

A graph state can be synthesized from small resource states via fusion by first synthesizing necessary components such as high-degree nodes and lines, then connecting them according to the graph geometry. In this section, we lay the foundation of synthesizing arbitrary graph states by defining a set of basic patterns. Then we demonstrate the process of graph synthesis from 3-qubit resource states and generalize it to more generic resource states.

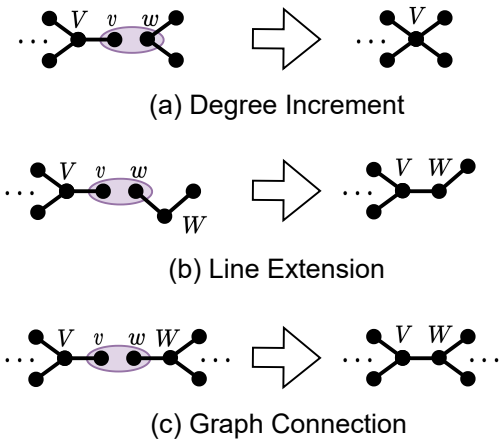
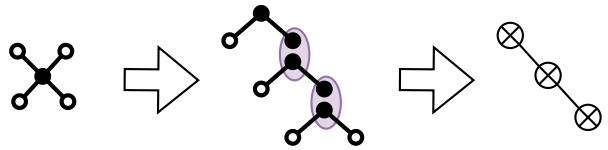


Figure 3.7. Basic fusion patterns.

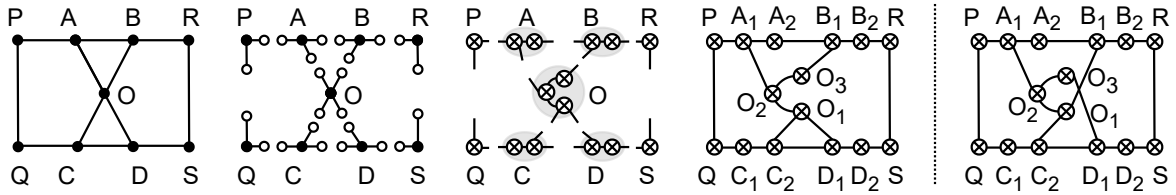
Synthesis of arbitrary graph states can be realized by appropriate strategies based on some basic fusion patterns, as shown in Figure 3.7. First, the degree of a node V can be increased by a *degree increment* pattern, which fuses a leaf node (degree $= 1$) connecting to V with a node of degree > 1. For example, the degree of node V in Figure 3.7(a) is increased by 1 by fusing

node v with a 2-degree node w. Second, a line of nodes can be extended by a *line extension* pattern, which fuses a leaf node of the line with that of another line. For example, the 2-node line Vv in Figure 3.7(b) is extended to a 3-node line by fusing the leaf node v with the leaf node w. Third, two resource states can be connected while maintaining their structures (except the fused nodes) by a *graph connection* pattern, which fuses two leaf nodes from each of them respectively. For example, the two 4-node graphs in Figure 3.7(c) are connected by a fusion between node v and w, with the nodes V and W connected by an edge in the formed graph state and the remaining structures of them left unchanged.



(a) (b) (c)

Figure 3.8. Synthesize a high degree node in (a) by a basic fusion pattern in (b). Represent the fusion pattern by a fusion graph in (c), with each '⊗' node representing a resource state, each edge representing a fusion operation.



(a) (b) (c) (d) (e)

Figure 3.9. Fusion graph generation. (a) A planar embedding of the graph state geometry. (b) Split the edges. (c) Replace each high-degree node with a piece of fusion graph, with each '⊗' node representing a resource state. (d) Connect the pieces of fusion graphs while keeping the clockwise (counter-clockwise) edge orders in the planar embedding so as to obtain a planar fusion graph. (e) The planarity may be broken if not keeping the clockwise (counter-clockwise) edge orders.

With these basic patterns, an arbitrary graph state can be synthesized from resource states as small as 3-qubit resource states. Specifically, high-degree nodes in the graph states can be

synthesized by a repetitive application of the degree increment pattern until the required degree is satisfied, as shown in Figure 3.8(a)(b). The resulting fusion pattern can be represented by a *fusion graph* in Figure 3.8(c), with each node representing a resource state, and each edge representing a fusion operation. Similarly, lines can be synthesized by a repetitive application of the line extension pattern until the required length is satisfied. Then, the high degree nodes and lines can be connected via the graph connection pattern (Figure 3.7(c)). We demonstrate it with an example of synthesizing a graph in Figure 3.9(a). In Figure 3.9(b), we break down the graph into high degree nodes A, B, C, D, O and lines PQ, RS. Each high-degree node or each line is connected with some nodes in hollow dots, which represent its neighbors in Figure 3.9(a). Those nodes are referred to as *virtual nodes*. Note that multiple virtual nodes are allowed to represent the same node. We synthesize the high-degree nodes and lines with the basic patterns Figure 3.7(a)(b), pairing up the virtual nodes according to the original graph in Figure 3.9(a), and connect these components by applying the basic pattern Figure 3.7(c) on the paired virtual nodes.

For generic resource states, this synthesis process can be made applicable by a modification containing two steps. First, nodes of the fusion pattern in Figure 3.8(c) can be reduced by utilizing qubits of degree > 2. In particular, for resource states with max degree m, an n-degree node can be synthesized by $n//m+1$ resource states, where the extra $(n \bmod m)$ qubits would be removed by Z-measurements. Therefore, the fusion graph in Figure 3.8(c) becomes a linear graph containing $n//m+1$ nodes, instead of $n-1$ nodes in the case of 3-qubit resource states. Second, edges in the fusion graph for 3-qubit GHZ states can be contracted by utilizing the connections between qubits within the resource states. This can be achieved by a pattern matching process that traverses the graph, searching for lines or cycles that match the intrinsic geometry of the resource states. During the synthesis process, qubits in resource states can be removed by Z-measurements when necessary. For example, a n-qubit ring-like resource state can be tailored to a $(n-1)$-qubit linear resource state by removing one qubit, if linear structures, rather than rings, need to be synthesized.

Planarity Preservation The planarity condition imposed in the graph partition can be preserved in the process of fusion graph generation. In other words, if the original graph is planar, then the resulting fusion graph can also be planar. This can be achieved by first finding a planar embedding of the original graph, i.e., a way of drawing it on the plane with no crossing edges, and then keeping the clockwise (counterclockwise) orders among the edges when connecting the synthesized high-degree nodes. The search for the planar embedding can be efficiently done by existing tools such as Networkx [101]. For example, in the planar graph in Fig. 3.9(a), the 4-degree node O has neighbors A, B, D, C in the clockwise order. So we replace it with a fusion graph component consisting of 3 nodes (component O in the grey circle at the center of Fig. 3.9(c)), and connect it with fusion graph components of A, B, D, C in a clockwise manner (Fig. 3.9(d)). As a comparison, Fig. 3.9(e) shows a fusion graph that breaks planarity since the clockwise (counterclockwise) edge orders are not maintained.

3.6 Fusion Mapping and Routing

Finding the most compact layout of a graph on a grid is known to be an NP-hard problem [169], and it is particularly challenging in our case due to two additional features of the grid. First, each layer of grid has limited area because each physical layer of resource states is generated by limited number of RSGs. Second, the space-time coupling in the hardware has a 3D nature, which makes a direct search of mapping inefficient. In particular, each edge of a node can be mapped or routed toward not only different directions around the node but also any of the previous or subsequent layers up to the limit of delay time, which increases the search space and leads to longer search times.

To address these challenges, we propose an algorithm consisting of an *in-layer mapping* and an *inter-layer shuffling*, which is applicable for grids of various geometry (triangular, orthogonal, etc.). It finds the 3D layout of the scheduled graph by finding a series of 2D compact layouts and connecting the them by a shuffling of nodes. Each 2D layout is found by a boundary-aware heuristic search restricted to a physical layer or extended physical layer. The *incomplete*

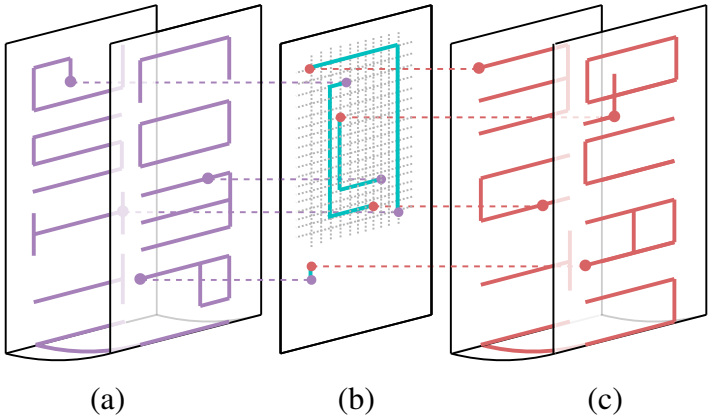
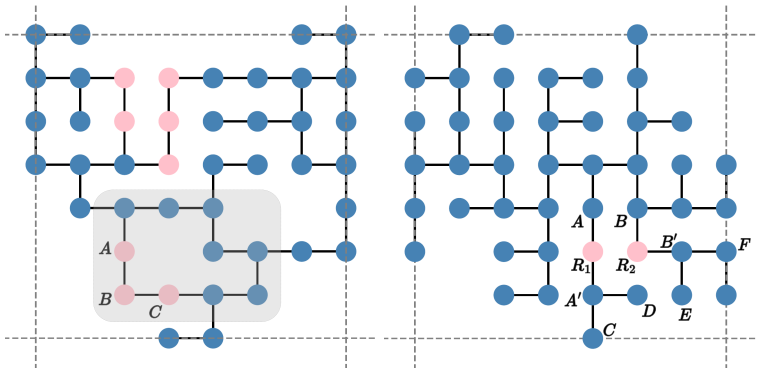


Figure 3.10. Mappings of ‘QUANTUM’ (red) and ‘COMPUTING’ (purple) connected through a shuffling layer (b) in between. Red dots on (c) represent the incomplete nodes whose edges are not all mapped yet, with the other ends of them being the purple dots on (a). Corresponding nodes are connected by a shuffling on the intermediate layer (green lines in (b)).

nodes on each layer, whose edges are not all mapped due to the limited grid area are then connected by a node shuffling on additional physical layer between the 2D layouts. For example, in Figure 3.10, the graph ‘QUANTUM’ (red) and ‘COMPUTING’ (purple) are mapped onto two extended physical layers in 3.10(c) and 3.10(a), respectively. Red dots on 3.10(c) represent the incomplete nodes whose edges are not all mapped yet, with the other ends of those edges being the incomplete nodes represented by the purple dots on 3.10(a). These incomplete nodes are connected by a shuffling on the intermediate layer 3.10(b), with the green lines being the routing paths between corresponding nodes.

In the in-layer mapping, each edge is mapped onto the physical layer or extended physical layer either directly or via routing. The routing is triggered in two scenarios. First, it can map the edges between existing nodes in the layout, thus allowing the fusion between non-neighboring nodes and accommodating the irregularity in the structure of the scheduled graph. Second, it is used to reduce the blockage between mapped nodes and future nodes, thus increasing the number of edges that can be mapped to the layer and maximizing compactness of the layout. A node is defined as *blocked* if the number of its remaining unmapped edges r exceeds the number of its adjacent available positions on the grid s, with the case of $s > 0$ called *partially blocked* and the

case of $s = 0$ called *totally blocked*. When a node is partially blocked by other mapped nodes, we can route the node to a position that has enough available degrees. When the direct mapping of an edge result in a total blockage for some other nodes in the map, we can route the node to a broader area before the mapping, such that it leaves enough degrees for the other nodes.



(a) (b)

Figure 3.11. (a) Mapping of fusion graph for 3-qubit QFT. (b) Mapping of fusion graph for 8-qubit BV with secret string '11111111'. Blue and green dots correspond to the nodes '⊗' existing in the fusion graph, with the green ones being the incomplete nodes whose edges are not all mapped yet. Pink dots represent the auxiliary resource states used for routing.

This can be more clearly demonstrated with two layout examples. In Figure 3.11(a), the routing path ABC, represented by pink nodes, is used to connect the existing nodes and form the cyclic structure in the shaded area. In Figure 3.11(b), routing was triggered for twice. First, node A' is routed from R_1 by one step to avoid the total blockage for node B, because if node A' is directly mapped to the location R_1, one of its edges $A'C$ and $A'D$ would have to take location R_2, which is the only remaining location around B. Second, node B' is routed by one step to address its partial blockage by $A'D$, because while node B' can be directly mapped to the location R_2, the only one remaining location around it would be insufficient for mapping its two future edges $B'E$ and $B'F$. Note that in real cases the length of routing paths should be ≥ 2.

The heuristic search algorithm can be stated as below. We traverse the edges in a cycle-prioritized breadth-first order, giving priority to edges involved in cycles over those in trees, because tree edges are more flexible for mapping. We choose to map each edge either directly

or via routing according to the scenarios above. After each new edge is mapped, the layout is evaluated by the heuristic cost function

$$\begin{aligned} H = \ & \text{occupied\_area} \\ & + \#\text{partially\_blocked\_nodes} \\ & + \alpha \cdot \#\text{totally\_blocked\_nodes} \end{aligned}$$

where occupied_area refers to the area of the smallest rectangle that can enclose the mapped edges. α should be larger than 1 as a total blockage is more harmful. It can be typically set as the maximum degree of each physical layer. The layout with the lowest cost is then selected as the best candidate and used in the search for future edges.

In the inter-layer shuffling, incomplete nodes in different 2D layouts are connected via routing on the physical layers allocated between them. This shuffling can be achieved by first pairing up the incomplete nodes, sorting the node pairs according to their distances, and then finding the shortest routing paths to connect the node pairs in ascending order of the distances. Specifically, if a single physical layer is not enough to accommodate all the routing paths, more physical layers can be allocated dynamically until all the node pairs are connected. The inter-layer shuffling can be used to support the subgraph connection need arising in two circumstances. First, as the graph partition process partitions the whole graph state into subgraphs, the fusion graphs of different subgraphs need to be connected to form the whole graph state. Second, the mapping of each partition may generate multiple 2D layouts, as some nodes may be inevitably blocked due to the limited area of physical layers and the complexity of the graph geometry. While positions of incomplete nodes in the previous layers are taken into account in the mapping of subsequent layers to increase their proximity, those 2D layouts may still need to be connected by the inter-layer shuffling if not all conflicts can be resolved within the in-layer mapping.

Planarity-Aware Search When the given fusion graph is planar, the heuristic search for in-layer mapping can be made planarity-aware to maximize the number of edges accommodated

by each physical layer. Specifically, we can first find a planar embedding of the fusion graph, and make the mapping of each edge follow the planar embedding, which can be achieved by forcing the mapped edge to leave enough degrees for its clockwise or counterclockwise neighboring edges connected to the same node. This planarity-awareness naturally prevents routing paths from crossing each other and reduces the demand for inter-layer shuffling. This approach is more effective when the area of physical layers is large enough, because otherwise the edges would be blocked by the limited area anyway. This area can be increased by extension of physical layers, as shown in Figure 3.5(b).

3.7 Evaluation

3.7.1 Experiment Setup

Metric As discussed in Section 3.3.2, the first metric we consider is the physical depth, i.e., the number of physical layers consumed in the execution of the program. The second metric is the number of fusion operations performed in the execution of the program, which is denoted as '# fusion'. A better compiler should compile a quantum program into a smaller depth which indicates a shorter execution time and less consumption of resources, and a smaller '# fusion' which indicates less overhead and being less error-prone. For comparison with the baseline, we define an improvement factor of physical depth as the ratio of the depth in the baseline to the depth in our framework, and an improvement factor of '# fusion' as the ratio of the number of fusions consumed in the baseline to the that consumed in our framework.

Baseline Our optimized baseline is based on the basic MBQC interpreter introduced in Section 3.2 but with two preprocessing steps. The first step is synthesizing a sufficiently large 3D cluster state by fusions from small resource states generated by the RSGs. The sufficient size of each 2D layer of the cluster state is referred to as *cluster area*, while the number of RSGs required to form a cluster state of that size is referred to as *physical area*. The second step is finding a layout for the circuit so that measurement patterns of all gates in the circuit can be

joined in a way shown as Figure 3.4.

For the first step, instead of relying on any specific synthesis strategy, we adopt a lower bound of the physical area, which indicates the least number of required resource states when ignoring the spatial constraints for fusion routing. For the second step, to ensure an efficient implementation of a two-qubit gate with MBQC, it is necessary to place the strips of the involved qubits close to each other in the cluster state, similar to that in the circuit model with solid-state-qubit quantum computing. As a result, we utilize the qubit mapping and routing functions of Qiskit [173] to address the issues related to far-apart 2-qubit gates prior to mapping the circuit onto the cluster state. Specifically, this is achieved by using a 2D coupling graph that shares the same structure with each 2D layer of the cluster state as the hardware coupling graph to Qiskit.

Benchmark programs We select Quantum Fourier transform (QFT), quantum approximate optimization algorithm (QAOA), Ripple-Carry Adder (RCA) [64] and Bernstein-Vazirani (BV) algorithm as our benchmarks, which include both building blocks of quantum programs (QFT and RCA) and application driven programs targeting at solving real-world problems (QAOA and BV). For QAOA, we choose the graph maxcut problem on randomly generated graphs. Specifically, the graphs are generated by randomly selecting half of its all possible edges. For BV, we select the secret strings randomly, with approximately half of the digits being 0 and half being 1. In table 3.1, we list the number of qubits and the number of gates for each benchmark, as well as the cluster area and physical area required by the baseline approach. For each benchmark, our framework is evaluated on the same physical area with the baseline to make them more comparable.

3.7.2 Experiment Result

We first show the results of the baseline and our framework for 3-qubit resource states in Table 3.2, as the number of qubits increases for each benchmark. It can be seen that our framework outperforms the baseline by orders of magnitude in both physical depth and the number of fusions. As the number of qubits increases, this improvement of performance stays

Table 3.1. Benchmark programs.

Name	#qubit	cluster area	physical area
Quantum Fourier Transform (QFT)	16	7x7	16x16
	25	9x9	21x21
	36	11x11	25x25
Quantum Approximate Optimization Alg (QAOA)	16	7x7	16x16
	25	9x9	21x21
	36	11x11	25x25
Ripple-Carry Adder (RCA)	16	7x7	16x16
	25	9x9	21x21
	36	11x11	25x25
Bernstein Vazirani (BV)	16	7x7	16x16
	25	9x9	21x21
	100	19x19	43x43

stable or slightly increases.

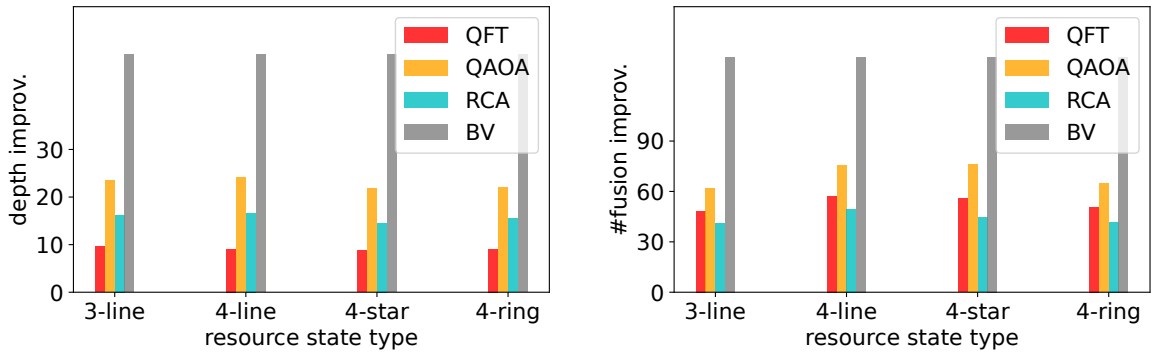
Among the benchmarks, our framework has the most significant improvement for BV. This is because the graph state of BV is acyclic and planar, making it easier the heuristic search in our framework to find a compact layout for the graph and map it to only a few physical layers. In contrast, the graph states of the QFT, QAOA and RCA contain a lot of short-range and long-range cycles, which makes the structure more complex and increases the demand for routing. However, for these benchmarks, our framework still achieves a reduction on the physical depth by a factor of 15 on average (geomean), and a reduction on the number of fusions by a factor of 47 on average (geomean).

General resource states To achieve greater generality, our optimization modules are designed to be extendable to a wide range of resource states. This is made possible by accommodating flexible fusion strategies for synthesizing graph states and representing each strategy with a fusion graph. Figure 3.12 shows the results when the resource states are 3-qubit line-shaped graph state, 4-qubit line-shaped, star-shaped and ring-shaped graphs states, respectively. The improvements in physical depth and the number of fusions are shown in (a) and (b), respectively.

Table 3.2. The results of our framework and its relative performance to the baseline.

Name-#qubits	Baseline Depth	Our Depth	Improv. Factor	Baseline #Fusions	Our #Fusions	Improv. Factor
QFT-16	787	83	9	201,472	8,167	25
QFT-25	1,518	162	9	669,438	26,921	25
QFT-36	2,712	324	8	1,695,000	66,830	25
QAOA-16	595	29	20	152,320	2,578	59
QAOA-25	1,287	63	20	567,567	8,343	68
QAOA-36	2,648	122	22	1,655,000	21,302	78
RCA-16	734	46	16	187,904	4,568	41
RCA-25	1,273	65	20	561,393	8,915	63
RCA-36	1,934	85	23	1,208,750	14,115	86
BV-16	94	1	92	24,064	63	382
BV-25	181	1	181	79,821	114	700
BV-100	787	4	197	1,455,163	644	2,260

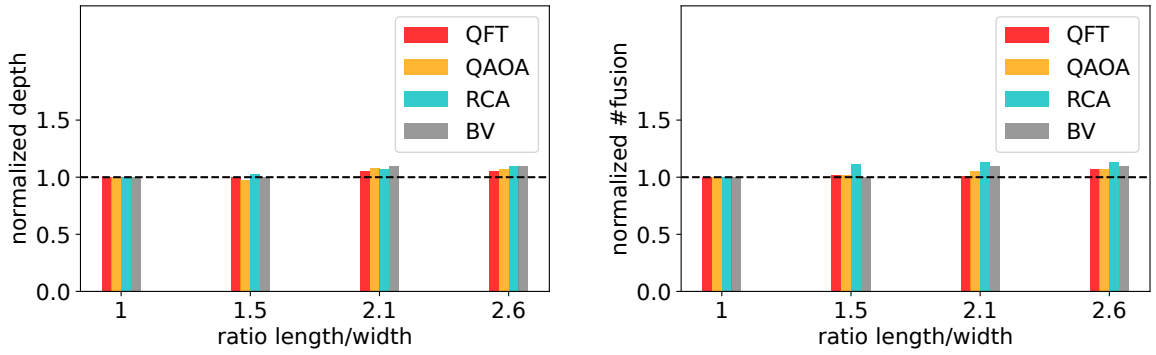
It can be seen that our framework achieves similar levels of improvement in performance for these various resource states.



(a) (b)

Figure 3.12. Improvement factors of physical depth (a) and # fusions (b) of 16-qubit QFT, QAOA, RAC and BV for different basic resource states.

Flexible coupling structure While the results in Table 3.2 are on physical layers of $N \times N$ square coupling structure, our framework is applicable to more general coupling structures. Figure 3.13 shows the physical depth (3.13(a)) and the number of fusions (3.13(b)) when the



(a) (b)

Figure 3.13. Normalized physical depth (a) and # fusions (b) of 16-qubit QFT, QAOA, RCA and BV on different shapes of physical layers. Results on rectangular physical layers (ratio > 1) are normalized by those on square physical layers (ratio $= 1$).

ratio of each physical layer is 16x16 (ratio = 1), 20x13 (ratio = 1.5), 23x11 (ratio = 2.1) and 26x10 (ratio = 2.6) respectively. For comparison, the results are normalized by those on the square physical layer. It can be seen that our framework achieves similar levels of performance on these various shapes of physical layers. Moreover, the area of physical layers is adjustable by regarding multiple consecutive physical layers as an extended physical layer, which can be used to accommodate more global structure of the graph. As a demonstration, Figure 3.14 shows a slice of mapping generated by our framework for a 16-qubit QFT program. The whole rectangular 13×39 extended physical layer is composed of 3 layers of 13×13 grid. The first physical layer ranges from column 1 to column 13, the second ranges from column 14 to column 26, and the third from column 27 to column 39 respectively. In the figure, blue and green dots correspond to the nodes '$\otimes$' existing in the fusion graph, with the green ones representing the incomplete nodes, which means that some of edges connected to them have not been mapped onto the grid. Pink dots represent the auxiliary resource states used for routing. In principle, the optimization techniques in our framework is also applicable when the coupling structure between RSGs are not orthogonal (e.g., triangular, hexagonal) since none of our optimization modules has explicit dependency on a specific coupling architecture.

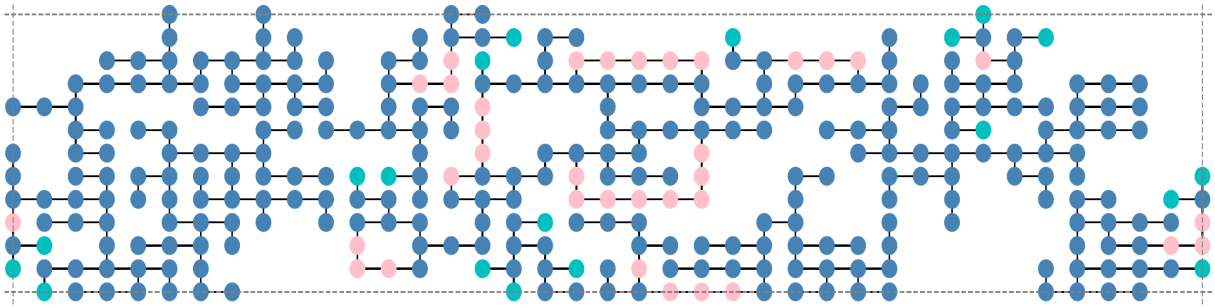
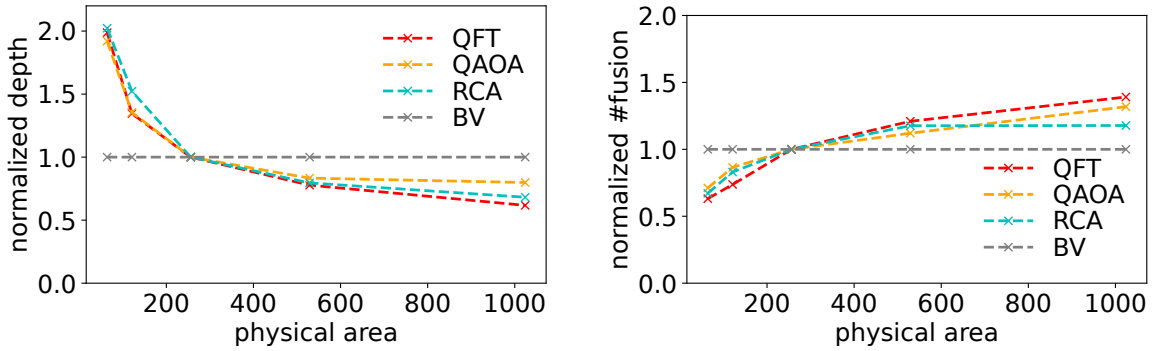


Figure 3.14. Mapping of the fusion graph of a 16-qubit QFT program with a more global optimization by considering an extended physical layer consisting of 3 successive physical layers of size 13×13. Blue and green dots correspond to the nodes '$\otimes$' existing in the fusion graph, with the green ones being the incomplete nodes whose edges are not all mapped yet. Pink dots represent the auxiliary resource states used for routing.

Effect of physical area In contrast to the fixed size of physical layers required by the baseline approach, our framework does not impose strict constraints on the size of physical layers, allowing programs to be compiled onto hardware of different sizes. Specifically, the physical depth can be decreased by allowing longer routing paths on larger physical layers, with a cost of more fusions. Figure 3.15 shows the physical depth and the number of fusions when the programs are compiled onto different sizes of physical layers. For comparison, the results are normalized by those on the physical area required by the baseline approach, which is 16x16 when the number of qubits is 16. It can be seen that when the physical area increases, the physical depth initially decreases rapidly, but eventually reaches a plateau when the physical area becomes sufficiently large. On the other hand, the number of fusions shows a reverse trend, increasing as the physical area increases. Note that if physical layers are larger than necessary, they can be truncated by restricting compilation to a portion of the physical layers.

3.8 Conclusion

In this work, we provide in-depth analysis and discussion of the unique challenges for photonic one-way computing at both programming and hardware levels. We build the first end-to-end compilation framework for optimizing the mapping of any quantum programs onto photonic



(a) (b)

Figure 3.15. Normalized physical depth (a) and # fusions (b) of 16-qubit QFT, QAOA, RAC and BV for different physical areas. Results on different physical areas are normalized by those on the same physical area as required by the baseline approach.

devices supported by our hardware abstract model. With this being said, many of the optimization problems here are NP-hard problems, and thus we believe there is still significant potential for fully exploring the entire optimization space. As the first work in this direction, we hope our work could attract more effort from the computer architecture and compiler community to explore the advantages of photonic quantum architectures and overcome the unique challenges.

3.9 Acknowledgments

This chapter is a full reprint of material published as: Hezi Zhang, Anbang Wu, Yuke Wang, Gushu Li, Hassan Shapourian, Alireza Shabani, and Yufei Ding, "OneQ: A Compilation Framework for Photonic One-Way Quantum Computation" published in Proceedings of the 50th Annual International Symposium on Computer Architecture. The dissertation author is the primary investigator and author of this paper.

Chapter 4

OnePerc: A Randomness-aware Compiler for Photonic Quantum Computing

4.1 Introduction

Photonic platform holds great promise for universal quantum computing due to the unique advantages of photonic qubits [162, 37], including their great scalability, long coherence time and easy integration with quantum networks. Besides the experimental demonstration of quantum supremacy on photonic systems [240, 239, 145], PsiQuantum has proposed their technology roadmap towards one million qubits using silicon photonics [180, 39]. The potential high clock speed of this approach [39] could make photonic platform advantageous for near-term quantum algorithms.

Photonic quantum computing differs from other platforms such as superconducting [75], ion trap [50] and neutral atoms [182], as it is scaled up by a probabilistic operation known as *fusion* [180]. Fusion plays a key role of forming large-scale entanglements between photonic qubits by merging small entangled states into larger ones upon success. Its probabilistic feature comes intrinsically from the degeneracy in fusions' outputs for different input states [99], bringing significant randomness to the computing process. On the hardware side, improvement of fusion success probability to a high value requires an impractical amount of ancillary resources [84, 99]. On the software side, this randomness is not taken into consideration by existing software

infrastructures for the circuit-based model [159] (e.g., Qiskit [173], Tket [193]). This is because the weak interaction between photons makes it hard to realize 2-qubit gates in the circuit model, but favors a different computing model known as measurement-based quantum computation (MBQC) or one-way quantum computation (1WQC) [176, 48].

Recently, as an initial effort towards efficient photonic MBQC, a compilation framework OneQ [235] has been proposed to significantly reduce the depth of compiled programs and the number of required fusions. However, it overlooks the severe randomness brought by fusion failures, simply assuming that fusions always succeed. As a compiler, OneQ translates the construction of a program-specific entangled state called *graph state* [48] (Fig. 4.1(a)) into a fusion pattern between the small entangled *resource states* [180] available on the hardware (Fig. 4.1(b)). However, when fusion failures occur in real-time execution, as illustrated in Fig. 4.1(b), the resulting state becomes a random graph state deviating from the target structure. Thus the execution needs to be retried until success, which is non-scalable given that a practical fusion success probability in the near term is merely around 75% [84, 99]. From now on, we will refer to the graph states required by programs (e.g., Fig. 4.1(a)) as *program graph states* and those generated by fusions (e.g., Fig. 4.1(c)) as *physical graph states*.

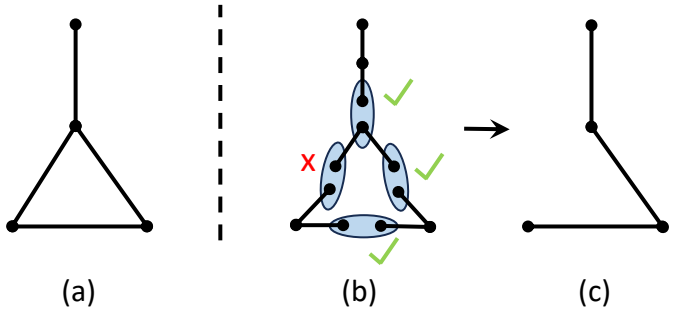


Figure 4.1. Randomness brought by fusion failures.

The objective of this paper is to present a scalable and efficient compilation framework capable of effectively handling fusion failures in the computing process. This is intuitively a hard problem. Firstly, with a failure rate as high as 25%, it seems impossible to ensure the

formation of any specific entanglement structure among the photonic qubits. Secondly, failed entanglements such as the disconnected edges in Fig. 4.1 cannot be recovered since the photons involved in the fusion are completely destroyed by the fusion. Thirdly, the limited lifetime of photons refuses the execution of over-complex algorithms in real-time, as photons are prone to an increasing loss rate with prolonged storage time in fibers [139].

Fortunately, there are some nice features of fusions and graph states that we can leverage. Firstly, fusion failures are heralded [126], allowing real-time awareness and enabling the incorporation of classical feed-forward [126, 233, 184] to adjust subsequent operations based on prior fusion outcomes. Secondly, when the fusion success probability exceeds a threshold, the resulting physical graph state contains a long-range-connected component with a high probability. This widely studied phenomenon, known as percolation [97, 167, 49], plays a crucial role in providing viable computing resource, inspiring our framework's name, OnePerc (one-way quantum computing based on percolation). Thirdly, a random graph state can be reshaped into any subgraph of it by eliminating the redundant qubits, which can be achieved by measuring them out in Z-bases [176].

However, leveraging these features is highly nontrivial. In addition to the absence of a general fusion strategy to achieve percolation for generic resource states, and the structural mismatch between the high-level program graph state and the low-level random physical graph state, we need to keep in mind the limited time for real-time passes. Specifically, the process associated with the formation of long-range connectivity and the reshaping of the random physical graph state both need to be carried out in real-time, leading to a high demand on their lightweight design. This creates a conflict between the real-time scalability and the program execution efficiency, as a flexible optimization strategy for efficient program execution may require complex algorithms that are not feasible in real-time.

To this end, we propose a randomness-aware compiler to efficiently scale up quantum computing on photonic systems, as illustrated in Fig. 4.2. Our framework achieves concurrent real-time scalability and program execution efficiency through the combination of an online

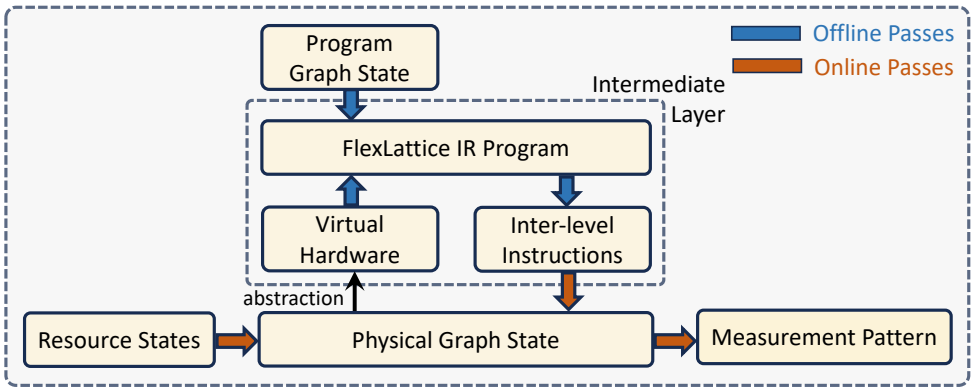


Figure 4.2. High-level design of OnePerc.

pass and an offline pass. The online pass handles real-time randomness in a scalable manner through percolation and reshaping. In particular, it provides a general fusion strategy for various resource states and exposes the reshaped physical graph states to programs by the abstraction of a virtual hardware. Motivated by the features of this virtual hardware, we propose a *FlexLattice* intermediate representation (IR), which preserves the high-level program information and provides maximal optimization space supported by the virtual hardware. This allows offline passes to address the mismatch between program and physical graph states by transforming the program to an efficient IR program, which can then be translated to intermediate-level instructions to guide real-time operations.

Our contributions in this paper are summarized as follows:

- We propose a randomness-aware compiler for photonic quantum computing through a combination of online and offline passes, which are bridged by a novel FlexLattice IR facilitated with an intermediate-level instruction set.
- The online pass handles real-time randomness in a scalable manner by formation of long-range connectivity with various resource states and efficient reshaping of the random long-range connected structures.
- The FlexLattice IR provides programs with an optimization space that balances the com-

plexity of online structure reshaping and the flexibility of the reshaped structures. This enables an offline pass to enhance the efficiency of program execution through optimized mapping algorithms.

- Our evaluation demonstrates a significant outperformance over the efficient baseline in a scalable manner, implying a first-time concurrent achievement of scalability and efficiency in compilation of photonic quantum computing.

4.2 Background

4.2.1 MBQC Basics

MBQC is a universal but conceptually distinct computational model from the circuit model. In MBQC, computation is driven by 1-qubit projective measurements, rather than 1-qubit and 2-qubit gates, on an initial entangled state called *graph state*, whose graph structure $G = (V, E)$ is determined by the quantum program [48]. As exemplified in 4.1(a), 4.3(b), 4.5(a), each vertex in the graph state stands for a qubit, with the state formally defined as the eigenstates of operator

$$s = X_i \bigotimes_{j \in n_i} Z_j, \quad \forall i \in V$$

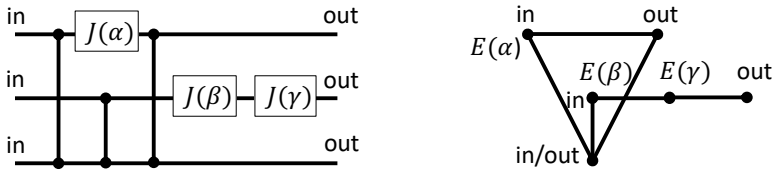
where X_i, Z_j are the Pauli operators on qubit i, j respectively, and n_i is the set of neighboring qubits of $i \in V$ on graph G. On the graph state, computation can be driven by a set of *equatorial measurements* $E(\alpha)$, i.e., measurements on the X-Y plane of Bloch sphere at an angle α, along with Z-measurements. The measurement basis of each qubit is predetermined by the quantum program, known as a *measurement pattern*, but are subject to a real-time adjustment according to the measurement outcomes of prior qubits, with the angles adjusted from α to $(-1)^s\alpha + t\pi$ where $s, t \in \{0, 1\}$. This feed-forward mechanism is used to address the non-determinism of quantum measurement outcomes.

MBQC has the same computation power with the circuit model in the sense that they are

both universal computing models. There is a straightforward translation [48] from a circuit in the universal gate set $\{J(\alpha), \mathrm{CZ}\}$ into a measurement pattern on a graph state, where

$$J(\alpha) = \begin{bmatrix} 1 & e^{i\alpha} \\ 1 & -e^{i\alpha} \end{bmatrix}$$

For example, Fig. 4.3 shows the translation from 4.3(a) to 4.3(b), each vertex in (b) representing a qubit, with 'in' and 'out' denoting their roles of being input or output qubits. It can be seen that the gates $J(\alpha)$, $J(\beta)$,$J(\gamma)$ are translated to equatorial measurements with corresponding angles, i.e., $E(\alpha)$, $E(\beta)$, $E(\gamma)$, while the CZ gates are translated to edges of the graph states. This process can be rigorously described in ZX-calculus and optimized by available tools such as PyZX [122].



(a) (b)

Figure 4.3. Translation from a circuit (a) to a measurement pattern on a graph state (b).

4.2.2 Photonic Platform

The weak interaction between photons, despite ensuring low cross-talk between photonic qubits, also poses significant challenges for realizing multi-qubit gates in the circuit model. Therefore, as a computing model that only requires measurements, MBQC [194] emerges as highly suitable for photonic quantum computing. Besides the experimental demonstration of small-scale photonic MBQC [216, 212, 202], the photonic platform is rapidly scaling up with integrated waveguides and optical chips [87, 220, 88, 188, 54, 39]

Practical photonic hardware scales up by creating small *resource states*, e.g., 4-qubit, 6-qubit graph states, and connecting them through *fusions* [180]. In particular, identical resource

states are periodically generated by an array of *resource state generators* (RSGs) every cycle, with those generated in the same RSG cycle forming a 2D resource state layer (RSL). Along with the time dimension, this creates a 3D array of resource states in the space-time.

The resource states can then be merged probabilistically into larger graph states through (type II [126]) fusions, which can be regarded as concurrent measurements of $X \otimes Z$ and $Z \otimes X$, on two photonic qubits from different resource states. Resource states on the same RSL can fuse with their neighbors by a spatial routing of photons, while resource states generated by the same RSG but on different RSLs can fuse with each other by a temporal routing that controls the arrival times of photons at measurement devices.

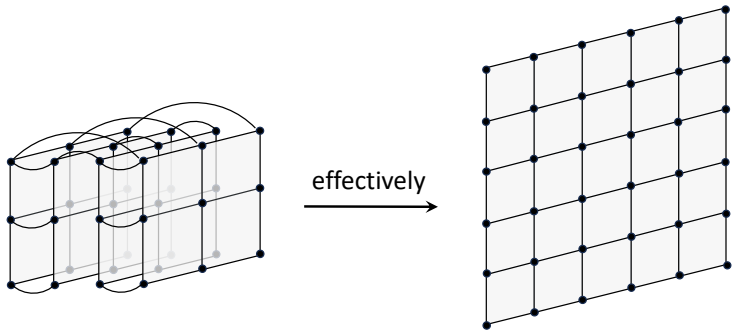


Figure 4.4. Form a large RSL from multiple small RSLs.

With the advanced integrated silicon photonics, hardware components described above can operate on the scales characterized by GHz clock rates [164, 81, 218], potentially leading to a time scale ~ 1 ns for RSG cycles [39]. Spatial routing can be adjusted in every RSG cycle with switches, while temporal routing can be achieved by temporarily storing photonic qubits in a high-capacity quantum memory known as *delay lines*, realized by optical fiber technology. With a low transmission loss rate of $< 5\%$ per km [139], photons can have a lifetime of around 5000 RSG cycles in the delay lines. Moreover, the size of RSL is not completely constrained by the number of RSGs, but can be extended by leveraging the tradeoff between spatial and temporal fusions [39]. For example, the large RSL depicted in Fig. 4.4(b) can be formed by fusing the edges of several small RSLs as depicted in Fig. 4.4(a), resembling folding a paper

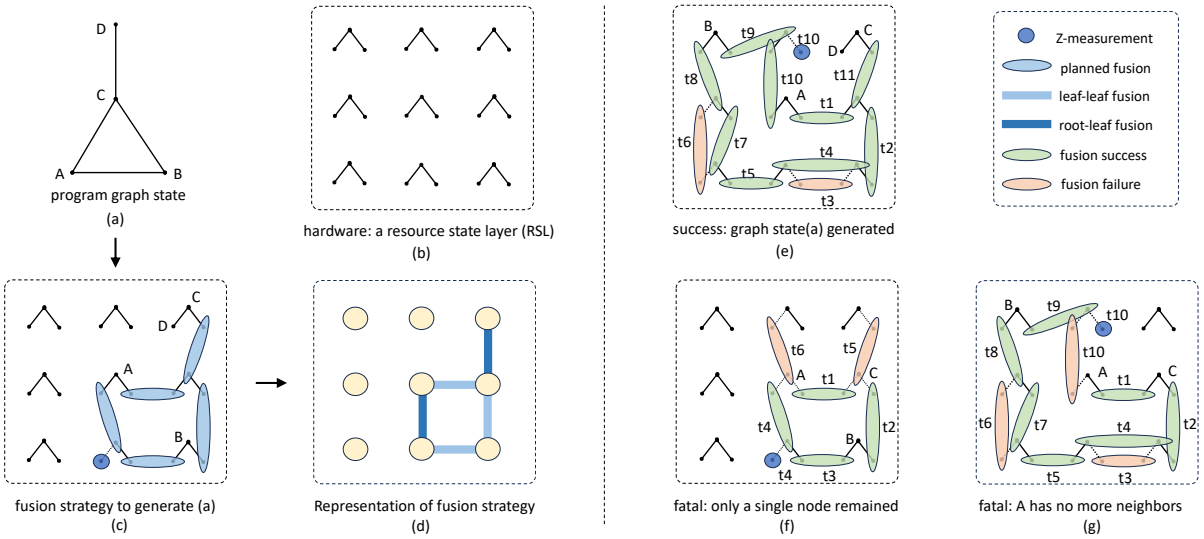


Figure 4.5. Why OneQ does not work.

by twice (4.4(b)→ 4.4(a)). With a photon lifetime around 5000 RSG cycles, this allows for an extension of RSL size by up to 5000 times.

However, as the key operation for merging resource states, fusions are intrinsically probabilistic. By allowing two ancilla photons, their success probability can be practically boosted to 75% [84, 99]. While no conceptual limit has been found yet, so far the maximum known success probability attainable using linear optics is 78% by injecting 8 ancilla single photons [84]. Reaching a higher success probability not only requires a larger number of ancilla photons but could also require the ancilla photons to be entangled. For example, it would take 30 entangled ancilla photons to reach a success probability over 95% [99].

4.3 Motivation and Overview

In this section, we present a motivating example to demonstrate that a straightforward adaption of OneQ is insufficient to yield a scalable compiler in the presence of fusion failures. Then we provide an overview of our innovative randomness-aware compiler designed to effectively overcome these challenges.

4.3.1 Motivating Example

OneQ + Retry

Fig. 4.5(c) illustrates OneQ's strategy with an example of generating a graph state in Fig. 4.5(a) from a single RSL depicted in Fig. 4.5(b), with the strategy represented more compactly by Fig. 4.5(d). Since the resource states have a star-like tree structure, we refer to the qubits of degree 1 as *leaf qubits*, and those of degree > 1 as *root qubits*. Light blue lines in Fig. 4.5(d) denote *leaf-leaf fusions* (i.e., fusions between two leaf qubits) of the resource states (yellow circles), while dark blue lines denote *root-leaf fusions* (i.e., fusions between a root and a leaf qubit).

To handle real-time randomness, a straightforward adaption is to introduce a retry mechanism. For example, the strategy in Fig. 4.5(c) can result in a dynamic implementation in Fig. 4.5(e) according to the fusion successes and failures (green and red ellipses), with these fusions performed sequentially from t1 to t11. If a fusion such as t3 fails, we retry the fusion using another two qubits at t4, and the same approach is applied to t6 and t7. This allows us to successfully generate the graph state in Fig. 4.5(a).

However, it is worth noting that some fatal failures may necessitate the retry of the entire compilation. For example, in Fig. 4.5(f), the triangular structure ABC is successfully generated from t1 to t4, but subsequent failures at t5 and t6 deplete the qubits in ABC, only leaving the isolated qubit B. In Fig. 4.5(g), a 5-qubit linear graph state forms from t1 to t9, which provides the potential for generating the triangle ABC if the fusion at t10 succeeds in fusing the two qubits at the line ends. Unfortunately, this fusion fails and consumes the last neighboring resource state of qubit A (except C), leaving A with no chance to fuse with other qubits.

Critical Issues

From this example, we can find some critical issues of this dynamic retry mechanism. First, adapting to prior fusion outcomes necessitates a sequential execution of fusions. This considerably extends the processing time for each RSL, resulting in a time inefficiency as

subsequent RSLs must wait for the completion of the current one. Second, since the decision-making process for responding to prior fusions occurs in real time, this extended processing time could exceed the limited lifetime of photons, especially for large RSLs. This would result in substantial photon loss, compromising the overall fidelity as computing scales up. Third, the frequent retries in real-time implementation lead to significant deviations from the planned strategy in Fig. 4.5(c). This undermines the benefits of the proactive planning, eroding the efficiency achieved by the mapping strategy of OneQ.

4.3.2 Framework Overview

Tolerating randomness in the compilation while maintaining efficiency presents a significant challenge. To address this, we propose an innovative framework that achieves scalability and efficiency simultaneously through a synergy of online and offline passes. The online pass prioritizes the real-time scalability by maximizing the concurrency among fusions and the parallelism of the associated path searching. The offline pass focuses on the efficient deployment of high-level program graph states onto the randomness-eliminated computing resource guaranteed by the online pass. The bridge between the online and offline passes is established through an intermediate software layer positioned between the low-level physical layer and the high-level program layer. This is achieved through a novel FlexLattice IR, along with an instruction set supported by the online pass and fulfilling the requirements of the offline pass.

To provide a concise overview, we exemplify the compilation flow by compiling a simple program graph state in Fig. 4.6(a) onto the hardware in Fig. 4.6(b), which is 3 layers of that in Fig. 4.5(b). Indeed, while Fig. 4.5(b) depicts only a single RSL, the incorporation of additional layers is both allowed and necessary for larger graph states. Steps (b)→(d)→(c) demonstrate the online pass, while step (a)→(c) illustrates the offline pass. In the online pass, fusions are conducted concurrently in a predetermined pattern (Fig. 4.6(d)) without individual retries of the failed ones, which eliminates the necessity for sequential operations. In this simple example, the resource states would result in a 3×3 lattice if all fusions succeed, since the 3 resource states on

the same locations of different layers would form a 4-degree star-like graph state (as depicted in the legend), while these star-like graph states would be joined into a lattice. In the presence of fusion failures, the resulting physical graph state becomes a subgraph of the 3×3 lattice, which is then reshaped to a smaller lattice (Fig. 4.5(c)). The target structure of the reshaping is program-agnostic, with its simple and regular structure facilitating the enhancement of real-time efficiency. When the fusion success probability exceeds the percolation threshold, this reshaping process attains near-deterministic success as the RSL size increases. This eliminates the necessity for repetitive retries of the entire compilation. With this near-determinism, the offline pass can be employed to improve the efficiency by mapping the program graph state compactly onto the reshaped lattice (bold blue lines in Fig. 4.5(c)).

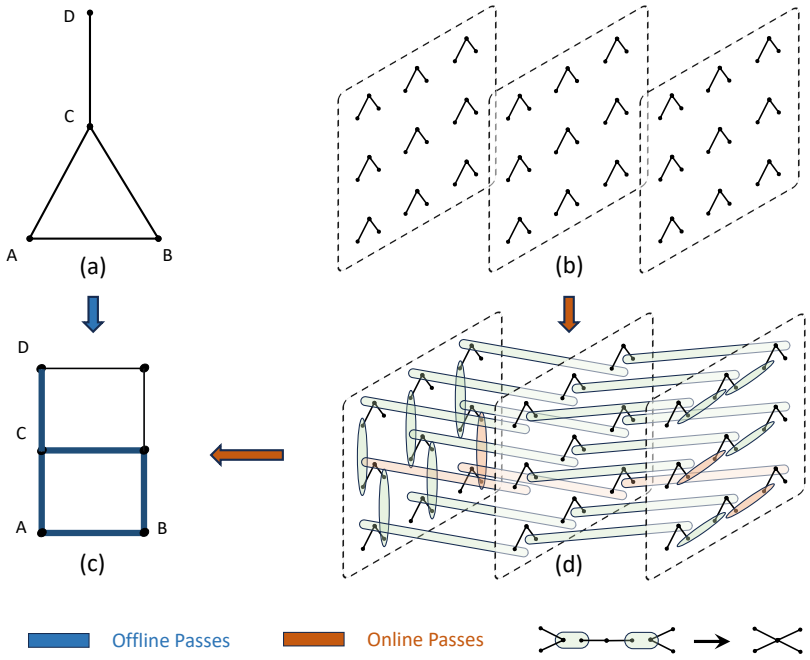


Figure 4.6. Overview of the compilation flow.

Note that the compilation of general programs can be considerably more intricate than the example presented here. First, the fusion strategy among resource states is more complex than Fig. 4.6(d). Specifically, it enables the formation of a 3D structure rather than 2D, being adaptable to various resource states and allowing collective retries with a small overhead. Second,

the complexity of the reshaping algorithm is carefully reduced to enhance its real-time scalability. This is achieved by a modular design on each RSL that improves the parallelism of path searching. Third, the reshaping process is heterogeneous in the spatial and temporal dimensions, with the temporal dimension supporting connections both between adjacent layers and non-adjacent layers. These flexible connections provides a larger optimization space for the offline mapping than Fig. 4.6(c). Forth, the online and offline passes are further bridged by posing a FlexLattice IR, which guides the low-level operations by its translation to an instruction set. For general programs, the compilation flow can be summarized as the following.

1. Before program execution, an offline pass transforms the program graph state to an efficient FlexLattice IR, which is then translated to intermediate-level instructions to guide real-time operations (Section 4.6).

2. During real-time execution, fusions between resource states are performed concurrently in a predetermined pattern, allowing collective retries of failed connections to improve the long-range-connectivity of the resulting physical graph state (Section 4.4).

3. The resulting physical graph state is then reshaped to a 3D structure that fulfills the requirement of the IR program, with measurements performed on qubits according to the IR program (Section 4.5).

The following sections (Section 4.4, 4.5, 4.6) will provide a bottom-up introduction to the framework.

4.4 Resource State Fusion

In this section, we discuss the semi-static fusion strategy for generic star-like resource states. This strategy is static in that it is predetermined independent of high-level programs, yet semi-static in that it allows collective retries which induces only a constant overhead. In Fig. 4.2, this corresponds to the online pass from resource states to physical graph states.

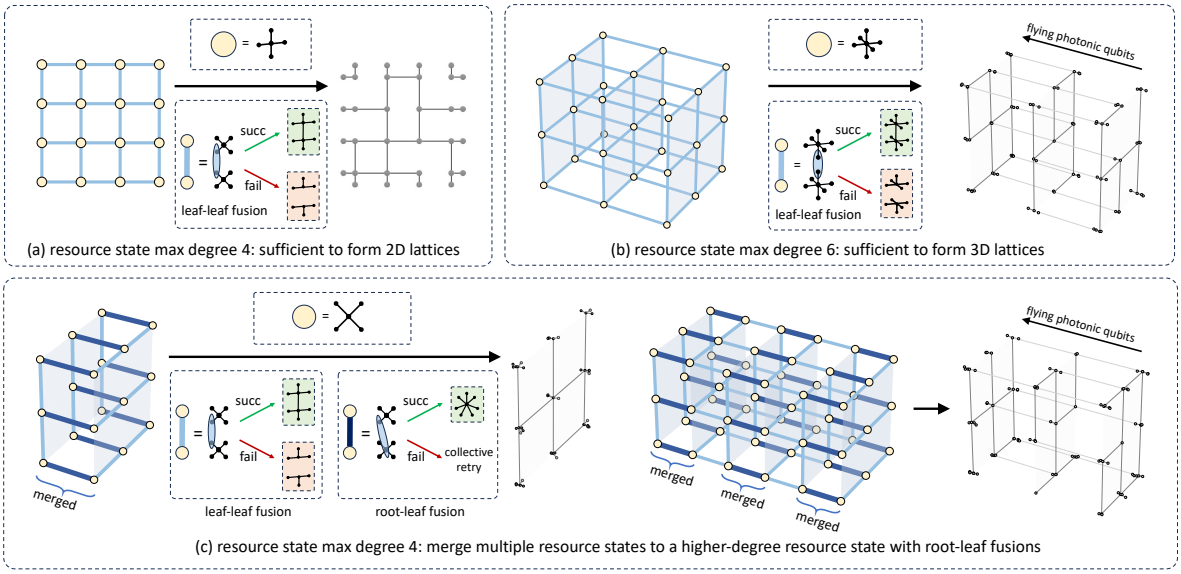


Figure 4.7. Fuse small resource states into 3D lattices.

4.4.1 Sufficient / Insufficient Degree

The predetermined strategy attempts to create a lattice structure from the resource states, which is straightforward when resource states have sufficient node degrees, i.e., the maximum degree in the resource states surpasses that in the lattice. For example, Fig. 4.7(a) shows the strategy of forming a 2D square lattice using 4-degree resource states. On the left side, each light blue line represents a leaf-leaf fusion of the resource states (yellow circles). The right side displays the resulting physical graph state, with the consequences of successful and failed fusions depicted in the middle box. Similarly, Fig. 4.7(b) demonstrates the case of forming a 3D cubic lattice from 6-degree resource states.

As resource states on realistic hardware may lack sufficient degrees, such as forming 3D cubic lattices using 4-degree resource states, we can increase resource state degrees by merging multiple RSLs into one layer using root-leaf fusions, represented by the dark blue lines in Fig. 4.7(c). Upon fusion success, while the two qubits in the fusion vanish, the two set of neighboring qubits of them would be connected in pairwise. Hence a successful root-leaf fusion between two 4-degree resource states can generate a 7-degree graph state, which then has sufficient degree to form a 3D lattice.

4.4.2 Removal of Irregular Structures

However, a failed root-leaf fusion may result in irregular cyclic structures in the generated graph state, leading to significant challenges for subsequent reshaping process. For example, a failed root-leaf fusion between two resource states A_0 and B_0 in Fig. 4.8 generates a star-like graph state A and a fully connected cyclic graph state B. This is because a failed fusion on a qubit v can be regarded as removing the qubit after a process of *local complementation* (LC) on v, denoted as $\tau_v(G)$. Specifically, LC is defined as: among all neighbors of qubit v in its resource state, if there was an edge between a pair of neighbors, then that edge is deleted; otherwise, an edge is added between that pair of neighbors.

To remove these cyclic structures, resource states with failed root-leaf fusions can be transformed to their local complementations of star-like structures (from B to C in Fig. 4.8) by applying the following sequence of 1-qubit operators [107]

$$U_v(G) = \exp(-\mathrm{i}\frac{\pi}{4}X_v)\prod_{u\in N_v}\exp(\mathrm{i}\frac{\pi}{4}Z_u)$$

with N_v being the neighbors of v in G. Computation on these local complementation states is equivalent to that on the original states. This is because we can interchange the orders of measurements and LC operators by adjusting the measurement bases, with the rules summarized in Theorem. 4.4.1. Similarly, the LC operators can also be interchanged with fusion operations, with the rules summarized in Theorem 4.4.2. Consequently, all LC operators can be postponed to the end of the computing process, which eliminates the necessity to implement them in the real-time.

Theorem 4.4.1. *The local operator* $U_Z^{\pm} = \exp(\pm\mathrm{i}\frac{\pi}{4}Z)$ *or* $U_X^{\pm} = \exp(\pm\mathrm{i}\frac{\pi}{4}X)$ *can be propagated through a* $Z-$*measurement or a 1-qubit equatorial measurement on the Bloch sphere, i.e., a measurement in the basis of* $\cos\phi X + \sin\phi Y$ *where* $\phi \in [0, 2\pi)$*, by a change of measurement basis.*

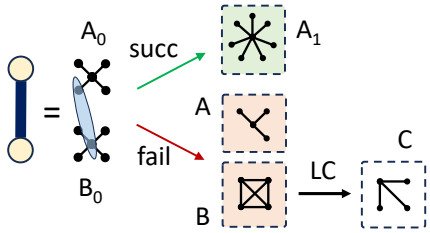


Figure 4.8. Root-leaf fusion failure.

Proof. When measuring a 1-qubit state $|\psi\rangle$ along A-basis, the state collapses to $|\psi'\rangle \equiv \mathrm{M}_{[A]}|\psi\rangle = \frac{\mathbb{I}\pm A}{2}|\psi\rangle$, with the sign $\pm$ determined by the measurement outcome, 0 or 1. Therefore,

$$\begin{aligned}
\mathrm{M}_Z U_Z^{\pm} &= U_Z^{\pm}\mathrm{M}_Z \\
\mathrm{M}_Z U_X^{\pm} &= U_X^{\pm}\mathrm{M}_{[\mp Y]} \\
\mathrm{M}_{[\cos\phi X+\sin\phi Y]} U_Z^{\pm} &= U_Z^{\pm}\mathrm{M}_{[\pm(\cos\phi Y-\sin\phi X)]} \\
\mathrm{M}_{[\cos\phi X+\sin\phi Y]} U_X^{\pm} &= U_X^{\pm}\mathrm{M}_{[\cos\phi X\pm\sin\phi Z]}
\end{aligned}$$

□

Theorem 4.4.2. *The local operator $U_Z^{\pm} = \exp(\pm \mathrm{i}\frac{\pi}{4}Z)$ or $U_X^{\pm} = \exp(\pm\mathrm{i}\frac{\pi}{4}X)$ can be propagated through a 2-qubit XZ,ZX fusion, i.e., a joint measurement of X_1Z_2, Z_1X_2 on qubit 1 and qubit 2, by a change of fusion basis.*

Proof. When measuring a 2-qubit state $|\psi\rangle$ along basis A_1B_2, the state collapses to $|\psi'\rangle \equiv \mathrm{M}_{[A_1B_2]}|\psi\rangle = \frac{\mathbb{I}\pm A_1B_2}{2}|\psi\rangle$, with $\pm$ determined by the measurement outcome, 0 or 1. Therefore

$$\begin{aligned}
\mathrm{M}_{[X_1Z_2]}\mathrm{M}_{[Z_1X_2]} U_{Z_1}^{\pm_1} U_{Z_2}^{\pm_2} &= U_{Z_1}^{\pm_1} U_{Z_2}^{\pm_2}\mathrm{M}_{[\pm_1 Y_1 Z_2]}\mathrm{M}_{[\pm_2 Z_1 Y_2]} \\
\mathrm{M}_{[X_1Z_2]}\mathrm{M}_{[Z_1X_2]} U_{X_1}^{\pm_1} U_{X_2}^{\pm_2} &= U_{X_1}^{\pm_1} U_{X_2}^{\pm_2}\mathrm{M}_{[\mp_1 Y_1 X_2]}\mathrm{M}_{[\mp_2 X_1 Y_2]} \\
\mathrm{M}_{[X_1Z_2]}\mathrm{M}_{[Z_1X_2]} U_{Z_1}^{\pm_1} U_{X_2}^{\pm_2} &= U_{Z_1}^{\pm_1} U_{X_2}^{\pm_2}\mathrm{M}_{[\pm_1\mp_2 Y_1 Y_2]}\mathrm{M}_{[Z_1X_2]}
\end{aligned}$$

□

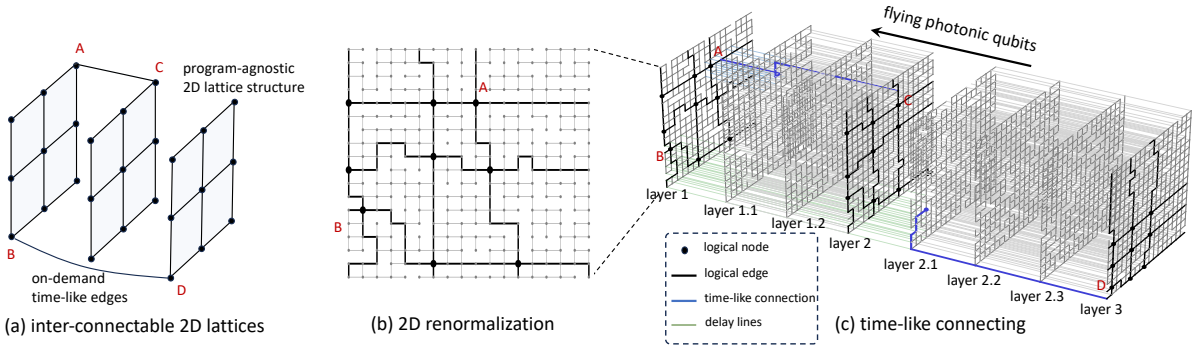


Figure 4.9. (2+1)-D reshaping for handling random graph states generated by fusions.

4.4.3 Collective Feed-forward

The semi-static fusion strategy allows collective feed-forward in the granularity of RSL, which can be pipelined to reduce the overhead. On one hand, the propagation of LC operators through measurements and fusions requires an adaptive adjustment of measurement and fusion bases. With this dependency, each RSL are fused in two batches: a batch of root-leaf fusions and a batch of leaf-leaf fusions. On the other hand, the connectivity of the physical graph state can be enhanced by retries of the failed connections, including retrying failed leaf-leaf fusions with redundant degrees (e.g., with the 7^{th} degree of A_1 in Fig. 4.8) and retrying failed root-leaf fusions with remaining degrees (e.g., with A and C in Fig. 4.8). In this way, each RSL may undergo more batches of fusions. However, since earlier batches of later RSLs can be conducted concurrently with later batches of earlier RSLs, this only introduces a constant overhead to program execution.

4.5 Random State Reshaping

In this section, we delve into the reshaping of physical graph states, which is characterized by a (2+1)-D design, motivated by the continuous generation of RSLs over time and the presence of delay lines. In Fig. 4.2, this corresponds to the online pass from physical graph states to measurement patterns.

4.5.1 Efficient 2D Renormalization

On each (merged) RSL, we apply a process known as renormalization [49], which reshapes the largest connected component of the physical graph state to a coarse-grained 2D lattice. The key to its viability lies in the percolation phenomenon [97, 167, 49]. That is, when the fusion success probability exceeds a certain threshold, the random physical graph state undergoes a phase transition from short-range connectivity to long-range connectivity, leading to the largest connected component reaching a comparable size with the original graph state. Since fusions on each RSL are constrained as a squared lattice, the percolation threshold is only 0.5 [118], lower than the achievable fusion success probability.

Identifying intersections of horizontal and vertical paths in the largest connected component reveals a coarse-grained square lattice, represented by bold nodes and edges in Fig. 4.9(b). This is achieved in the following way. We search for vertical paths from left to right and horizontal paths from bottom to top, enforcing distinct vertical or horizontal paths to maintain a separation of at least one qubit. When searching for vertical (horizontal) paths, a connectivity check is conducted between nodes at the top (left) and bottom (right), facilitated by a disjoint-set data structure to reduce the complexity. Upon confirming connectivity, a breadth-first search (BFS) is applied to determine the shortest path, ensuring it remains free of self-tangling. To further prevent tangling between vertical and horizontal paths, we remove the surrounding qubits of each identified path after discovery, preventing their interference with subsequent searches. Considering the removals, an alternating search of vertical and horizontal paths emerges as an effective searching order.

To improve real-time scalability, the 2D renormalization is designed to allow **modularity**, with areas on the RSL renormalized concurrently and then joined together. As shown in Fig. 4.10, the RSL is divided into several modules of size $L_{Module} \times L_{Module}$, with some intervals of length $L_{interval}$ left in between for joining the modules by connected paths. With the path searching algorithm above, the complexity of a modular 2D renormalization is $O(L^2_{\text{module}}) \sim O(N^2/m)$,

where m is the number of modules. Since an entire path can only be established if all inter-module paths involved are successful (e.g., the orange path), the potential for failed inter-module paths could lead to the renormalized lattice size being smaller than the total size of all individual modules. A suitable ratio of L_{Module} and $L_{Interval}$, defined as *MI ratio*, can help mitigate this resource overhead.

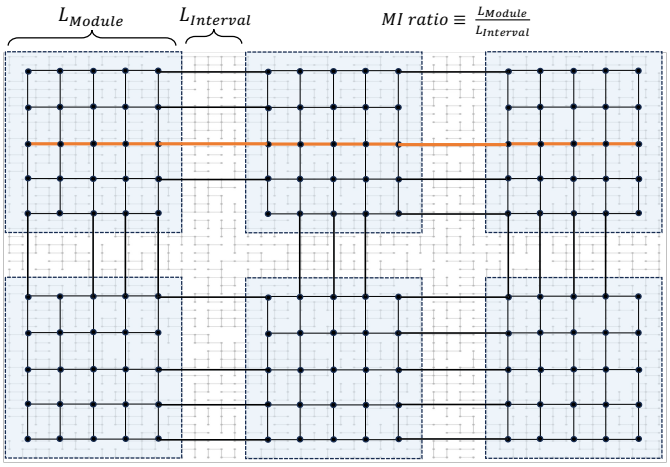


Figure 4.10. Modular renormalization.

4.5.2 Flexible Time-like Connections

Nodes on the renormalized 2D lattices can be connected along the time dimension, referred to as *time-like connections*. Connections between adjacent 2D lattices and across non-adjacent 2D lattices are called *adjacent-layer* connections and *cross-layer* connections, respectively. Before program execution, the connections to establish are given by the IR program as a 3D graph, as illustrated in Fig. 4.9(a).

The process of generating this 3D graph is illustrated in Fig. 4.9(c) on 8 RSLs, with the adjacent-layer connection *AC* and the cross-layer connection *BD* implemented through the bold blue paths. This involves an attempt of 2D renormalization on each RSL. The successful ones then serve as *logical layers*, indexed by integers in Fig. 4.9(c), with the renormalized nodes on them referred to as *logical nodes*. In contrast, the RSLs with failed renormalization serve as

routing layers, which are indexed by decimals in Fig. 4.9(c). The renormalization on an RSL is considered successful if:

1. The renormalized 2D lattice reaches a target size, which is equivalent to a choice of average node size, where

$$\text{average\_node\_size} \equiv \frac{\text{RSL\_size}}{\text{renormalized\_lattice\_size}}$$

2. The RSL can establish all necessary time-like connections with prior logical layers through the following procedure.

To establish a time-like connection between two nodes, a set of physical qubits around the preceding node are fused with corresponding qubits on a correct subsequent RSL. For adjacent-layer connections such as *AC*, the qubits around *A* are directly fused to the next RSL, which is layer 1.1 in Fig. 4.9(c). For cross-layer connections such as *BD*, the qubits around *B* are temporarily stored in delay lines, depicted by the green thin lines in Fig. 4.9(c), until they can be fused to layer 2.1, which is the first RSL between the current attempting RSL (layer 3) and its prior logical layer (layer 2). Subsequently, a path searching between the two nodes is conducted within the physical graph state, exemplified by the bold blue lines *AC* and *BD* in Fig. 4.9(c). Again, this is achieved by a connectivity check utilizing a disjoint-set data structure and a BFS for the shortest path. If the connectivity check yields a negative result, it indicates that the current RSL fails to meet the second condition and would become a routing layer. It is worth mentioning that the reshaping process can tolerate photon loss, since a fusion is considered as successful only if both two photons are detected. Effectively, the presence of photon loss causes a reduction of the fusion success probability, possibly leading to more routing layers between logical layers.

In contrast to the logical layers, all qubits of each routing layer are directly fused with their next RSL, as depicted by the grey thin lines in Fig. 4.9(c). This is because before obtaining the next successful renormalization, we can't predict where the logical node would locate and

which fusions around it would succeed. Moreover, in contrast to the simple case in Fig. 4.9 where there is only one connection between layer 1 and layer 2, in practice we may need to establish multiple connections between logical layers. This makes it even harder to predict which fusions are redundant before executing the fusions.

4.6 Offline Optimization with IR

In this section, we introduce the virtual hardware, FlexLattice IR, offline mapping and intermediate-level instructions. This covers the offline pass in the compilation flow (Fig. 4.2).

Before program execution, the mismatch between program graph states and physical graph states can be addressed by mapping the program graph state onto the virtual hardware, leading to an IR graph state that maintains the high-level program information. This IR program can then be transformed to a set of intermediate-level instructions, which guides real-time physical operations through the reshaping algorithm above.

4.6.1 Virtual Hardware

The virtual hardware abstracts the adjustable structures supported by the reshaping algorithm. It pocesses a (2+1)-D structure, characterized by the following features, as illustrated in Fig. 4.11(b).

1. The virtual hardware consists of consecutive layers of 2D lattices in a fixed size, with a virtual memory located on each 2D coordinate.

2. Nodes on the same 2D coordinate of different layers, either on adjacent or non-adjacent layers, can be connected along the third dimension, with the connections between non-adjacent layers realized by temporary storage of nodes in the virtual memory.

3. Each connection within or between 2D layers can be enabled or disabled on demand, but each node can have at most one connection with preceding layers and at most one connection with subsequent layers.

While this virtual hardware can be used to generate 3D cluster states (i.e., lattice-like graph states), which serve as the universal computing resource of MBQC in previous work [176], it is more advantageous in the following aspects. First, an individual connection can be flexibly enabled or disabled without removing any logical node or affecting other edges. This is in contrast to the cluster state, wherein the removal of edges is usually achieved by removing involved vertices and all their edges. Second, the connections among 2D layers exhibit greater flexibility than cluster states. Specifically, inter-layer connections between nodes on the same 2D coordinates extend beyond adjacent layers, encompassing cross-layer connections as well.

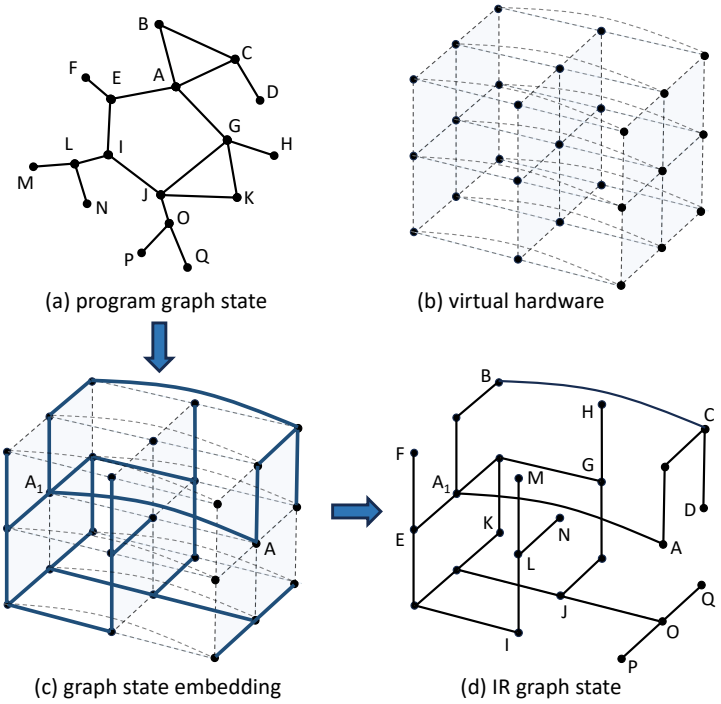


Figure 4.11. Offline mapping onto the virtual hardware.

4.6.2 FlexLattice IR and Offline Mapping

With this virtual hardware, graph state mapping algorithms such as that in OneQ can be utilized as an offline pass to enhance the efficiency of program execution. Specifically, the mapping onto virtual hardware transforms a program graph state to an equivalent IR program

with compatible structure with the virtual hardware, which is referred to as a *FlexLattice* IR based on its structural features. This process is illustrated by Fig. 4.11(a) $\rightarrow$ (c) $\rightarrow$ (d).

To further improve the mapping efficiency and scale to larger programs, we extend OneQ's mapping algorithm with three optimizations.

- First, to map graph nodes as early as possible, we replace the static partition in OneQ with a dynamic scheduling. Specifically, we analyze the dependency among graph state qubits [71], representing it with a directed acyclic graph (DAG) and updating the front layer of the DAG as nodes are consumed by the mapping.

- Second, to reserve enough space for routing and avoid node congestion, we enforce an upper-limit for the occupancy of incomplete nodes on each virtual hardware layer (25% by default), with incomplete nodes defined as those mapped nodes whose edges are not all mapped yet.

- Third, to mitigate the increasing demand on classical memory for graph information storage, we propose a refresh mechanism, which periodically retrieves all nodes stored in the virtual memory, refreshing them by mapping onto multiple layers of the virtual hardware, and then storing them again.

4.6.3 Instruction Set

A FlexLattice IR program can be executed by transforming to a set of intermediate-level instructions, which guides the real-time physical operations to generate necessary connections among logical nodes through the reshaping algorithm in Section 4.5. By default, qubits in the physical graph state are subject to Z-measurements, which means that edges are disabled on the virtual hardware unless explicitly enabled by the intermediate-level instructions. We list the intermediate-level instructions as the following, with nodes in the high-level program graph state denoted as `g_node` and nodes on the virtual hardware denoted as `v_node`.

```
map_v_node(v_node, g_node)
make_v_node_ancilla(v_node)
store_v_node(v_node)
retrieve_v_node(v_node, position)
enable_spatial_v_edge(v_node, adjacent_v_node)
enable_temporal_v_edge(v_node, adjacent_v_node)
```

By `map_v_node()` and `make_v_node_ancilla()`, a virtual node can be mapped by a `g_node` or used as an ancilla node to facilitate routing. In the former case, the physical qubit corresponding to `v_node` will be measured in the basis of the `g_node`, while in the latter case, it will be measured in X- or Y-basis to play as a wire (depending on whether the wire length is even or odd). By `store_v_node()` and `retrieve_v_node()`, a virtual node can also be stored into or retrieved from the virtual memory by pushing or poping its surrounding physical qubits to or from the delay lines. By `enable_spatial_v_edge()`, a spatial edge between adjacent nodes on the same layer can be enabled by setting associated qubits to X- or Y-measurements.

By `enable_temporal_v_edge()`, a temporal edge between logical nodes at the same coordinate of adjacent layers can be enabled. Establishment of a cross-layer edge between layer m and layer n $(> m)$ can be realized through the combination of three instructions: storing the node at layer m into the virtual memory, retrieving it at layer $n-1$, and enabling a temporal edge between layer $n-1$ and layer n. For example, the cross-layer temporal edge between ancilla node A_1 at (1,1,0) and graph node A at (1,1,2) in Fig. 4.11(d) can be implemented with the instructions below. Note that retrieving `v_node` at layer $n-1$ does not conflict with the original `v_node` at layer $n-1$ (i.e., node N in Fig. 4.11(d)). This is because the original node at layer $n-1$ would not have an edge with layer n, since each node in a FlexLattice IR has at most one

edge with preceding layers. This implies that the original node will either have no further edges or will be stored in the virtual memory at layer $n-1$.

```
make_v_node_ancilla((1, 1, 0))
store_v_node((1, 1, 0))
...
retrieve_v_node((1, 1, 0), (1, 1, 1))
enable_temporal_v_edge((1, 1, 1), (1, 1, 2))
map_v_node((1, 1, 2), A)
```

4.7 Evaluation

4.7.1 Experiment Setup

Baseline

We compare the performance of our framework with the efficient photonic MBQC compiler OneQ. Since OneQ is not able to handle fusion failures, we employ it with a repeat-until-success strategy. Specifically, for each RSL we conduct the fusions instructed by OneQ repeatedly until all fusions are successful. Subsequently, the successful RSL is fused with its preceding RSLs. If failures occur in the inter-RSL fusions, the entire compilation is restarted and repeated until success.

Metrics

Aligning with OneQ, we evaluate the performance of compilation with two metrics: the number of consumed RSL, denoted by #RSL, and the number of required fusions, denoted by #fusion. In particular, a smaller #RSL indicates less execution time of the program and less chance for photon loss, while a smaller #fusion implies less operations and less chance for error occurrence.

Table 4.1. Benchmark Programs.

Fusion Success Rate	#Qubits	Virtual Hardware Size	RSL Size
0.90	4	2x2	24x24
	9	3x3	36x36
	25	5x5	60x60
0.75	4	2x2	48x48
	25	5x5	120x120
	64	8x8	192x192
	100	10x10	240x240

Photonic Hardware Model

We adopt the same photonic hardware architecture with OneQ, as introduced in Section 4.2. In the main experiment (Table 4.2, 4.3), the comparison with OneQ is performed on 4-qubit star-like resource states, with the sizes of hardware for different benchmarks listed in Table 4.1. Experiments for further analysis are conducted with 7-qubit star-like resource states, which naturally have sufficient degrees for forming 3D lattice-like graph states.

Benchmark Programs

We select a set of benckmark programs including Quantum Approximate Optimization Algorithm (QAOA), Quantum Fourier transform (QFT), Ripple-Carry Adder (RCA) [64] and Variational Quantum Eigensolver (VQE). For QAOA, we choose the graph maxcut problem on randomly generated graphs. Specifically, the graphs are generated by randomly connecting half of all its possible edges. For VQE, we follow the commonly used full-entanglement ansatz, which proves to be an expressive ansatz [8, 232]. In table 4.1, we list the benchmarks with their numbers of qubits in the circuit representation. We also list the sizes of virtual hardware layers, which are chosen to correspond with the qubit quantities, along with the required sizes of RSLs needed to generate them, which are determined through Fig. 4.16, as explained later.

4.7.2 Experiment Result

In this subsection, we first show the performance of our compiler in comparison with OneQ, then analyze the effects of underlying resource states, hardware size and fusion success probability, for which we only focus on the #RSL metric. This is because unlike OneQ, the #fusion in OnePerc is predictable from its #RSL, thus following a same trend with #RSL.

Table 4.2. The results of OnePerc and its relative performance to the baseline.

Fusion Success Rate	Benchmark Name	OneQ #RSL	OnePerc #RSL	#RSL Improv.	OneQ #Fusion	OnePerc #Fusion	#Fusion Improv.
0.90 (hyper-advanced)	QAOA-4	304	84	3.62	13,990	117,664	0.12
	QFT-4	3,759	174	21.59	180,634	274,155	0.66
	RCA-4	3,107	237	13.11	63,814	373,646	0.17
	VQE-4	56	22	2.55	1,707	33,526	0.05
	QAOA-9	$> 10^6$	240	$> 10^3$	$> 10^{10}$	855,354	$> 10^4$
	QFT-9		570	$> 10^3$		2,031,813	$> 10^3$
	RCA-9		1,017	$> 10^2$		3,627,950	$> 10^3$
	VQE-9		156	$> 10^3$		555,065	$> 10^4$
	QAOA-25		768	$> 10^3$		7,637,711	$> 10^3$
	QFT-25		2,418	$> 10^2$		24,065,102	$> 10^2$
	RCA-25		3,111	$> 10^2$		30,962,172	$> 10^2$
	VQE-25		705	$> 10^3$		7,010,656	$> 10^3$
0.75 (practical)	QAOA-4	1,708	48	35.58	119,731	169,431	0.71
	QFT-4	$> 10^6$	210	$> 10^3$	$> 10^{10}$	746,977	$> 10^4$
	RCA-4	$> 10^6$	201	$> 10^3$	$> 10^{10}$	714,835	$> 10^4$
	VQE-4	1,017	23	44.22	25,354	96,332	0.26
	QAOA-25	$> 10^6$	882	$> 10^3$	$> 10^{10}$	19,743,350	> 10
	QFT-25		2,271	$> 10^2$		50,835,771	
	RCA-25		3,252	$> 10^2$		72,795,212	
	VQE-25		759	$> 10^3$		17,292,345	
	QAOA-64		3,339	$> 10^2$		191,341,276	
	QFT-64		9,000			515,801,985	
	RCA-64		9,324			534,311,489	
	VQE-64		3,042			174,321,702	

Performance

Table 4.2 presents the comparison of our framework with OneQ. The results indicate a significant reduction of #RSL by our framework, as well as a significant reduction of #fusion when the circuits are beyond 4 qubits. Specifically, the experiments show that OneQ can work only in the region of small programs and high fusion success probabilities. When the fusion success probability decreases to a practical value around 0.75, it takes more than 10^6 RSLs to

Table 4.3. Effect of refresh on the performance of OnePerc, considering 4-qubit resource state, a fusion success rate of 0.75, refresh rate of 50 logical layers, and 32GB of RAM.

Benchmark	#Qubits	Non-refreshed #RSL	Refreshed #RSL
QAOA	25	882	999
	64	-	4,284
	100	-	8,325
QFT	25	2,271	2,637
	64	-	9,945
	100	-	19,494
RCA	25	3,252	3,870
	64	-	10,206
	100	-	16,056
VQE	25	759	774
	64	-	3,555
	100	-	7,551

even execute the 4-qubit benckmarks. This implies OneQ's non-scalability due to its lack of capability in systematically handling the randomness of fusion failure. In contrast, our framework can work well with a practical success probability, demonstrating an increasing outperformance over OneQ as the programs scale up.

An obstacle of scaling up the experiments in Table 4.2 is the large classical memory required in the real-time stage for the storage of graph information. Indeed, the 64-qubit benchmarks in Table 4.2 takes a RAM as much as 192 GB. This can be overcome by the refresh mechanism proposed in Section 4.6, with an overhead of increased #RSL. Under the practical fusion success rate of 0.75, Table 4.3 shows the effect of refresh given 32 GB RAM. It can be seen that while the 32 GB RAM can only afford 25-qubit benchmarks without refresh, it allows for benchmarks of up to 100 qubits with a refresh every 50 logical layers. Compared with the performance of 25-qubit benchmarks (Table 4.2 or 4.3) and 64-qubit benchmarks (Table 4.2) without refresh, the introduction of refresh leads to an average increase of 15.6% in #RSL for 25-qubit benchmarks and an average increase of 13.3% in #RSL for 64-qubit benchmarks.

4.7.3 Sensitivity Analysis

Resource State Size

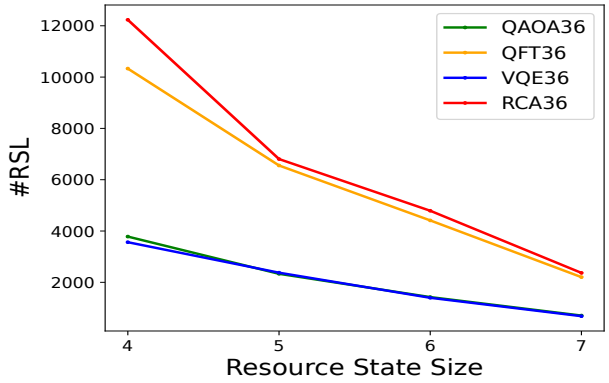
Our compiler has a general applicability to the underlying resource states of various sizes. Fig. 4.12(a) illustrates the varying #RSL when executing the programs with star-like resource states of different sizes, i.e., consisting of different numbers of photonic qubits. It can be seen that the #RSL decreases as the size of resource states increases. This is because a larger resource state can participate in fusions with more qubit degrees, without the need of increasing the degrees by merging multiple RSLs.

Hardware Size

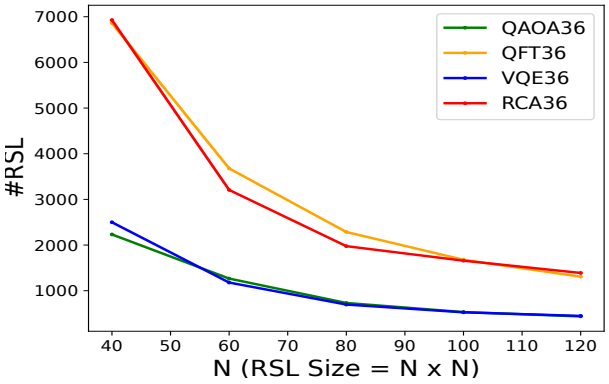
Our compiler has an adaptability to various hardware sizes. Fig. 4.12(b) shows the varying #RSL when executing the programs on photonic hardware of different RSL sizes. It can be seen that a larger photonic hardware leads to a reduced #RSL, which indicates that our framework can effectively utilize the computing resource as it scales up. In particular, a larger RSL can enable a larger renormalized lattice, thus a larger virtual hardware. This provides the offline mapping with an increased space for flexible routing, thereby reducing the required logical layer and the #RSL.

Fusion Success Probability

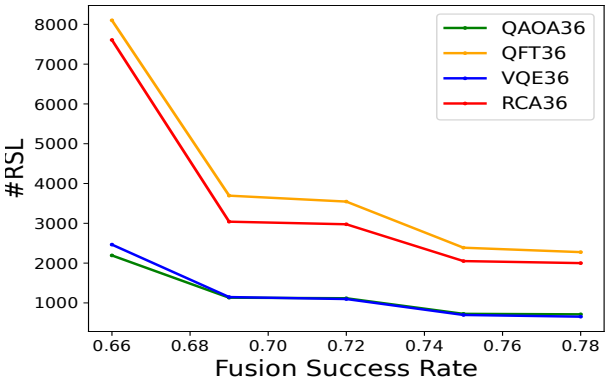
Our compiler has a capability of tolerating fusion failures at a practical level. Fig. 4.12(c) shows the varying #RSL when executing the programs under different fusion success probabilities. It can be seen that our compiler can tolerate a fusion success probability as low as 0.66, with the #RSL decreasing as the fusion success probability increases. This is because a higher fusion success probability results in a larger renormalized lattice on RSLs, enabling a larger virtual hardware. This provides the offline mapping with an increased space for flexible routing, thereby reducing the required logical layer and the #RSL.



(a)



(b)



(c)

Figure 4.12. Effects of resource state size (a), hardware size (b) and fusion success probability (c), with the resource states being 7-qubit ones for (b)(c), hardware size being 84x84 for (a)(c), and the fusion success probability being 0.75 for (a)(b).

4.7.4 Scalability

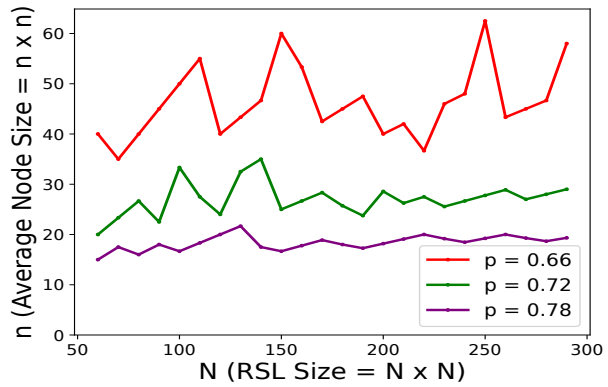
Resource Consumption

Our compiler presents a great scalability in resource consumption, characterized by the stable overhead as the computing scales up. Fig. 4.13(a) shows the suitable average node size of 2D renormalization as the hardware size increases, corresponding to the average node size at which the renormalization success probability approaches 1 in Fig. 4.16. As can be seen, it keeps stable against the variation of hardware size, being smaller with a higher fusion success probability. Fig. 4.13(b) shows the average ratio of RSL to logical layers as the program size increases. It first increases with the program size and then soon gets stable at a value around 3, implying the successful formation of a logical layer about every 3 RSLs. These stable behaviours provide a predictability of the resource consumption and ensures the scalability of our framework.

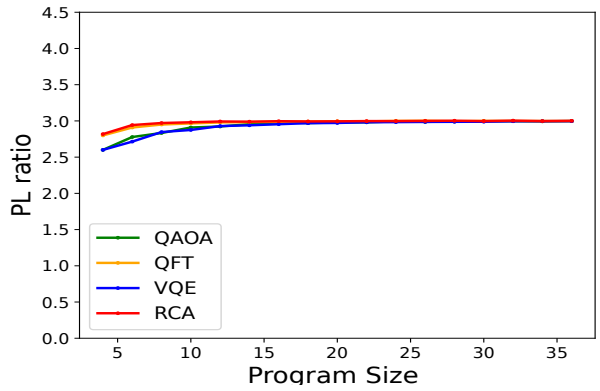
Modularity Overhead

The real-time scalability of our framework can be greatly enhanced through a modular 2D renormalization, which reduces the latency for each RSL by a factor corresponding to the modular number. However, this comes with an overhead, as the presence of intervals between the modules (as illustrated in Fig. 4.10) reduces the available resource on each RSL. To evaluate this resource overhead, Fig. 4.13(c) depicts the size of the renormalized 2D lattice against the number of modules, with the MI ratio (as defined in Fig. 4.10) ranging from 2 to 19. For comparison, the red dots represent the renormalized 2D lattice size by a non-modular algorithm in an unlimited time, while the black dots represent the renormalized size by the non-modular algorithm in a time restricted by that consumed by the modular approach.

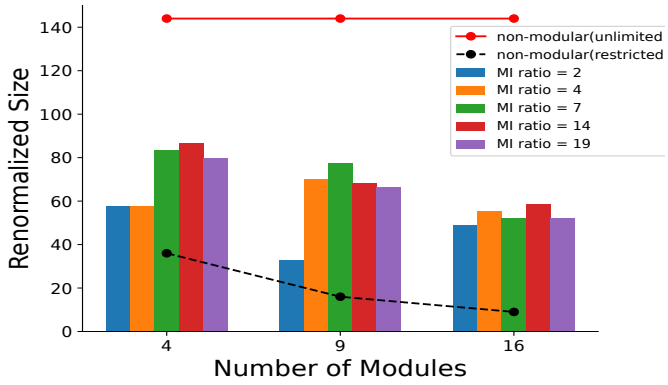
It can be seen that the size of renormalized 2D lattice by the modular approach is around 60% of that by the non-modular approach with unlimited time (red), which decreases slightly with the number of modules. This is because an increased number of modules leads to a higher probability of being unable to connect the corresponding paths across different modules. However, the renormalized lattice is significantly larger than that can be achieved by the non-



(a)



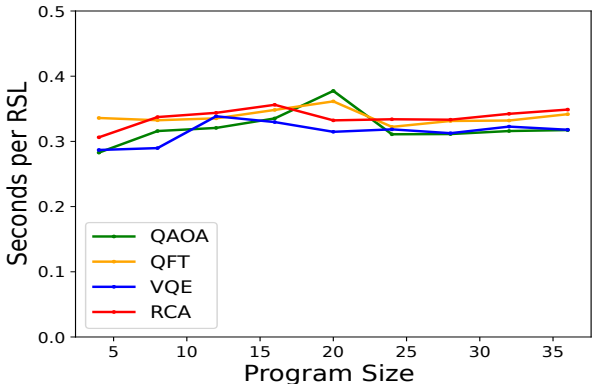
(b)



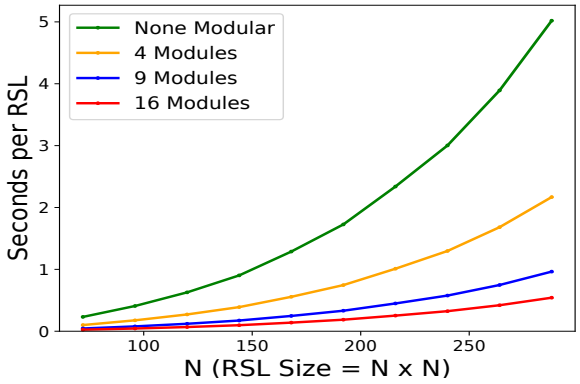
(c)

Figure 4.13. Scalability and parallelism of OnePerc with 7-qubit resource states. The node size $n \times n$ in (a) corresponds to the smallest node size where the renormalization success rate approaches to 1 in Fig. 4.16.

modular approach restricted in the same time (black), ranging from 2× to 6× as the number of modules increases from 4 to 16. This is very important since the time for the online algorithm is always restricted by the limited lifetime of photons. Overall, Fig. 4.13(c) indicates that the modular approach in our framework can significantly improve the real-time scalability with a reasonable overhead of computing resource.



(a)

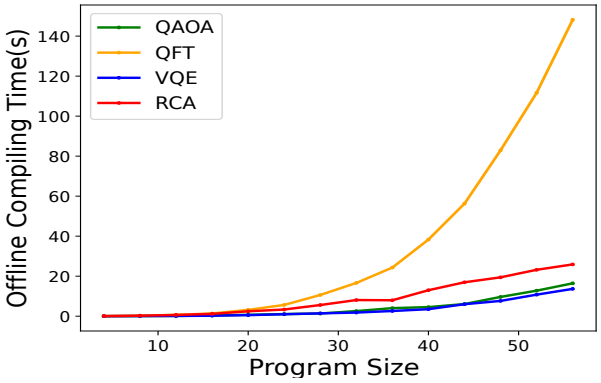


(b)

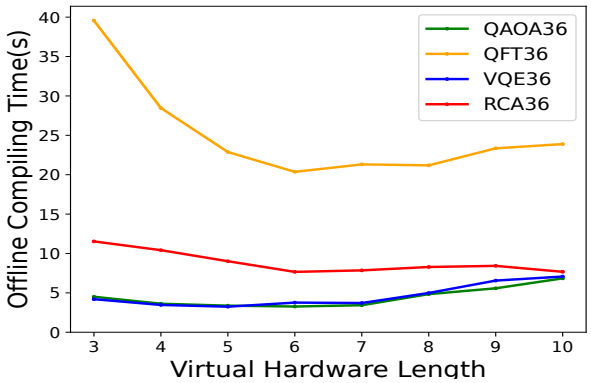
Figure 4.14. Oneline proccessing time for each RSL with 7-qubit resource states. RSL size is 96×96 for (a); fusion success rate is 0.75 for (a)(b); average node size is chosen as 24×24 for (a)(b); MI ratio is chosen as 7 for (b).

Compilation Time

We show the online and offline compilation time of the benchmarks in Fig. 4.14 and Fig. 4.15, with the compiler implemented in Python. From Fig. 4.14(a) it can be seen that the



(a)



(b)

Figure 4.15. Offline compilation time on a virtual hardware, with the virtual hardware size being 4×4 for (a). The virtual hardware sizes correspond to the sizes that can be formed by the RSL settings in Fig. 4.14.

online processing time for each RSL stays stable as the program size increases. From Fig. 4.14(b), which takes an average of all 36-qubit benchmarks, it can be seen that the processing time for each RSL increases with RSL size, but can be significantly reduced by employing a modular renormalization. For offline compilation time, Fig. 4.15(a) shows that it increases with the program size. Fig. 4.15(b) shows that it decreases with the virtual hardware size first and then increases. This occurs because an excessively small virtual hardware size leads to a significant total depth, whereas an overly large virtual hardware size results in extended compilation times for each logical layer.

4.7.5 Hyper Parameters

MI Ratio

The sizes of renormalized lattices rely on a suitable choice of MI ratio (defined in Fig. 4.10). Fig. 4.13(c) illustrates the renormalized lattice size with different choices of MI ratios. It can be seen that the renormalization size first increases with the MI ratio and then slightly decreases, peaking at a value around 7. This is because an excessively low MI ratio leads to a waste of resource with its wide interval space, while an overly high MI ratio increases the probability of unable to connect corresponding paths with its restricted routing space in the intervals.

Average Node Size

A suitable choice of average node size is also important, as it determines the target size of a successful 2D renormalization. Fig.4.16 illustrates the success probability of reaching different predetermined lattice sizes, i.e., different choices of average node size. It can be seen that the success probability approaches 1 rapidly as the target lattice becomes more coarse-grained. This sharp transition motivates us to choose the smallest average node size that brings the success probability close to 1.

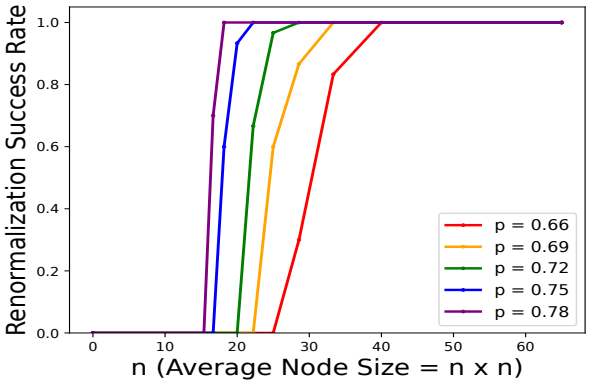


Figure 4.16. Effect of choices of average node size, with RSL size being 200×200.

4.8 Conclusion

In this work, we provide in-depth analysis and discussion of the challenges for photonic quantum compilation brought by the probabilistic operations involved in the computing. We propose a randomness-aware compiler to handle these probabilistic operations, demonstrating a concurrent achievement of scalability and efficiency on photonic systems. Nevertheless, we believe that there is still significant potential for fully exploring the optimization space. We hope that our work could attract more effort from the computer architecture and compiler community to explore the advantages of photonic quantum computing and overcome the unique challenges.

4.9 Acknowledgements

This chapter is a full reprint of material published as: Hezi Zhang, Jixuan Ruan, Hassan Shapourian, Ramana Rao Kompella, and Yufei Ding, "OnePerc: A Randomness-aware Compiler for Photonic Quantum Computing" published in Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. The dissertation author is the primary investigator and author of this paper.

Chapter 5

OneAdapt: Resource-Adaptive Compilation of Measurement-Based Quantum Computing for Photonic Hardware

5.1 Introduction

Photonic systems are a promising platform for universal quantum computing due to their unique advantages [162, 37], including great scalability, long coherence time and easy integration with quantum networks. In addition to experimental demonstrations of quantum supremacy on photonic systems [240, 239, 145], significant manufacturing progress has been demonstrated recently. This includes the achievement of 99.98% fidelity for state preparation and measurement [6], 99.22% fidelity for fusion [6] (not to be confused with fusion success rate [84, 99]), and the demonstration of small-scale error correction codes [4].

In contrast to other quantum computing platforms such as superconducting [174, 117, 219, 124], trapped ion [171, 179, 161] and neutral atom systems [227, 82, 36], photonic platforms are well suited for a different computing model called measurement-based quantum computing (MBQC) or one-way quantum computing (1WQC) [176]. In this model, computation is driven by single-qubit measurements on an entangled state. A quantum program needs to be transformed into a measurement pattern on this entangled state, while physical hardware is responsible for generating this entangled state by merging small resource states, such as through fusions [180]

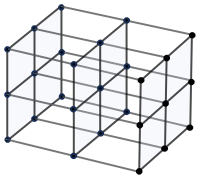
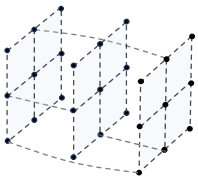
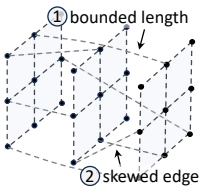
IR	graph state [20-22]	cluster state [18]	FlexLattice [24]	this paper
				
real-time hardware generation	harder	easier	easier	easier
required hardware size	larger	larger	smaller	smaller
required fusion devices	unbounded	bounded	unbounded	bounded

Figure 5.1. Different IRs for measurement-based quantum computing.

or macro-node approach [42, 211]. As a result, this entangled state can serve as an intermediate representation (IR) between high-level programs and low-level hardware.

Early work in MBQC transformed programs into hardware-agnostic IRs, without considering hardware feasibility or fully exploiting hardware capabilities. Some approaches assume the availability of an entangled state with arbitrary graph structures, creating an extremely flexible IR called graph state [108, 70, 48] (Fig. 5.1). Others assume an entangled state with a fixed 3D lattice structure (or 2D [140]), resulting in a rigid IR called cluster state [176] (Fig. 5.1). These IRs lack simultaneous support for scalable hardware generation and efficient program transformation. For example, due to the probabilistic nature of fusions [84, 99], generating a graph state IR with arbitrary structures at runtime requires endless fusion retries. While the cluster state IR avoids this with their rigid lattice structures on each 2D layer, this rigidity limits the optimization space of program transformation.

Recent work OnePerc [234] introduces an approach to generate a FlexLattice IR between these two extremes, with simulation based on the fusion-based architecture [180]. FlexLattice is structured with two spatial dimensions and one temporal dimension. It maintains a lattice structure within each 2D layer, but allows both adjustable edges (depicted as dashed lines) and connections between nodes at the same locations of any 2D layers (Fig. 5.1), referred to as temporal edges. These flexible temporal edges allow programs to be transformed onto a flexible

2D size, lowering the required hardware size (i.e., less chiplets on the fusion-based architecture, as shown in Fig. 5.2) and runtime overhead. Note that in FlexLattice IR, every node can have two temporal edges, one with preceding layers and one with subsequent layers, although we only depict some of the temporal edges to keep the figure clear.

At the architectural level, the flexible temporal edges in FlexLattice are enabled by the utilization of delay lines, which are equipped in various photonic architectures [180, 39, 4, 42, 211]. This represents an alternative scaling strategy versus simply increasing the hardware size. Taking the fusion-based architecture as an example, integrating different lengths of delay lines on each chiplet allows photons to be delayed for different clock cycles before being fused. Hence photons generated at different times can be fused together [39], thereby enabling temporal edges of different lengths. For example, in Fig. 5.2, the two purple fibers represent a no-delay and a two-layer delay: photons generated two layers apart can be routed through these fibers to arrive simultaneously at the fusion device.

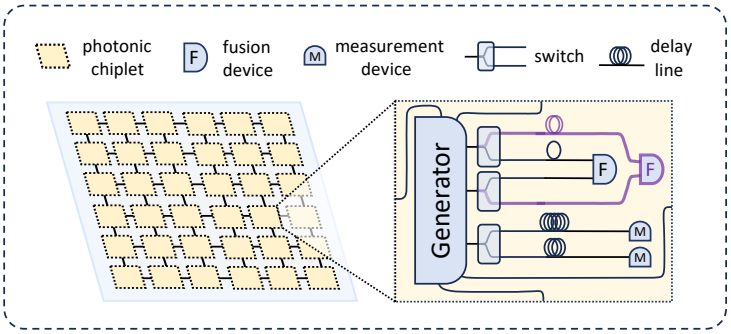


Figure 5.2. Physical hardware of photonic platform.

However, this IR also has its own limitations. First, its flexibility may lead to an unbounded requirement on the number of hardware resources without a resource-adaptive compiler (Fig. 5.1). This is because supporting edges of arbitrary temporal lengths from 1 to n requires $O(n)$ delay lines of different lengths, with each pair of delay lines connected to a fusion device [39]. Hence a resource-adaptive compiler is essential for exploring the aforementioned two scaling strategies and understanding their interplay and tradeoffs in architectural design.

Second, this IR does not fully exploit the capability of fusion-based architectures. While FlexLattice IR only permits temporal edges between nodes at the same 2D location, we observe that its hardware implementation can accommodate skewed edges without requiring additional hardware capabilities. When the skew is bounded to a small 2D range, implementation of these skewed edges incurs negligible overhead.

With these insights, we identify two key opportunities to improve the FlexLattice IR, as depicted in Fig. 5.1. **First**, we capture the hardware constraint of limited delay lines by restricting the lengths of temporal edges in the IR within a certain limit. **Second**, we enable a larger program optimization space by introducing skewed temporal edges, which connect nodes at nearby 2D positions of different 2D layers. These additional IR features necessitate innovative compilation techniques, for which we identify three optimization objectives: (1) minimizing the 1D depth of IR programs to reduce their execution time; (2) minimizing the required 2D size of the IR to reduce the required hardware size, while remaining adaptive to a larger 2D size when hardware allows; (3) being adaptive to a limit on temporal edge lengths in the IR. Simultaneously achieving these objectives is highly non-trivial as they can easily conflict with each other without a careful balance.

To this end, we propose two novel optimization passes for this new IR. **First**, we design a dynamic node refresh mechanism to restrict the lengths of temporal edges, with a smaller 1D depth overhead than OnePerc's periodic refresh [234]. It refreshes nodes based on their storage time in delay lines, ensuring a refresh before the temporal edge length of each node approaches the limit. These node refreshes are balanced with mapping of new nodes by dynamically adjusting the percentage of refreshed nodes on each 2D layer, thereby avoiding a significant increase in 1D depth. **Second**, we develop a 2D-bounded temporal routing to exploit the optimization space enlarged by the skewed edges, with a modified version of this pass proposed for architectures not allowing skewed IR edges. Specifically, temporal edges between different 2D layers are skewed in suitable directions to bring nodes closer on subsequent layers, with its use cases including, but not limited to, resembling a native SWAP gate in the circuit model. This reduces both the 2D

size and the 1D depth of IR without necessarily increasing the lengths of temporal edges.

Our contributions in this paper can be summarized as below:

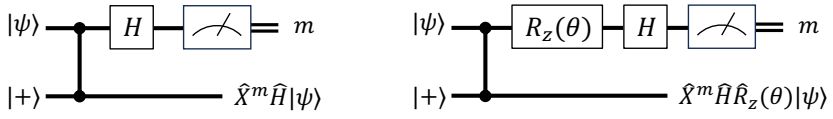
- We propose a new IR by restricting the lengths of temporal edges and allowing skewed edges between 2D layers, which reflects the constraint of hardware resources while opening up broader program optimization space.
- With this new IR, we propose a resource-adaptive compiler with two optimization passes: a dynamic node refresh to ensure the restriction of temporal edge lengths and a 2D-bounded temporal routing to enable more efficient routing with the skewed edges.
- Our compiler achieves resource-adaptivity along three dimensions: it adapts to (1) the available hardware size, (2) the number of delay lines on the hardware, and (3) whether the hardware can support skewed IR edges or not.
- Compared to FlexLattice IR compilers, our compiler reduces the 1D depth by 3.48× while constraining temporal edge lengths. Compared to cluster state IR compilers, it can reduce the 1D depth by 6.7× or reduce the minimal 2D size by 7×. We also extend our framework to the FTQC setting using surface code, demonstrating a 3.33× reduction in 1D depth.

5.2 Background and Related Work

5.2.1 MBQC Basics

One-qubit Teleportation. The foundation of MBQC is the well-known concept of 1-qubit teleportation. As shown in Fig. 5.3(a), if two qubits are in an entangled state $CZ(|\psi\rangle \otimes |+\rangle)$, measuring the first qubit $|\psi\rangle$ is in the X-basis (implemented by applying a Hadamard gate H before a Z-basis measurement) would result in the second qubit becoming $X^m H|\psi\rangle$, where X is a Pauli correction and m is the measurement outcome. When the first qubit is measured in a general basis on the X-Y plane of the Bloch sphere (implemented by applying a Z-rotation $R_z(\theta)$

before the X-basis measurement), the state of the second qubit would become $X^m H R_z(\theta)|\psi\rangle$, as shown in Fig. 5.3(b). In these examples, the resulting state of the second qubit is determined by the measurement basis of the first qubit.



(a) (b)

Figure 5.3. 1-qubit teleportation (a) and its generalization (b).

Graph State. Computation in the MBQC paradigm is driven by single-qubit measurements on an entangled state, called *graph state*. Formally, it is an entangled state among qubits located on a graph $G(V,E)$, defined as the eigenstates of operator

$$s = X_i \bigotimes_{j \in n_i} Z_j, \quad \forall i \in V$$

where n_i are the neighboring qubits of $i \in V$ on graph G. When the graph G has a lattice structure, it is called a *cluster state*. A graph state can be created by preparing all qubits in $|+\rangle$, with some input qubits in the input states, and then applying a CZ gate between each pair of qubits connected by a graph edge. Computation on a graph state can be implemented by a pattern of *equatorial measurements* $E(\alpha)$, i.e., measurements on the X-Y plane of Bloch sphere at an angle α. The measurement basis of each qubit is predetermined by the quantum program, together referred to as a *measurement pattern*. But they are subject to a runtime adjustment according to the measurement outcomes of prior qubits, with the angles adjusted from α to $(-1)^s\alpha + t\pi$ where $s,t \in \{0,1\}$ [71].

MBQC has the same computation power with the circuit model since they are both universal computing models [176]. The required structure G of the graph state for a quantum program can be obtained by a straightforward translation [48] from a circuit in the universal gate set $\{J(\alpha), \mathrm{CZ}\}$ into a measurement pattern. This is because a measurement $E(\alpha)$ yielding an

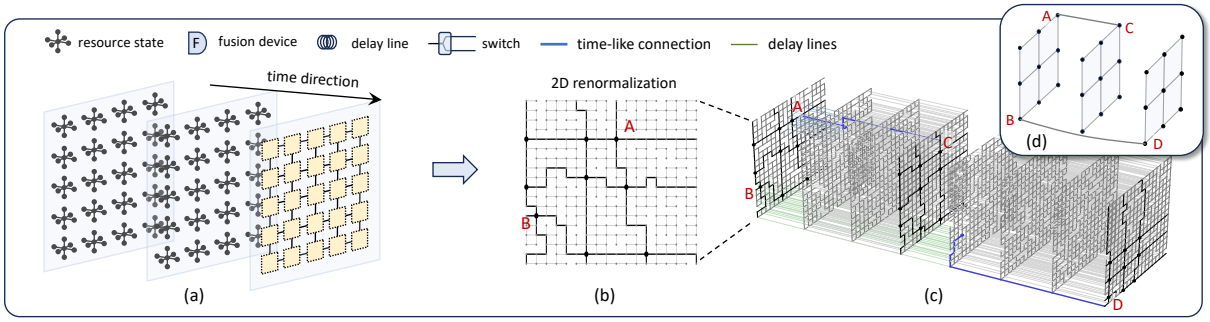


Figure 5.4. (a) RSLs generated over time by a chiplet array. (b) Identified lattice (black) on an RSL by 2D renormalization, with missing edges on the grid (grey) standing for failed fusions. (c) Generation of IR program in (d) on multiple RSLs: bold paths correspond to edges in (d); green lines stand for photon delay; the 1st, 4th and 8th RSLs serve as the 3 logical layers in (d).

outcome $m \in \{0,1\}$ is equivalent to a $J(\alpha)$ gate followed by a Z-measurement M_z yielding the same value, i.e., $E(\alpha)|\psi\rangle \to m$ corresponds to $M_z J(\alpha)|\psi\rangle \to m$, where

$$J(\alpha) = \begin{bmatrix} 1 & e^{i\alpha} \\ 1 & -e^{i\alpha} \end{bmatrix}$$

This process can be described in ZX-calculus [213] and optimized by available tools such as PyZX [122].

5.2.2 Photonic Platform

Fusion-based Architecture. On photonic platforms, a large graph state can be generated by merging small resource graph states through fusions [180], which are concurrent measurements on two photonic qubits in two entangled bases, such as $X \otimes Z$ and $Z \otimes X$.

In the fusion-based architecture [180], a photonic processor consists of an array of interconnected chiplets as shown in Fig. 5.4(a). Each chiplet can be equipped with a resource state generator (RSG), delay lines, fusion and measurement devices, as shown earlier in Fig. 5.2. Each RSG generates a copy of resource state in every cycle, which will be directed to different fusion or measurement devices by switching between different delay lines. Resource states

generated in the same RSG cycle form a 2D resource state layer (RSL), as shown in Fig. 5.4(a). Due to connectivity constraints, resource states on the same RSL are restricted to fuse with their neighbors, while those on different RSLs are restricted to fuse with other resource states generated by the same RSG. These fusions lead to a 3D entanglement structure in space-time, as shown in Fig. 5.4(c). Here the resource states are assumed to be 7-qubit star-shaped, as demonstrated in Fig. 5.4(a). When the available resource states are smaller, this can also be achieved by merging more RSLs [234].

Experimental Progress. Our design is fully built on technologies that have been physically demonstrated by previous experiments. Key hardware components required by this architecture include resource state generation (demonstrated with up to 8 qubits [208, 89, 63, 62, 207, 228, 32, 115, 187]), heralded (type II [126]) fusions (demonstrated with 99.22% fidelity [208, 6]) or boosted fusions [84, 99] (demonstrated with 89% fidelity and 69.3% success rate [106]), single-qubit measurements (demonstrated with 99.98% fidelity [6]), chip-to-chip qubit interconnect (demonstrated with 99.72% fidelity [6]) and delay lines with a low photon loss (potentially as low as 0.2 dB/km [139]). Moreover, the feasibility of MBQC has been demonstrated by various small-scale quantum algorithms on photonic qubits [216, 212, 202], and simple QEC codes are also demonstrated recently [15]. While current photonic hardware still lags behind other platforms in the scale of supported programs, ongoing improvements in resource generation and noise reduction suggest strong long-term scalability.

5.2.3 MBQC on Photonic Platform

Previous Work. Early work in MBQC proposes a cluster state IR with a lattice structure, demonstrating universality of MBQC by translating a universal gate set in the circuit model to corresponding measurement patterns on cluster states [176]. Recently, a compiler FMCC [140] is proposed to optimize this translation, with the cluster state restricted to a 2D scenario. Another work OnePerc [234] proposes a more flexible FlexLattice IR, which allows edges along the third dimension to connect nodes at the same 2D positions of any 2D layers, as depicted in Fig. 5.1

and Fig. 5.4(d). Moreover, each edge in this IR can be enabled or disabled on demand. For compilation, OnePerc adopts a mapping algorithm adapted from [235].

IR Generation. To generate the FlexLattice IR, resource states on each RSL are fused with their neighbors, while different RSLs are fused according to the IR structure by switching between different delay lines. In Fig. 5.4(b)(c), gray lines represent the entanglement formed by successful fusions, while green lines represent photon delay. Fusion failures [84, 99], as indicated by the missing edges, can be handled by percolation-based [118, 167] algorithms, so that the IR can be generated with a high success rate [234].

(1) *2D Lattice Identification.* On each RSL, a path searching is conducted among successful fusions to identify a connected lattice of a certain size (3×3 bold lattice in Fig. 5.4(b)), referred to as *renormalization* [49, 155]. Successfully renormalized RSLs (the 1st, 4th, 8th RSLs in Fig. 5.4(c)) can serve as *logical layers*, which correspond to the 2D layers in the IR (the three layers in Fig. 5.4(d)).

(2) *Temporal Edge Generation.* To generate an IR structure as shown in Fig. 5.4(d), two temporal edges *AC* and *BD* need to be generated. This is achieved by finding connected paths between the renormalized nodes *A* and *C*, and between *B* and *D*, as depicted by bold blue paths in Fig. 5.4(c). Note that since *BD* connects two non-adjacent 2D layers in the IR, the resource states around *B* are delayed to fuse with the RSL after the second logical layer, unlike other resource states that are fused with their next RSL.

PL Ratio. Due to fusion failures, the formation of each logical layer costs multiple RSLs. This is because both 2D renormalization and time-like path searching can fail. The ratio between the number of physical RSLs and logical layers is thus defined as a *PL ratio*. With a practical fusion success probability of 75% [84, 99], simulation in [234] reveals this ratio as ~ 3.1 on average, which can vary across different logical layers. In the runtime, this ratio can be fixed to avoid real-time feed-forward overhead. To ensure a high success probability of IR generation, a reasonable choice would be slightly larger than the average ratio, such as 4, meaning that a logical layer is assigned every 4 RSLs.

5.2.4 FTQC on Photonic Platform

To realize photonic FTQC, multiple physical qubits need to be encoded into a logical qubit through quantum error correction (QEC) codes [159]. Existing work has shown that any QEC code in the CSS category can be implemented on MBQC in a foliated manner [38]. In this approach, each RSL mimics a time slice of QEC, with the fusion patterns on the RSL determined by the code structure. For surface code, one of the most promising QEC codes [90], these RSLs are fused into a Raussendorf cluster state [178], with the structure of each layer resembling the structure of surface code. This correspondence between an MBQC layer and a time slice of QEC enables implementation of universal logical gates in the MBQC paradigm [110, 40].

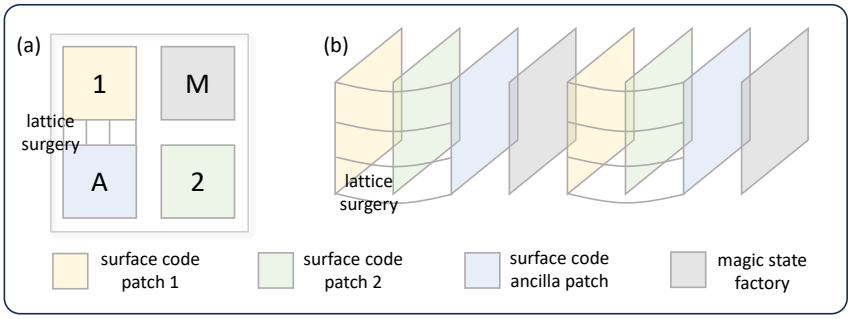


Figure 5.5. Periodic refresh for QEC.

Fusion-based architectures require a large QEC code distance due to the need to tolerate fusion failures and photon loss [40], leading to a requirement for large RSLs. This resource overhead can be mitigated by a space-time trading that allocates different logical qubits onto different RSLs. Fig 5.5(a) illustrates a simple example with four surface code patches, including an ancilla patch and a magic state factory to facilitate logical gates with lattice surgery [113, 141, 222] and magic state distillation [91, 34]. As depicted in Fig. 5.5(b), these patches can be assigned onto RSLs periodically in an interleaved manner, with the period being four layers here. As a lattice surgery [113, 141, 222] between two patches only involves operations on the patch boundaries (Fig. 5.5(a)), it can be realized through fusing the boundaries of different layers with the help of delay lines (Fig. 5.5(b)).

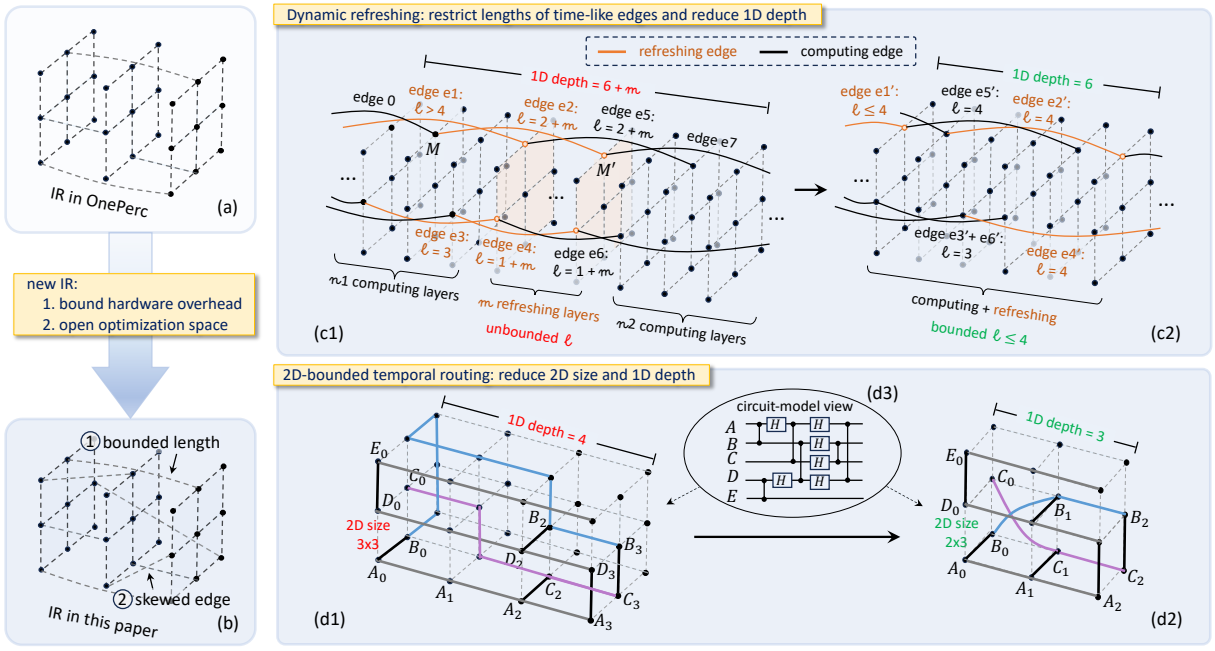


Figure 5.6. FlexLattice IR in OnePerc (a) and our IR (b). Dynamic refreshing that prevents temporal edges from exceeding a length limit (c) and 2D-bounded temporal routing that exploits skewed edges in the IR (d).

5.3 Design Overview

5.3.1 Motivation and Challenges

Our IR differs from the previous FlexLattice IR with two additional features, as depicted in Fig. 5.6(b). First, we impose a limit on the lengths of temporal edges, which can vary with evolving hardware capabilities. Specifically, this limit is determined by the number of delay lines on the hardware, with $2D_f$ delay lines per chiplet leading to a length limit of D_f. Second, we introduce more flexibility by allowing skewed temporal edges between nodes at nearby 2D positions. These skewed edges reflect more efficient hardware utilization, as they can be readily implemented by the same architecture as OnePerc with a slight algorithm modification, without requiring additional hardware capabilities such as temporal connections between different RSGs.

Among the three optimization objectives, a reduced 1D depth implies a shorter program

execution time, which is essential for achieving quantum speedup, as practical quantum applications can run for days or even years [35, 244]; a reduced 2D size leads to a smaller requirement for the number of chiplets, as well as a faster renormalization in the runtime; while a bounded temporal edge length is necessary for adapting to any practical hardware. Achieving these optimization objectives simultaneously is highly non-trivial, as they can conflict with each other without careful optimizations. The first conflict lies between the 1D depth and the lengths of temporal edges. While temporal edge lengths can be reduced by periodically refreshing IR nodes after every fixed 1D depth [234], this can increase the total 1D depth as each round of refresh requires multiple additional 2D layers. These additional 2D layers can in turn extend the node storage time in delay lines, thereby preventing a guaranteed restriction of temporal edge lengths. The second conflict arises between the 2D size and the lengths of temporal edges. Specifically, the reduction of 2D size in OnePerc [234] is achieved by leveraging the flexible temporal edges, at the cost of incurring long temporal edges. This challenges our goal of restricting temporal edge lengths while reducing 2D size. Our compiler achieves a balance between these optimization objectives through the following two optimization passes.

5.3.2 Dynamic Refreshing

Our compiler is designed to be adaptive to a length limit of temporal edges by introducing dynamic refreshing, which also leads to a smaller 1D depth than OnePerc. We illustrate this with an example in Fig. 5.6(c1)(c2), which shows a bounded length limit of 4 for all temporal edges and a reduction in 1D depth from $6+m$ to 6.

In Fig. 5.6(c1)(c2), black edges represent computing edges, which are edges forming the measurement pattern used for computation, while orange edges represent refreshing edges, which are edges measured in fixed bases merely for node refresh. A node refresh can reduce the storage time of the node in the delay line by remapping it onto the current 2D layer, where computation on that node can be continued after it is refreshed. For example, in Fig. 5.6(c1), node M is refreshed by refreshing edge $e2$ to a new node M', which reduces the length of computing edge

$e0$. After this refresh, the computation on node M can be continued on M' with computing edge $e7$.

In previous work OnePerc [234], computing and refreshing are conducted alternately, as shown in Fig. 5.6(c1), which we refer to as *periodic refreshing*. After each certain depth of computation, all nodes stored in delay lines are refreshed by allocating additional multiple 2D layers for refresh only (the orange layers). Although this can prevent very long temporal edges, the length limit is still unbounded since it is hard to control the length of each temporal edge when all nodes are refreshed together. In particular, the number of required additional 2D layers m varies in every round of refresh based on how many nodes need to be refreshed and how many of them have conflicted 2D coordinates. This induces an unpredictable overhead to the lengths of temporal edges. For example, the lengths of edge $e2, e4, e5$ and $e6$ are $2+m, 1+m, 2+m$ and $1+m$, all relying on the number of additional refreshing layers m.

In our compiler, each node is refreshed dynamically based on their need, without a distinction between computing layers and refreshing layers. On every 2D layer, nodes are refreshed selectively to prevent temporal edges from exceeding the length limit. This is demonstrated in Fig. 5.6(c2), where the length limit of temporal edges is set to be 4. Each edge e' in Fig. 5.6(c2) is a correspondence of edge e in Fig. 5.6(c1). This dynamic refreshing exhibits the following advantages. **First**, it ensures a bounded length limit for temporal edges. In Fig. 5.6(c1), the reason why edge $e1$ has a length > 4 is because its refresh has to wait until a round of refreshing begins. In Fig. 5.6(c2), the length of edge $e1'$ can be reduced, since refresh is enforced whenever the limit is reached, without a distinction between computing layers and refreshing layers. Similarly, the lengths of edge $e2, e4, e5$ and $e6$ can also be reduced to 4, no longer relying on an unbounded value m. **Second**, the elimination of dedicated refreshing layers naturally reduces the 1D depth, as computation on the refreshed nodes do not need to wait until the next computing round begins. In Fig. 5.6(c2), the 1D depth is reduced from $6+m$ (Fig. 5.6(c1)) to 6.

5.3.3 2D-bounded Temporal Routing

Our compiler exploits the broader optimization space enabled by the new IR with a 2D-bounded temporal routing, which can reduce the 2D size and 1D depth of IR programs. We demonstrate this with an example in Fig. 5.6(d1)(d2), which shows a 2D size reduction from 3×3 to 2×3 and a 1D depth reduction from 4 to 3.

On layer 0, node A, B, C, D and E are located in $(0,0), (1,0), (1,1)$, $(0,1)$ and $(0,2)$, denoted by A_0, B_0, C_0, D_0 and E_0, with the subscripts indicating their 2D layers. To implement an MBQC program equivalent to a circuit shown in Fig. 5.6(d3), we aim to exchange the 2D positions of B and C through subsequent layers so that AC, BD, AD and BC can all be connectable. In previous work OnePerc [234], this exchange can be achieved as shown in Fig. 5.6(d1), by the blue routing path B_0B_2 and the purple routing path C_0C_2, so that AC and BD are connected on layer 2 as A_2C_2 and B_2D_2 while AD and BC are connected on layer 3 as A_3D_3 and B_3C_3. This requires a 2D size of 3×3 and a 1D depth of 4.

In our compiler with an extended IR allowing skewed temporal edges, this can be realized more easily by skewed temporal edges, as shown by the blue skewed edge B_0B_1 and the purple skewed edge C_0C_1 in Fig. 5.6(d2). This reduces the required 2D size to 2×3 and reduces the 1D depth to 3. In this example, the skewed edges facilitate an easy swap of node positions, which resembles a native SWAP gate in the circuit model. In general cases, these temporal edges can be skewed in more ways, leading to a flexible routing strategy beyond swapping. Even with an IR without skewed edges, this flexible routing strategy can be modified to achieve a more efficient routing than that of OnePerc [234].

5.4 Framework Design

In this section, we first present the overall compilation flow along with the complexity analysis of key algorithmic components, and then provide a detailed description of each optimization module.

5.4.1 Compilation Flow and Complexity

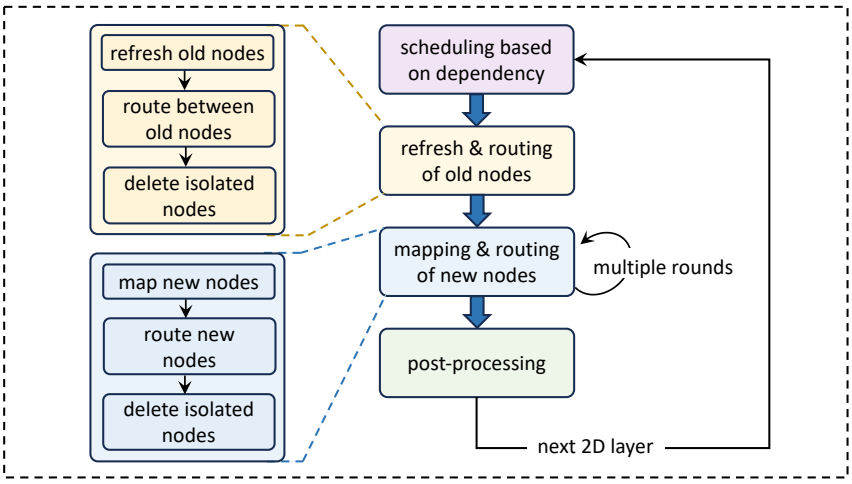


Figure 5.7. Compilation flow.

Scheduling. As shown in Fig. 5.7, given an MBQC program represented by a graph state, our compiler first schedules the graph nodes based on their dependencies. This is achieved by constructing a directed acyclic graph (DAG), with each directed edge indicating a dependency between graph state nodes. To eliminate the need for a node to wait in delay lines before measurement, every time we only take the nodes in the front layer of the DAG, thus making sure that the target node of each directed edge is mapped to a later 2D layer than the source node.

Refreshing old nodes. For each 2D layer, we first select some old nodes stored in delay lines and refresh them (yellow in Fig. 5.7). The only exception is that, in the beginning of compilation when there are no nodes in the delay lines, we first map all input nodes in the graph state, allowing them to occupy multiple 2D layers if the given 2D size is small. Selected old nodes would be refreshed by mapping them to 2D positions near their original ones through 2D-bounded temporal routing. With the heap structure in our codes for node storage, the complexity of this step is $O(S \log S)$ where S is the 2D size of IR.

Routing old nodes. Next, we map graph state edges between these refreshed nodes (yellow in Fig. 5.7). For each refreshed node, we check if their neighbors in the graph state are also refreshed. If so, we connect them through shortest paths if allowed by the remaining space

on the 2D layer. This routing is followed by a deletion of isolated nodes to save 2D space for subsequent steps. That is, if a refreshed node is not connected with any other node after this routing step, this isolated node will be removed from the 2D layer. The only exception is when its storage time in the delay line is about to exceed the limit. Shortest paths are obtained via BFS on the unweighted grid, with a complexity of $O(S^2)$.

Mapping and routing new nodes. After refreshing and connecting old nodes, we then start mapping and routing new nodes (blue in Fig. 5.7). Among nodes in the current front layer, every time we select a graph state neighbor of a node mapped on the current 2D layer, and map it to an adjacent position of that mapped node if allowed by the remaining 2D space. Then we route these new nodes to connect them through shortest paths if they have edges in the graph state. Shortest paths are obtained via BFS on the unweighted grid, with a complexity of $O(S^2)$.

Multiple rounds of mapping and routing. The mapping and routing process is performed for multiple rounds (by default 3) if 2D space remains (circular arrow in Fig. 5.7). For each round after the first one, we first select some new nodes randomly from the front layer and map them to random positions, then map their graph state neighbors in the front layer to their adjacent positions. After that, we connect these new nodes through shortest paths if they have other edges. Finally, if some selected nodes are isolated after this round, we remove them from the 2D layer to save space for subsequent rounds.

Post-processing. After refreshing, mapping and routing on each 2D layer, a post-processing (green in Fig. 5.7) is conducted to address a special case where a node on the current 2D layer has only one unmapped neighbor left in the graph state, thereby saving the space of subsequent layers. The complexity of this step is $O(S)$.

5.4.2 Scheduling to Maintain Dependency Order

Due to the randomness of quantum measurement outcomes, the measurement bases of some graph nodes must be adjusted at runtime, depending on the measurement outcomes of other nodes. These dependencies result in a partial order among different nodes, which can be derived

using measurement calculus [71, 48]. As a preprocessing step, we construct a DAG to capture the dependency relations among graph state nodes. For each directed edge, the measurement basis of the target node depends on the measurement outcome of the source node.

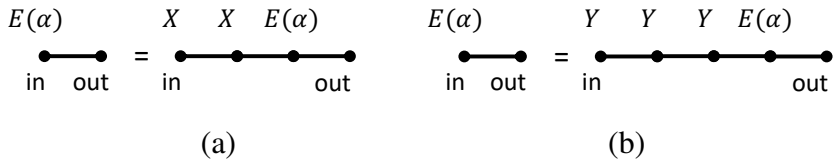
The maintenance of dependency order in the compilation refers to mapping the target node of each directed edge to a later 2D layer than the source node. This ensures that *mapping-complete* nodes can be measured immediately without waiting for other nodes to be mapped As each node can have only one temporal edge with subsequent layers, we call a node mapping-complete if it is mapped and has only one unmapped neighbor left in the graph state. For each 2D layer, we first update the DAG by removing mapping-complete nodes, then select nodes from the front layer of the DAG for refresh and mapping. The front layer consists of nodes without in-degrees in the DAG, meaning that all the nodes they depend on have been mapping-complete on preceding 2D layers. In this way, we make sure that a node is always scheduled on a later 2D layer than nodes it depends on, thus eliminating the necessity for nodes to wait in delay lines before measurements.

A worth-mentioning subtlety about the DAG is the signal shifting in measurement calculus [71, 48]. While signal shifting can help remove redundant dependencies among nodes, it can also result in connections between very preceding and very subsequent nodes in the dependency relations, which compromises the compiler's capability of restricting temporal edge lengths. Hence the DAG we construct here is based on dependencies without signal shifting.

5.4.3 Dynamic Refreshing

Node Refresh. The feasibility of node refresh lies in the fact that two consecutive X- or three consecutive Y-measurements can act as an identity operation that transfers an input state to another qubit up to some Pauli corrections [176, 48], as illustrated in Fig. 5.8. Note that while a refresh needs two additional X- or three additional Y-measurements, in the fusion-based architecture we do not need to remap the node twice or three times, because each edge in the IR is realized by a connected path of successful fusions on the physical hardware (Fig. 5.4), which

already consists of multiple photonic qubits. The choice of consecutive X- and Y-measurements along the path is determined by whether the path has an even (XX$\cdots$XX) or odd (XX$\cdots$YYY) length.



(a) (b)

Figure 5.8. Insertion of identity patterns (a) XX and (b) YYY.

Refresh Prioritization. We classify node refreshes into two types according to their purposes. The first type is *constraint-driven refresh*, which is performed to prevent temporal edges from exceeding the length limit. The second type is *computation-driven refresh*, which remaps old nodes to the current 2D layer to continue pending computation blocked by node dependencies. Note that these two types are not mutually exclusive, as a node mapped by constraint-driven refresh can also participate in computation.

For each 2D layer, we first perform all constraint-driven refreshes, and then perform computation-driven refreshes if space permits. Our insight is that strategically prioritizing computation-driven refreshes can proactively reduce the future demand for constraint-driven refreshes. To achieve this, we rank old nodes into the following three categories according to their computational relevance, with the categories listed in descending order of priority. Within each category, nodes are further prioritized in descending order of their storage time. Performing computation-driven refresh according to this prioritization allows long temporal edges to be eliminated earlier, thereby alleviating refresh pressure in subsequent layers.

- Nodes with graph edges to constraint-driven refreshed nodes
- Nodes with graph edges to other old nodes in delay lines
- Nodes with graph edges only to new unmapped nodes

These categories are prioritized for the following reasons. The first category includes nodes that are graph state neighbors of constraint-driven refreshed nodes. Since a constraint-driven refreshed node has already been mapped onto the current layer, mapping its graph state neighbors brings both ends of a graph state edge onto this layer. This can advance computation if a routing path exists, allowing constraint-driven refreshes to contribute to computation as well. The second category consists of nodes that are graph state neighbors of other old nodes in the delay lines. Mapping these nodes similarly brings both ends of graph state edges onto the current layer, enabling computation progress when routing paths are available in the 2D space. The third category contains the remaining nodes that only have edges with new unmapped nodes. Mapping these nodes offers fewer immediate computational opportunities compared to the other categories, as their impact depends on the subsequent mapping of new nodes.

Refresh Percentage Tuning. Balancing the refresh of old nodes with the mapping of new nodes is crucial for minimizing the 1D depth. Our insight is that the number of constraint-driven refreshes can serve as a probe of this balance for subsequent layers, motivating a dynamic adjustment of the ratio between old and new nodes. When constraint-driven refreshes become excessive, it signals that the ratio of node refresh should be increased to mitigate the accumulation of old nodes.

On each 2D layer, we limit the fraction of positions that can be used for node refresh through a refreshing bound b_r. Constraint-driven refreshes are allowed to exceed this bound, but computation-driven refreshes cannot. That is, if the number of constraint-driven refreshed nodes is $m < b_r S$, at most $b_r S - m$ nodes can be selected for computation-driven refresh. This refreshing bound b_r is dynamically adjusted based on the number of constraint-driven refreshed nodes m_{last} on the last 2D layer and the size of 2D layers S, in the following way:

$$b_r = \min\{m_{\text{last}}/S + p,\ \ 1\} \tag{5.1}$$

where p is an adjustable parameter with a default value of 0.4. When p is too small, the limited

computation-driven refresh is a bottleneck for computation progress. When p is too large, it prevents new nodes from being mapped, which also slows down the computation. As a result, a sweet spot needs to be identified empirically.

5.4.4 2D-bounded Temporal Routing

With Skewed Edges. Based on the 2D position where a node was last mapped on preceding layers, each node v stored in the delay lines is labeled with a 2D coordinate (x_v, y_v). When refreshed onto the current 2D layer, the node can be mapped to a position (x', y') around (x_v, y_v) within a range allowed by skewed temporal edges. To reduce resource overhead, our IR restricts this range to a Hamming distance of 1. That is,

$$(x', y') \in \text{near}((x_v, y_v)) \equiv (x_v, y_v) \cup \text{adj}((x_v, y_v))$$

where

$$\text{adj}((x_v, y_v)) \equiv \{(x_v + 1, y_v), (x_v - 1, y_v), (x_v, y_v + 1), (x_v, y_v - 1)\}$$

While the motivating example in Fig. 5.6(d2) illustrates skewed edges for node swapping only, broader optimization can be supported by more flexible routing of individual nodes. To this end, we design a heuristic for 2D-bounded temporal routing that minimizes the distance between neighboring nodes in the graph state and maximizes the available spaces around mapped nodes. Specifically, nodes selected for refresh, denoted as R, may or may not have graph state edges with each other. For those with edges, we minimize their mutual distances by remapping each node v to a position (x', y') with the following heuristic

$$\min_{(x',y') \in \text{near}((x_v, y_v))} \sum_{n \in R \cap \text{neigh}(v)} \text{dist}((x', y'), (x_n, y_n))$$

where $\text{neigh}(v)$ stands for the graph state neighbors of v, and the coordinate (x_n, y_n) stands for

the position of each neighboring node n in the storage. For those without edges, we maximize their surrounding spaces by remapping each node v with the following heuristic

$$\max_{(x',y')\in \text{near}((x_v,y_v))} \sum_{u\in \text{adj}((x',y'))} \Theta(u \text{ not occupied})$$

where Θ is the step function with a value 1 when the condition is satisfied and a value 0 otherwise.

Besides node refresh, new nodes can also be mapped with a similar heuristic to minimize the distance between neighboring nodes in the graph state and maximize the available spaces around mapped nodes. After that, a post-processing step will be conducted to handle a special case before storing nodes into the delay lines. That is, if a node mapped on the current 2D layer has only one unmapped neighbor remaining in the graph state, then we mark this node so that the next time it is refreshed, the refreshed node is directly replaced with its unmapped neighbor.

No Skewed Edges. Our routing strategy can extend to work on IR without skewed edges, thereby broadening its applicability to other architectures [42, 211]. It is feasible as we can realize a temporal swap of two nodes (A, B in Fig. 5.9) with two 2D layers by storing one of the nodes (node B in Fig. 5.9(b)) in the delay lines. We refer to the paths like A_0A_1 and B_0B_2 in Fig. 5.9(b) as L-shaped edges.

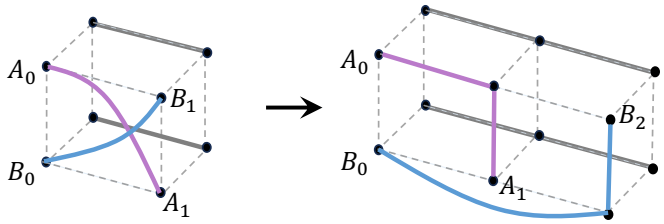


Figure 5.9. Swap node positions without skewed edges.

The primary difference from the algorithm with skewed edges lies in the node refresh stage. After we obtain each node's position by searching in the range $(x', y') \in \text{near}((x_v, y_v))$, we detect which nodes are involved in position swaps with other nodes. If two nodes intend to swap their positions, we rank them based on their priorities. When both nodes are in the highest

priority of constraint-driven refresh, we remap them to (x_v, y_v), the same positions where they were mapped last time. Otherwise, we choose the one with higher priority and remap it with an L-shaped path, storing the other node in the delay lines to wait for subsequent layers. For nodes not involved in position swaps, we simply remap them to their previous positions (x_v, y_v).

5.4.5 Implementation of Skewed Edges

The skewed edges in our new IR can be readily implemented by the same architecture with OnePerc without requiring additional hardware capabilities. This is because each edge in the IR is formed by a path of successful fusions, which is obtained through a (2+1)-D path searching among successful fusions, as illustrated earlier in Fig. 5.4(b)(c)(d). Since two successive logical layers are separated by several physical layers in between (Fig. 5.4(c)), a skewed IR edge between two logical nodes can be formed easily by a skewed fusion path through the physical layers in between, consisting of successful temporal fusions between the same RSG and successful spatial fusions between different RSGs. As a result, the only modification we need is to allow path searching between qubits corresponding to IR nodes at **nearby** 2D locations of different IR layers.

Intuitively, this implementation may incur two overheads: (1) The more flexible path searching can increase resource overhead by increasing the PL ratio, as the paths for different skewed IR edges in various directions can more easily conflict with each other. However, through simulation, we find that this overhead is negligible when the 2D range of skewed IR edges is restricted to only 1. (2) The skewed IR edges tend to have longer fusion paths than straight IR edges, introducing fidelity degradation per IR edge. However, the substantial reduction in 1D depth with this new IR also reduces the total edge count significantly, thus improving the overall fidelity.

5.5 Evaluation

5.5.1 Experiment Setup

Metrics. We evaluate the performance of our compiler with the following four metrics of transformed IR programs: 1D depth, 2D size, the maximal length of temporal edges D_f (i.e., the maximal wait time for fusion), and the maximal wait time for measurement. We only explain the last one here as the other three have been explained in earlier sections. Besides the unbounded number of fusion devices, OnePerc [234] also requires an unbounded number of measurement devices due to its over-flexible scheduling, which can cause an IR node to wait excessively long in delay lines before measurement. Hence we list this wait time for measurement required by compiled programs as the fourth metric.

Benchmark Programs. We select a set of benchmark programs for NISQ and FTQC, respectively. Benchmarks for NISQ include Quantum Approximate Optimization Algorithm (QAOA), Ripple-Carry Adder (RCA) [64] and Variational Quantum Eigensolver (VQE). QAOA is implemented on randomly generated 3-regular graphs, where ZZ gates are applied only between graph-connected qubits. For VQE, we first adopt a full-entanglement ansatz as default, then use the UCCSD ansatz to demonstrate simulations of realistic molecular systems, including H_2O, CH_4 and C_2H_6, with circuits compressed by standard active space approximations with frozen core orbitals. Benchmarks for FTQC include Quantum Fourier transform (QFT), Grover's algorithm and quantum simulation (QSIM). Among them, QFT is a crucial component of Shor's algorithm [191] as well as other applications in signal processing [147, 17] and cryptography [7]. Grover's algorithm [100] can provide quadratic quantum speedup for unstructured search problems. For QSIM [210, 135], we simulate the time evolution of quantum spin systems governed by the XXZ Heisenberg Hamiltonian—a widely used model in condensed matter physics for studying quantum magnetism, entanglement dynamics, and many-body effects, with circuits constructed by Trotterization. In the evaluation, each 'benchmark-n' stands for a program corresponding to an n-qubit circuit in the circuit model.

Hardware Settings. The best known scheme for fusion implementation has a theoretical success rate of 78% [84], whereas a success rate of 75% is achievable with significantly less resource overhead [84, 99]. As a result, we adopt the 75% fusion success rate in the experiments that demonstrate the feasibility of IR extension, and provide a sensitivity analysis by varying it from 66% to 78%. By fixing the PL ratio as 4 (i.e., a slightly higher value than the natural PL ratio of ~ 3.1 to ensure success of IR realization), our IR implementation requires D_f fusion devices with connected delay lines on each chiplet, with the longest storage time of photons being $4D_f$ RSLs. With state-of-the-art fiber technology with a loss rate of 0.2dB/km [139], photons can be stored for 1500 RSLs with $\sim 5\%$ loss, given an RSG clock cycle of $\sim 1ns$ [39]. As an example, when $D_f = 10$, the photon loss rate after being stored for $4D_f$ RSLs is hence as small as 0.12%.

Baseline 1. We compare our framework with state-of-the-art photonic MBQC compiler **OnePerc** [234]. Since OnePerc is unable to bound lengths of temporal edges and wait time for measurement, we adopt the following strategies to make it more comparable with our compiler, at the cost of increasing its 1D depth to some extent. First, when a D_f is specified, we ask OnePerc to perform its periodic refresh after every D_f layers and record the longest temporal edge. Second, we reduce its measurement wait time by posing a more strict scheduling. That is, we restrict the number of dependency layers that can appear on each layer (2 is achievable) and record the longest wait time for measurement under this restriction. Note that while these strategies can mitigate the problems, they can not guarantee an adaptive bound for these limits.

Baseline 2. Since OnePerc is not capable of bounding temporal edge lengths and measurement wait time, we compare with a secondary baseline with this capability, to demonstrate the performance of our compiler under such restrictions. This baseline is constructed by first decomposing a circuit into $J(\alpha) + \mathrm{CZ}$ gates, compiling it with **Qiskit** [173] — state-of-the-art compiler in the circuit model, and then translating the compiled results into the cluster state IR. The feasibility of adapting a circuit model compiler to an MBQC compiler lies in the correspondence between the space-time structure of a circuit and the 3D structure of a cluster state [176].

This baseline guarantees a length limit of $D_f = 1$ for temporal edges and a wait time limit of 0 for measurements. However, it requires a minimal 2D size of $n \times n$ for each n-qubit benchmark and results in a longer 1D depth.

5.5.2 Experiment Results.

This subsection shows our experiment results compared with the baselines. Due to the natural tradeoff between 2D size and 1D depth [234], we first fix the 2D size to a *standard 2D size* of $\sqrt{n} \times \sqrt{n}$ for each n-qubit program, while comparing the other three metrics with the baselines. Subsequent analysis will vary the 2D size to demonstrate the adaptivity of our compiler.

Table 5.1. Results of our compiler and the baseline. Target $D_f = 20$.

Benchmark - #Qubit	Depth: One-Perc	Depth: One-Perc + Skew	Depth: Ours	Depth Improv. vs One-Perc	Depth Improv. vs One-Perc + Skew	D_f (compiled): OnePerc	D_f (compiled): OnePerc + Skew	D_f (compiled): Ours	Meas Wait (layers): One-Perc	Meas Wait (layers): One-Perc + Skew	Meas Wait (layers): Ours
QAOA-36	1,108	584	342	**3.24** ×	1.71 ×	37	36	20	32	34	**0**
RCA-36	1,547	1,125	712	**2.17** ×	1.58 ×	27	27	20	24	24	**0**
VQE-36	874	530	254	**3.44** ×	2.09 ×	32	32	17	99	138	**0**
QAOA-64	2,686	2,451	789	**3.4** ×	3.11 ×	42	43	20	39	42	**0**
RCA-64	1,813	2,276	1,444	**1.26** ×	1.58 ×	27	30	20	24	28	**0**
VQE-64	1,932	1,715	648	**2.98** ×	2.65 ×	38	36	20	800	893	**0**
QAOA-100	6,209	6,126	1,540	**4.03** ×	3.98 ×	45	56	20	430	53	**0**
RCA-100	4,294	3,941	2,311	**1.86** ×	1.71 ×	29	30	20	27	27	**0**
VQE-100	4,012	3,109	1,504	**2.67** ×	2.07 ×	45	39	20	1,444	2,395	**0**
H_2O	2,462	2,836	581	**4.24** ×	4.88 ×	39	50	20	37	431	**0**
CH_4	2,958	1,604	524	**5.65** ×	3.06 ×	43	42	20	1,084	40	**0**
C_2H_6	15,369	15,749	2,248	**6.84** ×	7.01 ×	64	70	20	61	67	**0**

Compared to Baseline 1. Table 5.1 presents the comparison of our framework with OnePerc when the specified target length limit of temporal edges is $D_f = 20$, while the 2D size of IR is fixed to the standard size. The improvement factors are obtained from the ratios between the baseline and our compiler. The results show that our compiler achieves a significant reduction in 1D depth, with an average improvement factor of 3.48×. Moreover, the lengths of temporal edges in our compiler are all within the specified D_f, which demonstrates its ability of restricting

temporal edge lengths. In contrast, OnePerc has a significant excess over the specified limit, with the excess increasing as the size of programs increases. Furthermore, our compiler also successfully reduces the wait time for measurement to 0, thus requiring only one measurement device connected with one delay line on each chiplet. In contrast, OnePerc leads to a long maximal wait time for measurement.

We further demonstrate the advantage of our dynamic refresh over periodic refresh by comparing our compiler against a strengthened version of OnePerc, which allows skewed edges and incorporates our 2D-bounded temporal routing. As shown in Table 5.1, while these enhancements improve OnePerc's performance, our compiler still achieves significantly lower 1D depth, with an average reduction of 2.95×. This highlights the effectiveness of dynamic refresh in suppressing 1D depth overhead.

Table 5.2. Results of our compiler when $D_f = 1$ and the baseline.

Benchmark-#Qubit	Qiskit Depth	Qiskit + Skew Depth	Our Depth	Improv. vs Qiskit	Improv. vs Qiskit +Skew
QAOA-36	2,065.0	1,089.0	311.0	6.64	3.5
RCA-36	4,078.0	2,930.0	666.0	6.12	4.4
VQE-36	1,843.0	732.0	220.0	8.38	3.33
QAOA-64	5,246.0	2,896.0	656.0	8.0	4.41
RCA-64	7,689.0	5,337.0	1,227.0	6.27	4.35
VQE-64	3,298.0	1,623.0	508.0	6.49	3.19
QAOA-100	12,129.0	5,689.0	1,172.0	10.35	4.85
RCA-100	12,058.0	8,656.0	1,888.0	6.39	4.58
VQE-100	9,317.0	3,638.0	975.0	9.56	3.73
H_2O	1,529.0	1,099.0	426.0	3.59	2.58
CH_4	1,769.0	1,663.0	401.0	4.41	4.15
C_2H_6	5,666.0	4,303.0	1,337.0	4.24	3.22

Compared to Baseline 2. When the length limit of temporal edges is restricted to $D_f = 1$, Table 5.2 presents the comparison of our framework with the secondary baseline adapted from Qiskit, with the 2D size of IR fixed to the standard size. It can be seen that our compiler significantly reduces the 1D depth, with an average improvement factor of 6.7×. We have omitted the wait time for measurement in Table 5.2, as both the baseline and our compiler have a wait time of 0.

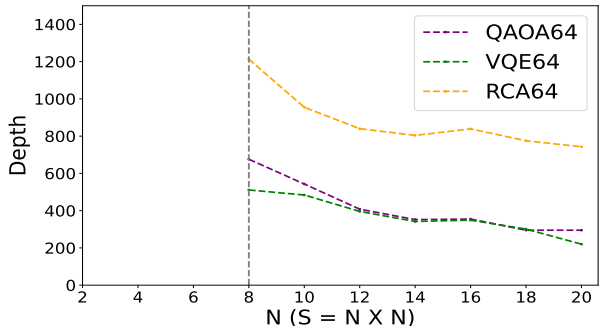
We further compare our compiler with a strengthened version of baseline 2. That is, we allow native SWAP gates in baseline 2 to reduce its 1D depth, with the native SWAP gates implemented by the skewed edges. As shown in Table 5.2, our compiler also achieves a significantly smaller 1D depth than this strengthened baseline 2, with an average improvement factor of 3.86×. This is because our 2D-bounded temporal routing is more flexible than a native SWAP. In the routing process, our compiler tries to find the best 2D position for each node considering the overall mapping, not limited to swapping 2D positions of different nodes.

5.5.3 Resource Adaptivity

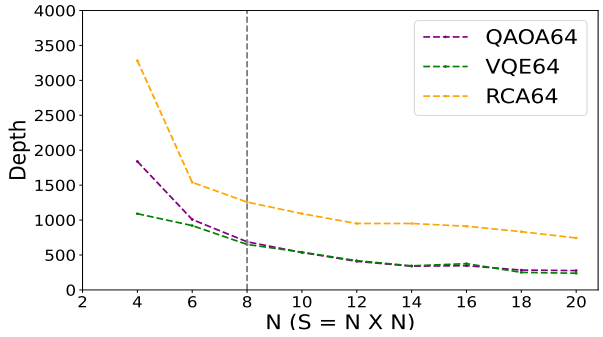
Section 5.5.2 has demonstrated the adaptivity of our compiler to the number of delay lines by experiments with $D_f = 20$ and $D_f = 1$. This subsection further shows its adaptivity to the hardware size and to whether the hardware can support skewed IR edges or not.

Adaptive to 2D Size. Our framework can adapt to different 2D sizes of the IR while restricting temporal edge lengths to a given limit. This facilitates exploration of the two scaling paths of photonic platforms: increasing the number of chiplets and increasing the photon delay resources on each chiplet. By evaluating program performance under these scaling strategies, our framework can be leveraged to support hardware-software co-design, including the identification of resource-efficient configurations when hardware cost models are available.

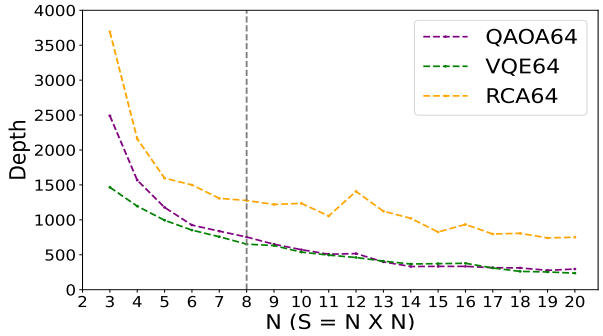
Fig. 5.10(a)(b)(c) show the 1D depth of 64-qubit benchmarks as the 2D size $S = N \times N$ increases when $D_f = 1, 5, 10$, respectively. The 1D depth decreases as the 2D size increases, demonstrating its capability of exploiting larger hardware for more parallel execution allowed by MBQC. As D_f increases, our compiler exhibits more adaptivity by being able to execute on smaller 2D sizes. These results are obtained by compiling each program to a progressively smaller 2D size for each D_f, until the compilation fails due to node congestion. The vertical dashed lines in Fig. 5.10 correspond to the standard 2D size of 8×8. When $D_f = 10$, the 2D size of these 64-qubit programs can be reduced to as small as 3×3, which is 7× smaller than the standard size.



(a)



(b)



(c)

Figure 5.10. Effects of 2D size on 64-qubit programs when $D_f = 1$ (a), $D_f = 5$ (b) and $D_f = 10$ (c). Dashed lines correspond to the standard 2D size, which is $S = 8 \times 8$ for 64-qubit programs.

Adaptive to IR Support. Our framework can be applied to architectures with and without support of skewed IR edges while maintaining its capability of restricting temporal edge lengths. We demonstrate the effect of this IR support by enabling and disabling skewed edges in the IR, applying our 2D-bounded temporal routing and its adapted version introduced in Section 5.4.4 for the two cases, respectively.

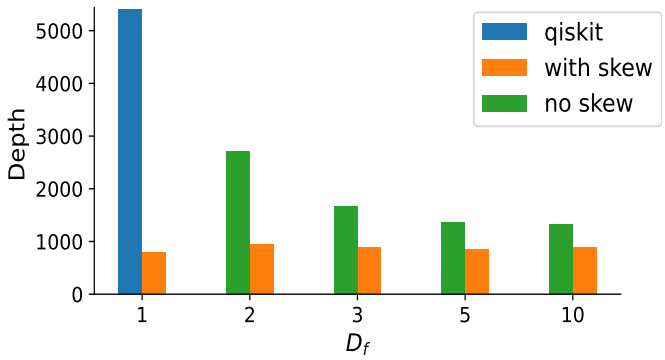


Figure 5.11. 1D depth of Qiskit when $D_f = 1$, and our 1D depth on IRs with and without skewed edges under varying D_f.

Fig. 5.11 shows the average 1D depths of 64-qubit benchmarks on the standard 2D size with a varying D_f, with skewed edges enabled and disabled in the IR. The results show that even without skewed edges, the temporal edge lengths can still be restricted to a limit as low as $D_f = 2$, with the 1D depth decreasing as D_f increases. With skewed edges, temporal edges can be further restricted to $D_f = 1$, with a lower 1D depth than that without skewed edges. In this case, the 1D depth first increases as D_f increases from 1 to 2, and then becomes stable for larger D_f since mapping and routing do not rely heavily on node storage when the IR is more flexible. The reason why the 1D depth is smaller at $D_f = 1$ can be because, when the 2D size is allowed to be the standard size, D_f =1 forces that every circuit-model qubit has a corresponding graph state node mapped onto every 2D layer, ensuring that all qubits participate in computation and thereby accelerating computation progress.

5.5.4 More Analysis

Offline Compilation. Our compiler exhibits a shorter compilation time than OnePerc, primarily because its reduction in 1D depth reduces the number of layers that the compiler needs to search. As shown in Table 5.3, when compiling 36-qubit programs onto a 6×6 IR, our framework reduces the offline compilation time by an average factor of 1.66.

Runtime Overhead. We evaluate the runtime overhead of our compiler with a varying p

Table 5.3. 1D depth, compilation time and runtime overhead varying with p, when $D_f = 20$.

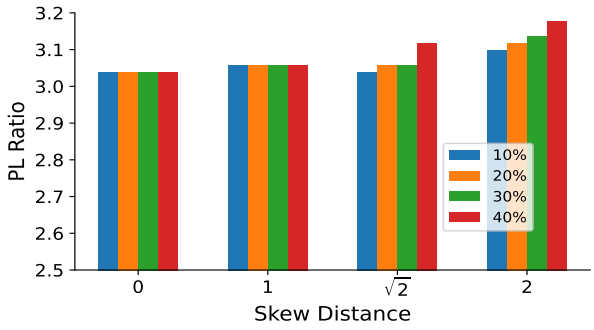
p	Benchmark -#Qubit	OnePerc 1D Depth	Our 1D Depth	Improv.	OnePerc Runtime (s/layer)	Our Runtime (s/layer)	Improv.	OnePerc Offline Comp. Time (s)	Our Offline Comp. Time (s)	Improv.
0.2	QAOA-36	1,150	406	2.83	0.07	0.06	1.12	18.14	10.7	1.7
	RCA-36	1,697	712	2.38	0.08	0.06	1.31	65.71	47.14	1.39
	VQE-36	916	292	3.14	0.07	0.06	1.2	16.13	9.24	1.75
0.4	QAOA-36	1,150	322	3.57	0.07	0.09	0.8	18.14	9.95	1.82
	RCA-36	1,697	702	2.42	0.08	0.08	1.05	65.71	36.74	1.79
	VQE-36	916	256	3.58	0.07	0.08	0.88	16.13	7.95	2.03
0.6	QAOA-36	1,150	399	2.88	0.07	0.1	0.73	18.14	9.98	1.82
	RCA-36	1,697	931	1.82	0.08	0.11	0.74	65.71	40.25	1.63
	VQE-36	916	354	2.59	0.07	0.09	0.76	16.13	8.23	1.96

in equation 5.1. We compile 36-qubit programs onto a 6×6 IR, renormalized from 144×144 RSLs with a fusion success probability of 75%. Table 5.3 shows the average runtime per RSL of OnePerc and our compiler. Note that the total runtime is not particularly meaningful, as our compiler is written in Python and not optimized for speed. A more informative measure is the runtime relative to OnePerc. When $p = 0.2$, our compiler has a similar runtime with OnePerc while achieving a significantly lower 1D depth. This runtime increases with p, because a larger p allows more computation-driven refresh on each 2D layer, thereby increasing the density of temporal edges among 2D layers.

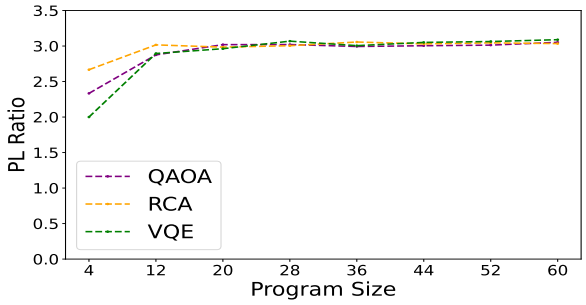
Choice of Refreshing Bound. An optimized 1D depth requires a suitable p in dynamic refresh. We compile 36-qubit programs onto a 6×6 IR with a varying p. Table 5.3 shows that as p increases, the 1D depth first decreases and then increases, with $p = 0.4$ being a sweet spot. This is because an excessively small p limits the computation progress with a restricted computation-driven refresh, while an excessively large p slows down the computation by preventing new nodes from being mapped.

5.5.5 Effect and Feasibility of Skewed Edges

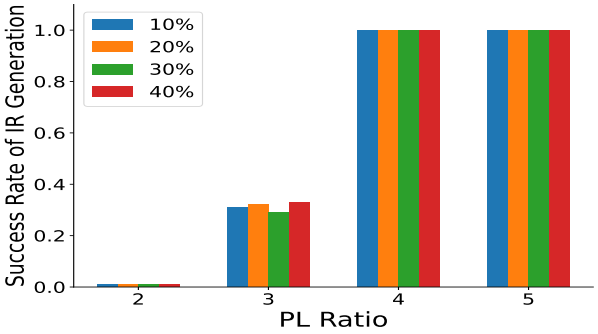
Negligible Effect on PL Ratio. We evaluate the PL ratio of a varying skew distance for temporal edges. As a preliminary estimation, we take an extended IR with 6×6 2D size, randomly selecting 10% - 40% of all possible skewed edges between 2D layers, skewing them in



(a)



(b)



(c)

Figure 5.12. Effects of IR extension.

random directions for a random 2D distance within the allowed range. Fig. 5.12(a) shows that when the skew distance is only 1, the PL ratio is about the same as that of FlexLattice IR, which corresponds to skew distance of 0.

As a result, our extended IR restricts the 2D distance of skewed edges to only 1. To illustrate with real programs, we further compile our benchmarks of various sizes to the extended IR with a 2D size of 6×6. Fig. 5.12(b) shows that the PL ratio first increases with the program

size and then becomes stable at a value ~ 3.1. This aligns with the result of PL ratio ~ 3.1 for FlexLattice IR [234], implying that the introduction of skewed edges in our new IR incurs a negligible resource overhead.

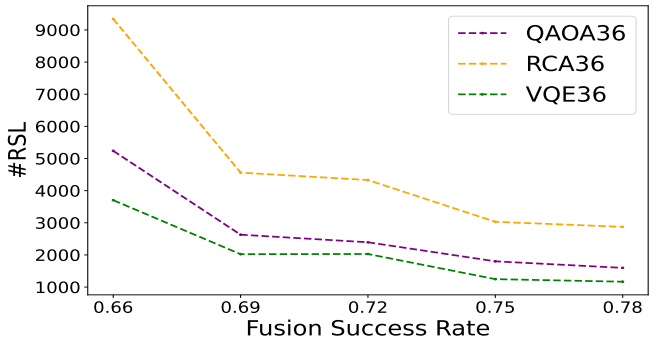


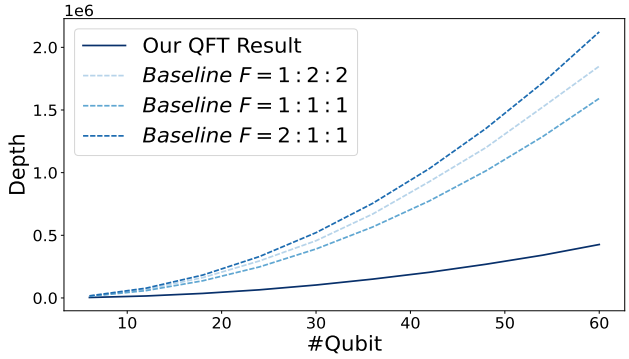
Figure 5.13. Tolerance of fusion failures.

Feasibility under Probabilistic Fusion. We demonstrate the probability of successfully generating IR programs when the PL ratio is fixed to a chosen value, with the same random programs in Fig. 5.12(a). Fig. 5.12(c) shows that when PL ratio is fixed to a small value, the IR generation fails with a high probability. However, when it reaches a threshold of 4, the IR program can be generated with a success probability of ~ 1 via percolation.

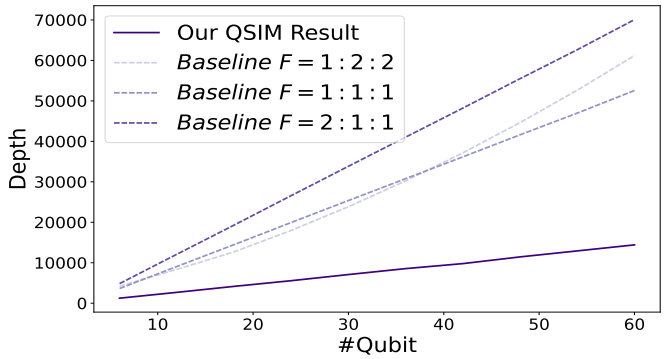
Moreover, our extended IR implementation does not reduce the tolerance for fusion failure rate when compared to OnePerc [234]. Fig. 5.13 shows the number of RSLs when executing 36-qubit programs under different fusion success probabilities. The results show that our compiler can allow a fusion success probability as low as 0.66, with the number of RSLs decreasing with a higher fusion success probability. This is consistent with the results of allowing 0.66 fusion success probability in OnePerc [234].

5.5.6 Incorporation of QEC

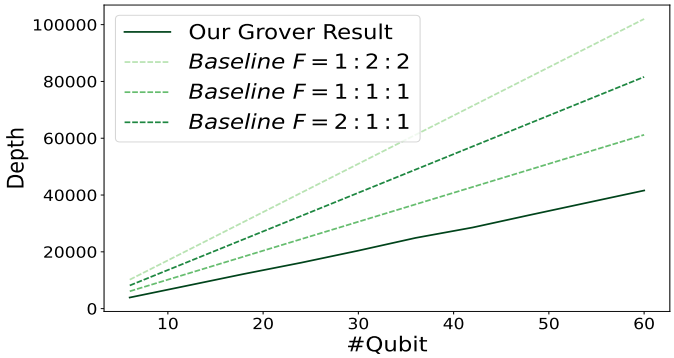
Our compiler can also play an important role in FTQC. We demonstrate its effect in FTQC by comparing to a static strategy that interleaves QEC patches uniformly over different RSLs, as illustrated in Fig. 5.5(b). We demonstrate with surface code as it is a leading QEC



(a)



(b)



(c)

Figure 5.14. Effect of dynamic refresh when QEC is incorporated.

code with relatively high error threshold and well-known universal gate implementation [90]. Its implementation on MBQC has also been well studied [178, 110, 40]. We leave the incorporation of other codes potentially suitable for photonic platforms (e.g., qLDPC code [209, 31] and Floquette code [105]) to future work.

We adopt the existing scheme of integrating QEC into MBQC, which transforms implementation of universal logical operations in the circuit model to measurement patterns on a Raussendorf cluster state [178, 110, 40]. We set the code distance to $d = 20$, according to prior simulations for fusion-based architectures [40]. Specifically, we first decompose programs in the circuit model into the Clifford + T basis [159], adopting a transversal implementation of logical single-qubit Clifford gates with subsequent deformation [113], an implementation of logical CNOT gates through lattice surgery with ancilla patches [113], and an implementation of logical T gates with magic state distillation [91, 34]. Then, we transform these logical operators into MBQC, with code patches of different logical qubits allocated in an interleaving manner shown in Fig. 5.5(b). This results in a resource overhead of $(N_a + N_m)/N_l$, with N_l, N_a, N_m being the number of code patches allocated for algorithmic qubits, ancilla qubits and magic state distillation, respectively. Proper resource allocation is crucial for minimizing depth, as insufficient N_a or N_m increases the depth by blocking logical gates, while excessive N_a or N_m increases the depth by occupying too many RSLs. Hence in the baseline, we vary the ratio $F = N_l : N_a : N_m$ from F=2:1:1 to F=1:1:1 and F=1:2:2. To enable fault-tolerance, d rounds of syndrome extraction are conducted for each logical operation.

The dynamic refresh mechanism can be easily extended to integrate with QEC, by replacing physical qubits in NISQ with logical blocks in FTQC. We allocate RSLs for different logical qubits based on two factors: their storage time in delay lines and whether they are involved in logical operations. Specifically, we divide logical qubits into active and inactive ones, with the active ones including qubits currently under logical operations such as lattice surgery and magic state distillation. The dynamic refresh is implemented as the following: (1) Algorithmic logical qubits (both active and inactive) and active logical qubits of other types have to be refreshed within every D_f layers, with D_f depending on the noise level in the delay lines and the number of delay lines available on the hardware. (2) When refresh is not needed, RSLs are allocated only to active logical qubits, with the allocation prioritized for logical qubits with longer storage time in delay lines.

Reduction in 1D Depth. Fig. 5.14 shows the number of consumed RSLs across benchmarks of varying sizes, with and without dynamic refresh. As the ratio F of different logical qubit types increases, the baseline circuit depths first decrease and then increase, reaching a minimum at the 1:1:1 configuration. It can be seen that even compared against the best-performing baseline configuration, our technique achieves an average depth reduction of 3.33×.

Effect of Refresh Period. Fig. 5.15 demonstrates the adaptivity of our dynamic refresh to a varying refresh period. For 30-qubit benchmarks, the baseline requires a refresh period D_f of at least 90 to operate. In contrast, Fig. 5.15 shows that our dynamic refresh remains effective at significantly lower D_f values around 40, with a moderate increase in 1D depth as D_f decreases.

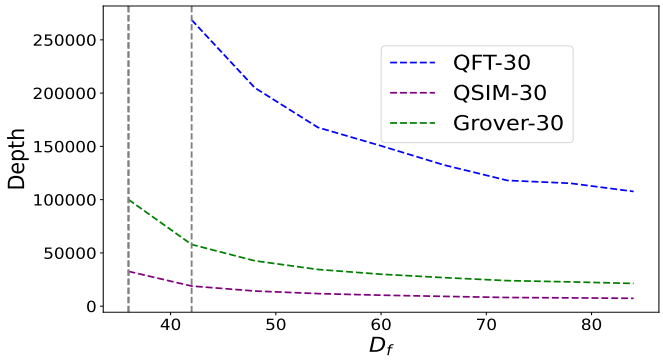


Figure 5.15. Effects of refresh period limit on QEC depth.

5.6 Conclusion

In this work, we present a resource-adaptive compiler of measurement-based quantum computing for photonic hardware. At its core is a novel intermediate representation (IR), which enables two key optimization passes: dynamic refresh and 2D-bounded temporal routing. Our evaluation shows that, compared to FlexLattice IR compilers, our compiler reduces the 1D depth by 3.48× while constraining temporal edge lengths within an adaptive limit (as low as 1). Compared to cluster state IR compilers, it can achieve a 6.7× 1D depth reduction or significantly reduce the 2D size (e.g., from 8×8 to 3×3 for 64-qubit programs). Additionally, when applied to FTQC with surface code, the dynamic refresh reduces the 1D depth by a factor of 3.33×.

5.7 Acknowledgments

This chapter is a full reprint of material published as: Hezi Zhang, Jixuan Ruan, Dean Tullsen, Yufei Ding, Ang Li, and Travis Humble, "OneAdapt: Resource-Adaptive Compilation of Measurement-Based Quantum Computing for Photonic Hardware" published in Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture. The dissertation author is the primary investigator and author of this paper.

Bibliography

[1] Ibm quantum. https://quantum-computing.ibm.com.

[2] Mqt qmap - a tool for quantum circuit mapping written in c++. https://github.com/cda-tum/mqt-qmap.

[3] Exponential suppression of bit or phase errors with cyclic error correction. *Nature*, 595(7867):383–387, 2021.

[4] H Aghaee Rad, T Ainsworth, RN Alexander, B Altieri, MF Askarani, R Baby, L Banchi, BQ Baragiola, JE Bourassa, RS Chadwick, et al. Scaling and networking a modular photonic quantum computer. *Nature*, pages 1–8, 2025.

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, 38(4):63–74, 2008.

[6] Koen Alexander, Andrea Bahgat, Avishai Benyamini, Dylan Black, Damien Bonneau, Stanley Burgos, Ben Burridge, Geoff Campbell, Gabriel Catalano, Alex Ceballos, et al. A manufacturable platform for photonic quantum computing. *arXiv preprint arXiv:2404.17570*, 2024.

[7] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key {Exchange—A} new hope. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 327–343, 2016.

[8] Max Alteg, Baptiste Chevalier, Octave Mestoudjian, and Johan-Luca Rossi. Study of adaptative derivative-assemble pseudo-trotter ansatzes in vqe through qiskit api. *arXiv preprint arXiv:2210.15438*, 2022.

[9] Matthew Amy and Vlad Gheorghiu. staq - a full-stack quantum processing toolkit. *Quantum Science and Technology*, 5, 05 2020.

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

[11] Pablo Andres-Martinez and Chris Heunen. Automated distribution of quantum circuits via hypergraph partitioning. *Physical Review A*, 100(3):032308, 2019.

[12] James Ang, Gabriella Carini, Yanzhu Chen, Isaac Chuang, Michael Demarco, Sophia Economou, Alec Eickbusch, Andrei Faraon, Kai-Mei Fu, Steven Girvin, et al. Arquin: architectures for multinode superconducting quantum computers. *ACM Transactions on Quantum Computing*, 5(3):1–59, 2024.

[13] James Ang, Gabriella Carini, Yanzhu Chen, Isaac Chuang, Michael Austin DeMarco, Sophia E Economou, Alec Eickbusch, Andrei Faraon, Kai-Mei Fu, Steven M Girvin, et al. Architectures for multinode superconducting quantum computers. 2022.

[14] J. Arrazola, Ville Bergholm, K. Brádler, T. Bromley, M. Collins, I. Dhand, A. Fumagalli, T. Gerrits, A. Goussev, Lukas Helt, J. Hundal, T. Isacsson, Robert Israel, J. Izaac, S. Jahangiri, R. Janik, N. Killoran, S. Kumar, J. Lavoie, and Y. Zhang. Quantum circuits with many photons on a programmable nanophotonic chip. *Nature*, 591:54–60, 03 2021.

[15] JM Arrazola, V Bergholm, K Brádler, TR Bromley, MJ Collins, I Dhand, A Fumagalli, T Gerrits, A Goussev, LG Helt, et al. Quantum circuits with many photons on a programmable nanophotonic chip. *Nature*, 591(7848):54–60, 2021.

[16] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando Brandao, David Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, and John Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574:505–510, 10 2019.

[17] Alán Aspuru-Guzik, Anthony D Dutoi, Peter J Love, and Martin Head-Gordon. Simulated quantum computation of molecular energies. *Science*, 309(5741):1704–1707, 2005.

[18] David Awschalom, Karl Berggren, Hannes Bernien, Sunil Bhave, Lincoln Carr, Paul Davids, Sophia Economou, Dirk Englund, Andrei Faraon, Martin Fejer, Saikat Guha, Martin Gustafsson, Evelyn Hu, Liang Jiang, Jungsang Kim, Boris Korzh, Prem Kumar, Paul Kwiat, Marko Loncar, and Zheshen Zhang. Development of quantum interconnects (quics) for next-generation information technologies. *PRX Quantum*, 2, 02 2021.

[19] David Awschalom, Karl K Berggren, Hannes Bernien, Sunil Bhave, Lincoln D Carr, Paul Davids, Sophia E Economou, Dirk Englund, Andrei Faraon, Martin Fejer, et al. Development of quantum interconnects (quics) for next-generation information technologies. *Prx Quantum*, 2(1):017002, 2021.

[20] Koji Azuma, Sophia E Economou, David Elkouss, Paul Hilaire, Liang Jiang, Hoi-Kwong Lo, and Ilan Tzitrin. Quantum repeaters: From quantum networks to the quantum internet. *arXiv preprint arXiv:2212.10820*, 2022.

[21] Jonathan Baker, Casey Duckering, Alexander Hoover, and Frederic Chong. Time-sliced quantum circuit partitioning for modular architectures. pages 98–107, 05 2020.

[22] Jonathan M Baker, Casey Duckering, Alexander Hoover, and Frederic T Chong. Time-sliced quantum circuit partitioning for modular architectures. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*, pages 98–107, 2020.

[23] Konrad Banaszek, Alfred B U'Ren, and Ian A Walmsley. Generation of correlated photons in controlled spatial modes by downconversion in nonlinear waveguides. *Optics letters*, 26(17):1367–1369, 2001.

[24] Sara Bartolucci, Patrick Birchall, Hector Bombin, Hugo Cable, Chris Dawson, Mercedes Gimeno-Segovia, Eric Johnston, Konrad Kieling, Naomi Nickerson, Mihir Pant, Fernando Pastawski, Terry Rudolph, and Chris Sparrow. Fusion-based quantum computation. *arXiv preprint arXiv:2101.09310*, 2021.

[25] Sara Bartolucci, Patrick Birchall, Hector Bombin, Hugo Cable, Chris Dawson, Mercedes Gimeno-Segovia, Eric Johnston, Konrad Kieling, Naomi Nickerson, Mihir Pant, Fernando Pastawski, Terry Rudolph, and Chris Sparrow. Fusion-based quantum computation. *arXiv preprint 2101.09310*, 2021.

[26] Sara Bartolucci, Patrick Birchall, Damien Bonneau, Hugo Cable, Mercedes Gimeno-Segovia, Konrad Kieling, Naomi Nickerson, Terry Rudolph, and Chris Sparrow. Switch networks for photonic fusion-based quantum computing. *arXiv preprint arXiv:2109.13760*, 2021.

[27] Robert Beals, Stephen Brierley, Oliver Gray, Aram W Harrow, Samuel Kutin, Noah Linden, Dan Shepherd, and Mark Stather. Efficient distributed quantum computing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 469(2153):20120686, 2013.

[28] Charles H Bennett, Gilles Brassard, Sandu Popescu, Benjamin Schumacher, John A Smolin, and William K Wootters. Purification of noisy entanglement and faithful teleportation via noisy channels. *Physical review letters*, 76(5):722, 1996.

[29] Eric Bersin, Matthew Grein, Madison Sutula, Ryan Murphy, Yan Qi Huan, Mark Stevens, Aziza Suleymanzade, Catherine Lee, Ralf Riedinger, David J Starling, et al. Development of a boston-area 50-km fiber quantum network testbed. *Physical Review Applied*, 21(1):014024, 2024.

[30] Eric Bersin, Madison Sutula, Yan Qi Huan, Aziza Suleymanzade, Daniel R Assumpcao, Yan-Cheng Wei, Pieter-Jan Stas, Can M Knaut, Erik N Knall, Carsten Langrock, et al. Telecom networking with a diamond quantum memory. *PRX Quantum*, 5(1):010303, 2024.

[31] Noah Berthusen and Daniel Gottesman. Partial syndrome measurement for hypergraph product codes. *Quantum*, 8:1345, 2024.

[32] Jean-Claude Besse, Kevin Reuer, Michele C Collodo, Arne Wulff, Lucien Wernli, Adrian Copetudo, Daniel Malz, Paul Magnard, Abdulkadir Akin, Mihai Gabureac, et al. Realizing a deterministic source of multipartite-entangled photonic qubits. *Nature communications*, 11(1):4877, 2020.

[33] Hans KC Beukers, Matteo Pasini, Hyeongrak Choi, Dirk Englund, Ronald Hanson, and Johannes Borregaard. Tutorial: Remote entanglement protocols for stationary qubits with photonic interfaces. *arXiv preprint arXiv:2310.19878*, 2023.

[34] Michael Beverland, Vadym Kliuchnikov, and Eddie Schoute. Surface code compilation via edge-disjoint paths. *PRX Quantum*, 3(2):020342, 2022.

[35] Michael E Beverland, Prakash Murali, Matthias Troyer, Krysta M Svore, Torsten Hoefler, Vadym Kliuchnikov, Guang Hao Low, Mathias Soeken, Aarthi Sundaram, and Alexander Vaschillo. Assessing requirements to scale to practical quantum advantage. *arXiv preprint arXiv:2211.07629*, 2022.

[36] Dolev Bluvstein, Simon J Evered, Alexandra A Geim, Sophie H Li, Hengyun Zhou, Tom Manovitz, Sepehr Ebadi, Madelyn Cain, Marcin Kalinowski, Dominik Hangleiter, et al. Logical quantum processor based on reconfigurable atom arrays. *Nature*, 626(7997):58–65, 2024.

[37] S. Bogdanov, M. Y. Shalaginov, A. Boltasseva, and V. M. Shalaev. Material platforms for integrated quantum photonics. *Opt. Mater. Express*, 7(1):111–132, Jan 2017.

[38] A Bolt, G Duclos-Cianci, D Poulin, and TM Stace. Foliated quantum error-correcting codes. *Physical review letters*, 117(7):070501, 2016.

[39] H Bombin, IH Kim, D Litinski, N Nickerson, M Pant, F Pastawski, S Roberts, and T Rudolph. Interleaving: Modular architectures for fault-tolerant photonic quantum computing (2021). *arXiv preprint arXiv:2103.08612*.

[40] Hector Bombin, Chris Dawson, Ryan V Mishmash, Naomi Nickerson, Fernando Pastawski, and Sam Roberts. Logical blocks for fault-tolerant topological quantum computation. *PRX Quantum*, 4(2):020303, 2023.

[41] Hector Bombin, Isaac H Kim, Daniel Litinski, Naomi Nickerson, Mihir Pant, Fernando Pastawski, Sam Roberts, and Terry Rudolph. Interleaving: Modular architectures for fault-tolerant photonic quantum computing. *arXiv preprint arXiv:2103.08612*, 2021.

[42] J Eli Bourassa, Rafael N Alexander, Michael Vasmer, Ashlesha Patil, Ilan Tzitrin, Takaya Matsuura, Daiqin Su, Ben Q Baragiola, Saikat Guha, Guillaume Dauphinais, et al. Blueprint for a scalable photonic fault-tolerant quantum computer. *Quantum*, 5:392, 2021.

[43] Sergey Bravyi, Andrew W Cross, Jay M Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627(8005):778–782, 2024.

[44] Teresa Brecht, Wolfgang Pfaff, Chen Wang, Luigi Frunzio, Michel Devoret, and Robert Schoelkopf. Multilayer microwave integrated quantum circuits for scalable quantum computing. *npj Quantum Information*, 2, 09 2015.

[45] H-J Briegel, Wolfgang Dür, Juan I Cirac, and Peter Zoller. Quantum repeaters: the role of imperfect local operations in quantum communication. *Physical Review Letters*, 81(26):5932, 1998.

[46] Hans J Briegel, David E Browne, Wolfgang Dür, Robert Raussendorf, and Maarten Van den Nest. Measurement-based quantum computation. *Nature Physics*, 5(1):19–26, 2009.

[47] Hans J Briegel and Robert Raussendorf. A one-way quantum computer. *Physical Review Letters*, 86(22), 2003.

[48] Anne Broadbent and Elham Kashefi. Parallelizing quantum circuits. *Theoretical computer science*, 410(26):2489–2510, 2009.

[49] Daniel E Browne, Matthew B Elliott, Steven T Flammia, Seth T Merkel, Akimasa Miyake, and Anthony J Short. Phase transition of computational power in the resource states for one-way quantum computation. *New Journal of Physics*, 10(2):023010, 2008.

[50] 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.

[51] Angela Sara Cacciapuoti, Marcello Caleffi, Francesco Tafuri, Francesco Saverio Cataliotti, Stefano Gherardini, and Giuseppe Bianchi. Quantum internet: Networking challenges in distributed quantum computing. *IEEE Network*, 34(1):137–143, 2019.

[52] Marcello Caleffi, Michele Amoretti, Davide Ferrari, Daniele Cuomo, Jessica Illiano, Antonio Manzalini, and Angela Sara Cacciapuoti. Distributed quantum computing: a survey. *arXiv:2212.10609*, 2022.

[53] Yudong Cao, Jonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Mária Kieferová, Ian D. Kivlichan, Tim Menke, Borja Peropadre, Nicolas P.D. Sawaya, Sukin Sim, Libor Veis, and Alán Aspuru-Guzik. Quantum Chemistry in the Age of Quantum Computing. *Chemical Reviews*, 119:10856–10915, 2019.

[54] Jacques Carolan, Christopher Harrold, Chris Sparrow, Enrique Martín-López, Nicholas J Russell, Joshua W Silverstone, Peter J Shadbolt, Nobuyuki Matsuda, Manabu Oguma, Mikitaka Itoh, et al. Universal linear optics. *Science*, 349(6249):711–716, 2015.

[55] Liangyu Chen, Hang-Xi Li, Yong Lu, Christopher Warren, Christian Križan, Sandoko Kosen, Marcus Rommel, Shahnawaz Ahmed, Amr Osman, Janka Biznárová, Anita Fadavi, Benjamin Lienhard, Marco Caputo, Kestutis Grigoras, Leif Grönberg, Joonas Govenius, Anton Kockum, Per Delsing, Jonas Bylander, and Giovanna Tancredi. Transmon qubit readout fidelity at the threshold for quantum error correction without a quantum-limited amplifier. *npj Quantum Information*, 9, 03 2023.

[56] Yu-Fang Chen, Kai-Min Chung, Ondřej Lengál, Jyun-Ao Lin, Wei-Lun Tsai, and Di-De Yen. An automata-based framework for verification and bug hunting in quantum circuits (technical report). *arXiv preprint arXiv:2301.07747*, 2023.

[57] Hyeongrak Choi, Marc G Davis, Álvaro G Iñesta, and Dirk R Englund. Scalable quantum networks: Congestion-free hierarchical entanglement routing with error correction. *arXiv preprint arXiv:2306.09216*, 2023.

[58] Hyeongrak Choi, Mihir Pant, Saikat Guha, and Dirk Englund. Percolation-based architecture for cluster state creation using photon-mediated entanglement between atomic memories. *npj Quantum Information*, 5(1):104, 2019.

[59] Kevin Chou, Jacob Blumoff, Christopher Wang, Philip Reinhold, Christopher Axline, Yvonne Gao, L. Frunzio, M. Devoret, Liang Jiang, and R. Schoelkopf. Deterministic teleportation of a quantum gate between two logical qubits. *Nature*, 561, 09 2018.

[60] Jerry Chow. Quantum intranet. *IET Quantum Communication*, 2:26–27, 04 2021.

[61] A. Clerk, K. Lehnert, P. Bertet, Renato Petta, and Y. Nakamura. Hybrid quantum systems with circuit quantum electrodynamics. *Nature Physics*, 16:1–11, 03 2020.

[62] Dan Cogan, Zu-En Su, Oded Kenneth, and David Gershoni. Deterministic generation of indistinguishable photons in a cluster state. *Nature Photonics*, 17(4):324–329, 2023.

[63] N Coste, DA Fioretto, N Belabas, SC Wein, P Hilaire, R Frantzeskakis, M Gundin, B Goes, N Somaschi, M Morassi, et al. High-rate entanglement between a semiconductor spin and indistinguishable photons. *Nature Photonics*, 17(7):582–587, 2023.

[64] Steven A Cuccaro, Thomas G Draper, Samuel A Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit. *arXiv preprint quant-ph/0410184*, 2004.

[65] Daniele Cuomo, Marcello Caleffi, and Angela Sara Cacciapuoti. Towards a distributed quantum computing ecosystem. *IET Quantum Communication*, 1(1):3–8, 2020.

[66] A. Córcoles, Maika Takita, Ken Inoue, Scott Lekuch, Zlatko Minev, Jerry Chow, and Jay Gambetta. Exploiting dynamic quantum circuits in a quantum algorithm with superconducting qubits. *Physical Review Letters*, 127, 08 2021.

[67] 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.

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

[69] Axel Dahlberg, Matthew Skrzypczyk, Tim Coopmans, Leon Wubben, Filip Rozpedek, Matteo Pompili, Arian Stolk, Przemysław Pawełczak, Robert Knegjens, Julio de Oliveira Filho, Ronald Hanson, and Stephanie Wehner. A Link Layer Protocol for Quantum Networks. *arXiv e-prints*, page arXiv:1903.09778, March 2019.

[70] Vincent Danos and Elham Kashefi. Determinism in the one-way model. *Physical Review A—Atomic, Molecular, and Optical Physics*, 74(5):052310, 2006.

[71] Vincent Danos, Elham Kashefi, and Prakash Panangaden. The measurement calculus. *Journal of the ACM (JACM)*, 54(2):8–es, 2007.

[72] Zohreh Davarzani, Mariam Zomorodi-Moghadam, Mahboobeh Houshmand, and Mostafa Nouri-Baygi. A dynamic programming approach for distributing quantum circuits by bipartite graphs. *Quantum Information Processing*, 19:1–18, 2020.

[73] Jarn de Jong, Frederik Hahn, Nikolay Tcholtchev, Manfred Hauswirth, and Anna Pappa. Extracting ghz states from linear cluster states. Technical report.

[74] C Delle Donne, M Iuliano, B Van Der Vecht, GM Ferreira, H Jirovská, TJW Van Der Steenhoven, A Dahlberg, M Skrzypczyk, D Fioretto, M Teller, et al. An operating system for executing applications on quantum network nodes. *Nature*, 639(8054):321–328, 2025.

[75] Michel H Devoret and Robert J Schoelkopf. Superconducting circuits for quantum information: an outlook. *Science*, 339(6124):1169–1174, 2013.

[76] Michel H Devoret, Andreas Wallraff, and John M Martinis. Superconducting qubits: A short review. 2004.

[77] Stephen Diadamo, Marco Ghibaudi, and James Cruise. Distributed quantum computing and network control for accelerated vqe. *IEEE Transactions on Quantum Engineering*, PP:1–1, 02 2021.

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

[79] Duncan Earl, K Karunaratne, Jason Schaake, Ryan Strum, Patrick Swingle, and Ryan Wilson. Architecture of a first-generation commercial quantum network. *arXiv preprint arXiv:2211.14871*, 2022.

[80] Sepehr Ebadi, Tout Wang, Harry Levine, Alexander Keesling, Giulia Semeghini, Ahmed Omran, Dolev Bluvstein, Rhine Samajdar, Hannes Pichler, Wen Wei Ho, Soonwon Choi, Subir Sachdev, Markus Greiner, Vladan Vuletic, and Mikhail Lukin. Quantum phases of matter on a 256-atom programmable quantum simulator. *Nature*, 595:227–232, 07 2021.

[81] Felix Eltes, Gerardo E Villarreal-Garcia, Daniele Caimi, Heinz Siegwart, Antonio A Gentile, Andy Hart, Pascal Stark, Graham D Marshall, Mark G Thompson, Jorge Barreto, et al. An integrated optical modulator operating at cryogenic temperatures. *Nature Materials*, 19(11):1164–1168, 2020.

[82] Simon J Evered, Dolev Bluvstein, Marcin Kalinowski, Sepehr Ebadi, Tom Manovitz, Hengyun Zhou, Sophie H Li, Alexandra A Geim, Tout T Wang, Nishad Maskara, et al. High-fidelity parallel entangling gates on a neutral-atom quantum computer. *Nature*, 622(7982):268–272, 2023.

[83] Fabian Ewert and Peter van Loock. 3/4-efficient bell measurement with passive linear optics and unentangled ancillae. *Physical review letters*, 113(14):140403, 2014.

[84] Fabian Ewert and Peter van Loock. 3/4-efficient bell measurement with passive linear optics and unentangled ancillae. *Physical review letters*, 113(14):140403, 2014.

[85] Davide Ferrari, Angela Sara Cacciapuoti, Michele Amoretti, and Marcello Caleffi. Compiler design for distributed quantum computing. *IEEE Transactions on Quantum Engineering*, PP:1–1, 01 2021.

[86] 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.

[87] Simone Ferrari, Carsten Schuck, and Wolfram Pernice. Waveguide-integrated superconducting nanowire single-photon detectors. *Nanophotonics*, 7(11):1725–1758, 2018.

[88] Vinicius S Ferreira, Gihwan Kim, Andreas Butler, Hannes Pichler, and Oskar Painter. Deterministic generation of multidimensional photonic cluster states with a single quantum emitter. *arXiv preprint arXiv:2206.10076*, 2022.

[89] Vinicius S Ferreira, Gihwan Kim, Andreas Butler, Hannes Pichler, and Oskar Painter. Deterministic generation of multidimensional photonic cluster states with a single quantum emitter. *Nature Physics*, 20(5):865–870, 2024.

[90] Austin Fowler, Matteo Mariantoni, John Martinis, and Andrew Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86, 08 2012.

[91] Austin G Fowler and Craig Gidney. Low overhead quantum computation using lattice surgery. *arXiv preprint arXiv:1808.06709*, 2018.

[92] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A—Atomic, Molecular, and Optical Physics*, 86(3):032324, 2012.

[93] Jay Gambetta. Expanding the ibm quantum roadmap to anticipate the future of quantum-centric supercomputing. https://research.ibm.com/blog/ibm-quantum-roadmap-2025.

[94] Scarlett Gauthier, Gayane Vardoyan, and Stephanie Wehner. A control architecture for entanglement generation switches in quantum networks. In *Proceedings of the 1st Workshop on Quantum Networks and Distributed Quantum Computing*, pages 38–44, 2023.

[95] Craig Gidney. Tetrationally compact entanglement purification. *arXiv preprint arXiv:2311.10971*, 2023.

[96] Mercedes Gimeno-Segovia, Pete Shadbolt, Dan E Browne, and Terry Rudolph. From three-photon ghz states to universal ballistic quantum computation. 2015.

[97] Mercedes Gimeno-Segovia, Pete Shadbolt, Dan E Browne, and Terry Rudolph. From three-photon ghz states to universal ballistic quantum computation. 2015.

[98] Alysson Gold, J. Paquette, Anna Stockklauser, Matthew Reagor, M. Alam, Andrew Bestwick, Nicolas Didier, Ani Nersisyan, Feyza Oruc, Armin Razavi, Ben Scharmann, Eyob Sete, Biswajit Sur, Davide Venturelli, Cody Winkleblack, Filip Wudarski, Mike Harburn, and Chad Rigetti. Entanglement across separate silicon dies in a modular superconducting qubit device. *npj Quantum Information*, 7:142, 09 2021.

[99] Warren P Grice. Arbitrarily complete bell-state measurement using only linear optical elements. *Physical Review A*, 84(4):042331, 2011.

[100] 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.

[101] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[102] Craig S Hamilton, Regina Kruse, Linda Sansoni, Sonja Barkhofen, Christine Silberhorn, and Igor Jex. Gaussian boson sampling. *Physical review letters*, 119(17):170501, 2017.

[103] Xu Han, Wei Fu, Chang-Ling Zou, Liang Jiang, and Hong Tang. Microwave-optical frequency conversion. *Optica*, 8, 06 2021.

[104] Thomas Häner, Damian S Steiger, Torsten Hoefler, and Matthias Troyer. Distributed quantum computing with qmpi. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2021.

[105] Matthew B Hastings and Jeongwan Haah. Dynamically generated logical qubits. *Quantum*, 5:564, 2021.

[106] Nico Hauser, Matthias J Bayerbach, Simone E D'Aurelio, Raphael Weber, Matteo Santandrea, Shreya P Kumar, Ish Dhand, and Stefanie Barz. Boosted bell-state measurements for photonic quantum computation. *npj Quantum Information*, 11(1):41, 2025.

[107] Marc Hein, Wolfgang Dür, Jens Eisert, Robert Raussendorf, M Nest, and H-J Briegel. Entanglement in graph states and its applications. *arXiv preprint quant-ph/0602096*, 2006.

[108] Marc Hein, Wolfgang Dür, Jens Eisert, Robert Raussendorf, Maarten Van den Nest, and H-J Briegel. Entanglement in graph states and its applications. In *Quantum computers, algorithms and chaos*, pages 115–218. IOS Press, 2006.

[109] B. Hensen, H. Bernien, A. Dréau, Andreas Reiserer, Norbert Kalb, M. Blok, J. Ruitenberg, R. Vermeulen, R. Schouten, Carlos Abellan, Waldimar Amaya, V. Pruneri, Morgan Mitchell, M. Markham, Daniel Twitchen, David Elkouss, Stephanie Wehner, Tim Taminiau, and R. Hanson. Loophole-free bell inequality violation using electron spins separated by 1.3 kilometres. *Nature*, 526, 10 2015.

[110] Daniel Herr, Alexandru Paler, Simon J Devitt, and Franco Nori. Lattice surgery on the raussendorf lattice. *Quantum Science and Technology*, 3(3):035011, 2018.

[111] Jared Hertzberg, Eric Zhang, Sami Rosenblatt, Easwar Magesan, John Smolin, Jeng-Bang Yau, Vivekananda Adiga, Martin Sandberg, Markus Brink, Jerry Chow, and Jason Orcutt. Laser-annealing josephson junctions for yielding scaled-up superconducting quantum processors. *npj Quantum Information*, 7:129, 08 2021.

[112] Stefan Hillmich, Alwin Zulehner, and Robert Wille. Exploiting quantum teleportation in quantum circuit mapping. pages 792–797, 01 2021.

[113] Dominic Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[114] N. Isailovic, Y. Patel, M. Whitney, and John Kubiatowicz. Interconnection networks for scalable quantum computers. volume 2006, pages 366–377, 02 2006.

[115] D Istrati, Y Pilnyak, JC Loredo, C Antón, N Somaschi, P Hilaire, H Ollivier, M Esmann, L Cohen, L Vidro, et al. Sequential generation of linear cluster states from a single photon emitter. *Nature communications*, 11(1):5501, 2020.

[116] Ziyue Jia and Lin Chen. On fidelity-oriented entanglement distribution for quantum switches. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2024.

[117] Petar Jurcevic, Ali Javadi-Abhari, Lev S Bishop, Isaac Lauer, Daniela F Bogorin, Markus Brink, Lauren Capelluto, Oktay Günlük, Toshinari Itoko, Naoki Kanazawa, et al. Demonstration of quantum volume 64 on a superconducting quantum computing system. *Quantum Science and Technology*, 6(2):025020, 2021.

[118] Harry Kesten et al. The critical probability of bond percolation on the square lattice equals 1/2. *Communications in mathematical physics*, 74(1):41–59, 1980.

[119] Nader Khammassi, Imran Ashraf, Hans van Someren, Razvan Nane, Anneriet Krol, M. Rol, Lingling Lao, Koen Bertels, and Carmen Almudever. Openql: A portable quantum programming framework for quantum accelerators. *ACM Journal on Emerging Technologies in Computing Systems*, 18:1–24, 01 2022.

[120] Nathan Killoran, Josh Izaac, Nicolás Quesada, Ville Bergholm, Matthew Amy, and Christian Weedbrook. Strawberry fields: A software platform for photonic quantum computing. *Quantum*, 3:129, 2019.

[121] Junpyo Kim, Dongmoon Min, Jungmin Cho, Hyeonseong Jeong, Ilkwon Byun, Junhyuk Choi, Juwon Hong, and Jangwoo Kim. A fault-tolerant million qubit-scale distributed quantum computer. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 1–19, 2024.

[122] Aleks Kissinger and John van de Wetering. Pyzx: Large scale automated diagrammatic reasoning. *arXiv preprint arXiv:1904.04735*, 2019.

[123] Can M Knaut, Aziza Suleymanzade, Y-C Wei, Daniel R Assumpcao, P-J Stas, Yan Qi Huan, Bartholomeus Machielse, Erik N Knall, Madison Sutula, Gefen Baranes, et al. Entanglement of nanophotonic quantum memory nodes in a telecom network. *Nature*, 629(8012):573–578, 2024.

[124] Jin Ming Koh, Shi-Ning Sun, Mario Motta, and Austin J Minnich. Measurement-induced entanglement phase transition on a superconducting quantum processor with mid-circuit readout. *Nature Physics*, 19(9):1314–1319, 2023.

[125] Pieter Kok, William J Munro, Kae Nemoto, Timothy C Ralph, Jonathan P Dowling, and Gerard J Milburn. Linear optical quantum computing with photonic qubits. *Reviews of modern physics*, 79(1):135, 2007.

[126] Pieter Kok, William J Munro, Kae Nemoto, Timothy C Ralph, Jonathan P Dowling, and Gerard J Milburn. Linear optical quantum computing with photonic qubits. *Reviews of modern physics*, 79(1):135, 2007.

[127] Sebastian Krinner, Simon Storz, Philipp Kurpiers, P. Magnard, Johannes Heinsoo, R. Keller, J. Lütolf, C. Eichler, and Andreas Wallraff. Engineering cryogenic setups for 100-qubit scale superconducting circuit systems. *EPJ Quantum Technology*, 6, 05 2019.

[128] Gershon Kurizki, P. Bertet, Yuimaru Kubo, Klaus Mølmer, David Petrosyan, Peter Rabl, and Jörg Schmiedmayer. Quantum technologies with hybrid systems. *Proceedings of the National Academy of Sciences*, 112:3866, 03 2015.

[129] Philipp Kurpiers, Paul Magnard, Theo Walter, Baptiste Royer, Marek Pechal, Johannes Heinsoo, Yves Salathé, Abdulkadir Akin, Simon Storz, Jean-Claude Besse, Simone Gasparinetti, Alexandre Blais, and Andreas Wallraff. Deterministic quantum state transfer and generation of remote entanglement using microwave photons. *Nature*, 558, 06 2018.

[130] Nicholas Lambert, Alfredo Rueda, Florian Sedlmeir, and Harald Schwefel. Coherent conversion between microwave and optical photons—an overview of physical implementations. *Advanced Quantum Technologies*, 3, 12 2019.

[131] N LaRacuente, KN Smith, P Imany, KL Silverman, and FT Chong. Modeling short-range microwave networks to scale superconducting quantum computation. 2023.

[132] Nicholas LaRacuente, Kaitlin N Smith, Poolad Imany, Kevin L Silverman, and Frederic T Chong. Modeling short-range microwave networks to scale superconducting quantum computation. *arXiv preprint arXiv:2201.08825*, 2022.

[133] Mikkel V Larsen, Xueshi Guo, Casper R Breum, Jonas S Neergaard-Nielsen, and Ulrik L Andersen. Deterministic multi-mode gates on a scalable photonic quantum computing platform. *Nature Physics*, 17(9):1018–1023, 2021.

[134] Nikolai Lauk, Neil Sinclair, Sh Barzanjeh, Jacob Covey, Mark Saffman, Maria Spiropulu, and Christoph Simon. Perspectives on quantum transduction. *Quantum Science and Technology*, 5, 02 2020.

[135] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation. *ACM Transactions on Quantum Computing*, 4(2):1–26, 2023.

[136] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. pages 1001–1014, 04 2019.

[137] Jian Li, Mingjun Wang, Kaiping Xue, Ruidong Li, Nenghai Yu, Qibin Sun, and Jun Lu. Fidelity-guaranteed entanglement routing in quantum networks. *IEEE Transactions on Communications*, 70(10):6748–6763, 2022.

[138] Jian Li, Mingjun Wang, Kaiping Xue, Ruidong Li, Nenghai Yu, Qibin Sun, and Jun Lu. Fidelity-guaranteed entanglement routing in quantum networks. *IEEE Transactions on Communications*, 70(10):6748–6763, 2022.

[139] Ming-Jun Li and Tetsuya Hayashi. Advances in low-loss, large-area, and multicore fibers. In *Optical Fiber Telecommunications VII*, pages 3–50. Elsevier, 2020.

[140] Yingheng Li, Aditya Pawar, Zewei Mo, Youtao Zhang, Jun Yang, and Xulong Tang. Fmcc: Flexible measurement-based quantum computation over cluster state. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, pages 111–126, 2024.

[141] Daniel Litinski. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum*, 3:128, 2019.

[142] Junyu Liu, Connor T Hann, and Liang Jiang. Data centers with quantum random access memory and quantum networks. *Physical Review A*, 108(3):032610, 2023.

[143] Junyu Liu and Liang Jiang. Quantum data center: Perspectives. *IEEE Network*, 2024.

[144] Daniel Llewellyn, Yunhong Ding, Imad Faruque, Stefano Paesani, Davide Bacco, Raffaele Santagati, Yan-Jun Qian, Yan Li, Yun-Feng Xiao, Marcus Huber, Mehul Malik, Gary Sinclair, Xiaoqi Zhou, Karsten Rottwitt, Jeremy O'Brien, John Rarity, Qihuang Gong, Leif Oxenlowe, Jianwei Wang, and Mark Thompson. Chip-to-chip quantum teleportation and multi-photon entanglement in silicon. *Nature Physics*, 16:1–6, 02 2020.

[145] Lars S. Madsen, Fabian Laudenbach, Mohsen Falamarzi. Askarani, Fabien Rortais, Trevor Vincent, Jacob F. F. Bulmer, Filippo M. Miatto, Leonhard Neuhaus, Lukas G. Helt, Matthew J. Collins, Adriana E. Lita, Thomas Gerrits, Sae Woo Nam, Varun D. Vaidya, Matteo Menotti, Ish Dhand, Zachary Vernon, Nicolás Quesada, and Jonathan Lavoie. Quantum computational advantage with a programmable photonic processor. *Nature*, 606(7912):75–81, Jun 2022.

[146] P. Magnard, Simon Storz, Philipp Kurpiers, J. Schär, F. Marxer, J. Lütolf, T. Walter, Jean-Claude Besse, Mihai Gabureac, K. Reuer, Abdulkadir Akin, B. Royer, A. Blais, and Andreas Wallraff. Microwave quantum link between superconducting circuits housed in spatially separated cryogenic systems. *Physical Review Letters*, 125, 12 2020.

[147] Stephane Mallat. A wavelet tour of signal processing, 1999.

[148] Dmitri Maslov, Gerhard W Dueck, D Michael Miller, and Camille Negrevergne. Quantum circuit simplification and level compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(3):436–444, 2008.

[149] Francesco Mazza, Marcello Caleffi, and Angela Sara Cacciapuoti. Quantum lan: On-demand network topology via two-colorable graph states. In *2024 International Conference on Quantum Communications, Networking, and Computing (QCNC)*, pages 127–134. IEEE, 2024.

[150] Francesco Mazza, Caitao Zhan, Joaquin Chung, Rajkumar Kettimuthu, Marcello Caleffi, and Angela Sara Cacciapuoti. Simulation of entanglement-enabled connectivity in qlans using sequence. *arXiv preprint arXiv:2411.11031*, 2024.

[151] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C Benjamin, and Xiao Yuan. Quantum computational chemistry. *Reviews of Modern Physics*, 92(1):015003, 2020.

[152] Mohammad Mirhosseini, Alp Sipahigil, Mahmoud Kalaee, and Oskar Painter. Superconducting qubit to optical photon transduction. *Nature*, 588:599–603, 12 2020.

[153] Christopher Monroe, Robert Raussendorf, Alex Ruthven, Kenneth R Brown, Peter Maunz, L-M Duan, and Jungsang Kim. Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects. *Physical Review A*, 89(2):022317, 2014.

[154] Gary Mooney, Gregory White, Charles Hill, and Lloyd Hollenberg. Whole-device entanglement in a 65-qubit superconducting quantum computer. *Advanced Quantum Technologies*, 4:2100061, 09 2021.

[155] Sam Morley-Short, Sara Bartolucci, Mercedes Gimeno-Segovia, Pete Shadbolt, Hugo Cable, and Terry Rudolph. Physical-depth architectural requirements for generating universal photonic cluster states. *Quantum Science and Technology*, 3(1):015005, 2017.

[156] William J. Munro, Koji Azuma, Kiyoshi Tamaki, and Kae Nemoto. Inside quantum repeaters. *IEEE Journal of Selected Topics in Quantum Electronics*, 21(3):78–90, 2015.

[157] Hartmut Neven. Suppressing quantum errors by scaling a surface code logical qubit. https://ai.googleblog.com/2023/02/suppressing-quantum-errors-by-scaling.html.

[158] Michael A Nielsen. Optical quantum computation using cluster states. *Physical review letters*, 93(4):040503, 2004.

[159] Michael A Nielsen and Isaac L Chuang. Quantum computation and quantum information. *Phys. Today*, 54(2):60, 2001.

[160] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[161] Crystal Noel, Pradeep Niroula, Daiwei Zhu, Andrew Risinger, Laird Egan, Debopriyo Biswas, Marko Cetina, Alexey V Gorshkov, Michael J Gullans, David A Huse, et al. Measurement-induced quantum phases realized in a trapped-ion quantum computer. *Nature Physics*, 18(7):760–764, 2022.

[162] Jeremy L. O'Brien, Akira Furusawa, and Jelena Vučković. Photonic quantum technologies. *Nature Photonics*, 3(12):687, 2009.

[163] Mark Oskin, Frederic Chong, Isaak Chuang, and John Kubiatowicz. Building quantum wires: The long and the short of it. volume 31, pages 374–385, 01 2003.

[164] Stefano Paesani, Yunhong Ding, Raffaele Santagati, Levon Chakhmakhchyan, Caterina Vigliar, Karsten Rottwitt, Leif K Oxenløwe, Jianwei Wang, Mark G Thompson, and Anthony Laing. Generation and sampling of quantum states of light in a silicon chip. *Nature Physics*, 15(9):925–929, 2019.

[165] Mihir Pant, Hari Krovi, Don Towsley, Leandros Tassiulas, Liang Jiang, Prithwish Basu, Dirk Englund, and Saikat Guha. Routing entanglement in the quantum internet. npj quantum information 5 (1): 1–9. *arXiv preprint arXiv:1708.07142*, 2019.

[166] Mihir Pant, Don Towsley, Dirk Englund, and Saikat Guha. Percolation thresholds for photonic quantum computing. *Nature communications*, 10(1):1–11, 2019.

[167] Mihir Pant, Don Towsley, Dirk Englund, and Saikat Guha. Percolation thresholds for photonic quantum computing. *Nature communications*, 10(1):1070, 2019.

[168] Ashlesha Patil, Mihir Pant, Dirk Englund, Don Towsley, and Saikat Guha. Entanglement generation in a quantum network at distance-independent rate. *npj Quantum Information*, 8(1):51, 2022.

[169] Maurizio Patrignani. On the complexity of orthogonal compaction. *Computational Geometry*, 19(1):47–67, 2001.

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

[171] Juan M Pino, Jennifer M Dreiling, Caroline Figgatt, John P Gaebler, Steven A Moses, MS Allman, CH Baldwin, Michael Foss-Feig, David Hayes, Karl Mayer, et al. Demonstration of the trapped-ion quantum ccd computer architecture. *Nature*, 592(7853):209–213, 2021.

[172] A. Pirker and W. Dür. A quantum network stack and protocols for reliable entanglement-based networks. *New Journal of Physics*, 21(3):033003, March 2019.

[173] Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023.

[174] Google AI Quantum, Collaborators*†, Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Sergio Boixo, Michael Broughton, Bob B Buckley, et al. Hartree-fock on a superconducting qubit quantum computer. *Science*, 369(6507):1084–1089, 2020.

[175] IBM Quantum. A high-fidelity, two-qubit cross-resonance gate using interference couplers. https://research.ibm.com/publications/a-high-fidelity-two-qubit-cross-resonance-gate-using-interference-couplers.

[176] Robert Raussendorf. Measurement-based quantum computation with cluster states. *International Journal of Quantum Information*, 7(06):1053–1203, 2009.

[177] Robert Raussendorf, Daniel E Browne, and Hans J Briegel. Measurement-based quantum computation on cluster states. *Physical review A*, 68(2):022312, 2003.

[178] Robert Raussendorf, Jim Harrington, and Kovid Goyal. A fault-tolerant one-way quantum computer. *Annals of physics*, 321(9):2242–2270, 2006.

[179] Martin Ringbauer, Michael Meth, Lukas Postler, Roman Stricker, Rainer Blatt, Philipp Schindler, and Thomas Monz. A universal qudit quantum processor with trapped ions. *Nature Physics*, 18(9):1053–1057, 2022.

[180] Terry Rudolph. Fusion based photonic quantum computing. In *APS March Meeting Abstracts*, volume 2022, pages D28–001, 2022.

[181] Lidia Ruiz-Perez and Juan Carlos Garcia-Escartin. Quantum arithmetic with the quantum fourier transform. *Quantum Information Processing*, 16:1–14, 2017.

[182] Mark Saffman. Quantum computing with atomic qubits and rydberg interactions: progress and challenges. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 49(20):202001, 2016.

[183] Uday Saha, James D Siverns, John Hannegan, Qudsia Quraishi, and Edo Waks. Low-noise quantum frequency conversion of photons from a trapped barium ion to the telecom o-band. *ACS Photonics*, 10(8):2861–2865, 2023.

[184] Atsushi Sakaguchi, Shunya Konno, Fumiya Hanamura, Warit Asavanant, Kan Takase, Hisashi Ogawa, Petr Marek, Radim Filip, Jun-ichi Yoshikawa, Elanor Huntington, et al. Nonlinear feedforward enabling quantum computation. *Nature Communications*, 14(1):3817, 2023.

[185] Kaoru Sanaka, Karin Kawahara, and Takahiro Kuga. New high-efficiency source of photon pairs for engineering quantum entanglement. *Physical Review Letters*, 86(24):5620, 2001.

[186] Rodrigo Santiago, Sergi Abadal, Carmen Almudéver, and Eduard Alarcón. Modelling short-range quantum teleportation for scalable multi-core quantum computing architectures. pages 1–7, 09 2021.

[187] Ido Schwartz, Dan Cogan, Emma R Schmidgall, Yaroslav Don, Liron Gantz, Oded Kenneth, Netanel H Lindner, and David Gershoni. Deterministic generation of a cluster state of entangled photons. *Science*, 354(6311):434–437, 2016.

[188] Peter J Shadbolt, Maria R Verde, Alberto Peruzzo, Alberto Politi, Anthony Laing, Mirko Lobino, Jonathan CF Matthews, Mark G Thompson, and Jeremy L O'Brien. Generating, manipulating and measuring entanglement and mixture with a reconfigurable photonic circuit. *Nature Photonics*, 6(1):45–49, 2012.

[189] Hassan Shapourian, Eneet Kaur, Troy Sewell, Jiapeng Zhao, Michael Kilzer, Ramana Kompella, and Reza Nejabati. Quantum data center infrastructures: A scalable architectural design perspective. *arXiv preprint arXiv:2501.05598*, 2025.

[190] Shouqian Shi and Chen Qian. Concurrent entanglement routing for quantum networks: Model and designs. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 62–75, 2020.

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

[192] 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*, 6(1):014003, 2020.

[193] 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*, 6, 04 2020.

[194] Sergei Slussarenko and Geoff J Pryde. Photonic quantum information processing: A concise review. *Applied Physics Reviews*, 6(4):041303, 2019.

[195] Kaitlin Smith, Gokul Ravi, Jonathan Baker, and Frederic Chong. Scaling superconducting quantum computers with chiplet architectures. pages 1092–1109, 10 2022.

[196] Lars Steffen, Yves Salathé, M Oppliger, Philipp Kurpiers, M Baur, Christian Lang, C Eichler, Gabriel Puebla-Hellmann, Arkady Fedorov, and Andreas Wallraff. Deterministic quantum teleportation with feed-forward in a solid state system. *Nature*, 500:319–22, 08 2013.

[197] Samuel Stein, Shifan Xu, Andrew W Cross, Theodore J Yoder, Ali Javadi-Abhari, Chenxu Liu, Kun Liu, Zeyuan Zhou, Charles Guinn, Yufei Ding, et al. Architectures for heterogeneous quantum error correction codes. *arXiv preprint arXiv:2411.03202*, 2024.

[198] Michal Stepanovsky. A comparative review of mems-based optical cross-connects for all-optical networks from the past to the present day. *IEEE Communications Surveys & Tutorials*, 21(3):2928–2946, 2019.

[199] L J Stephenson, D. P. Nadlinger, B. C. Nichol, Shuoming An, P. Drmota, Timothy G. Ballance, K. Thirumalai, Joseph Francis Goodwin, David M. Lucas, and Chris Ballance. High-rate, high-fidelity entanglement of qubits across an elementary quantum network. *Physical review letters*, 124 11:110501, 2019.

[200] HUBER SUHNER. "Polatis technology-directlight beam-steering all-optical switch". https://www.polatis.com/polatis-all-optical-switch-technology-lowestloss-highest-performancedirectlight-beam-steering.asp. Accessed: 2024.

[201] Shuntaro Takeda and Akira Furusawa. Toward large-scale fault-tolerant universal photonic quantum computing. *APL Photonics*, 4(6):060902, 2019.

[202] Mark S Tame, Bryn A Bell, Carlo Di Franco, William J Wadsworth, and John G Rarity. Experimental realization of a one-way quantum computer algorithm solving simon's problem. *Physical Review Letters*, 113(20):200501, 2014.

[203] 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.

[204] Sébastien Tanzilli, Hugues De Riedmatten, Wolfgang Tittel, Hugo Zbinden, Pascal Baldi, Marc De Micheli, Daniel Barry Ostrowsky, and Nicolas Gisin. Highly efficient photon-pair source using periodically poled lithium niobate waveguide. *Electronics Letters*, 37(1):26–28, 2001.

[205] Sébastien Tanzilli, Anthony Martin, Florian Kaiser, Marc P De Micheli, Olivier Alibart, and Daniel B Ostrowsky. On the genesis and evolution of integrated quantum optics. *Laser & Photonics Reviews*, 6(1):115–143, 2012.

[206] Jon Tate, Pall Beck, Peter Clemens, Santiago Freitas, Jeff Gatz, Michele Girola, Jason Gmitter, Holger Mueller, Ray O'Hanlon, Veerendra Para, et al. *IBM and Cisco: together for a world class data center*. IBM Redbooks, 2013.

[207] Philip Thomas, Leonardo Ruscio, Olivier Morin, and Gerhard Rempe. Efficient generation of entangled multiphoton graph states from a single atom. *Nature*, 608(7924):677–681, 2022.

[208] Philip Thomas, Leonardo Ruscio, Olivier Morin, and Gerhard Rempe. Fusion of deterministically generated photonic graph states. *Nature*, 629(8012):567–572, 2024.

[209] Jean-Pierre Tillich and Gilles Zémor. Quantum ldpc codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory*, 60(2):1193–1202, 2013.

[210] Teague Tomesh, Pranav Gokhale, Victory Omole, Gokul Subramanian Ravi, Kaitlin N Smith, Joshua Viszlai, Xin-Chuan Wu, Nikos Hardavellas, Margaret R Martonosi, and Frederic T Chong. Supermarq: A scalable quantum benchmark suite. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 587–603. IEEE, 2022.

[211] Ilan Tzitrin, Takaya Matsuura, Rafael N Alexander, Guillaume Dauphinais, J Eli Bourassa, Krishna K Sabapathy, Nicolas C Menicucci, and Ish Dhand. Fault-tolerant quantum computation with static linear optics. *PRX Quantum*, 2(4):040353, 2021.

[212] Giuseppe Vallone, Gaia Donati, Natalia Bruno, Andrea Chiuri, and Paolo Mataloni. Experimental realization of the deutsch-jozsa algorithm with a six-qubit cluster state. *Physical Review A*, 81(5):050302, 2010.

[213] John van de Wetering. Zx-calculus for the working quantum computer scientist. *arXiv preprint arXiv:2012.13966*, 2020.

[214] Maarten Van den Nest, Akimasa Miyake, Wolfgang Dür, and Hans J Briegel. Universal resources for measurement-based quantum computation. *Physical review letters*, 97(15):150504, 2006.

[215] Tim van Leent, Matthias Bock, Florian Fertig, Robert Garthoff, Sebastian Eppelt, Yiru Zhou, Pooja Malik, Matthias Seubert, Tobias Bauer, Wenjamin Rosenfeld, et al. Entangling single atoms over 33 km telecom fibre. *Nature*, 607(7917):69–73, 2022.

[216] Philip Walther, Kevin J Resch, Terry Rudolph, Emmanuel Schenck, Harald Weinfurter, Vlatko Vedral, Markus Aspelmeyer, and Anton Zeilinger. Experimental one-way quantum computing. *Nature*, 434(7030):169–176, 2005.

[217] Yong Wan, Daniel Kienzler, Stephen Erickson, Karl Mayer, Ting Tan, Jenny Wu, Hilma Vasconcelos, Scott Glancy, Emanuel Knill, David Wineland, A. Wilson, and Dietrich Leibfried. Quantum gate teleportation between separated qubits in a trapped-ion processor. *Science*, 364:875–878, 05 2019.

[218] Cheng Wang, Mian Zhang, Xi Chen, Maxime Bertrand, Amirhassan Shams-Ansari, Sethumadhavan Chandrasekhar, Peter Winzer, and Marko Lončar. Integrated lithium niobate electro-optic modulators operating at cmos-compatible voltages. *Nature*, 562(7725):101–104, 2018.

[219] Chenlu Wang, Xuegang Li, Huikai Xu, Zhiyuan Li, Junhua Wang, Zhen Yang, Zhenyu Mi, Xuehui Liang, Tang Su, Chuhong Yang, et al. Towards practical quantum computers: Transmon qubit with a lifetime approaching 0.5 milliseconds. *npj Quantum Information*, 8(1):3, 2022.

[220] Jianwei Wang, Stefano Paesani, Yunhong Ding, Raffaele Santagati, Paul Skrzypczyk, Alexia Salavrakos, Jordi Tura, Remigiusz Augusiak, Laura Mančinska, Davide Bacco, et al. Multidimensional quantum entanglement with large-scale integrated optics. *Science*, 360(6386):285–291, 2018.

[221] Jianwei Wang, Fabio Sciarrino, Anthony Laing, and Mark G Thompson. Integrated photonic quantum technologies. *Nature Photonics*, 14(5):273–284, 2020.

[222] George Watkins, Hoang Minh Nguyen, Keelan Watkins, Steven Pearce, Hoi-Kwan Lau, and Alexandru Paler. A high performance compiler for very large scale surface code computations. *Quantum*, 8:1354, 2024.

[223] Anbang Wu, Yufei Ding, and Ang Li. Qucomm: Optimizing collective communication for distributed quantum computing. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 479–493, 2023.

[224] Anbang Wu, Hezi Zhang, Gushu Li, Alireza Shabani, Yuan Xie, and Yufei Ding. Autocomm: A framework for enabling efficient communication in distributed quantum programs. pages 1027–1041, 10 2022.

[225] Anbang Wu, Hezi Zhang, Gushu Li, Alireza Shabani, Yuan Xie, and Yufei Ding. Autocomm: A framework for enabling efficient communication in distributed quantum programs. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1027–1041. IEEE, 2022.

[226] Yulin Wu, Wan-Su Bao, Sirui Cao, Fusheng Chen, Ming-cheng Chen, Xiawei Chen, Tung-Hsun Chung, Hui Deng, Yajie Du, Daojin Fan, Ming Gong, Cheng Guo, Guo Chu, Shaojun Guo, Lianchen Han, Linyin Hong, He-Liang Huang, Yong-Heng Huo, Liping Li, and Jian-Wei Pan. Strong quantum computational advantage using a superconducting quantum processor. *Physical Review Letters*, 127, 10 2021.

[227] Jonathan Wurtz, Alexei Bylinskii, Boris Braverman, Jesse Amato-Grill, Sergio H Cantu, Florian Huber, Alexander Lukin, Fangli Liu, Phillip Weinberg, John Long, et al. Aquila: Quera's 256-qubit neutral-atom quantum computer. *arXiv preprint arXiv:2306.11727*, 2023.

[228] Chao-Wei Yang, Yong Yu, Jun Li, Bo Jing, Xiao-Hui Bao, and Jian-Wei Pan. Sequential generation of multiphoton entanglement with a rydberg superatom. *Nature Photonics*, 16(9):658–661, 2022.

[229] Anocha Yimsiriwattana and Jr Lomonaco. Generalized ghz states and distributed quantum computing. 03 2004.

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

[231] SJ Ben Yoo, Sandeep Kumar Singh, Mehmet Berkay On, Gamze Gül, Gregory S Kanter, Roberto Proietti, and Prem Kumar. Quantum wrapper networking. *IEEE Communications Magazine*, 62(3):76–81, 2024.

[232] Jia-Bin You, Dax Enshan Koh, Jian Feng Kong, Wen-Jun Ding, Ching Eng Png, and Lin Wu. Exploring variational quantum eigensolver ansatzes for the long-range xy model. *arXiv preprint arXiv:2109.00288*, 2021.

[233] Guilherme Luiz Zanin, Maxime J Jacquet, Michele Spagnolo, Peter Schiansky, Irati Alonso Calafell, Lee A Rozema, and Philip Walther. Fiber-compatible photonic feed-forward with 99% fidelity. *Optics Express*, 29(3):3425–3437, 2021.

[234] Hezi Zhang, Jixuan Ruan, Hassan Shapourian, Ramana Rao Kompella, and Yufei Ding. Oneperc: A randomness-aware compiler for photonic quantum computing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 738–754, 2024.

[235] Hezi Zhang, Anbang Wu, Yuke Wang, Gushu Li, Hassan Shapourian, Alireza Shabani, and Yufei Ding. Oneq: A compilation framework for photonic one-way quantum computation. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023.

[236] J. Zhang, Guido Pagano, P. Hess, A. Kyprianidis, Patrick Becker, H. Kaplan, A. Gorshkov, Zhe-Xuan Gong, and C. Monroe. Observation of a many-body dynamical phase transition with a 53-qubit quantum simulator. *Nature*, 551, 11 2017.

[237] Yangming Zhao and Chunming Qiao. Redundant entanglement provisioning and selection for throughput maximization in quantum networks. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[238] Yangming Zhao, Gongming Zhao, and Chunming Qiao. E2e fidelity aware routing and purification for throughput maximization in quantum networks. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 480–489. IEEE, 2022.

[239] Han-Sen Zhong, Yu-Hao Deng, Jian Qin, Hui Wang, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Dian Wu, Si-Qiu Gong, Hao Su, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Yuxuan Li, Xiao Jiang, Lin Gan, Guangwen Yang, Lixing You, Zhen Wang, Li Li, Nai-Le Liu, Jelmer J. Renema, Chao-Yang Lu, and Jian-Wei Pan. Phase-programmable gaussian boson sampling using stimulated squeezed light. *Phys. Rev. Lett.*, 127:180502, Oct 2021.

[240] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Yuxuan Li, Xiao Jiang, Lin Gan, Guangwen Yang, Lixing You, Zhen Wang, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020.

[241] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Li Yuxuan, Xiao Jiang, Lin Gan, Guangwen Yang, L. You, and Jian-Wei Pan. Quantum computational advantage using photons. *Science (New York, N.Y.)*, 370:1460–1463, 12 2020.

[242] Youpeng Zhong, Hung-Shen Chang, Audrey Bienfait, Étienne Dumur, Ming-Han Chou, Christopher Conner, Joel Grebel, Rhys Povey, Haoxiong Yan, David Schuster, and Andrew

Cleland. Deterministic multi-qubit entanglement in a quantum network. *Nature*, 590:571–575, 02 2021.

[243] Chao Zhou, Pinlei Lu, Matthieu Praquin, Tzu-Chiao Chien, Ryan Kaufman, Xi Cao, Mingkang Xia, Roger Mong, Wolfgang Pfaff, David Pekker, et al. A modular quantum computer based on a quantum state router. 2021.

[244] Hengyun Zhou, Casey Duckering, Chen Zhao, Dolev Bluvstein, Madelyn Cain, Aleksander Kubica, Sheng-Tao Wang, and Mikhail D Lukin. Resource analysis of low-overhead transversal architectures for reconfigurable atom arrays. *arXiv preprint arXiv:2505.15907*, 2025.

[245] Yiru Zhou, Pooja Malik, Florian Fertig, Matthias Bock, Tobias Bauer, Tim van Leent, Wei Zhang, Christoph Becher, and Harald Weinfurter. Long-lived quantum memory enabling atom-photon entanglement over 101 km of telecom fiber. *PRX Quantum*, 5(2):020307, 2024.

[246] Mariam Zomorodi-Moghadam, Mahboobeh Houshmand, and Monireh Houshmand. Optimizing teleportation cost in distributed quantum circuits. *International Journal of Theoretical Physics*, 57:848–861, 2018.