OnePerc: A Randomness-aware Compiler for Photonic Quantum Computing

Hezi Zhang
hezi@ucsd.edu
University of California,
San Diego, USA

Jixuan Ruan
j3ruan@ucsd.edu
University of California,
San Diego, USA

Hassan Shapourian
hshapour@cisco.com
Cisco Quantum Lab
San Jose, USA

Ramana Rao Kompella
rkompell@cisco.com
Cisco Quantum Lab
San Jose, USA

Yufei Ding
yufeiding@ucsd.edu
University of California,
San Diego, USA

Abstract

The photonic platform holds great promise for quantum computing. Nevertheless, the intrinsic probabilistic characteristic of its native fusion operations introduces substantial randomness into the computing process, posing significant challenges to achieving scalability and efficiency in program execution. In this paper, we introduce a randomness-aware compilation framework designed to concurrently achieve scalability and efficiency. Our approach leverages an innovative combination of offline and online optimization passes, with a novel intermediate representation serving as a crucial bridge between them. Through a comprehensive evaluation, we demonstrate that this framework significantly outperforms the most efficient baseline compiler in a scalable manner, opening up new possibilities for realizing scalable photonic quantum computing.

ACM Reference Format:
Hezi Zhang, Jixuan Ruan, Hassan Shapourian, Ramana Rao Kompella, and Yufei Ding. 2024. OnePerc: A Randomness-aware Compiler for Photonic Quantum Computing. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3620666.3651372

1 Introduction

Photonic platform holds great promise for universal quantum computing due to the unique advantages of photonic qubits [1, 2], including their great scalability, long coherence time and easy integration with quantum networks. Besides the experimental demonstration of quantum supremacy on photonic systems [3–5], PsiQuantum has proposed their technology roadmap towards one million qubits using silicon photonics [6, 7]. The potential high clock speed of this approach [7] could make photonic platform advantageous for near-term quantum algorithms.

Photonic quantum computing differs from other platforms such as superconducting [8], ion trap [9] and neutral atoms [10], as it is scaled up by a probabilistic operation known as *fusion* [6]. 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 [11], 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 [11, 12]. On the software side, this randomness is not taken into consideration by existing software infrastructures for the circuit-based model [13] (e.g., Qiskit [14], Tket [15]). 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) [16, 17].

Recently, as an initial effort towards efficient photonic MBQC, a compilation framework OneQ [18] 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* [17] (Fig. 1(a)) into a fusion pattern between the small entangled *resource states* [6] available on the hardware (Fig. 1(b)). However, when fusion failures occur in real-time execution, as illustrated in Fig. 1(b), the resulting state becomes a random graph state deviating from the target structure. Thus the execution needs to be retried

ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA

ACM ISBN 979-8-4007-0386-7/24/04.
https://doi.org/10.1145/3620666.3651372

until success, which is non-scalable given that a practical fusion success probability in the near term is merely around 75% [11, 12]. From now on, we will refer to the graph states required by programs (e.g., Fig. 1(a)) as *program graph states* and those generated by fusions (e.g., Fig. 1(c)) as *physical graph states*.

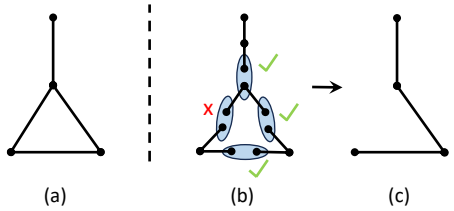


Figure 1. Randomness brought by fusion failures.

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. 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 [19].

Fortunately, there are some nice features of fusions and graph states that we can leverage. Firstly, fusion failures are heralded [20], allowing real-time awareness and enabling the incorporation of classical feed-forward [20–22] 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 [23–25], 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 [16].

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.

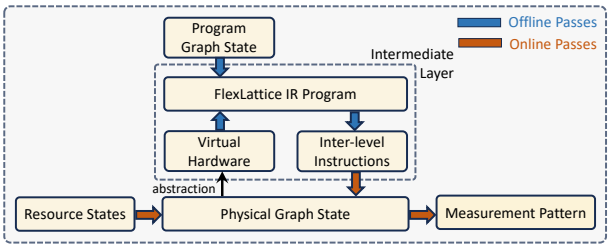


Figure 2. High-level design of OnePerc.

To this end, we propose a randomness-aware compiler to efficiently scale up quantum computing on photonic systems, as illustrated in Fig. 2. Our framework achieves concurrent real-time scalability and program execution efficiency through the combination of an online 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 complexity 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.

2 Background

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 [17]. As exemplified in 1(a), 3(b), 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 [17] 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. 3 shows the translation from 3(a) to 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 [26].

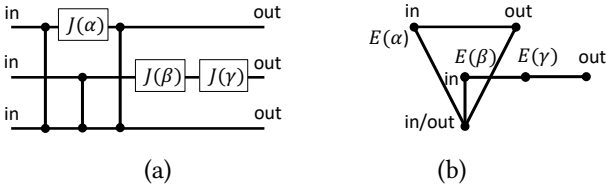


Figure 3. Translation from a circuit (a) to a measurement pattern on a graph state (b).

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 [27] emerges as highly suitable for photonic quantum computing. Besides the experimental demonstration of small-scale photonic MBQC [28–30], the photonic platform is rapidly scaling up with integrated waveguides and optical chips [7, 31–35]

Practical photonic hardware scales up by creating small *resource states*, e.g., 4-qubit, 6-qubit graph states, and connecting them through *fusions* [6]. 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 [20]) 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.

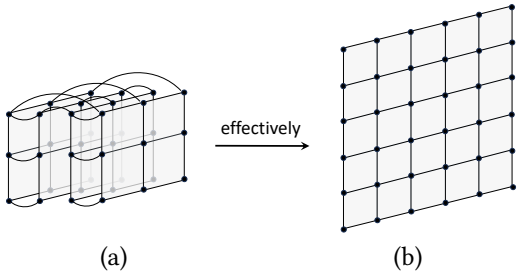


Figure 4. Form a large RSL from multiple small RSLs.

With the advanced integrated silicon photonics, hardware components described above can operate on the scales characterized by GHz clock rates [36–38], potentially leading to a time scale ~ 1 ns for RSG cycles [7]. 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 [19], 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 [7]. For example, the large RSL depicted in Fig. 4(b) can be formed by

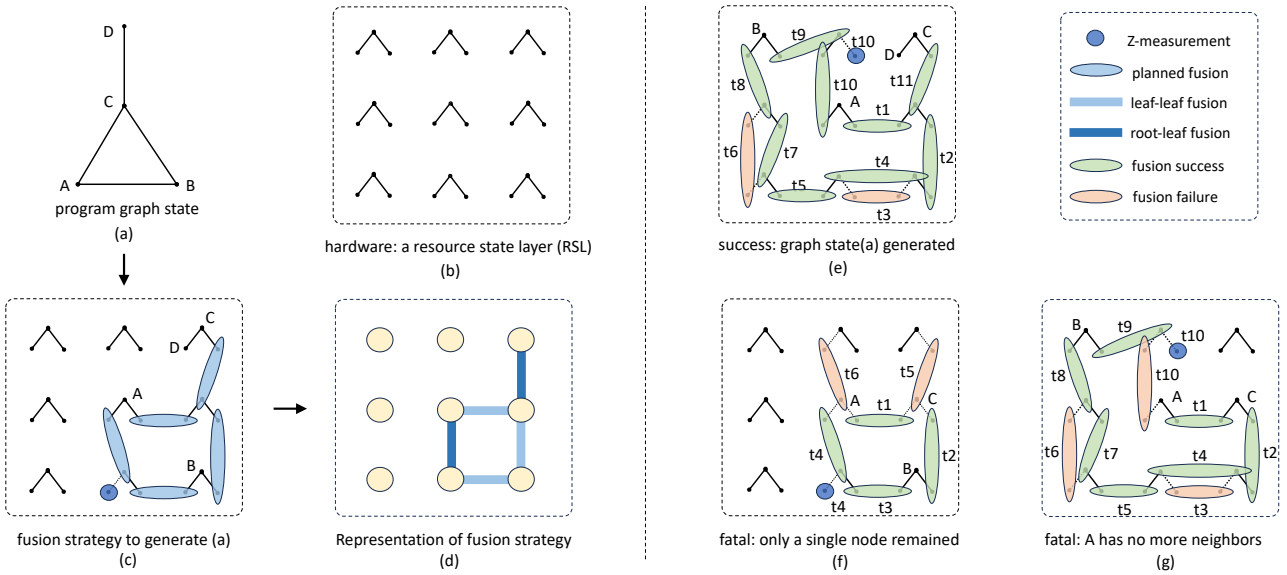


Figure 5. Why OneQ does not work.

fusing the edges of several small RSLs as depicted in Fig. 4(a), resembling folding a paper by twice (4(b)→ 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% [11, 12]. 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 [12]. 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% [11].

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.

3.1 Motivating Example

OneQ + Retry. Fig. 5(c) illustrates OneQ's strategy with an example of generating a graph state in Fig. 5(a) from a single RSL depicted in Fig. 5(b), with the strategy represented more compactly by Fig. 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. 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. 5(c) can result in a dynamic implementation in Fig. 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. 5(a).

However, it is worth noting that some fatal failures may necessitate the retry of the entire compilation. For example, in Fig. 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. 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. 5(c). This undermines the benefits of the proactive planning, eroding the efficiency achieved by the mapping strategy of OneQ.

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. 6(a) onto the hardware in Fig. 6(b), which is 3 layers of that in Fig. 5(b). Indeed, while Fig. 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. 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. 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. 5(c)).

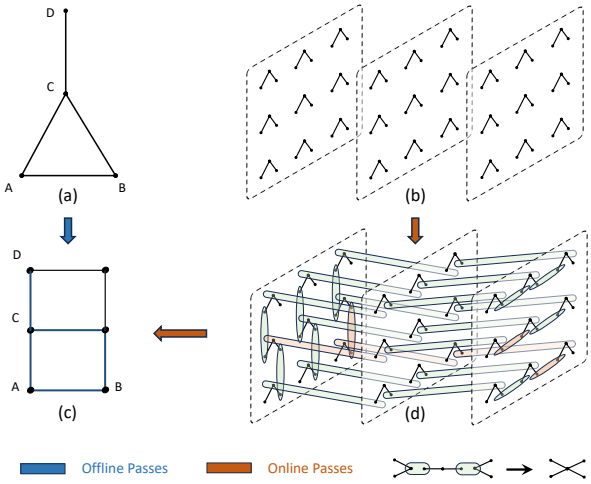


Figure 6. Overview of the compilation flow.

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. 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. 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 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).
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 5).

The following sections (Section 4, 5, 6) will provide a bottom-up introduction to the framework.

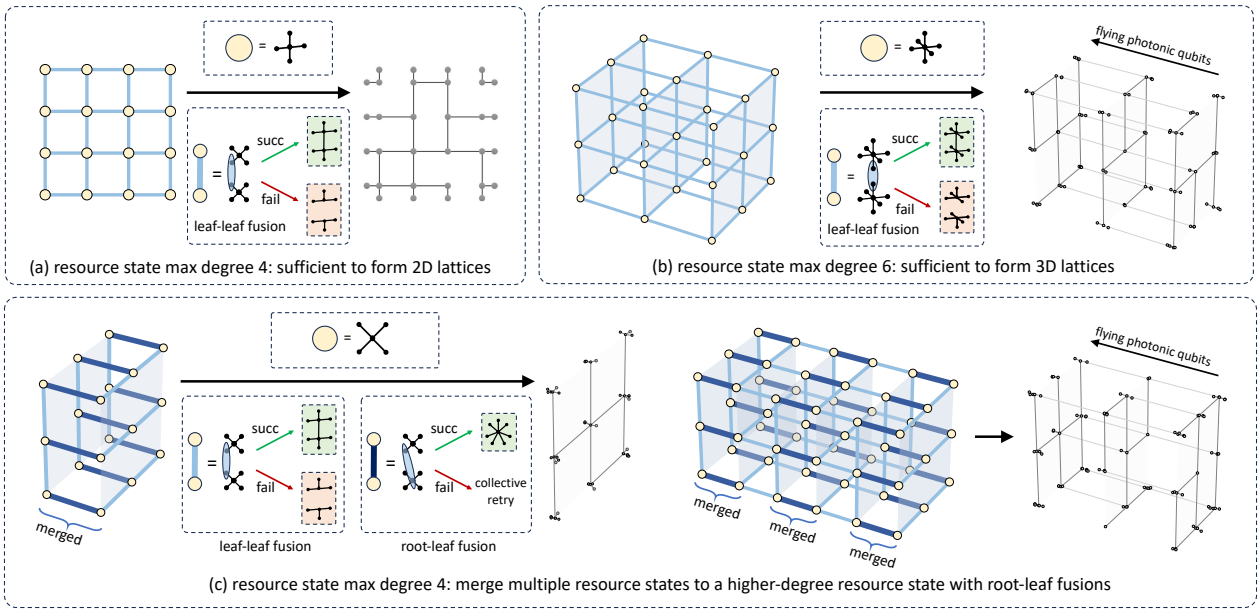


Figure 7. Fuse small resource states into 3D lattices.

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. 2, this corresponds to the online pass from resource states to physical graph states.

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. 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. 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. 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.

4.2 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. 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. 8) by applying the following sequence of 1-qubit operators [39]

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

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. 4.1. Similarly, the LC operators can also be interchanged with fusion operations, with the rules summarized in Theorem 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.

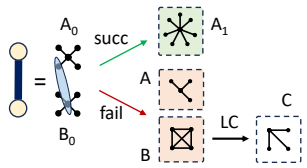


Figure 8. Root-leaf fusion failure.

Theorem 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.*

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 4.2. *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

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

□

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. 8) and retrying failed root-leaf fusions with remaining degrees (e.g., with A and C in Fig. 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.

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. 2, this corresponds to the online pass from physical graph states to measurement patterns.

5.1 Efficient 2D Renormalization

On each (merged) RSL, we apply a process known as renormalization [25], 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 [23–25]. 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 [40], 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. 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. 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 $O(L_{\text{module}}^2) \sim O(N^2/m)$, 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.

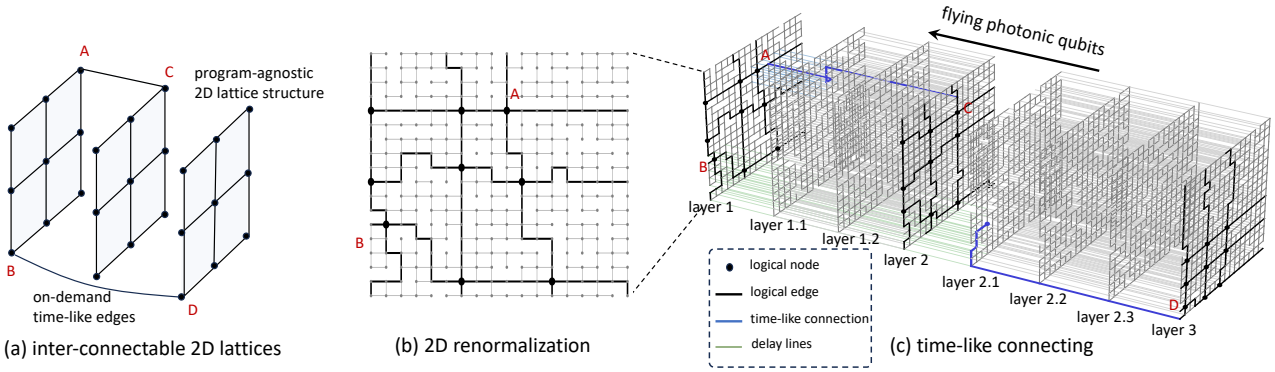


Figure 9. (2+1)-D reshaping for handling random graph states generated by fusions.

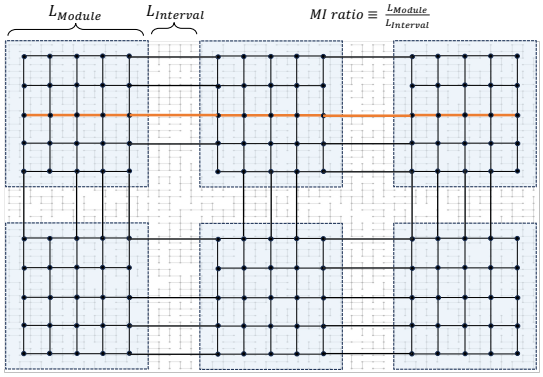


Figure 10. Modular renormalization.

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. 9(a).

The process of generating this 3D graph is illustrated in Fig. 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. 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. 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. 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. 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. 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. 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. 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.

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. 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.

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. 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 [16], 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.

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. 11(a) → (c) → (d).

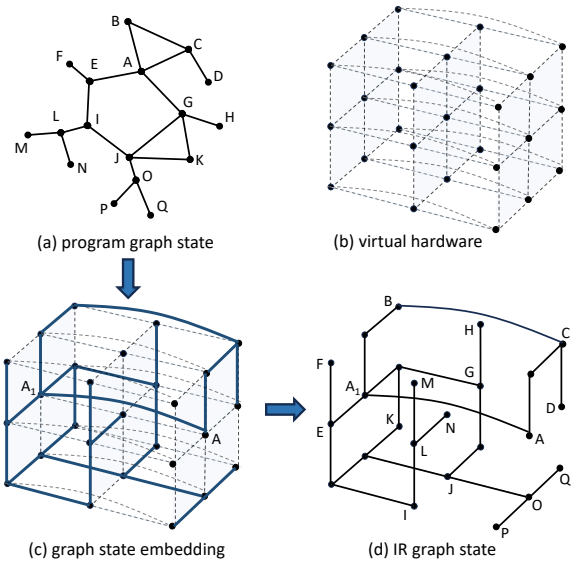


Figure 11. Offline mapping onto the virtual hardware.

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 [41], 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.

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 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. 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. 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)
```

7 Evaluation

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.

Table 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

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.

Photonic Hardware Model. We adopt the same photonic hardware architecture with OneQ, as introduced in Section 2. In the main experiment (Table 2, 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 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) [42] 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 [43, 44]. In table 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

Table 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	

sizes of RSLs needed to generate them, which are determined through Fig. 16, as explained later.

Table 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

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.

Performance. Table 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 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 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 2 takes a RAM as much as 192 GB. This can be overcome by the refresh mechanism proposed in Section 6, with an overhead of increased #RSL. Under the practical fusion success rate of 0.75, Table 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 2 or 3) and 64-qubit benchmarks (Table 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.

7.3 Sensitivity Analysis

Resource State Size. Our compiler has a general applicability to the underlying resource states of various sizes. Fig. 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. 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. 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.

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. 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. 16. As can be seen, it keeps stable against the variation of hardware size, being smaller with a higher fusion success probability. Fig. 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. 10) reduces the available resource on each RSL. To evaluate this resource overhead, Fig. 13(c) depicts the size of the renormalized 2D lattice against the number of modules, with the MI ratio (as defined in Fig. 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-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. 13(c) indicates that the modular approach in our framework can significantly improve the real-time scalability with a reasonable overhead of computing resource.

Compilation Time. We show the online and offline compilation time of the benchmarks in Fig. 14 and Fig. 15, with the compiler implemented in Python. From Fig. 14(a) it can be seen that the online processing time for each RSL stays stable as the program size increases. From Fig. 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. 15(a) shows that it increases with the program size. Fig. 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.

7.5 Hyper Parameters

MI Ratio. The sizes of renormalized lattices rely on a suitable choice of MI ratio (defined in Fig. 10). Fig. 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

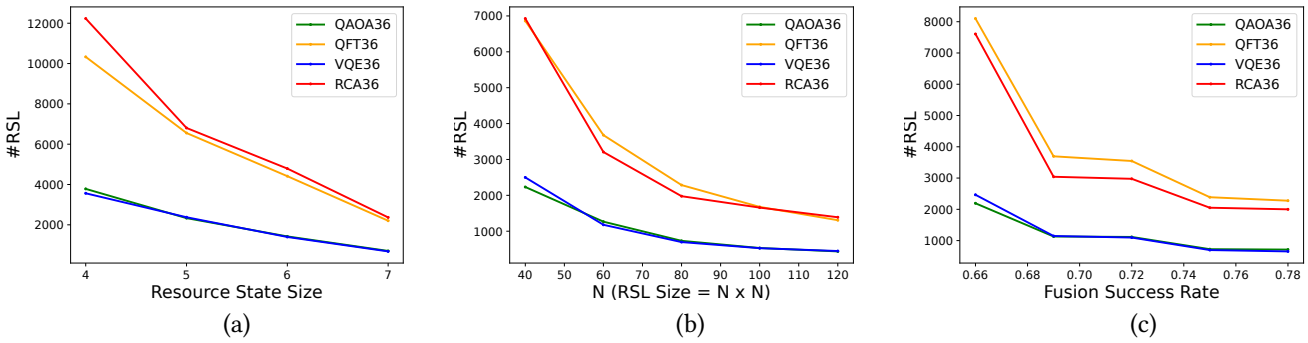


Figure 12. Effects of resource state size (a), hardware size (b) and fusion success probability (c), with the resource states being 7-qubit ones for (b)(c), hardware size being 84x84 for (a)(c), and the fusion success probability being 0.75 for (a)(b).

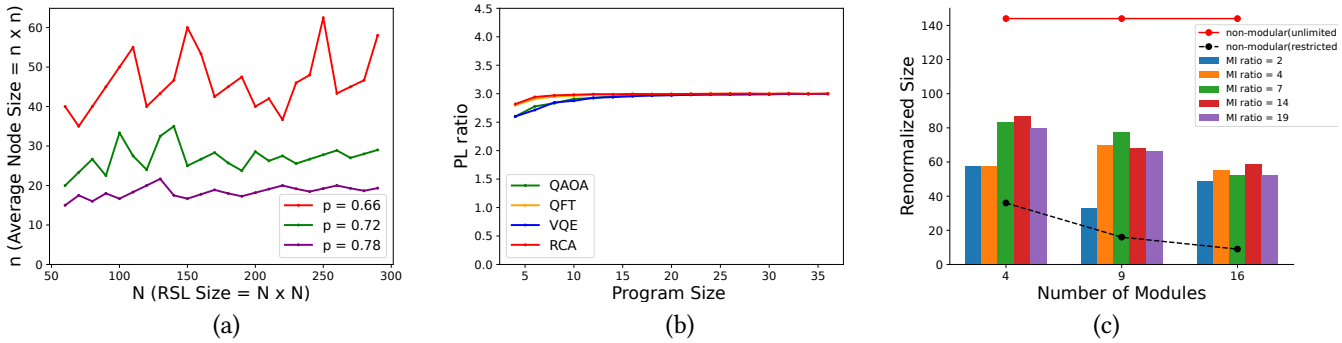


Figure 13. Scalability and parallelism of OnePerc with 7-qubit resource states. The node size $n \times n$ in (a) corresponds to the smallest node size where the renormalization success rate approaches to 1 in Fig. 16.

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.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.

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.

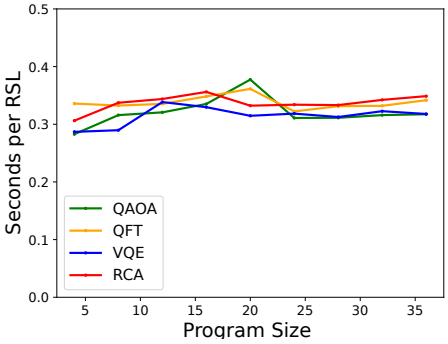
9 Acknowledgement

We thank the anonymous reviewers for their constructive feedback and the cloud bank [45]. This work is supported in part by Cisco Research, NSF 2048144 and Robert N.Noyce Trust.

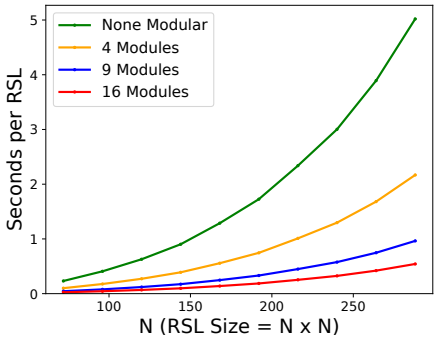
A Artifact Appendix

A.1 Abstract

The artifact contains source codes of OnePerc and necessary code scripts to reproduce key results (Table 2, 3, Fig. 12,13,14,15,16) and compare with the baselines in our evaluation. The hardware requirement is a regular X86 server. The software dependencies only contain common python

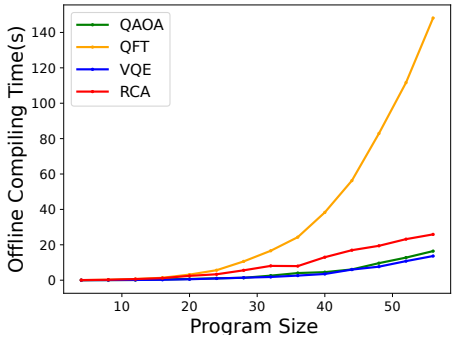


(a)

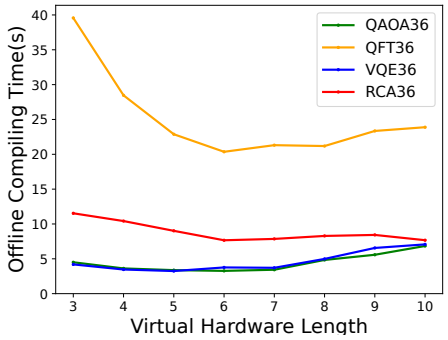


(b)

Figure 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).



(a)



(b)

Figure 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. 14.

packages. As results in Section 7 are averaged over multiple executions, slight deviation is expected in the reproduction.

A.2 Artifact check-list (meta-information)

- **Algorithm:** OnePerc contains two major algorithms.
  - The directory Graph_State_Mapping/ is dedicated to the offline passes, comprising several essential components:
    - Construct_Test_Circuit.py creates benchmark circuits with a specific number of qubits.
    - Graph_State.py transforms the generated quantum circuits into corresponding program graph states.
    - Determine_Dependency.py examines the dependency relationships within the entire graph state.
    - Mapping_Routing.py maps the entire graph state onto a virtual hardware of a specific size.
  - The directory Renormalization/ is dedicated to the online passes, comprising the following key components:
    - Percolate.py simulates probabilistic fusion within a real physical scenario to generate a physical graph state.
    - Renormalization.py reshapes the generated physical graph state to the desired shape of the IR graph state obtained in the offline pass.

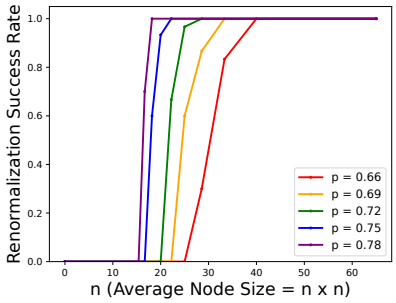


Figure 16. Effect of choices of average node size, with RSL size being 200 × 200.

    - Draw_Grid.py executes 2D renormalization within a single resource state layer.
    - Check_Connectivity.py verifies the presence of time-like connections defined in the previous offline pass.
- **Output:** The output of the compilation process is the reshaped physical graph state identified within the layers of the physical resource state.
- **Run-time environment:** Python, Jupyter Notebook.

- **Hardware:** Memory size depends on the benchmark size and whether the refresh is enabled (the largest benchmarks without refresh can be processed with 192 GB RAM).
- **Experiments:** Compiling the benchmark programs with OnePerc, using OneQ compiler as the baseline.
- **Required disk space (approximately):** When selecting the refresh option, only 32GB of disk space is necessary, whereas opting out could necessitate 130GB of disk space.
- **Metrics:** Resource state depth and fusion cost.
- **Time needed to complete experiments:** The approximate execution time for each benchmark ranges from 10 seconds to 2 hours with program size expanding for OnePerc. For OneQ basline, the execution time can be infinit. In the experiement setting, it is given a upper bound to cost 10^6 resource state layers, after which the execution time varies from 3 minutes to 6 hours with program size expanding. It will take hundreds of CPU hours to fully reproduce all results in Table 2, 3 and Fig. 12, 13, 14, 15, 16.
- **Publicly available:** Yes
- **Code licenses:** Apache License 2.0
- **Workflow framework used:** Jupyter, Qiskit, PyZX
- **Archived repo:** https://zenodo.org/records/10799879
- **DOI:** 10.5281/zenodo.10799879

A.3 Description

A.3.1 How to access. This artifact can be downloaded at the link https://zenodo.org/records/10799879.

A.3.2 Hardware dependencies. A standard server with Intel CPUs can effectively run our artifact, with the capacity of RAM potentially constraining the scale of benchmarks that can be executed. In our experiments, we allocated 192GB of RAM to accommodate the execution of all benchmarks. However, activating the refresh option would reduce this requirement to just 32GB of RAM.

A.3.3 Software dependencies. The artifact is developed using Python 3.10, and we require Jupyter Notebook for its utilization. We have prepared files containing scripts to facilitate the automatic and interactive reproduction of results for convenient validation. Pyzx is employed for generating specific quantum circuits, while other dependencies such as NetworkX, Matplotlib, and NumPy are also utilized.

A.4 Installation

To use our artifact, you may download the repo to your local machine from https://zenodo.org/records/10799879 and install the software dependencies by running commands:

```
conda create -n oneperc python=3.10
pip3 install -r requirements.txt
```

A.5 Evaluation and expected results

After downloading the artifact and installing all software dependencies, you can open the following jupyter notebook files to reproduce experimental data of baseline and OnePerc for corresponding table and figures.

- Compiler.ipynb (Table 2)
- refesh.ipynb (Table 3)
- sensitivity.ipynb (Fig. 12)
- scalability.ipynb (Fig. 13, 16)
- time.ipynb (Fig. 14, 15)

The previous experimental data has already been saved to data/. In scalability.ipynb, sensitivity.ipynb, time.ipynb and refresh.ipynb, setting '`RunAgain = False`' will generate the plots directly from original data, while setting '`RunAgain = True`' will run the experiments again, generating new data and new plots. Note that running with a parameter N corresponds to an N^2-qubit benchmark instead of N-qubit.

The generation of Table 2 is the most time-consuming procedure in the evaluation. This is because OneQ performs badly for large programs. Although we force the compilation to terminate when the consumed #RSL reaches 10^6, it can take hours to reach this limit. As a result, we provide three code blocks in Compiler.ipynb.

- The first code block allows users to run OneQ for individual benchmarks. By changing the value of N, users can obtain the result of OneQ for an N^2-qubit benchmark. We recommend users to try benchmarks from small to large and feel free to stop at the scale they obtain a 10^6 #RSL for OneQ since larger scales should also lead to a 10^6 #RSL.
- The second code block allows users to run OnePerc for individual benchmarks. By changing the value of N, users can obtain the result of OnePerc for an N^2-qubit benchmark. In this process, users can monitor the consumed RAM manually in the terminal.
- The third code block allows users to obtain all results of OnePerc in Table 2 in one shot.

The experiment results of these code blocks will be automatically saved to data/. After the experiments, users can run Compiler_Table.ipynb to read the saved data and generate Table 2.

Table 3 can be generated by refresh.ipynb, which runs the offline mapping and obtain an estimated #RSL from the number of logical layer. This is because from Fig. 13(b), we know that #RSL has a stable relation with the number of logical layers. The results of non-refreshed #RSL for 25-qubit benchmarks can be obtained directly from Table 2. The '-' in Table 3 means that the compilation utilizes more than 32 GB RAM. In the execution of OnePerc for individual benchmarks (code block 2 in Compiler.ipynb), users can monitor the consumed RAM manually in the terminal (e.g., using `htop` on Linux). When running the compiler with large enough RAM, users will observe that the consumed RAM exceeds 32 GB for benchmarks larger than 25 qubits. When running the compiler with only 32 GB RAM, users will observe that the compilation of benchmarks larger than 25 qubits would be killed after some time.

References

[1] Jeremy L. O'Brien, Akira Furusawa, and Jelena Vučković. Photonic quantum technologies. *Nature Photonics*, 3(12):687, 2009. URL: https://www.nature.com/articles/nphoton.2009.229, doi:10.1038/nphoton.2009.229.

[2] 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. URL: http://opg.optica.org/ome/abstract.cfm?URI=ome-7-1-111, doi:10.1364/OME.7.000111.

[3] 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. URL: https://www.science.org/doi/10.1126/science.abe8770, doi:10.1126/science.abe8770.

[4] 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. URL: https://link.aps.org/doi/10.1103/PhysRevLett.127.180502, doi:10.1103/PhysRevLett.127.180502.

[5] 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. URL: https://www.nature.com/articles/s41586-022-04725-x, doi:10.1038/s41586-022-04725-x.

[6] Terry Rudolph. Fusion based photonic quantum computing. In *APS March Meeting Abstracts*, volume 2022, pages D28–001, 2022. URL: https://www.nature.com/articles/s41467-023-36493-1, doi:10.1038/s41467-023-36493-1.

[7] 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*. URL: https://arxiv.org/abs/2103.08612, doi:10.48550/arXiv.2103.08612.

[8] Michel H Devoret and Robert J Schoelkopf. Superconducting circuits for quantum information: an outlook. *Science*, 339(6124):1169–1174, 2013. URL: https://science.org/doi/10.1126/science.1231930, doi:10.1126/science.1231930.

[9] 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. URL: https://pubs.aip.org/aip/apr/article-abstract/6/2/021314/570103/Trapped-ion-quantum-computing-Progress-and?redirectedFrom=fulltext, doi:10.1063/1.5088164.

[10] 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. URL: https://iopscience.iop.org/article/10.1088/0953-4075/49/20/202001, doi:10.1088/0953-4075/49/20/202001.

[11] Warren P Grice. Arbitrarily complete bell-state measurement using only linear optical elements. *Physical Review A*, 84(4):042331, 2011. URL: https://journals.aps.org/pra/abstract/10.1103/PhysRevA.84.042331, doi:10.1103/PhysRevA.84.042331.

[12] 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. URL: https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.113.140403, doi:10.1103/PhysRevLett.113.140403.

[13] Michael A Nielsen and Isaac L Chuang. Quantum computation and quantum information. *Phys. Today*, 54(2):60, 2001. URL: https://cds.cern.ch/record/465953/files/0521635039_TOC.pdf.

[14] Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023. URL: https://zenodo.org/records/2562111, doi:10.5281/zenodo.2573505.

[15] 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. URL: https://iopscience.iop.org/article/10.1088/2058-9565/ab8e92, doi:10.1088/2058-9565/ab8e92.

[16] Robert Raussendorf, Dan Browne, and Hans Briegel. Measurement-based quantum computation on cluster states. *Raussendorf, R. and Browne, D.E. and Briegel, H.J. (2003) Measurement-based quantum computation on cluster states. Physical Review A, 68 (2). 022312.1-022312.32. ISSN 10502947*, 68, 08 2003. URL: https://journals.aps.org/pra/abstract/10.1103/PhysRevA.68.022312, doi:10.1103/PhysRevA.68.022312.

[17] Anne Broadbent and Elham Kashefi. Parallelizing quantum circuits. *Theoretical computer science*, 410(26):2489–2510, 2009. URL: https://www.sciencedirect.com/science/article/pii/S0304397508009377?via%3Dihub, doi:10.1016/j.tcs.2008.12.046.

[18] 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. URL: https://dl.acm.org/doi/10.1145/3579371.3589047, doi:10.1145/3579371.3589047.

[19] 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. URL: https://www.sciencedirect.com/science/article/abs/pii/B9780128165027000014, doi:10.1016/B978-0-12-816502-7.00001-4.

[20] 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. URL: https://journals.aps.org/rmp/abstract/10.1103/RevModPhys.79.135, doi:10.1103/RevModPhys.79.135.

[21] 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. URL: https://opg.optica.org/oe/fulltext.cfm?uri=oe-29-3-3425&id=446800, doi:10.1364/OE.409867.

[22] Atsushi Sakaguchi, Shunya Konno, Fumiya Hanamura, Warit Asavanant, Kan Takase, Hisashi Ogawa, Petr Marek, Radim Filip, Junichi Yoshikawa, Elanor Huntington, et al. Nonlinear feedforward enabling quantum computation. *Nature Communications*, 14(1):3817, 2023. URL: https://www.nature.com/articles/s41467-023-39195-w, doi:10.1038/s41467-023-39195-w.

[23] Mercedes Gimeno-Segovia, Pete Shadbolt, Dan E Browne, and Terry Rudolph. From three-photon ghz states to universal ballistic quantum computation. 2015. URL: https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.115.020502, doi:10.1103/PhysRevLett.115.020502.

[24] Mihir Pant, Don Towsley, Dirk Englund, and Saikat Guha. Percolation thresholds for photonic quantum computing. *Nature communications*, 10(1):1070, 2019. URL: https://www.nature.com/articles/s41467-019-08948-x, doi:10.48550/arXiv.1701.03775.

[25] 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. URL: https://iopscience.iop.org/article/10.1088/1367-2630/10/2/023010, doi:

10.1088/1367-2630/10/2/023010.

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

[27] Sergei Slussarenko and Geoff J Pryde. Photonic quantum information processing: A concise review. *Applied Physics Reviews*, 6(4):041303, 2019. URL: https://pubs.aip.org/aip/apr/article/6/4/041303/997349/Photonic-quantum-information-processing-A-concise, doi:10.1063/1.5115814.

[28] 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. URL: https://www.nature.com/articles/nature03347, doi:10.1038/nature03347.

[29] 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. URL: https://journals.aps.org/pra/abstract/10.1103/PhysRevA.81.050302, doi:10.1103/PhysRevA.81.050302.

[30] 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. URL: https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.113.200501, doi:10.1103/PhysRevLett.113.200501.

[31] Simone Ferrari, Carsten Schuck, and Wolfram Pernice. Waveguide-integrated superconducting nanowire single-photon detectors. *Nanophotonics*, 7(11):1725–1758, 2018. URL: https://www.degruyter.com/document/doi/10.1515/nanoph-2018-0059/html?lang=en, doi:10.1515/nanoph-2018-0059.

[32] 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. URL: https://www.science.org/doi/10.1126/science.aar7053, doi:10.1126/science.aar7053.

[33] 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. URL: https://arxiv.org/abs/2206.10076, doi:10.48550/arXiv.2206.10076.

[34] 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. URL: https://www.nature.com/articles/nphoton.2011.283, doi:10.1038/nphoton.2011.283.

[35] 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. URL: https://www.science.org/doi/10.1126/science.aab3642, doi:10.1126/science.aab3642.

[36] 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. URL: https://www.nature.com/articles/s41567-019-0567-8, doi:10.1038/s41567-019-0567-8.

[37] 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. URL: https://pubmed.ncbi.nlm.nih.gov/32632281/, doi:10.1038/s41563-020-0725-5.

[38] 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. URL: https://www.nature.com/articles/s41586-018-0551-y, doi:10.1364/OPTICA.415762.

[39] 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. URL: https://arxiv.org/abs/quant-ph/0602096, doi:10.48550/arXiv.quant-ph/0602096.

[40] 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. URL: https://link.springer.com/article/10.1007/BF01197577, doi:10.1007/BF01197577.

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

[42] 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. URL: https://arxiv.org/abs/quant-ph/0410184, doi:10.48550/arXiv.quant-ph/0410184.

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

[44] 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. 2021. arXiv:2109.00288.

[45] Michael Norman, Vince Kellen, Shava Smallen, Brian DeMeulle, Shawn Strande, Ed Lazowska, Naomi Alterman, Rob Fatland, Sarah Stone, Amanda Tan, et al. Cloudbank: Managed services to simplify cloud access for computer science research and education. In *Practice and Experience in Advanced Research Computing*, pages 1–4. 2021.