Optimizing Circuits for Continuous-Variable Quantum Simulation of Many-Body Dynamics

Sam Cochran

May 2024

1 Introduction

Quantum computing is expected to provide a powerful tool for accelerating computational tasks across various disciplines. One candidate for demonstrating quantum advantage is quantum simulation of quantum dynamics. The primary focus within this domain has traditionally been on simulating discrete quantum systems, such as spin chains, using digital quantum simulation (DQS). DQS is not limited to discrete systems, however, and it can also be extended to handle continuous-variable quantum systems, offering the prospect of exponential speedup in certain scenarios. A prominent example of this capability is the Zalka-Wiesner algorithm, which demonstrates how continuous-variable quantum systems can be effectively simulated on a digital quantum platform.

A comparatively less explored possibility is the quantum simulation of classical dynamics, using approaches like Koopman von Neumann (KvN) theory to map statistical mechanical density functions onto quantum states. Previous work [1] has identified a key obstacle in KvN simulations using the DQS framework arising from the discretization needed to map the problem onto qubit-based quantum devices. The existence of this barrier to implementation strongly motivates alternative approaches.

The emerging field of continuous-variable quantum computation (CV-QC) offers a novel approach for simulating both classical and quantum systems. CV-QC leverages continuous quantum information processing units (qumodes), mapping each continuous degree of freedom in the system being simulated onto a corresponding qumode of the quantum device. This paradigm presents a distinct contrast to the digital quantum simulation methods, such as the Zalka-Wiesner method, which discretize continuous-variable systems for simulation on digital quantum computers. Using gate synthesis methods to decompose a problem Hamiltonian into elementary continuous-variable quantum gates, CV-QC allows us to simulate continuous dynamics using hardware that is inherently continuous.

In this work, we will focus on the problem of simulating continuous systems with up to quartic Hamiltonians using CV-QC. Such systems will require interactions between different subsets of four qumodes at a time. This problem setup presents a unique challenge when it comes to scheduling operations at compilation time, as SWAP operations will be needed to bring subsets of qumodes together for interaction. Previous work [2] has provided a structured method for optimizing QAOA circuits in the case of digital quantum computing, where pairwise interactions between qubits are needed. We first note that in the case of quadratic Hamiltonians, this exact approach can be directly applied to CV-QC in order to optimize the compiled circuit. We then use ideas from this work as a jumping off point for developing a schedule for interactions between 4-qumode subsets that minimizes the number of SWAPs needed.

In order to develop this schedule, we propose a tree-based hardware topology that is efficient for implementing this problem. We then introduce a schedule tailored to this topology that allows us to achieve every possible 4-qumode subset in the case of an 8-qumode system without introducing any extraneous SWAPs. We quantify the number of SWAPs needed in this case, and suggest directions for future work to generalize this schedule to larger systems.

2 Background and Motivation

2.1 Qumode Systems

The digital quantum computing framework uses qubits, or more generally qudits, as the basic unit of computation. Qubits are 2-level systems, while qudits are d-level systems. In contrast, the basic units of computation in the CV-QC framework are quantum modes, or qumodes. Each qumode models the Hilbert space of a quantum harmonic oscillator. As such, the state space of each qumode is countably infinite, with states corresponding to the discrete energy levels of the oscillator. Operations performed on a single qumode can be described in terms of the bosonic annihilation and creation operators $\hat{a}$ and $\hat{a}^\dagger$, which respectively lower and raise the energy of the mode. These operators can be used to define position and momentum quadrature operators Q and P as

$$Q = \frac{1}{\sqrt{2}}(\hat{a} + \hat{a}^\dagger), \quad P = -\frac{i}{\sqrt{2}}(\hat{a} - \hat{a}^\dagger). \tag{1}$$

The quadrature operators on two qumodes j and k satisfy the canonical commutation relation

$$[Q_j, P_k] = i\delta_{jk}, \tag{2}$$

where δ_{jk} is the Kronecker delta.

The physically implementable set of operations available in CV-QC is the set of unitary transformations. Arbitrary unitary transformations can be decomposed in terms of a set of universal gates. These elementary gates that can be implemented natively on continous-variable hardware can be written in terms of either the annihilation and creation or the quadrature operators. Below is a non-exhaustive list of elementary gates that will be needed to implement the simulation problem

1. quadratic phase $\mathcal{P}(s) = \exp\left(i\frac{s}{2}Q^2\right)$
2. controlled-X $\mathcal{CX}_{jk}(s) = \exp(-isQ_jP_k)$
3. controlled-phase $\mathcal{CZ}_{jk}(s) = \exp(isQ_jQ_k)$
4. cubic phase $\mathcal{V}(s) = \exp\left(i\frac{s}{3}Q^3\right)$
5. rotation $\mathcal{R}(s) = \exp\big(is\hat{a}^\dagger\hat{a}\big)$.

In continuous-variable quantum mechanics, the Fourier transform rotates between the position and momentum eigenbases. An important observation of the CV-QC formulation is that the Fourier transform can be described using the above gate sense. Specifically, the Fourier transform on a given mode can be implemented using a single rotation of angle $s = \pi/2$. We call this gate the Fourier gate

$$\begin{aligned} \mathcal{F} &= \mathcal{R}(\pi/2) \\ \mathcal{F}^\dagger Q \mathcal{F} &= -P \\ \mathcal{F}^\dagger P \mathcal{F} &= Q. \end{aligned} \tag{3}$$

Any operation to be done on a cv-quantum computer must be decomposed into a sequence of elementary gates, which can then be compiled to the hardware.

2.2 Existing Hardware Connectivity

We will now discuss two existing continuous-variable quantum hardware implementations built by Xanadu. We will then propose our own hardware topology based on observations of the existing hardware.

2.2.1 Xanadu Borealis

First, we will examine Xanadu's Borealis device. Built for Boson sampling, Borealis consists of 216 squeezed modes entangled with 3-dimensional connnectivity [3]. This means that each qumode is connected to between 3 and 6 other qumodes. In fact, most of the qumodes have 6 connections, with lower connection numbers occurring only at the boundaries of the three-dimensional lattice topology. This high level of connectivity is achieved through time-multiplexing, with delays of 1, 6, and 36 time intervals inducing the 3-dimensional structure discussed above (see Figure 1).

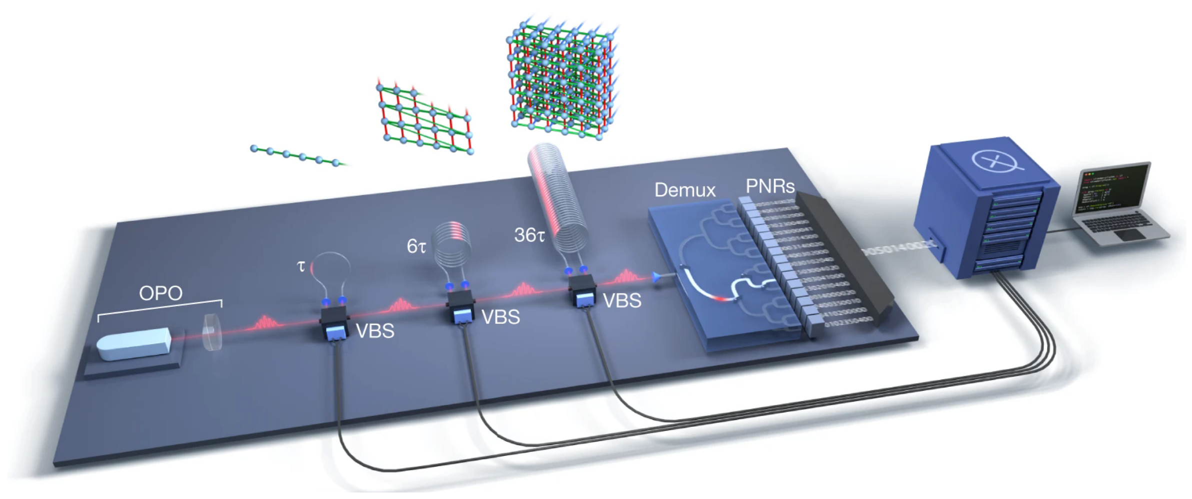

Figure 1: Diagram of Xanadu's Borealis device. Grids at each time delay show connectivity achieved up to that point, with 3-dimensional lattice connectivity achieved after all delays are accounted for. Image credit [3].

2.2.2 Xanadu X8

We will now examine Xanadu's X8 device, a nanophotonic chip able to run continuous-variable quantum algorithms involving up to 8 qumodes [4]. The X8 chip consists of 4 signal modes and 4 idler modes. These two sets of modes are entangled in a pairwise fashion to obtain 2-mode squeezed states. An arbitrary unitary U_4 involving up to 4 qumodes is then applied to both the signal and idler modes, after which a Fock basis measurement is performed. The unitary U_4 must consist of beamsplitter gates, Mach–Zehnder gates, rotation gates, and interferometer operations. Notably, each mode within the signal and idler sets is connected to each other mode within the same set, resulting in full conectivity between sets of 4 qumodes. See Figure 2 for a diagram.

2.3 Circuit Optimization for QAOA

Ref. [2] provides a structured method for optimizing the compilation of QAOA circuits. A key observation for this method is that each layer of a QAOA circuit consists of 2-qubit interactions that commute with each other. This means that we have full control over the order in which we apply each 2-qubit interaction. The structure of the approach is to first establish a schedule for realizing connections between every pair of qubits with minimal SWAP cost, and then other methods can be used to optimize the circuit based on which pairwise interactions are actually present in the problem Hamiltonian.

The core methodology of the method is introduced using 1- and 2-dimensional lattices. First, a method is established for achieving every pairwise interaction in the 1-dimensional lattice. Then, using a 2-dimensional lattice as the baseline structure, different types of SWAP operations are defined. Units are defined as the rows of the lattice, so the 1-dimensional case is used to apply all SWAPs needed to realize interactions between each qubit in a single row (intra-unit interaction). Next, a method is described to apply all SWAPs needed for achieving pairwise interactions between qubits in neighboring rows (inter-unit interaction). Finally, a method for applying SWAPs to exchange two rows is defined (unit exchange). Using these three types of SWAP operations, it is possible to realize every possible pairwise interaction efficiently.

This paper was written with superconducting qubit-based hardware in mind, where in practice, the level of connectivity that can be achieved is lower than that needed for the 2-dimensional lattice. This can be addressed by simply determining a way to add extra SWAPs needed to apply each of the three types of operations (intra-unit, inter-unit, and unit exchange).

In the case of quantum simulation of quadratic Hamiltonians, this procedure can be adapted directly to the continuous-variable framework to provide an optimal compilation strategy. In fact, because existing hardware already supports connectivity beyond the two-dimensional lattice, no further work is

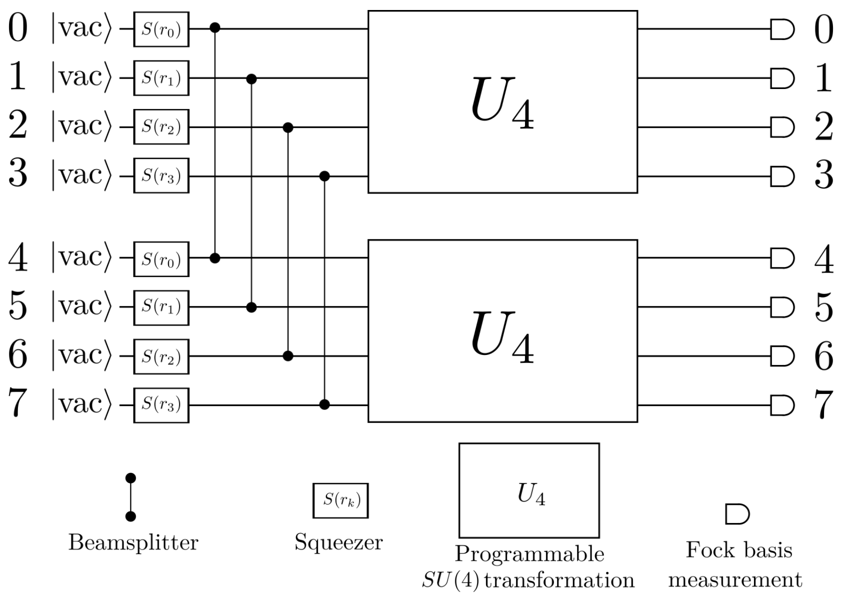

Figure 2: Diagram of Xanadu's X8 chip. An arbitrary unitary U_4 is applied to a set of 4 signal modes and 4 idler modes, which are initially entangled as 2-mode squeezed states. Note the full connectivity within each set. Image credit [4].

needed; the schedule built for the lattice can be used exactly as is to simulate quadratic Hamiltonians. For Hamiltonians containing higher-order terms, more work must be done to establish a way to efficiently realize each possible higher-order interaction.

2.4 Gate Synthesis for Higher-Order Terms

Moving beyond quadratic Hamiltonians, we now consider the problem of simulating time evolution under a quartic Hamiltonian with the general form

$$H = f(P) + \sum_j F_j Q_j + \sum_{jk} F_{jk} Q_j Q_k + \sum_{jkl} F_{jkl} Q_j Q_k Q_l + \sum_{jklm} F_{jklm} Q_j Q_k Q_l Q_m. \tag{4}$$

This is done by implementing the Trotterized time evolution operator

$$\begin{aligned} \exp(-itH) \approx \Big(& \exp(-i\delta t f(P)) \prod_j \exp(-i\delta t F_j Q_j) \prod_{jk} \exp(-i\delta t F_{jk} Q_j Q_k) \\ & \prod_{jkl} \exp(-i\delta t F_{jkl} Q_j Q_k Q_l) \prod_{jklm} \exp(-i\delta t F_{jklm} Q_j Q_k Q_l Q_m) \Big)^{t/\delta t}. \end{aligned} \tag{5}$$

In order to simplify the discussion, we will assume that $f(P)$ is simply the Kinetic energy operator proportional to P^2 applied on each qumode. This can be implemented using Fourier gates and quadratic phase gates. We will focus the rest of our discussion on the position quadrature terms, noting that, without loss of generality, more general momentum quadrature terms can be addressed in the same way using Fourier gates.

Each linear and quadratic term in (5) can be implemented in terms of elementary gates (specifically displacement, quadratic phase, and controlled-phase gates). The single-mode cubic term also corresponds directly to an elementary gate (the cubic phase gate). All remaining terms, including the cubic cross terms and each quartic term, must be reduced to a sequence of elementary gates using gate decomposition methods. Following the procedure outlined in Ref. [5], each gate decomposition can be boiled down to repeated applications of gate sequences of the form

$$e^{ia_4 Q_n P_1} ... e^{ia_3 Q_3 P_1} e^{ia_2 Q_2 P_1} e^{iCQ_1^n} e^{-ia_2 Q_2 P_1} e^{-ia_3 Q_3 P_1} ... e^{-ia_4 Q_n P_1} \tag{6}$$

where n is either 3 or 4. See appendix B for details. Importantly, each exponential in (6) except for the one in the center corresponds to a controlled-X gate. In the case that $n = 3$, the center exponential corresponds to a cubic phase gate. In the case of $n = 4$, however, the center operation takes the form $e^{iCQ_1^4}$, and a further decomposition is needed which requires an additional ancilla qumode. This ancilla is accounted for in the proposed hardware topology below. Additionally, the indices of each quadrature operator in (6) can be permuted, meaning that when it comes to compiling circuits implementing this equation for a given subset of four qumode indices, the only requirement will be that one of the indices be connected to each other index in the subset. Finally, it should be noted that if we achieve an interaction between every possible subset of four qumodes, we can automatically realize interactions between every possible 2- and 3-qumode subset as well by simply interleaving these interactions the first time each two or three qumode subset is achieved during the scheduled 4-qumode interactions.

3 Proposed Method

The method for optimizing the compilation of QAOA circuits presented in Ref. [2] is based on the observation that all terms in a given layer requiring two-qubit interactions commute. This commutation relationship allows us full control over the order in which we implement each interaction. QAOA is in fact a time evolution algorithm, and the structure of QAOA circuits is quite similar to that of the time evolution circuits we consider here. Noting that we also have flexibility with respect to the order in which we implement each exponential product in (5), we seek to define an ordering of these terms that minimizes the number of SWAP gates required.

As noted above, in the case of a quadratic Hamiltonian, we can use the exact schedule introduced in [2]. In the case of the general quartic Hamiltonian given in (4), however, things become more complicated. Instead of requiring only interactions between two qumodes at a time, we now have repeated strings of gates with two-qumode interactions drawn from a subset of four qumodes. An example of one such string is given in (6). These two qumode interactions must be applied within a specific sequence of gates, and cannot commute with interactions from other strings of gates. In other words, we must now consider interactions of unique subsets of four qumodes at a time.

It turns out that this adds a significant level of complexity to the problem of SWAP scheduling. In the quadratic case, two qumodes brought together once never need to be together again within a single Trotter layer. In the quartic case, they need to be brought together multiple times, for every possible combination four qumodes that contains them. This means that not only does the total number of configurations to be achieved grow significantly when we move from the quadratic case to the quartic case (increasing from $\binom{n}{2}$ to $\binom{n}{4}$), the scheduling becomes more complicated, as we must find a way to achieve each configuration exactly once without introducing unnecessary SWAP cycles.

In order to approach this problem, we will first propose a hardware topology that is well tailored to this problem. We will argue that this topology is well motivated by existing hardware. We will then introduce a schedule that is in some sense optimal for this topology.

3.1 Proposed Hardware Topology

Motivated by the units defined in [2], we will propose a hardware structure that fits the connectivity requirements of the operation given in (6). This operation will involve up to four qumodes, and only one of these qumodes is involved in every gate. We will call this the root qumode, and designate the remaining three qumodes as leaves. The center exponential in (6) corresponds either to a cubic phase gate on the root qumode or a decomposition of a quartic gate that will require interactions between the root qumode and an ancilla qumode. Refer to appendix B for details. Every other exponential in (6) is simply a controlled-X gate between the root qumode and one of the leaf qumodes.

These interaction requirements motivate a specific hardware topology. We note that this topology could correspond to a structure physically implemented at the hardware level, but it can also be viewed as a purely theoretical tool that could be embedded into an existing hardware topology with sufficient connectivity to support it. To begin, we introduce a tree-based structure, with a root node connected to three leaf nodes and one ancilla node. We add connections between each leaf node in order to support the necessary SWAPs to realize all possible configurations, but note that the ancilla mode only needs to be connected to the root. See Figure 3a for a diagram of the basic tree structure.

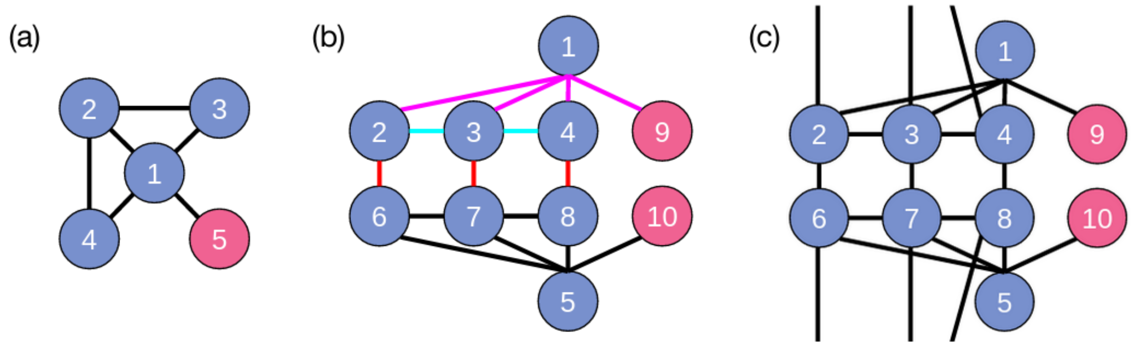

Figure 3: (a) A single tree structure. The root and leaf nodes are blue while the ancilla node is shown in pink. (b) Two trees connected together via their leaf nodes. Edges corresponding to outer SWAPs are shown in red, inner SWAPs in cyan, and root SWAPs in magenta. (c) More edges connecting leaf nodes allow for a string of trees to be connected. Note that any node in a string following this pattern would have at most 5 edges.

This tree structure constitutes our basic unit, following the ideas from [2]. Two trees can be connected via their leaf nodes, as shown in Figure 3b. From here, we must define three types of SWAP operations. The goal is to schedule these SWAP operations so that the every possible configuration of 4 qumodes is achieved exactly once in exactly one of the two connected trees with the minimum number of SWAP cycles possible.

1. We define an **Outer SWAP** as a SWAP between two adjacent leaf nodes in separate trees. Outer SWAPs are carried out along the edges colored red in Figure 3b. New configurations of qumodes in a tree are only achievable through outer SWAPs.

2. We define an **Inner SWAP** as a SWAP between two adjacent leaf nodes in the same tree. Inner SWAPs are carried out along the edges colored cyan in Figure 3b. Inner SWAPs never result in new configurations, but are necessary as intermediate steps to make new configurations reachable through subsequent outer SWAPs. We can simplify the discussion by assuming inner SWAPs are only ever carried out on one of the two connected trees.

3. We define a **Root SWAP** as a SWAP between the root node and one of the leaf nodes of a single tree. Root SWAPs are carried out along the edges colored magenta in Figure 3b. Root SWAPs also never result in new configurations, but are necessary as intermediate steps to make new configurations reachable through subsequent outer and inner SWAPs. We can simplify the discussion by assuming root SWAPs are only ever carried out on one of the two connected trees.

Though we will limit our discussion here to the two tree case, a longer string of trees can be achieved by adding more connections between leaf nodes, as shown in Figure 3c. Any single node in a string of trees connected through their leaf nodes in this way would never require more than 5 edges, meaning that this tree-based topology could theoretically be embedded in any hardware with 3-dimensional grid connectivity. In addition, it suggests that chip-based photonic devices might feasibly be designed to have sufficient connectivity to support the tree-based structure proposed here, noting that the existing X8 architecture already allows for groups of 4 fully connected qumodes, with at least 5 total connections on each qumode.

Finally, though we have focused our analysis thus far on the continuous-variable quantum simulation problem, this tree-based device topology, along with the schedule we will describe in the next section, are also applicable to algorithms designed for qubit-based devices requiring interactions between subsets of 4 qubits. In this case, the tree-based structure could be embedded into any device with high enough connectivity. For example, trapped ion and neutral atom based quantum computers theoretically offer full connectivity by physically moving qubits around, so the topology proposed here could certainly be embedded in these types of devices.

3.2 Scheduling Interactions to Minimize SWAPs

As noted above, new configurations can only be achieved through outer SWAPs, as inner SWAPs and root SWAPs only permute the indices within a single tree. In order to achieve an optimal schedule, we first must ensure that a new configuration is achieved after every outer SWAP cycle. Next, we note that inner and root SWAPs must always be followed by an outer SWAP cycle if a new configuration is to be achieved, meaning that any time we do an inner or root SWAP, we are inherently forced to carry out two SWAP cycles in a row with no new configuration achieved between. Because of this, we should exhaust all configurations available using only outer SWAPs before carrying out an inner SWAP, and we should exhaust all configurations available using only inner and outer SWAPs before carrying out a root SWAP. This should maximize the number of new configurations achieved with only one SWAP cycle needed between unique configurations. Finally, we note that inner and root SWAPs cannot be done in parallel, and must always occupy their own cycle. In contrast, up to three outer SWAPs can be done in parallel in a single cycle.

Following these observations, the algorithm for scheduling SWAP cycles involves repeating the following three steps until no new configurations remain:

1. Starting from the current configuration, realize all new configurations achievable using only outer SWAPs.

2. Carry out an inner SWAP and repeat step 1. Repeat this procedure until no new configurations are achievable using only inner and outer SWAPs.

3. Carry out a root SWAP and repeat steps 1 and 2. Repeat this procedure until all possible configurations have been achieved.

This algorithm works by isolating simpler subproblems at each step and solving these easier problems separately. For example, during steps 1 and 2, we essentially consider the root nodes as fixed. In the first pass of the algorithm, this reduces the number of possible configurations from $\binom{8}{4} = 70$ to $2\binom{6}{3} = 40$. Once these 40 configurations have been realized, a root SWAP is carried out and the procedure is repeated. Similar logic applies to the inner SWAPs; between each inner SWAP operation, we are reducing the problem by considering only the new configurations that can be realized through outer SWAPs alone. There will be a total of 3 root SWAPs needed, with fewer new configurations each time. Specifically, there are 30 configurations remaining after applying the first root SWAP. After the second root SWAP, there are 10, and after the third, just 2 remain.

As a note, at each step in the above procedure, we only carry out SWAPs that result in a new configuration that has not yet been realized. As such, there will be fewer new configurations reachable through outer SWAPs with each new inner and root SWAP as the total number of possible configurations dwindles. It is easy to check which outer SWAPs will give new configurations and which will not by tracking the configurations that have already been realized. With this in mind, we see that there is a clear pattern to the first pass of inner and outer SWAPs, but subsequent passes after the first root SWAP will be shorter, with SWAPs missing from the pattern where we omit SWAPs which would result in a configuration that has already been achieved in a previous cycle with a different ordering of the indices. The full schedule giving the order of all SWAP operations needed to achieve every possible configuration for the 8-mode case is provided in appendix A.

A helpful way to think about the schedule outlined above is to consider each outer SWAP as a switch that can be turned on or off. Realizing every new configurations achievable using only outer SWAPs as described in step 1 of the algorithm is equivalent to implementing all possible configurations of active and inactive SWAPs. A similar approach can also be used to see a pattern in the choices of inner SWAPs. A simple code providing a class for carrying out outer, inner, and root SWAPs and tracking tree configurations yet to be realized is available on GitHub [6].

3.3 Discussion of Optimality

Carrying out the schedule described above and listed out in appendix A, we end up executing a total of 48 SWAP cycles. The total number of root SWAPs is 3, the total number of inner SWAPs is 11, and the total number of outer SWAPs is 41, with some of these done in parallel.

We can be certain that this schedule is optimal at least in the sense of outer SWAP minimization; a new configuration is achieved after every single outer SWAP. Additionally, no matter how we organize the other SWAPs, we will always require at least 3 root SWAPs. This is because we consider the root node of the lower tree as fixed, so we must cycle through at least 4 roots in the upper tree in order to achieve every combination of 4 nodes. The schedule as listed therefore also arrives at the smallest possible number of root SWAPs.

Global optimality over each SWAP type, however, remains to be proven. Specifically, it is unclear whether a different organization of inner and outer SWAPs could potentially result in fewer inner SWAPs overall. In addition, the very last outer SWAP in the schedule is preceded by a root SWAP and an inner SWAP together. This means 3 SWAP cycles in a row before achieving a new configuration. While this is the only time in the schedule that this happens, with all other root and inner SWAPs followed directly by an outer SWAP, it is possible that a different ordering of the SWAP operations could eliminate this root-inner-outer sequence at the very end. Further investigation is needed to evaluate these possibilities before we can confidently claim optimality in terms of total SWAP cycles.

4 Conclusion and Future Work

In this work, we propose a hardware topology on which continuous-variable quantum simulation of quartic Hamiltonians can be efficiently implemented. We define different types of SWAP operations on this hardware, and provide an algorithm for scheduling these SWAP operations on this hardware in the case of 8 qumode systems. We show that this schedule is optimal in the sense of minimizing outer and root SWAP operations, though we note that more work must be done to ensure optimality over all types of SWAP operations.

It should be noted that this is only the first step for providing a complete compiler optimizer for implementing the simulation problem. What we have done here is to provide an algorithm for efficiently scheduling SWAP operations in the worst case, where every possible fourth order term in (4) is nonzero. In practice, it is likely that only some of these terms are nonzero, in which case further optimization must be done. Future work might involve translating some of the ideas from Ref. [2] designed for the 2-node interactions of QAOA to the case of the quartic interactions discussed here. These approaches include graph partition and subgraph isomorphism methods.

Finally, we designed the schedule only for realizing all possible 4-node interactions between two tree units. In the case of a larger system following our proposed structure with more than 2 trees, the problem becomes much more difficult. It is possible that the 8-node case considered here could be the base case for a recursive algorithm involving more connected trees. Future work would involve developing this idea to extend the algorithm presented here to this more general case.

References

[1] Y. T. Lin, R. B. Lowrie, D. Aslangil, Y. Subaşı, and A. T. Sornborger, "Koopman von neumann mechanics and the koopman representation: A perspective on solving nonlinear dynamical systems with quantum computers," 2022.

[2] Y. Jin, J. Luo, L. Fong, Y. Chen, A. B. Hayes, C. Zhang, F. Hua, and E. Z. Zhang, "A structured method for compilation of qaoa circuits in quantum computing," 2022.

[3] L. S. Madsen, F. Laudenbach, M. F. Askarani, *et al.*, "Quantum computational advantage with a programmable photonic processor," *Nature*, vol. 606, pp. 75–81, Jun 2022.

[4] J. M. Arrazola, V. Bergholm, K. Brádler, *et al.*, "Quantum circuits with many photons on a programmable nanophotonic chip," *Nature*, vol. 591, pp. 54–60, Mar 2021.

[5] T. Kalajdzievski and N. Quesada, "Exact and approximate continuous-variable gate decompositions," *Quantum*, vol. 5, p. 394, Feb 2021.

[6] S. Cochran, "GitHub - samco7/cv-qc-circuit-opt — github.com." https://github.com/samco7/cv-qc-circuit-opt/tree/main. [Accessed 02-05-2024].

[7] T. Kalajdzievski and N. Quesada, *Exact and approximate continuous-variable gate decompositions.* PhD thesis, York University, 2020.

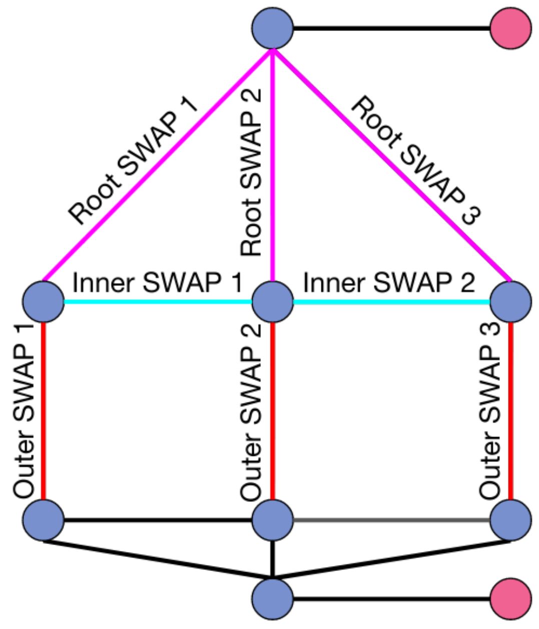

Figure 4: Two-tree structure with labels assigned to each defined SWAP operation. Note that the lower tree will only participate in outer SWAP operations, with inner and root SWAPs being performed exclusively on the upper tree.

A Listing all SWAP Operations for the 8-Mode Case

In this section we provide a full ordered list of the SWAP operations needed for the 8-mode case. Carrying out this schedule will achieve each of the 70 possible configurations, with a new configuration achieved after every outer SWAP. In this list, we label each operation in terms of the edge labels rather than node labels, following the diagram in Figure 4. Operations done in parallel are denoted as a single SWAP cycle using the & symbol.

1. Outer SWAP 1
2. Outer SWAP 2
3. Outer SWAP 1
4. Outer SWAP 3
5. Outer SWAP 2
6. Outer SWAP 1
7. Outer SWAP 2
8. Inner SWAP 1
9. Outer SWAP 1
10. Outer SWAP 3
11. Outer SWAP 1 & 2
12. Outer SWAP 3
13. Inner SWAP 2
14. Outer SWAP 2
15. Outer SWAP 1
16. Outer SWAP 2 & 3
17. Outer SWAP 1
18. Inner SWAP 1
19. Outer SWAP 2
20. Outer SWAP 1 & 2 & 3
21. Inner SWAP 2
22. Outer SWAP 2
23. Outer SWAP 1 & 2 & 3
24. Root SWAP 1
25. Outer SWAP 1
26. Outer SWAP 2
27. Outer SWAP 3
28. Outer SWAP 2
29. Inner SWAP 1
30. Outer SWAP 2
31. Outer SWAP 3
32. Inner SWAP 2
33. Outer SWAP 2
34. Outer SWAP 2 & 3
35. Inner SWAP 1
36. Outer SWAP 2
37. Inner SWAP 2
38. Outer SWAP 3
39. Root SWAP 3
40. Outer SWAP 3
41. Outer SWAP 2
42. Inner SWAP 2
43. Outer SWAP 2
44. Inner SWAP 1
45. Outer SWAP 2
46. Root SWAP 3
47. Inner SWAP 2
48. Outer SWAP 2

B Gate Decomposition Details

In this section, we give more details about the gate decompositions needed to implement the cubic cross terms and quartic terms in (5). These decompositions follow the derivations provided in [5, 7].

B.1 Cubic Cross Terms

Because quadratures on different qumodes commute with each other, each of the cubic cross terms can be reduced to one of two cases. For the first case, we consider terms of the form

$$\exp\bigl(-i\delta t F_{jjk} Q_j^2 Q_k\bigr). \tag{7}$$

It can be shown [7] that

$$\begin{aligned}\exp\bigl(i3\alpha^2 t Q_j^2 P_k\bigr) &= \exp(i\alpha Q_j Q_k)\exp\bigl(itP_k^3\bigr)\exp(-i\alpha Q_j Q_k)\exp\bigl(-itP_k^3\bigr)\\ &\quad \exp(-i\alpha Q_j Q_k)\exp\bigl(itP_k^3\bigr)\exp(i\alpha Q_j Q_k)\exp\bigl(-itP_k^3\bigr)\exp\left(i\frac{3\alpha^3 t}{4}Q_j^3\right).\end{aligned} \tag{8}$$

Note that the right hand side of (8) can be implemented using only elementary gates. Choosing $\alpha = \sqrt{F_{jjk}}$ and $t = -\delta t/3$ allows us to implement (7) as $\mathcal{F}_k^\dagger \exp\bigl(i3\alpha^2 t Q_j^2 P_k\bigr)\mathcal{F}_k$.

For the second case, we consider terms of the form

$$\exp(-i\delta t F_{jkl} Q_j Q_k Q_l). \tag{9}$$

It can be shown [5] that

$$\begin{aligned}\exp(i\alpha Q_j Q_k Q_l) &= \exp\Bigl(i\frac{\alpha}{6}(Q_j+Q_k+Q_l)^3\Bigr)\exp\Bigl(-i\frac{\alpha}{6}(Q_j+Q_k)^3\Bigr)\\ &\quad\exp\Bigl(-i\frac{\alpha}{6}(Q_j+Q_l)^3\Bigr)\exp\Bigl(-i\frac{\alpha}{6}(Q_k+Q_l)^3\Bigr)\\ &\quad\exp\Bigl(i\frac{\alpha}{6}Q_j^3\Bigr)\exp\Bigl(i\frac{\alpha}{6}Q_k^3\Bigr)\exp\Bigl(i\frac{\alpha}{6}Q_l^3\Bigr),\end{aligned} \tag{10}$$

where

$$\exp\bigl(i\alpha(Q_j+Q_k)^3\bigr) = \exp(iP_jQ_k)\exp\bigl(i\alpha Q_j^3\bigr)\exp(-iP_jQ_k) \tag{11}$$

and

$$\exp\bigl(i\alpha(Q_j+Q_k+Q_l)^3\bigr) = \exp(iP_jQ_l)\exp\bigl(i\alpha(Q_j+Q_k)^3\bigr)\exp(-iP_jQ_l). \tag{12}$$

These equations can be used to implement (9) using elementary gates. These two cases, combined with the cubic phase gate, cover all possible cubic terms that can exist in (5).

B.2 Quartic Terms

Observe that all remaining terms will have the form

$$\exp\bigl(-itFQ_i^{s_1}Q_k^{s_2}Q_j^{s_3}Q_l^{s_4}\bigr), \tag{13}$$

where $s_1+s_2+s_3+s_4 = 4$. It can be shown [5] that

$$Q_j^{s_1}Q_k^{s_2}Q_l^{s_3}Q_m^{s_4} = \frac{1}{4!}\sum_{v_1=0}^{s_1}\sum_{v_2=0}^{s_2}\sum_{v_3=0}^{s_3}\sum_{v_4=0}^{s_4}(-1)^{\sum_{i=1}^4 v_i}\binom{s_1}{v_1}\binom{s_2}{v_2}\binom{s_3}{v_3}\binom{s_4}{v_4}\Bigl(\sum_{i=1}^{4}h_iQ_i\Bigr)^4 \tag{14}$$

where $s = s_1+s_2+s_3+s_4$ and $h_i = s_i/2 - v_i$. Because all of the terms in this sum commute, the exponent of the sum will equal the product of the exponents of the individual summands. Each summand on the right hand side can be constructed in terms of elementary gates using unitary conjugation [5], as we will show below. Each quartic term in (5) can be written in one of five forms:

1. $\exp\bigl(-i\delta t F_{jjjj} Q_j^4\bigr)$
2. $\exp\bigl(-i\delta t F_{jjjk} Q_j^3 Q_k\bigr)$
3. $\exp\bigl(-i\delta t F_{jjkk} Q_j^2 Q_k^2\bigr)$
4. $\exp\bigl(-i\delta t F_{jjkl} Q_j^2 Q_k Q_l\bigr)$
5. $\exp\bigl(-i\delta t F_{jklm} Q_j Q_k Q_l Q_m\bigr)$

As shown in [5], the first form can be decomposed as follows. First, we use the identity

$$Q_j^4 = (Q_j^2 + Q_k)^2 - Q_k^2 - 2Q_j^2 Q_k, \tag{15}$$

which implies that

$$e^{iCQ_j^4} = e^{iC(Q_j^2+Q_k)^2} e^{-iCQ_k^2} e^{-iC2Q_j^2Q_k}. \tag{16}$$

The second term on the right hand side of (16) can be implemented using a quadratic phase gate, and a decomposition for the third term is given in (8) (up to a single-mode Fourier transform). The first term can be decomposed as

$$e^{iC(Q_j^2+Q_k)^2} = e^{iQ_j^2P_k} e^{iCQ_k^2} e^{-iQ_j^2P_k}, \tag{17}$$

which is now also in terms of quadratic phase gates and the decomposition from (8). We now have a decomposition of $e^{iCQ_j^4}$ in terms of elementary gates.

Applying (14) to any of the remaining four forms (2-5) will result in a product of exponents where each factor takes the form

$$\exp\bigl(iC(a_1Q_1 + a_2Q_2 + a_3Q_3 + a_4Q_4)^4\bigr), \tag{18}$$

where each $a_j \in \{\frac{N}{2} : N \in \mathbb{Z}\}$ and C is a scalar constant. Note that a_j may be zero in some cases. Assume without loss of generality that $a_1 = 1$. Using the unitary conjugation formula given in [7], we can decompose (18) as

$$e^{iC(Q_1+a_2Q_2+a_3Q_3+a_4Q_4)^4} = e^{i(a_4Q_4P_1)} e^{i(a_3Q_3P_1)} e^{i(a_2Q_2P_1)} e^{iCQ_1^4} e^{-i(a_2Q_2P_1)} e^{-i(a_3Q_3P_1)} e^{-i(a_4Q_4P_1)}, \tag{19}$$

resulting in an expression that is exactly equivalent to a sequence of elementary gates. We simply use the decomposition given in (16) and (17) surrounded by controlled-X gates.