Optimized Compilation of Aggregated Instructions for Realistic Quantum Computers

Yunong Shi
The University of Chicago
yunong@uchicago.edu

Nelson Leung
The University of Chicago
nelsonleung@uchicago.edu

Pranav Gokhale
The University of Chicago
pranavgokhale@uchicago.edu

Zane Rossi
The University of Chicago
zmr@uchicago.edu

David I. Schuster
The University of Chicago
david.schuster@uchicago.edu

Henry Hoffmann
The University of Chicago
hankhoffmann@cs.uchicago.edu

Frederic T. Chong
The University of Chicago
chong@cs.uchicago.edu

Abstract

Recent developments in engineering and algorithms have made real-world applications in quantum computing possible in the near future. Existing quantum programming languages and compilers use a quantum assembly language composed of 1- and 2-qubit (quantum bit) gates. Quantum compiler frameworks translate this quantum assembly to electric signals (called control pulses) that implement the specified computation on specific physical devices. However, there is a mismatch between the operations defined by the 1- and 2-qubit logical ISA and their underlying physical implementation, so the current practice of directly translating logical instructions into control pulses results in inefficient, high-latency programs. To address this inefficiency, we propose a universal quantum compilation methodology that aggregates multiple logical operations into larger units that manipulate up to 10 qubits at a time. Our methodology then optimizes these aggregates by (1) finding commutative intermediate operations that result in more efficient schedules and (2) creating custom control pulses optimized for the aggregate (instead of individual 1- and 2-qubit operations). Compared to the standard gate-based compilation, the proposed approach realizes a deeper vertical integration of high-level quantum software and low-level, physical quantum hardware. We evaluate our approach on important near-term quantum applications on simulations of superconducting quantum architectures. Our proposed approach provides a mean speedup of 5×, with a maximum of 10×. Because latency directly affects the feasibility of quantum computation, our results not only improve performance but also have the potential to enable quantum computation sooner than otherwise possible.

ACM Reference Format:
Yunong Shi, Nelson Leung, Pranav Gokhale, Zane Rossi, David I. Schuster, Henry Hoffmann, and Frederic T. Chong. 2019. Optimized Compilation of Aggregated Instructions for Realistic Quantum Computers. In *2019 Architectural Support for Programming Languages and Operating Systems (ASPLOS '19), April 13–17, 2019, Providence, RI, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3297858.3304018

Introduction

The past twenty years have seen the world of quantum computing moving closer to solving classically intractable problems [2, 9, 50]. With developments in Noisy Intermediate-Scale Quantum (NISQ) [45] devices like IBM's quantum machine with 50 qubits and Google's quantum machine with 72 qubits, we may soon be able to demonstrate computations not possible on classical supercomputers [2, 9]. Exciting classical-quantum hybrid algorithms tailored for NISQ machines, like Quantum Approximate Optimization Algorithm (QAOA) [8] and Variational Quantum Eigensolver (VQE) [36, 44] will power up the first real-world quantum computing applications with scientific and commercial value.

Computation latency is a major challenge for near-term quantum computing. While all computing systems benefit from reduced latency, in a quantum system the output fidelity decays at least exponentially with latency [41]. Thus, in near-term quantum computers, reducing latency is not just a minor convenience—latency reduction actually enables new computations on near-term machines by ensuring that the computation finishes before the qubits decohere and produce a useless result. Thus latency reduction is critical

ASPLOS '19, April 13–17, 2019, Providence, RI, USA

ACM ISBN 978-1-4503-6240-5/19/04…$15.00
https://doi.org/10.1145/3297858.3304018

to enabling quantum computing applications on near-term NISQ devices.

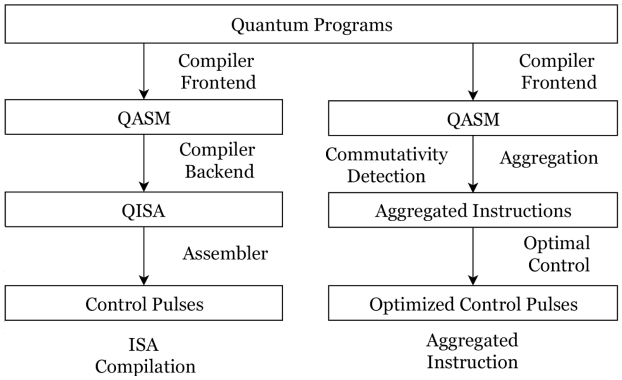


Fig. 1. Comparison of two compilation schemes. Gate-based compilation with ISA abstraction (left) follows a classical compilation approach, but could generate unoptimized quantum operations in the hardware. Our proposed approach (right) produces highly optimized control pulses.

Unfortunately, existing quantum computing abstractions (which mirror classical computer system stacks, as shown on the left side of Figure 1) introduce inefficiencies that greatly impact latency. In these *gate-based* approaches, programs are compiled into quantum assembly instructions (or *gates*) that specify 1- and 2-qubit operations [10, 18, 51]. This quantum assembly is a virtual ISA which represents a rich set of operations. These gates must then be translated into *control pulses*—the electrical signals that implement the specified operations on the underlying physical hardware. Typically though, the underlying hardware implements a different set of operations, and there is a mismatch between the expressive logical gates and the set of instructions that can be efficiently implemented on a real system. In contrast, physicists have developed a set of techniques—*quantum optimal control*—that ignore abstraction barriers and produce customized control pulses that minimize latency for a particular computation on a physical system [12]. To draw an analogy to classical computer systems, the gate-based compilation approach is similar to the compiler-architecture-microarchitecture stack, while quantum optimal control is similar to customized circuit design. Quantum optimal control techniques do not scale, however, and are impractical for computations using more than 10 qubits [32], i.e., emerging NISQ systems.

In this paper we propose a quantum compilation technique that optimizes across existing abstraction barriers to greatly reduce latency while still being practical for large numbers of qubits. Specifically, rather than directly translating 1- and 2-qubit gates to control pulses, our framework aggregates these small gates into larger operations, as illustrated in the right side of Figure 1. Our framework can then manipulate these aggregates in two ways. First, it finds commutative operations that allow for much more efficient schedules of control pulses. Second, it uses quantum optimal control on the aggregates to produce a set of control pulses that is optimized for the the underlying physical architecture. Our technique greatly improves efficiency over the existing gate-based compilation methods while mitigating the scalability problem of quantum optimal control methods. Because ours is a software-based approach, these results can see practical implementation much faster than experimental approaches for improving physical device latency. We compare our methodology to standard gate-based compilation on important near-term quantum algorithms and find that our technique produces a mean speedup of 5× with a maximum speedup of 10×.

We achieve these speedups via two novel techniques:

- detecting diagonal unitaries and scheduling commutative instructions to reduce the critical path of computation.
- blocking quantum circuits in a way that scales optimal control beyond 10 qubits without compromising parallelism

For quantum computers, achieving these speedups (and thereby reducing latency) is do-or-die: if circuits take too long, the qubits decohere by the end of the computation. By reducing latency 2-10x, our methodology provides an accelerated pathway to running useful quantum algorithms, without needing to wait years for hardware with 2-10x longer qubit lifetimes.

Background

This section presents a brief overview of the relevant background on quantum computation and quantum optimal control.

Principles of quantum computation

The qubit (quantum bit) is the basic element of a quantum computing system. In contrast to classical bits, qubits are capable of living in a superposition of the logical states $|0\rangle$ and $|1\rangle$. The general quantum state of a qubit is represented as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α, β are complex coefficients with $|\alpha|^2 + |\beta|^2 = 1$. When measured in the 0/1 basis, the quantum state collapses to $|0\rangle$ or $|1\rangle$ with probability of $|\alpha|^2$ and $|\beta|^2$, respectively. It is helpful to visualize a qubit as a point on a 3D sphere called the Bloch sphere [1, 41], as depicted in Figure 2.1. Qubits can be realized on different Quantum Information Processing (QIP) platforms, including superconducting circuits [7], ion traps [30], and quantum dots systems [33].

The number of quantum logical states grows exponentially with the number of qubits in a quantum system. For example, a system with 3 qubits can live in the superposition of 8 logical states: $|000\rangle$, $|001\rangle$, $|010\rangle$, ..., $|111\rangle$. This property sets the foundation of potential quantum speedup over classical computation—an exponential number of correlated

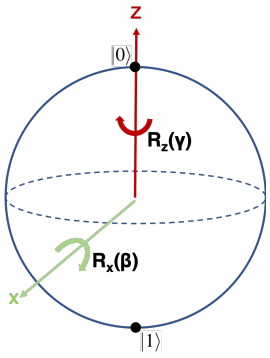


Fig. 2. The Bloch Sphere represents a single qubit. The $|0\rangle$ state is on the North Pole, the $|1\rangle$ state is on the South pole, and superposition states are in between. Single qubit gates correspond to rotations on the Bloch sphere. For instance, the $R_x(\beta)$ gate rotates a qubit by angle β about the x-axis.

logical states can be stored and processed simultaneously by a quantum system with a linear number of qubits.

Quantum gates

In the process of quantum compilation, quantum algorithms are first decomposed into a set of universal 1- and 2-qubit discrete quantum operations called logical quantum gates. All gates are represented in matrix form as unitary matrices. 1-qubit gates correspond to rotations along a particular axis on the Bloch sphere. In the standard ISA for quantum computation, the 1-qubit gate set includes rotations along the x-, y-, z-axes of the Bloch sphere, *i.e.* R_x, R_y, R_z gate. It also includes the Hadamard gate, which corresponds to rotation about the diagonal x+z axis. An example of a 2-qubit logical gate is the Controlled-NOT (CNOT) gate, which flips the state of the target qubit iff the control qubit is $|1\rangle$. For example, the CNOT gate sends $|10\rangle$ to $|11\rangle$, sends $|11\rangle$ to $|10\rangle$, and preserves the other logical states.

Because it is typically not obvious how to implement the CNOT gate directly on a physical platform, a CNOT gate is further decomposed into physical gates in standard gate-based compilation. Appendix A provides a description of 2-qubit physical gates on different quantum platforms. For the benchmarks we present in this paper (Section 5), we focus on superconducting architectures with the iSWAP physical gate because it is easy to implement and its optimized compilation is relatively unexplored.

Quantum control

Quantum computing systems can be continuously driven by external physical operations to any state in the space spanned by the logical states. The physical operations, called control fields, are specific to the underlying system, with control fields and system characteristics controlling a unique and time-dependent quantity called the Hamiltonian. The Hamiltonian determines the evolution path of the quantum states. For example, in superconducting systems, we can drive a qubit to rotate continuously on the Bloch sphere by applying microwave electrical signals [3]. By varying the intensity of the microwave signal, we can control the speed of the qubit's rotation. The ability to engineer the system Hamiltonian in real-time allows us to direct the qubits to the quantum state of interest through precise control of related control fields. Thus, quantum computing is achieved by constructing a quantum system in which the Hamiltonian evolves in a way that aligns with a computational task, yielding the desired result with high probability upon final measurement of the qubit system. In general, the path to a final quantum state is not unique and finding the optimal evolution path is an open problem [12, 32, 49].

In the context of quantum control, quantum gates can be regarded as a set of pre-programmed control fields performed on the quantum system.

The mismatch between gates and control

The coarse-grained abstraction of quantum gates can complicate the continuous evolution of the underlying quantum states, meaning that the pre-programmed control fields might not lead to the optimal evolution path of the quantum system. We consider two simple examples to illustrate this mismatch.

- In the first example, consider the gate sequence of a CNOT gate followed by a R_z gate. In standard gate-based compilation, these two logical gates will be further decomposed into physical gates and be executed sequentially. However, on superconducting platforms, the control fields that implement the two gates can be applied simultaneously. Hence, in this case, the gate model is suboptimal as it serializes the circuit and thus increases the circuit latency.
- As another example, consider the traditional ISA decomposition of the SWAP operation into three Controlled-NOT (CNOT) operations, as realized in the circuit below. This decomposition is equivalent to the implementation of in-place memory SWAPs with three alternating XORs in classical computation. For systems like quantum dots [33], the SWAP operation is directly supported by applying particular constant control fields for a certain period of time. In this case, decomposing a SWAP into three CNOTs introduces substantial overhead.

In experimental physics settings, equivalences from simple gate sequences to control pulses can be hand optimized [48]. However, when circuits become larger and more complicated, this kind of hand optimization become less efficient and the standard decomposition becomes less favorable, motivating a shift toward numerical optimization methods that are not limited by the ISA abstraction.

Quantum optimal control

Quantum optimal control algorithms find the optimal evolution path from a starting quantum state to a final quantum

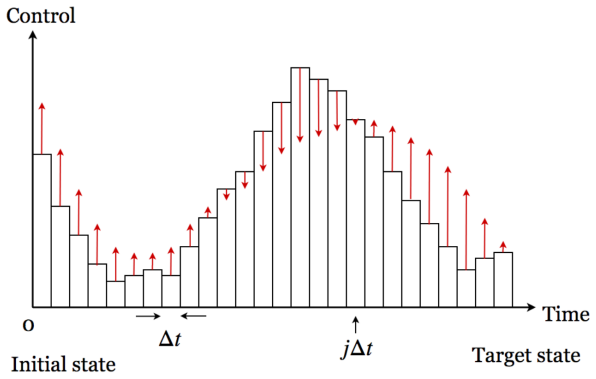


Fig. 3. Quantum optimal control based on gradient descent, for a simplified single-pulse-type example. The black bars indicate the current iteration's proposed sequence of control pulse amplitudes by time interval, $\mu(j)$. The red arrows indicate the gradient of the output fidelity with respect to each $\mu(j)$. Thus, at the next iteration, each amplitude should be updated to $\mu(j)+\epsilon\frac{\partial L}{\partial \mu(j)}$, where L is the targeted loss function and ϵ is the adaptive step size.

state, typically by performing gradient descent methods, such as the GRadient Ascent Pulse Engineering (GRAPE) [5, 27] algorithm. For a quantum system with a set of external control fields $u_1, \ldots, u_M$ that can be tuned in real-time, optimal control minimizes deviations from a target state by adjusting each control field u. In GRAPE, at every iteration the gradient of the target loss function (usually fidelity) with respect to a control field μ_k at time step j in the evolution can be explicitly calculated by solving Schrödinger's equation. The algorithm will update the control field $\mu_k(j)$ in the direction of the gradient with adaptive step size ϵ [5, 27, 32] (Figure 3). With enough iterations, the converged control pulses are expected to drive the system from the initial state to the final state along an optimized path.

Gradient methods' running time and memory use grow exponentially with the size of the quantum system. In our work, we are able to numerically optimize quantum systems of up to 10 qubits with the GPU accelerated optimal control unit [32].

Compilation methodology

In this section, we demonstrate by example the advantage of our approach over standard gate-based compilation. Next we present our compilation methodology and introduce its end-to-end tool flow, including the frontend, backend, the optimal control unit, and verification procedure. In Section 4, we will detail the instruction aggregation algorithms.

Gate	CNOT	SWAP	H	$R_z(\gamma)$	$R_x(\beta)$
Time (ns)	47.1	50.1	13.7	9.8	6.1
Gate	G_1	G_2	G_3	G_4	G_5
Time (ns)	54.9	13.7	42.0	31.4	6.1

Tab. 1. Instruction execution time for QAOA circuit in Figure 4 (a). The pulse time for each gate in this table is optimized by an optimal control unit (see section 3.5). For the SWAP gate, we don't use the standard 3 alternating CNOTs implementation but optimize it individually.

An example of QAOA circuit

Figure 4 (a) shows a quantum circuit that solves the MAX-CUT problem for a triangle.[1] The circuit is decomposed into a standard gate set. This circuit (or variants of it up to single qubit gates) can be reproduced by most quantum software platforms, including ScaffCC [20], QISKit [4] and Pyquil [52]. We generate this circuit using ScaffCC. To keep our example small and realistic, we assume 1D nearest neighbor qubit connectivity and a underlying superconducting architecture. A SWAP gate is inserted to satisfy the qubit connectivity constraint. We choose to set the 1-qubit control field limit 5× the 2-qubit control field limit as a representative of real experimental settings [3]. The total execution time using gate-based compilation in Figure 4 (a) is found by adding up the pulse time of each individual gate on the critical path of the circuit: $6T(CNOT)+T(SWAP)+T(H)+3T(R_z)+T(R_x) = 381.9$ns using the numbers in Table 1.

In contrast, our compiler automatically generates the aggregated instruction set $G_1 - G_5$ as indicated in Figure 4 (b), and uses optimal control to produce minimal latency pulses for each. The pulse time for the circuit has critical path: $T(G_1) + T(G_3) + T(G_4) = 128.3$ns. In this example, our proposed aggregated instruction compilation reduces the pulse duration by about 2.97× compared to standard gate-based compilation methods. Figure 4 (c) and (d) compare the pulses for G_3 generated by gate-based compilation and generated by the optimal control unit.

Methodology overview

Figure 5 illustrates the key innovations in our proposed compilation scheme compared to standard gate-based compilation. Both approaches take a quantum program as input and proceed through a series of transformations to produce the control pulses that implement the computation on the physical qubits. In the traditional gate-based approach, the compiler first produces flattened quantum assembly codes, then generates a schedule of the logical instructions in the assembly codes. This schedule is later turned into a schedule of physical instructions by decomposing the logical instructions into physical instructions, which are converted into

[1] Specifically, the circuit implements the QAOA [8], one of the most promising near-term quantum algorithms, with angle parameters γ and β determined by variational methods [36] and set to 5.67 and 1.26.

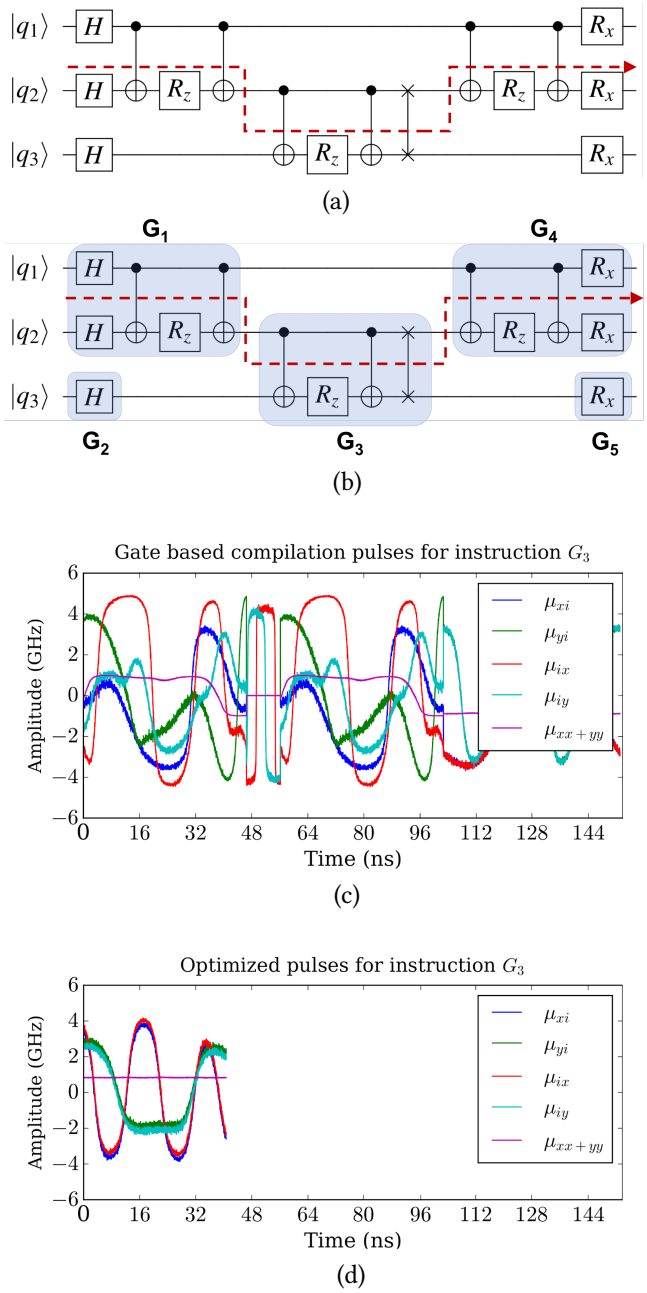
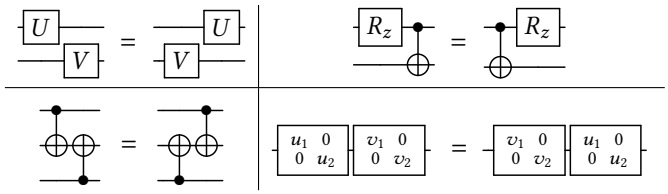


Fig. 4. Example of a QAOA circuit demonstrating the difference between gate-based compilation and our compilation methodology. (a) Standard circuit (red arrow indicates the critical path). (b) Circuit with aggregated instructions. (c) Standard compilation pulses for G_3. (d) Aggregated compilation pulses for G_3. Each line represents the intensity of a control field. The pulse sequence in (d) is much shorter in duration and easier to implement than that of (c).



Tab. 2. Examples of gate commutation relations. Clockwise from top-left: gates acting on different qubits commute, controls commute with Z-axis rotations, gates with diagonal matrices commute, and CNOTs with disjoint controls commute.

control pulses. We note that in the traditional gate-based approach, the physical properties of the underlying hardware are "localized" in each physical instruction. Compared to the traditional approach, our compilation process first converts assembly codes to a logical schedule that explores more commutativity by aggregating highly commutative instructions. Unlike traditional logical scheduling, our compiler aggregate highly commutative intermediate instructions in the assembly codes and generates a much more efficient logical schedule by re-arranging the new instructions. The logical schedule is then converted to a physical schedule after qubit mapping and SWAP gate insertion. At this point the compiler aggregates the final instructions and applies optimal control to the aggregated instructions. The goal is to find the optimal aggregation that produces the lowest-latency control pulses for the specified computation while considering aggregations that are small enough to be processed by the quantum optimal control unit. Output is an optimized physical schedule along with the corresponding optimized control pulses.

Compilation frontend

The compiler frontend accepts quantum programs from the user, lowering high-level descriptions of quantum algorithms to a logical assembly that retains gate dependence relations. The compiler frontend performs program level analysis and preliminary logical level optimization, including loop unrolling, module flattening, commutativity detection, and logical level scheduling. The logical assembly output from the compiler frontend can be abstracted as a gate dependence graph (GDG) for each program.

Quantum GDG:

The main difference between a quantum GDG and a classical program dependence graph (PDG) is that quantum commutation rules apply in quantum GDG. More specifically, in a quantum GDG, consecutive commuting gates do not have parent-child relations [11] and can be scheduled in any order. Important commutation relations are depicted in Table 2.

In our compiler frontend, commutation relations between two gates A, B are resolved by explicitly checking the equality of unitary operators $\hat{A}\hat{B}$ and $\hat{B}\hat{A}$.

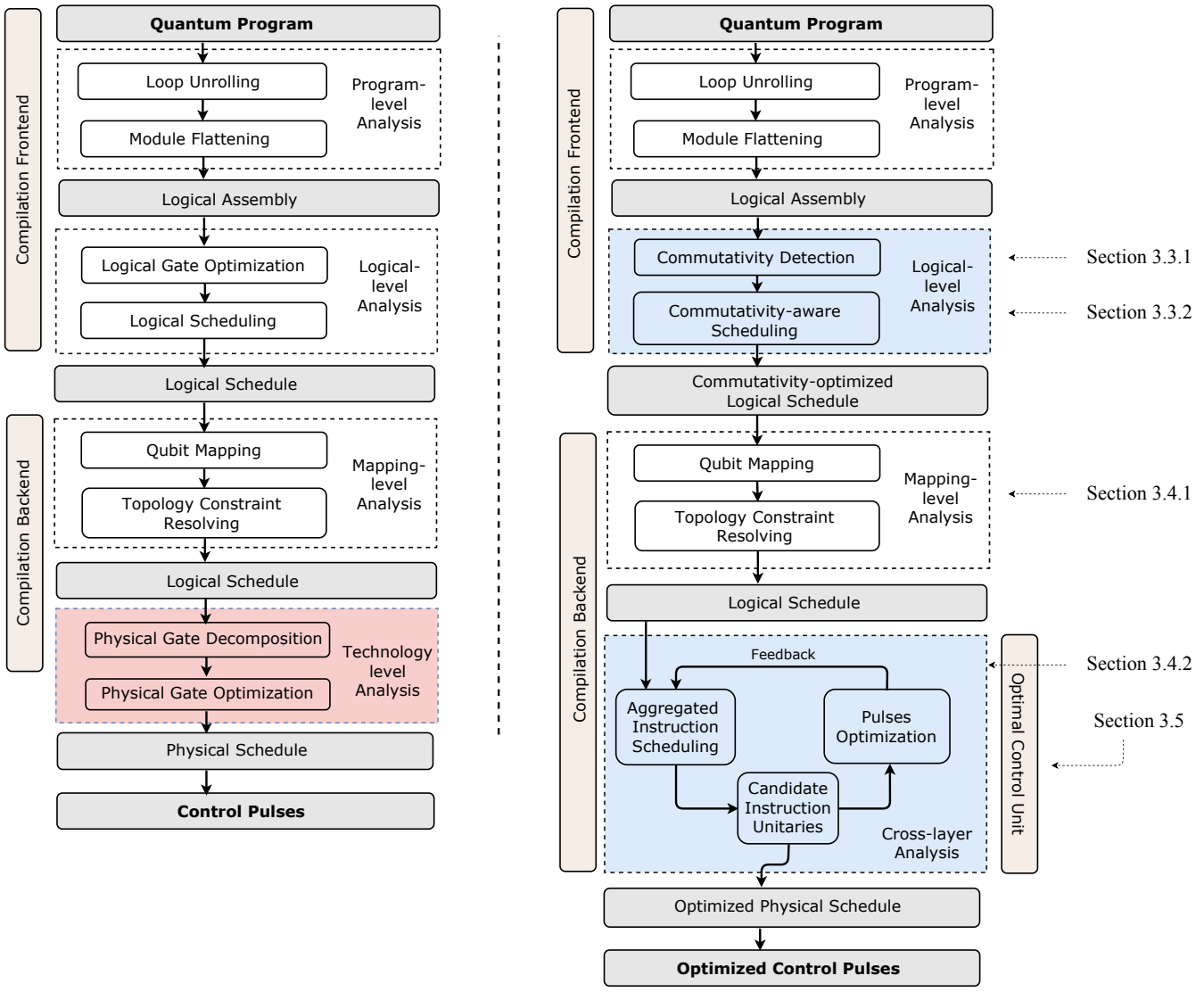


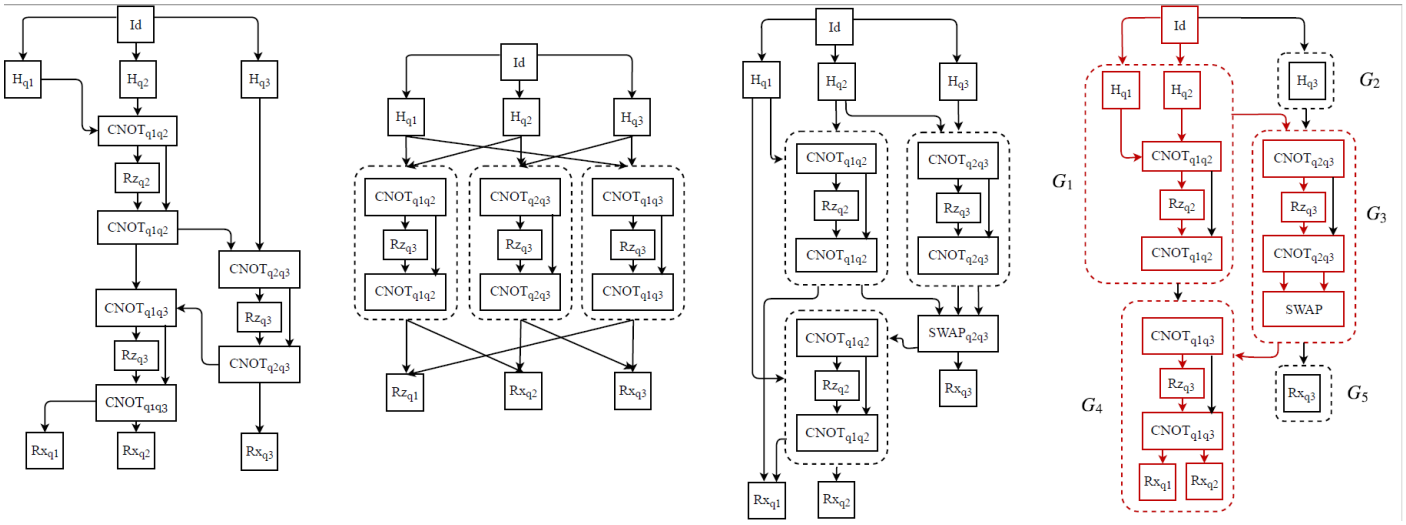
Fig. 5. The comparison between standard gate-based compilation (left) and our compilation approach (right). The key differences are highlighted by the colored areas. In the first blue box, our compiler detects potential commutativity, which opens up opportunities for much more efficient scheduling. Then our logical scheduling takes advantage of commutativity for better parallelization. In the second blue box, by iterating with the optimal control unit, the instruction aggregation procedure breaks the well-encapsulated abstraction of 1- and 2-qubit logical gates and eliminates the physical gate layer (red box) that encodes only coarse-grained hardware information.

Figure 6 shows the GDG of the QAOA circuit in Figure 4 at different compilation stages. We insert an identity instruction as a virtual root for every GDG to connect instructions at depth 0. Because this virtual root is the identity instruction, it does not interfere with the computational result or latency. In our GDG, each path is labelled by a corresponding qubit name.

Commutativity detection:

Prior to commutativity detection, every consecutively scheduled pair of gates has a parent-child dependence. However, if a pair of gates commutes, then their relationship is a false dependence and the gates can be scheduled in either order. Our compilation technique relies heavily on the flexible scheduling of gates, so detecting commutativity and removing false dependencies in the GDG is critical for the rest of the compilation process.

In many near-term quantum applications, it is common for instructions *within* an instruction block to not commute, but for the full instruction blocks to commute with each other [8, 29]. As an example, in Figure 4, the CNOT-Rz-CNOT



(a) Module flattening (b) Commutativity detection (c) Scheduling and mapping (d) Gate aggregation

Fig. 6. The evolution of GDG for the circuit in Figure 4. In the compiler frontend, GDG in (a) is constructed for the flattened quantum program. By detecting commutative CNOT-Rz-CNOT instructions, the compiler transforms the GDG in (a) to GDG in (b) for more scheduling flexibility. Then, after scheduling and mapping, GDG has SWAP gates inserted and becomes GDG in (c). Finally, after the final aggregation, we arrive at the final GDG in (d), which is optimized both for parallelism and pulses generation. Each path in GDGs represents a qubit. The qubit name for each path is omitted in the figure for cleanness. The red paths in part (d) are the final critical paths.

structures commute with each other (these structures correspond to diagonal unitaries), but each CNOT and Rz in these structures does not commute. Thus in the GDG in Figure 6 (b), after contracting the consecutive CNOT, Rz, CNOT instructions, the compiler is able to schedule new commuting CNOT-Rz-CNOT instructions in any order, while in the GDG in Figure 6 (a), scheduling options are limited. This observation opens up opportunities for more efficient scheduling. In our design, the commutativity detection step achieves the goal of forming a highly commutative instruction set for the input quantum circuit. We detail the algorithm for commutativity detection in Section 4.

Commutativity-aware Logical Scheduling (CLS):

CLS uses commutativity, either detected from the last step or inherited from the original circuit, to extract more parallelism. With our GDG construction, it's natural to define the commutation group data structure on qubits. Each qubit maintains a list of commutation groups that, on that qubit, all the consecutive and commutative gates are in the same group. Two gates commute iff they are in the same commutation group on all the common qubits they share. This data structure facilitates more flexible scheduling and more optimization. For example, the two CNOTs in a CNOT-Rz-CNOT structure are in the same commutation group on the control qubit, but in different commutation groups on the target qubit. Then, with the commutation group data structure, we can correctly identify that any Rz gates on their control qubit can travel through these two CNOTs even though these two CNOTs do not commute. Our CLS iterates the commutation groups on qubits in circuits. At each iteration, the CLS draws candidate gates to schedule from the first non-empty commutation groups on qubits, and schedules greedily. At each step, the candidate gates form a computational graph G_c with qubits as vertices and gates as edges (1-qubit gates are self-loops on a single vertex). The computational graph of candidate gates can conflict by sharing a qubit, in which case these gates cannot be scheduled simultaneously. The CLS then finds the maximal cardinality matching of G_c to resolve the conflicts. Figure 7 illustrates an example of the maximal matching process. Algorithm 1 describes the CLS process.

Similar to previous work [11], our strategy is intended to maximize parallelism, and not to minimize the number of SWAP gates in the backend. Our motivation for this strategy in our compilation scheme is the finding that SWAP gates can be beneficial in reducing latency on superconducting architectures [48], so we don't aim to reduce the amount of SWAP gates. We also believe that a precise cost model that correctly discriminates the latency of each SWAP gate in circuits leads to more efficient scheduling strategies. We propose it as an exciting open problem.

Algorithm 1 CLS

Input:quantum GDG G_q, the list of commutation groups on qubits $\{com\_list[q_i] \mid q_i \in$ all qubits $\}$.
Output: logical schedule S.
Initialize current gates cg, next_time_point np, current commutation groups $\{com\_group[q_i] \mid q_i \in$ all qubits$\}$.
while cg not empty **do**
  candidate gates ng ={g can be scheduled at np| g in cg}
  gates to be scheduled gs = find_max_matching(ng)
  S+=gs; cg-=gs
  Update np
  for all $q_i \in$ all qubits **do**
    if $com\_group[q_i]$ empty **then**
      $com\_group[q_i] = com\_list[q_i].pop()$
    end if
  end for
  cg+=$\{g \mid g \in com\_group[q]$ for $q \in op(g)\}$
  Update com_group
end while
return S

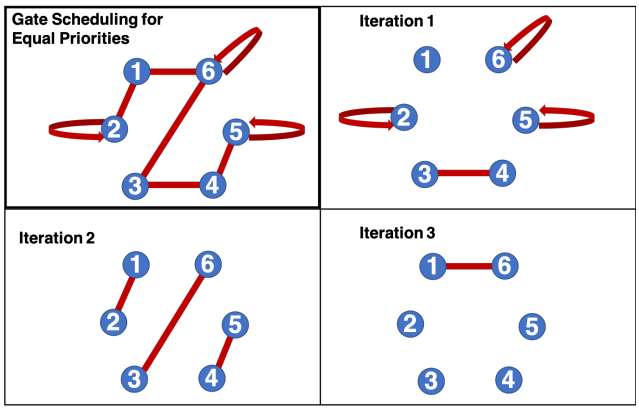


Fig. 7. A computational graph with six qubits, all instructions have the same latency. The scheduler finds a maximal matching of non-adjacent edges and schedules them. The subsequent round repeats this process on the subgraph of remaining edges.

Compiler backend

The backend is responsible for mapping level optimization and final pulse generation. The backend executes the following steps: qubit mapping, topological constraint solving, and final instruction set aggregation.

Qubit mapping & topological constraint resolution:

Our logically-scheduled instructions do not account for the topological connectivity constraints of the underlying hardware. For the benchmarks presented in this paper (Table 3), we assume a rectangular-grid qubit topology with two-qubit operations only permitted between direct neighbors. This topology is representative of typical near-term superconducting quantum computers [46].

To conform to this topology, the logically-scheduled instructions are processed in two steps. First, we place frequently interacting qubits near each other by bisecting the qubit interaction graph along a cut with few crossing edges, computed by the METIS graph partitioning library [26]. As described in previous work [13, 19], this strategy is applied recursively on the partitions, yielding a heuristic mapping that reduces the distances of CNOT operations.

Once the initial mapping is generated, two-qubit operations between non-neighboring qubits are prepended with a sequence of SWAP rearrangements that move the control and target qubits to be adjacent.

Instruction aggregation:

In this step, the compiler iterates with the optimal control unit to generate the circuit's final aggregated instructions. The optimal control unit optimizes each instruction individually. We describe how our instruction aggregation algorithm preserves parallelism in Section 4.3.

Finally, using the CLS from Section 3.3.2, the compiler schedules the circuit of aggregated instructions and sends the concatenated pulse sequences to the underlying hardware.

Optimal control unit

The optimal control unit in our compiler backend [32] provides optimized control pulses for each aggregated instruction. Our GPU accelerated quantum optimal control algorithm is based on automatic differentiation and the Tensorflow framework. Automatic differentiation allows users to specify advanced optimization criteria and easily incorporate them in pulse generation. These criteria include realistic experimental concerns like suppressing unwanted qubit levels, avoiding large voltage fluctuation, and most importantly, pulse latency.

The optimal control algorithm in our unit has been validated against real hardware and used in real experimental environments [16, 17].

Verification

Our framework uses the popular QuTip [21, 22] simulation backend to verify the quantum unitaries defined by the aggregated instructions and the resulting pulses generated by the optimal control unit. This verification procedure provides users confidence in the numerical accuracy of the results.

For our simulation (Section 5), we sample 10 aggregated instructions for each benchmark to verify that the control pulses of all instructions produce the correct unitary.

Instruction aggregation

This section details the two algorithms for aggregating instructions in Section 3.3.1 and Section 3.4.2. We first discuss the allowed action space. Then we move onto our aggregation algorithms.

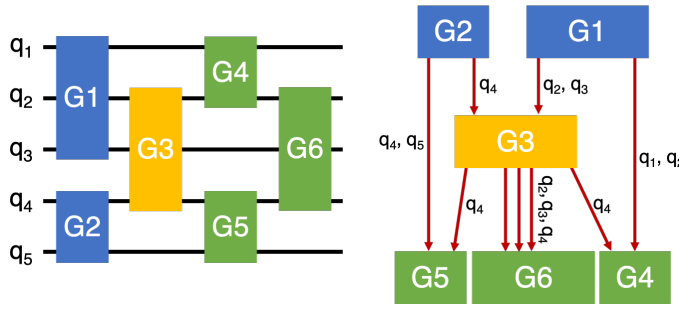


Fig. 8. A circuit demonstrating the action space of instruction aggregation, with the corresponding GDG to the right. Gates in the same color group commute. G_3 can aggregate with any of the other gates. Only the action of aggregating G_3 and G_6 is monotonic in this circuit. All other aggregation pairs induce serialization upon the circuit by delaying a dependent aggregated instruction.

Action space for instruction aggregation

Here we define the allowed action space on GDG, where two instructions can aggregate if the following are true: 1. the two instructions overlap (share some common qubits); 2. one is the parent of the other on every qubit path they share or they are siblings; 3. If the two gates have parent-children relations, the parent (the children) either commutes with all gates in its commutation group on their common qubits or can be scheduled last (first) in the commutation group. In this way, we enforce the pulses inside an aggregated instruction to be continuous. In practice, we also limit the number of qubits in an aggregated instruction (instruction width) because of the scalability of the optimal control unit.

Diagonal unitaries aggregation for commutativity detection

To our knowledge, the most common commutative instructions are instructions representing diagonal unitaries because diagonal unitaries are used widely in decomposition methods of quantum chemistry applications [29] and near-term optimization algorithms [8]. To preserve parallelism, we only detect diagonal unitaries in blocks with a width of 2 qubits. To aggregate diagonal unitaries, we exhaustively search the action space defined in Section 4.1 within 2-qubit wide blocks(typically no longer than 10 gates).

Instruction aggregation

The main challenge of aggregating proper multi-qubit instructions is the conflict between parallelism and the need for larger instruction size for more speedup. Aggregating new instructions may potentially compromise parallelism. For example, in Figure 4, if G_5 is merged with G_3, then the circuit is serialized by the delay of G_4, which is dependent on G_3. To protect parallelism without querying optimal control unit too often, we make the following observation: for each aggregated instruction, the larger the instruction is, the more optimized the control pulses will be. Also, we notice that there is a set of allowed actions that will not delay critical paths even if the pulses in the new instruction are not optimized. We call these actions monotonic actions because in these actions, the reward of reducing circuit latency from aggregating a collection of instructions is strictly higher than aggregating a subset of the collection, as parallelism is not compromised. Monotonic actions can be checked by explicitly calculating the original circuit depth with the depth upon executing the action.

Our strategy is first to traverse the GDG. For each instruction in the GDG, we search the monotonic action set and keep the best action in a global table. After traversal on the GDG, we perform the global best action, and update the GDG and action table. We repeat until no more actions can be made. Then we update the latency of each aggregated instruction by querying the optimal control unit. This updated instruction latency could change the circuit structure and potentially create more monotonic actions, so we iteratively execute the above procedure until the GDG converges. For example, for the GDG in Figure 6 (c), after one iteration of instruction aggregation, we transform it to the circuit in Figure 6 (d). Figure 8 also illustrates an example of maintaining parallelism in the action space.

Evaluation

In this section, we present our simulation results. We first introduce our benchmark methodology. Then we present the main result — the latency between different compilation strategies. We conclude by analyzing the different factors that affect the final latency, including instruction width, parallelism, commutativity, and spatial locality.

Experimental setup

We perform our numerical study on superconducting architecture with XY interaction (Appendix A) and set the control field limit of XY interaction to be $\mu_{max} = 0.02$GHz and single qubit rotation control field to be $5\mu_{max}$.By setting the control field strength to less than typical transmon anharmonicity, we model transmon operations with low leakage to high level states [3].

Benchmark methodology

We select several important classical-quantum hybrid algorithms and traditional quantum applications from the NISQ era as our benchmarks. The benchmarks are chosen to have different program characteristics that will affect the improvement from the aggregated instruction compilation. The complete list of benchmarks is shown in Table 3. The first three benchmarks are QAOA circuits solving MAXCUT problems [8, 20]. These circuits are highly commutative. Ising model is a family of highly parallel circuits with limited commutativity [20]. Square root circuits use Grover's algorithm [14, 20] to find the square root of polynomials and they are very serialized. UCCSD stands for Unitary Coupled Cluster

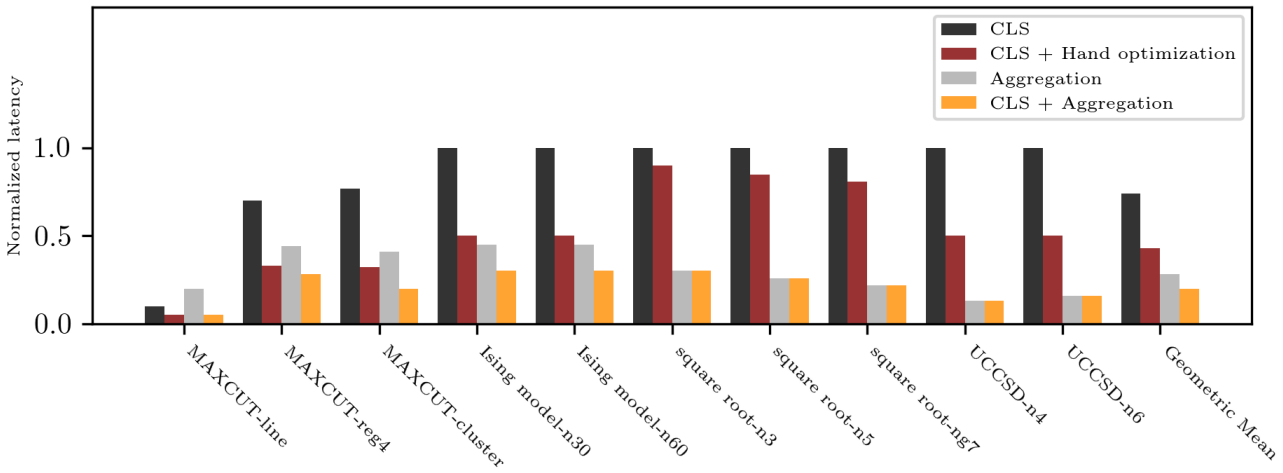


Fig. 9. Normalized circuit latency of different strategies (ISA compilation is the baseline with latency 1.0).

Benchmark	Application Purpose	Qubits	Parallelism	Spatial locality	Commutativity
MAXCUT-line	MAXCUT on a linear graph	20	Low	High	High
MAXCUT-reg4	MAXCUT on a random 4 regular graph	30	High	Medium	High
MAXCUT-cluster	MAXCUT on a cluster graph	30	Medium	Low	High
Ising model	Find ground state of Ising model	30	High	High	Medium
Ising model	Find ground state of Ising model	60	High	High	Medium
square root-n3	Grover algorithm for polynomial search	17	Low	High	Low
square root-n4	Grover algorithm for polynomial search	30	Low	High	Low
square root-n5	Grover algorithm for polynomial search	47	Low	High	Low
UCCSD-n4	UCCSD ansatz for VQE	4	Low	High	Low
UCCSD-n6	UCCSD ansatz for VQE	6	Low	Medium	Low

Tab. 3. Benchmarks

Single-Double ansatz [47] for the variational quantum eigensolver [36]. This ansatz is derived from the Jordan-Wigner or Bravyi-Kitaev transformations [29, 47] and is considered to be a machine unaware ansatz [47]. We include it to address that with optimal control, physics induced ansatzs can be made more hardware efficient on superconducting architectures and more competitive relative to machine-inspired ansatzs [24].

Latency

We present our main result in Figure 9. We compare four different strategies with unoptimized gate-based compilation. CLS refers to commutativity-aware scheduling (Section 3.3.2) Aggregation represents executing the instruction aggregation step (Section 4.3) without CLS; CLS + aggregation is self-explanatory. For hand optimization scheme, to the best of our knowledge, there are limited optimization methods documented for architectures with iSWAP gates ([39, 48]). Here hand optimization refers to mechanically applying the known methods ([39, 48]) with our best effort.

Across all 9 benchmarks, our compilation scheme achieves a geometric mean of 5.07× pulse time reduction. CLS + hand optimization achieves a geometric mean of 2.338× pulse time reduction. The program characteristics of each benchmark heavily affect the level of optimization by logical scheduling and aggregation, but our compilation scheme achieves better circuit latency than gate-based compilation with hand optimization for every benchmark studied here.

Discussion

Commutativity vs Scheduling

In our study, the level of optimization from CLS scales with the commutativity of the circuit. In applications with little to no commutativity (like square-root, QFT and UCCSD), CLS has no effect as expected. In highly commutative circuits like MAXCUT circuits, CLS alone achieves up to 5× circuit length reduction.

CLS also facilitates instruction aggregation. As shown in the Ising model-n15 example in Figure 9, CLS alone has no optimization, but CLS + aggregation arrives at a better

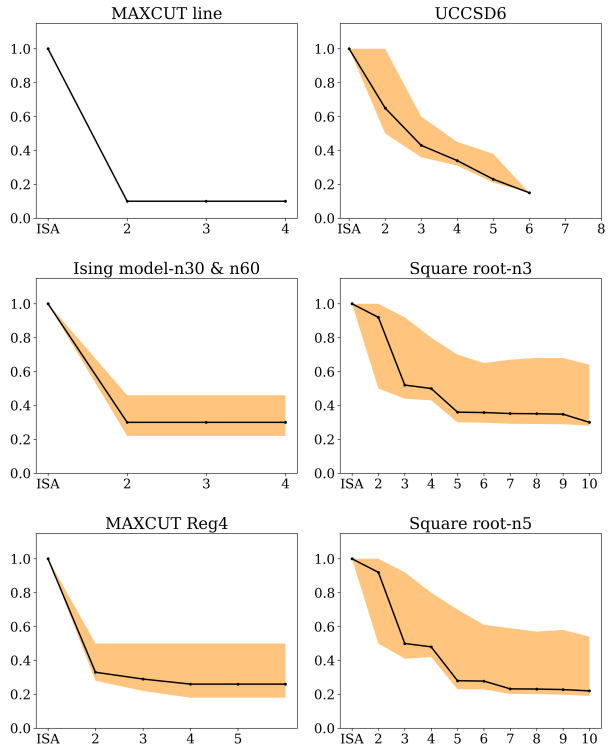


Fig. 10. Allowed instruction width vs normalized latency in selected benchmarks. The black line is the normalized latency of the entire circuit. The upper (lower) edge of the filled area is the normalized latency of the instruction on the critical path that has the least (most) pulse optimization. The three applications in the left column are parallelized, either originally or after CLS. The three applications in the right column are serialized. Increasing the allowed instruction width will benefit serialized applications more.

optimization of 3.44× circuit length reduction than 2.22× for aggregation alone.

Parallelism vs Instruction width

Figure 10 illustrates how circuit latency reduction scales with the allowed instruction width for several applications. For highly-parallel applications such as QAOA and the Ising model, the parallelism in the circuits places limits on the instruction width of aggregated instructions. Allowing a larger instruction width, therefore does not reduce latency. For serialized applications such as Square root and UCCSD, the latency reduction does not saturate until we reach the instruction width set by the scalability of the quantum optimal control.

In Figure 10, the lower bound of the yellow areas represents the largest latency reduction in an instruction on the critical path. In serialized applications, the total circuit latency reduction approaches this lower bound as instruction width increases. Thus, in these highly-serial applications, instructions with the largest latency reduction dominate the critical path, thus our

Spatial locality vs Aggregation

To show how spatial locality affects the pulse optimization in our scheme, we compare the three instances of QAOA application in our benchmarks: MAXCUT-line, MAXCUT-reg4, MAXCUT-cluster. After CLS, all of the three instances are highly paralleled and they have similar instruction sets that are composed of the CNOT-Rz-CNOT instruction and single qubit instructions. The main difference between the three instances is the spatial locality. The less spatially localized the instance is, the more SWAP gates must be inserted in the circuit.

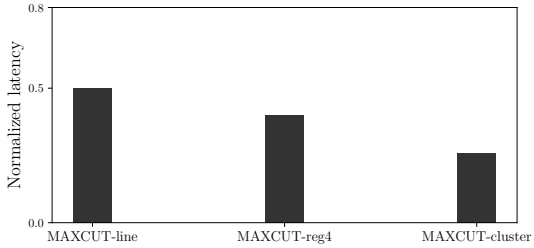


Fig. 11. The normalized latency of 3 instances of QAOA applications in aggregated instruction compilation scheme. For each of these instances, the latency after performing CLS is set to be 1 as baseline. From left to right, the 3 instances have high, medium, and low spatial locality.

Figure 11 shows that the MAXCUT-cluster instance has the lowest latency and MAXCUT-line has the highest latency comparing to the normalized latency after performing CLS. For the same application, aggregated instruction compilation has larger improvements on circuits with low spatial locality.

Information encoding scheme vs Pulse optimization

Information encoding schemes affect improvement due to pulse optimization. We evaluate the effect of the information encoding scheme on pulse optimization by comparing across spatially localized instances of our benchmark applications. QAOA applications encode the MAXCUT objective function directly onto the system Hamiltonian. In this simple encoding, inefficiency arises from the manual decomposition of diagonal unitaries generated by the objective Hamiltonian onto CNOT-Rz-CNOT instructions. In the spatially localized QAOA benchmark MAXCUT-line, hand optimization achieves about the same level of optimization as our compilation. UCCSD applications map molecular structures by performing the Jordan-Wigner transformation [29] and then decomposing the corresponding diagonal unitaries onto CNOT-Rz-CNOT chains. In this more complicated information encoding scheme, our tool realizes 3.12× greater circuit latency reduction than hand optimization in spatially localized instance UCCSD-n4. Our square root application involves reversible logic synthesis and quantum level decomposition, resulting in an encoding scheme that is more sophisticated than QAOA and UCCSD. In our Square root

application, our tool realizes 3.68× more circuit latency reduction than hand optimization.

From above observations, we see the trend that the more complicated the information encoding scheme is, the more advantageous our compilation is compared to hand optimization. This is expected, as simple hand optimization by replacing strategy is not efficient in finding the optimal path for complex quantum evolution with many degrees of freedom, especially when it involves sophisticated information encoding.

Related work

Standard gate-based compilation is a well studied subject [4, 10, 18, 23, 51]. Practical techniques have been developed to improve the standard gate-based compilation from the reversible logic level down to the technology level, including studies of hand optimization, discrete [34, 35] and continuous [38] template matching, and rule-based rewriting [37, 53]. Template matching methods achieved impressive gate reduction on small and intermediate-scale circuits, though they are limited by having to manually search for new template rules for each specific gate library (for example, there is no library for iSWAP gates). Rule-based rewriting methods suffer from the huge search space of rewriting strategies and apply mainly to reversible level decomposition.

Because the abstraction of logical level instruction remains intact in the frontend of our compilation, our workflow is compatible with most of the optimization methods described above at logical gate level. However, these upper level optimization efforts might be canceled in the backend, *e.g.*, if template matching takes place within an aggregated instruction, it will cause no effect because the output unitary is the same.

Recent work has moved beyond standard ISA abstraction. Chuang et al [15] design a new Hamiltonian simulation method that reduces the problem of quantum simulation to optimal quantum control of single qubit rotations [15]. Google proposes a plan to construct random circuits to demonstrate quantum supremacy at the pulse level [40].

The use of optimal control to compile large-scale quantum circuits was first explored by Schulte-Herbrueggen et al. [49] in their restricted recursive-style complex quantum instructions where they report speedups up to 300%. The researchers, however, did not provide an instruction aggregation algorithm.

In this work, we provide a systematic and universal way to reduce the circuit latency the overcomes the disadvantages of previous works.

Conclusion

In this paper, we present and analyze a new compilation methodology utilizing quantum optimal control theory. This compilation aggregates multi-qubit instructions and in this way breaks the ISA abstraction in the standard gate-based compilation scheme, resulting in a competitive pulse time reduction. Our implementation of this compilation methodology shows that in several important near-term quantum applications, our compilation process achieves up to 10X circuit latency reduction on superconducting architectures, which helps enable many appealing applications. We further analyze how different program characteristics, including parallelism, commutativity, and connectivity, interact with the level of optimization by instruction aggregation. We observe that our compilation scheme is most advantageous for quantum circuits that are highly serial, have low spatial locality, and utilize sophisticated information encoding.

Future work

There are several promising directions we propose for future study.

Compared to gate-based compilation, our scheme requires more computational resources and has a longer compilation time. For our benchmarks, the compilation time can be as long as several hours if the circuit has aggregated gates of 10 qubits. For classical-hybrid applications sensitive to long compilation time, future improvement of our compilation method is required. Partial compilation is a promising direction for solving this problem.

Our compilation method customizes aggregated instructions for each circuit, which leads to an increase in calibrations performed in experimental settings. The conflict between amount of calibration and circuit latency can potentially be resolved by incorporating realistic error modeling into our optimal control tool [55].

Another interesting area for future work is the theoretical study on optimization of circuits on superconducting architectures with the iSWAP gate. With our numerical study, we expect to see progress in the development of new techniques for circuit transformation and application level optimization targeted for these platforms.

We believe that finding a precise cost model for SWAP gates on superconducting architecture for better scheduling and mapping is an important problem.

Lastly, the instruction aggregation algorithm might be further improved by machine learning and tensor contraction techniques.

Appendix

Physical quantum gates

Below, we list some physical gates in different architectures:

- In platforms with Heisenberg interaction Hamiltonian, such as quantum dots [25], the directly directly implementable 2-qubit physical gate is the $\sqrt{\text{SWAP}}$ gate (which implements a SWAP when applied twice).
- In platforms with ZZ interaction Hamiltonian, such as superconducting systems of Josephson flux qubits [42, 43] and NMR quantum systems [54], the physical

gate is the CPhase gate, which is identical to the CNOT gate up to single qubit rotations.
- In platforms with XY interaction Hamiltonian, such as capacitively coupled Josephson charge qubits (*e.g* transmon qubits [28]), the 2-qubit physical gate is iSWAP gate.
- For trapped ion platforms with dipole-chain interaction, two popular physical 2-qubit gates are the geometric phase gate [31] and the XX gate [6].

Acknowledgment

The authors would like to thank Yao Lu, Ali Javadi Abhari, Ken Brown, James Leung, Isaac X. Shi for useful discussions. This work is funded in part by EPiQC, an NSF Expedition in Computing, under grant CCF-1730449. This work was also funded in part by NSF Phy-1818914 and a research gift from Intel. Additional funding for Henry Hoffmann comes from the DARPA BRASS program and a DoE Early Career Award. This work was completed in part with resources provided by the University of Chicago Research Computing Center.

References

[1] F. Bloch. Nuclear induction. *Phys. Rev.*, 70:460–474, Oct 1946.
[2] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven. Characterizing Quantum Supremacy in Near-Term Devices. *ArXiv e-prints*, July 2016.
[3] M. J. Chow. *Quantum Information Processing with Superconducting Qubits*. PhD thesis, New Haven, CT, USA, 2010. AAINQ98874.
[4] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta. Open Quantum Assembly Language. *ArXiv e-prints*, July 2017.
[5] P. de Fouquieres, S. G. Schirmer, S. J. Glaser, and I. Kuprov. Second order gradient ascent pulse engineering. *Journal of Magnetic Resonance*, 212:412–417, October 2011.
[6] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536:63 EP –, Aug 2016.
[7] M. H. Devoret and R. J. Schoelkopf. Superconducting circuits for quantum information: An outlook. *Science*, 339(6124):1169–1174, 2013.
[8] E. Farhi, J. Goldstone, and S. Gutmann. A Quantum Approximate Optimization Algorithm. *ArXiv e-prints*, November 2014.
[9] E. Farhi and A. W Harrow. Quantum Supremacy through the Quantum Approximate Optimization Algorithm. *ArXiv e-prints*, February 2016.
[10] X. Fu, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. An experimental microarchitecture for a superconducting quantum processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pages 813–825, New York, NY, USA, 2017. ACM.
[11] G. Giacomo Guerreschi and J. Park. Gate scheduling for quantum algorithms. *ArXiv e-prints*, July 2017.
[12] Steffen J. Glaser, Ugo Boscain, Tommaso Calarco, Christiane P. Koch, Walter Köckenberger, Ronnie Kosloff, Ilya Kuprov, Burkhard Luy, Sophie Schirmer, Thomas Schulte-Herbrüggen, Dominique Sugny, and Frank K. Wilhelm. Training schrödinger's cat: quantum optimal control. *The European Physical Journal D*, 69(12):279, Dec 2015.
[13] Pranav Gokhale. Github: graph-mapper. https://github.com/singular-value/graph-mapper, 2018.
[14] L. K. Grover. A fast quantum mechanical algorithm for database search. *eprint arXiv:quant-ph/9605043*, May 1996.
[15] G. Hao Low and I. L. Chuang. Optimal Hamiltonian Simulation by Quantum Signal Processing. *ArXiv e-prints*, June 2016.
[16] Reinier W. Heeres, Philip Reinhold, Nissim Ofek, Luigi Frunzio, Liang Jiang, Michel H. Devoret, and Robert J. Schoelkopf. Implementing a universal gate set on a logical qubit encoded in an oscillator. *Nature Communications*, 8(1):94, 2017.
[17] Ling Hu, Yuwei Ma, Weizhou Cai, Xianghao Mu, Yuan Xu, Weiting Wang, Yukai Wu, Haiyan Wang, Yipu Song, Changling Zou, S. M. Girvin, L-M. Duan, and Luyan Sun. Demonstration of quantum error correction and universal gate set on a binomial bosonic logical qubit. *arXiv e-prints*, page arXiv:1805.09072, May 2018.
[18] Thomas HÃďner, Damian S Steiger, Krysta Svore, and Matthias Troyer. A software methodology for compiling quantum programs. *Quantum Science and Technology*, 3(2):020501, 2018.
[19] Ali Javadi-Abhari, Pranav Gokhale, Adam Holmes, Diana Franklin, Kenneth R. Brown, Margaret Martonosi, and Frederic T. Chong. Optimized surface code communication in superconducting quantum computers. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pages 692–705, New York, NY, USA, 2017. ACM.
[20] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. Scaffcc: A framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, CF '14, pages 1:1–1:10, New York, NY, USA, 2014. ACM.
[21] J.R. Johansson, P.D. Nation, and Franco Nori. Qutip: An open-source python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 183(8):1760 – 1772, 2012.
[22] J.R. Johansson, P.D. Nation, and Franco Nori. Qutip 2: A python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 184(4):1234 – 1240, 2013.
[23] N. C. Jones, R. Van Meter, A. G. Fowler, P. L. McMahon, J. Kim, T. D. Ladd, and Y. Yamamoto. Layered Architecture for Quantum Computing. *Physical Review X*, 2(3):031007, July 2012.
[24] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549:242 EP –, Sep 2017.
[25] B. E. Kane. A silicon-based nuclear spin quantum computer. *Nature*, 393:133 EP –, May 1998. Article.
[26] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. In *SIAM Journal on Scientific Computing*, volume 20, 02 1970.
[27] Navin Khaneja, Timo Reiss, Cindie Kehlet, Thomas Schulte-HerbrÃijggen, and Steffen J. Glaser. Optimal control of coupled spin dynamics: design of nmr pulse sequences by gradient ascent algorithms. *Journal of Magnetic Resonance*, 172(2):296 – 305, 2005.
[28] J. Koch, T. M. Yu, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf. Introducing the Transmon: a new superconducting qubit from optimizing the Cooper Pair Box. *eprint arXiv:cond-mat/0703002*, February 2007.
[29] B. P. Lanyon, J. D. Whitfield, G. G. Gillett, M. E. Goggin, M. P. Almeida, I. Kassal, J. D. Biamonte, M. Mohseni, B. J. Powell, M. Barbieri, A. Aspuru-Guzik, and A. G. White. Towards quantum chemistry on a quantum computer. *Nature Chemistry*, 2:106–111, February 2010.
[30] Bjoern Lekitsch, Sebastian Weidt, Austin G. Fowler, Klaus Mølmer, Simon J. Devitt, Christof Wunderlich, and Winfried K. Hensinger. Blueprint for a microwave trapped ion quantum computer. *Science Advances*, 3(2), 2017.
[31] A. Lemmer, A. Bermudez, and M. B. Plenio. Driven geometric phase gates with trapped ions. *New Journal of Physics*, 15(8):083001, August 2013.
[32] Nelson Leung, Mohamed Abdelhafez, Jens Koch, and David Schuster. Speedup for quantum optimal control from automatic differentiation based on graphics processing units. *Physical Review A*, 95:042318, Apr 2017.

[33] Daniel Loss and David P DiVincenzo. Quantum computation with quantum dots. *Physical Review A*, 57(1):120, 1998.

[34] D. Maslov, G. W. Dueck, and D. M. Miller. Toffoli network synthesis with templates. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(6):807–817, June 2005.

[35] D. Maslov, G. W. Dueck, D. M. Miller, and C. Negrevergne. Quantum circuit simplification and level compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(3):436–444, March 2008.

[36] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and AlÃąn Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, 2016.

[37] D. M. Miller and Z. Sasanian. Lowering the quantum gate cost of reversible circuits. In *2010 53rd IEEE International Midwest Symposium on Circuits and Systems*, pages 260–263, Aug 2010.

[38] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov. Automated optimization of large quantum circuits with continuous parameters. *ArXiv e-prints*, October 2017.

[39] M. Neeley, R. C. Bialczak, M. Lenander, E. Lucero, M. Mariantoni, A. D. O'Connell, D. Sank, H. Wang, M. Weides, J. Wenner, Y. Yin, T. Yamamoto, A. N. Cleland, and J. M. Martinis. Generation of three-qubit entangled states using superconducting phase qubits. *Nature*, 467:570–573, September 2010.

[40] C. Neill, P. Roushan, K. Kechedzhi, S. Boixo, S. V. Isakov, V. Smelyanskiy, A. Megrant, B. Chiaro, A. Dunsworth, K. Arya, R. Barends, B. Burkett, Y. Chen, Z. Chen, A. Fowler, B. Foxen, M. Giustina, R. Graff, E. Jeffrey, T. Huang, J. Kelly, P. Klimov, E. Lucero, J. Mutus, M. Neeley, C. Quintana, D. Sank, A. Vainsencher, J. Wenner, T. C. White, H. Neven, and J. M. Martinis. A blueprint for demonstrating quantum supremacy with superconducting qubits. *Science*, 360:195–199, April 2018.

[41] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 10th edition, 2011.

[42] T. P. Orlando, J. E. Mooij, Lin Tian, Caspar H. van der Wal, L. S. Levitov, Seth Lloyd, and J. J. Mazo. Superconducting persistent-current qubit. *Phys. Rev. B*, 60:15398–15413, Dec 1999.

[43] H. Paik, A. Mezzacapo, M. Sandberg, D. T. McClure, B. Abdo, A. D. Córcoles, O. Dial, D. F. Bogorin, B. L. T. Plourde, M. Steffen, A. W. Cross, J. M. Gambetta, and J. M. Chow. Experimental Demonstration of a Resonator-Induced Phase Gate in a Multiqubit Circuit-QED System. *Physical Review Letters*, 117(25):250502, December 2016.

[44] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5:4213 EP –, Jul 2014. Article.

[45] J. Preskill. Quantum Computing in the NISQ era and beyond. *ArXiv e-prints*, January 2018.

[46] IBM Qiskit. Github: qiskit-backend-information. https://github.com/Qiskit/qiskit-backend-information, July 2018.

[47] J. Romero, R. Babbush, J. R. McClean, C. Hempel, P. Love, and A. Aspuru-Guzik. Strategies for quantum computing molecular energies using the unitary coupled cluster ansatz. *ArXiv e-prints*, January 2017.

[48] N. Schuch and J. Siewert. Natural two-qubit gate for quantum computation using the XY interaction. *Physical Review A*, 67(3):032301, March 2003.

[49] T. Schulte-Herbrueggen, A. Spoerl, and S. J. Glaser. Quantum CISC Compilation by Optimal Control and Scalable Assembly of Complex Instruction Sets beyond Two-Qubit Gates. *ArXiv e-prints*, December 2007.

[50] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *eprint arXiv:quant-ph/9508027*, August 1995.

[51] R. S. Smith, M. J. Curtis, and W. J. Zeng. A Practical Quantum Instruction Set Architecture. *ArXiv e-prints*, August 2016.

[52] Robert S Smith, Michael J Curtis, and William J Zeng. A practical quantum instruction set architecture, 2016.

[53] Mathias Soeken and Michael Kirkedal Thomsen. White dots do matter: Rewriting reversible logic circuits. In Gerhard W. Dueck and D. Michael Miller, editors, *Reversible Computation*, pages 196–208, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[54] L. M. K. Vandersypen and I. L. Chuang. NMR techniques for quantum control and computation. *Reviews of Modern Physics*, 76:1037–1069, October 2004.

[55] M. Yuezhen Niu, S. Boixo, V. Smelyanskiy, and H. Neven. Universal Quantum Control through Deep Reinforcement Learning. *ArXiv e-prints*, March 2018.