Partial Compilation of Variational Algorithms for Noisy Intermediate-Scale Quantum Machines

Pranav Gokhale*
University of Chicago

Yongshan Ding
University of Chicago

Thomas Propson
University of Chicago

Christopher Winkler
University of Chicago

Nelson Leung
University of Chicago

Yunong Shi
University of Chicago

David I. Schuster
University of Chicago

Henry Hoffmann
University of Chicago

Frederic T. Chong
University of Chicago

ABSTRACT

Quantum computing is on the cusp of reality with Noisy Intermediate-Scale Quantum (NISQ) machines currently under development and testing. Some of the most promising algorithms for these machines are *variational* algorithms that employ classical optimization coupled with quantum hardware to evaluate the quality of each candidate solution. Recent work used GRadient Descent Pulse Engineering (GRAPE) to translate quantum programs into highly optimized machine control pulses, resulting in a significant reduction in the execution time of programs. This is critical, as quantum machines can barely support the execution of short programs before failing.

However, GRAPE suffers from high compilation latency, which is untenable in variational algorithms since compilation is interleaved with computation. We propose two strategies for *partial compilation*, exploiting the structure of variational circuits to pre-compile optimal pulses for specific blocks of gates. Our results indicate significant pulse speedups ranging from 1.5x-3x in typical benchmarks, with only a small fraction of the compilation latency of GRAPE.

CCS CONCEPTS

• **Computer systems organization → Quantum computing**.

KEYWORDS

quantum computing, optimal control, variational algorithms

ACM Reference Format:
Pranav Gokhale, Yongshan Ding, Thomas Propson, Christopher Winkler, Nelson Leung, Yunong Shi, David I. Schuster, Henry Hoffmann, and Frederic T. Chong. 2019. Partial Compilation of Variational Algorithms for Noisy Intermediate-Scale Quantum Machines. In *The 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-52), October 12–16, 2019, Columbus, OH, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3352460.3358313

*Corresponding author: pranavgokhale@uchicago.edu

MICRO-52, October 12–16, 2019, Columbus, OH, USA

ACM ISBN 978-1-4503-6938-1/19/10…$15.00
https://doi.org/10.1145/3352460.3358313

1 INTRODUCTION

In the Noisy Intermediate-Scale Quantum (NISQ) era, we expect to operate hardware with hundreds or thousands of quantum bits (qubits), acted on by imperfect gates [42]. Moreover, connectivity in these NISQ machines will be sparse and qubits will have modest lifetimes. Given these limitations, NISQ era machines will not be able to execute large-scale quantum algorithms like Shor Factoring [45] and Grover Search [20], which rely on error correction that requires millions of qubits [38, 48].

However, recently, *variational algorithms* have been introduced that are well matched to NISQ machines. This new class of algorithms has a wide range of applications such as molecular ground state estimation [41], MAXCUT approximation [14], and prime factorization [2]. The two defining features of a variational algorithm are that:

(1) the algorithm complies with the constraints of NISQ hardware. Thus, the circuit for a variational algorithm should have modest requirements in qubit count (circuit width) and runtime (circuit depth / critical path).
(2) the quantum circuit for the algorithm is parametrized by a list of angles. These parameters are optimized by a classical optimizer over the course of many iterations. For this reason, variational algorithms are also termed as hybrid quantum-classical algorithms [42]. Typically, a classical optimizer that is robust to small amounts of noise (e.g. Nelder-Mead) is chosen [32, 41].

Standard non-variational quantum algorithms are fully specified at compile time and therefore can be fully optimized by static compilation tools as in previous work [23, 26]. By contrast, each iteration of a variational algorithm depends on the results of the previous iteration–hence, compilation must be interleaved through the computation. As even small instances of variational algorithms will require thousands of iterations [24], the compilation latency for each iteration therefore becomes a serious limitation. This feature of variational algorithms is a significant departure from previous non-variational quantum algorithms.

To cope with this limitation on compilation latency, past work on variational algorithms has performed compilation under the standard gate-based model. This methodology has the advantage of extremely fast compilation–a lookup table maps each gate to a sequence of machine-level control pulses so that compilation simply amounts to concatenating the pulses corresponding to each

gate. We note that this compilation procedure is a conservative picture of experimental approaches to gate-based compilation. In practice, parametrized gates may be handled by a step-function lookup table that depends on the run-time parameters, with the aim of reducing errors, as demonstrated in [4, 34, 36].

The gate-based compilation model is known to fall short of the GRadient Ascent Pulse Engineering (GRAPE) [17, 25] compilation technique, which compiles directly to the level of the machine-level control pulses that a quantum computer actually executes. In past work [1, 27, 44], GRAPE has been used to achieve 2-5x pulse speeedups over gate-based compilation for a range of quantum algorithms. Since fidelity decreases exponentially in time, with respect to the extremely short lifetimes of qubits, reducing the pulse duration is critical to ensuring that a computation completes before being completely scrambled by quantum decoherence effects. Thus, 2-5x pulse speedups translate to an even bigger advantage in the success probability of a quantum circuit.

However, GRAPE-based compilation has a substantial cost: compilation time. Running GRAPE control on a circuit with just four qubits takes several minutes. For representative four qubit circuits, we observed compile times ranging from 10 minutes to 1 hour, even with state-of-the-art hardware and GPU acceleration. This would amount to several weeks or months of total compilation latency over the course of thousands of iterations (and millions of iterations will be needed for larger problems). By contrast, typical pulse times for quantum circuits are on the order of microseconds, so the compilation latency imposed by GRAPE is untenable. Thus, GRAPE-based compilation is not practical out-of-the-box for variational algorithms.

In this paper, we introduce *partial compilation*, a strategy that approaches the pulse duration speedup of GRAPE, but with a manageable overhead in compilation latency. With this powerful new compiler capability, **we enable the architectural choice of pulse-level instructions**, which supports more complex near-term applications through lower latencies and thus much lower error rates. This architectural choice would be infeasible without our compiler support. Our specific contributions include:

- Demonstration of the advantage of GRAPE over gate based compilation for variational algorithms
- Strict partial compilation, a strategy that pre-computes optimal pulses for parametrization-independent blocks of gates. This strategy is strictly better than gate-based compilation– it achieves a significant pulse speedup (approaching GRAPE results), with no overhead in compilation latency.
- Flexible partial compilation, a strategy that performs as well as full GRAPE, but with a dramatic speedup in compilation latency via precomputed hyperparameter optimization.

The rest of this paper is organized as follows. Section 2 gives prerequisite background on quantum computation and Section 3 describes related work from prior research. Section 4 describes characteristics of our benchmark variational algorithms, with particular attention to the structural properties that our compilation strategies exploit. Section 5 explains the GRAPE compilation methodology. Sections 6 and 7 explain our partial compilation strategies and Section 8 discusses our results. We conclude in Section 9 and propose future work in Section 10. Appendix A presents the system Hamiltonian that we consider in GRAPE.

2 BACKGROUND ON QUANTUM COMPUTATION

2.1 Qubits

The fundamental unit of quantum computation is a quantum bit, or qubit. A qubit has two basis states, which are represented by *state vectors* denoted

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix} \text{ and } |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}.$$

Unlike a classical bit, the state of a qubit can be in a *superposition* of both $|0\rangle$ and $|1\rangle$. In particular, the space of valid qubit states are $\alpha|0\rangle + \beta|1\rangle$, normalized such that $|\alpha|^2 + |\beta|^2 = 1$. When a qubit is measured, its quantum state *collapses* and either $|0\rangle$ or $|1\rangle$ is measured, with probabilities $|\alpha|^2$ and $|\beta|^2$ respectively.

A two-qubit system has four basis states:

$$|00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \ |01\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, |10\rangle = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \text{and } |11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

and any two-qubit state can be expressed as the superposition $\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$ (normalized so that $|\alpha| + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$). More generally, an N-qubit system has 2^N basis states. Therefore, 2^N numbers, called amplitudes, are needed to describe the state of a general N-qubit system. This exponential scaling gives rise to both the difficulty of classically simulating a quantum system, as well as the potential for quantum computers to exponentially outperform classical computers in certain applications.

2.2 Quantum Gates

A quantum algorithm is described in terms of a quantum circuit, which is a sequence of 1- and 2- input quantum gates. Every quantum gate is represented by a square matrix, and the action of a gate is to left-multiply a state vector by the gate's matrix. Because quantum states are normalized by measurement probabilities, these matrices must preserve l^2-norms. This corresponding set of matrices are *unitary* (orthogonal) matrices. The unitary matrices for two important single-qubit gates are:

$$R_x(\theta) = \begin{pmatrix} i\cos\frac{\theta}{2} & \sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & i\cos\frac{\theta}{2}\end{pmatrix} \text{ and } R_z(\phi) = \begin{pmatrix}1 & 0\\ 0 & e^{i\phi}\end{pmatrix}$$

At $\theta = \pi$, the $R_x(\pi)$ gate has matrix $\left(\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right)$, which acts as a NOT gate: left-multiplying by it swaps between the $|0\rangle$ and $|1\rangle$ states. This bit-flip gate is termed the X gate.

Similarly, at $\phi = \pi$, the $R_z(\pi)$ gate has matrix $\left(\begin{smallmatrix}1&0\\0&-1\end{smallmatrix}\right)$, which applies a -1 multiplier to the amplitude of $|1\rangle$; this type of gate is unique to the quantum setting, where amplitudes can be negative (or complex). This 'phase'-flip gate is termed the Z gate.

Gate	$R_z(\phi)$	$R_x(\theta)$	H	CX	SWAP
Time (ns)	0.4	2.5	1.4	3.8	7.4

Table 1: Library of the compiler's gate set and corresponding pulse durations (in nanoseconds) for each gate. The runtimes of circuits under gate-based compilation are indexed to these pulse durations.

An important 2-input quantum gate is

$$\text{CX} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

The CX gate, often referred to as the CNOT or Controlled-NOT gate, applies an action that is controlled on the first input. If the first input is $|0\rangle$, then the CX gate has no effect. If the first input is $|1\rangle$, then it applies an $X = R_x(\pi)$ to the second qubit.

The CX gate is an *entangling gate*, meaning that its effect cannot be decomposed into independent gates acting separately on the two qubits. An important result in quantum computation states that the set of all one qubit gates, plus a single entangling gate, is sufficient for universality [37]. Since the $R_x(\theta)$ and $R_z(\phi)$ gates span the set of all one qubit gates, we see that, $\{R_x(\theta), R_z(\phi), CX\}$ is a universal gate set.

In practice, we seek to implement a quantum algorithm using the most efficient quantum circuit possible, with efficiency defined in terms of circuit width (number of qubits) and depth (length of critical path, or runtime of the circuit). Accordingly, quantum circuits are optimized by repeatedly applying gate identities that reduce the resources consumed by the circuit. All circuits that are presented in this paper were optimized using IBM Qiskit's Transpiler, which applies a variety of circuit identities–for example, aggressive cancellation of CX gates and 'Hadamard' gates. We also augmented the IBM optimizer with our own compiler pass for merging rotation gates–e.g. $R_x(\alpha)$ followed by $R_x(\beta)$ merges into $R_x(\alpha + \beta)$–which we found to further reduce circuit sizes.

2.3 Gate-Based Compilation

At the lowest level of hardware, quantum computers are controlled by analog pulses. Therefore, quantum compilation must translate from a high level quantum algorithm down to a sequence of control pulses. Once a quantum algorithm has been decomposed into a quantum circuit comprising single- and two- qubit gates, gate-based compilation simply proceeds by concatenating a sequence of pulses corresponding to each gate. In particular, a lookup table maps from each gate in the gate set to a sequence of control pulses that executes that gate. Table 1 indicates the total pulse duration for each gate in the compilation basis gate set. These pulse durations are based on the gmon-qubit [7] quantum system described in Appendix A.

As previously noted, the $\{R_x(\theta), R_z(\phi), \text{CX}\}$ gate set alone is sufficient for universality, so in principle the H and SWAP gates could be removed from the compilation basis gate set. However, we include the generated pulses (using GRAPE as described below) for these gates in our compilation set, because quantum assembly languages typically include them in their basis set [19, 22, 33, 46, 47, 50].

The advantage of the gate-based approach is its short pulse compilation time, as the lookup and concatenation of pulses can be accomplished almost instantaneously. However, it prevents the optimization of pulses from happening across the gates, because there might exists a global pulse for the entire circuit that is shorter and more accurate than the concatenated one. The quality of the concatenated pulse relies heavily on an efficient gate decomposition of the quantum algorithm.

2.4 GRAPE

GRadient Pulse Engineering (GRAPE) is a strategy for compilation that numerically finds the best control pulses needed to execute a quantum circuit or subcircuit by following a gradient descent procedure [10, 25]. We use the Tensorflow implementation of GRAPE described in [27]. In contrast to the gate based approach, GRAPE does not have the limitation incurred by the gate decomposition. Instead, it directly searches for the optimal control pulse for the input circuit as a whole. Our full GRAPE procedure is described further detail in Section 5.

3 RELATED WORK

Past publications of variational algorithm implementations have relied on gate-based compilation, using parametrized gates such as $R_x(\theta)$ and $R_z(\phi)$. Existing quantum languages offer support for such parametrized gates [19, 22, 33, 46, 47, 53]. In most languages, the angles must be declared at compile time–thus at every iteration of a variational algorithm, a new circuit is compiled based on the new parametrization. Rigetti's Quil [46] language goes a step further by supporting runtime resolution of the parameters in parameters gates, which allows dynamic implementations of variational algorithms. However, as acknowledged in the Quil specifications, this approach hampers circuit optimization, because the actual parameters are not known until runtime.

While this paper treats gate-based compilation as a simple lookup table between gates and pulses, experimental implementations have already moved directionally towards GRAPE-style, because pulse sequences can depend on the input angles in a complicated fashion. For example, in [4], a parametrized $U(\phi)$ gate has five different pulse sequence decompositions, each corresponding to ϕ in ranges set by the breakpoints $[-\pi, 2.25, -0.25, 0.25, 2.25, \pi]$. [36] and [34] have similar step-function gate-to-pulse translation.

The growing overhead of compilation latency has been recognized, and recent work has proposed the development of specialized FPGAs for the compilation of variational algorithms [35]. More broadly, we note that pulse level control is at the cusp of industry adoption. An open specification for pulse-level control, OpenPulse, was standardized recently [33], and IBM plans to introduce an API for pulse level control in 2019 [21]. Pulse access to quantum machines will open the door to experimental realizations of GRAPE, including for variational algorithms as proposed in this paper.

4 VARIATIONAL BENCHMARKS

Variational quantum algorithms are important in the near-term because they comply with the constraints of NISQ hardware. In

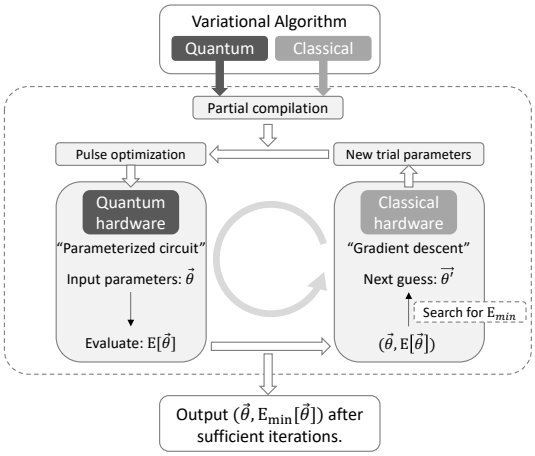


Figure 1: Illustration of a variational quantum algorithm that alternates between a quantum circuit and a classical optimizer. In this process, the quantum circuit (parameterized by $\vec{\theta}$) evaluates some cost function $E[\vec{\theta}]$, and the classical optimizer gradient descends for the next set of parameters.

Molecule	Width (# of Qubits)	# of Params	Gate-Based Runtime
H_2	2	3	35 ns
LiH	4	8	872 ns
BeH_2	6	26	5308 ns
NaH	8	24	5490 ns
H_2O	10	92	33842 ns

Table 2: Benchmarked circuits for VQE, using the UCCSD ansatz. Each circuit was optimized, parallel-scheduled, mapped using IBM Qiskit's tools, augmented by an additional optimization pass we wrote to merge consecutive rotation gates. The Gate-Based Runtime is indexed to the pulse durations for each gate reported in Table 1.

particular, variational algorithms have innate error resilience, due to the hybrid alternation with a noise-robust classical optimizer [32, 41]. Every iteration of a variational algorithm is parameterized by a list of angles. In general, the parameter space explored by a variational algorithm is not known a priori–the classical optimizer picks the next iteration's parameters based on the results of the previous iterations. Consequently, the compilation for each iteration is interleaved with the actual computation. A schematic of this process is illustrated in Figure 1.

There are two variational quantum algorithms: Variational Quantum Eigensolver and Quantum Approximate Optimization Algorithm. We discuss both below.

4.1 Variational Quantum Eigensolver

The Variational Quantum Eigensolver (VQE) is used to find the ground state energy of a molecule, a task that is exponentially difficult in general for a classical computer, but is believed to be efficiently solvable by a quantum computer [31]. Estimating the molecular ground state has important applications to chemistry such as determining reaction rates [13] and molecular geometry [40]. A conventional quantum algorithm for solving this problem is called the Quantum Phase Estimation (QPE) algorithm [28]. However, for a target precision ϵ, QPE requires a quantum circuit with depth $O(1/\epsilon)$, whereas VQE algorithm requires $O(1/\epsilon^2)$ iterations of depth-$O(1)$ circuits[52]. The latter assumes a much more relaxed fidelity requirement on the qubits and gate operations, because the higher the circuit depth, the more likely the circuit experiences an error at the end. At a high level, VQE can be conceptualized as a guess-check-repeat algorithm. The check stage involves the preparation of a quantum state corresponding to the guess. This preparation stage is done in polynomial time on a quantum computer, but would incur exponential cost (owing to the 2^N state vector scaling) in general on a classical computer. This contrast gives rise to a potential quantum speedup for VQE.

The quantum circuit corresponding to the guess is termed an ansatz. While many ansatz choices are possible, Unitary Coupled Cluster Single-Double (UCCSD), an ansatz motivated by principles of quantum chemistry, is considered the gold standard [6, 43]. The UCCSD ansatz is also promising because it could circumvent the barren plateaus issue that affects many other ansatzes [31].

We benchmark the UCCSD ansatz for five molecules: H_2, LiH, BeH_2, NaH, H_2O. These molecules span the state of the art for experimental implementations of VQE: H_2O is the largest molecule addressed by VQE [36] to date. We generated our UCCSD ansatz circuits using the IBM Qiskit implementation described in [5] as well as the PySCF Python package [49] to manage molecular data.

Both the the circuit depth and number of ansatz parameters in UCCSD scale as $O(N^4)$ in the circuit width [3]. Table 2 specifies the exact circuit width, number of variational parameters, and gate-based runtime (circuit depth) for each of the benchmarks. The reported gate-based runtimes are indexed to the pulse durations of each gate reported in Table 1. Each circuit was optimized using IBM Qiskit's circuit optimizer pass system, Qiskit's circuit mapper (to conform to nearest neighbor connectivity), and a custom compiler pass to merge neighboring rotation gates on the same axis. We also exploit parallelism to simultaneously schedule as many gates as posisble; the reported gate-based runtimes are for the critical path through the parallelized circuit. These circuit optimizations form a fair baseline for the best circuit runtimes achievable by gate based compilation. Our full circuit optimization code, along with the results of optimization applied to our benchmarks, is available on our Github repository [18].

4.2 QAOA

Quantum Approximate Optimization Algorithm (QAOA) is an algorithm for generating approximate solutions to problems that are hard to solve exactly. At an intuitive level, QAOA can be understood as an alternating pattern of Mixing and Cost-Optimization steps. At each Mixing step, QAOA applies diffusion so that every possible state is explored in quantum superposition. At each Cost-Optimization step, a bias is applied to boost the magnitudes of quantum states that minimize a cost function. Thereafter, measuring

can yield an approximate solution close to optimal with high probability. The number of alternating Mixing and Cost-Optimization rounds is known as p. Even for small p, QAOA has competitive results against classical approximation algorithms. For example, at $p = 1$, QAOA applied to the NP-hard MAXCUT problem yields a cut of size at least 69% of the optimal cut size [14]. At $p = 5$, simulations have demonstrated that QAOA achieves mean parity with the best-known classical algorithm, Goemans-Williamson, for 10 node graphs [9]. For larger p, QAOA is expected to outperform classical approximation algorithms even for worst-case bounds, although theoretical guarantees have not been established yet. QAOA is of particular interest in the near term because recent work has shown that it is computationally universal [29]. Moreover, QAOA has shown experimental resilience to noise [39]. For these reasons, QAOA is a leading candidate for quantum supremacy [15], the solution of a classically-infeasible problem using a quantum computer.

Similarly to VQE, QAOA is a guess-check-repeat algorithm. In the case of QAOA, the guesses correspond to "Mixing magnitude during iteration $1 \leq i \leq p$" and "Cost-Optimization magnitude during iteration $1 \leq i \leq p$". Hence, the number of parameters in a QAOA circuit is $2p$: one scalar for Mixing magnitude and one for Cost-Optimization magnitude, for each of the p rounds.

We benchmark QAOA for $N = 6$ and 8 node graphs, with the number of QAOA rounds p spanning from 1 to 8. For each (N, p) pair, we benchmark for two types of random graphs: 3-regular (each node is connected to three neighbors) and Erdos-Renyi (each possible edge is included with 50% probability). This yields $2 \times 8 \times 2 = 32$ benchmarks circuits for QAOA. The gate-based runtimes for each of these benchmarks are reported in Table 3. As with the VQE benchmarks, the runtimes are computed after circuit mapping and optimizations, to form a fair baseline.

	$N = 6$		$N = 8$	
	3-Regular	Erdos-Renyi	3-Regular	Erdos-Renyi
$p = 1$	113 ns	84 ns	163 ns	157 ns
$p = 2$	199 ns	151 ns	365 ns	297 ns
$p = 3$	277 ns	223 ns	530 ns	443 ns
$p = 4$	356 ns	296 ns	695 ns	596 ns
$p = 5$	434 ns	368 ns	860 ns	750 ns
$p = 6$	512 ns	440 ns	1025 ns	903 ns
$p = 7$	590 ns	512 ns	1191 ns	1056 ns
$p = 8$	668 ns	584 ns	1356 ns	1209 ns

Table 3: Gate-based runtimes for our 32 benchmark QAOA MAXCUT circuits. Our benchmarks consider two types of random graphs: 3-Regular and Erdos-Renyi. We consider both 6 and 8 node graphs–the number of qubits in the circuit is the same as the number of nodes in the graph. We benchmarked over p, the number of repetitions of the basic QAOA block, ranging from 1 to 8, which represents a range of p that is of both theoretical and practical interest [9]. As in Table 2, the gate-based runtimes are based on the gate times in Table 1, after each circuit has been optimized, parallel-scheduled, and mapped.

5 GRAPE COMPILATION

In this section, we describe GRAPE (GRadient Ascent Pulse Engineering), a compilation technique that aims to produce the optimal possible sequence of analog control pulses needed to realize the unitary matrix transformation for a targeted quantum circuit. At an abstract level, GRAPE simply treats the underlying quantum computer as a black box. The black box accepts time-discretized control pulses as input and outputs the unitary matrix of the transformation that is realized by the input control pulses. GRAPE performs gradient descent over the space of possible control pulses to search for the optimal sequence of input signals that achieve the targeted unitary matrix up to a specified fidelity. We used the Tensorflow-based implementation of GRAPE described in [27], which has demonstrated good performance. The gradients are computed analytically and backpropogated with automatic differentiation.

In this paper, we define the optimal sequence of control pulse as the one of shortest duration–thus, we seek to speed up the pulse time with respect to gate-based compilation. Reducing the pulse time is important in quantum computation because qubits have short lifetimes due to quantum decoherence effects. The decoherence error increases exponential with time, so the effect of a pulse time speedup enters the power of an exponential term. We focus on this error metric because it is one of the dominant error terms for superconducting qubits and it is well understood. However, in principle, GRAPE can be used to control other sources of error such as gate errors, State Preparation and Measurement (SPAM) errors, and qubit crosstalk, as demonstrated in past work [1, 8, 12].

5.1 Speedup Sources

Because GRAPE translates directly from a unitary matrix to hardware-level control pulses–without the overhead of an intermediate set of quantum gates–it achieves more optimized control pulses than gate-based compilation does. In particular, we observed significant pulse speedups from GRAPE due to the following factors:

- **ISA alignment**. Gate based compilation incurs a significant overhead because the set of basis gates will not be *directly* implementable on a target machine. For example, while quantum circuits are typically compiled down to CX (CNOT) gates as the default two-qubit instructions, actual quantum computers implement a wide range of native two-qubit operations such as the MS gate or the iSWAP gate. Compiling gates to pulses incurs a significant overhead from this ISA misalignment.
- **Fractional gates**. A unique feature of quantum computing is that all operations can be fractionally performed–for example, $CX^{1/2}$ is a valid quantum gate, as is CX^p more generally for any power. Often, a fractional application of a basis gate is sufficient to execute a larger quantum operation. The fixed basis set of gate based compilation misses these optimizations, whereas GRAPE works in a continuous basis and realizes fractional gates when beneficial.
- **Control Field Asymmetries**. While gate based compilation puts R_x and R_z gates on an equal footing, at a physical level, there is often a significant asymmetry between the

speed and reliability of these operations. As described in A, we model a representative quantum system in which Z-axis qubit rotations are 15 times faster than X-axis qubit rotations. GRAPE's search for the shortest pulse realization will therefore leverage this asymmetry, preferring Z rotations when possible. For example, the H gate is typically implemented by the $R_x(\frac{-\pi}{2})R_z(\frac{-\pi}{2})R_x(\frac{-\pi}{2})$ pulse sequence, which involves two X-axis rotations and one Z-axis rotation. We observe that our GRAPE system instead discovers the equivalent $R_z(\frac{-\pi}{2})R_x(\frac{-\pi}{2})R_z(\frac{-\pi}{2})$ pulse sequence, which only requires one X-axis rotation and therefore executes significantly faster.

- **Maximal circuit optimization**. Although quantum circuits can be optimized at the gate-level by repeatedly applying a set of circuit identity templates, the set of templates must be finite. Opportunities for optimization between distant gates (both in width and depth) may be overlooked. By contrast, GRAPE subsumes all circuit optimizations by working directly in terms of the unitary matrix of the circuit, as opposed to the gate decomposition.

5.2 Circuit Blocking for GRAPE

While GRAPE can achieve significant pulse speedups, it is limited by two factors:

- The unitary matrix of the targeted quantum circuit must be specified as input to the GRAPE program. An N-qubit circuit has a $2^N \times 2^N$ matrix (due to the exponential state space of an N-qubit space), which imposes a bound on the maximum circuit size that GRAPE can handle.
- The total convergence time for GRAPE's gradient descent scales exponentially in the size of the target quantum circuit [27]. For example, it typically takes our GRAPE implementation several minutes to find the pulses for a 4 qubit QAOA MAXCUT circuit. Experientially, we also found difficulty consistently finding convergence for deep quantum circuits with $N > 5$ qubits.

For this reason, it is necessary to partition large quantum circuits into blocks of manageable width. We blocked into subcircuits of up to 4 qubits, using the aggregation methodology discussed in [44]. Specifically, we select maximal subcircuits of 4 qubit width, such that partitioning the subcircuit does not delay the execution of subcircuits. This methodology ensures that full GRAPE is strictly better than gate based compilation–otherwise, subcircuits may induce serialization that underperforms gate based compilation. Details are discussed in Section 4.3 of [44].

5.3 Binary Search for Minimum Pulse Time

In prior work [27, 44], the pulse length is specified as a static 'upper bound' parameter, `total_time`. Pulse speedups are then performed by adding a term to the cost function that rewards pulses that realize the targeted unitary matrix in time shorter than `total_time`. However, to comply with the automatic differentiation methodology for analytically computing gradients, this cost function term is continuous and rewards *gradual* progress of the pulse towards the target unitary matrix. By contrast, our ultimate goal is to find the *binary* cutoff point specifying the minimal possible time needed to achieve a pulse. Moreover, setting the relative weighting of the speedup term to the fidelity term in the cost function is difficult. Poor choices of weights can either prevent GRAPE from achieving any speedup or realizing the target fidelity.

As proposed by the prior work [27], our methodology adaptively changes the `total_time` by binary searching for the shortest `total_time` needed to achieve a target unitary matrix. While this incurs the overhead of running on the more iterations [1], it is worthwhile because minimizing the pulse time is exponentially critical in terms of reducing errors.

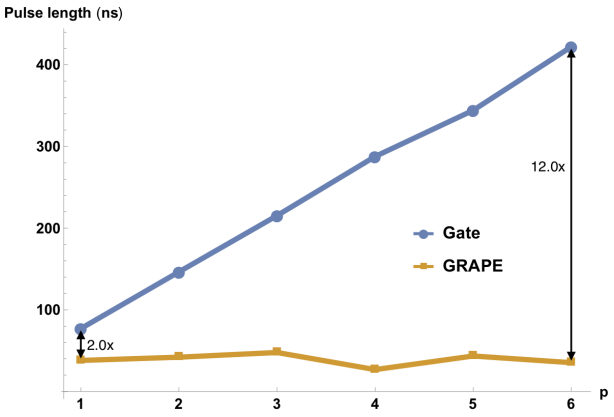


Figure 2: Pulse lengths from gate based compilation and full GRAPE for MAXCUT on the 4-node clique. While the gate based pulse times are simply linear in the number of QAOA rounds p, the GRAPE based times asymptote to an upper bound. For each p, a random parametrization was set. The ratio varies from 2.0x at $p = 1$ to 12.0x at $p = 6$.

5.4 GRAPE Compilation for QAOA

There are a range of theoretical results setting upper bounds on the circuit complexity needed to achieve a particular quantum operation. For example, it is known that 3 CX gates, sandwiched by single-qubit rotations, is sufficient to implement any two qubit operation. These results were recently generalized to the context of quantum optimal control (a generalization of GRAPE) with a proof that any N-qubit operation can be achieved in $O(4^N)$ time via optimal control [30].

This implies that GRAPE can achieve a significant advantage over gate-based compilation in algorithms like QAOA that have p repeated blocks. While the pulse length from gate-based compilation scales linearly in the p, the GRAPE based pulse length is upper bounded by the maximum time it takes to implement any transformation for an N-qubit circuit. Figure 2 demonstrates this behavior for QAOA MAXCUT on the 4-node clique problem. While the pulse length from gate based compilation scales linearly in the number of QAOA rounds p, it asymptotes below 50 ns for GRAPE based pulse lengths. Thus, the pulse speedup advantage of GRAPE increases with p.

[1] Specifically, on the order of $log(M/\Delta t)$ iterations where M is the upper bound on `total_time` and Δt is the desired precision, which we set to 0.3 ns.

As our QAOA benchmarks have circuit widths of 6 and 8 qubits–larger than the 4 qubit blocks we feed to GRAPE–the number of serial blocks will scale linearly with p. Therefore, we don't expect to see an unboundedly growing speedup of GRAPE with increasing p, but we still expect to see gains within each 4 qubit block.

6 STRICT PARTIAL COMPILATION

While full quantum optimal control generates the fastest possible pulse sequence for a target circuit, its compilation latency on the order of several minutes is untenable for variational algorithms, in which compilation is interleaved with computation. In order to approach the pulse speedup of GRAPE without incurring the full cost in compilation latency, it is necessary to exploit the structure of the variational circuits. We term this structural analysis as *partial compilation*, and it is executed as pre-computation step prior to executing the variational algorithm on a quantum computer.

Our first strategy, Strict Partial Compilation, stems from the observation that for typical circuits in variational algorithms, most of the gates are independent of the parametrization. For example, Figure 3a shows an example variational circuit. While the circuit has many gates, only four of them depend on the variational θ_i parameters. All of the other gates can be blocked into maximal parametrization-independent subcircuits. Figure 3b demonstrates the application of strict partial compilation to the variational circuit from Figure 3a. The sequence of resulting subcircuits is [Fixed, $R_z(\theta_1)$, Fixed, $R_z(\theta_1)$, Fixed, $R_z(\theta_2)$, Fixed, $R_z(\theta_3)$], which exhibits *strict* alternation between 'Fixed' subcircuits that don't depend on any θ_i and $R_z(\theta_i)$ gates that do depend on the parametrization.

After the strict partial compilation blocking is performed, we use full GRAPE to pre-compute the shortest pulse sequence needed to execute each Fixed subcircuit. These static precompiled pulse sequences can be defined as microinstructions in a low-level assembly such as eQASM [16]. Thereafter, at runtime, the pulse sequence for any parametrization can be generated by simply concatenating the pre-computed pulse sequences for Fixed blocks with the control pulses for each parametrization-dependent $R_z(\theta_i)$ gate. Thus, strict partial compilation retains the extremely fast (essentially instant) compilation time of standard gate based compilation. However, since each Fixed block was compiled by GRAPE, the resulting pulse duration is shorter than if the Fixed blocks had been compiled by gate based compilation. Thus, strict partial compilation achieves pulse speedups over gate-based compilation, with no increase in compilation latency.

Full discussion of the results is deferred to Section 8. A priori, we note that the performance of strict partial compilation is tied to the depth of the Fixed subcircuits. For deeper Fixed subcircuits, GRAPE has more opportunities for optimization and can achieve a greater advantage over gate-based compilation. From inspection of Figure 3a, we see that the depth of Fixed blocks is determined by the frequency of $R_z(\theta_i)$ gates. For our benchmarked VQE-UCCSD circuits, $R_z(\theta_i)$ gates comprise only 5-8% of the total number of gates, so the Fixed subcircuits have reasonably long depths. For our benchmarked QAOA circuits however, the $R_z(\theta_i)$ gates comprise 15-28% of the total number of gates, so the Fixed subcircuits have short depths and the potential advantage of strict partial compilation is limited. This motivates us to consider other strategies that more closely match the pulse speedups of full GRAPE.

7 FLEXIBLE PARTIAL COMPILATION

As strict partial compilation is bottlenecked by the depth of Fixed subcircuits, we are motivated to consider strategies that create deeper subcircuits. The core idea behind flexible partial compilation is to create subcircuits that are only 'slightly' parametrized, in that they depend on at most one of the θ_i variational parameters. As discussed below, we can perform hyperparameter tuning to ensure that GRAPE finds optimized pulses for single-angle parametrized subcircuits much faster than for general subcircuits.

7.1 Parameter Monotonicity

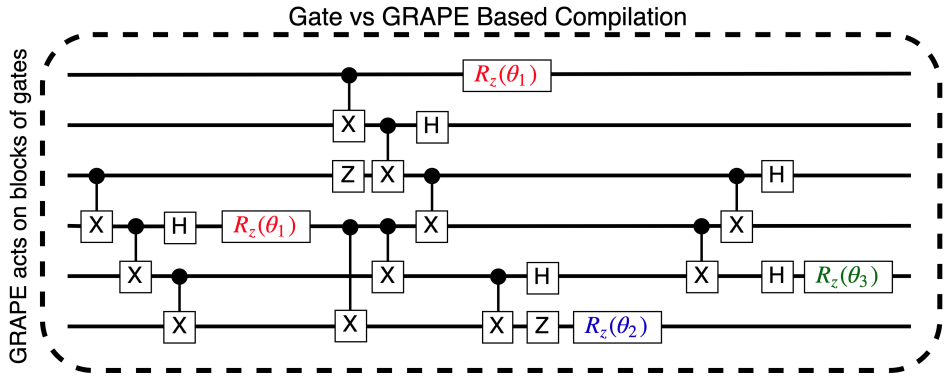
An initial strategy for creating these single-angle parametrized subcircuits would be to merge each consecutive pair of Fixed and $R_z(\theta_i)$ subcircuits into a single subcircuit that only depends on θ_i. However, this strategy would add at most one gate of depth to each subcircuit, which would not lead to significantly better pulses. However, we make a key observation which we term *parameter monotonicity*. For both the VQE UCCSD and QAOA circuits, the appearances of θ_i-dependent gates is monotonic in i–once a θ_i dependent gate appears, the subsequent parametrization-dependent gates must be θ_j for $j \geq i$. As a result, subcircuits with the same value of θ_i must be *consecutive*. For example, the sequence of angles in parametrization-dependent gates could be $[\theta_1, \theta_1, \theta_2, \theta_3]$ as in Figure 3a, but not $[\theta_1, \theta_2, \theta_3, \theta_1]$.

At a high level, parameter monotonicity for VQE/UCCSD and QAOA arise because their circuit constructions sequentially apply a circuit corresponding to each parameter exactly once. For instance, in QAOA, each parameter corresponds to the magnitude of Mixing or Cost-Optimization during the ith round–once the corresponding Mixing or Cost-Optimization has been applied, the circuit no longer depends on that parameter. Parameter monotonicity is not immediately obvious from visual inspection of variational circuits, because the circuit constructions and optimizations transform individual θ_i-dependent gates to ones that are parametrized in terms $-\theta_i$ or $\theta_i/2$. We resolve these latent dependencies by explicitly tagging the dependent parameter in software during the variational circuit construction phase.

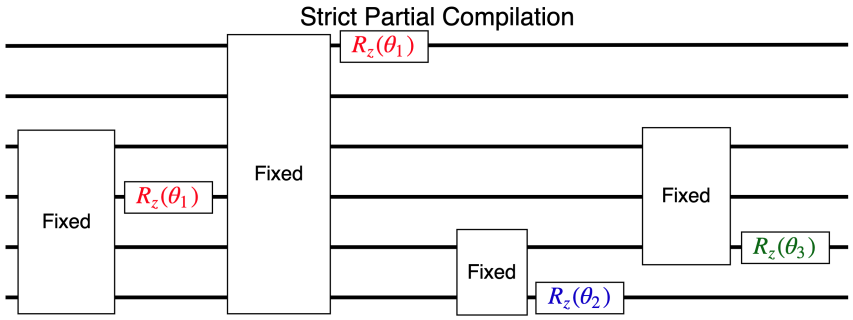
The implication of parameter monotonicity is that the subcircuits considered by flexible partial compilation are significantly deeper than the ones considered by strict partial compilation. Figure 3c demonstrates a small example; note that the θ_1-dependent subcircuit indicated by red dashed lines is significantly deeper than the subcircuits generated by strict partial compilation.

7.2 Hyperparameter Optimization

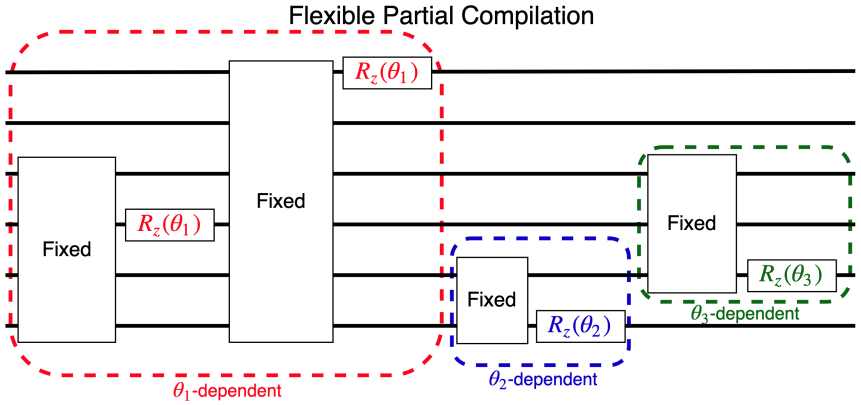
In GRAPE, an optimal control pulse is one that minimizes a set of cost functions corresponding to control amplitude, target state infidelity, and evolution time, among others[27]. To obtain an optimal control pulse, the GRAPE algorithm manipulates a set of time-discrete control fields that act on a quantum system. It may analytically compute gradients of the cost functions to be minimized with respect to the control fields. These gradients are used to update control fields with an optimizer such as ADAM or L-BFGS-B.



(a) This is a representative variational circuit, decomposed into gates. In gate-based compilation, each gate is translated by a lookup table to analog control pulses. Compilation amounts to simple concatenation of these control pulses. GRAPE (denoted by the dashed line) considers the unitary matrix for the full circuit and performs gradient descent to find the shortest control pulses that realize the circuit. GRAPE achieves significant pulse speedups, but has substantial compilation latency.



(b) Strict partial compilation blocks the circuit into a strictly alternating sequence of Fixed (parametrization-independent) subcircuits and $R_z(\theta_i)$ gates. Each Fixed subcircuit is precompiled with GRAPE, so that compilation at runtime simply involves concatenating the pulses for each subcircuit.



(c) Flexible partial compilation blocks the circuit into subcircuits that depend on exactly one parameter, θ_i. Parameter monotonicity ensures that these subcircuits have significantly longer depth than the Fixed blocks of strict partial compilation. We use hyperparameter optimization to precompute good hyperparameters (learning rate and decay rate) for each subcircuit. When all θ_i are specified at runtime, we used the tuned hyperparameters to quickly find optimized pulses for each subcircuit.

Figure 3: Comparison of compilation strategies. Subfigure (a) depicts gate-based and GRAPE- based compilation for a variational circuit. These two compilation approaches represent opposite ends of a spectrum trading off between between compilation latency and control pulse speedup. We introduce two new compilation strategies, strict and flexible partial compilation, that approach the pulse speedup of GRAPE without the large compilation latency. Subfigures (b) and (c) demonstrates strict and flexible partial compilation respectively.

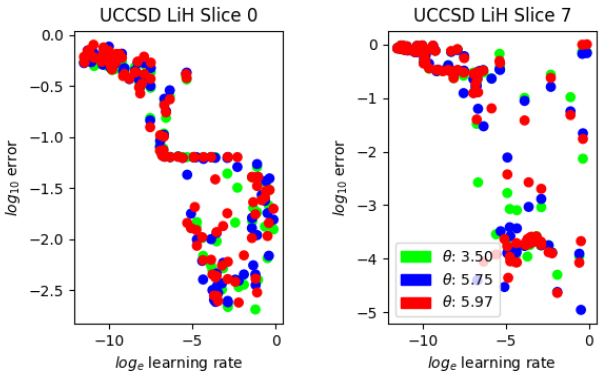


Figure 4: The 0^{th} single-angle dependent subcircuit of the UCCSD LiH circuit has two angle dependent gates, the 7^{th} has eight. These four qubit circuits are representative of the circuits studied in this work as well as larger future circuits due to the necessity of circuit blocking for circuits with more than four qubits. The graphs above plot GRAPE error against ADAM learning rate. For each permutation of the argument of the angle dependent gates in the subcircuits, the same range of learning rate values achieves the lowest error.

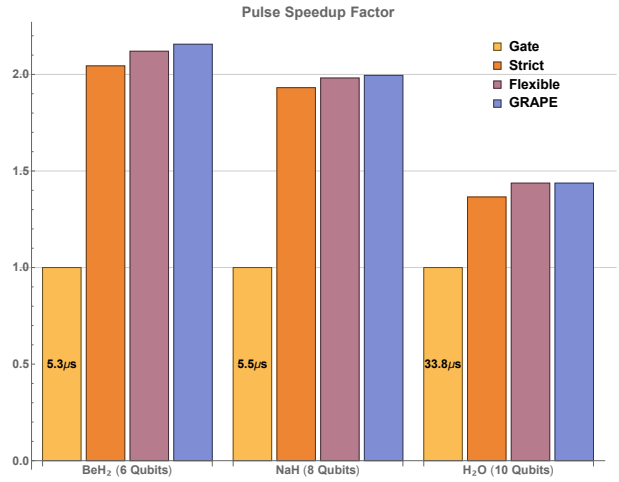


Figure 5: Pulse speedup factors (relative to gate based compilation) for VQE circuits. Full QOC 1.5-2x reductions in pulse durations for these circuits. Strict and flexible partial compilation recover 95% and 99% of this speedup respectively. Detailed results are reported in Table 4.

As opposed to the control fields, which are parameters manipulated by GRAPE, these optimizers have their own parameters such as learning rate and learning rate decay. These parameters are termed hyperparameters because they are set before the learning process begins.

Because they are inputs to the learning process, there is no closed form expression relating hyperparameters and the cost functions a learning model is minimizing. This makes hyperparameter optimization an ideal candidate for derivative free optimization techniques. Recent work has shown that tuning hyperparameters with methods such as bayesian optimization and radial basis functions can significantly improve performance for stochastic and expensive objectives such as minimizing the training error of neural networks [11, 51]. In our work, we employ hyperparameter optimization on GRAPE's ADAM optimizer. We realize faster convergence to a desired error rate over the baseline, significantly reducing compilation latency.

In particular, we make the observation that a high-performing hyperparameter configuration for a single-angle parameterized subcircuit is robust to changes in the argument of its θ_i-dependent gates, as shown in Figure 4. Therefore, we are able to precompute high-performing hyperparameter configurations for each single-angle parameterized subcircuit and employ them in compilation. For each iteration of a variational algorithm, the argument of the θ_i-dependent gates of each subcircuit will change, but the same hyperparameters are specified to GRAPE's optimizer, maintaining the same reduced compilation latency.

8 RESULTS

Our results were collected using over 200,000 CPU-core hours on Intel Xeon E5-2680 processors, using up to 64 GB of memory per GRAPE process. The large volume of compute is a result of both the high cost of running GRAPE and the number and large circuit size of benchmarks. We fixed randomization seeds when appropriate for both reproducibility and consistency between identical benchmarks. Our results are available in Jupyter notebooks on our Github repository [18].

8.1 Pulse Speedups

Figure 5 shows the pulse times speedup factors across our QAOA benchmarks for partial compilation and for full GRAPE, normalized to the gate-based compilation baseline. We present the normalized speedup factors, because the H_2O VQE-UCCSD benchmark is 10x larger; the raw pulse times are presented in Table 4.

For the BeH_2 and NaH VQE-UCCSD benchmarks, full GRAPE gives a 2.15x and 2.00x speedup in pulse duration respectively. Strict partial compilation is able to recover almost this full advantage, with speedups at 2.04x and 1.93x respectively. As discussed in Section 6, this matches the expectations, because the VQE-UCCSD benchmarks have relatively deep Fixed subcircuits. Finally, the speedups for flexible partial compilation are 2.12x and 1.98x, which nearly closes the gap between strict partial compilation and GRAPE.

The H_2O benchmark has similar relative speedups between strict, flexible, and GRAPE, with factors of 1.37, 1.44, 1.44.[2] However, the advantage over gate based compilation is smaller than for the BeH_2 and NaH benchmarks.

Figure 6 shows results for QAOA benchmarks. Strict partial compilation has speedups of 1.22x and 1.33x across the $N = 6$

[2]In fact, the pulse speedup for flexible partial compilation exactly matches GRAPE, because each 4-qubit block handled by GRAPE depends on at most one parameter.

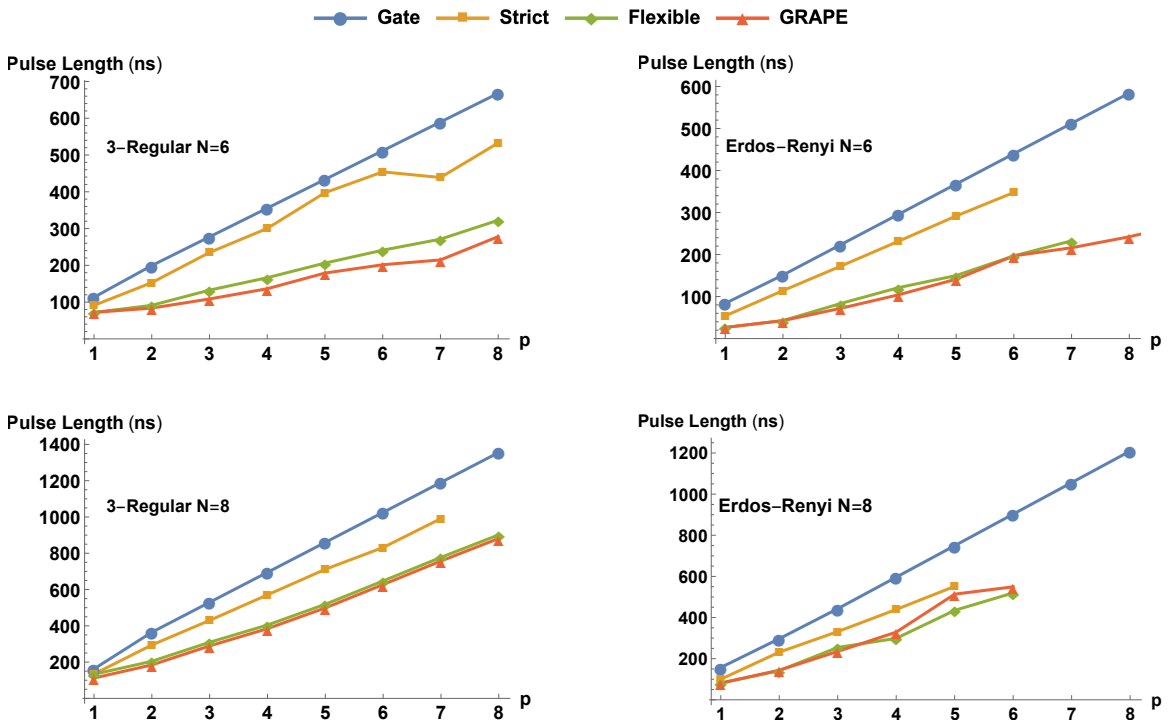


Figure 6: Pulse durations for QAOA MAXCUT benchmarks under the four compilation techniques, across all benchmarks. The gate based pulse time always increases linearly in p, the number of repeated rounds in the QAOA circuit. The average GRAPE pulse speedup is 2.6x for 6-node graphs and 1.8x for 8-node graphs. Strict partial compilation only achieves a modest speedup over gate based compilation, but flexible partial compilation essentially matches the GRAPE speedup exactly. The omitted data points correspond to computations that did not complete in 12 CPU-core hours, even after parallelizing subcircuit jobs.

						Max-Cut							
Compilation	UCCSD					3-Regular, N=6		Erdos-Renyi, N=6		3-Regular, N=8		Erdos-Renyi, N=8	
Techniques	H_2	LiH	BeH_2	NaH	H_2O	p=1	p=5	p=1	p=5	p=1	p=5	p=1	p=5
Gate-based	35.3	871.1	5308.3	5490.4	33842.2	113.2	433.6	83.7	367.8	162.5	860.0	157.1	749.5
Strict Partial	15.0	307.0	2596.5	2842.7	24781.4	91.2	397.6	54.0	291.8	134.0	711.6	100.0	551.7
Flexible Partial	5.0	84.0	2503.8	2770.8	23546.7	72.0	206.2	26.4	150.0	112.0	498.9	80.5	434.8
Full GRAPE	3.1	19.3	2461.7	2752.0	23546.7	72.0	179.0	26.6	141.2	112.0	498.9	81.6	513.7

Table 4: Experimental results for pulse durations (in nanoseconds) across the VQE-UCCSD and QAOA benchmarks.

and $N = 8$ qubit benchmarks respectively. By contrast, flexible partial compilation has average speedups of 2.3x and 1.8x across the $N = 6$ and $N = 8$ benchmarks, which almost matches the results from GRAPE. This separation between strict and flexible partial compilation matches the expected results discussed in Section 6. The high frequency of parametrized gates in QAOA limits the depth of Fixed blocks, so strict blocking has limited mileage. However, due to the four-qubit maximum subcircuit size for GRAPE, each block will rarely depend on more than one θ_i parameter. On these single-angle dependent blocks, flexible partial compilation achieves the same pulse speedups as GRAPE.

8.2 Compilation Latency Reduction

Figure 7 shows the compilation latency reduction achieved by flexible partial compilation, relative to full GRAPE compilation. As described in Section 7.2, flexible partial compilation is able to dramatically speed up the gradient descent's convergence by tuning the learning rate and decay rate hyperparameters on a per-subcircuit basis. We note that the 3-regular graphs achieve particularly high compilation latency reduction factors of 80.3x and 81.9x. Across all benchmarks, the reduction in compile time is from hours to minutes, which is critical in the context of variational algorithms.

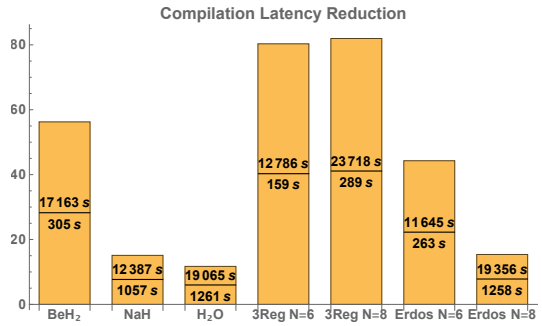


Figure 7: Reduction factors in compilation latency. The ratios indicate the average compilation latency using flexible partial compilation divided by latency using full GRAPE compilation. Flexible partial compilation uses about an hour of pre-compute time to determine the best learning rate - decay rate pair for each subcircuit.

8.3 Simulation with Realistic Pulses

While we performed our GRAPE runs without accounting for error or noise sources for simplicity, it can be adapted to account for these sources. For example, we could demand well-shaped pulses, account for leakage into higher states outside the binary qubit abstraction, or explicitly model the qubit decoherence times. To demonstrate that these sources can be accounted for, we re-ran two of our VQE and QAOA benchmarks with Full GRAPE using these more realistic assumptions:

- Allowing only 1 pulse datapoint every nanosecond (1 GSa/s), versus 20 GSa/s in the other results presented in this paper.
- Including leakage into the qutrit *leakage* level. Other results in this paper use the binary-qubit approximation, as outlined in the system Hamiltonian in Appendix A.
- Application of aggressive pulse regularization in GRAPE to ensure that the pulse shapes follow a Gaussian envelope and have smooth 1st and 2nd derivatives.

Table 4 compares the pulse speedups due to GRAPE, under both our standard (less realistic) GRAPE settings and under the more realistic settings that account for the three items above. For VQE and QAOA applications, the GRAPE speedups speedups are 11.4x (standard) vs 8.8x (realistic) and 4.5x (standard) vs. 3.0x (realistic) respectively. While the more realistic pulses do seem to have somewhat lower pulse speedup factors, they are similar and still feature significant speedup over gate based compilation.

8.4 Aggregate Impact on Total Runtime

For a quantitative sense of aggregate impact, we note that VQE requires thousands of iterations, even for small molecules. For instance, past work in VQE, towards estimating the ground state energy of BeH_2, required 3500 iterations [24]. Per Figure 7, this would amount to over 2 years of runtime compilation latency via Full-GRAPE. By contrast, strict partial compilation achieves zero runtime compilation latency via lookup table, and the pre-computed pulses for the fixed blocks were compiled in under 1 hour. Since the UCCSD ansatz has quartic scaling in the number of parameters

Gate ns → GRAPE ns (reduction)	H_2 VQE	Erdos-Renyi $N = 3$
Standard	35.3 → 3.1 (11.4x)	15.0 → 3.3 (4.5x)
More Realistic	420 → 48 (8.8x)	285 → 96 (3.0x)

Table 5: Speedups due to GRAPE compilation under the standard settings and under more realistic settings, which account for lower sampling rates, qutrit leakage, and pulse regularization. Results are given for the H_2 VQE benchmark and for the Erdos-Renyi $N = 3$ QAOA benchmark. The speedup factors due to GRAPE are similar with and without the more realistic assumptions.

[43], the number of iterations required scales aggressively for bigger molecules and the advantage of our approach will scale favorably. Further experimental work is needed to estimate the advantage of our approach for larger molecules in terms of total runtime, but extrapolation from small molecules BeH_2 seems promising. Similarly, while the improvement in quality-of-result due to the shorter pulse times from GRAPE is difficult to quantify without concrete experiments, we emphasize that the error due to decoherence scales exponentially with quantum runtime. Therefore, we again expect favorable results, owing to the significant pulse time speedup of our techniques relative to gate based compilation.

9 CONCLUSION

Variational quantum algorithms such as VQE and QAOA are strong candidates for demonstrating a quantum advantage in problems such as molecular ground state estimation, MAXCUT approximation, and prime factorization. Unlike prior algorithms, variational algorithms are parametrized, with the parameters at each iteration determined based on the results of previous iterations. Consequently, compilation is interleaved with computation. As a result, it is not practical to each variational circuit with out-of-the-box GRAPE, which takes several minutes to find an optimized pulse even on small (4-qubit) circuit.

Our partial compilation techniques offer a path to achieving the pulse speedups of GRAPE, without incurring its compilation latency. On the VQE-UCCSD circuits, our strict partial compilation strategy achieves 1.5x-2x pulse speedups over gate based compilation, almost matching the speedups from full GRAPE. Strict partial compilation is performed by precomputing optimal pulses for Fixed blocks. During execution, it has the same–essentially instant–lookup table based compilation procedure as gate based compilation. Thus, strict partial compilation is strictly better than gate based compilation.

For QAOA circuits, while strict partial compilation only achieves modest pulse speedup, we find that flexible partial compilation almost exactly matches the pulse speedups of GRAPE. Flexible partial compilation precomputes the best hyperparameters for each slice, so that when the θ_i parameters are specified at runtime, an optimized pulse sequence can be computed rapidly. For our benchmarked circuits, we found 10-100x reductions in compilation latency from flexible partial compilation, relative to full GRAPE compilation.

We emphasize that achieving optimized pulses is critical because error due to decoherence error is exponential in the pulse duration.

Thus, our pulse speedups are not merely about wall time speedups for quantum circuits, but moreso about making computations possible in the first place, before the qubits decohere.

10 FUTURE WORK

The industry adoption of the OpenPulse standard will usher an experimental era for pulse-level control. Running our partial compilation schemes on an actual machine will be valuable in terms of determining exactly how to weigh tradeoffs between pre-computation resources, compilation latency, and pulse durations.

On the computational side, we also see significant potential for extending the scalability of GRAPE. While past work has successfully used GRAPE on 10 qubit widths with very simple circuits (for example, 10 identical single-qubit rotations in parallel), we found that for complicated circuits, GRAPE only converges reliably with widths up to 4 qubits. This 4-qubit blocking width limits the depths of the subcircuits that both GRAPE and our partial compilation schemes can consider. For example, in the additional two VQE-UCCSD molecules benchmarks (H_2 and LiH) reported in Table 4, flexible partial compilation and full GRAPE achieve 7-50x pulse speedups because the benchmarks are 2 and 4 qubits in width. Thus, investigating the convergence properties of GRAPE and extending the circuit widths it reliably converges for will substantially extend the advantage that these techniques can achieve over gate based compilation.

ACKNOWLEDGMENTS

This work is funded in part by EPiQC, an NSF Expedition in Computing, under grant CCF-1730449 and in part by STAQ, under grant NSF Phy-1818914. Pranav Gokhale is supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program. 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] Mohamed Abdelhafez, David I. Schuster, and Jens Koch. 2019. Gradient-based optimal control of open quantum systems using quantum trajectories and automatic differentiation. arXiv:arXiv:1901.05541

[2] Eric R. Anschuetz, Jonathan P. Olson, Alán Aspuru-Guzik, and Yudong Cao. 2018. Variational Quantum Factoring. arXiv:arXiv:1808.08927

[3] Ryan Babbush, Jarrod McClean, Dave Wecker, Alán Aspuru-Guzik, and Nathan Wiebe. 2015. Chemical basis of Trotter-Suzuki errors in quantum chemistry simulation. *Phys. Rev. A* 91 (Feb 2015), 022311. Issue 2. https://doi.org/10.1103/PhysRevA.91.022311

[4] R. Barends, A. Shabani, L. Lamata, J. Kelly, A. Mezzacapo, U. Las Heras, R. Babbush, A. G. Fowler, B. Campbell, Yu Chen, Z. Chen, B. Chiaro, A. Dunsworth, E. Jeffrey, E. Lucero, A. Megrant, J. Y. Mutus, M. Neeley, C. Neill, P. J. J. O'Malley, C. Quintana, P. Roushan, D. Sank, A. Vainsencher, J. Wenner, T. C. White, E. Solano, H. Neven, and John M. Martinis. 2016. Digitized adiabatic quantum computing with a superconducting circuit. *Nature* 534 (08 Jun 2016), 222 EP –. https://doi.org/10.1038/nature17658

[5] Panagiotis Kl. Barkoutsos, Jerome F. Gonthier, Igor Sokolov, Nikolaj Moll, Gian Salis, Andreas Fuhrer, Marc Ganzhorn, Daniel J. Egger, Matthias Troyer, Antonio Mezzacapo, Stefan Filipp, and Ivano Tavernelli. 2018. Quantum algorithms for electronic structure calculations: Particle-hole Hamiltonian and optimized wave-function expansions. *Phys. Rev. A* 98 (Aug 2018), 022322. Issue 2. https://doi.org/10.1103/PhysRevA.98.022322

[6] Rodney J. Bartlett and Monika Musiał. 2007. Coupled-cluster theory in quantum chemistry. *Rev. Mod. Phys.* 79 (Feb 2007), 291–352. Issue 1. https://doi.org/10.1103/RevModPhys.79.291

[7] Yu Chen, C. Neill, P. Roushan, N. Leung, M. Fang, R. Barends, J. Kelly, B. Campbell, Z. Chen, B. Chiaro, A. Dunsworth, E. Jeffrey, A. Megrant, J. Y. Mutus, P. J. J. O'Malley, C. M. Quintana, D. Sank, A. Vainsencher, J. Wenner, T. C. White, Michael R. Geller, A. N. Cleland, and John M. Martinis. 2014. Qubit Architecture with High Coherence and Fast Tunable Coupling. *Phys. Rev. Lett.* 113 (Nov 2014), 220502. Issue 22. https://doi.org/10.1103/PhysRevLett.113.220502

[8] Yi Chou, Shang-Yu Huang, and Hsi-Sheng Goan. 2015. Optimal control of fast and high-fidelity quantum gates with electron and nuclear spins of a nitrogen-vacancy center in diamond. *Phys. Rev. A* 91 (May 2015), 052315. Issue 5. https://doi.org/10.1103/PhysRevA.91.052315

[9] Gavin E. Crooks. 2018. Performance of the Quantum Approximate Optimization Algorithm on the Maximum Cut Problem. arXiv:arXiv:1811.08419

[10] P. de Fouquieres, S. G. Schirmer, S. J. Glaser, and I. Kuprov. 2011. Second order gradient ascent pulse engineering. *Journal of Magnetic Resonance* 212 (Oct. 2011), 412–417. https://doi.org/10.1016/j.jmr.2011.07.023 arXiv:quant-ph/1102.4096

[11] Gonzalo I Diaz, Achille Fokoue-Nkoutche, Giacomo Nannicini, and Horst Samulowitz. 2017. An effective algorithm for hyperparameter optimization of neural networks. *IBM Journal of Research and Development* 61, 4/5 (2017), 9–1.

[12] Florian Dolde, Ville Bergholm, Ya Wang, Ingmar Jakobi, Boris Naydenov, Sébastien Pezzagna, Jan Meijer, Fedor Jelezko, Philipp Neumann, Thomas Schulte-Herbrüggen, Jacob Biamonte, and Jörg Wrachtrup. 2014. High-fidelity spin entanglement using optimal control. *Nature Communications* 5 (28 Feb 2014), 3371 EP –. https://doi.org/10.1038/ncomms4371 Article.

[13] Henry Eyring. 1935. The Activated Complex in Chemical Reactions. *The Journal of Chemical Physics* 3, 2 (1935), 107–115. https://doi.org/10.1063/1.1749604 arXiv:https://doi.org/10.1063/1.1749604

[14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A Quantum Approximate Optimization Algorithm. arXiv:arXiv:1411.4028

[15] Edward Farhi and Aram W Harrow. 2016. Quantum Supremacy through the Quantum Approximate Optimization Algorithm. arXiv:arXiv:1602.07674

[16] X. Fu, L. Riesebos, M. A. Rol, J. van Straten, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, V. Newsum, K. K. L. Loh, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. 2019. eQASM: An Executable Quantum Instruction Set Architecture. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 224–237. https://doi.org/10.1109/HPCA.2019.00040

[17] 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. 2015. Training Schrödinger's cat: quantum optimal control. *The European Physical Journal D* 69, 12 (17 Dec 2015), 279. https://doi.org/10.1140/epjd/e2015-60464-1

[18] Pranav Gokhale, Yongshan Ding, Thomas Propson, and Christopher Winkler. 2019. Code and Results: Partial Compilation of Variational Algorithms. https://github.com/EPiQC/PartialCompilation.

[19] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: A Scalable Quantum Programming Language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '13)*. ACM, New York, NY, USA, 333–342. https://doi.org/10.1145/2491956.2462177

[20] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Annual ACM Symposium on Theory of Computing*. ACM, 212–219.

[21] IBM. 2019. The Qiskit Roadmap 2019. https://github.com/Qiskit/qiskit/blob/master/docs/development_strategy.rst.

[22] Ali Javadi-Abhari, Arvin Faruque, Mohammad Javad Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, Fred Chong, Margaret Martonosi, Martin Suchara, Ken Brown, Massoud Pedram, and Todd Brun. 2012. Scaffold: Quantum Programming Language.

[23] Ali Javadi-Abhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. 2014. ScaffCC: A Framework for Compilation and Analysis of Quantum Computing Programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers (CF '14)*. ACM, New York, NY, USA, Article 1, 10 pages. https://doi.org/10.1145/2597917.2597939

[24] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. 2017. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature* 549 (13 Sep 2017), 242 EP –. https://doi.org/10.1038/nature23879

[25] Navin Khaneja, Timo Reiss, Cindie Kehlet, Thomas Schulte-Herbrüggen, and Steffen J. Glaser. 2005. Optimal control of coupled spin dynamics: design of NMR pulse sequences by gradient ascent algorithms. *Journal of Magnetic Resonance* 172, 2 (2005), 296 – 305. https://doi.org/10.1016/j.jmr.2004.11.004

[26] Daniel Kudrow, Kenneth Bier, Zhaoxia Deng, Diana Franklin, Yu Tomita, Kenneth R. Brown, and Frederic T. Chong. 2013. Quantum Rotations: A Case Study in Static and Dynamic Machine-code Generation for Quantum Computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13)*. ACM, New York, NY, USA, 166–176. https://doi.org/10.1145/2485922.2485937

[27] Nelson Leung, Mohamed Abdelhafez, Jens Koch, and David Schuster. 2017. Speedup for quantum optimal control from automatic differentiation based

on graphics processing units. *Phys. Rev. A* 95 (Apr 2017), 042318. Issue 4. https://doi.org/10.1103/PhysRevA.95.042318
[28] Seth Lloyd. 1996. Universal quantum simulators. *Science* (1996), 1073–1078.
[29] Seth Lloyd. 2018. Quantum approximate optimization is computationally universal. arXiv:arXiv:1812.11075
[30] Seth Lloyd and Reevu Maity. 2019. Efficient implementation of unitary transformations. arXiv:arXiv:1901.03431
[31] Sam McArdle, Suguru Endo, Alan Aspuru-Guzik, Simon Benjamin, and Xiao Yuan. 2018. Quantum computational chemistry. arXiv:arXiv:1808.10402
[32] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. 2016. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics* 18, 2 (feb 2016), 023023. https://doi.org/10.1088/1367-2630/18/2/023023
[33] David C. McKay, Thomas Alexander, Luciano Bello, Michael J. Biercuk, Lev Bishop, Jiayin Chen, Jerry M. Chow, Antonio D. Córcoles, Daniel Egger, Stefan Filipp, Juan Gomez, Michael Hush, Ali Javadi-Abhari, Diego Moreda, Paul Nation, Brent Paulovicks, Erick Winston, Christopher J. Wood, James Wootton, and Jay M. Gambetta. 2018. Qiskit Backend Specifications for OpenQASM and OpenPulse Experiments. arXiv:arXiv:1809.03452
[34] David C. McKay, Christopher J. Wood, Sarah Sheldon, Jerry M. Chow, and Jay M. Gambetta. 2017. Efficient Z gates for quantum computing. *Phys. Rev. A* 96 (Aug 2017), 022330. Issue 2. https://doi.org/10.1103/PhysRevA.96.022330
[35] Nikolaj Moll, Panagiotis Barkoutsos, Lev S Bishop, Jerry M Chow, Andrew Cross, Daniel J Egger, Stefan Filipp, Andreas Fuhrer, Jay M Gambetta, Marc Ganzhorn, Abhinav Kandala, Antonio Mezzacapo, Peter Müller, Walter Riess, Gian Salis, John Smolin, Ivano Tavernelli, and Kristan Temme. 2018. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology* 3, 3 (jun 2018), 030503. https://doi.org/10.1088/2058-9565/aab822
[36] Yunseong Nam, Jwo-Sy Chen, Neal C. Pisenti, Kenneth Wright, Conor Delaney, Dmitri Maslov, Kenneth R. Brown, Stewart Allen, Jason M. Amini, Joel Apisdorf, Kristin M. Beck, Aleksey Blinov, Vandiver Chaplin, Mika Chmielewski, Coleman Collins, Shantanu Debnath, Andrew M. Ducore, Kai M. Hudek, Matthew Keesan, Sarah M. Kreikemeier, Jonathan Mizrahi, Phil Solomon, Mike Williams, Jaime David Wong-Campos, Christopher Monroe, and Jungsang Kim. 2019. Ground-state energy estimation of the water molecule on a trapped ion quantum computer. arXiv:arXiv:1902.10171
[37] Michael A. Nielsen and Isaac L. Chuang. 2011. *Quantum Computation and Quantum Information: 10th Anniversary Edition* (10th ed.). Cambridge University Press, New York, NY, USA.
[38] Joe O'Gorman and Earl T. Campbell. 2017. Quantum computation with realistic magic-state factories. *Phys. Rev. A* 95 (Mar 2017), 032338. Issue 3. https://doi.org/10.1103/PhysRevA.95.032338
[39] J. S. Otterbach, R. Manenti, N. Alidoust, A. Bestwick, M. Block, B. Bloom, S. Caldwell, N. Didier, E. Schuyler Fried, S. Hong, P. Karalekas, C. B. Osborn, A. Papageorge, E. C. Peterson, G. Prawiroatmodjo, N. Rubin, Colm A. Ryan, D. Scarabelli, M. Scheer, E. A. Sete, P. Sivarajah, Robert S. Smith, A. Staley, N. Tezak, W. J. Zeng, A. Hudson, Blake R. Johnson, M. Reagor, M. P. da Silva, and C. Rigetti. 2017. Unsupervised Machine Learning on a Hybrid Quantum Computer. arXiv:arXiv:1712.05771
[40] Hauke Paulsen and Alfred X Trautwein. 2004. Density functional theory calculations for spin crossover complexes. (2004), 197–219.
[41] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications* 5 (23 Jul 2014), 4213 EP –. https://doi.org/10.1038/ncomms5213 Article.
[42] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (Aug. 2018), 79. https://doi.org/10.22331/q-2018-08-06-79
[43] Jonathan Romero, Ryan Babbush, Jarrod R McClean, Cornelius Hempel, Peter J Love, and Alán Aspuru-Guzik. 2018. Strategies for quantum computing molecular energies using the unitary coupled cluster ansatz. *Quantum Science and Technology* 4, 1 (oct 2018), 014008. https://doi.org/10.1088/2058-9565/aad3e4
[44] 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 *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 1031–1044.
[45] Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (Oct. 1997), 1484–1509. https://doi.org/10.1137/S0097539795293172
[46] Robert S. Smith, Michael J. Curtis, and William J. Zeng. 2016. A Practical Quantum Instruction Set Architecture. *CoRR* abs/1608.03355 (2016).
[47] Damian S. Steiger, Thomas Häner, and Matthias Troyer. 2018. ProjectQ: an open source software framework for quantum computing. *Quantum* 2 (Jan. 2018), 49. https://doi.org/10.22331/q-2018-01-31-49
[48] Martin Suchara, Arvin Faruque, Ching-Yi Lai, Gerardo Paz, Frederic T. Chong, and John Kubiatowicz. 2013. Comparing the Overhead of Topological and Concatenated Quantum Error Correction. Part of the work was in Proceedings of IEEE International Conference on Computer Design (ICCD) 2013. *arXiv preprint arXiv:1312.2316* (2013). https://doi.org/10.1109/ICCD.2013.6657074 arXiv:arXiv:1312.2316
[49] Qiming Sun, Timothy C. Berkelbach, Nick S. Blunt, George H. Booth, Sheng Guo, Zhendong Li, Junzi Liu, James D. McClain, Elvira R. Sayfutyarova, Sandeep Sharma, Sebastian Wouters, and Garnet Kin-Lic Chan. 2017. PySCF: the Python-based simulations of chemistry framework. , e1340 pages. https://doi.org/10.1002/wcms.1340 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/wcms.1340
[50] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q#: Enabling Scalable Quantum Computing and Development with a High-level DSL. In *Proceedings of the Real World Domain Specific Languages Workshop 2018 (RWDSL2018)*. ACM, New York, NY, USA, Article 7, 10 pages. https://doi.org/10.1145/3183895.3183901
[51] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 847–855.
[52] Daochen Wang, Oscar Higgott, and Stephen Brierley. 2018. A Generalised Variational Quantum Eigensolver. *arXiv preprint arXiv:1802.00171* (2018).
[53] Dave Wecker and Krysta M. Svore. 2014. LIQUi|>: A Software Design Architecture and Domain-Specific Language for Quantum Computing. arXiv:arXiv:1402.4467

A SYSTEM HAMILTONIAN

Although our techniques are general and apply to any quantum computer, the pulses produced by GRAPE are specific to the underlying hardware platform. We chose to compile to control pulses for a quantum computer with gmon superconducting qubits [7], because this qubit type is one of the leader contenders for scalable quantum machines. For instance, the gmon qubit is central to Google's experimental efforts for demonstrating quantum supremacy.

The control pulse inputs that we specified to GRAPE were based on the gmon's system Hamiltonian. Each qubit, j, has a flux-drive control pulse and a charge-drive control pulse which have respective Hamiltonians, truncated to the qubit subspace:

$$H_{c,j}(t) = \sum_{j=1}^{N} \Omega_{c,j}(t)(a_j^\dagger + a_j) = \sum_{j=1}^{N} \Omega_{c,j}(t) \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

and

$$H_{f,j}(t) = \sum_{j=1}^{N} \Omega_{f,j}(t)(a_j^\dagger a_j) = \sum_{j=1}^{N} \Omega_{f,j} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

It can be seen from exponentiating these matrices that the control pulses correspond to $R_x(\theta)$ and $R_z(\phi)$ type gates respectively. We chose maximium drive amplitudes of $|\Omega_{c,j}(t)| \leq 2\pi \times 0.1$ GHz and $|\Omega_{f,j}(t)| \leq 2\pi \times 1.5$ GHz. These values, including the asymmetry between charge and flux drive, are representative of typical machines.

In addition to these single qubit terms, there is a control pulse for each pair of connected qubits. We consider a rectangular-grid topology with nearest-neighbor connectivity. Between each connected pair of qubits j and k, the corresponding control Hamiltonian is

$$H_{j,k}(t) = g(t)(a_j^\dagger + a_j)(a_k^\dagger + a_k)$$

This two-qubit interaction type corresponds to the entangling iSWAP gate (which swaps two qubits and also applies a phase factor). We use a maximum coupling strength of $|g(t)| \leq 2\pi \times 50$ MHz

Within the GRAPE software, we discretized the control pulses to 0.05 ns time slices. We set a target fidelity of 99.9% for each invocation of GRAPE. Raw data from all of our GRAPE runs are available at our Github repository [18].