PAPER

Domain-specific compilers for dynamic simulations of quantum materials on quantum computers

Quantum Science and Technology

RECEIVED
2 September 2020

ACCEPTED FOR PUBLICATION
6 October 2020

PUBLISHED
23 December 2020

PAPER

Domain-specific compilers for dynamic simulations of quantum materials on quantum computers

Lindsay Bassman[1,*], Sahil Gulania[2], Connor Powers[1], Rongpeng Li[3], Thomas Linker[1], Kuang Liu[1], T K Satish Kumar[4], Rajiv K Kalia[1], Aiichiro Nakano[1] and Priya Vashishta[1]

[1] Collaboratory for Advanced Computing and Simulations, University of Southern California, Los Angeles, CA, United States of America
[2] Department of Chemistry, University of Southern California, Los Angeles, CA, United States of America
[3] Department of Physics, University of Southern California, Los Angeles, CA, United States of America
[4] Department of Computer Science, University of Southern California, Los Angeles, CA, United States of America
[*] Author to whom any correspondence should be addressed.

E-mail: LBassman@lbl.gov

Keywords: quantum compiler, materials simulations, quantum computing

Supplementary material for this article is available online

Abstract
Simulation of the dynamics of quantum materials is emerging as a promising scientific application for noisy intermediate-scale quantum (NISQ) computers. Due to their high gate-error rates and short decoherence times, however, NISQ computers can only produce high-fidelity results for those quantum circuits smaller than some given circuit size. Dynamic simulations, therefore, pose a challenge as current algorithms produce circuits that grow in size with each subsequent time-step of the simulation. This underscores the crucial role of quantum circuit compilers to produce executable quantum circuits of minimal size, thereby maximizing the range of physical phenomena that can be studied within the NISQ fidelity budget. Here, we present two domain-specific (DS) quantum circuit compilers for the Rigetti and IBM quantum computers, specifically designed to compile circuits simulating dynamics under a special class of time-dependent Hamiltonians. The compilers outperform state-of-the-art general-purpose compilers in terms of circuit size reduction by around 25%–30% as well as wall-clock compilation time by around 40% (dependent on system size and simulation time-step). Drawing on heuristic techniques commonly used in artificial intelligence, both compilers scale well with simulation time-step and system size. Code for both compilers is open-source and packaged into a full-stack quantum simulation software with tutorials included for ease of use for future researchers wishing to perform dynamic simulations of quantum materials on quantum computers. As our DS compilers provide significant improvements in both compilation time and simulation fidelity, they provide a building block for accelerating progress toward physical quantum supremacy.

1. Introduction

Quantum computers promise to solve certain classes of problems that are intractable on the most advanced classical computers by massively reducing either time-to-solution, computational resource requirements, or both. By storing and processing information on quantum bits, or qubits, quantum computers take advantage of quantum mechanical phenomena, such as superposition and entanglement, to beat performance of their classical counterparts. Nearly forty years after their theoretical conception as universal quantum simulators [1], quantum computers are beginning to produce early successes in simulating quantum systems. However, currently available and near-future quantum computers, commonly known as noisy intermediate-scale quantum (NISQ) computers, suffer from high gate- and measurement-error rates, as well as qubit decoherence [2]. Furthermore, due to their small numbers of qubits, NISQ computers are unable to exploit robust error-correcting schemes which are expected to give rise to fault-tolerant quantum computers in the future. As a result, the only physical systems that have been simulated on quantum computers to date [3–8] are too small to see a quantum advantage (i.e., these systems can still be simulated on classical computers). While such simulations have provided encouraging proofs-of-concept, the next great challenge is to learn new physics by simulating a quantum material on a quantum computer, which cannot feasibly be simulated on any classical computer.

A particularly promising route to discovering new physics with NISQ computers is the dynamic simulation of a strongly-correlated quantum material. On the one hand, the complexity of such simulations on classical computers grows exponentially with the number of particles in the system, quickly making simulations of even modestly-sized systems impossible to run on state-of-the-art classical supercomputers. On the other hand, only modestly-sized systems are required to study the dynamics of various physical models, making it possible to fit such simulations onto NISQ computers. Together, this means there may exist a 'goldilocks' quantum material, too large for classical computers but small enough for NISQ computers to simulate, with which to achieve physical quantum supremacy.

Dynamic simulations are carried out on quantum computers by creating a different quantum circuit for each time-step of the simulation. Current algorithms generally produce quantum circuits that grow in size with each time-step [9–12]. Since NISQ computers can only execute circuits up to a certain size with high fidelity, due to high gate-error rates and low qubit decoherence times, this puts a limit on the number of time-steps an NISQ computer can feasibly simulate. It is therefore crucial to develop quantum circuit compilers that can minimize circuit sizes enough to enable dynamic simulations on NISQ computers.

Still in their infancy, quantum circuit compilers are an integral part of the quantum computing software stack [13]. Their function is to transform a high-level quantum circuit, usually the output of an algorithm, into a circuit that can be executed on a quantum computer. High-level quantum circuits are defined by a set of arbitrary quantum logic gates, each of which can be represented by a unitary matrix, acting on different subsets of qubits in the system. Quantum computers, however, are only designed with the ability to perform a small, universal set of one- and two-qubit gates, called their native gate set. Products of these native gates can be designed to be equivalent to any N-qubit gate, and thus can be used to execute any high-level quantum circuit. A complicating factor in the NISQ-era is that different quantum computers have different native gate sets. Therefore, the executable quantum circuit produced by the quantum circuit compiler must only consist of those native gates specific to the quantum machine upon which it will be executed.

On top of mapping high-level circuits to native-gate circuits, the quantum circuit compiler is further expected to optimize the native-gate circuit, which in the NISQ-era equates to circuit size minimization. There are two main approaches for quantum circuit optimization: circuit synthesis and circuit optimization. Circuit synthesis takes as input a $2^N \times 2^N$ unitary matrix representing the action of all the gates of the circuit on the entire N-qubit system and outputs an optimal or near-optimal circuit of native gates derived by various methods [14–22]. For very small system sizes (2 and 3 qubits), methods have been developed that produce provably optimal (i.e. minimum-sized) circuits [23–27]. The upside of the circuit synthesis approach for dynamic simulations is that they can, in some cases, yield constant-depth circuits for every time-step. While the parameters within the output circuits, which are derived from the $2^N \times 2^N$ input unitary matrix, vary per time-step, the overall size of circuit remains constant. Therefore, if the output circuits are small enough for high-fidelity results on the NISQ computer, then one can, in principle, simulate evolution of a system out to an arbitrary number of time-steps. The downside, however, is that for an N-qubit circuit, the algorithm must work with a $2^N \times 2^N$ unitary matrix, causing the memory resources and wall-clock time of this approach to grow exponentially with system size. This may render the circuit synthesis approach unsuitable for compilation of the larger-scale simulation circuits required for new discoveries on NISQ computers.

Less well studied, is the circuit optimization approach which takes as input a high-level circuit and uses algebraic identities to perform gate permutations, transformations, insertions, and deletions to get an equivalent, near-optimal circuit of native gates [28, 29]. The drawback of this method is that while this method decreases circuit size for each time-step, the executable circuits still grow with increasing time-step. The upside, however, is that this method can be made to scale efficiently with system size, giving it the potential to work with circuits simulating large enough systems to demonstrate physical quantum supremacy. The main difficulty in applying this method is the large number of identities, along with the enormous number of permutations in which they can be applied, makes optimizing this method an NP-hard [30, 31] problem.

Since as few as $\mathcal{O}(1)$ gates can make a significant difference in the fidelity of a quantum circuit executed on an NISQ computer, the importance of quantum circuit optimization is paramount. Successfully discovering new physics with NISQ computers will require larger numbers of qubits, making the circuit synthesis method untenable due to its poor scaling behavior with system size. Therefore, near-future NISQ simulations require the scalability of the circuit optimization method. To address the NP-hardness of this approach, we propose developing compilers for a specific native gate set and a specific class of quantum circuits. Creating such domain-specific (DS) compilers allows the designer to take advantage of the structure of the specific problem to develop heuristics that enable better and faster optimization compared to a general-purpose compiler.

Here, we present two DS quantum circuit compilers for the cloud-accessible NISQ computers provided by Rigetti and IBM. Each DS compiler compiles circuits into the native gate set of one of these machines. Both are specifically designed for compiling circuits that simulate time evolution under a special class of Hamiltonians known as the transverse field Ising model (TFIM) with a time-dependent transverse field [32, 33]. The TFIM is considered a quintessential model for studying quantum phase transitions [34], as well as myriad condensed matter systems, such as ferroelectrics [35] and magnetic spin glasses [36]. When the transverse field is time-dependent, non-equilibrium effects such as dynamic phase transitions and quantum hysteresis can be studied [37–40]. Simulation of a time-dependent TFIM spin system on an NISQ computer is therefore an excellent candidate for the discovery of new physics and we choose this important model to demonstrate how NISQ-era dynamic simulations can benefit from DS compilers. To date, dynamic simulations of TFIM systems on NISQ computers have been limited to small systems sizes [12] and limited time-steps [11].

The DS compilers are able to reduce quantum circuit sizes by around 25%–30% and compilation times by around 40% compared to state-of-the-art general-purpose compilers (values vary depending on system size and simulation time-step). We have made the code for both DS compilers available [41], and furthermore, have integrated the compilers into a full-stack quantum simulation software (see SI for details (https://stacks.iop.org/QST/6/014007/mmedia)) so that the broader research community can more easily take advantage of the compilers to perform dynamic simulations of quantum materials on quantum computers. We anticipate that our DS compilers will enable longer-time dynamic simulations on near-future NISQ computers which would not otherwise be possible with general-purpose compilers.

2. Results

2.1. Importance of optimized quantum circuit compilers

To showcase the importance of optimized quantum circuit compilers, we begin by showing results from simulations performed on Rigetti's quantum computer for a two-qubit system using a general-purpose compiler with and without optimization. For two-qubit systems, Rigetti's and IBM's optimized general-purpose compilers use the circuit synthesis approach, enabling production of optimal constant-depth circuits for all time-steps. Without optimization, the general-purpose compilers use a naïve method that simply translates high-level circuits into native gate circuits with basic gate reduction schemes like concatenation of adjacent rotation gates, thereby creating executable circuits that grow with each time-step. The stark difference in results from simulations using the naïve general-purpose two-qubit compiler (blue) versus the optimized general-purpose two-qubit compiler (red) is shown in figure 1(a). Since the optimized, constant-depth circuits are small enough to produce high-fidelity results, the NISQ computer is able to simulate the system out to arbitrary time-steps, as can be seen by the close correspondence of the quantum computer results (red) to the numerical results (black) from a simulated quantum computer. The naïve compiler initially produces circuits that are small enough to achieve close correspondence with the numerical results. After a certain number of time-steps, however, it produces circuits that are too large, generating too much error and leading to results that drastically diverge from the ground truth. As all factors are held constant between the two simulations except the compiler, these results emphasize the importance of quantum circuit compilers for achieving high-fidelity results on NISQ computers.

2.2. Domain-specific compilers for TFIM circuits

In this article, we present two DS compilers for compilation of circuits simulating spin dynamics under a time-dependent TFIM Hamiltonian into the native gate set of either Rigetti's or IBM's quantum computer. The TFIM Hamiltonian, defined by

$$H(t) = -J_z\sum_{i=1}^{N-1}\sigma_i^z\sigma_{i+1}^z - J_x(t)\sum_{i=1}^{N}\sigma_i^x \tag{1}$$

models a system of spins with nearest-neighbor exchange interactions of strength J_z, in the presence of a transverse magnetic field with a time-dependent amplitude defined by $J_x(t)$. Here, σ_i^α is the α-Pauli matrix acting on qubit i. The system can be simulated a quantum computer by mapping the spins of the TFIM to

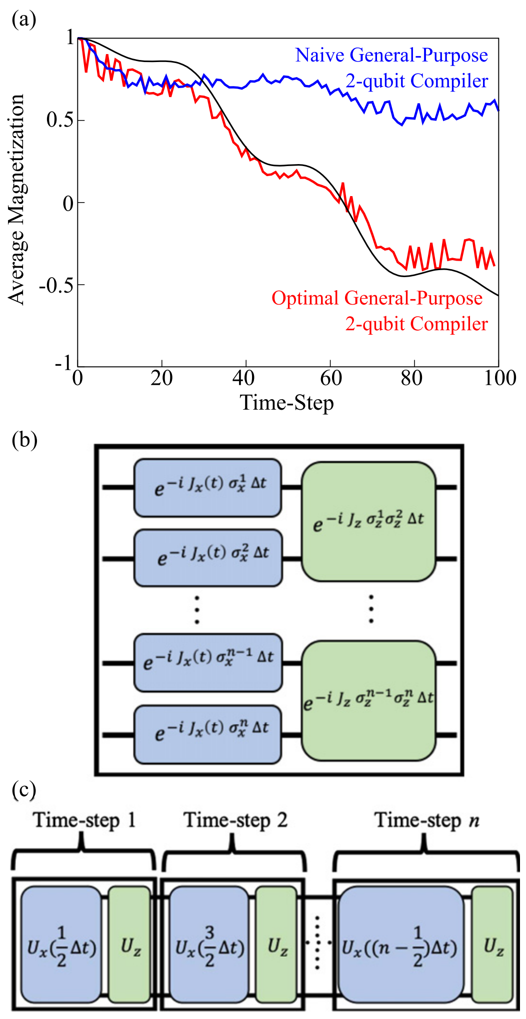


Figure 1. The importance of quantum circuit compilation and the general structure of TFIM circuits. (a) Comparison of simulation results from Rigetti's quantum computer for a two-qubit system compiled naively (blue) versus using optimized circuit synthesis (red). (b) Block of gates representing the evolution of the system by one time-step. (c) High-level quantum circuit depicting the evolution of the system by n time-steps.

the spins of the qubits. The dynamics of the system is simulated by applying to the qubits a sequence of quantum logic gates that is equivalent to time-evolution under Hamiltonian (1), given by the time-ordered unitary operator $U(t) \equiv U(0,t) = \mathcal{T}\ \exp(-i\int_0^t H(t)\mathrm{d}t)$ in the atomic unit, where $\mathcal{T}$ denotes the time-ordering operator. Deriving this sequence of quantum logic gates involves discretizing time and applying the Trotter approximation [42] to $U(t)$ to arrive at the following approximate time-evolution operator:

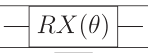
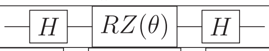
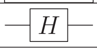
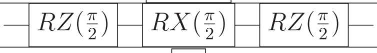
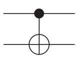
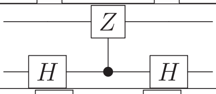
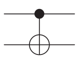
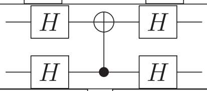
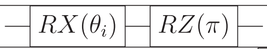
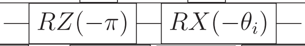
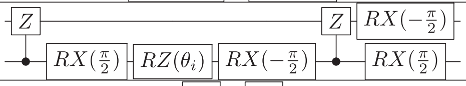
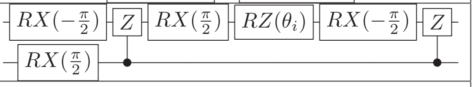
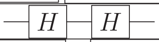
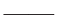
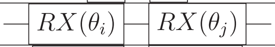
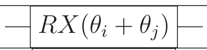
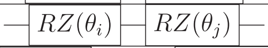
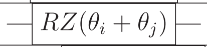
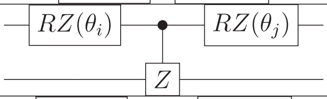
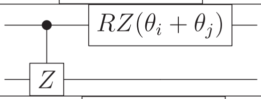
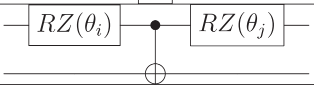
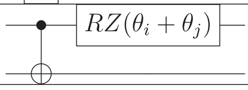
$$U(n\Delta t) = \prod_{j=0}^{n-1} \mathrm{e}^{-\mathrm{i}H_x((j+1/2)\Delta t)\Delta t}\ \mathrm{e}^{\mathrm{i}H_z\Delta t} + \mathcal{O}(\Delta t), \tag{2}$$

where, $H_x(t) = -J_X(t)\sum_{i=1}^{N} \sigma_i^x$ and $H_z = -J_z\sum_{i=1}^{N-1} \sigma_i^z\sigma_{i+1}^z$. More details can be found in the methods section.

Dynamic simulations are performed by creating a different quantum circuit for each time-step of the total simulation, where the circuit for time-step n simulates evolution from time $t = 0$ to $t = n\Delta t$ by implementing $U(n\Delta t)$. Examination of equation (2) shows that the circuit for time-step n consists of the product of a number of operators proportional to n. Circuits, therefore, grow in size with increasing time-step. The general form of the circuits is shown in figures 1(b) and (c).

Figure 1(b) shows a block of gates enacting evolution of the system by one time-step Δt. The blue boxes represent single-qubit operators that apply the transverse magnetic field to each qubit for a time Δt, while the green boxes represent two-qubit operators that apply the exchange interaction between nearest-neighbor qubits for a time Δt. Figure 1(c) shows a high-level circuit diagram for simulating evolution of the system for a total time $n\Delta t$. Note how the block of gates shown in figure 1(b) is repeated n times in figure 1(c), although the parameters of each block vary per time-step due to the time-dependence

Table 1. Algebraic identities for gate sets commonly found in TFIM circuits.

No.	Common gate set in TFIM circuits	Equivalent
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		

of the Hamiltonian. Nonetheless, a regular structure appears in the circuits for TFIM simulations, which can be exploited for developing heuristics for a DS quantum circuit compiler.

A closer examination of this circuit structure in terms of quantum gates gives insight into how to tailor such a DS compiler for TFIM circuits. First, all single-qubit operators (blue blocks of figure 1(b)) are of the form $\exp(-i\alpha\sigma_j^x)$, where α is a real variable and j identifies the qubit on which the operator acts. This action can be applied to qubit j with the gate $RX(2\alpha)[j]$, which rotates qubit j around the x-axis by an angle 2α. Second, all two-qubit operators (green blocks of figure 1(b)) are of the form $\exp(-i\beta\sigma_j^z\sigma_{j+1}^z)$, where β is a real variable and $j, j+1$ identify the qubits on which the operator acts. This action can be applied to qubits $j, j+1$ with the gate sequence $(CNOT[j, j+1], RZ(2\beta)[j], CNOT[j, j+1])$, where $RZ(\theta)[j]$ rotates qubit j around the z-axis by an angle θ and $CNOT$ is a two-qubit controlled-not gate. Therefore, when given a TFIM circuit, the compiler should expect to see a set of RX gates on all qubits interleaved with the sequence $CNOT, RZ, CNOT$ on nearest neighbor qubits. The DS compilers are therefore designed to take as input high-level circuits containing gates from the set $\{RX(\theta), RZ(\theta), CNOT\}$.

The native gate sets on IBM and Rigetti's quantum processors are slightly different, which is why we design two separate DS compilers: one to compile circuits for execution on Rigetti, the other to compile circuits for execution on IBM. Each utilizes different sets of gate identities to transform and reduce circuits to smaller sizes. All algebraic identities used for gate transformation and reduction in the DS compilers are detailed in table 1. The left-hand column numbers the identities for identification. The middle column gives sets of gates commonly found in uncompiled TFIM circuits, while the right-hand column shows the corresponding equivalent to each set of gates. Identities 1–3 are used for converting high-level gates into native gates. Note that while they do not reduce the gate count, they are essential for producing circuits that are executable on the quantum processors. Identities 4–6 are gate-altering and re-organizing identities, which prime circuits for further reductions. Finally, identities 7–11 are simplifying identities that are used purely to reduce gate count. Note that if $\theta_i = -\theta_j$ in identities 8–11, then a rotation by zero radians results, which can be completely removed from the circuit.

Domain knowledge of TFIM circuits, namely the fact that they are comprised of repeating blocks of gates as detailed in figures 1(b) and (c), as well as knowledge of the target native gates sets is used to choose the gate identities, as well as their ordering, for each DS compiler. We emphasize that not only is it important to implement identities specifically useful for a given circuit type and native gate set, it is also important to use knowledge of the circuit structure to develop heuristics for optimal ordering of identity application. Different permutations of identity application can lead to varied resultant circuits. While the majority of the identities in table 1 are utilized in the native general-purpose compilers of Rigetti and IBM, our DS compilers order these identities in an intelligent way that enables smaller circuit volumes and shorter compilation times. Both compilers begin by transforming non-native gates into their respective target native gate sets. A series of loops implementing various gate identities are then applied which are specially chosen using knowledge of the types of motifs (i.e., sequences of gates) that are expected to arise after applications of a given gate identity in a previous loop. Details on how each DS compiler is implemented, including pseudocode, are described in the subsections below.

2.3. Rigetti native gate set domain-specific compiler

Rigetti's native gate set is $\{RX(\pm\frac{\pi}{2}), RZ(\theta), CZ\}$, where θ is an arbitrary angle and CZ is the two-qubit controlled-Z gate. Algorithm 1 shows pseudocode detailing the order in which a subset of the identities from table 1 are applied in our DS compiler for Rigetti's native gate set. The first task the compiler handles is converting non-native gates in the high-level TFIM circuits to native ones. This is accomplished in the first five loops in algorithm 1, where (i) identity 1 is used to transform $RX(\theta)$ gates into a to the sequence $(H, RZ(\theta), H)$, where H is the Hadamard gate, since only RZ gates can handle arbitrary theta; (ii) identity 3 is used to transform non-native $CNOT[i, i+1]$ gates into the sequence $H[i+1], CZ[i, i+1], H[i+1]$ and (iii) inserted H gates are removed when two are adjacently applied on the same qubit (identity 7) or transformed into the native gate sequence $RZ(\frac{\pi}{2}), RX(\frac{\pi}{2}), RZ(\frac{\pi}{2})$ (identity 2).

Loops 1–5 were designed and ordered using knowledge of the structure of gates in the high-level TFIM circuit. For example, after loops 1 and 2, we know from the TFIM circuit structure that neighboring H gates will arise on the same qubit, and thus implement loop 3 to remove these redundant gates. Likewise, loop 4 identifies a motif that arises in the circuit after execution of loop 2 and transforms this motif into a cleverly chosen sequence of native gates, priming the circuit for application of identity 6 in loop 7. Once the entire circuit comprises only gates from Rigetti's native gate set, the compiler can then apply a sequence of gate identities on subsets of gates to optimize the volume of the circuit. Loops 6–10 take advantage of domain knowledge of TFIM circuits to search for motifs in the circuit we know will arise due to application gate identities in prior loops. In particular, the identity used in loop 7 is carefully crafted based on knowledge of the TFIM circuit structure after the first 6 loops to prepare the circuit for further gate reductions by moving a pair of RX gates across a gate set it commutes with to allow for concatenation of adjacent RX gates on the same qubit. Finally, not shown in the pseudocode, is the removal of any trailing gates at the end of the circuit that only alter the phase of the system, and thus do not affect any measurement gates that immediately follow.

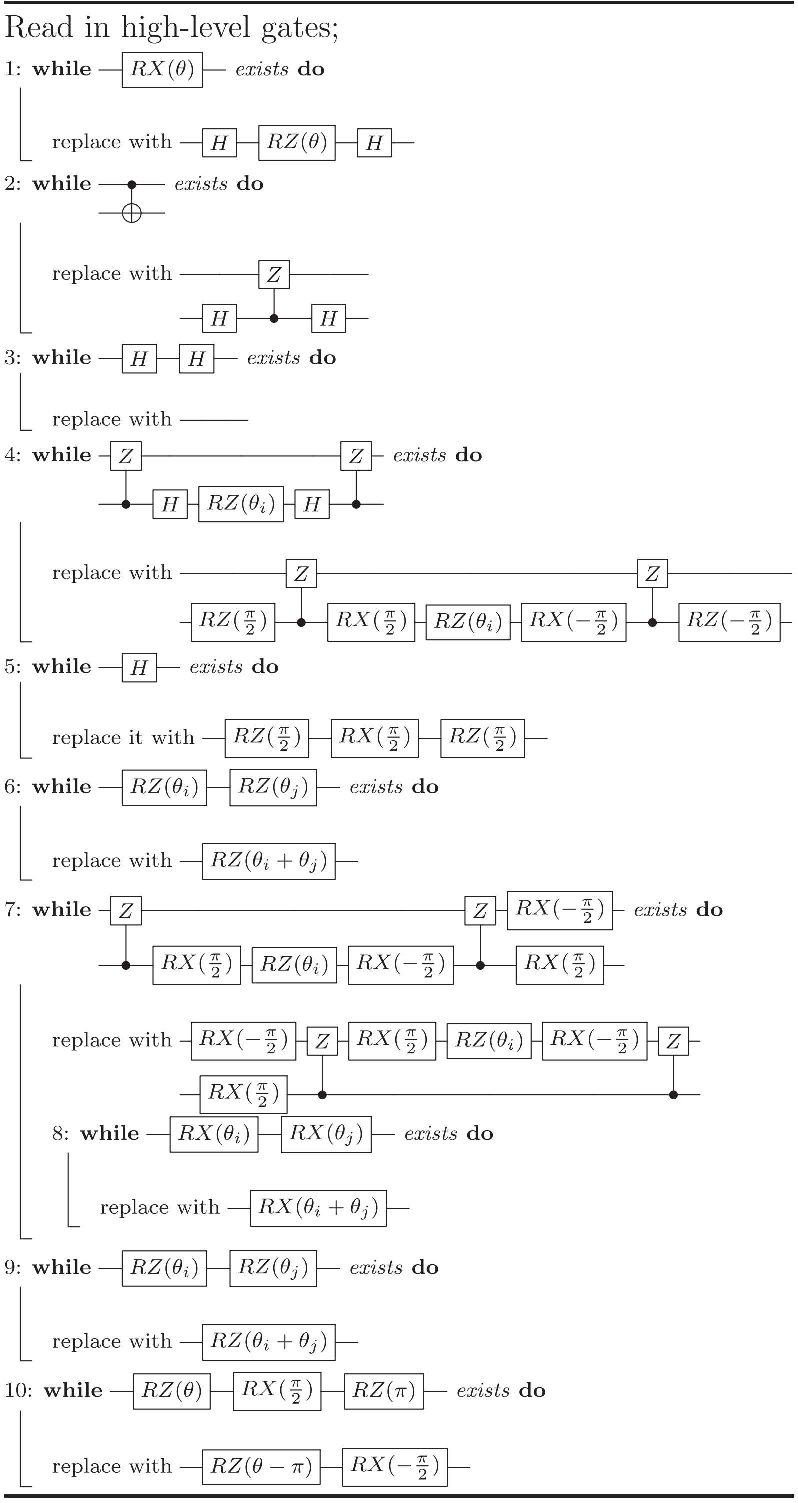
Of particular interest in algorithm 1 is the relationship between loop 7, which performs gate rearrangement, and its inner loop (loop 8), which performs gate reduction. We observe that before each subsequent iteration of the outer loop, the inner loop is applied until no more reductions are possible. In this way, gate reductions enabled by the inner loop are carried out fully before each subsequent step of gate rearrangement. This strategy is similar to the ones used in heuristic search under the name of *unit propagation*. In heuristic search, and particularly in solvers built for satisfiability (SAT) problems, unit propagation is very useful [43]. It reduces two disjuncts in the search space to a single disjunct whenever such a reduction is valid. Calling it in each step of the search serves as an efficient and effective lookahead technique [44].

To test the performance of this compiler, we constructed an algorithm to create a sequence of high-level circuits that simulate evolution through time of a system of spins under a TFIM Hamiltonian, as given in equation (1). Details of the algorithm can be found in the methods section. The high-level circuits were fed into the DS compiler as well as Rigetti's general-purpose compiler, and we compared performance based on the number of native gates in the compiled circuits and the wall-clock time for compilation for each time-step, for systems with varying numbers of qubits. Results can be found in figure 2. Figure 2(a) shows the absolute native gate count reduction when using the DS compiler over Rigetti's general-purpose compiler. As shown, the gate reduction increases with increasing time-step for all system sizes, which implies that the circuit-reduction benefit of the DS compiler scales well with simulation time-step. Furthermore, for each time-step, the number of reduced gates increases with additional qubits, indicating that the circuit-reduction benefit of the DS compiler also scales well with growing system size. Figure 2(b) shows the number of reduced gates as a percentage of the circuit size produced by Rigetti's general-purpose compiler. The percent reduction in native gate count asymptotes with growing time-step for all system sizes in a range around 30%. Figure 2(c) shows the percent reduction in wall-clock compilation time of the DS compiler compared to Rigetti's general-purpose compiler, which asymptotes to values ranging within 30%–60%.

2.4. IBM native gate set domain-specific compiler

IBM's native gate set is $\{RX(\frac{\pi}{2}), RZ(\theta), CNOT\}$, for arbitrary angle θ. Algorithm 2 shows pseudocode detailing the order in which a subset of the identities from table 1 are applied in the DS compiler for IBM's native gate set. Like the Rigetti DS compiler, the IBM DS compiler begins in loop 1 by converting $RX(\theta)$

Algorithm 1. Pseudocode for DS compilation of TFIM circuits into Rigetti's native gate set.


Read in high-level gates;
1: while $RX(\theta)$ exists do
replace with H $RZ(\theta)$ H
2: while exists do
replace with Z H H
3: while H H exists do
replace with
4: while Z Z H $RZ(\theta_i)$ H exists do
replace with Z Z $RZ(\frac{\pi}{2})$ $RX(\frac{\pi}{2})$ $RZ(\theta_i)$ $RX(-\frac{\pi}{2})$ $RZ(-\frac{\pi}{2})$
5: while H exists do
replace it with $RZ(\frac{\pi}{2})$ $RX(\frac{\pi}{2})$ $RZ(\frac{\pi}{2})$
6: while $RZ(\theta_i)$ $RZ(\theta_j)$ exists do
replace with $RZ(\theta_i+\theta_j)$
7: while Z Z $RX(-\frac{\pi}{2})$ $RX(\frac{\pi}{2})$ $RZ(\theta_i)$ $RX(-\frac{\pi}{2})$ $RX(\frac{\pi}{2})$ exists do
replace with $RX(-\frac{\pi}{2})$ Z $RX(\frac{\pi}{2})$ $RZ(\theta_i)$ $RX(-\frac{\pi}{2})$ Z $RX(\frac{\pi}{2})$
8: while $RX(\theta_i)$ $RX(\theta_j)$ exists do
replace with $RX(\theta_i+\theta_j)$
9: while $RZ(\theta_i)$ $RZ(\theta_j)$ exists do
replace with $RZ(\theta_i+\theta_j)$
10: while $RZ(\theta)$ $RX(\frac{\pi}{2})$ $RZ(\pi)$ exists do
replace with $RZ(\theta-\pi)$ $RX(-\frac{\pi}{2})$

gates to the sequence $(H, RZ(\theta), H)$, since RX gates on IBM cannot handle arbitrary angles. Unlike Rigetti, IBM's native gate set includes *CNOT* gates, and we use identity X in loop 2 to switch the control and target qubits of these gates. This trick enables two avenues for circuit reduction. First, it adds H gates next to existing ones, enabling cancellation of pairs of adjacent H's, taken care of in loop 3. Second, since RZ gates can commute across the control bit of a *CNOT* gate, but not the target bit, swapping *CNOT* orientation primes the circuit for further reduction by enabling more pairs RZ gates on opposite sides of control bits to be combined into one, as performed in loop 5.

Loops 4–7 all search for a motif created by the previous loop and transform this motif either from non-native gates to native gates (loop 4 and 6), or to a shorted gate sequence (loops 5 and 7). Finally, not shown in the pseudocode, is the removal of any trailing gates at the end of the circuit that only alter the phase of the system, and thus do not affect a measurement gate that immediately follows. Like in

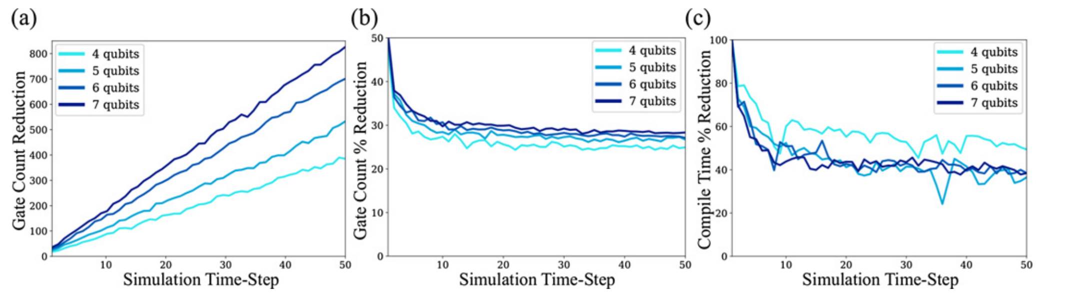
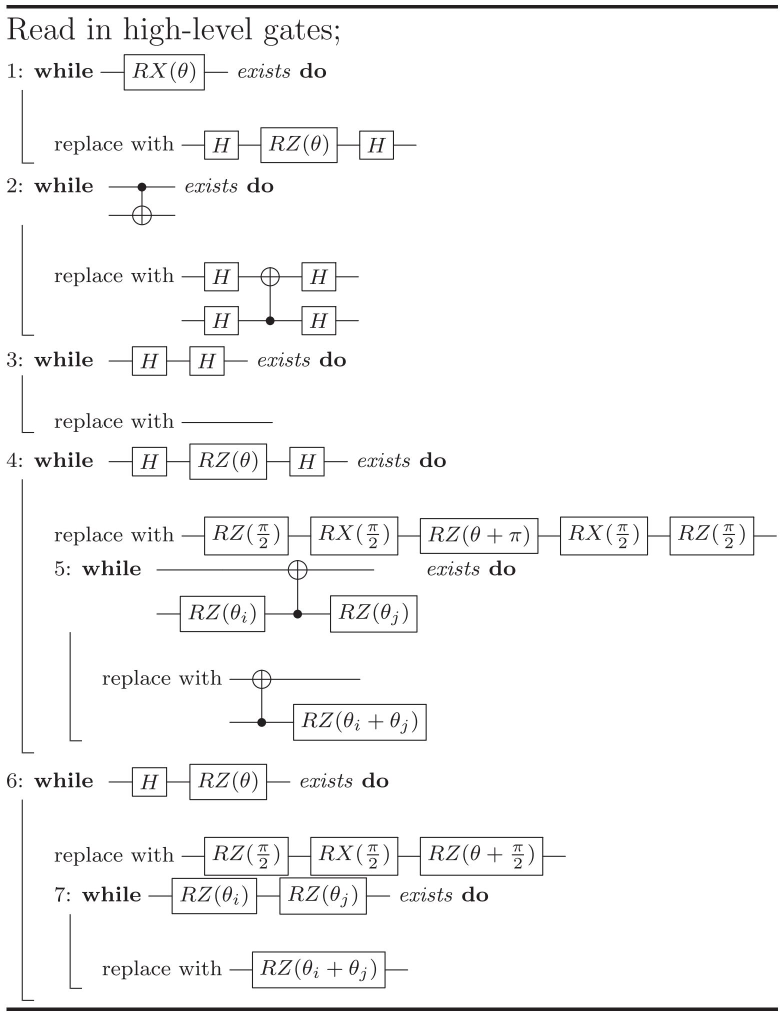


Figure 2. Performance comparison of the DS compiler to Rigetti's general-purpose compiler. (a) Absolute reduction in gate count using the DS compiler over Rigetti's general-purpose compiler, for varying system size. (b) Percent reduction in gate count using the DS compiler with respect to Rigetti-compiled circuit size, for varying system size. (c) Percent reduction in wall-clock compilation time using the DS compiler with respect to Rigetti-compiled circuit size, for varying system size.

Algorithm 2. Pseudocode for DS compilation of TFIM circuits into IBM's native gate set.



```
Read in high-level gates;
1: while —$\boxed{RX(\theta)}$— exists do
     replace with —$\boxed{H}$—$\boxed{RZ(\theta)}$—$\boxed{H}$—
2: while —$\bullet$— / —$\oplus$— exists do
     replace with —$\boxed{H}$—$\oplus$—$\boxed{H}$—
                  —$\boxed{H}$—$\bullet$—$\boxed{H}$—
3: while —$\boxed{H}$—$\boxed{H}$— exists do
     replace with ————
4: while —$\boxed{H}$—$\boxed{RZ(\theta)}$—$\boxed{H}$— exists do
     replace with —$\boxed{RZ(\frac{\pi}{2})}$—$\boxed{RX(\frac{\pi}{2})}$—$\boxed{RZ(\theta+\pi)}$—$\boxed{RX(\frac{\pi}{2})}$—$\boxed{RZ(\frac{\pi}{2})}$—
     5: while ——————$\oplus$———— exists do
              —$\boxed{RZ(\theta_i)}$—$\bullet$—$\boxed{RZ(\theta_j)}$
          replace with —$\oplus$————
                       —$\bullet$—$\boxed{RZ(\theta_i+\theta_j)}$
6: while —$\boxed{H}$—$\boxed{RZ(\theta)}$— exists do
     replace with —$\boxed{RZ(\frac{\pi}{2})}$—$\boxed{RX(\frac{\pi}{2})}$—$\boxed{RZ(\theta+\frac{\pi}{2})}$—
     7: while —$\boxed{RZ(\theta_i)}$—$\boxed{RZ(\theta_j)}$— exists do
          replace with —$\boxed{RZ(\theta_i+\theta_j)}$—
```

algorithm 1, a heuristic similar to the technique of unit propagation is applied in algorithm 2 in the third and fifth loops, both of which are outer loops of gate rearrangement with inner loops of gate reduction.

To test the performance of this compiler, we used the same simulation algorithm used for testing the Rigetti compilers to create a sequence of high-level TFIM circuits. Results can be found in figure 3. Figure 3(a) shows the absolute native gate count reduction when using the DS compiler over IBM's general-purpose compiler. Again, the gate reductions increase with increasing time-step for all system sizes, indicating that the circuit-reduction benefit of the DS compiler scales well with simulation time-step; furthermore, for each time-step, the number of reduced gates increases with additional qubits, indicating that the circuit-reduction benefit of the DS compiler also scales well with growing system size. Figure 3(b) shows the number of reduced gates as a percentage of the circuit size produced by IBM's general-purpose compiler. The percent reduction in gate count asymptotes with growing time-step for all system sizes in a

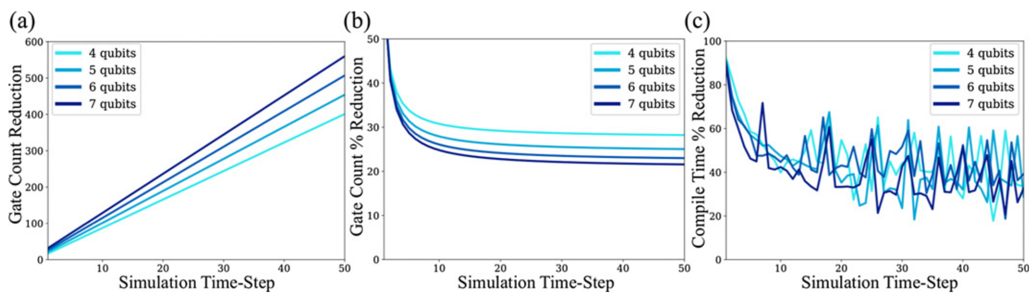


Figure 3. Performance comparison of the DS compiler to IBM's general-purpose compiler. (a) Absolute reduction in gate count using the DS compiler over IBM's general-purpose compiler, for varying system size. (b) Percent reduction in gate count using the DS compiler with respect to IBM-compiled circuit size, for varying system size. (c) Percent reduction in wall-clock compilation time using the DS compiler with respect to IBM-compiled circuit size, for varying system size.

range around 25%. Figure 3(c) shows the percent reduction in wall-clock compilation time of the DS compiler compared to IBM's general-purpose compiler, which asymptotes to values ranging within 20%–60%.

3. Discussion

While sufficient quantum hardware exists for demonstrating quantum supremacy with random circuits [45], a major bottleneck to demonstrating physical quantum supremacy is the development of new quantum circuit compilers for optimizing such circuits. Though numerous software packages with general-purpose compilers have recently been developed [46–49], the development of DS compilers, optimized for certain classes of circuits and for specific native gate sets, may be a necessary intermediate technology for the success of early NISQ-era simulations. The two DS compilers we developed for currently available NISQ computers, using heuristics derived from domain knowledge of the structure of TFIM circuits, target native gate sets, and artificial intelligence (AI) techniques, demonstrated 25%–30% reductions in circuit size compared to the general-purpose compilers. This significant reduction in circuit size may soon allow relevant dynamic simulations of quantum materials on near-future NISQ computers, providing new insights from results not achievable with classical computers.

It should be noted that our DS compilers are extensible to future quantum processors provided that the native gate set of the new quantum machine matches the target native gate set of the DS compiler. While the native gate sets of Rigetti and IBM are slightly different (hence the need to develop two separate DS compilers), they both represent what is currently considered a standard universal gate set. Therefore, it is likely that native gate sets of future quantum machines will at least contain these gate sets as a subset. If this is true, the DS compilers can easily be extended to future quantum computers.

In future work, we would like to explore the use of other optimization tools developed in AI. Several tools for solving SAT, constraint satisfaction, and weighted constraint satisfaction problems are currently being used to solve many real-world problems after proper reformulation. These tools encapsulate many decades of AI research for optimization and work well in practice for NP-hard problems. Some attempts have already been made for minimizing the number of gates in digital logic circuits by invoking powerful SAT solvers [50]. In addition TFIM circuits, DS compilers can also be created for other classes of quantum circuits for dynamic simulation [10, 11, 51].

4. Methods

4.1. Decomposition of time-evolution operator

In order to map the time-ordered exponential unitary operator into a set of one- and two-qubit gates, two approximations must be applied [52]. First, the time-dependence of $H(t)$ must be ignored on time scales smaller than some chosen, minimal time-step Δt. The Hamiltonian can then be approximated as a piece-wise constant function that takes the constant value $H((j+\frac{1}{2})\Delta t)$ during the time interval $[j\Delta t,(j+1)\Delta t]$, where j is some integer, resulting in the following decomposition: $U(n\Delta t)\approx\prod_{j=0}^{n-1}\mathrm{e}^{-\mathrm{i}H((j+\frac{1}{2})\Delta t)\Delta t}$. Second, each matrix exponential in this product must be approximated with the Trotter decomposition [42]. To perform the Trotter decomposition, the Hamiltonian is divided into components, each of which is efficiently diagonalizable. For the TFIM Hamiltonian, this can be accomplished with the following decomposition: $H(t)=H_x(t)+H_z$ where $H_x(t)=-B(t)\sum_{i=1}^{N}\sigma_i^x$ and

Table 2. IBM basis and native gate equivalents.

Basis gate	Native gate equivalent
$U1(\lambda)$	$RZ(\lambda)$
$U2(\phi,\lambda)$	$RZ(\phi+\frac{\pi}{2})$ — $RX(\frac{\pi}{2})$ — $RZ(\lambda-\frac{\pi}{2})$
$U3(\theta,\phi,\lambda)$	$RZ(\phi+3\pi)$ — $RX(\frac{\pi}{2})$ — $RZ(\theta+\pi)$ — $RX(\frac{\pi}{2})$ — $RZ(\lambda)$

$H_z = -J_z\sum_{i=1}^{N-1}\sigma_i^z\sigma_{i+1}^z$. Thus, the time evolution operator is finally approximated as: $U(n\Delta t) = \prod_{j=0}^{n-1} \mathrm{e}^{-\mathrm{i}H_x((j+1/2)\Delta t)\Delta t}\ \mathrm{e}^{\mathrm{i}H_z\Delta t} + \mathcal{O}(\Delta t)$, as was given in equation (2).

4.2. Algorithm for circuit generation

An algorithm was written to generate circuits that simulate the time evolution of a system of spins under the TFIM Hamiltonian given in equation (1). For a simulation to time $n\Delta t$, n different circuits are created. A physically reasonable value for material systems was chosen for J_z, and a sinusoidal function was chosen for $B(t)$, with amplitude and frequency also assigned physically reasonable values. For a given circuit, the algorithm proceeds by appending alternating sets of gates that each propagate the system forward by a time-step Δt according to the two exponentials given in the product in equation (2). Note that the parameters in the set of gates carrying out the exponential $\mathrm{e}^{-\mathrm{i}H_x((j+\frac{1}{2})\Delta t)\Delta t}$ will change for each subsequent propagation by Δt because our Hamiltonian in equation (1) is time-dependent. The set of gates representing the exponential $\mathrm{e}^{-\mathrm{i}H_z\Delta t}$ is identical for each application. Once gate sets representing the two exponentials have been alternately applied n times each, the circuit simulating to time $n\Delta t$ is complete and the algorithm begins building the circuit for simulation to the subsequent time-step. Circuits for all time-steps are appended to a list, which serves as input to the various compilers for conversion to native gates and compression.

4.3. Development and performance analysis of compilers

Written in the Python programming language, both compilers were developed to reduce circuit size for circuits designed to simulate dynamic evolution of TFIM systems. The compilers take as input a list of circuits comprised of high-level gates represented as a serial list of gates and the qubits upon which they act. The compilers output a list of circuits of reduced size comprised solely of native gates. Since the native gate sets for Rigetti and IBM differ, separate compilers were developed for each platform.

Two points should be mentioned about our compilers. First, note that the compilers take advantage of the fact that for these dynamic simulations, only measurement probabilities of different states need to be conserved when applying transformation and compression identities. This means that the overall global phase of the system, which has no observable effect on measurement probabilities, need not be conserved, allowing for a larger number of useful circuit identities to be used. Second, we note that the compilers are deterministic in their approach; a given high-level circuit will be compiled to the same executable circuit on each run. This is as opposed to other available compilers which return different compiled circuits, of varying size, for the same high-level circuit on different runs.

Both Rigetti's general-purpose compiler and our DS compiler compile circuits into Rigetti's native gate set. To compare performance of the two compilers in terms of circuit size, we therefore counted the number of native gates in the circuits produced for each time-step by each compiler. One complication is that Rigetti's compiler is not deterministic, meaning that running the same input circuit through the compiler two different times can result in two different output circuits. This is the root of the fluctuation seen in figures 2(a) and (b) but not in figures 3(a) and (b). To minimize these fluctuations, we ran Rigetti's general-purpose compiler three times and averaged the number of native gates in the circuits over the three runs. These averaged native gate counts for each time step were used for comparison with the native gate counts from our deterministic DS compiler. For comparing compiler run times, wall-clock times were recorded for three separate runs each of Rigetti's compiler and our DS compiler, and the average over these runs was used for comparison.

Performance comparison in terms of native gate count was slightly more subtle for the IBM compilers. IBM's general-purpose compiler compiles circuits into a set of basis gates, comprised of $U1(\lambda)$, $U2(\phi,\lambda)$, $U3(\theta,\phi,\lambda)$, $CNOT$. The $U1$, $U2$, $U3$ basis gates can be written in terms of native gates as shown in table 2. Even though its general-purpose compiler compiles to this basis gate set, the IBM quantum computer nonetheless only physically executes gates from the native gate set. For this reason, we chose to implement our DS compiler to output circuits comprised of native gates.

As shown in table 2, different basis gates are comprised of different numbers of native gates. Therefore, to fairly compare the circuit sizes produced by IBM's general-purpose compiler to those output by our DS compiler, we counted all *U*1 and *CNOT* gates in each IBM-compiled circuit as one native gate each, *U*2 gates as three native gates, and *U*3 gates as five native gates. All gate count comparisons for the IBM compilers, therefore, were in terms of native gate count. This also makes the gate count performance of the IBM compilers easier to compare with the gate count performance of the Rigetti compilers.

Acknowledgments

This work was supported as part of the Computational Materials Sciences Program funded by the US Department of Energy, Office of Science, Basic Energy Sciences, under Award No. DE-SC0014607.

Code availability

Code for the DS compilers can be found at https://github.com/USCCACS/DS_Compilers_Full_Stack_Package. For user convenience they have been packaged into a full-stack quantum simulation software package that allows the user to create, compile, run, and post-process circuits on the Rigetti and IBM quantum backends. See the GitHub repository for tutorials.

ORCID iDs

Lindsay Bassman https://orcid.org/0000-0003-3542-1553
Connor Powers https://orcid.org/0000-0003-1848-3525
Thomas Linker https://orcid.org/0000-0002-0504-4876
Aiichiro Nakano https://orcid.org/0000-0003-3228-3896
Priya Vashishta https://orcid.org/0000-0003-4683-429X

References

[1] Feynman R P 1982 Simulating physics with computers *Int. J. Theor. Phys.* **21** 467
[2] Preskill J 2018 Quantum computing in the NISQ era and beyond *Quantum* **2** 79
[3] Lanyon B P *et al* 2010 Towards quantum chemistry on a quantum computer *Nat. Chem.* **2** 106
[4] Kandala A, Mezzacapo A, Temme K, Takita M, Brink M, Chow J M and Gambetta J M 2017 Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets *Nature* **549** 242–6
[5] Cervera-Lierta A 2018 Exact Ising model simulation on a quantum computer *Quantum* **2** 114
[6] Colless J I *et al* 2018 Computation of molecular spectra on a quantum processor with an error-resilient algorithm *Phys. Rev.* X **8** 011021
[7] Hempel C *et al* 2018 Quantum chemistry calculations on a trapped-ion quantum simulator *Phys. Rev.* X **8** 031022
[8] Lamm H and Lawrence S 2018 Simulation of nonequilibrium dynamics on a quantum computer *Phys. Rev. Lett.* **121** 170501
[9] Wiebe N, Berry D W, Høyer P and Sanders B C 2011 Simulating quantum dynamics on a quantum computer *J. Phys. A: Math. Theor.* **44** 445308
[10] Martinez E A *et al* 2016 Real-time dynamics of lattice gauge theories with a few-qubit quantum computer *Nature* **534** 516–9
[11] Smith A, Kim M, Pollmann F and Knolle J 2019 Simulating quantum many-body dynamics on a current digital quantum computer *npj Quantum Inf.* **5** 1–13
[12] Bassman L *et al* 2020 Towards simulation of the dynamics of materials on quantum computers *Phys. Rev.* B **101** 184305
[13] Chong F T, Franklin D and Martonosi M 2017 Programming languages and compiler design for realistic quantum hardware *Nature* **549** 180–7
[14] Tucci R R 1999 A rudimentary quantum compiler (2cnd ed.) (arXiv:quant-ph/9902062)
[15] Vartiainen J J, Möttönen M and Salomaa M M 2004 Efficient decomposition of quantum gates *Phys. Rev. Lett.* **92** 177902
[16] Möttönen M, Vartiainen J J, Bergholm V and Salomaa M M 2004 Quantum circuits for general multiqubit gates *Phys. Rev. Lett.* **93** 130502
[17] De Vos A and De Baerdemacker S 2016 Block-*zxz* synthesis of an arbitrary quantum circuit *Phys. Rev.* A **94** 052317
[18] Iten R, Colbeck R, Kukuljan I, Home J and Christandl M 2016 Quantum circuits for isometries *Phys. Rev.* A **93** 032318
[19] Martinez E A, Monz T, Nigg D, Schindler P and Blatt R 2016 Compiling quantum algorithms for architectures with multi-qubit gates *New J. Phys.* **18** 063029
[20] Iten R *et al* 2019 Introduction to universalqcompiler (arXiv:1904.01072)
[21] Khatri S, LaRose R, Poremba A, Cincio L, Sornborger A T and Coles P J 2019 Quantum-assisted quantum compiling *Quantum* **3** 140
[22] Younis E, Sen K, Yelick K and Iancu C 2020 Qfast: quantum synthesis using a hierarchical continuous circuit space (arXiv:2003.04462)
[23] Shende V V, Markov I L and Bullock S S 2004 Minimal universal two-qubit controlled-not-based circuits *Phys. Rev.* A **69** 062321
[24] Bullock S S and Markov I L 2003 An arbitrary twoqubit computation in 23 elementary gates or less *Proc. of the 40th Annual Design Automation Conf.* pp 324–9
[25] Vidal G and Dawson C M 2004 Universal quantum circuit for two-qubit transformations with three controlled-not gates *Phys. Rev.* A **69** 010301

[26] Vatan F and Williams C 2004 Optimal quantum circuits for general two-qubit gates *Phys. Rev.* A **69** 032315
[27] Vatan F and Williams C P 2004 Realization of a general three-qubit quantum gate (arXiv:quant-ph/0401178)
[28] Murali P, Baker J M, Javadi-Abhari A, Chong F T and Martonosi M 2019 Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers *Proc. of the 24th Int. Conf. on Architectural Support for Programming Languages and Operating Systems* pp 1015–29
[29] Cincio L, Rudinger K, Sarovar M and Coles P J 2020 Machine learning of noise-resilient quantum circuits (arXiv:2007.01210)
[30] Herr D, Nori F and Devitt S J 2017 Optimization of lattice surgery is np-hard *npj Quantum Inf.* **3** 35
[31] Botea A, Kishimoto A and Marinescu R 2018 On the complexity of quantum circuit compilation *11th Annual Symp. on Combinatorial Search*
[32] Lo W S and Pelcovits R A 1990 Ising model in a time-dependent magnetic field *Phys. Rev.* A **42** 7471
[33] Chakrabarti B K, Dutta A and Sen P 2008 *Quantum Ising Phases and Transitions in Transverse Ising Models* vol 41 (Berlin: Springer)
[34] Suzuki S, Inoue J-i and Chakrabarti B K 2012 *Quantum Ising Phases and Transitions in Transverse Ising Models* vol 862 (Berlin: Springer)
[35] Blinc R, Žekš B, Sampaio J F, Pires A S T and Barreto F C S 1979 Ising model in a transverse tunneling field and proton-lattice interaction in H-bonded ferroelectrics *Phys. Rev.* B **20** 1991
[36] Wu W, Ellman B, Rosenbaum T, Aeppli G and Reich D 1991 From classical to quantum glass *Phys. Rev. Lett.* **67** 2076
[37] Tomé T and de Oliveira M J 1990 Dynamic phase transition in the kinetic Ising model under a time-dependent oscillating field *Phys. Rev.* A **41** 4251
[38] Acharyya M and Chakrabarti B K 1995 Response of Ising systems to oscillating and pulsed fields: hysteresis, ac, and pulse susceptibility *Phys. Rev.* B **52** 6550
[39] Acharyya M 1998 Nonequilibrium phase transition in the kinetic Ising model: is the transition point the maximum lossy point? *Phys. Rev.* E **58** 179
[40] Sides S W, Rikvold P A and Novotny M A 1998 Kinetic Ising model in an oscillating field: finite-size scaling at the dynamic phase transition *Phys. Rev. Lett.* **81** 834
[41] Bassman L, Gulania S and Powers C 2020 Full stack quantum simulation package for domain-specific compilers https://github.com/USCCACS/DS_Compilers_Full_Stack_Package
[42] Trotter H F 1959 On the product of semi-groups of operators *Proc. Am. Math. Soc.* **10** 545
[43] Li C M and Anbulagan A 1997 Heuristics based on unit propagation for satisfiability problems *Proc. of the 15th Int. Joint Conf. on Artifical Intelligence* vol 1 pp 366–71
[44] Edelkamp S and Schroedl S 2011 *Heuristic Search: Theory and Applications* (Amsterdam: Elsevier)
[45] Arute F *et al* 2019 Quantum supremacy using a programmable superconducting processor *Nature* **574** 505–10
[46] Wecker D L 2015 *Proc. of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis* (New York: ACM)
[47] Smith R S, Curtis M J and Zeng W J 2016 A practical quantum instruction set architecture (arXiv:1608.03355)
[48] Paler A, Polian I, Nemoto K and Devitt S J 2017 Fault-tolerant, high-level quantum circuits: form, compilation and description *Quantum Sci. Technol.* **2** 025003
[49] Häner T, Steiger D S, Svore K and Troyer M 2018 A software methodology for compiling quantum programs *Quantum Sci. Technol.* **3** 020501
[50] Sapra S, Theobald M and Clarke E 2003 Sat-based algorithms for logic minimization *Proc. 21st Int. Conf. on Computer Design* (Piscataway, NJ: IEEE) pp 510–7
[51] Zhukov A, Remizov S, Pogosov W and Lozovik Y E 2018 Algorithmic simulation of far-from-equilibrium dynamics using quantum computer *Quantum Inf. Process.* **17** 223
[52] Poulin D, Qarry A, Somma R and Verstraete F 2011 Quantum simulation of time-dependent Hamiltonians and the convenient illusion of Hilbert space *Phys. Rev. Lett.* **106** 170501