The Fermionic Quantum Emulator

Nicholas C. Rubin[1], Klaas Gunst[2], Alec White[2], Leon Freitag[2], Kyle Throssell[2], Garnet Kin-Lic Chan[3], Ryan Babbush[1], and Toru Shiozaki[2]

[1]Google Quantum AI, Mountain View, CA, 94043
[2]Quantum Simulation Technologies, Inc., Cambridge, MA 02139
[3]Division of Chemistry and Chemical Engineering, California Institute of Technology, Pasadena CA 91125
10/15/2021

The fermionic quantum emulator (FQE) is a collection of protocols for emulating quantum dynamics of fermions efficiently taking advantage of common symmetries present in chemical, materials, and condensed-matter systems. The library is fully integrated with the OpenFermion software package and serves as the simulation backend. The FQE reduces memory footprint by exploiting number and spin symmetry along with custom evolution routines for sparse and dense Hamiltonians, allowing us to study significantly larger quantum circuits at modest computational cost when compared against qubit state vector simulators. This release paper outlines the technical details of the simulation methods and key advantages.

1 Introduction

High accuracy simulation of fermionic systems is an important and challenging problem and is a major motivation behind current attempts to develop quantum computers [1, 4, 12, 31]. There has been significant experimental progress towards realizing the simulation of fermionic systems on current quantum devices [3, 19]. As these experiments scale in size there is a growing need to understand the possibilities for quantum advantage, with one approach being to characterize the classical emulation complexity of the corresponding quantum circuits. In addition, the efficient emulation of near-term fermionic simulation experiments is crucial for experiment design, algorithm design, and testing. In this work, we describe an implementation of protocols to efficiently emulate quantum circuits describing time evolution under fermionic generators. We name the library that implements these protocols the Fermionic Quantum Emulator (FQE).[1]

There have been many developments in quantum circuit simulation and emulation. Broadly these advancements can be classified into algorithmic improvements [6, 7, 14, 14, 16–18, 26] and computational implementation improvements [24, 37, 39, 41]. However, despite this progress, there remains potential to improve the emulation of circuits relevant

Nicholas C. Rubin: Corresponding author: nickrubin@google.com
Garnet Kin-Lic Chan: Corresponding author: garnetc@caltech.edu
Ryan Babbush: Corresponding author: babbush@google.com
Toru Shiozaki: Corresponding author: shiozaki@qsimulate.com

[1]We make a distinction between simulating quantum circuits directly and emulating the quantum circuits when the quantum circuit corresponds to dynamics of fermions.

to fermionic simulation. For example, in general circuit emulation it is necessary to include additional circuit elements to handle the fermion encoding, but these can be eliminated and the fermionic sign accounted for implicitly in a specialized fermionic emulator. In addition, many fermionic systems have symmetries that can be used to reduce the resource requirements of the classical emulation. For example, molecular and material problems are often described by Hamiltonians that commute with a variety of global symmetry operators, such as the total particle number, total spin, time-reversal, and point-group and crystallographic symmetries. Though there are many open source software packages to carry out quantum chemistry calculations of fermionic systems using these symmetries [35, 38, 40], these are not designed to support computations using the quantum circuit model. Similarly, existing quantum circuit simulators and the corresponding computational techniques used within them are not naturally suited to efficiently working within symmetry reduced Hilbert spaces.

The FQE is a simulator of fermions represented in second quantization. It corresponds to a statevector simulator in that the wavefunction is explicitly stored. In the first release of the emulator, number and spin (S_z) symmetry are used to minimize the wavefunction memory footprint by using a generalization of the Knowles–Handy [21] determinant based configuration interaction wavefunction organization scheme. Time evolution of the state under a fermionic Hamiltonian is implemented in two ways: (1) via a custom routine similar to the cosine-sine construction for nilpotent operators applicable to sparse fermion generators, and (2) via a series expansion of the propagator for dense Hamiltonians taking advantage of efficient intermediate data structures. The library completely integrates with the fermion quantum simulation library OpenFermion [27] and has built in functions to convert wavefunction objects in FQE to the format used in OpenFermion as well as for the general purpose circuit simulator Cirq [9]. For example, the time evolution of a wavefunction can be generated using OpenFermion's `FermionOperator` objects or Hamiltonians defined through FQE utility functions. The FQE is an open source library distributed under the Apache 2.0 license [33]. The library is implemented in Python to facilitate code extension and reuse. However, some hot spots have been addressed with high-performance kernels written in the C language. These C extensions can be enabled or disabled by the user. This allows the library to function as either a high-performance library for computationally demanding applications or as a pure-Python library implemented with understandable and easily modifiable expressions.

This work describes the key technical aspects of the library, demonstrates how it can be used in quantum algorithm development, and makes single-thread and multi-threaded timing comparisons for key circuit primitives against Cirq [9] and the highly performant general purpose quantum circuit simulator Qsim [32]. We close with a perspective on the development of the library and future directions.

2 Wavefunctions and Hilbert space organization

When simulating spin-$\frac{1}{2}$ fermions in second quantization, the many-particle fermionic Hilbert space (sometimes called the Fock space) generated by a basis of L single-fermion states (termed orbitals) $\{|\phi_1\rangle, |\phi_2\rangle, \ldots |\phi_L\rangle\}$, is spanned by basis states (termed determinants) labelled by L-digit binary strings $\{n_1 n_2 \ldots n_L\}$ where $n \in \{0, 1\}$, and n_i is the occupancy of orbital i. It is evident that the fermionic Hilbert space is isomorphic to the Hilbert space of L qubits, with the qubit computational basis corresponding to the fermion determinant basis. This allows one to use the traditional state vector representation of qubits to encode fermionic states.

For many molecular and materials systems, the electronic Hamiltonian is an operator that commutes with various global symmetry operators. These symmetries thus provide useful quantum numbers to label sectors of Hilbert space. Because the Hamiltonian does not mix sectors, many simulations can be performed either in a single sector, or in a collection of independent sectors. A simple example is simulating a fixed number of electrons n. In this case, the Hilbert space contains only $\binom{L}{n}$ states. Because the determinants are eigenstates of the particle number operator, the spanning basis can be chosen to be determinants with total occupancy $\sum_i n_i = n$. If we further assume the orbitals are eigenstates of S_z (i.e. spin-orbitals), then the determinants are also eigenstates of S_z. Then a Hilbert space sector labelled by n and S_z is spanned by determinants only with the given n and S_z.

The FQE uses such a decomposition into symmetry sectors to store the wavefunction compactly. A user specifies symmetry sectors of fixed particle number and given S_z. The total wavefunction is then stored in the direct sum of these sectors. Considering all possible particle sectors and S_z sectors corresponds to working in the full many-particle fermionic Hilbert space, and thus storing the full set of 2^L qubit amplitudes.

An efficient way to represent wavefunctions for a fixed particle number and S_z sector was first introduced by Siegbahn [36] and refined by Knowles and Handy [21] in the context of the exact diagonalization of chemistry Hamiltonians. The binary string encoding a determinant is separated into a string for the spin-up (α) spin-orbitals and the spin-down (β) spin-orbitals. Commonly these two binary strings (integers) are referred to as the α- and β-string respectively. A further simplification arises by assuming that the α and β spin-orbitals share a common set of spatial functions, known as spatial orbitals. Given M spatial orbitals in total, the total number of spin-orbitals is thus $L = 2M$.

The following code example initializes a state with four electrons in four spatial orbitals over a superposition of all possible S_z values.

```
wfn = fqe.Wavefunction([[4, 4, 4], [4, 2, 4], [4, 0, 4], [4, -2, 4], [4, -4, 4]])
```

Each element of the input list labels a Hilbert space sector as the triple (n-electrons, S_z, M-spatial-orbitals). The wavefunction in each sector is stored as a matrix of size $\binom{M}{n_\alpha} \times \binom{M}{n_\beta}$ where n_α and n_β are the total number of α and β electrons. Each row of the matrix corresponds to a specific α-string, and each column to a β-string. The mapping between the integer values of the α- and β-strings to the row and column indices is somewhat arbitrary; we use the lexical ordering proposed by Knowles and Handy [20]. By storing the wavefunction coefficients as matrices we can leverage vectorized multiplications when performing updates or contractions on the wavefunction coefficients.

The memory savings from simulating in specific symmetry sectors is substantial when compared to traditional state-vector simulators which use the full 2^L qubit space. In Figure 1 we show the relative size of half filling ($n = L/2$) and quarter filling ($n = L/4$) subspaces with $S_z = 0$ along with the wavefunction memory footprint in gigabytes.

To make the FQE interoperable with other simulation tools we provide a number of utilities for transforming the FQE `Wavefunction` representation. We also provide human readable printing, saving wavefunctions as binary data in `numpy`'s `.npy` format, various wavefunction initialization routines for random, Hartree-Fock, or user defined states, and conversion functions to OpenFermion and Cirq wavefunction representations. The printing functionality is demonstrated in the code snippet below.

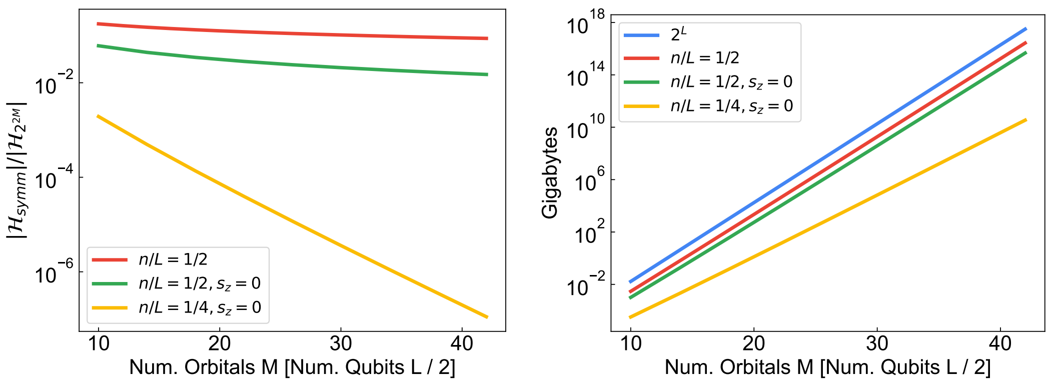

Figure 1: *left:* Relative sizes of Hilbert space sectors at half and quarter filling compared against full qubit Hilbert space size. *right:* Gigabytes required to represent spaces of $S_z = 0$, half and quarter filling, compared against the full qubit Hilbert space and half filling with no S_z symmetry restriction.

```
import fqe

# two sectors N=2 Sz=[0,-2] on 4 orbitals
wf = fqe.Wavefunction([[2, 0, 4], [2, -2, 4]])
wf.set_wf(strategy='random')
wf.print_wfn()
```

```
Sector N = 2 : S_z = -2
a'0000'b'0011' (0.10858096183738326-0.34968321927438234j)
a'0000'b'0101' (-0.09372404351124913-0.19775900492450152j)
... elided output
Sector N = 2 : S_z = 0
a'0001'b'0001' (0.09251611820297777+0.1450093827514917j)
a'0001'b'0010' (0.15516111875500693+0.17752453798314619j)
a'0001'b'0100' (0.12499026886480313+0.028546904174259435j)
... elided output
```

The conversion to OpenFermion is handled by converting each coefficient in the `Wavefunction` object with its associated α, β-string to an ordered string of creation operators acting on a vacuum state, with the string represented by an OpenFermion `FermionOperator` object. This allows the user to leverage the normal ordering routines in OpenFermion. By using the `FermionOperator` intermediate we can also directly map the string of normal ordered operators acting on a vacuum to a state vector for integration with Cirq simulations.

```
of_ops = fqe.fqe_to_fermion_op(wf)

cirq_wf = fqe.to_cirq(wf)
new_fqe_wf = fqe.from_cirq(cirq_wf, thresh=1.0E-12)  # same as original wf
```

3 Unitary evolution

3.1 Evolution of Sparse Hamiltonians

The FQE can evolve a state by any fermionic excitation Hamiltonian. A fermionic excitation Hamiltonian is of the form

$$\hat{H}_{\text{excite}} = \epsilon \left(\hat{g} + \hat{g}^\dagger \right) \tag{1}$$

$$\hat{g} = g \prod_{p=1}^{N_{\text{op}}} \hat{a}^\dagger_{i_p \sigma_p} \prod_{q=1}^{N_{\text{op}}} \hat{a}_{j_q \rho_q} \tag{2}$$

where g is a complex number and $\hat{g}$ is a single product of an arbitrary number (N_{op}) of creation and annihilation operators that create/annihilate orbitals with spatial orbital indices labelled by i, j and S_z indices labelled by σ, ρ. We refer to this Hamiltonian as the excitation Hamiltonian because it only involves a single "excitation" term $\hat{g}$ and its Hermitian conjugate. To specify $\hat{g}$ the FQE can digest a `FermionOperator` from OpenFermion. If $\hat{H}_{\text{excite}}$ is diagonal, i.e. a polynomial of number operators (such as $\hat{a}^{\dagger}_{1\alpha}\hat{a}^{\dagger}_{2\alpha}\hat{a}_{1\alpha}\hat{a}_{2\alpha} = -\hat{n}_{1\alpha}\hat{n}_{2\alpha}$, with $\hat{n}_{1\alpha} = \hat{a}^{\dagger}_{1\alpha}a_{1\alpha}$ and similarly for $\hat{n}_{2\alpha}$) evolution of the wave function can be performed using the techniques described later in Sec. 3.3.1. When $\hat{H}_{\text{excite}}$ is not diagonal and contains no repeated indices (such as $\hat{a}^{\dagger}_{4\alpha}\hat{a}^{\dagger}_{2\beta}\hat{a}_{1\alpha}\hat{a}_{3\beta}$), evolution of the wavefunction is accomplished by

$$\begin{aligned} e^{-i(\hat{g}+\hat{g}^{\dagger})\epsilon} = 1 &+ \left[\cos(\epsilon|g|) - 1 - i\hat{g}^{\dagger}\frac{\sin(\epsilon|g|)}{|g|}\right]\hat{\mathcal{P}}_{\hat{g}\hat{g}^{\dagger}} \\ &+ \left[\cos(\epsilon|g|) - 1 - i\hat{g}\frac{\sin(\epsilon|g|)}{|g|}\right]\hat{\mathcal{P}}_{\hat{g}^{\dagger}\hat{g}} \end{aligned} \tag{3}$$

where we define $\hat{\mathcal{P}}_x$ as the projector onto the basis of determinants that are not annihilated by $\hat{x}$. For other recent applications of this relation in quantum chemistry see e.g. Refs. [13], [11], and Ref. [8] in the context of quantum and quantum-inspired algorithms. A derivation of Eq. (3) is presented in Appendix A. When $\hat{H}_{\text{excite}}$ is not diagonal but has several repeated indices as well (such as $\hat{a}^{\dagger}_{1\alpha}\hat{a}^{\dagger}_{2\beta}\hat{a}_{1\alpha}\hat{a}_{3\beta} = -\hat{n}_{1\alpha}\hat{a}^{\dagger}_{2\beta}\hat{a}_{3\beta}$), a hybrid approach is used with the only additional complication being the fermion parity evaluation that arises from operator reordering.

Time evolution of a FQE wavefunction is implemented as a method of the `Wavefunction` object and can be easily accessed through the `Wavefunction` object interface. For example, the code snippet

```
from openfermion import FermionOperator
import fqe

wf = fqe.Wavefunction([[4, 0, 6]])
wf.set_wfn(strategy='random')
i, j, theta = 0, 1, 2 / 3
op = (FermionOperator(((2 * i, 1), (2 * j, 0)), coefficient=-1j * theta) +
      FermionOperator(((2 * j, 1), (2 * i, 0)), coefficient=1j * theta))
new_wfn = fqe_wfn.time_evolve(1.0, op)
```

performs evolution of a random state ψ_0 with a one-particle excitation Hamiltonian acting on the α-spin sector

$$|\psi_f\rangle = e^{-i\theta(\hat{a}^{\dagger}_{i\alpha}\hat{a}_{j\alpha}+\hat{a}^{\dagger}_{j\alpha}\hat{a}_{i\alpha})}|\psi_0\rangle. \tag{4}$$

The `Wavefunction` object correctly evolves this type of sparse Hamiltonian so long as the Hamiltonian commutes with the underlying wavefunction symmetries. However, the user is responsible for providing a Hamiltonian of this form, and the FQE does not correctly handle the evolution of Hamiltonians which break the specified wavefunction symmetry.

Evolving under a single fermionic excitation Hamiltonian is the basis for arbitrary fermionic quantum circuit evolution and can be used to simulate arbitrary fermionic Hamiltonian evolution. Because of the isomorphism between fermion and qubit spaces, general qubit Hamiltonians, so long as they are symmetry preserving in the same sense, can be simulated with little overhead within the same scheme either by modifying the code to ignore signs arising from fermion parity or by explicitly inserting swap operations. This raises the possibility to simulate large quantum circuits associated with non-fermionic Hamiltonian evolution, that benefit from the memory saving associated with simulating the fixed "particle" or "spin" sectors of the corresponding qubit Hilbert space.

Though evolution by single excitation Hamiltonian provides a primitive to implement many fermionic circuit simulations, the FQE also implements efficient time-evolution routines for Hamiltonians with special forms and for generic quantum chemical Hamiltonians. These will be discussed in more detail in the following sections.

3.2 Evolution of Dense Hamiltonians

The FQE provides special routines to evolve sums of excitation Hamiltonians through series expansions. We begin by discussing dense Hamiltonians, which for the purposes of this work we define as a weighted sum over all possible excitation Hamiltonians with the same index structure. A dense two-particle Hamiltonian, for which the FQE implements special routines, would take the form

$$\hat{H} = \sum_{ijkl} \sum_{\sigma\sigma'\rho\rho'} V_{i\sigma,j\rho,k\sigma',l\rho'} \hat{a}^\dagger_{i\sigma} \hat{a}^\dagger_{j\rho} \hat{a}_{k\sigma'} \hat{a}_{l\rho'} \tag{5}$$

where $\hat{H}$ is Hermitian. In addition, specialized code is implemented for spin-conserving spin-orbital Hamiltonians ($\sigma = \sigma'$ and $\rho = \rho'$ in Eq. (5)),

$$\hat{H} = \sum_{ijkl} \sum_{\sigma\rho} V_{i\sigma,j\rho,k\sigma,l\rho} \hat{a}^\dagger_{i\sigma} \hat{a}^\dagger_{j\rho} \hat{a}_{k\sigma} \hat{a}_{l\rho}. \tag{6}$$

and for spin-free Hamiltonians ($V_{ijkl} \equiv V_{i\sigma,j\rho,k\sigma',l\rho'}$ in Eq. (6)),

$$\hat{H}_{\text{sf}} = \sum_{ijkl} \sum_{\sigma\rho} V_{ijkl} \hat{a}^\dagger_{i\sigma} \hat{a}^\dagger_{j\rho} \hat{a}_{k\sigma} \hat{a}_{l\rho}, \tag{7}$$

which frequently appear in molecular and materials systems. Other cases which have specialized subroutines discussed below include sparse Hamiltonians (arbitrary sums of excitation Hamiltonians), quadratic Hamiltonians, and diagonal pair Hamiltonians.

Time evolution with such dense Hamiltonians is performed by means of series expansions such as the Taylor and Chebyshev expansions. When the time step t is taken to be small, the operator exponential can be efficiently computed by a Taylor expansion that converges rapidly,

$$\exp(-i\hat{H}t)|\Psi\rangle = \sum_n \frac{1}{n!} (-i\hat{H}t)^n |\Psi\rangle. \tag{8}$$

We evaluate this by recursively computing the action of the operator on the wave functions,

$$|\Psi_{n+1}\rangle = \hat{H}|\Psi_n\rangle \tag{9}$$

with $|\Psi_0\rangle \equiv |\Psi\rangle$, using which the previous expression can be rewritten as

$$\exp(-i\hat{H}t)|\Psi\rangle = \sum_n \frac{(-it)^n}{n!} |\Psi_n\rangle \tag{10}$$

We evaluate the norm of each term, $|t^n|||\Psi_n\rangle|/n!$, and when the norm becomes smaller than the given threshold, we stop the computation. Alternatively, one can specify the number of terms in the expansion, or both.

When the spectral range of the operator (i.e., extremal eigenvalues $[\epsilon_{\min}, \epsilon_{\max}]$) is known or can be estimated in advance, the expansion can be made more robust by using the

Chebyshev expansion [22]. The Chebyshev expansion is known to be a near optimal approximation of the exponential in a minimax sense. First, we scale the operator such that the eigenvalues are in $[-1, +1]$,

$$\hat{H}' = \frac{\hat{H} + \epsilon_{\text{shift}}}{\Delta\epsilon} \tag{11}$$

where $\Delta\epsilon = (\epsilon_{\text{max}} - \epsilon_{\text{min}})/2w'$ and $\epsilon_{\text{shift}} = -(\epsilon_{\text{max}} + \epsilon_{\text{min}})/2$. The factor w' is a number that is slightly smaller than 1 to ensure that the eigenvalues do not lie outside the window due to numerical noise or insufficient accuracy in the estimation of extremal eigenvalues. In the FQE, we use $w' = 0.9875$. Then we expand as

$$\begin{aligned}\exp(-i\hat{H}t)|\Psi\rangle &= \exp(i\epsilon_{\text{shift}}t)\exp(-i\Delta\epsilon\hat{H}'t)|\Psi\rangle \\ &= \exp(i\epsilon_{\text{shift}}t)\sum_n 2J_n(\Delta\epsilon t)(-i)^n|\tilde{\Psi}_n\rangle \end{aligned}\tag{12}$$

Here $J_n(x)$ is the modified Bessel function of the first kind, and $|\tilde{\Psi}_n\rangle$ is obtained by recursion as

$$|\tilde{\Psi}_{n+1}\rangle = 2\hat{H}'|\tilde{\Psi}_n\rangle - |\tilde{\Psi}_{n-1}\rangle \tag{13}$$

We stop the expansion when the contribution from rank n becomes smaller than the given threshold, as in the Taylor expansion above.

The dense evolution routines can be accessed through the `Wavefunction` interface which intelligently dispatches to the specialized routines depending on the form of the Hamiltonian the user provides. As an example, below is a code snippet assuming the user has defined an OpenFermion `MolecularData` object called ‘molecule’. The Wavefunction interface provides access to the `apply_generated_unitary` routine which has options for each series expansion evolution described above.

```python
from fqe.openfermion_utils import integrals_to_fqe_restricted

dt = 0.23  # evolution time
oei, tei = molecule.get_integrals()
# A dense fqe.RestrictedHamiltonian object
fqe_rham = integrals_to_fqe_restricted(oei, tei)

wfn = fqe.Wavefunction([[molecule.n_electrons, 0, molecule.n_orbitals]])
wfn.set_wfn(strategy='hartree-fock')
new_wfn1 = wfn.time_evolve(dt, fqe_rham)

# equivalent path
# Taylor expansion algorithm
new_wfn2 = wfn.apply_generated_unitary(dt, 'taylor', fqe_rham)
```

The above algorithms that use the series expansion are also employed for time evolution with sparse Hamiltonians, which are arbitrary weighted sums of multiple excitation Hamiltonians. In this case, the `apply` function used in Eqs. 9 and 13 is overloaded by a special function for sparse Hamiltonians, in which the action of the operator is evaluated one term at a time.

3.3 Special routines for other structured Hamiltonians

3.3.1 Diagonal pair Hamiltonians

Hamiltonians that are diagonal in the determinant basis are particularly simple as they are polynomials of number operators with terms that all commute. Evolution under such Hamiltonians appears as an important quantum circuit for chemical Hamiltonian Trotter steps and certain variational ansätze [23, 29]. The FQE leverages the mutual commuting nature of the terms of Hamiltonians of this form to efficiently evolve under the generated unitaries. Specifically, the action of the unitary generated by diagonal pair Hamiltonians of the form

$$\hat{H}_{\text{diag}} = \sum_{rs} W_{rs}\hat{n}_r\hat{n}_s, \tag{14}$$

where $\hat{n}_r$ is the spin-summed number operator for spatial orbital r (i.e., $\hat{n}_r = \sum_\sigma \hat{a}^\dagger_{r\sigma}\hat{a}_{r\sigma}$), is constructed by iterating over all determinants and generating a phase associated with the coefficient W_{rs} for each determinant. A direct wall clock time comparison to other simulators is made in Figure 2 by translating evolution under a random diagonal pair Hamiltonian (of the form in Eq. (14)) into a series of CPHASE gates with phases corresponding to $2W_{rs}$. For the half filling circuits on 14 orbitals (i.e., 14 electrons in 28 spin orbitals to be represented by 28 qubits) we observe that the python reference implementation of FQE is approximately eight times faster than Cirq and five times faster than single-threaded Qsim. The C-accelerated FQE single threaded is five hundred times faster than Qsim single threaded with default gate fusion. The C-accelerated FQE on twenty threads is twenty times faster than QSim using twenty threads and gate-fusion level eight for the same 14 orbital half filled system. FQE performance gains for lower filling fractions are even more substantial.

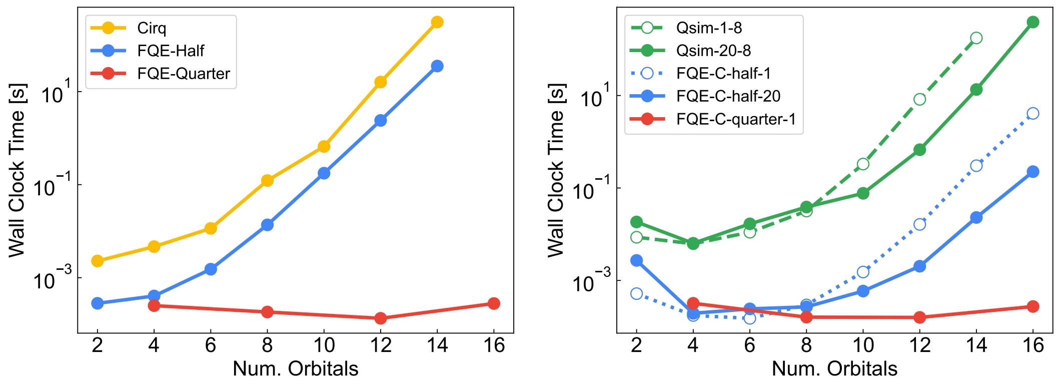

Figure 2: Run time comparisons for implementing time evolution of a random diagonal Hamiltonian: *left* The Python reference implementation of FQE is compared against Cirq's Python circuit evolution routines; *right* C-accelerated FQE is compared against Qsim, a highly optimized C++ circuit simulator with and without threading. FQE computes the results in-place. Qsim is executed in single precision mode with one (green dashed line) and twenty (green solid line) threads with gate gate-fusion set to eight through the `qsimcirq` python interface. Quantum circuits for the time evolution generated by the diagonal Coulomb Hamiltonian are constructed using $4M^2$ CPHASE gates where M is the number of spatial orbitals. FQE at half filling with one and twenty threads is shown in blue while quarter (single thread) filling is in red.

3.3.2 Quadratic Hamiltonians

Evolving states by quadratic Hamiltonians is another important algorithmic primitive. In the quantum circuit context, it is closely related to matchgate circuit simulation [42]. The FQE performs this task by transforming the wave functions into the orbital representation that diagonalizes the quadratic Hamiltonian, evolving in the quadratic Hamiltonian eigenbasis, and then returning to the original basis. The algorithm is based on those in the quantum chemistry literature [5, 25, 28] that utilize the LU decomposition, with an improvement in the handling of pivoting in the LU decomposition. Using the unitary $\mathbf{X}$ that diagonalizes the operator matrix in $\hat{A} = \sum_{ij} (\mathbf{A})_{ij} \hat{a}_i^\dagger \hat{a}_j$, i.e., $\mathbf{X}^\dagger \mathbf{A} \mathbf{X} = \mathbf{a}$, one obtains

$$\exp(-i\hat{A}t)|\Psi\rangle = \hat{T}(\mathbf{X}^\dagger) \exp(-i\hat{a}t) \hat{T}(\mathbf{X})|\Psi\rangle \tag{15}$$

where $\hat{a}$ is the diagonal operator after orbital rotation, and $\hat{T}(\mathbf{X})$ is the transformation on the wave function due to the change in the orbital basis described by the unitary $\mathbf{X}$. Here we ignore the pivoting for brevity; see Appendix B for more algorithmic details. The overall cost of rotating the wave function (i.e., action of $\hat{T}(\mathbf{X})$) is roughly equivalent to the cost to apply a single dense one-particle linear operator to a wave function. The diagonal operator is then applied to the rotated wave function, followed by the back transformation to the wave function in the original orbital basis.

Figure 3 compares the wall clock time required to evolve a random quadratic Hamiltonian. For Cirq and Qsim, a circuit is generated using the optimal Givens rotation decomposition construction of Clements [10] in OpenFermion which is translated into a sequence of $2M^2$ gates of the form $\exp\left(-i\theta(XY - YX)/2\right)$ and Rz. For the half filling circuits on fourteen orbitals (twenty eight qubits) we observe that the python-FQE LU decomposition algorithm is approximately forty times faster than Cirq. The single-thread C-accelerated FQE is approximately ten times faster than single-threaded Qsim with gate-fusion set to four. For twenty threads the C-accelerated FQE is five times faster than Qsim. Lower filling fractions show even more substantial savings. This illustrates the large advantage physics, symmetry, and algorithmic considerations can provide when targeting specialized quantum circuit emulation.

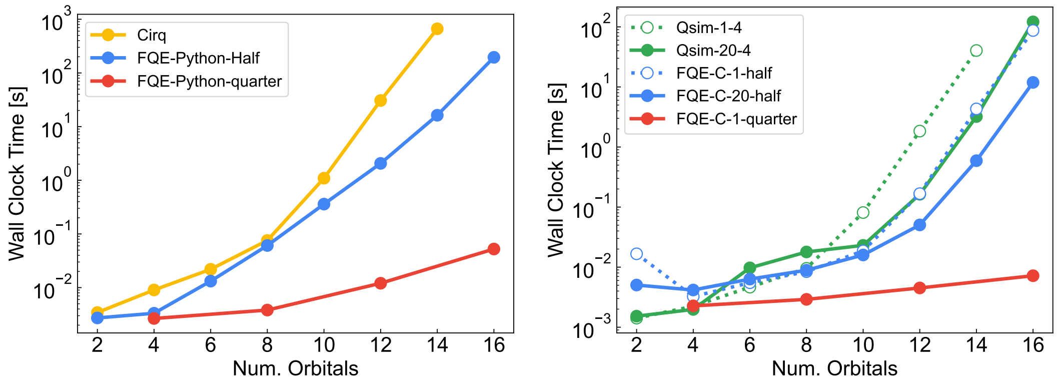

Figure 3: Wall clock time comparison for a random quadratic Hamiltonian evolution at half filling and quarter filling, *left* between Cirq and the Python implementation in FQE and *right* between Qsim and C-accelerated FQE. Note that Cirq and Qsim run times are unaffected by filling fraction. Qsim is executed accessed through the `qsimcirq` Python interface. Each Qsim trace is labeled by how many threads are used (one or twenty) and the gate-fusion level (four). Quantum circuits for the time evolution generated by the quadratic Hamiltonian are constructed using $2M^2$ $\exp\left(-i\theta(XY - YX)/2\right)$ gates where M is the number of spatial orbitals. FQE evolves the quadratic Hamiltonian in Section 3.3.2.

4 Operator Action on the Wavefunction

The FQE provides a variety of methods to apply fermionic operators to states. The general code allows a user to define an arbitrary individual excitation operator and apply it to a wavefunction. This functionality constructs the action of the user defined operator by computing the sparse matrix elements of such an operator in the determinant basis, which are then contracted against the wavefunction coefficients to obtain the action of the operator on the state.

The FQE also provides efficient methods to apply dense fermionic operators (such as a dense Hamiltonian) to wave functions, because these are the fundamental building blocks for fermionic time evolution under dense Hamiltonians (see, for example, Eqs. (9) and (13)). It is equally important for reduced density matrix (RDM) construction, efficient access to which is a crucial aspect of many fermionic simulation algorithms. In the following, we present the algorithms to apply dense spin conserving, spin-free Hamiltonians (Eq. (7)), but the FQE implements similar algorithms for S_z conserving spin-orbital Hamiltonians (Eq. (6)) and S_z non-conserving Hamiltonians (Eq. (5)) as well.

4.1 Knowles–Handy algorithm

In the Knowles–Handy algorithm [21], a resolution of the identity (RI) is inserted in the n-electron determinant space after operator reordering ($\hat{1} = \sum_K |K\rangle\langle K|$) as

$$\begin{aligned}\langle I|\hat{H}_{\mathrm{sf}}|\Psi\rangle = &-\sum_{\sigma\rho}\sum_{ik,K}\langle I|\hat{a}^\dagger_{i\sigma}\hat{a}_{k\sigma}|K\rangle\left(\sum_{jl,I}V_{ijkl}\langle K|\hat{a}^\dagger_{j\rho}\hat{a}_{l\rho}|J\rangle C_J\right)\\ &+\sum_{\sigma}\sum_{il}\left(\sum_k V_{ikkl}\right)\langle I|\hat{a}^\dagger_{i\sigma}\hat{a}_{l\sigma}|J\rangle C_J,\end{aligned} \tag{16}$$

where I, J, and K label determinants and C_I is the wave function coefficient associated with determinant $|I\rangle$. The second term is trivially evaluated. The primitive operations for the first term are

$$D^J_{ij} = \sum_I\sum_\sigma \langle J|\hat{a}^\dagger_{i\sigma}\hat{a}_{j\sigma}|I\rangle C_I, \tag{17}$$

$$\tilde{D}^J_{ij} = \sum_{kl} V_{ikjl}D^J_{kl}, \tag{18}$$

$$R^I = \sum_J\sum_\sigma \langle I|\hat{a}^\dagger_{i\sigma}\hat{a}_{j\sigma}|J\rangle \tilde{D}^J_{ij}. \tag{19}$$

This form of the RI insertion has the advantage that it can be easily generalized to apply 3- and 4-particle operators, since these primitive operations can be performed recursively, i.e.,

$$E^J_{ij,kl} = \sum_I\sum_\sigma \langle J|\hat{a}^\dagger_{i\sigma}\hat{a}_{j\sigma}|I\rangle D^I_{kl}, \tag{20}$$

$$D^I_{ij} = \sum_J\sum_\sigma \langle I|\hat{a}^\dagger_{k\sigma}\hat{a}_{l\sigma}|J\rangle E^J_{ij,kl}. \tag{21}$$

The step in Eq. (18) is typically a computational hot spot and is performed using efficient `numpy` functions.

For efficiency, the expectation values $\langle I_\sigma|\hat{a}^\dagger_{i\sigma}\hat{a}_{j\sigma}|J_\sigma\rangle$ are precomputed for the α- and β-strings and stored in an FQE object `FciGraph`. To facilitate the evaluation, the intermediate tensors are also stored using the α- and β-strings; for example, D^I_{ij} in Eq. (17) is stored as a four-index tensor $D^{I_\alpha I_\beta}_{ij}$ where $I = I_\alpha \otimes I_\beta$. When the Hamiltonian breaks spin symmetry (and, hence, S_z symmetry), the mappings between strings that have $N_\sigma \pm 1$ electrons are also generated ($\langle I'_\sigma|\hat{a}^\dagger_{i\sigma}|J_\sigma\rangle$ and $\langle I''_\sigma|\hat{a}_{i\sigma}|J_\sigma\rangle$) which are stored in the FQE object `FciGraphSet`. From these quantities, $\langle I|\hat{a}^\dagger_{i\sigma}\hat{a}_{j\sigma}|J\rangle$ can be easily computed on the fly.

4.2 Harrison–Zarrabian algorithm for low filling

Low filling cases are computed using the Harrison–Zarrabian algorithm [15], which is closely related to the algorithm developed by Olsen *et al.* [30]. This algorithm inserts the RI in the $n-2$ electron space ($\hat{1} = \sum_L |L\rangle\langle L|$) as

$$\langle I|\hat{H}_{\mathrm{sf}}|\Psi\rangle = \sum_{\sigma\rho}\sum_{ij,L}\langle I|\hat{a}^\dagger_{i\sigma}\hat{a}^\dagger_{j\rho}|L\rangle \left(\sum_{kl,J} V_{ijkl}\langle L|\hat{a}_{k\sigma}\hat{a}_{l\rho}|J\rangle C_J\right), \tag{22}$$

where I and J label a determinant with n electrons and L labels determinants with $n-2$ electrons. This is computed by the following primitive steps,

$$F^L_{ij,\sigma\rho} = \sum_I \langle L|\hat{a}_{k\sigma}\hat{a}_{l\rho}|I\rangle C_I, \tag{23}$$

$$\tilde{F}^L_{ij,\sigma\rho} = \sum_{kl} V_{ijkl}F^L_{kl,\sigma\rho}, \tag{24}$$

$$R^I = \sum_L\sum_{\sigma\rho}\langle I|\hat{a}^\dagger_{i\sigma}\hat{a}^\dagger_{j\rho}|L\rangle \tilde{F}^L_{ij,\sigma\rho}. \tag{25}$$

The advantage of this algorithm, in comparison to the Knowles–Handy algorithm, is that the number of determinants for the RI can be far smaller than the number of the original determinants. This is pronounced when the wave function is at low filling. The disadvantage, however, is that there is certain overhead in computational cost because one cannot perform spin summation in Eq. (23). Therefore, the FQE switches to this algorithm when the filling is smaller than 0.3. Note that a similar algorithm can be devised for the high-filling cases by sorting the operators in the opposite order, but this is not implemented in the FQE. When such a system is considered, one can reformulate the high-filling problem into a low-filling one by rewriting the problem in terms of the hole operators $\hat{b}_{i\sigma} = 1 - \hat{a}_{i\sigma}$.

4.3 Olsen's algorithm

The approach of Knowles and Handy, while being simple to implement with optimized linear algebra routines in Python, can be further improved by taking advantage of the sparsity in the D^J_{ij} appearing in Equation 17. For example, D^J_{ij} will be zero if determinant J has orbital i unoccupied or orbital j doubly occupied. This means that extra work is performed during the most expensive step of the algorithm, Equation 18. Olsen and co-workers were able to leverage this sparsity by avoiding explicit reference to intermediate states [30]. In this approach, the same replacement lists used by Knowles and Handy in the construction of the D^J_{ij} are used to loop over only the non-zero Hamiltonian elements connecting bra and ket. The essence of the algorithm is to structure the loops in such a way that vector operations can be used. Details of the algorithm are given by Olsen *et al.* [30] and clearly reviewed by Sherrill and Schaefer [34]. While Olsen's algorithm cannot be efficiently implemented in pure Python, it is the basis of the FQE C extension for applying dense, two-particle Hamiltonians.

4.4 RDM computation

The FQE provides efficient routines using the intermediate tensors in the above algorithms to efficiently compute 1- through 4-RDMs. For example, a spin-summed 2-RDM element Γ_{ijkl} defined as

$$\Gamma_{ijkl} = \sum_{\sigma\rho}\langle\Psi|\hat{a}^\dagger_{i\sigma}\hat{a}^\dagger_{j\rho}\hat{a}_{k\sigma}\hat{a}_{l\rho}|\Psi\rangle \tag{26}$$

can be computed either using the intermediate tensor in the Knowles–Handy algorithm (Eq. (17)),

$$\Gamma_{ijkl} = -\sum_I D^{I*}_{ki} D^I_{jl} + \delta_{jk} \sum_I D^{I*}_{li} C_I, \tag{27}$$

or, in low filling cases, using the intermediate in the Harrison–Zarrabian algorithm,

$$\Gamma_{ijkl} = -\sum_{\sigma\rho} \sum_L F^{L*}_{ij,\sigma\rho} F^L_{kl,\sigma\rho}. \tag{28}$$

The corresponding generalization of Olsen's algorithm is used when the C extensions are enabled. In both Python algorithms, efficient `numpy` functions for matrix–matrix and matrix–vector multiplication are used. Spin-orbital RDMs (i.e. without the spin summation in Eq. (26)) are computed similarly. In addition to the standard (i.e., particle) RDMs, the FQE can compute RDMs that correspond to other operator orders, e.g., hole RDMs. This is done by performing Wick's theorem as implemented in `fqe.wick` and evaluating the resulting expression using the particle RDMs. For example, the spin-summed 2-body hole RDM element (Γ^h_{ijkl}) defined as

$$\Gamma^h_{ijkl} = \sum_{\sigma\rho} \langle \Psi | \hat{a}_{i\sigma} \hat{a}_{j\rho} \hat{a}^\dagger_{k\sigma} \hat{a}^\dagger_{l\rho} | \Psi \rangle \tag{29}$$

is computed as

$$\Gamma^h_{ijkl} = \Gamma_{klij} + \delta_{jk}\Gamma_{il} - 2\delta_{ik}\Gamma_{jl} - 2\delta_{jl}\Gamma_{ik}. \tag{30}$$

The FQE implements the spin-orbital counterpart of this feature as well.

For spin conserving Hamiltonians, the FQE also provides up to 4-body spin-summed RDMs and up to 3-body spin-orbital RDMs in the OpenFermion format. An example is:

```
wf = fqe.Wavefunction([[4, 0, 6]])
wf.set_wfn(strategy='random')

spin_sum_opdm = wf.expectationValue('i^ j')
spin_sum_oqdm = wf.expectationValue('i j^')
spin_sum_tpdm = wf.expectationValue('i^ j^ k l')

opdm, tpdm = wf.sector((4, 0)).get_openfermion_rdms()
d3 = wf.sector((4, 0)).get_d3()  # returns spinful 3-RDM in openfermion ordering
```

The implementation to compute RDMs offers a significant improvement over OpenFermion's native RDM generators which map Pauli operator expectations to RDM elements. As an example of this efficient RDM computation the FQE library provides a base implementation of the energy gradient due to a two-particle generator $\hat{G}_{i\sigma,j\rho,k\sigma',l\rho'} = \hat{a}^\dagger_{i\sigma}\hat{a}^\dagger_{j\rho}\hat{a}_{k\sigma'}\hat{a}_{l\rho'}$, which corresponds to evaluating $g_{i\sigma,j\rho,k\sigma',l\rho'} = \langle\Psi|[\hat{H}, \hat{G}_{i\sigma,j\rho,k\sigma',l\rho'}]|\Psi\rangle$ for all spatial and spin indices.

5 Closing thoughts and future directions

The Fermionic Quantum Emulator (FQE) described above is an open source library under the Apache 2.0 license [33]. Currently, the library is completely implemented in Python to facilitate extension and code reuse. Despite not being written in a high performance programming language, the FQE's algorithmic advantages allow us to outperform even heavily optimized quantum circuit simulators. In future releases, current computational bottlenecks will be addressed with additional performance improvements. The Fermionic

Quantum Emulator (FQE) provides a user-friendly and developer-friendly code stack without sacrificing performance. Algorithmic advantages allow the FQE to clearly outperform both Python-based and heavily optimized quantum circuit simulators, even at half filling, with our C-accelerated code. Away from half filling, the efficiency of FQE is even more pronounced. Extending the code to GPUs or other architectures may lead to even better performance.

Though this first release of the FQE focuses on exact evolution in a statevector representation, our long-term goal is to provide implementations of fermionic circuit emulation that exploit symmetry within different simulation strategies. For example, approximate representation using fermionic matrix product states~~,~~ can also be adapted to this setting. Ultimately, a variety of such techniques will need to be explored to reach an honest assessment of possible quantum advantage when simulating molecular and materials systems.

6 Acknowledgments

NCR thanks Ryan LaRose for testing and feedback on the code.

References

[1] Daniel S Abrams and Seth Lloyd. Simulation of Many-Body Fermi Systems on a Universal Quantum Computer. *Phys. Rev. Lett.*, 79(13):4, 1997. DOI: https://doi.org/10.1103/PhysRevLett.79.2586.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback). URL `https://www.netlib.org/lapack/lug/`.

[3] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Sergio Boixo, Michael Broughton, Bob B. Buckley, David A. Buell, Brian Burkett, Nicholas Bushnell, Yu Chen, Zijun Chen, Benjamin Chiaro, Roberto Collins, William Courtney, Sean Demura, Andrew Dunsworth, Edward Farhi, Austin Fowler, Brooks Foxen, Craig Gidney, Marissa Giustina, Rob Graff, Steve Habegger, Matthew P. Harrigan, Alan Ho, Sabrina Hong, Trent Huang, William J. Huggins, Lev Ioffe, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Cody Jones, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Seon Kim, Paul V. Klimov, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Pavel Laptev, Mike Lindmark, Erik Lucero, Orion Martin, John M. Martinis, Jarrod R. McClean, Matt McEwen, Anthony Megrant, Xiao Mi, Masoud Mohseni, Wojciech Mruczkiewicz, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Hartmut Neven, Murphy Yuezhen Niu, Thomas E. O'Brien, Eric Ostby, Andre Petukhov, Harald Putterman, Chris Quintana, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Doug Strain, Kevin J. Sung, Marco Szalay, Tyler Y. Takeshita, Amit Vainsencher, Theodore White, Nathan Wiebe, Z. Jamie Yao, Ping Yeh, and Adam Zalcman. Hartree-fock on a superconducting qubit quantum computer. *Science*, 369(6507):1084–1089, 2020. ISSN 0036-8075. DOI: 10.1126/science.abb9811. URL `https://science.sciencemag.org/content/369/6507/1084`.

[4] Alan Aspuru-Guzik, Anthony D Dutoi, Peter J Love, and Martin Head-Gordon. Simulated Quantum Computation of Molecular Energies. *Science*, 309(5741):1704, 2005. DOI: 10.1126/science.1113479.

[5] Gregory J. Atchity and Klaus Ruedenberg. Orbital transformations and configurational transformations of electronic wavefunctions. *J. Chem. Phys.*, 111(7):2910–2920, 1999. DOI: 10.1063/1.479573. URL https://doi.org/10.1063/1.479573.

[6] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, and Hartmut Neven. Simulation of low-depth quantum circuits as complex undirected graphical models. *arXiv preprint arXiv:1712.05384*, 2017. URL https://arxiv.org/abs/1712.05384.

[7] Sergey Bravyi, Dan Browne, Padraic Calpin, Earl Campbell, David Gosset, and Mark Howard. Simulation of quantum circuits by low-rank stabilizer decompositions. *Quantum*, 3:181, 2019. URL https://dx.doi.org/10.22331/q-2019-09-02-181.

[8] Jia Chen, Hai-Ping Cheng, and James K. Freericks. Quantum-inspired algorithm for the factorized form of unitary coupled cluster theory. *J. Chem. Theory Comput.*, 17 (2):841–847, 2021. DOI: 10.1021/acs.jctc.0c01052. URL https://doi.org/10.1021/acs.jctc.0c01052.

[9] Cirq Developers. Cirq, March 2021. URL https://doi.org/10.5281/zenodo.4586899. See full list of authors on Github: https://github .com/quantumlib/Cirq/-graphs/contributors.

[10] William R Clements, Peter C Humphreys, Benjamin J Metcalf, W Steven Kolthammer, and Ian A Walmsley. Optimal design for universal multiport interferometers. *Optica*, 3(12):1460–1465, 2016. DOI: 10.1364/OPTICA.3.001460. URL https://www.osapublishing.org/optica/fulltext.cfm?uri=optica-3-12-1460.

[11] Francesco A. Evangelista, Garnet Kin-Lic Chan, and Gustavo E. Scuseria. Exact parameterization of fermionic wave functions via unitary coupled cluster theory. *J. Chem. Phys.*, 151(24):244112, 2019. DOI: 10.1063/1.5133059. URL https://doi.org/10.1063/1.5133059.

[12] Richard P Feynman. Simulating physics with computers. *Int. J. Theor. Phys.*, 21 (6-7):467–488, 1982. ISSN 00207748. DOI: 10.1007/BF02650179.

[13] Maria-Andreea Filip and Alex J. W. Thom. A stochastic approach to unitary coupled cluster. *J. Chem. Phys.*, 153(21):214106, 2020. DOI: 10.1063/5.0026141. URL https://doi.org/10.1063/5.0026141.

[14] Johnnie Gray and Stefanos Kourtis. Hyper-optimized tensor network contraction. *Quantum*, 5:410, 2021. URL https://dx.doi.org/10.22331/q-2021-03-15-410.

[15] Robert J. Harrison and Sohrab Zarrabian. An efficient implementation of the full-ci method using an (n–2)-electron projection space. *Chem. Phys. Lett.*, 158(5):393–398, 1989. ISSN 0009-2614. DOI: https://doi.org/10.1016/0009-2614(89)87358-0. URL https://www.sciencedirect.com/science/article/pii/0009261489873580.

[16] Cupjin Huang, Fang Zhang, Michael Newman, Junjie Cai, Xun Gao, Zhengxiong Tian, Junyin Wu, Haihong Xu, Huanjun Yu, Bo Yuan, et al. Classical simulation of quantum supremacy circuits. *arXiv preprint arXiv:2005.06787*, 2020. URL https://arxiv.org/abs/2005.06787.

[17] Yifei Huang and Peter Love. Approximate stabilizer rank and improved weak simulation of clifford-dominated circuits for qudits. *Phys. Rev. A*, 99:052307, May 2019. DOI: 10.1103/PhysRevA.99.052307. URL https://link.aps.org/doi/10.1103/PhysRevA.99.052307.

[18] Yifei Huang and Peter Love. Feynman-path-type simulation using stabilizer projector decomposition of unitaries. *Phys. Rev. A*, 103:022428, Feb 2021. DOI: 10.1103/PhysRevA.103.022428. URL https://link.aps.org/doi/10.1103/PhysRevA.103.022428.

[19] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. Hardware-efficient variational quantum

eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, 2017. DOI: https://doi.org/10.1038/nature23879. URL https://www.nature.com/articles/nature23879.

[20] Peter J. Knowles and Nicholas C. Handy. A determinant based full configuration interaction program. *Comput. Phys. Comm.*, 54(1):75–83, 1989. ISSN 0010-4655. DOI: https://doi.org/10.1016/0010-4655(89)90033-7. URL https://www.sciencedirect.com/science/article/pii/0010465589900337.

[21] P.J. Knowles and N.C. Handy. A new determinant-based full configuration interaction method. *Chem. Phys. Lett.*, 111(4):315–321, 1984. ISSN 0009-2614. DOI: https://doi.org/10.1016/0009-2614(84)85513-X. URL https://www.sciencedirect.com/science/article/pii/000926148485513X.

[22] R Kosloff. Propagation methods for quantum molecular dynamics. *Annu. Rev. Phys. Chem.*, 45(1):145–178, 1994. DOI: 10.1146/annurev.pc.45.100194.001045. URL https://doi.org/10.1146/annurev.pc.45.100194.001045.

[23] Joonho Lee, William J Huggins, Martin Head-Gordon, and K Birgitta Whaley. Generalized unitary coupled cluster wave functions for quantum computation. *J. Chem. Theory Comput.*, 15(1):311–324, 2018. DOI: https://doi.org/10.1021/acs.jctc.8b01004. URL https://pubs.acs.org/doi/10.1021/acs.jctc.8b01004.

[24] Xiu-Zhe Luo, Jin-Guo Liu, Pan Zhang, and Lei Wang. Yao. jl: Extensible, efficient framework for quantum algorithm design. *Quantum*, 4:341, 2020. DOI: https://doi.org/10.22331/q-2020-10-11-341. URL https://quantum-journal.org/papers/q-2020-10-11-341/.

[25] Per Åke Malmqvist. Calculation of transition density matrices by nonunitary orbital transformations. *Int. J. Quantum Chem.*, 30(4):479–494, 1986. DOI: https://doi.org/10.1002/qua.560300404. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/qua.560300404.

[26] Igor L Markov and Yaoyun Shi. Simulating Quantum Computation by Contracting Tensor Networks. *SIAM J. Comput.*, 38(3):963–981, 2008. ISSN 0097-5397. DOI: 10.1137/050644756. URL http://epubs.siam.org/doi/abs/10.1137/050644756.

[27] Jarrod R McClean, Nicholas C Rubin, Kevin J Sung, Ian D Kivlichan, Xavier Bonet-Monroig, Yudong Cao, Chengyu Dai, E Schuyler Fried, Craig Gidney, Brendan Gimby, et al. Openfermion: the electronic structure package for quantum computers. *Quantum Science and Technology*, 5(3):034014, 2020. DOI: 10.1088/2058-9565/ab8ebc. URL https://doi.org/10.1088/2058-9565/ab8ebc.

[28] A. Mitrushchenkov and H.-J. Werner. Calculation of transition moments between internally contracted mrci wave functions with non-orthogonal orbitals. *Mol. Phys.*, 105(9):1239–1249, 2007. DOI: 10.1080/00268970701326978. URL https://doi.org/10.1080/0026897070132697.

[29] Mario Motta, Erika Ye, Jarrod R McClean, Zhendong Li, Austin J Minnich, Ryan Babbush, and Garnet Kin-Lic Chan. Low rank representations for quantum simulation of electronic structure. *arXiv preprint arXiv:1808.02625*, 2018. URL https://arxiv.org/abs/1808.02625.

[30] Jeppe Olsen, Björn O. Roos, Poul Jørgensen, and Hans J. "Aa". Jensen. Determinant based configuration interaction algorithms for complete and restricted configuration interaction spaces. *J. Chem. Phys.*, 89(4):2185–2192, 1988. DOI: 10.1063/1.455063. URL https://doi.org/10.1063/1.455063.

[31] G Ortiz, J Gubernatis, E Knill, and R Laflamme. Quantum algorithms for fermionic simulations. *Phys. Rev. A*, 64(2):22319, 2001. ISSN 1050-2947. DOI: 10.1103/PhysRevA.64.022319. URL http://link.aps.org/doi/10.1103/PhysRevA.64.022319.

[32] Quantum AI team and collaborators. qsim, September 2020. URL https://doi.org/10.5281/zenodo.4023103.

[33] Nicholas C. Rubin, Toru Shiozaki, Kyle Throssell, Garnet K.-L. Chan, and Ryan Babbush. The Fermionic Quantum Emulator: https://github.com/quantumlib/openfermion-fqe, Aug 2020. URL https://github.com/quantumlib/OpenFermion-FQE.

[34] C. David Sherrill and Henry F. Schaefer. The Configuration Interaction Method: Advances in Highly Correlated Approaches. *Advances in Quantum Chemistry*, 34(C): 143–269, 1999. ISSN 00653276. DOI: 10.1016/S0065-3276(08)60532-8.

[35] Toru Shiozaki. Bagel: Brilliantly advanced general electronic-structure library. *Wiley Interdiscip. Rev. Comput. Mol. Sci.*, 8(1):e1331, 2018. DOI: https://doi.org/10.1002/wcms.1331. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1331.

[36] Per E.M. Siegbahn. A new direct ci method for large ci expansions in a small orbital space. *Chem. Phys. Lett.*, 109(5):417–423, 1984. ISSN 0009-2614. DOI: https://doi.org/10.1016/0009-2614(84)80336-X. URL https://www.sciencedirect.com/science/article/pii/000926148480336X.

[37] Mikhail Smelyanskiy, Nicolas PD Sawaya, and Alán Aspuru-Guzik. qhipster: The quantum high performance software testing environment. *arXiv preprint arXiv:1601.07195*, 2016. URL https://arxiv.org/abs/1601.07195.

[38] Daniel GA Smith, Lori A Burns, Andrew C Simmonett, Robert M Parrish, Matthew C Schieber, Raimondas Galvelis, Peter Kraus, Holger Kruse, Roberto Di Remigio, Asem Alenaizan, et al. Psi4 1.4: Open-source software for high-throughput quantum chemistry. *J. Chem. Phys.*, 152(18):184108, 2020. DOI: https://doi.org/10.1063/5.0006002. URL https://aip.scitation.org/doi/10.1063/5.0006002.

[39] Nicholas H Stair and Francesco A Evangelista. Qforte: an efficient state simulator and quantum algorithms library for molecular electronic structure. *arXiv preprint arXiv:2108.04413*, 2021. URL https://arxiv.org/abs/2108.04413.

[40] 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, et al. Pyscf: the python-based simulations of chemistry framework. *Wiley Interdiscip. Rev. Comput. Mol. Sci.*, 8(1):e1340, 2018. DOI: https://doi.org/10.1063/5.0006002. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1340.

[41] Yasunari Suzuki, Yoshiaki Kawase, Yuya Masumura, Yuria Hiraga, Masahiro Nakadai, Jiabao Chen, Ken M Nakanishi, Kosuke Mitarai, Ryosuke Imai, Shiro Tamiya, et al. Qulacs: a fast and versatile quantum circuit simulator for research purpose. *arXiv preprint arXiv:2011.13524*, 2020. URL https://arxiv.org/abs/2011.13524.

[42] Barbara M. Terhal and David P. DiVincenzo. Classical simulation of noninteracting-fermion quantum circuits. *Phys. Rev. A*, 65:032325, Mar 2002. DOI: 10.1103/PhysRevA.65.032325. URL https://link.aps.org/doi/10.1103/PhysRevA.65.032325.

A Evolution under a Hermitian Hamiltonian generated by a sum of excitation operators

Here we discuss the unitary evolution associated with a fermionic excitation (Eq. (2)) that is not diagonal and has no repeated orbital indices. In this case, one can trivially show that $\hat{g}\hat{g} = 0$ and $\hat{g}^\dagger\hat{g}^\dagger = 0$. Using these relations, it can be shown that

$$(\hat{g} + \hat{g}^\dagger)^n = \hat{g}\hat{g}^\dagger\hat{g}\cdots + \hat{g}^\dagger\hat{g}\hat{g}^\dagger\cdots . \tag{31}$$

In addition, $\hat{g}^\dagger\hat{g}$ and $\hat{g}\hat{g}^\dagger$ are both diagonal in the computational basis, because any of the unpaired operators in $\hat{g}$ would be paired with its conjugate in $\hat{g}^\dagger\hat{g}$ and $\hat{g}\hat{g}^\dagger$. For example, let $\hat{g} = g\hat{a}_4^\dagger\hat{a}_2^\dagger\hat{a}_3\hat{a}_1$ (we have omitted the spin indices for brevity), then

$$\begin{aligned}\hat{g}\hat{g}^\dagger &= |g|^2\hat{a}_4^\dagger\hat{a}_2^\dagger\hat{a}_3\hat{a}_1\hat{a}_1^\dagger\hat{a}_3^\dagger\hat{a}_2\hat{a}_4 \\ &= |g|^2\hat{n}_4\hat{n}_2(1-\hat{n}_3)(1-\hat{n}_1) \end{aligned} \tag{32}$$

It is worth noting that the parity associated with this reordering for normal-ordered g (all creation operators to the left of annihilation operators) is always even. We define the square root of this diagonal operator with the following phase convention,

$$\sqrt{\hat{g}\hat{g}^\dagger} \equiv |g|\hat{n}_4\hat{n}_2(1-\hat{n}_3)(1-\hat{n}_1) \tag{33}$$

where we used the fact that the action of the number operators gives 0 or 1, and therefore, they are idempotent. Using these expressions, the Taylor expansion of the evolution operator is exactly re-summed (where the summations are performed separately for the odd and even rank terms) as

$$\begin{aligned} e^{-i(g+g^\dagger)\epsilon} =& \sum_{n=0}^{\infty}\frac{(-i\epsilon)^n}{n!}(g+g^\dagger)^n \\ =& -1 + \sum_{n=0}^{\infty}(-1)^n\frac{\left[(\hat{g}\hat{g}^\dagger)^n + (\hat{g}^\dagger\hat{g})^n\right]\epsilon^{2n}}{(2n)!} \\ & - i\sum_{n=0}^{\infty}(-1)^n\frac{\left[\hat{g}(\hat{g}^\dagger\hat{g})^n + \hat{g}^\dagger(\hat{g}\hat{g}^\dagger)^n\right]\epsilon^{2n+1}}{(2n+1)!} \\ =& -1 + \cos(\epsilon\sqrt{\hat{g}\hat{g}^\dagger}) + \cos(\epsilon\sqrt{\hat{g}^\dagger\hat{g}}) \\ & - ig^\dagger\frac{1}{\sqrt{\hat{g}\hat{g}^\dagger}}\sin(\epsilon\sqrt{\hat{g}\hat{g}^\dagger}) - i\hat{g}\frac{1}{\sqrt{\hat{g}^\dagger\hat{g}}}\sin(\epsilon\sqrt{\hat{g}^\dagger\hat{g}}). \end{aligned} \tag{34}$$

Denoting the projection to the set of determinants that are not annihilated by an operator $\hat{x}$ as $\hat{\mathcal{P}}_x$, together with the convention in Eq. (33), this can be further simplified to Eq. (3).

B Basis change: Evolution under quadratic Hamiltonians

In this work, we make use of the following primitive. Given an orbital basis $\{|\phi\rangle\}$ and many-electron wave function $|\Psi\rangle = \sum_{I(\phi)} C_{I(\phi)}|I(\phi)\rangle$ and a linear transformation $\hat{X}|\phi\rangle \to |\phi'\rangle$, we wish to re-express $|\Psi\rangle = \sum_{I(\phi')} C_{I(\phi')}|I(\phi')\rangle$ where $|I(\phi')\rangle$ is a determinant in the new orbital basis $\{|\phi'\rangle\}$. There have been several discussions of how to implement this transformation efficiently [5, 25, 28]. In the original work by Malmqvist [25], it was understood that the transformation can be performed by the successive application of one-body operators to the wave functions, in which the operator was computed from the LU decomposition of the orbital transformation matrix $\mathbf{X}$ ($X_{ij} = \langle\phi_i|\phi'_j\rangle$). Mitrushchenkov and Werner (MW) [28] later reported that the pivoting in the LU decomposition is necessary to make the transformation stable. In their work, a LAPACK [2] function was used to perform the LU decomposition with pivoting on the rows of the orbital transformation matrix. This leads to reordering of the orbitals in the determinants and additional phase factors in $C_{I(\phi')}$ that make it complicated to, for example, evaluate the overlap and expectation values. Therefore, we have implemented a column-pivoted formulation of the MW algorithm, which is presented below. We will show the spin-free formulation (i.e. same transformation for α and β orbitals) for spin-conserving wave functions as an example, but the procedure for the spin-broken case can be derived in the same way.

First, we obtain the column pivoted LU decomposition. This is done using `numpy.linalg.lu` (which pivots rows) for the transpose of the orbital transformation matrix $\mathbf{X}$,

$$\mathbf{X}^T = \bar{\mathbf{P}}\bar{\mathbf{L}}\bar{\mathbf{U}}, \tag{35}$$

where $\bar{\mathbf{L}}$ and $\bar{\mathbf{U}}$ are lower- and upper-triangular matrices, and the diagonal elements of $\bar{\mathbf{L}}$ are unit. It is then easily seen that, by taking the transpose, one obtains

$$\mathbf{X} = \bar{\mathbf{U}}^T\bar{\mathbf{L}}^T\bar{\mathbf{P}}^T, \tag{36}$$

where $\bar{\mathbf{U}}^T$ and $\bar{\mathbf{L}}^T$ are lower and upper triangular, respectively. If one scales the rows and columns of $\bar{\mathbf{U}}^T$ and $\bar{\mathbf{L}}^T$, respectively, such that the diagonal elements of $\bar{\mathbf{U}}^T$ become unit, this can be rewritten as

$$\mathbf{X} = \mathbf{LUP}, \tag{37}$$

in which $\mathbf{L}$ and $\mathbf{U}$ are lower- and upper-triangular matrices with the diagonal elements of $\mathbf{L}$ being unit, and $\mathbf{P} = \bar{\mathbf{P}}^T$. These are done in the `ludecomp` and `transpose_matrix` subfunctions in the `wavefunction.transform` function.

When pivoting is not considered (i.e., $\mathbf{P} = \mathbf{1}$), wave functions are transformed as follows (see Ref. [25] for details). Suppose that the spin-free MO coefficients are rotated as

$$\tilde{C}_{ri} = \sum_j C_{rj}(\mathbf{LU})_{ji} \tag{38}$$

Using $\mathbf{L}$ and $\mathbf{U}$, we compute the matrix elements

$$\mathbf{F} = \mathbf{U}^{-1} - \mathbf{L} - \mathbf{I} \tag{39}$$

which is performed in the `process_matrix` function. It has been shown [25] that this operator can be used to transform the many-body wave functions. Let $\hat{T}$ be the one-particle operator associated with this transformation, i.e.,

$$|\Psi'\rangle = \hat{T}(\mathbf{LU})|\Psi\rangle. \tag{40}$$

We perform

$$|\Psi_{k+1,\sigma}\rangle = (1 + \hat{F}_{k\sigma})|\Psi_{k,\sigma}\rangle \tag{41}$$

in which $\hat{F}_{k\sigma}$ is a one-body, spin-orbital operator whose matrix elements are the kth column of $\mathbf{F}$, i.e.,

$$\hat{F}_{k\sigma} = \sum_i F_{ik}\hat{a}^\dagger_{i\sigma}\hat{a}_{k\sigma} \tag{42}$$

This operation is performed recursively first for $\sigma = \alpha$ from $k = 0$ ($|\Psi_{0,\alpha}\rangle = |\Psi\rangle$) to $k = n-1$ where $n = \dim(X)$ using a specialized `apply` function for "one-column" one-particle operator, followed by the same procedure for $\sigma = \beta$ from $k = 0$ ($|\Psi_{0,\beta}\rangle = |\Psi_{n,\alpha}\rangle$) to $k = n-1$. The resulting wave function, $|\Psi'\rangle = |\Psi_{n,\beta}\rangle$, then has the same determinant expansion coefficients $C_{I(\phi)}$ as the coefficients of $|\Psi\rangle$ ($C_{I(\phi')}$) when expressed in the determinants of the new orbital basis.

In the context of exact evolution with a quadratic Hermitian operator, the orbital transformation is performed to diagonalize the quadratic operator. Assume that the operator is $\hat{A} = \sum_{ij} A_{ij}\hat{E}_{ij}$ with $\hat{E}_{ij} = \sum_\sigma a^\dagger_{i\sigma}a_{j\sigma}$; we diagonalize it so that $\mathbf{X}^\dagger\mathbf{A}\mathbf{X} = \mathbf{a}$ where $\mathbf{a}$ is a diagonal matrix. Note that $\mathbf{X}$ is unitary. Using the column-pivoted LU decomposition (Eq. 37), the propagation can be performed as

$$\exp(-i\hat{A}t)|\Phi\rangle = \hat{T}((\mathbf{LU})^\dagger)\exp(-i\hat{a}'t)\hat{T}(\mathbf{LU})|\Phi\rangle \tag{43}$$

where $\hat{a}'$ is a diagonal operator,

$$\hat{a}' = \sum_i (\mathbf{PaP}^T)_{ii}\sum_\sigma \hat{a}^\dagger_{i\sigma}\hat{a}_{i\sigma} \tag{44}$$

To summarize, in this column-pivoted formulation, the pivot matrix $\mathbf{P}$ is used only to reorder the orbitals in the diagonal operator. This is more trivial and efficient than reordering the orbitals in the wave functions.