HighTemperatureSuperConductorExample

May 1, 2024

1 Fermi Hubbard Ground State Energy Estimation with Quantum Circuits

1.1 Single Band

The Fermi-Hubbard model is often used to predict the behavior of the electrons of proposed superconductors. It has the advantage of being simply stated while describing a variety of phenomena relating to the phases of real-world materials such as the transition between superconducting and Mott insulator phases. In order to understand the behavior of the Fermi-Hubbard model at high temperatures, it is first critical to understand the phase diagram of the Fermi-Hubbard model at absolute zero (with no heat-bath present). To achieve this, one must understand the ground state energy of the model with various filling coefficients. In this notebook, we will show the necessary steps perform this analysis on a quantum computer.

The most basic form of the Fermi-Hubbard model with a single band can be represented through the following equation:

$$H = -\sum_{(i,j)} \sum_{\sigma \in \{\uparrow,\downarrow\}} t_{i,j} c^\dagger_{i,\sigma} c_{j,\sigma} + U \sum c^\dagger_{i,\uparrow} c^\dagger_{i,\downarrow} c_{i,\downarrow} c_{i,\uparrow} \tag{1}$$

which can be equivavlently written as

$$H = -\sum_{(i,j)} \sum_{\sigma \in \{\uparrow,\downarrow\}} t_{i,j} c^\dagger_{i,\sigma} c_{j,\sigma} + U \sum n_{i,\uparrow} n_{i,\downarrow} \tag{2}$$

where $c^\dagger_{i,\uparrow}$ is the fermionic creation operator on site i with a spin $\uparrow$ and $c_{i,\uparrow}$ is the fermionic annihilation operator on site i with a spin $\uparrow$. The number operator on site i with spin $\uparrow$ is given by $n_{i,\uparrow} = c^\dagger_{i,\uparrow} c_{i,\uparrow}$. We will assume that the interactions are occurring on a 2-d square lattice, defining the site connectivity (i, j) The coefficients $t_{i,j}$ represent the affinity for tunnelling between two orbitals at sites i and j and the coefficient U represents the strength of the Coulomb interaction between electrons on the same orbital.

There are 3 steps to estimate the ground state energy of this Hamiltonian:

1. Prepare an initial state believed to have non-vanishing overlap with the actual ground state of the input Hamiltonian. This is believed to be reasonably achieved by first using the classical Hartree-Fock Approximation to prepare a product state with low energy and applying local unitary rotations to the $|0...0\rangle$ state.

2. Perform Ground State Energy Estimation to determine the configuration of low energy states of the input Hamiltonian. The Hamiltonian must first be transformed into a Hamiltonian represented in the Pauli Basis by applying a transformation such as either the Jordan-Wigner Transformation or the Bravyi-Kitaev Transformation.

3. Measure autocorrelation operators $\frac{1}{2}\langle \Delta_{i,j} + \Delta_{i,j}^{\dagger} \rangle$ where $\Delta_{i,j} = \langle c_{i+}c_{j-} - c_{i-}c_{j+} \rangle$. This can be achieved by applying the same transformation used in 2. and applying that Pauli operator as a rotation to translate the observable into the Pauli basis.

This procedure can be repeated for input states with various electron fillings to compute the phase diagram for the system.

[1]:
```
import time
import openfermion as of
import numpy as np
from pyLIQTR.PhaseEstimation.pe import PhaseEstimation
from networkx import get_node_attributes, draw, draw_networkx_edge_labels
from qca.utils.utils import circuit_estimate
from qca.utils.hamiltonian_utils import generate_two_orbital_nx,␣
↪nx_to_two_orbital_hamiltonian
```

[2]:
```
#Generating a 20x20 Fermi Hubbard model with a single band.  The ratio between␣
↪Tunneling and Coulomb parameters can be swept to search for the appropriate␣
↪mean electron filling.
n = 20
tunneling = 1
coulomb = 8
ham_one_band = of.fermi_hubbard(n, n, tunneling=tunneling, coulomb=coulomb,␣
↪periodic=False) #returns an aperiodic fermionic hamiltonian
```

Now that we have the Hamiltonian generated, we need to generate an initial state with non-vanishing overlap with the ground state (or the low energy subspace). This can reasonably be achieved by using mean field methods like Hartree-Fock. This is an advantageous approach since Hartree-Fock will generate a product state, which should be easily prepared using local operations. We are currently looking for good implementations for the Fermi-Hubbard model, so we do not currently have this portion incorporated into this notebook, but it will be implemented in the coming weeks.

[3]:
```
#TODO: Incorporate Hartree-Fock into this section to prepare the initial state␣
↪for QPE for GSEE.
#This should provide a low depth initialization circuit relative to the depth␣
↪of the QPE, while giving access to a low-energy subspace
```

Once the initial state has been prepared, we can now perform Quantum Phase Estimation to estimate the ground state energy. Currently we are using a short evolution time and a second order trotterization with a single step. We use scaling arguments to determine the final resources since generating the full circuit for a large number of trotter steps with many bits of precision is quite costly. The circuit depth scales linearly with the number of trotter steps and exponentially base 2 for the number of bits of precision. This means that all of the resource estimates will be

rather large. It should be noted that more recently, there has been an implementation released in pyLIQTR for using Quantum Phase Estimation with Quantum Signal Processing as a sub-process. Whether this can yield an improvement in resource requirements has yet to be explored. We would like to find results with a precision on the order of 10^{-5}, so we are using 16 bits of precision.

[4]:
```
trotter_order_one_band = 2
trotter_steps_one_band = 1

#this scales the circuit depth proportional to 2 ^ bits_precision
bits_precision_one_band = 16

E_min_one_band = -len(ham_one_band.terms)
E_max_one_band = 0
one_band_omega = E_max_one_band-E_min_one_band
t_one_band = 2*np.pi/one_band_omega
one_band_phase_offset = E_max_one_band*t_one_band

args_one_band = {
    'trotterize' : True,
    'mol_ham'    : ham_one_band,
    'ev_time'    : t_one_band,
    'trot_ord'   : trotter_order_one_band,
    'trot_num'   : 1 #handling adjustment in resource estimate to save time - scales circuit depth linearly.
}


init_state_one_band = [0] * n * n * 2 #TODO: use Fock state from Hartree-Fock as initial state

print('starting')

t0 = time.perf_counter()
gse_inst_one_band = PhaseEstimation(
    precision_order=1, #actual precision bits accounted as scaling factors in the resource estimate
    init_state=init_state_one_band,
    phase_offset=one_band_phase_offset,
    include_classical_bits=False, # Do this so print to openqasm works
    kwargs=args_one_band)
gse_inst_one_band.generate_circuit()
t1 = time.perf_counter()
print(f'One band GSEE time to generate high level Circuit: {t1 - t0}')

gse_circuit_one_band = gse_inst_one_band.pe_circuit
```

```
starting
One band GSEE time to generate high level Circuit: 7.689720785943791
```

Now we need to convert the GSEE circuit to Clifford + T and write the data to a file

[5]:
```
print('Estimating one_band')
t0 = time.perf_counter()
estimate = circuit_estimate(circuit=gse_circuit_one_band,
                            outdir='GSE/FermiHubbard/',
                            circuit_name='one_band',
                            algo_name='GSE_Step',
                            write_circuits=True,
                            bits_precision=bits_precision_one_band,
                            numsteps=trotter_steps_one_band)
t1 = time.perf_counter()
print(f'Time to estimate one_band: {t1-t0}')
```

```
Estimating one_band
   Time to decompose high level <class 'cirq.ops.common_gates.HPowGate circuit:
0.0005054879002273083 seconds
   Time to transform decomposed <class 'cirq.ops.common_gates.HPowGate circuit
to Clifford+T: 0.00010548694990575314 seconds
   Time to decompose high level <class 'cirq.ops.identity.IdentityGate circuit:
0.00014014006592333317 seconds
   Time to transform decomposed <class 'cirq.ops.identity.IdentityGate circuit
to Clifford+T: 1.4876946806907654e-05 seconds
   Time to decompose high level <class
'pyLIQTR.PhaseEstimation.pe_gates.PhaseOffset circuit: 0.00048470403999090195
seconds
   Time to transform decomposed <class
'pyLIQTR.PhaseEstimation.pe_gates.PhaseOffset circuit to Clifford+T:
0.00166576006449759 seconds
   Time to decompose high level <class
'pyLIQTR.PhaseEstimation.pe_gates.Trotter_Unitary circuit: 438.78053596708924
seconds
   Time to transform decomposed <class
'pyLIQTR.PhaseEstimation.pe_gates.Trotter_Unitary circuit to Clifford+T:
327.61232339404523 seconds
   Time to decompose high level <class
'cirq.ops.measurement_gate.MeasurementGate circuit: 0.4579515589866787 seconds
   Time to transform decomposed <class
'cirq.ops.measurement_gate.MeasurementGate circuit to Clifford+T:
0.000228530028834939 seconds
Time to estimate one_band: 1111.585771051934
```

After Ground State Energy Estimation has been performed, we need to measure the autocorrelation, $\frac{1}{2}\langle\Delta_{i,j} + \Delta_{i,j}^\dagger\rangle$ where $\Delta_{i,j} = \langle c_{i+}c_{j-} - c_{i-}c_{j+}\rangle$. For the case where $i = j = 0$ is in the middle of the lattice, one can rotate into the Pauli basis using the following transformation. We will label the qubit representing site $i+$ as qubit 1 and the qubit representing site $i-$ as qubit 2. Given $U = CNOT_{i+,i-}H_{i+}CNOT_{i+,i-}$, it can be shown that $U^\dagger(c_{i+}c_{i-} - c_{i-}c_{i+})U = -2|\uparrow\uparrow\rangle\langle\uparrow\uparrow| + 2|\downarrow\downarrow\rangle\langle\downarrow\downarrow|$. This means that by simply applying the circuit U, then measuring, we can observe the

autocorrelation. This introduces 3 Clifford operations and no T gates.

A similar operation, though acting on 4 qubits rather than 2 qubits, can be achieved in a similarly low constant circuit depth for the case where $i \neq j$.

1.2 Two band

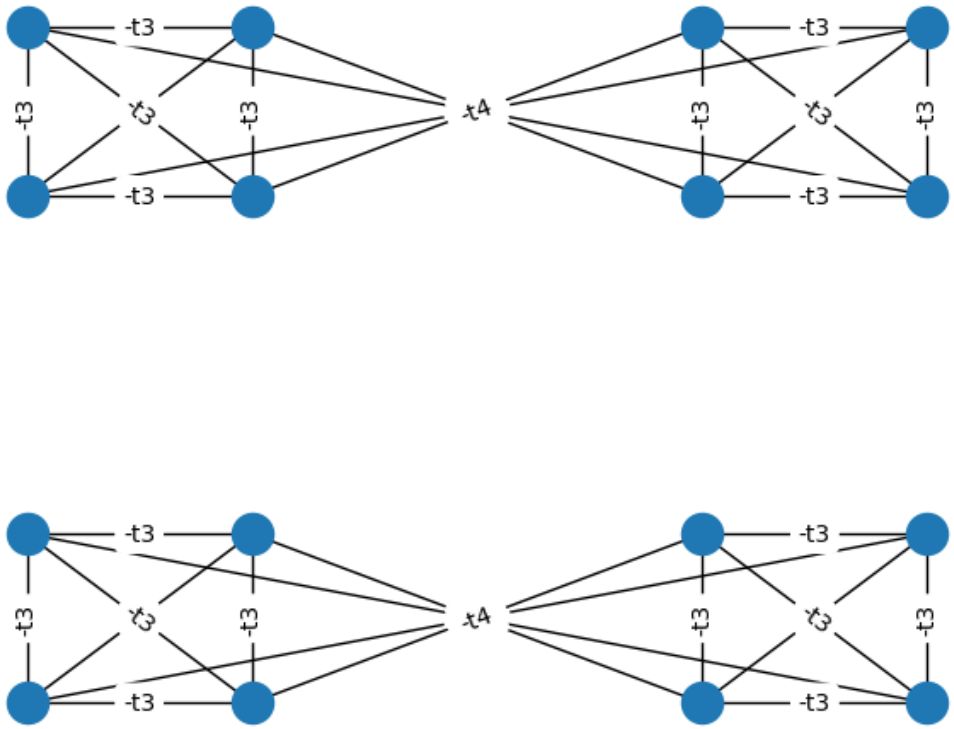
We can now apply the same process for the two band model as well. Consider the two orbital tight binding Fermi Hubbard Model for cuprate superconductors (seen for example here) on a square lattice:

$$\begin{aligned} H_{TB} = &-t_1 \sum_{i,\sigma} \left(d^\dagger_{i,x,\sigma} d_{i+\hat{y},x,\sigma} + d^\dagger_{i,y,\sigma} d_{i+\hat{x},y,\sigma} + h.c. \right) \\ &- t_2 \sum_{i,\sigma} \left(d^\dagger_{i,x,\sigma} d_{i+\hat{x},x,\sigma} + d^\dagger_{i,y,\sigma} d_{i+\hat{y},y,\sigma} + h.c. \right) \\ &- t_3 \sum_{i,\hat{\mu},\hat{\nu},\sigma} \left(d^\dagger_{i,x,\sigma} d_{i+\hat{\mu}+\hat{\nu},x,\sigma} + d^\dagger_{i,y,\sigma} d_{i+\hat{\mu}+\hat{\nu},y,\sigma} + h.c. \right) \\ &+ t_4 \sum_{i,\sigma} \left(d^\dagger_{i,x,\sigma} d_{i+\hat{x}+\hat{y},y,\sigma} + d^\dagger_{i,y,\sigma} d_{i+\hat{x}+\hat{y},x,\sigma} + h.c. \right) \\ &- t_4 \sum_{i,\sigma} \left(d^\dagger_{i,x,\sigma} d_{i+\hat{x}-\hat{y},y,\sigma} + d^\dagger_{i,y,\sigma} d_{i+\hat{x}-\hat{y},x,\sigma} + h.c. \right) \\ &- \mu \sum_i (n^x_i + n^y_i) \end{aligned} \tag{3}$$

where $t_1 = -1.0, t_2 = 1.3, t_3 = t_4 = -0.85$ with operators $d^\dagger_{i,\alpha,\sigma}$ and $d_{i,\alpha,\sigma}$ respectively create or annihilate electrons on an atom at site i, with orbital α, and spin σ. The operator n^α_i represents the number operator of an atom at a given orbital, i.e. $n^x_i = \sum_\sigma \left(d^\dagger_{i,x,\sigma} d_{i,x,\sigma} \right)$. The indices in the Hamiltonian are assigned in the following manner: the index i will correspond to a 2-tuple (m, n) indicating the x and y coordinates of the atom in the lattice. Since there are 2 orbitals, $\alpha \in \{x, y\}$. Lastly, $\sigma \in \{\uparrow, \downarrow\}$. For the sake of simplicity of coding, we will remap these labels to integer values, meaning we will assign indices $m \in L_x, n \in L_y, a \in \mathbb{Z}^2, s \in \mathbb{Z}^2$ where L_x is the x dimension of the square lattice, and L_y is the y dimension of the square lattice. In the summations, $\hat{x}$ and $\hat{y}$ correspond to adjacent sites in the lattice, rather than the orbitals. The summation over unit vectors $\hat{\mu}$ and $\hat{\nu}$ correspond to the summing over unit vectors in all 8 cardinal and intercardinal directions without double counting.

We will consider two instances: one which represents the current capabilities of the best classical solvers (a 6x7 lattice), and one which represents the ideal capabilities of a quantum solver (a 20x20 lattice) of sites. We want to obtain the results with a precision on the order of 10^{-5}, so we will use 16 bits of precision.

[6]:
```
g_example = generate_two_orbital_nx(2,2)
pos = get_node_attributes(g_example, 'pos')
edge_labels = dict([((n1, n2), d['label']) for n1, n2, d in g_example.
 ↪edges(data=True)]);
draw(g_example, pos)
draw_networkx_edge_labels(g_example,pos, edge_labels = edge_labels);
```



[7]:
```
t1 = -1
t2 = 1.3
t3 = 0.85
t4 = 0.85
mu = 1
nx_to_two_orbital_hamiltonian(g_example, t1, t2, t3, t4, mu)
```

[7]:
```
-1.0 [0^ 0] +
-0.85 [0^ 4] +
-0.85 [0^ 8] +
-0.85 [0^ 12] +
0.85 [0^ 14] +
-1.0 [1^ 1] +
-0.85 [1^ 5] +
-0.85 [1^ 9] +
-0.85 [1^ 13] +
0.85 [1^ 15] +
-1.0 [2^ 2] +
-0.85 [2^ 6] +
-0.85 [2^ 10] +
```

```
0.85 [2^ 12] +
-0.85 [2^ 14] +
-1.0 [3^ 3] +
-0.85 [3^ 7] +
-0.85 [3^ 11] +
0.85 [3^ 13] +
-0.85 [3^ 15] +
-0.85 [4^ 0] +
-1.0 [4^ 4] +
-0.85 [4^ 8] +
-0.85 [4^ 10] +
-0.85 [4^ 12] +
-0.85 [5^ 1] +
-1.0 [5^ 5] +
-0.85 [5^ 9] +
-0.85 [5^ 11] +
-0.85 [5^ 13] +
-0.85 [6^ 2] +
-1.0 [6^ 6] +
-0.85 [6^ 8] +
-0.85 [6^ 10] +
-0.85 [6^ 14] +
-0.85 [7^ 3] +
-1.0 [7^ 7] +
-0.85 [7^ 9] +
-0.85 [7^ 11] +
-0.85 [7^ 15] +
-0.85 [8^ 0] +
-0.85 [8^ 4] +
-0.85 [8^ 6] +
-1.0 [8^ 8] +
-0.85 [8^ 12] +
-0.85 [9^ 1] +
-0.85 [9^ 5] +
-0.85 [9^ 7] +
-1.0 [9^ 9] +
-0.85 [9^ 13] +
-0.85 [10^ 2] +
-0.85 [10^ 4] +
-0.85 [10^ 6] +
-1.0 [10^ 10] +
-0.85 [10^ 14] +
-0.85 [11^ 3] +
-0.85 [11^ 5] +
-0.85 [11^ 7] +
-1.0 [11^ 11] +
-0.85 [11^ 15] +
```

```
-0.85 [12^ 0] +
0.85 [12^ 2] +
-0.85 [12^ 4] +
-0.85 [12^ 8] +
-1.0 [12^ 12] +
-0.85 [13^ 1] +
0.85 [13^ 3] +
-0.85 [13^ 5] +
-0.85 [13^ 9] +
-1.0 [13^ 13] +
0.85 [14^ 0] +
-0.85 [14^ 2] +
-0.85 [14^ 6] +
-0.85 [14^ 10] +
-1.0 [14^ 14] +
0.85 [15^ 1] +
-0.85 [15^ 3] +
-0.85 [15^ 7] +
-0.85 [15^ 11] +
-1.0 [15^ 15]
```

[8]:
```
g_current_limit = generate_two_orbital_nx(6,7)
g_ideal = generate_two_orbital_nx(20,20)

##### START UNCOMMENT FOR TESTING
#n_test = 2
#g_current_limit = generate_two_orbital_nx(n_test,n_test)
#g_ideal = generate_two_orbital_nx(n_test,n_test)
##### END UNCOMMENT FOR TESTING
n_qubits_current_limit = len(g_current_limit)
n_qubits_ideal = len(g_ideal)
```

Now that we have the Hamiltonians for both the model which constitutes the current limit of classical solvers, and the ideal capability of a solver, we can perform resource estimation for the Hamiltonians. As with the single qubit model, we need to get a decent state initialization using Hartree-Fock. As above, this has not been implemented at this time, but should have low depth in the quantum circuit since Hartree-Fock outputs a product state.

[9]:
```
#TODO: Incorporate Hartree-Fock into this section to prepare the initial state␣
 ↪for QPE for GSEE.
#This should provide a low depth initialization circuit relative to the depth␣
 ↪of the QPE, while giving access to a low-energy subspace
```

Assuming that we have the output from the Hartree-Fock simulation, we may now perform QPE as above. Currently we are using a short evolution time and a second order trotterization with a single step. We will use scaling arguments to determine the final resources since generating the full circuit for a large number of trotter steps with many bits of precision is quite costly.

[10]:
```
ham_current_limit =␣
 ↪nx_to_two_orbital_hamiltonian(g_current_limit,t1,t2,t3,t4,mu)
ham_ideal = nx_to_two_orbital_hamiltonian(g_ideal,t1,t2,t3,t4,mu)
trotter_order_current_limit = 2
trotter_steps_current_limit = 1

trotter_order_ideal = 2
trotter_steps_ideal = 1

bits_precision_ideal = 16
bits_precision_current_limit = 16

current_limit_args = {
    'trotterize' : True,
    'mol_ham'    : ham_current_limit,
    'ev_time'    : 1,
    'trot_ord'   : trotter_order_current_limit,
    'trot_num'   : 1
}

ideal_args = {
    'trotterize' : True,
    'mol_ham'    : ham_ideal,
    'ev_time'    : 1,
    'trot_ord'   : trotter_order_ideal,
    'trot_num'   : 1
}
```

[11]:
```
E_min_ideal = -len(ham_ideal.terms)
E_max_ideal = 0
ideal_omega = E_max_ideal-E_min_ideal
t_ideal = 2*np.pi/ideal_omega
ideal_phase_offset = E_max_ideal*t_ideal

E_min_current_limit = -len(ham_current_limit.terms)
E_max_current_limit = 0
limited_omega = E_max_current_limit-E_min_current_limit
limited_t = 2*np.pi/limited_omega
limited_phase_offset = E_max_current_limit*limited_t

init_state_ideal = [0] * n_qubits_ideal
init_state_current_limit = [0] * n_qubits_current_limit

print('starting')
t0 = time.perf_counter()
gse_inst_current_limit = PhaseEstimation(
    precision_order=1,
```

```
    init_state=init_state_current_limit,
    phase_offset=limited_phase_offset,
    include_classical_bits=False, # Do this so print to openqasm works
    kwargs=current_limit_args)
gse_inst_current_limit.generate_circuit()
t1 = time.perf_counter()
print(f'current limit time to generate high level: {t1 - t0}')

t0 = time.perf_counter()
gse_inst_ideal = PhaseEstimation(
    precision_order=1,
    init_state=init_state_ideal,
    phase_offset=ideal_phase_offset,
    include_classical_bits=False, # Do this so print to openqasm works
    kwargs=ideal_args)
gse_inst_ideal.generate_circuit()
t1 = time.perf_counter()
print(f'ideal time to generate high level: {t1 - t0}')

gse_circuit_ideal = gse_inst_ideal.pe_circuit
gse_circuit_current_limit = gse_inst_current_limit.pe_circuit
```

```
starting
current limit time to generate high level: 0.5769948088563979
ideal time to generate high level: 53.01192815299146
```

[12]:
```
print('Estimating Current Limit')
t0 = time.perf_counter()
circuit_estimate(circuit=gse_circuit_current_limit,
                 outdir='GSE/FermiHubbard/',
                 circuit_name='current_limit',
                 algo_name='GSE_Step',
                 write_circuits=True,
                 bits_precision=bits_precision_current_limit,
                 numsteps=trotter_steps_current_limit)
t1 = time.perf_counter()
print(f'Time to estimate Current Limit: {t1-t0}')

print('Estimating Ideal')
t0 = time.perf_counter()
circuit_estimate(circuit=gse_circuit_ideal,
                 outdir='GSE/FermiHubbard/',
                 circuit_name='ideal',
                 algo_name='GSE_Step',
                 write_circuits=True,
                 bits_precision=bits_precision_ideal,
                 numsteps=trotter_steps_ideal)
```

```
t1 = time.perf_counter()
print(f'Time to estimate Ideal: {t1-t0}')
```

```
Estimating Current Limit
   Time to decompose high level <class 'cirq.ops.common_gates.HPowGate circuit:
0.00028850999660789967 seconds
   Time to transform decomposed <class 'cirq.ops.common_gates.HPowGate circuit
to Clifford+T: 7.502199150621891e-05 seconds
   Time to decompose high level <class 'cirq.ops.identity.IdentityGate circuit:
0.00016000214964151382 seconds
   Time to transform decomposed <class 'cirq.ops.identity.IdentityGate circuit
to Clifford+T: 1.6140053048729897e-05 seconds
   Time to decompose high level <class
'pyLIQTR.PhaseEstimation.pe_gates.PhaseOffset circuit: 0.0008881441317498684
seconds
   Time to transform decomposed <class
'pyLIQTR.PhaseEstimation.pe_gates.PhaseOffset circuit to Clifford+T:
0.0006389578338712454 seconds
   Time to decompose high level <class
'pyLIQTR.PhaseEstimation.pe_gates.Trotter_Unitary circuit: 108.23273092904128
seconds
   Time to transform decomposed <class
'pyLIQTR.PhaseEstimation.pe_gates.Trotter_Unitary circuit to Clifford+T:
123.83342746202834 seconds
   Time to decompose high level <class
'cirq.ops.measurement_gate.MeasurementGate circuit: 0.17254893109202385 seconds
   Time to transform decomposed <class
'cirq.ops.measurement_gate.MeasurementGate circuit to Clifford+T:
0.00020829890854656696 seconds
Time to estimate Current Limit: 396.31939033186063
Estimating Ideal
   Time to decompose high level <class 'cirq.ops.common_gates.HPowGate circuit:
0.0002847590949386358 seconds
   Time to transform decomposed <class 'cirq.ops.common_gates.HPowGate circuit
to Clifford+T: 7.091090083122253e-05 seconds
   Time to decompose high level <class 'cirq.ops.identity.IdentityGate circuit:
0.00013261707499623 3 seconds
   Time to transform decomposed <class 'cirq.ops.identity.IdentityGate circuit
to Clifford+T: 1.4044111594557762e-05 seconds
   Time to decompose high level <class
'pyLIQTR.PhaseEstimation.pe_gates.PhaseOffset circuit: 0.0004379739984869957
seconds
   Time to transform decomposed <class
'pyLIQTR.PhaseEstimation.pe_gates.PhaseOffset circuit to Clifford+T:
0.00033449800685048103 seconds
   Time to decompose high level <class
'pyLIQTR.PhaseEstimation.pe_gates.Trotter_Unitary circuit: 19428.95841940795
```

```
seconds
   Time to transform decomposed <class
'pyLIQTR.PhaseEstimation.pe_gates.Trotter_Unitary circuit to Clifford+T:
18580.469520466868 seconds
   Time to decompose high level <class
'cirq.ops.measurement_gate.MeasurementGate circuit: 298.3902921839617 seconds
   Time to transform decomposed <class
'cirq.ops.measurement_gate.MeasurementGate circuit to Clifford+T:
0.01565385516732931 seconds
Time to estimate Ideal: 46334.01467376598
```

After Ground State Energy Estimation has been performed, we need to measure the autocorrelation, $\frac{1}{2}\langle\Delta_{i,j} + \Delta_{i,j}^{\dagger}\rangle$ where $\Delta_{i,j} = \langle c_{i+}c_{j-} - c_{i-}c_{j+}\rangle$. As above, for the case where $i = j = 0$, this can be performed in the following way. One can rotate into the Pauli basis using the following transformation, where for simplicity we assume that $i = j = 0$ is in the middle of the lattice. We will label the qubit representing site $i+$ as qubit 1 and the qubit representing site $i-$ as qubit 2. Given $U = CNOT_{i+,i-}H_{i+}CNOT_{i+,i-}$, it can be shown that $U^{\dagger}(c_{i+}c_{i-} - c_{i-}c_{i+})U = -2|\uparrow\uparrow\rangle\langle\uparrow\uparrow| + 2|\downarrow\downarrow\rangle\langle\downarrow\downarrow|$. This means that by simply applying the circuit U, then measuring, we can observe the autocorrelation. This introduces 3 Clifford operations and no T gates.

A similar operation, though acting on 4 qubits rather than 2 qubits, can be achieved in a similarly low constant circuit depth for the case where $i \neq j$.

[ ]: