Compilation for Fast Hamiltonian Simulation

Wenyan Guan, Shubh Agrawal, and Gianluca Delgado

June 6, 2020

Project Overview

In this project, we examined the problem of Hamiltonian simulation, which allows us to break down a Hamiltonian into a quantum circuit of elementary gates. The most established protocol in the field are the deterministic Trotter-Suzuki decompositions. Yet it has been shown that randomized compilers can achieve similarly satisfying approximation with lower computational costs.

We studied one specific randomized Hamiltonian simulation protocol, the qDRIFT protocol by Earl Campbell [1]. The qDRIFT protocol weights the probability each gate proportional to the corresponding interaction strength in the Hamiltonian. Such design can eliminates the dependence of the gate count on L, the number of terms in the Hamiltonian and thus achieve a speed up for various practical simulations for quantum chemistry problem [1].

Although we were not able to fully simulate the dynamics of hydrogen molecule due to limitation on computing power, we implemented the qDRIFT protocol in Q# and compared its performance to that of first order Trotter-Suzuki decomposition [2] with several tests. Specifically, we tested the two protocols on their gate counts and total runtime varying the precision requirement ϵ, terms in Hamiltonian decomposition L, and desired simulation time t.

The Hamiltonian Simulation Problem

We formulate the Hamiltonian simulation problem formally as below. Given a Hamiltonian H which decomposes into

$$H = \sum_{j=1}^{L} h_j H_j, \tag{1}$$

we would like to find a quantum circuit that approximate e^{itH} as a sequence of $e^{i\tau H_j}$ gates with some precision ε. Moreover, we want to minimize the cost of the simulation, i.e. the number of unitaries $e^{i\tau H_j}$ used. Note that here all H_j's are Hermitian and normalized.

The Trotter-Suzuki Decomposition

The most established methods to the Hamiltonian simulation problems are currently the Trotter-Suzuki decompositions. Given a matrix exponential over matrices A and B, the Trotter product formula

$$e^{A+B} = \lim_{N\to\infty} (e^{\frac{A}{N}} e^{\frac{B}{N}})^N \tag{2}$$

See code at this GitHub repository

allows us to transform a d-dimensional quantum system into the corresponding $(d+1)$-dimensional classical system [3]. Generally, the Trotter-Suzuki formula S_n gives the n^{th} order approximation to the exponential. Here we only consider the first order formula, which simply says

$$V_r := \prod_{j=1}^{L} e^{ith_j H_j/r} \quad \xrightarrow{r\to\infty} \quad U_r := e^{itH/r}. \tag{3}$$

It follows that $V_r^r \to U$ in large r limit because $U_r^r = U$. It has been shown by previous work [4] that the error for Trotter-Suzuki decomposition is bounded by

$$\epsilon = \frac{L^2\Lambda^2 t^2}{2r} e^{\frac{\Lambda t L}{r}}, \quad \Lambda := \max_{j=1,\ldots,L} h_j. \tag{4}$$

When the error ϵ is small, it is easy to see that the smallest r needed $\sim \frac{L^2\Lambda^2t^2}{2\epsilon}$, which brings the gate count in of the Trotter-Suzuki approximation to $N = Lr \sim \frac{L^3\Lambda^2t^2}{2\epsilon}$. When the order of decomposition is large, the gate count roughly scales with $\mathcal{O}\big(L^2\Lambda t\big)$ [1]. It has been noticed that there are ways to improve the L dependence in the gate count [4], but never below quadratic. This imposes great limitation on the efficiency of the protocol. In fact, in chemistry problems. L normally $\sim \mathcal{O}\big(n^4\big)$, where n is the system size, which results in an unpleasant $\mathcal{O}\big(n^8\big)$ scaling. And the qDRIFT protocol is proposed in hope to resolve such problem.

The qDRIFT Protocol

The qDRIFT protocol leverages the power of randomized sampling to reduce the dependency on the number of terms in the Hamiltonian in the compilation of the simulation circuit. The protocol employs randomization using *an oracle* (denoted SAMPLE) to sample from the set of unitaries that compose the full Hamiltonian. It achieves the desired precision by introducing biases into the probability distribution employed by SAMPLE, which is correlated to the strength of the unitaries in the full Hamiltonian expression:

$$p_j = \frac{|h_j|}{\sum_j |h_j|}, \tag{5}$$

where p_j is probability of SAMPLE returning the unitary H_j. The desired precision ϵ, the simulation time t, and the terms in decomposition L together determine the number of times N that SAMPLE is called. Specifically, we may approximate N with

$$N \approx \left\lceil \frac{2\lambda^2 t^2}{\epsilon} \right\rceil \tag{6}$$

where $\lambda = \sum_j |h_j|$.

Each time SAMPLE is called the unitary $e^{-i\tau H_j}$, where $\tau = \frac{t\lambda}{N}$, is appended to the list of unitaries to be performed in the final circuit. Here, $\frac{t}{N}$ can be thought of the time step taken ,over which the unitary is applied, in analogy to the Trotter-Suzuki decomposition; λ is the strength of H_j over that time step. We eventually obtain a final circuit with N gates, where the strength of each gate is encapsulated in the number of times it appears in the circuit rather than the factor it appears along side in the exponential (which is uniform for all the gates). Thus, the length the circuit no longer explicitly depends on the number of terms in the hamiltonian decomposition L. One might argue that since N depends on λ, N does depend on L indirectly, yet it can be

demonstrated as in [1], that for practical applications N scales like $\mathcal{O}(L)$, which is a significant improvement from the $\mathcal{O}(L^2)$ barrier observed in the Trotter-Suzuki decompositions.

The advantage qDRIFT has over higher-order trotter methods is only observable over short simulation times as qDRIFT scales like $\mathcal{O}(t^2)$ while higher order trotter methods scale like $\mathcal{O}(t)$. Thus, qDRIFT holds the most promise in simulating the dynamics of systems over a short time period with a large number of terms in their hamiltonian decompositions.

qDRIFT Implementation

Implement the qDRIFT protocol with the following procedures:

1. Sum over the $|h_j|$ values of the decomposed Hamiltonian
2. Calculate N which determines the strength of each gate in the resulting circuit
3. Use the SAMPLE oracle to sample N times from the set of H_j unitaries with corresponding probability $|h_j|$ (suitably normalized)
4. Append each sampled unitary to the final list of gates V

Since qDRIFT does not make use of qubits to generate the circuit, we decided to implement it in Python, see `qDrift` in `host.py`. The function return the list of gates V and gate count N to pass on to the Q# program, `Operations.qs`, which then perform the simulation.

The function performing qDRIFT takes in three required (the index used to locate the Hamiltonian decomposition coefficients, the simulation time, and the desired precision) and one optional variable (the number of terms from the Hamiltonian to be used, default is set to full length of decomposition). The function then calculates $\lambda = \sum_j |h_j|$, the number of gates N, and the probability distribution for SAMPLE() in constructing the final list. An empty list V is created and a for loop is used to call SAMPLE() for N times and append a tuple of the return value from SAMPLE() and λ to V each time. Here, we define SAMPLE() with `numpy.random.choice` which allows one to specify the set that is to be sampled from and the corresponding probabilities for each member of the set. The set we chose to sample from was the range of integers from zero to the number of terms the user specified to be used. These will serve as indices Q# could then use to select the corresponding hamiltonian terms from an array specified within the Q# program. Once the for loop has completed its execution the qDRIFT function returns the list V along with the integer N.

Implementing Hamiltonian Simulation

Once the qDRIFT function has returned the list V and N, these values are passed on to the Q# function `sim_ham` which takes in a list of integer-double tuples which are the index of the local hamiltonian term to be applied and its corresponding strength respectively, an integer that represents how many time steps are to be performed (N), and the total simulation time (t). `sim_ham` then proceeds to initialize two qubits and for each element (j, λ_j) in V it applies the transformation $e^{i\frac{\lambda_j t}{N}H_j}$ to the two qubits. Note this construction of the `sim_ham` function allows one to have a different strength λ_j for each j which is taken advantage of when performing the trotter-suzuki decomposition since the strength of each H_j in that case is h_j which varies from term to term. For the purposes of this project the local hamiltonian terms correspond to the dynamics of the electronic structure of H2 and are hard coded at the beginning of `sim_ham`.

Testing and Results

In order to test the validity of our qDRIFT implementation, we compare the time it takes for qDRIFT to compile the gates and simulate the evolution to the time it takes the First Order Trotter-Suzuki method to accomplish the same task given the same simulation parameters (e.g. simulation time, target precision, Hamiltonian decomposition, etc.). We chose three metrics by which we compare the performance of qDRIFT and First Order Trotter: precision of result, number of terms, and simulation runtime.

All three types of test can be run through `host.py` by using different command line arguments. The test code runs on an inputted set of Hamiltonian Decomposition Coefficients for particular H_2 bond lengths randomly chosen for a hard-coded list of such coefficients `H2Coeff`, which we obtained from this paper [5]. The values in `H2IdentityCoeff` and `H2Terms` arrays are the corresponding Identity Transformation coefficients and 2-Local gates (0 = `Identity`, 1 = `PauliX`, 2 = `PauliY`, 3 = `PauliZ`) respectively.

The different tests along with the necessary details on running them are listed below. The code returns Usage statements if run incorrectly.

- **Precision Comparison Test.** Run `host.py` with command `python host.py 0`.

  Code selects a random index `bond_idx` representing the H_2 parameters. With values for logarithm of precision ranging from 0 to 2 (spacing of 0.05) and a simulation time of 1 second, simulations are run for this H_2 parameter set using both qDRIFT and Trotter-Suzuki protocols. The number of gates used (given by the length of `V`) and the time taken (using the `time` module) for either protocol at each log-precision value are recorded. Two plots `e_precision_time_comp.png` and `e_precision_gates_comp.png` are obtained from both these lists of simulation parameters.

- **Number Comparison Test.** Run `host.py` with command `python host.py 1`.

  Code selects a random index `bond_idx` representing the H_2 parameters. With values for the number of terms in the Hamiltonian ranging from 1 to 6 (inclusive, spacing of 1) and a simulation time of 1 second, simulations are run for this H_2 parameter set using both qDRIFT and Trotter-Suzuki protocols. The number of gates used (given by the length of `V`) and the time taken (using the `time` module) for either protocol at each Hamiltonian count are recorded. Two plots `num_terms_time_comp.png` and `num_terms_gates_comp.png` are obtained from both these lists of simulation.

- **Simulation Time Comparison Test.** Run `host.py` with command `python host.py 2`.

  Code selects a random index `bond_idx` representing the H_2 parameters. With values for the simulation time period ranging from 1 to 6 (100 points) and a log-precision of 0.1, simulations are run for this H_2 parameter set using both qDRIFT and Trotter-Suzuki protocols. The number of gates used (given by the length of `V`) and the time taken (using the `time` module) for either protocol at each Hamiltonian count are recorded. Two plots `sim_time_time_comp.png` and `sim_time_gates_comp.png` are obtained from both these lists of simulation.

These three tests were decided on for a variety of reasons. The precision test was implemented since the Campbell paper that introduces qDRIFT mostly neglects to mention how varying the precision would comparatively affect the qDRIFT and Trotter methods so it would be interesting to make such a comparison and ensure run time and gate count behave as expected given how the desired precision is used in the compilation process in both methods. Varying the simulation

time also proved to be valuable as Campbell does a very similar analysis in his paper for much more complicated systems so achieving similar results on the simpler system used in this analysis should give a good indication as to whether the implementations are behaving as expected. Finally, varying the number of terms in the hamiltonian decomposition allows for the testing of Campbell's claim that qDRIFT scales much better with this variable than the Trotter methods do and this advantage should be exacerbated when comparing qDRIFT to first order Trotter.

All three tests were run; the obtained plots (also included in the Plots of the GitHub repository) are included and analyzed below.

Plots with respect to precision

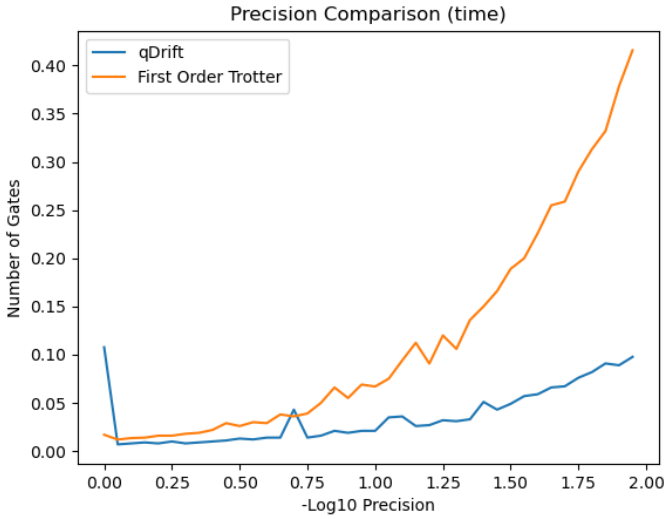

Figure 1: Linear-log space plot of runtime with respect to the precision limit of the simulation. Note that the runtime for qDRIFT grows slower than the Trotter method with respect to precision.

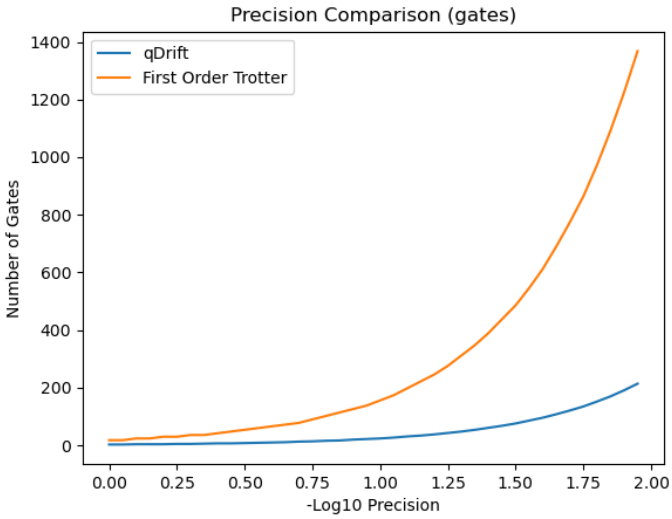

Figure 2: Linear-log space plot of the number of quantum gates needed for the precision limit of the simulation. Note that the number of gates, which represents the circuit complexity, grows slower for qDRIFT than for Trotter with respect to precision. The curve is also much smoother than that for the runtime, as expected, due to less noise associated with the deterministic process for choosing gates.

Plots with respect to number of Hamiltonian terms

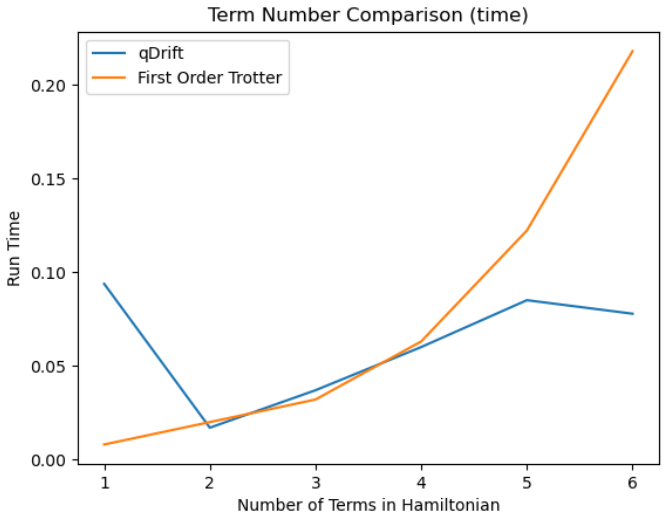

Figure 3: Linear space plot of the runtime with respect to the number of terms in the Hamiltonian being simulated. Note that the runtime for qDRIFT grows slower than the runtime for Trotter with respect to the Hamiltonian length, though it is much less observable than in the other two tests due to the smaller range of independent variable (number of terms) values.

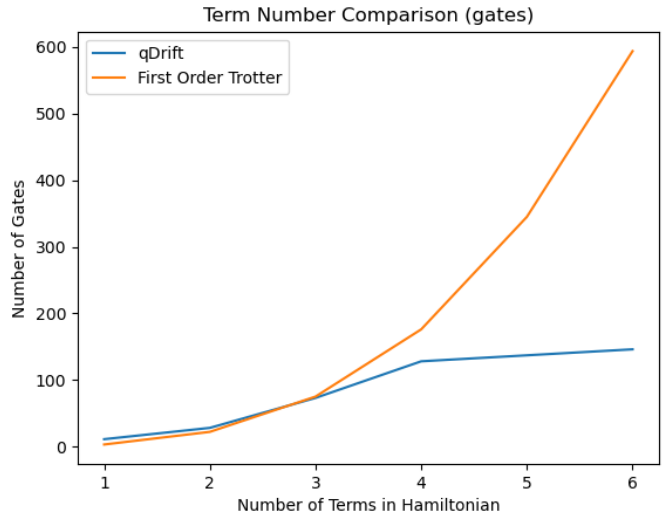

Figure 4: Linear space plot of the number of gates needed (representing circuit complexity) with respect to the number of terms in the Hamiltonian being simulated. Note that the complexity for qDRIFT seems to grow slower than the complexity for Trotter with respect to Hamiltonian length.

Plots with respect to simulation time period

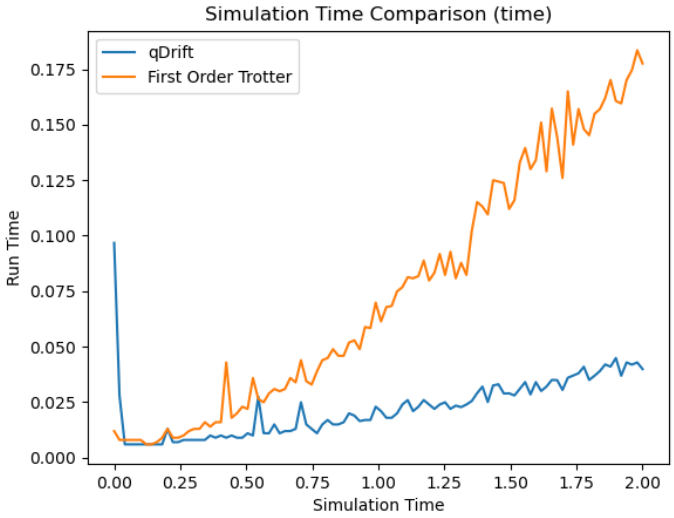

Figure 5: Linear space plot of the runtime with respect to the time period for which the system is simulated. Note that the runtime for qDRIFT again grows slower than Trotter with respect to simulation time. There is much more noise associated with the data due to the more dense range of independent variable.

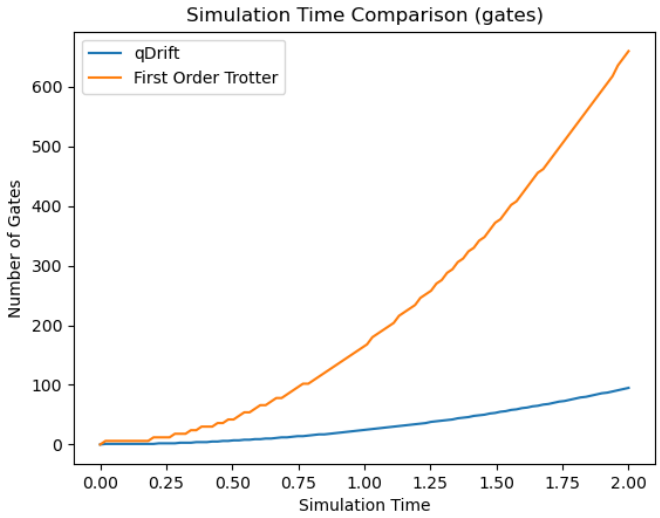

Figure 6: Linear space plot of the number of terms in the circuit with respect to the time period for which the system is simulated. Note that this circuit complexity grows slower for qDRIFT than the Trotter with respect to simulation time. The curve is also much smoother than that for the runtime, as expected, due to less noise associated with the deterministic process for choosing gates.

Summary

In general, it is noted from all three tests than both the time complexity and circuit complexity scale faster (with respect to any of the initial state parameters) for the first order Trotter than for the qDRIFT protocol.

References

[1] E. Campbell, "Random compiler for fast hamiltonian simulation," *Phys. Rev. Lett.*, vol. 123, p. 070503, Aug 2019.

[2] nathanwiebe2, "Simulating hamiltonian dynamics - microsoft quantum."

[3] M. Suzuki, "Quantum monte carlo methods — recent developments," *Physica A: Statistical Mechanics and its Applications*, vol. 194, no. 1, pp. 432 – 449, 1993.

[4] A. M. Childs, D. Maslov, Y. Nam, N. J. Ross, and Y. Su, "Toward the first quantum simulation with quantum speedup," *Proceedings of the National Academy of Sciences*, vol. 115, no. 38, pp. 9456–9461, 2018.

[5] P. J. J. O'Malley, R. Babbush, I. D. Kivlichan, J. Romero, J. R. McClean, R. Barends, J. Kelly, P. Roushan, A. Tranter, N. Ding, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, A. G. Fowler, E. Jeffrey, E. Lucero, A. Megrant, J. Y. Mutus, M. Neeley, C. Neill, C. Quintana, D. Sank, A. Vainsencher, J. Wenner, T. C. White, P. V. Coveney, P. J. Love, H. Neven, A. Aspuru-Guzik, and J. M. Martinis, "Scalable quantum simulation of molecular energies," *Phys. Rev. X*, vol. 6, p. 031007, Jul 2016.