Self-reflection Report for Team Coastal Alliance

Shuohao Ping[1], Kecheng Liu[2], Qingyue Jiao[3], Xiangyu Gao[4], Yuwei Jin[5], and Zirui Li[5]

[1]Department of Physics and Astronomy, University of California-Los Angeles
[2]Center on Frontiers of Computing Studies, Peking University
[3]Department of Computer Science, Columbia University
[4]New York University
[5]Rutgers Univeristy

The rapid evolution of drug discovery is continuously fueled by cutting-edge technologies, especially in understanding complex molecular interactions pivotal to medical therapies. Quantum computing, complemented by machine learning techniques, promises to redefine drug discovery by offering solutions previously deemed insurmountable for classical computers. At the epicenter of these advancements lies the hydroxyl cation (·OH), a molecule of profound pharmacological significance due to its central role in numerous drug interactions and its links to various health conditions like neurodegenerative disorders, cardiovascular diseases, and cancers. Leveraging the Variational Quantum Eigensolver (VQE) with the UCCSD ansatz, our study introduces advanced methodologies like Tetris frameworks and Pauli string pruning to address the hydroxyl cation's ground state energy problem. The integration of quantum computing and machine learning not only enhances our ability to predict and understand molecular behaviors with unprecedented accuracy but also paves the way for groundbreaking therapeutic developments.

1 Introduction

The contemporary landscape of drug analysis and discovery is undergoing a transformative evolution, propelled by the fusion of quantum computing methodologies with traditional pharmacological studies [2]. At the nexus of this evolution lies the intricate challenge of understanding and predicting molecular interactions, specifically those as critical as the hydroxyl cation (.OH). Hydroxyl cation, beyond its conventional understanding in the biochemical domain, holds profound implications for a multitude of therapeutic applications. However, delving into its core properties, such as the ground state energy, demands computational prowess beyond the capabilities of classical computing paradigms.

Enter the realm of quantum computing, a frontier that promises not just enhanced computational speed but also an unprecedented depth of analysis.

To address the ground state energy problem of the hydroxyl cation (.OH) using the Variational Quantum Eigensolver (VQE) [5], we employ the UCCSD ansatz due to its superior accuracy compared to other ansatz types. In our project, the VQE implementation encompasses four key components: circuit synthesis for the UCCSD ansatz, qubit mapping, circuit measurements across various bases, and parameter optimization through machine learning.

To begin, we have adapted the state-of-the-art VQE circuit compilation framework known as Paulihedral[4]. This adaptation enables the transformation of the abstract Pauli string representation into a gate-based circuit, as briefly outlined in Section 2.1.1. Additionally, we introduce a novel intermediate Pauli representation (IR) referred to as 'Tetris' to capture gate cancellation opportunities that might be overlooked by Paulihedral, as explained in Section 2.1.2. To address the challenges associated with hardware connectivity, Tetris is equipped with a unique compiler, the details of which are provided in Section 3.

Furthermore, in order to minimize the number of shots and resource usage, we implement the measurement grouping concept introduced in Varswa[3], with further details provided in Section 2.2.

Finally, in our pursuit of further circuit duration reduction, we introduce a Pauli string pruning method designed to eliminate non-essential Pauli strings. Within the UCCSD ansatz, each Pauli string is accompanied by a coefficient. Pauli strings with smaller coefficients are more likely to be removed without compromising final accuracy. For a more detailed explanation, please refer to Section 2.3. Please note that circuit parameter tuning is not covered in this report. We rely on the default QISKIT parameter tuning method, COBYLA, to tune the parameters and present them directly in the compiled circuit.

2 Opportunities

2.1 Circuit Synthesis

2.1.1 Basic Circuit Synthesis

To synthesize the quantum circuit for the given Hamiltonian, we employ the circuit synthesis method outlined in Paulihedral[4]. As described in Paulihedral, each Pauli string can be synthesized as a subcircuit with a tree structure. For instance, the Pauli string $I_4Z_3Y_2X_1X_0$ can be synthesized into two distinct circuits, as illustrated in Figure 1. Paulihedral introduced two optimizations. Firstly, it proposed a scheduler to enhance circuit parallelism by scheduling two Pauli strings together when they do not share logical qubits. Secondly, Paulihedral performs permutations on the order of Pauli strings to schedule those with high similarity together. This increases the opportunity for gate cancellation between consecutive Pauli strings. For example, by employing this approach, we can reduce the circuit to at most four CNOT gates, and cancel

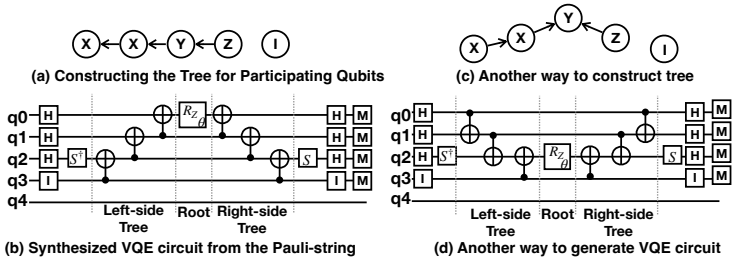

Figure 1: Two different VQE circuit constructions for the same Pauli string $I_4Z_3Y_2X_1X_0$. Note that I operator is an identity matrix. It does nothing, like a NOP in classical computer.

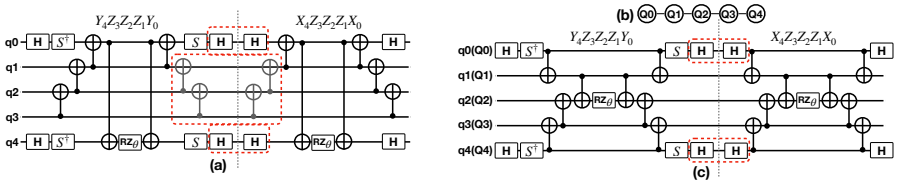

Figure 2: Gate Cancellation and limitation from the hardware connectivity. (a) The synthesized circuit for two consecutive Pauli strings, achieving the maximum number of gate cancellations. (b) The underlying hardware coupling graph. (c) Compiled circuit using hardware-efficient synthesis that does not include extra SWAP gates. Only four single-qubit gates are canceled.

two single-qubit gates in the case of the Pauli strings $YZZZY$ and $XZZZX$, as depicted in Figure 2(a).

2.1.2 Circuit Synthesis Optimization

In the process of implementing a quantum circuit, it's essential to map the logical circuit onto the hardware while addressing the connectivity constraints. This often involves the insertion of additional SWAP gates. The flexibility of circuit construction for a Pauli string allows us to create a hardware-efficient circuit that requires fewer SWAP gates for each Pauli string. For example, when considering the same two Pauli strings as shown in Figure 2(a), our goal is to execute the corresponding circuit on the hardware as depicted in Figure 2(b). The resulting hardware-efficient circuit is presented in Figure 2(c). This circuit can be executed on the given hardware without the need for additional SWAP gates. However, it's worth noting that this hardware-efficient circuit may limit the opportunity for CNOT gate cancellation.

To retain the capability of gate cancellation while simultaneously minimizing the cost of circuit compilation, we introduce a new Pauli Intermediate Representation (IR), referred to as *Tetris*. A Tetris symbolizes a group of Pauli strings

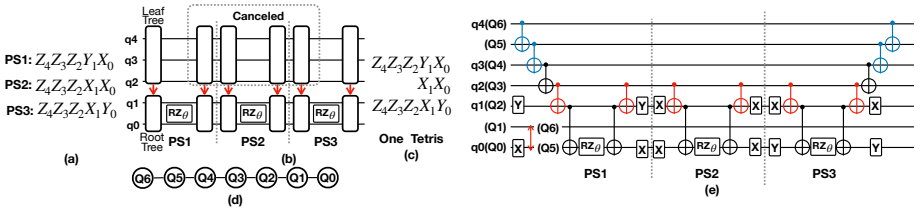

Figure 3: New Pauli IR. (a) A group of Pauli strings. (b) Gates are canceled on the leaf trees among three Pauli strings. (c) An illustration of new IR. (d) The underlying hardware coupling graph. (e) The compiled circuit of new IR with a SWAP gate and bridge CNOT gates. The single qubit gates are represented by the corresponding Pauli operators.

that share some common Pauli operators across a subset of qubits. The design of UCCSDs makes the similarity between two Pauli strings high and provides the opportunity for Tetris to include multiple Pauli strings.

Within a Tetris, each Pauli string still upholds a tree structure on either side of the root qubit. On each side, the original tree is bifurcated into two components: the leaf tree and the root tree, linked by a directional edge that connects the two subtrees. This edge represents a CNOT gate with the control qubit coming from the leaf tree and the target qubit from the root tree. The leaf tree comprises qubits that share the same Pauli operators across the Pauli strings, while the root tree houses the remaining qubits. For instance, in the three Pauli strings shown in Fig. 3(a), qubits q2, q3, and q4 share the same Pauli operators, Pauli-Z; hence, the leaf tree for each Pauli string incorporates those qubits. The root qubit of a single Pauli string is selected from the root tree. The generation procedure of this structure is shown in Fig.3(b), and an illustration of this structure can be found in Fig.3(c).

Due to gate dependencies, the cancellation of CNOT gates between two Pauli strings must proceed from the gates on the leaf qubits to the gates on qubits that are closer to the root qubit. For example, four CNOT gates are canceled from the leaf qubits as shown in Fig. 2(a). Within the Tetris structure, as long as the leaf trees have identical structures, gates on the leaf trees can be canceled. Simultaneously, Tetris maintains the high-level Pauli string abstraction, facilitating circuit synthesis that takes into account the underlying hardware connectivity. Fig. 3(e) details the compiled circuit for the Pauli strings in Fig. 3(a) in relation to the hardware in Fig. 3(d). There are eight CNOT gates canceled from those leaf trees.

2.2 Measurement Grouping

To reduce the number of shot, we adapt the measurement grouping strategy from Varsaw [**?**]. The authors of [**?**] proposed using BronKerbosch algorithm to minimize the number of measurement needed for calculating Hamiltonian. We will first build a connected graph G using a given Hamiltonian. The nodes

Figure 4: An example of connected graph that represents commutativity of the Pauli terms for a Hamiltonian. In this example, the Hamiltonian has 6 terms: ZIX, IYI, XII, XZX, XZI, IIZ. These terms are presented by nodes of the graph. If two Pauli terms commute, we add an edge between the two nodes, like IIZ and IYI. Note that we use the simplest commutativity rule for this graph, which is operations on different qubits commute to each other.

of the graph G are the Pauli terms in the Hamiltonian, and we will add a edge between two terms if both terms commute to each other. Hence, graph G represents commutativity between all Pauli terms.

After we generate the graph, we transform the problem of grouping Pauli strings to a max-clique problem. The goal is to find all clique with maximum number of nodes, and each clique represents a group of Pauli strings that can be measured simultaneously since a clique is defined as a complete subgraph, and in our case, a complete subgraph means all Pauli terms in the subgraph commute to each other, which means we can group all terms in a single measurement. The BronKerbosch algorithm is used to enumerate all max-clique in the graph with an complexity of $3^{n/3}$ where n is number of nodes in the graph according to [6]. In the example of figure 4, the max clique has a size of 3. But the max clique is not unique. For example, IIZ, IYI, and XII forms a max clique, and XZI, XII, XZX can also form a max clique. After we find all max clique, we will remove nodes within all max cliques and corresponding edges from the graph and repeat the same process until all nodes are removed from the graph. Note that since the max clique is not unique, the result of our grouping is also not unique, which means we can get different groupings each time we run the algorithm. Also, the choice of the max clique can have an influence on the later iteration of clique finding. A bad choice of max clique can lead to higher number of measurements since we will remove edges associated with nodes in the max clique, and this can make some complete subgraph disconnected or non complete. In our case, we didn't address this problem directly, and we will get different numbers of grouping terms, ranging from 138 to 145 terms. Furthermore, when we build the commutativity graph, we always ignore the Pauli string with I operation on every qubit because it commute to all other terms, and it always have a fixed expectation value, which means we don't need to do any measurement for this term.

2.3 Pauli String Pruning

Particularly in quantum simulations and quantum chemistry algorithms, Pauli strings play a pivotal role in representing quantum operations and observables. However, as quantum systems grow in size and complexity, the number of Pauli strings in quantum circuits can become exorbitantly large, leading to increased circuit depth and duration. Pauli string pruning emerges as a technique to address this challenge by systematically removing certain Pauli strings, thereby reducing the computational overhead without significantly compromising the accuracy of the quantum computation. Especially those geared towards quantum chemistry like the VQE, often require the evaluation of expectation values of certain observables. These observables are usually represented as linear combinations of tensor products of Pauli matrices, known as Pauli strings. Each Pauli string is associated with a coefficient, which represents its weight or importance in the computation.

The fundamental idea behind Pauli string pruning is rooted in the observation that not all Pauli strings contribute equally to the outcome of a quantum computation. In many cases, certain Pauli strings, especially those with small associated coefficients, have negligible impact on the final result. By identifying and eliminating these "non-essential" Pauli strings, the quantum circuit's depth and width can be significantly reduced. The number of required quantum gates diminishes, leading to quicker computations. The chances of errors introduced by gate operations decrease due to the reduced circuit size. The primary criterion for pruning is the magnitude of the coefficient associated with a Pauli string. Pauli strings with coefficients below a certain threshold are considered non-essential and are potential candidates for pruning. This threshold is typically determined empirically, ensuring that the removal of Pauli strings does not adversely affect the accuracy of the computation. Moreover, additional considerations might include the overall contribution of a Pauli string to the variance of the computation and the computational cost associated with measuring a particular Pauli string.

One of the primary advantages of pruning non-essential Pauli strings is the significant reduction in the quantum circuit's depth. As the circuit becomes shallower, the time required to execute it on a quantum processor decreases. And by removing certain Pauli strings, the total number of quantum gates in the circuit can be reduced. This not only speeds up computation but also simplifies the overall structure of the quantum circuit. Pruning can lead to circuits that require fewer qubits. This is especially valuable given the limited number of qubits in near-term quantum devices. In algorithms like VQE, where the expectation value of certain observables is required, pruning can reduce the number of different Pauli string measurements. This translates to fewer circuit runs and thus conserves computational resources. Quantum gates are not perfect and introduce errors. By decreasing the total number of gates through pruning, the cumulative errors introduced by these gates can be minimized. Depending on the specific requirements of a computation, the pruning threshold can be adjusted. If a high degree of accuracy is not paramount, more aggressive

pruning can be employed for faster results. Conversely, if accuracy is vital, only minimal pruning can be applied.

While pruning reduces computational overhead, there's always a trade-off with accuracy. It's crucial to ensure that the pruned Pauli strings indeed have a minimal impact on the final result. Determining the threshold below which Pauli strings can be pruned is often empirical and might require iterative testing. Pauli string pruning stands out as an indispensable technique for optimizing quantum circuits, especially in quantum chemistry simulations. While the approach offers significant advantages in terms of computational efficiency, care must be taken to balance the trade-off between efficiency and accuracy. As quantum computing matures, techniques like Pauli string pruning will be instrumental in harnessing the full power of quantum hardware.

3 Methodology

3.1 New IR Generation for Gate Cancellation

Once the Pauli strings are appropriately arranged, it is worth discussing the size of one Tetris. A Teris consists of a sequence of Pauli strings, as the example shown in Fig. 3. It separates each tree of a Pauli string into two subtrees. The leaf tree only contains qubits sharing the same Pauli operators across Pauli strings in the Tetris and the root tree contains the remaining qubits, including the root qubit. All leaf trees could be canceled in the logical circuit, except the first one and the last one. If we include more Pauli strings in a Tetris, more gate cancellation is guaranteed.

However, increasing the size of a Tetris can amplify discrepancies among Pauli strings. This can lead to an enlarged root tree, which is counterproductive for gate cancellation.

Since we use the Jordan-Wigner transformation, the UCCSD unitary can be written in terms of Pauli matrices as follows (for more details see [1])

$$\begin{aligned} U(\theta) = \prod_{p>r} exp\{\theta_{pr} \hat{c_p}^\dagger \hat{c_r} - H.c.\} \\ \times \prod_{p>q>r>s} exp\{\theta_{pqrs} \hat{c_p}^\dagger \hat{c_q}^\dagger \hat{c_r} \hat{c_s} - H.c.\} \end{aligned} \tag{1}$$

Referring to Equation 1, the UCCSD ansatz comprises blocks of single excitation operators and blocks of double excitation operators. Within each block, Pauli strings feature Pauli-Z operators on the same qubits, with the Pauli-Z operator being the dominant one. Consequently, we treat each block of Pauli strings as a single 'Tetris' unit and aim to maximize gate cancellation among these Pauli-Z operators.

3.2 Circuit Synthesis with Respect to Hardware

The process of circuit synthesis for a Tetris differs from that of a Pauli string. Due to the division within the logical tree, the connectivity constraints of the root tree must be addressed earlier, even though its corresponding circuit is synthesized later. Prioritizing the synthesis of the circuit for the leaf tree is a logical step. This approach ensures that gate cancellations between successive leaf trees remain unaffected by the root tree.

To guarantee the cancellation between two leaf trees, one way is to make sure all data qubits in the root tree are mapped together, such that the mapping of those qubits would not be moved and avoid interrupting the gate cancellation between leaf trees. For the efficient grouping of all root-tree qubits, we first identify a central point within the hardware coupling graph, $G_{coupling}$, among the qubits $\pi(q_i^r)$. Here, π represents the current qubit mapping, and q^r denotes the set of qubits in the root tree. Subsequently, we introduce SWAP gates to cluster all qubits q_i^r around this center. It's worth noting that the central point doesn't necessarily have to be mapped by a data qubit q_i^r, as a data qubit will eventually be swapped into this position.

Next, we proceed with the synthesis of the circuit for the leaf tree. Initially, we organize the qubits q_j^l in the leaf tree based on their proximity to the root tree. Following this, we introduce SWAP gates to shift the qubit q_j^l that has a smaller distance to a root tree qubit. This process continues until all q^l qubits are linked to the root tree. Subsequently, as we synthesize the circuit for the leaf tree, we incorporate the CNOT gate. The qubit further from the root tree is selected as the control qubit, while the one closer to the root serves as the target. This procedure is reiterated until CNOT gates have been applied to all q_j^l qubits. The same steps are then applied to q^r. The corresponding pseudocode is detailed in Algorithm 1.

4 Implementation

We have all the code to evaluate our ansatz in `evaluation.ipynb`.

Firstly we analyse the total number of shots.

We iterate all the 10 seeds. For each seed we have 3 results, corresponding to 3 noise models.

At last, we evaluate the circuit duration on 3 system models.

5 Result

- Ansatz: What we found interesting is that all our ansatz are just 8 X gates to generate Hartree Fock initial state and that's enough. Initially, we have a complete UCCSD ansatz with all zero parameters as initial parameters, and we use Tetris to make the circuit depth shorter. Then we pass our ansatz as a function to the SLSQP optimizer. After a certain number of iterations, we prune some parts of the parameters until we reach the final

Algorithm 1 Circuit Synthesis

procedure CIRCUIT_SYNTHESIS($G_{coupling}$, π, q^r,q^l)
  $circuit = \{\}$
  $tree = \{\}$
  // Find a center and the shortest path from a q_i^r to the center.
  $center$, $paths$ = findCenter($G_{coupling}$, π, q^r)
  for $q_i^r \in q^r$ **do**
    circuit.insert(SWAP($\pi(q_i^r)$, paths($\pi(q_i^r)$)))
    tree.insert(q_i^r)
  end for
  // Find the shortest path from a q_j^l to q^r
  $paths$ = findPath($G_{coupling}$, π, q^r, q^r)
  for $q_j^l \in q^l$ **do**
    circuit.insert(SWAP($\pi(q_j^l)$, paths($\pi(q_j^l)$)))
    tree.insert(q_i^r)))
  end for
  for leaf in tree **do**
    circuit.insert(CNOT(leaf, tree(leaf).getparent))
    tree.remove(leaf)
  end for
  return *circuit*
end procedure

pruning ratio. We found that we can still have the same accuracy even if we increase the pruning ratio to 100%, which means there are no parameters remaining. So our submitted ansatz are a UCCSD ansatz optimized by our Tetris compilation framework but with all the parameters pruned away.

- Ground State Energy Estimation Accuracy: should be around 98%
- Quantum Resource Consumption
  - Total Number of Shots of Quantum Circuits: 568000
  - Circuit Size (Duration): 160 on FakeMontreal, 160 on FakeKolkata, 96 on FakeCairo.

References

[1] Panagiotis Kl. Barkoutsos, Jerome F. Gonthier, Igor Sokolov, Nikolaj Moll, Gian Salis, Andreas Fuhrer, Marc Ganzhorn, Daniel J. Egger, Matthias Troyer, Antonio Mezzacapo, Stefan Filipp, and Ivano Tavernelli. Quantum algorithms for electronic structure calculations: Particle-hole hamiltonian and optimized wave-function expansions. *Physical Review A*, 98(2), aug 2018.

[2] Yudong Cao, Jhonathan Romero, and Alán Aspuru-Guzik. Potential of quantum computing for drug discovery. *IBM Journal of Research and Development*, 62(6):6–1, 2018.

[3] Siddharth Dangwal, Gokul Subramanian Ravi, Poulami Das, Kaitlin N. Smith, Jonathan M. Baker, and Frederic T. Chong. Varsaw: Application-tailored measurement error mitigation for variational quantum algorithms, 2023.

[4] Gushu Li, Anbang Wu, Yunong Shi, Ali Javadi-Abhari, Yufei Ding, and Yuan Xie. Paulihedral: A generalized block-wise compiler optimization framework for quantum simulation kernels, 2021.

[5] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[6] Akira; Takahashi Haruhisa Tomita, Etsuji; Tanaka. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1), jul 2006.