Vol.2022-QS-5 No.28
2022/3/25

Parallel grouping algorithm for a large set of Pauli operators

Ikko Hamamura[1,a)] Takashi Imamichi[1] Hiroshi Horii[1] Jun Doi[1] Nobuyuki Yoshioka[2]
Seetharami Seelam[3] Takahiro Sagawa[2] Antonio Mezzacapo[3]

Abstract: Estimating the energy of a Hamiltonian is one of the most important tasks and frequently addressed in a quantum-classical hybrid algorithm like VQE (Variational Quantum Eigensolver). Grouping of Pauli operators plays a key role to improve efficiency of the energy estimation taking advantage of simultaneous measurements. However, the grouping of large molecules is difficult in terms of both computation time and memory space required. We propose a parallelized algorithm of grouping to address this issue. We succeeded to compute grouping of a large set of Pauli operators, which could not be computed before.

Keywords: Joint measurements, VQE (Variational Quantum Eigensolver)

1. Background

VQE (Variational Quantum Eigensolver) has been extensively studied in the past few years. In particular, significant progress has been made in improving the efficiency of the estimation of quantum observables [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12].

Quantum observables are given by the following form:

$$A = \sum_{i=1}^{n} a_i P_i, \tag{1}$$

where a_i are real coefficients and $P_i \in \{ I, X, Y, Z \}^{\otimes N}$ are N qubits tensor products of Pauli operators. Note that tensor products of Pauli operators are called Pauli strings.

Let us consider computing an expectation value $\langle A \rangle = \left\langle \sum_{i=1}^{n} a_i P_i \right\rangle$ for a given quantum observable A using a quantum computer. There are various methods to compute this expectation value and the most commonly used method is to decompose using linearity and measure the expectation value of each Pauli string. In other words, expectation values $\langle P_i \rangle$ can be estimated using a quantum computer, and the expectation value $\langle A \rangle = \sum_{i=1}^{n} a_i \langle P_i \rangle$ can be obtained from its linear combination. The expectation value of the Pauli string $\langle P_i \rangle$ can be estimated by unitary pre-rotation and sampling. For example, the measurement for Pauli Z does not require a pre-unitary rotation, but the measurement for Pauli X uses Hadamard gates to transform the basis, and the measurement for Pauli Y uses $S^\dagger$ gates and Hadamard gates to transform the basis.

We note that some groups of Pauli strings can be measured simultaneously. A measurement using single qubit unitary pre-rotation is called a TPB (Tensor Product Basis) measurement. The TPB measurements were used for joint measurements when estimating the expectation value [13]. For example, Pauli strings $ZXZI$ and $ZIZX$ in Figure 1 are jointly measurable with TPB measurements since they can be estimated from a measurement for a Pauli $ZXZX$. The important point here is that we need to transform quantum observables into groups of Pauli strings that are mutually jointly measurable using TPB measurements. This problem is called *the grouping problem*, which corresponds to the vertex coloring problem of Pauli graphs, and is known to be NP-hard [1]. A *Pauli graph* of Pauli strings with TPB measurements is defined as the complement graph of a graph with the Pauli strings

[1] IBM Quantum, IBM Research – Tokyo, Chuo-ku, Tokyo, 103-8510, Japan
[2] Department of Applied Physics, University of Tokyo, 7-3-1 Hongo, Bunkyo-Ku, Tokyo 113-8656, Japan
[3] IBM Quantum, T. J. Watson Research Center, Yorktown Heights, New York 10598, USA
[a)] ikko.hamamura1@ibm.com

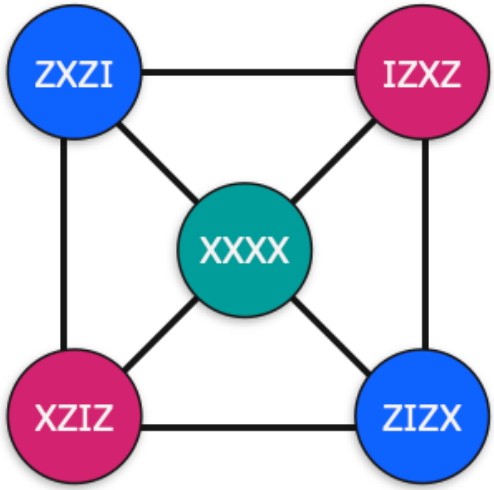


図 1: An example of a Pauli graph.

as nodes and edges on pairs of commutative Pauli strings for each qubit. In other words, an edge of a Pauli graph corresponds to a pair of Pauli strings that are not commutative at one or more qubits. The vertex coloring problem asks to minimize the number of node colors such that all two adjacent nodes have different colors. Since this grouping problem is known to be NP-hard, heuristic algorithms such as the LDFC (Largest Degree First Coloring) algorithm are often used to obtain approximate solutions in practice. Although LDFC solutions are not guaranteed to be optimal, several examples have shown that the gap between the optimal value and the lower bound is small for the Pauli graph of TPB [9].

There are challenges to compute grouping of a huge set of Pauli strings. Let n be the number of Pauli strings in quantum observable A and m be the number of edges of the Pauli graph of A with TPB. First, although LDFC requires $\mathcal{O}(n \log n)$ computational time, the entire grouping takes $\mathcal{O}(n^2)$ computational time if we include the time $\mathcal{O}(n^2)$ to construct the Pauli graph. Second, LDFC requires $\mathcal{O}(m)$ space if we store all the edges of the Pauli graph. In general, it holds that $m \gg n$.

To address these challenges, we propose the following approaches: 1) parallelize the construction of the edges of Pauli graphs, 2) avoid storing and loading the edge data of the Pauli graph in memory. These improvements have resulted in faster speed and less memory required.

2. Method

2.1 Existing method

We describe the existing method in more detail. The computation steps are as follows:

(1) Construct the Pauli graph.

(2) Perform the greedy coloring algorithm (Largest degree first coloring).

An example of Pauli graph is shown in Figure 1. Quantum observables which have edges are not commutative in at least one qubit. Constructing Pauli graph takes $\mathcal{O}(n^2)$ time because we need to check the commutativity for all pairs of operators. Then, we perform the coloring of the Pauli graph. LDFC is an algorithm that assigns colors in order of the largest degree to the smallest degree. In graph theory, degree is defined as the number of edges connected to each nodes. The LDFC algorithm repeats the operation of finding unassigned colors among neighboring nodes and assigning the color that is assigned by the node with the largest degree among them. As shown in the paper [9] numerically, LDFC generates near optimal solutions to some molecule data with TPB grouping.

Now let's think whether we actually need to store the entire information of the graph. As the name implies, LDFC assigns colors to nodes starting from the largest degree. Therefore, we first need to calculate the degrees for all nodes to sort by order of largest to smallest. The calculation of the degrees requires the commutativity check all pairs of operators. In the second step, assigning a color to a node, it is necessary to check whether the node is commutative with all nodes with the color. If so, we can assign the color to the node. Otherwise, we try another color. Again, we need to check the commutativity here.

If we store all edges of the Pauli graph, we can check the commutativity in the second step by just loading the edge data. However, storing all edge data requires extra time in addition to check commutativity of all pairs of Pauli strings in the first step.

2.2 Largest degree first coloring without storing and loading the Pauli graph

The above argument implies that we can avoid storing all the edges by checking commutativity again in the second step (assigning colors) instead of loading the edge data. We need to check whether a node is commutative with nodes with each color. We can stop the check if a node is commutative with all nodes with a color, i.e., assign the color to the node. Thus, the number of commutativity checks is expected to be smaller than the number of edges. Furthermore, instead of checking the commutativity with all Pauli strings of each color, finding the color to be assigned can be achieved to check the commutativity with the measurement basis for each color. The number of commutability checks can be reduced because only one

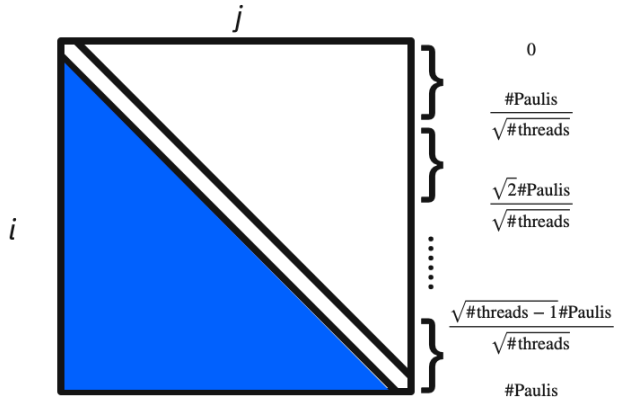


図 2: Illustration of parallelization.

commutability check is needed instead of checking commutability with all the Pauli strings within each color. If the data is held in bits as described below, the measurement basis can be calculated easily by bit OR operations. Therefore, checking commutativity again in the second step can be faster than storing all edges in the first step. We can also save the memory space for the edge data.

2.3 Parallelization of the computation of the degrees

Checking the commutativity ($\mathcal{O}(n^2)$) is the bottleneck of the grouping algorithm based on LDFC. However, we can parallelize the commutativity check. We check the commutativity between P_i and P_j for all $i \neq j$. Focusing on the symmetry about changing i and j, we only need to check the lower (or upper) triangular components. Now, we consider a parallelization for i as shown in the figure 2. If we divide the elements into equal intervals, the number of elements will be skewed. In order to divide the data uniformly, we propose to take the k-th partition from $\frac{\sqrt{k}}{\sqrt{\#\text{threads}}}$ to $\frac{\sqrt{k+1}}{\sqrt{\#\text{threads}}}$ as the width of i shown in Figure 2. Here, #threads is the number of threads in the parallelization. This is based on the fact that the sum of i up to K is $K(K+1)/2$, which scales with $\mathcal{O}(K^2)$. It can be divided if taken as a square root. Since this value is not an integer in general, it is rounded to the nearest number. It is also possible to divide the number of data equally, but for the sake of simple implementation, we implemented this approximate division.

2.4 Checking commutativity

A common task in calculating degrees and determining colors is to check commutativity of two Pauli strings. Two Pauli strings are commutative only if all their Pauli operators are commutative for every qubit, and two Pauli operators are commutative if they are the same or one of them is I.

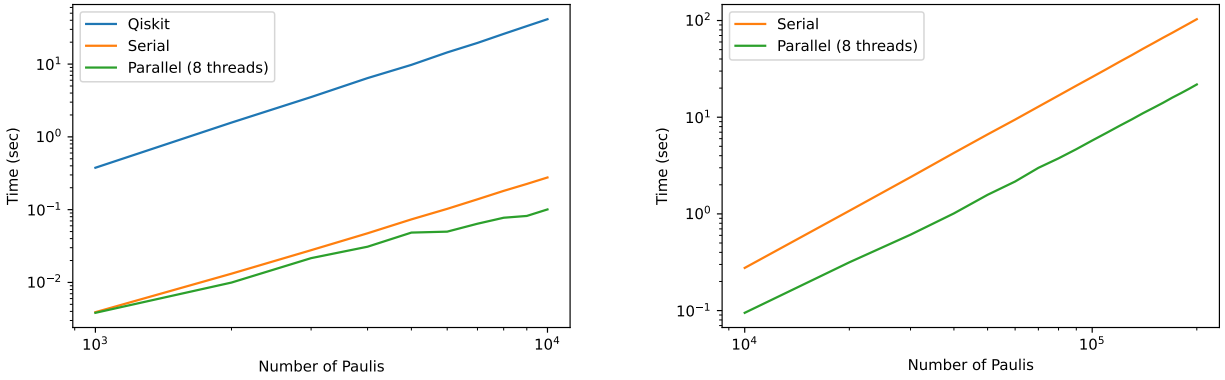
We represent a Pauli string P_i with a type `uint64_t` and every two-bits represent a Pauli operator: 00, 01, 11, and 10 means I, X, Y, and Z, respectively. With this representation, commutativity of two Pauli strings can be tested by calculating their XOR excepting 00. That is, our implementation first generates a mask m_i for P_i to indicate positions of X, Y, and Z in P_i with two-bits. For example, a Pauli string $ZXZI$ is represented as 10011000 and its mask is 11111100. Once all the masks of Pauli strings are calculated, commutativity of two Pauli strings P_i and P_j are tested by $(P_i \& m_i) \oplus (P_j \& m_j)$. Note that an array of `uint64_t` can be used for more than thirty-qubit Pauli strings and their masks.

SIMD instructions are effective to perform the above bit operations for commutativity check of Pauli strings. If 512-bit registers and their instructions are available, eight pairs of Pauli-strings can be simultaneously checked for their commutativities.

3. Result

We compare the computation times of grouping random Pauli strings with various methods. The random Pauli strings are generated from 70% I and 10% X, Y, Z each. The reason for this bias in the generation probability of Pauli strings is that if all strings were generated uniformly, the number of groups would be too large because joint measurements would not be possible in most cases. Compared to the Hamiltonian of molecules, it is the same that the non-identity Pauli strings exist $\mathcal{O}(N)$ for Jordan–Wigner and Parity transformation where N is the number of the qubits. However, since the Pauli strings exist $\mathcal{O}(\log N)$ for Bravyi–Kitaev transformation, the situation is different. The non-identity Pauli operator also has a biased appearance probability, but the bias depends on the transformations and molecules. For simplicity, we use the random Pauli string defined above.

We show the result of the benchmark in Figure 3. “Qiskit” in the legend represents grouping using the `group_qubit_wise_commuting` method of `PauliList` in Qiskit Terra 0.19 [14]. This method uses NumPy’s vectorization to speed up the construction of the Pauli graph, and uses retworkx [15] implemented in Rust for graph algorithms including the LDFC algorithm. “Serial” used the algorithm we presented above without parallelization. “Parallel (8 threads)” used the algorithm we presented with parallelization using 8 threads. The computer had



(a) Benchmark for small numbers of Pauli strings.

(b) Benchmark for large numbers of Pauli strings.

図 3: Benchmark of the computation time of grouping for random Pauli strings.

8 logic processors (8 CPU cores). This results shows our proposed method can improve the performance of grouping.

Using this fast grouping, we tried to group a large number of Pauli strings. Subspace expansion [16] and quantum computed moments correction [17] requires the estimation of the number of Pauli strings that are greater than the number of strings in the original Hamiltonian. For simplicity, we try grouping the square of the Hamiltonian. Since the Hamiltonian changes depending on the setting of the atomic distance and other factors, we use publicly available Hamiltonian data for benchmarking [18]. The results of the grouping are shown in Table 1. We succeeded in grouping of large number of Pauli strings. Interestingly, it turns out that the square Hamiltonian is harder to reduce than the original. This may be due to the fact that the squaring makes the string more dense.

4. Discussion

Some of the authors pointed out in the previous SIGQS 4th that other Greedy algorithms were better for all entangled measurements and that there was room for further improvement. It would be worth considering a parallel version of independent-set or the color interchange algorithm without constructing graphs.

Molecule	Transformation	#Qubits	#Pauli strings	#Groups	#Pauli strings (square)	#Groups (square)
LiH	JW	12	631	135	25542	2233
LiH	Parity	12	631	165	25542	2433
LiH	BK	12	631	212	25542	3464
BeH2	JW	14	1150	215	91831	7531
BeH2	Parity	14	1150	324	91831	10074
BeH2	BK	14	1150	341	91831	10708
H2O	JW	14	1858	380	179083	13603
H2O	Parity	14	1858	495	179083	16313
H2O	BK	14	1858	515	179083	18825
NH3	JW	16	4957	1052	1043405	73852
NH3	Parity	16	4957	1090	1043405	74559
NH3	BK	16	4957	1083	1043405	64454
HCl	JW	20	4427	906	1780408	156995
HCl	Parity	20	4427	1099	1780408	181924
HCl	BK	20	4427	1434	1780408	197685

表 1: The number of groups for the molecule Hamiltonian H and its square H^2.

参考文献

[1] Jena, A., Genin, S. and Mosca, M.: Pauli Partitioning with Respect to Gate Sets. Preprint at https://arxiv.org/abs/1907.07859 (2019).
[2] Yen, T.-C., Verteletskyi, V. and Izmaylov, A. F.: Measuring All Compatible Operators in One Series of Single-Qubit Measurements Using Unitary Transformations, *Journal of Chemical Theory and Computation*, Vol. 16, No. 4, pp. 2400–2409 (online), DOI: 10.1021/acs.jctc.0c00008 (2020). PMID: 32150412.
[3] Gokhale, P., Angiuli, O., Ding, Y., Gui, K., Tomesh, T., Suchara, M., Martonosi, M. and Chong, F. T.: Minimizing State Preparations in Variational Quantum Eigensolver by Partitioning into Commuting Families. Preprint at https://arxiv.org/abs/1907.13623 (2019).
[4] Izmaylov, A. F., Yen, T.-C. and Ryabinkin, I. G.: Revising the measurement process in the variational quantum eigensolver: is it possible to reduce the number of separately measured operators?, *Chem. Sci.*, Vol. 10, pp. 3746–3755 (online), DOI: 10.1039/C8SC05592K (2019).
[5] Huggins, W. J., McClean, J., Rubin, N., Jiang, Z., Wiebe, N., Whaley, K. B. and Babbush, R.: Efficient and Noise Resilient Measurements for Quantum Chemistry on Near-Term Quantum Computers. Preprint at https://arxiv.org/abs/1907.13117 (2019).
[6] Zhao, A., Tranter, A., Kirby, W. M., Ung, S. F., Miyake, A. and Love, P. J.: Measurement reduction in variational quantum algorithms, *Phys. Rev. A*, Vol. 101, p. 062322 (online), DOI: 10.1103/PhysRevA.101.062322 (2020).
[7] Crawford, O., van Straaten, B., Wang, D., Parks, T., Campbell, E. and Brierley, S.: Efficient quantum measurement of Pauli operators. Preprint at https://arxiv.org/abs/1908.06942 (2019).
[8] Gokhale, P. and Chong, F. T.: $O(N^3)$ Measurement Cost for Variational Quantum Eigensolver on Molecular Hamiltonians. Preprint at https://arxiv.org/abs/1908.11857 (2019).
[9] Hamamura, I. and Imamichi, T.: Efficient evaluation of quantum observables using entangled measurements, *npj Quantum Inf.*, Vol. 6, No. 1, p. 56 (online), DOI: 10.1038/s41534-020-0284-2 (2020).
[10] Huang, H.-Y., Kueng, R. and Preskill, J.: Predicting many properties of a quantum system from very few measurements, *Nature Physics*, Vol. 16, No. 10, pp. 1050–1057 (online), DOI: 10.1038/s41567-020-0932-7 (2020).
[11] Hadfield, C., Bravyi, S., Raymond, R. and Mezzacapo, A.: Measurements of Quantum Hamiltonians with Locally-Biased Classical Shadows (2020).
[12] Huang, H.-Y., Kueng, R. and Preskill, J.: Efficient estimation of Pauli observables by derandomization (2021).
[13] Kandala, A., Temme, K., Córcoles, A. D., Mezzacapo, A., Chow, J. M. and Gambetta, J. M.: Error mitigation extends the computational reach of a noisy quantum processor, *Nature*, Vol. 567, No. 7749, pp. 491–495 (online), DOI: 10.1038/s41586-019-1040-7 (2019).
[14] ANIS, M. S., Abraham, H., AduOffei, Agarwal, R. and et al.: Qiskit: An Open-source Framework for Quantum Computing, https://doi.org/10.5281/zenodo.2562110 (2021).
[15] Treinish, M., Carvalho, I., Tsilimigkounakis, G. and Sá, N.: retworkx: A High-Performance Graph Library for Python (2021).
[16] McClean, J. R., Kimchi-Schwartz, M. E., Carter, J. and de Jong, W. A.: Hybrid quantum-classical hierarchy for mitigation of decoherence and determination of excited states, *Phys. Rev. A*, Vol. 95, p. 042308 (online), DOI: 10.1103/PhysRevA.95.042308 (2017).
[17] Vallury, H. J., Jones, M. A., Hill, C. D. and Hollenberg, L. C. L.: Quantum computed moments correction to variational estimates, *Quantum*, Vol. 4, p. 373 (online), DOI: 10.22331/q-2020-12-15-373 (2020).
[18] Bravyi, S., Gambetta, J. M., Mezzacapo, A. and Temme, K.: Tapering off qubits to simulate fermionic Hamiltonians. Preprint at https://arxiv.org/abs/1701.08213 (2017).