PHOENIX: Pauli-based High-level Optimization Engine for Instruction Execution on NISQ Devices

Abstract*—Variational quantum algorithms (VQA) based on Hamilton simulation represent a specialized class of quantum programs well-suited for near-term quantum computing applications due to its modest resource requirements in terms of qubits and circuit depth. Unlike the conventional single-qubit/two-qubit gate sequence representation, Hamiltonian simulation programs are composed of disciplined subroutines known as Pauli exponentiations (Pauli strings with coefficients) that are variably arranged. To capitalize on these distinct program features, this study introduces PHOENIX, a highly effective compilation framework that primarily operates at the high-level Pauli-based intermediate representation (IR) for generic Hamiltonian simulation programs. PHOENIX exploits global program optimization opportunities to the largest extent, compared to existing SOTA methods despite some of them also utilizing similar IRs. PHOENIX employs the binary symplectic form (BSF) to formally describe Pauli strings and reformulates IR synthesis as a sequence of Clifford transformations on BSF. It comes with a heuristic BSF simplification algorithm that simultaneously reduces Pauli strings' weights to two-qubit operations by identifying the most appropriate Clifford operators. PHOENIX further performs a global ordering strategy in a Tetris-like fashion for these simplified IR groups, carefully balancing optimization opportunities for gate cancellation, minimizing circuit depth, and managing qubit routing overhead. Experimental results demonstrate that PHOENIX outperforms SOTA VQA compilers across diverse program categories, backend ISAs, and hardware topologies.

I. INTRODUCTION

[ZY: Problem, Motivation, Insight starting point]

Quantum computing ...

Variational quantum algorithms (VQA) is a class of leading application in near-term quantum computing due to their modest resources requirements in terms of qubits and circuit depth and their noise-resillence.

[ZY: There should be 1 figure for illustration]

II. RELATED WORKS

[ZY: Efficient fermionic mapping v.s. efficient IR exponentiations]

[ZY: Paulihedral first proposed VQA IR ... Pauli strings and Pauli blocks]

[ZY: TKet, Paulihedral, PCOAST, Tetris, 2QAN, ...]

III. BSF AND CLIFFORD FORMALISM

[ZY: To illustrate the theoretic foundation and how we formulate Pauli IR synthesis problem]

BSF is a tableau representation in which each row represents a single n-qubit Pauli string. Columns of the tableau are partitioned as two parts $[X \mid Z]$, such that $[X_{i,j} \mid Z_{i,j}]$ represents the j-th component of the i-th Pauli operator. The value is $[1|0]$ for X, $[0|1]$ for Z, $[1|1]$ for Y, and $[0|0]$ for I. For instance, the process in**??** that Pauli strings $[XXX; XYY]$ is transformed into $[IXX; IYY]$ is formulated as

$$\left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \end{array}\right] \xrightarrow{C(Z,X)_{1,0}} \left[\begin{array}{ccc|ccc} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \end{array}\right].$$

Different from sophisticated local optimization strategies for Pauli strings such as Paulihedral [1] and Tetris [2] that exploits near-neighbor gate cancellation opportunities exposed by variants of Pauli exponentiations' synthesis schemes, we propose the *BSF simplification* algorithm tailored to $\mathrm{SU}(4)$ primitives to achieve considerable simultaneous optimization effects for each set of equal-weight Pauli strings, utilizing the formal description of Pauli strings — "binary symplectic form" (BSF) [**?**].

The example in XXX-XYY-Simp gives a taste of our synthesis approach for Type-II IRs. Trivially, $e^{-i(\frac{\phi}{2}XYY + \frac{\theta}{2}XXX)}$ necessitates two V-shaped CNOT trees, resulting in 6 2Q gates. However, If the circuit is conjugated by the Clifford transformation $\mathrm{CNOT}_{1,0}$, it is simplified into two successive Ising gates acting on (q_1, q_2), equivalent with one $\mathrm{SU}(4)$. This case implies that a group of Pauli exponentiations can be simultaneously simplified by applying appropriate Clifford transformations. To formulate this scheme, we represent Pauli strings in BSF, a tableau representation in which each row represents a single n-qubit Pauli string. Columns of the tableau are partitioned as two parts $[X \mid Z]$, such that $[X_{i,j} \mid Z_{i,j}]$ represents the j-th component of the i-th Pauli operator. The value is $[1|0]$ for X, $[0|1]$ for Z, $[1|1]$ for Y, and $[0|0]$ for I. For instance, the process in XXX-XYY-Simp that Pauli strings $[XXX; XYY]$ is transformed into $[IXX; IYY]$ is formulated as

$$\left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \end{array}\right] \xrightarrow{\mathrm{CNOT}_{1,0}} \left[\begin{array}{ccc|ccc} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \end{array}\right].$$

With BSF representation, searching appropriate Clifford operators to simplify Pauli strings means lowering BSF's weights. We quantitatively characterize this procedure by defining a heuristic cost function

$$\begin{aligned} \mathrm{cost} = n_{\mathrm{nontrivial}}^2 * \Big(& \sum_{\langle i,j \rangle} \| r_x^{(i)} \vee r_z^{(i)} \vee r_x^{(j)} \vee r_z^{(j)} \| \\ & + \frac{1}{2} \sum_{\langle i,j \rangle} (\| r_x^{(i)} \vee r_x^{(j)} \| + \| r_z^{(i)} \vee r_z^{(j)} \|) \Big) \end{aligned} \quad (1)$$

which is the combined weight overlap of both X-part and Z-part among each pair of Pauli strings of a BSF involving "nontrivial" (nonlocal) Pauli strings. The ReQISC compiler

Algorithm 1: Pauli Strings Simplification in BSF

Input : Pauli strings list *pls*
Output: Reconfigured circuit components list *cfg*

cfg ← ∅; *bsf* ← BSF(*pls*); *cliffs_with_locals* ← ∅;
while *bsf*.TOTALWEIGHT() > 2 **do**
  local_bsf ← *bsf*.POPLOCALPAULIS();
  $C \leftarrow \emptyset$; `// Clifford2Q candidates`
  $B \leftarrow \emptyset$; `// Each element of` B `results from applying each Clifford2Q candidate on` *bsf*
  costs ← ∅ ; `// Cost functions calculated on each element of` B
  for *cg* ***in*** CLIFFORD_2Q_SET **do**
    for *i, j* ***in*** COMBINATIONS(RANGE(n), 2) **do**
      cliff ← *cg*.ON(i, j) ; `// qubits acted on`
      bsf′ ← *bsf*.APPLYCLIFFORD2Q(*cliff*);
      cost ← CALCULATEBSFCOST(*bsf*′);
      C.APPEND(*cliff*);
      B.APPEND(*bsf*′);
      costs.APPEND(*cost*);
    end
  end
  bsf ← BSFWITHMINCOST(B, *costs*);
  cliff ← CLIFFORDWITHMINCOST(C, *costs*);
  cliffs_with_locals.APPEND((*cliff*, *local_bsf*));
end
cfg.APPEND(*bsf*);
for *cliff, local_bsf* ***in*** *cliffs_with_locals* **do**
  `// Clifford2Q operators are added as conjugations, with local Pauli strings peeled before each epoch`
  cfg.PREPEND(*cliff*);
  cfg.APPEND(*local_bsf*);
  cfg.APPEND(*cliff*);
end

greedily searches the most appropriate Clifford operator from a set of 2Q Clifford group generators [3] with all possible qubit pairs in action. The Clifford operator that mostly minimizes the cost function is selected and the BSF gets updated accordingly. This process continues iteratively until the BSF can be directly synthesized by a single SU(4).

In practice, the ReQISC compiler performs a "grouping – simplification — ordering" pipeline to compile real-world Type-II programs involving heterogeneous-weight Pauli strings. It first groups Pauli strings with the same non-identity-acted qubit indices and then simplifies each group of Pauli strings. Finally, it greedily finds the optimal ordering of these simplified IR groups by exploiting opportunities for 2Q gate cancellation and reducing circuit depth, similar to the "selectively assembling" approach discussed above.

.....

IV. Our Propsal: Phoenix

A. Overview

B. BSF simplification for each IR group

To search suitable Clifford2Q operators, it suffices to just focus on a set of 2Q generators of the 2Q Clifford group. In Clifford group theory, the 6 universal controlled gates

$$\{C(X,X), C(Y,Y), C(Z,Z), C(X,Y), C(Y,Z), C(Z,X)\}$$

are independent of each other and span the entire 2Q Clifford group [3]. Each of them can reduce the weight of certain Pauli strings. For example, $C(X,Y)\,[0\,0\,|\,1\,1]\,C(X,Y) = [0\,0\,|\,1\,0]$ since $C(X,Y)_{a,b}$ exhibits the rule

$$[x_a,\, x_b \,|\, z_a,\, z_b] \rightarrow [x_a \oplus x_b \oplus z_b,\, z_a \oplus z_b \,|\, z_a,\, z_a \oplus z_b].$$

C. Ordering of IR groups in a Tetris style

V. Evaluation

We evaluate the effectiveness of Phoenix of across divere Hamiltonian simulation programs, quantum ISAs, and NISQ deive topologies. Phoenix is implemented in Python, capable of compiling large-scale VQA programs efficiently since the core algorithms of Phoenix exhibit linear computational complexity in terms of program size. All experiments are executed on a laptop (Apple M3 Max, 36GB memory).

A. Experimental settings

Metrics. We evaluate the effectiveness of Phoenix using following metrics: 2Q gate count, 2Q circuit depth, algorithmic error for VQA synthesis, and circuit fidelity. Notably, we exclude 1Q gates and their count when calculating circuit depth. On one hand, 1Q gates are generally considered "free" resources compared to 2Q gates due to their lower error rates and shorter durations. On the other hand, CNOT is not an native operation in most NISQ platforms. Its implementation typically requires extra 1Q drives before/after the native 2Q gate (e.g., Cross-Resonance [4] and Mølmer-Sørensen [5]). Including 1Q gates in the compilation metrics could therefore lead to misleading conclusions. Algorithmic error refers to the deviation between the synthesized circuit's unitary matrix and the ideal evolution under the original Hamiltonian, as measured by the infidelity between unitary matrices: $\mathrm{infid} = 1 - \frac{1}{N}|\mathrm{Tr}(U^\dagger V)|$. Circuit fidelity is assessed via the "Estimated Success Probability" (ESP) [6], which provides a theoretical estimation of the success probability based on the program and the hardware noise model.

Baselines. TKet [7], Paulihedral [1], and Tetris [2] are primary baselines for comparison with our method. For TKet, the `PauliSimp` and `FullPeepholeOptimise` passes are adopted for logical circuit optimization, providing a similar procedure to its default O2 compilation is adopted, in which the `PauliSimp` pass is particularly effective at optimizing Pauli gadgets; for Paulihedral, Qiskit O2 pass is equipped because the numerous gate cancellation opportunities exposed by Paulihedral necessitate inverse cancellation and commutative cancellation operations. For hardwar-aware compilation, all baselines and Phoenix are followed by Qiskit O3 pass with SABRE qubit mapping [8]. For QAOA benchmarking, 2QAN [9] is used as the SOTA baseline.

Benchmarks. We carefully select two representative Hamiltonian simulation benchmarks — (1) *UCCSD*: A set of UCCSD ansatz including CH_2, H_2O, LiH, and NH four

TABLE I
UCCSD BENCHMARK SUITE.

Benchmark	#Qubit	#Pauli	w_{max}	#Gate	#CNOT	Depth	Depth-2Q
CH2_cmplt_BK	14	1488	10	37780	19574	23568	19399
CH2_cmplt_JW	14	1488	14	34280	21072	23700	19749
CH2_frz_BK	12	828	10	19880	10228	12559	10174
CH2_frz_JW	12	828	12	17658	10344	11914	9706
H2O_cmplt_BK	14	1000	10	25238	13108	15797	12976
H2O_cmplt_JW	14	1000	14	23210	14360	16264	13576
H2O_frz_BK	12	640	10	15624	8004	9691	7934
H2O_frz_JW	12	640	12	13704	8064	9332	7613
LiH_cmplt_BK	12	640	10	16762	8680	10509	8637
LiH_cmplt_JW	12	640	12	13700	8064	9342	7616
LiH_frz_BK	10	144	9	2890	1442	1868	1438
LiH_frz_JW	10	144	10	2850	1616	1985	1576
NH_cmplt_BK	12	640	10	15624	8004	9691	7934
NH_cmplt_JW	12	640	12	13704	8064	9332	7613
NH_frz_BK	10	360	9	8303	4178	5214	4160
NH_frz_JW	10	360	10	7046	3896	4640	3674

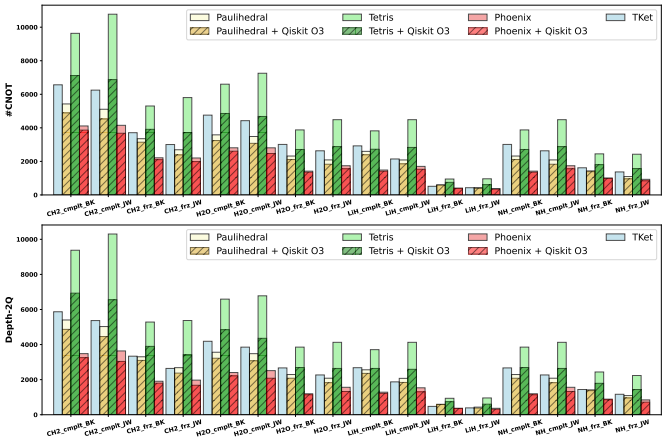

Fig. 1. Logical-level compilation (all-to-all topology).

categories of molecule simulation programs, totally 16 benchmarks, supporting the primary evaluation in our study. Each category is generated with STO-3G orbitals [10], encoded by Jordan-Wiger (JW) [11] and approximated by complete and frozen core, encoded by Bravyi-Kitaev (BK) [12] transformations, respectively. Details are shown in Tab. I. (2) *QAOA*: A set of 2-local Hamiltonian simulation programs corresponding to random graphs and regular graphs, for which description and evaluation is shown in Tab. IV.

TABLE II
AVERAGE (GEOMETRIC-MEAN) OPTIMIZATION RATES ON UCCSD.

Compiler	#CNOT opt.	Depth-2Q opt.
TKET	33.07%	30.14%
PAULIHEDRAL	28.41%	29.07%
PAULIHEDRAL + O3	25.72% (-8.54% v.s. no O3)	26.3% (-8.60% v.s. no O3)
TETRIS	53.66%	53.26%
TETRIS + O3	36.73% (-30.94% v.s. no O3)	36.37% (-31.08% v.s. no O3)
PHOENIX	21.15%	19.28%
PHOENIX + O3	19.56% (-6.58% v.s. no O3)	17.29% (-8.42% v.s. no O3)

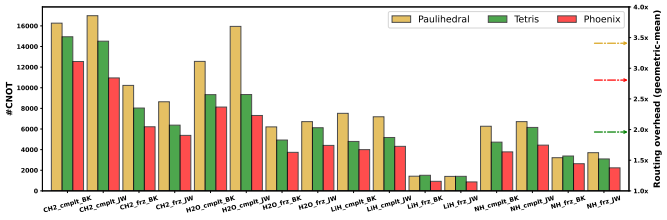

Fig. 2. Hardware-aware compilation on heavy-hex topology. Three dashed lines represent the average multiples of #CNOT within circuits after mapping relative to those after logical optimization, for PAULIHEDRAL (gold), TETRIS (green), and PHOENIX (coral), respectively.

B. Main results

Fig. 1 and Tab. II illustrates the main benchmarking results regarding logical program synthesis:

1) PHOENIX significantly outperforms baselines across all benchmarks, with an average 21.15% and 17.29% optimization rate[1] in #CNOT and Depth-2Q, respectively, relative to original circuits. That is mostly attributed to the group-wise BSF simplification mechanism, as PHOENIX adopts the same IR grouping method as PAULIHEDRAL and TETRIS.
2) In task of the logical-level synthesis, TETRIS performs the worst, falling far behind TKET and the earlier PAULIHEDRAL, let alone PHOENIX. This is because TETRIS focuses primarily on co-optimization techniques to reduce SWAP gates during qubit routing.
3) We also compare PAULIHEDRAL/TETRIS/PHOENIX with and without QISKIT optimization, to evaluate their proposed high-level optimization capabilities. QISKIT O3 improves PAULIHEDRAL and TETRIS a lot over our work. Therefore, PHOENIX's high-level optimization strategy is more impressive and leaves much less gate cancellation workloads to QISKIT O3.

C. Hardware-aware compilation

We choose the heavy-hex topology, specifically a 64-qubit coupling graph corresponding to IBM's Manhattan processor [13], as the limited-topology backend to perform hardware-aware compilation. Results are depicted in Fig. 2. TKET's results are not comparable to PAULIHEDRAL/TETRIS/PHOENIX, so it is not shown in Fig. 2. Although PHOENIX focuses primarily on high-level logical program optimization while integrates limited efforts on hardware-aware co-optimization, it still outperforms PAULIHEDRAL and TETRIS in terms of #CNOT and Depth-2Q of the ultimate mapped circuits, with an average 35.9% (22.3%) and 43.6% (27.8%) reduction, respectively, compared to PAULIHEDRAL (TETRIS).

The consideration of integrating qubit mapping transition overhead into the heuristic IR group ordering functionality in PHOENIX controls the routing overhead, resulting in 2.8x #CNOT on average. That is better than PAULIHEDRAL and

[1]For example, the #CNOT optimization rate is defined as $\frac{\#\text{CNOT}_{\text{after}}}{\#\text{CNOT}_{\text{before}}}$.

worse than TETRIS, since the latter specially exploits #CNOT cancellation opportunities for SWAP-based routing.

Consequently, even for hardware mapping on limited-topology devices, PHOENIX effectively controls qubit routing overhead and ultimately surpasses co-design local optimization strategies.

D. Comparison in diverse ISAs

Quantum instruction set architecture (ISA), or the native gate set in a narrow sense, serves as a contract between software and hardware implementation. The design of quantum ISA plays a crucial role in the performance of NISQ devices, in which the 2Q gate dominates its hardware accuracy and cost as well as the software expressivity. Although the conventional gate set includes 1Q gates and CNOT-based 2Q gates, recently some works propose integrating complex and continuous 2Q gates into the ISA design, such as the $\mathrm{XY}(\theta)$ gate family by Rigetti Computing [14], the fractional gates on IBM's Heron processors[2], and the AshN gate scheme [15] considering all possible 2Q gates within the $\mathrm{SU}(4)$ group as the quantum ISA.

Therefore, we further compare PHOENIX with baselines in different quantum ISAs, to showcase that PHOENIX's ISA-independent synthesis strategies proves to be a generally powerful optimization engine accommodating diverse ISAs.

Without loss of generality, we consider the most expressive $\mathrm{SU}(4)$ ISA with all-to-all and heavy-hex qubit topologies, respectively, in addition to the benchmarking in CNOT ISA above. For logical-level compilation, PHOENIX can directly generate $\mathrm{SU}(4)$-based circuits through its BSF simplification algorithm, while for baselines (PAULIHEDRAL and TETRIS are equipped with QISKIT O3 by default) an extra "rebase" (or "transpile") procedure is required to convert the CNOT-based circuits to $\mathrm{SU}(4)$-based ones; for hardware-aware compilation, all compilers require the rebase procedure, following the QISKIT O3 hardware-aware compilation pass. Detailed outcomes are summarized in Tab. III, wherein we highlight the geometric-mean relative optimization rates of PHOENIXcompared to baselines' results.

Again, PHOENIX significantly outperforms baselines when targeting $\mathrm{SU}(4)$ ISA. The optimization rates relative to baselines are more impressive than those in CNOT ISA, despite the baselines incorporating sophisticated optimization techniques specifically designed for the CNOT ISA. For instance, the multiple of PHOENIX's #2Q relative to PAULIHEDRAL's in CNOT ISA is 82.21% (62.63%), whereas this value decreases to 75.69% (39.97%) in $\mathrm{SU}(4)$ ISA for hardware-agnostic (hardware-aware) compilation. One exception is the hardware-aware compilation comparison with TKET, as TKET's hardware-aware compilation in CNOT ISA generates much larger 2Q gate count and circuit depth than other compilers, and there are many 2Q subcircuit fusing opportunities such that the rebased circuits involves much fewer $\mathrm{SU}(4)$ gates.

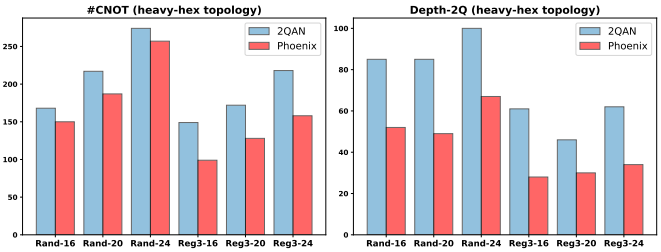

Fig. 3. QAOA benchmarking

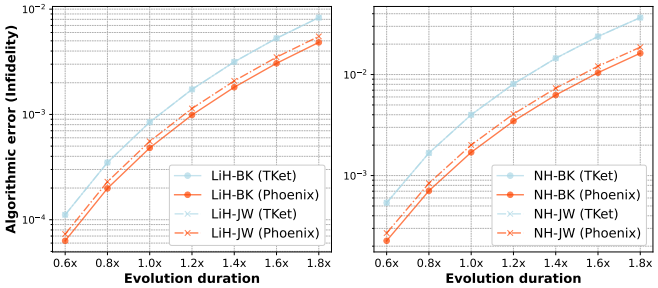

Fig. 4. Algorithmic error comparison of LiH and NH simulation.

E. QAOA benchmarking

For QAOA benchmarking, we focus on the the performance in hardware-aware compilation, since the 2Q gate count cannot be reduced and minimizing circuit depth is trivial in logical-level compilation, as both 2QAN and PHOENIX can generate depth-optimal QAOA circuits at the logical level at the in our field test. Fig. 3 and Tab. IV illustrate compilation results on heavy-hex topology, across six representative QAOA programs corresponding to both random graphs (each node with degree 4) and regular (each node with degree 3) graphs, with qubit sizes of 16, 20, and 24. PHOENIX outperforms 2QAN across all benchmarks in all metrics (e.g., #CNOT, #SWAP), especially in Depth-2Q, with an average 40.8% reduction compared to 2QAN. These results further demonstrate the effectiveness of the routing-aware IR group ordering method in PHOENIX.

F. Algorithmic error analysis

We further highlight the advantage of PHOENIX in reducing algorithmic errors for VQA programs, which limits the best accuracy of practical VQA's outcomes when executing on noisy hardware. From UCCSD benchmarks, we select those with qubit sizes no more than 10 for evaluation, given the matrix computation capabilities of standard PCs. We rescale the coefficients of Pauli strings to control their algorithmic errors within the range of 0.00005 to 0.01, which corresponds to different-scale evolution durations of molecular simulation, as suggested in Fig. 4. In contrast to TKET, PHOENIX typically leads to lower algorithmic errors for both JW and BK encoding schemes. Although this improvement is program-specific, it is more significant for Pauli string patterns of BK than those

TABLE III
COMPARISON FOR DIVERSE ISAS WITH ALL-TO-ALL AND LIMITED-TOPOLOGY.

	CNOT ISA (all-to-all)		SU(4) ISA (all-to-all)		CNOT ISA (heavy-hex)		SU(4) ISA (heavy-hex)	
PHOENIX's opt. rate	#CNOT	Depth-2Q	#SU(4)	Depth-2Q	#CNOT	Depth-2Q	#SU(4)	Depth-2Q
PHOENIX v.s. TKET	63.93%	63.96%	**56.12%**	**54.21%**	**40.79%**	**48.51%**	44.44%	50.65%
PHOENIX v.s. PAULIHEDRAL	82.21%	73.29%	**75.69%**	**65.18%**	62.63%	54.92%	**39.97%**	**35.02%**
PHOENIX v.s. TETRIS	57.58%	53.01%	**56.63%**	**50.54%**	76.27%	71.47%	**62.44%**	**58.66%**

TABLE IV
QAOA BENCHMARKING VERSUS 2QAN.

QAOA		#CNOT		Depth-2Q		#SWAP		Routing overhead	
Bench.	#Pauli	2QAN	Phoenix	2QAN	Phoenix	2QAN	Phoenix	2QAN	Phoenix
Rand-16	32	168	150	85	52	37	29	2.62x	2.34x
Rand-20	40	217	187	85	49	47	39	2.71x	2.34x
Rand-24	48	274	257	100	67	63	56	2.85x	2.68x
Reg3-16	24	149	99	61	28	44	17	3.10x	2.06x
Reg3-20	30	172	128	46	30	46	23	2.87x	2.13x
Reg3-24	36	218	158	62	34	62	30	3.03x	2.19x
Avg. improv.		-16.7%		-40.8%		-29.41%		-16.59%	

of JW, with 57% (42.7%) and 49.5% (34.1%) for NH (LiH) simulation, respectively. The algorithmic errors resulting from PAULIHEDRAL and TETRIS are comparable to PHOENIX and are not shown in Fig. 4, as they adopt the same Pauli string blocking approach. As a result, the impressive algorithmic error reduction effect of PHOENIX further pushes toward the quantum advantage on computational chemistry problems.

G. Fidelity analysis

[ZY: TODO or to delete]

VI. CONCLUSION AND OUTLOOK

[ZY: Summary of our work]

REFERENCES

[1] G. Li, A. Wu, Y. Shi, A. Javadi-Abhari, Y. Ding, and Y. Xie, "Paulihedral: a generalized block-wise compiler optimization framework for quantum simulation kernels," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 554–569.

[2] Y. Jin, Z. Li, F. Hua, T. Hao, H. Zhou, Y. Huang, and E. Z. Zhang, "Tetris: A compilation framework for vqa applications in quantum computing," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 277–292.

[3] D. Grier and L. Schaeffer, "The classification of clifford gates over qubits," *Quantum*, vol. 6, p. 734, 2022.

[4] C. Rigetti and M. Devoret, "Fully microwave-tunable universal gates in superconducting qubits with linear couplings and fixed transition frequencies," *Physical Review B—Condensed Matter and Materials Physics*, vol. 81, no. 13, p. 134507, 2010.

[5] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-ion quantum computing: Progress and challenges," *Applied Physics Reviews*, vol. 6, no. 2, 2019.

[6] E. Magesan, J. M. Gambetta, and J. Emerson, "Scalable and robust randomized benchmarking of quantum processes," *Physical review letters*, vol. 106, no. 18, p. 180504, 2011.

[7] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "$t|ket\rangle$: a retargetable compiler for nisq devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, 2020.

[8] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, 2019, pp. 1001–1014.

[9] L. Lao and D. E. Browne, "2qan: A quantum compiler for 2-local qubit hamiltonian simulation algorithms," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 351–365.

[10] W. J. Hehre, R. F. Stewart, and J. A. Pople, "Self-consistent molecular-orbital methods. i. use of gaussian expansions of slater-type atomic orbitals," *The Journal of Chemical Physics*, vol. 51, no. 6, pp. 2657–2664, 1969.

[11] P. Jordan and E. Wigner, "über das paulische äquivalenzverbot. z phys 47: 631," 1928.

[12] S. B. Bravyi and A. Y. Kitaev, "Fermionic quantum computation," *Annals of Physics*, vol. 298, no. 1, pp. 210–226, 2002.

[13] G. J. Mooney, G. A. White, C. D. Hill, and L. C. Hollenberg, "Whole-device entanglement in a 65-qubit superconducting quantum computer," *Advanced Quantum Technologies*, vol. 4, no. 10, p. 2100061, 2021.

[14] D. M. Abrams, N. Didier, B. R. Johnson, M. P. d. Silva, and C. A. Ryan, "Implementation of xy entangling gates with a single calibrated pulse," *Nature Electronics*, vol. 3, no. 12, pp. 744–750, 2020.

[15] J. Chen, D. Ding, W. Gong, C. Huang, and Q. Ye, "One gate scheme to rule them all: Introducing a complex yet reduced instruction set for quantum computing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. La Jolla, CA, USA: ACM, 2024, pp. 779–796.