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

Abstract*—Variational quantum algorithms 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. Overall, PHOENIX outperforms other SOTA dedicated compilers in benchmarking across diverse program categories, backend ISAs, and hardware topologies.

I. INTRODUCTION

Quantum computing ...

II. MOTIVATION

[ZY: Motivation and preliminary knowledge]

III. OUR PROPSAL: PHOENIX

A. Overall framework

B. BSF simplification for each IR group

C. Ordering of IR groups

IV. EVALUATION

A. Experimental settings

B. Benchmarks

1) Metrics:

2) Baselines:

- TKET
- PAULIHEDRAL
- TETRIS

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

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

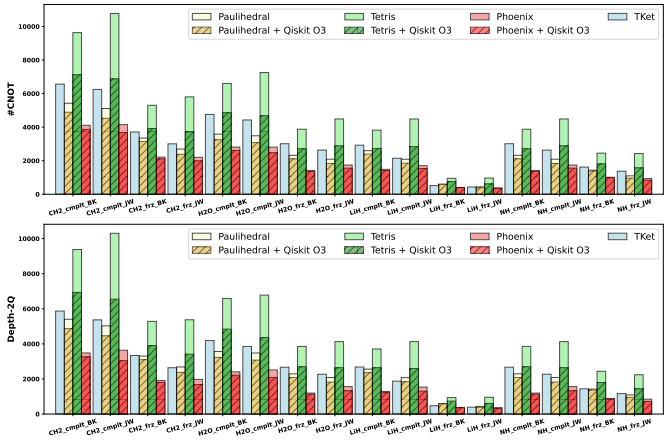

Fig. 1. Bechmarking on logical-level synthesis (all2all topology)

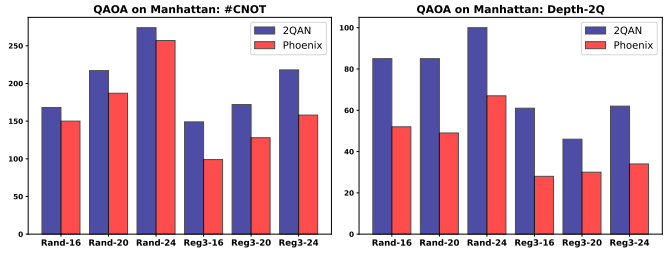

Fig. 2. QAOA benchmarking

TABLE II
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	2.62	2.34
Rand-20	40	217	187	85	49	47	39	2.71	2.34
Rand-24	48	274	257	100	67	63	56	2.85	2.68
Reg3-16	24	149	99	61	28	44	17	3.10	2.06
Reg3-20	30	172	128	46	30	46	23	2.87	2.13
Reg3-24	36	218	158	62	34	62	30	3.03	2.19
Avg. improv.		-16.7%		-40.8%		-29.41%		-16.59%	

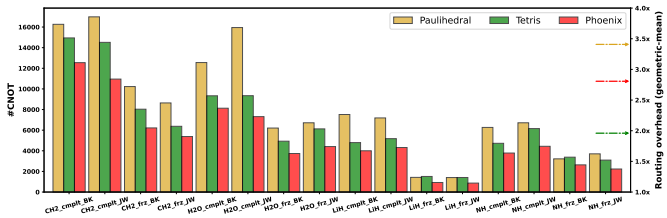

Fig. 3. Hardware-aware compilation for limited-topology NISQ device

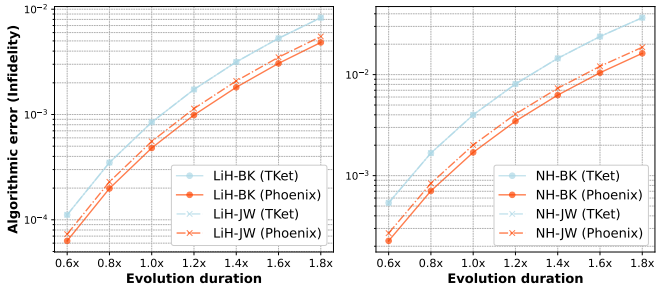

Fig. 4. Algorithmic error

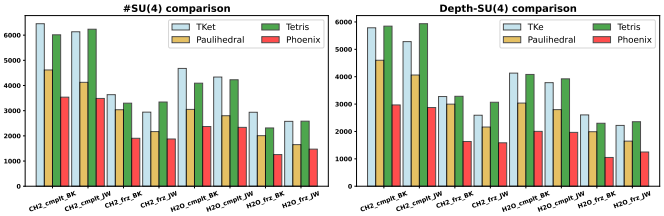

Fig. 5. SU(4) ISA comparison

C. Logical-level compilation

D. Breakdown analysis

E. QAOA benchmarking

F. Hardware-aware compilation

G. Diverse ISA comparison

H. Real system evaluation

I. Scalability