Circuit Compilation Methodologies for Quantum Approximate Optimization Algorithm

Mahabubul Alam Abdullah Ash- Saki Swaroop Ghosh

UC RIVERSIDE

CONTENT

UC RIVERSIDE

01 BACKGROUND

Quantum computing is one of the most transformative technologies of the present time. Prototypical quantum computers with 5-128 qubits are available or proposed from industry vendors like IBM, Google, Rigetti, etc.

Variational quantum-classical algorithms need to be explored to gain the quantum advantage for various problems

- Limited number of qubits and connectivity
- The near-term devices are suffered from all kind of errors such as gate-error, decoherence, crosstalk, etc.
- Therefore, quantum error correction are necessary for fault- tolerant computation.

However, QECCs have prohibitively high physical qubit overhead.

QAOA

Quantum Approximate Optimization Algorithm (QAOA): the most promising algorithm for solving optimization problems.

But quantum advantage may be lost due to the accumulation of gate errors and decoherence.

UC RIVERSIDE

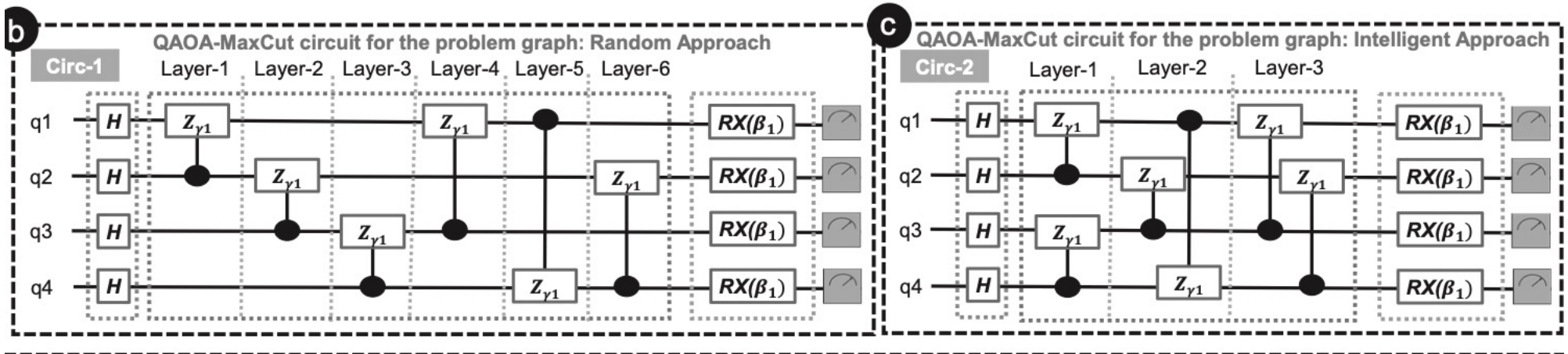
01 BACKGROUND

Gate Error, Decoherence and Crosstalk: **(less gate number, lower gate depth are better)**

- Quantum gates are error-prone. The qubits suffer from decoherence.
- The deeper circuit, more time to execute and gets affected by decoherence.
- More gates in the circuit also increase the accumulation of gate error.
- Parallel gate operations on different qubits can affect each others performance which is known as crosstalk.

CPHASE: A two-qubit unitary parametric quantum gate operating between a control and a target qubit. The CPHASE operations in a QAOA circuit are commutative i.e., the order of these CPHASE gates can be interchanged without affecting the output state of the quantum circuit.

SWAP gates are added between two layers to meet the hardware constraints.



UC RIVERSIDE

02 MOTIVATION

Conventional compilers can optimize QAOA-circuits using efficient gate scheduling strategies utilizing the commutation properties of the gates. However, incorporating gate-reordering strategies in a compiler is not straight forward.

- additional constraints need to be added
- Time consuming.

Challenges:

Finding a mapping with higher reliability (less impacted by noise) is important to extract maximum performance from QAOA.

Minimizing the depth/gate-count of the compiled circuit is crucial for QAOA applications.

Methodologies for larger problem sizes for powerful quantum hardware with 200-500 qubits) are absent nowadays.

Qubit Mapping:
(1)Selection of physical qubits (2) initial logical-to-physical qubit mapping (3) addition of SWAP gates to meet the hardware coupling constraints.
Each of these steps affects the quality of the compiled circuit i.e., depth, gate-count, and reliability.

03 METHODOLOGIES

NAÏVE APPROACH

A general-purpose compiler (qiskit compiler from IBM). Used for comparison of proposed methods.

QAIM

Is an intelligent qubit allocation and initial mapping approach which applies to any circuit compilation

IP, IC, and VIC

Using a backend compiler to add SWAP operations into the circuit with various target objectives.

Each of these methods has exclusive benefits and should be adopted based on the requirements of the target application.

UC RIVERSIDE

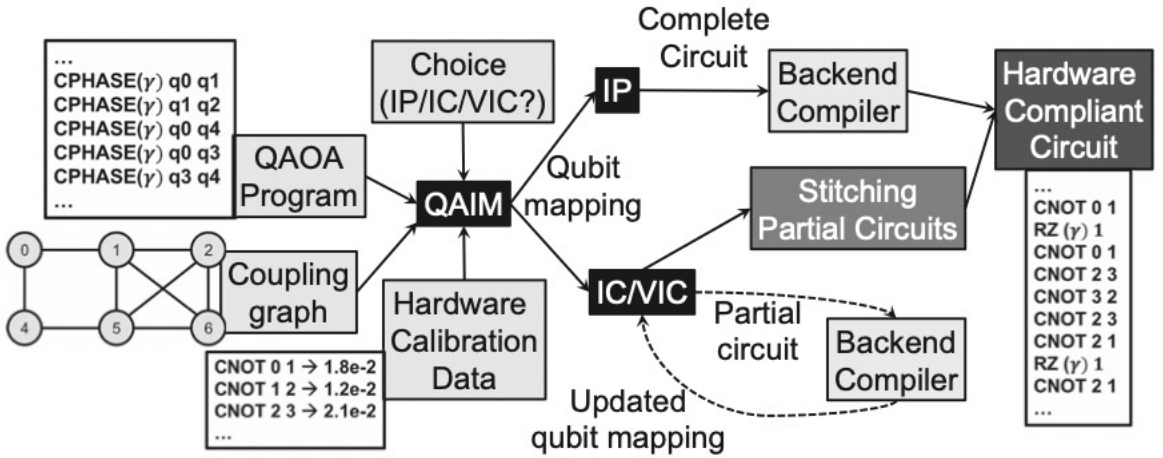
Basic Steps

QAIM generates an initial logical-to-physical qubit mapping.

passed to the chosen compilation procedure → IP/IC or VIC.

IP/IC/VIC uses a backend compiler to generate the hardware compliant circuit.

IP passes a complete circuit description to the backend, IC and VIC send partial circuit descriptions in multiple iterations and stitch the compiled circuits at the end.



UC RIVERSIDE

QAIM (Integrated Qubit Allocation and Initial Mapping)

QAIM combines the qubit allocation and initial mapping procedures in a single step and seeks to achieve three objectives:

1. Qubit allocation

2. Initial mapping

3. Achieving the first two objectives in a scalable way

QAIM uses efficient heuristics that exploit the following two:

Hardware Profiling

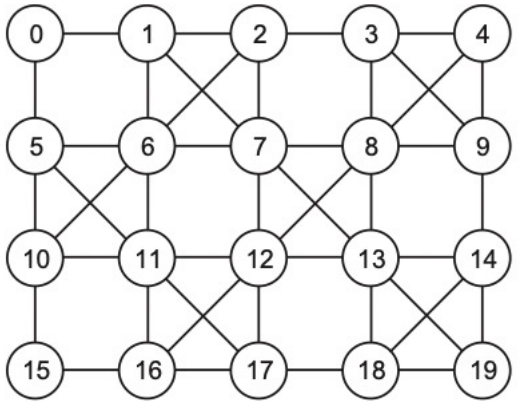
Program Profiling

QAIM

Hardware Profiling:

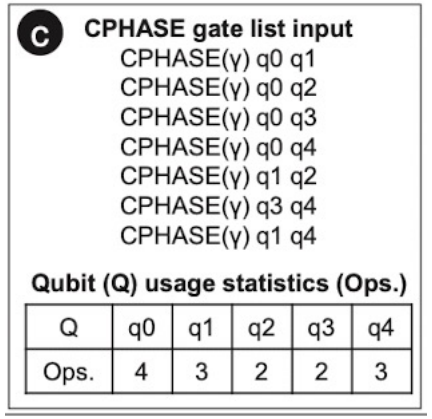
If the neighboring physical qubits have sparse connectivity, the logical qubits mapped to them may need to move frequently to meet the coupling constraint

To address this issue, the author defined **connectivity strength**:
The sum of its first and second neighboring qubits
(For instance, qubit-0 pic, the connectivity strength of qubit-0 is 7 (=2+5).



Program Profiling :

The program profile used in QAIM is similar to GreedyV. For any input QAOA-circuit, we calculate the number of CPHASE operations per logical qubit to create the program profile.



Q	q0	q1	q2	q3	q4
Ops.	4	3	2	2	3

UC RIVERSIDE

QAIM Procedure

Procedure:

1. The logical qubits are sorted (descending order)

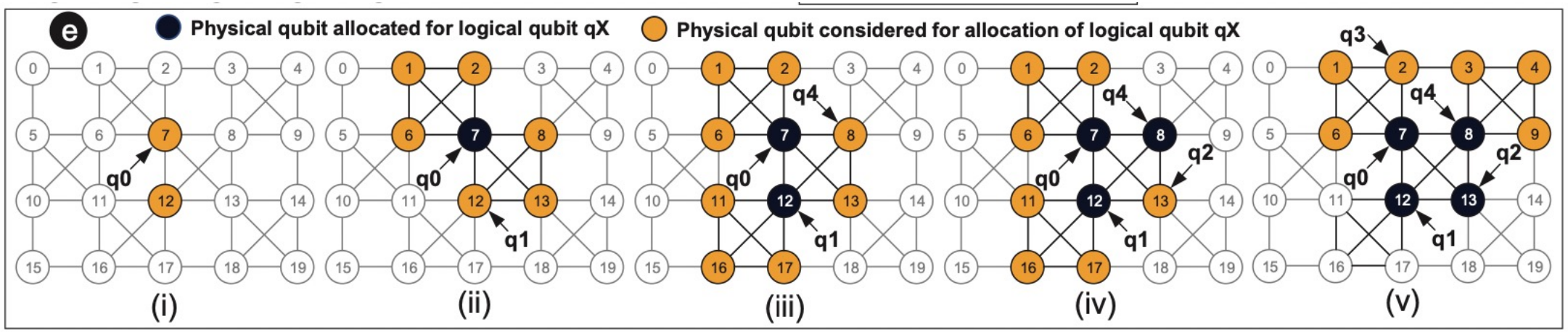
2. The first logical qubit is assigned to the physical qubit with the highest connectivity strength.

3. For the next logical qubit in the list, we check if any of its logical neighbors has ben already placed.
Else;
we pick the unallocated physical qubit with the highest connectivity strength for allocation.

Allocate physical qubits in order →
{q0, q1, q4, q2, q3}

d

If any logical neighbor placed →
Choose physical neighboring qubit with maximum
(qubit connectivity strength/ cumulative distance from placed neighbors)
Else →
Choose physical qubit with maximum
qubit connectivity strength



UC RIVERSIDE

Instruction Parallelization (IP)

After allocating physical qubits for the QAOA-circuit logical qubits using QAIM, we can go through the rest of the steps in the compilation procedure: SWAP insertion

IP: To maximize gate parallelization, the authors formulated the problem as a binary bin-packing problem and use the first- fit decreasing greedy heuristic.

- Because paralleling instructions in the QAOA-circuit may help to reduce the circuit depth.

IP also utilizes the program profile used in QAIM to rank the CPHASE operations (in descending order.

IP

Create the CPHASE layers

1. **Create MOQ empty layers of bins (each bin representing a qubit).**

2. **Take the operation from the sorted CPHASE list with the highest rank.**

3. **Repeat Step – 2 until the list is empty.**

4. **If unassigned CPHASE operations list is not empty, repeat from Step – 1 with this list.**

UC RIVERSIDE

IP Example

Create the CPHASE layers

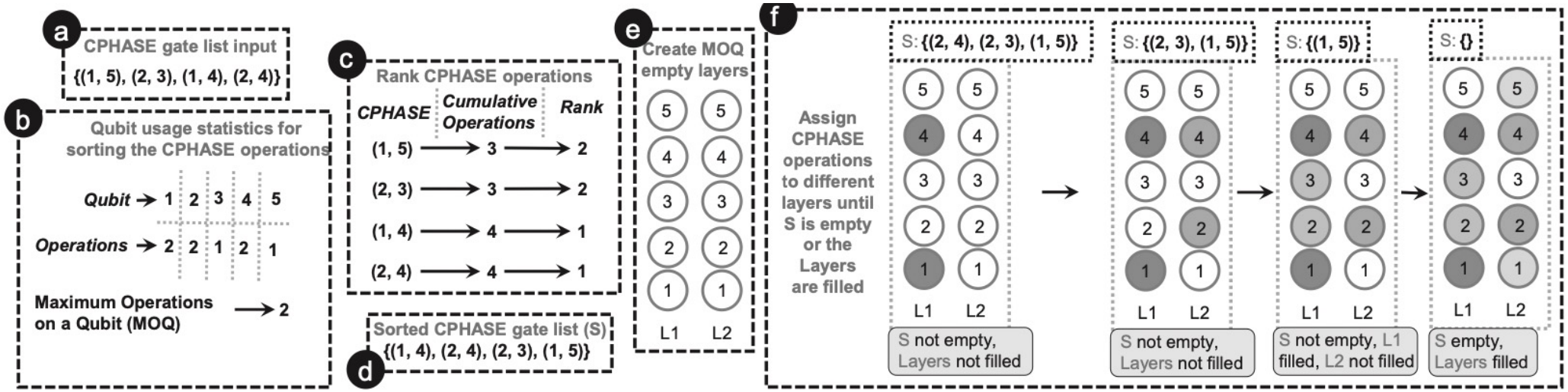
1. **The minimum number of required layers for this circuit (MOQ) for this example is 2 (a)(b)**

2. **The ranking of the CPHASE operations is shown in (c). In this example, cumulative operations for (2, 3) is '2 + 1 = 3'.**

2. **The sorted CPHASE operations (based on their ranks) are shown in Figure 4(d)**

4. The CPHASE operations in this sorted list are assigned one-by-one to the available qubit bins in the 2 layers (e)
how to assign?

5. In this example, the following CPHASE sequence is given as the input to the compiler (d)



Incremental Compilation (IC)

The authors proposed an incremental approach (IC) to compile QAOA circuits.

- The logical-to-physical qubit mapping changes with every added SWAP operation. Those are the dynamic changes.
- In IP, all the layers are formed at the same time and the complete circuit is passed to the backend for compilation. IC can greatly take advantage of these dynamic changes .

In IC, the circuit layers are formed one-at- a-time and partial circuits are compiled with single CPHASE layers. The partial circuits are stitched at the end to construct the whole circuit.

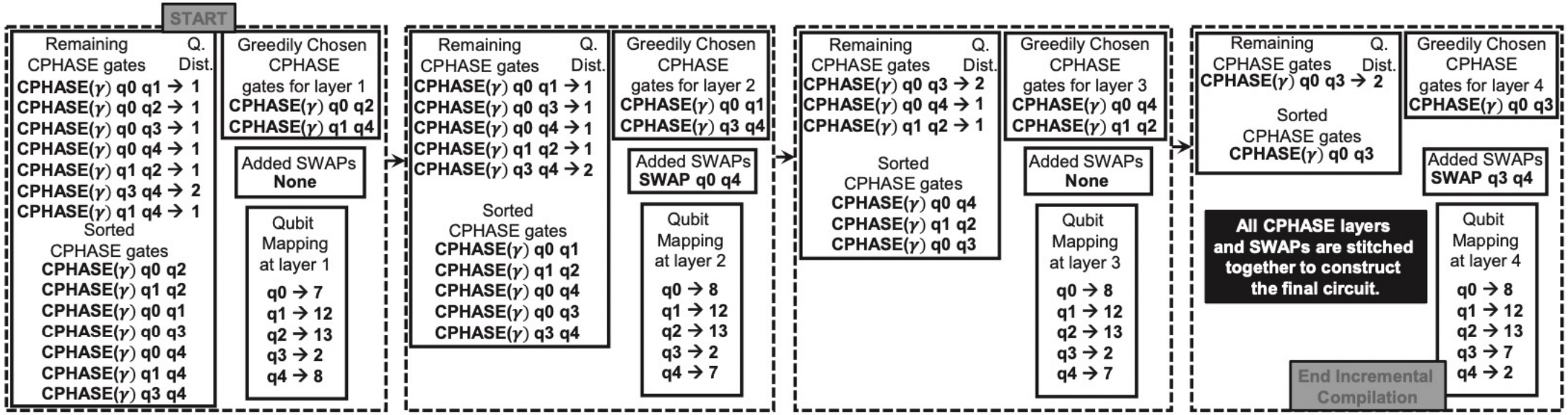
IC Steps and Example

1. **CPHASE operations are sorted in ascending order based on the distance between their control and target qubits for the initial logical-to-physical mapping.**

2. **A partial circuit is compiled with this layer and the initial mapping. The final mapping after SWAP insertion is saved.**

3. **The final mapping is set as the initial mapping. We repeat S tep – 1 until all the CPHASE operations are assigned to different layers and the corresponding partial circuits are compiled.**

4. **Stitch all the compiled partial circuits.**



RSIDE

Variation-aware IC (VIC)

Taking Variability into Consideration:

- To prioritize operations that may be executed with higher reliability during layer formation to maximize the success probability of the circuit.

VIC: To incorporate such strategy in IC, distance measurements between various qubits need to reflect the success probabilities of the intended operations. The higher the success rate, the lower the distance.

04 PERFORMANCE EVALUATION

1. Evaluation Metrics
2. Approximation Ratio Gap:
3. QAIM vs. GreedyV⋆ /NAIVE
4. IC/IP vs. QAIM
5. VIC vs. IC
6. Hardware Validation
7. Impact of Packing Density

Evaluation Metrics

Circuit depth:

A higher-depth circuit is more susceptible to decoherence errors

Gate-count:

A lower gate-count generally translates to less accumulation of gate errors.

Compilation time:

the time taken by the compiler to generate the hardware compliant circuit.

Success probability:

To quantify performance benefits with variation-aware compilation strategies.

Approximation Ratio Gap

The author proposed to a novel method that called Approximation Ratio Gap (ARG), to compare the performance of compiled QAOA-circuits on actual hardware.

- The conventional method to judge the quality of the compiled QAOA-circuits are time consuming.

- Sampling the output of the circuit a finite number of time to calculate the approximation ratio of the given cost function (r_0).

- Running the circuit on the target hardware and calculate the approximation ratio (r_h) using the formula $\{100 * (r_0 - r_h)/r_0\}$ as the approximation ratio gap or ARG.

- The lower ARG, the better, it indicates a performance closer to the noiseless scenario.

Problem sets:

Choosing 20-qubit ibmq 20 tokyo, 15-qubit ibmq 16 melbourne, and a hypothetical 36-qubit grid (6x6) architectures as the target hardware..

Randomly chosen (up to 36-nodes) erdos-renyi random graphs (with varied edge probabilities)

Regular graphs (with a varied number of edges/node) are used for the validation purpose

QAIM vs. GreedyV⋆ /NAÏVE

Varying Connectivity:
Edge probability from 0.1 to 0.6; edges/node from 3 to 8 (for regular graphs) and randomly picked 50 20-node graphs.

For sparse graphs , QAIM performs considerably better than both the NAIVE and GreedyV⋆ approach.

For dense graphs, all these three approaches perform similarly.

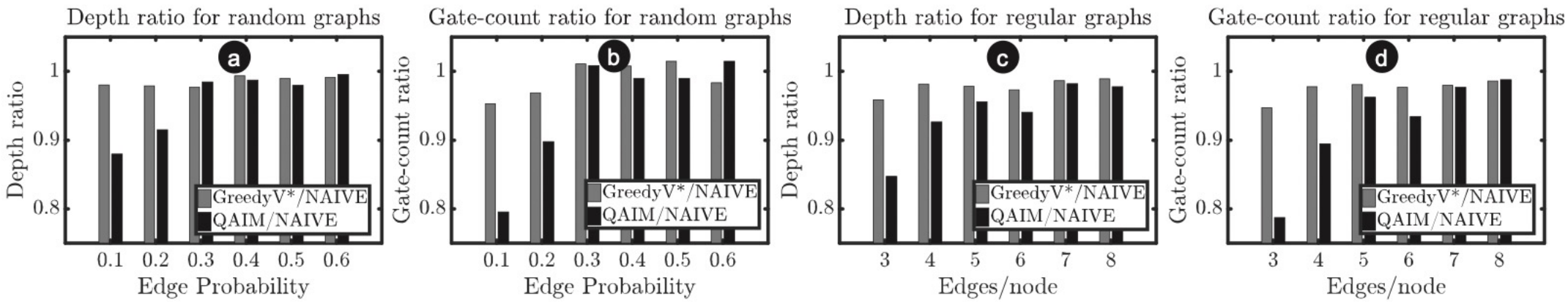


Fig. 7. Comparison among NAIVE, GreedyV⋆ [59], and QAIM in terms of (a) depth and, (b) gate-count ratios for erdos-renyi random graphs; (c) depth and, (d) gate-count for regular graphs (mean of 50 randomly chosen 20-node MaxCut-QAOA instances for each bars, ibmq_20_tokyo target hardware, qiskit backend, and a lower value is better).

VERSIDE

QAIM vs. GreedyV⋆ /NAÏVE

Varying Problem Size:

By picking 3-regular graphs with the number of nodes varying between 12 to 20

For smaller problem sizes both GreedyV⋆ and QAIM performed better than NAIVE.

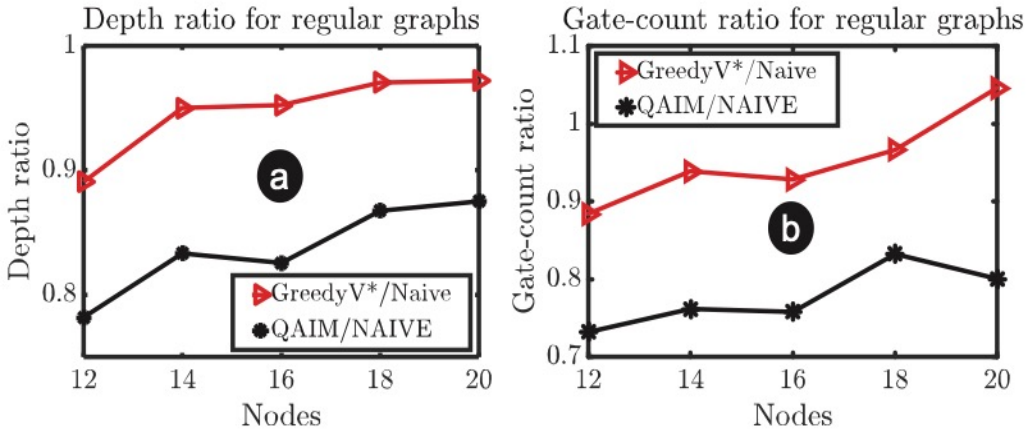


Fig. 8. Comparison among NAIVE, GreedyV⋆ [59], and QAIM in terms of (a) depth and, (b) gate-count ratios for 3-regular graphs with varying problem sizes (mean of 20 randomly chosen MaxCut-QAOA instances for each data points, ibmq_20_tokyo target hardware, qiskit backend, and a lower value is better).

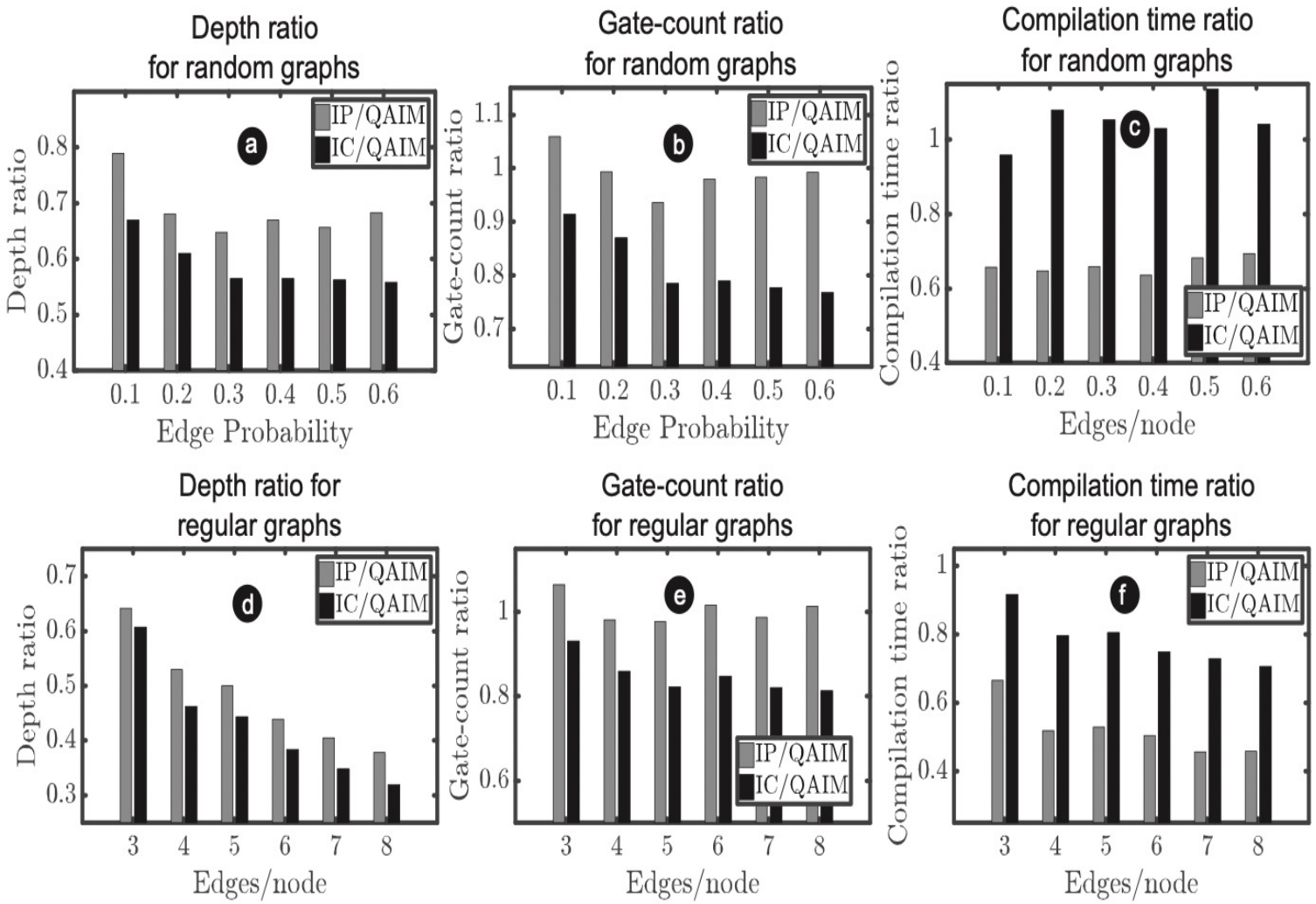
UC RIVERSIDE

IC/IP vs. QAIM

To quantify benefits of IP and IC over QAIM-only compilation

Result: both IP and IC generated circuits with significantly smaller depths than the QAIM-only approach and the differences were more pronounced for dense graphs.

BUT IP circuits have almost identical gate-counts to QAIM.



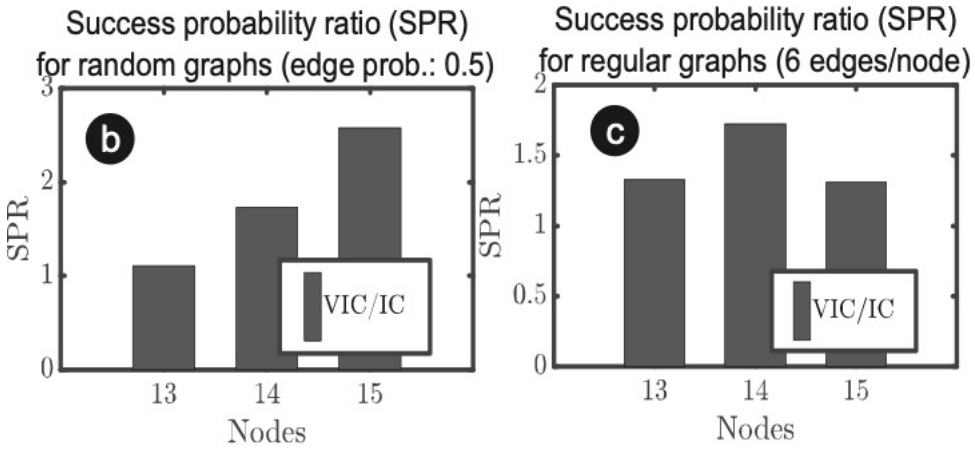
UC RIVERSIDE

VIC vs. IC

VIC adds variation awareness to IC with the goal to increase the success probability of the compiled circuits.

Result:
VIC showed an 80% better success probability over IC on average for the random graphs.

The improvement in the success probability is found to be quite smaller for the regular graphs (where layers were heavily packed



UC RIVERSIDE

Performance Summary

On average, 45% reduction in the compilation time has been achieved using IP over the NAIVE approach.

IC and VIC both provided similar performance in reducing depth (by ≈53%), gate-count (by ≈23%), and compilation time (by ≈15%).

VIC and IC show similar performance, but VIC offers higher success probability than IC (b)(c)

Method	Circuit Depth	Gate-count	Comp. time
NAIVE	1	1	1
QAIM	0.95	0.94	~1
IP	0.54	0.92	0.55
IC	0.47	0.77	0.85
VIC	0.48	0.77	0.86

Average performance over 600 20-node graphs (erdos-renyi + regular)

a

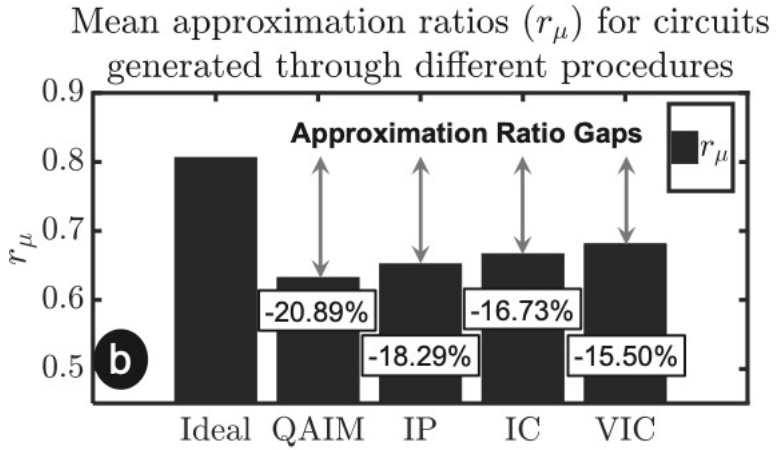
UC RIVERSIDE

Hardware Validation

The authors used the proposed strategies to compile the circuit and used the results to calculate their corresponding ARG's

Result: VIC (+QAIM), IC (+QAIM), and IP (+QAIM) all provided smaller ARG values compared to the QAIM-only circuits (b)

The results reflect the improvement in the compiled circuits in terms of depth, gate-count, and success probability.

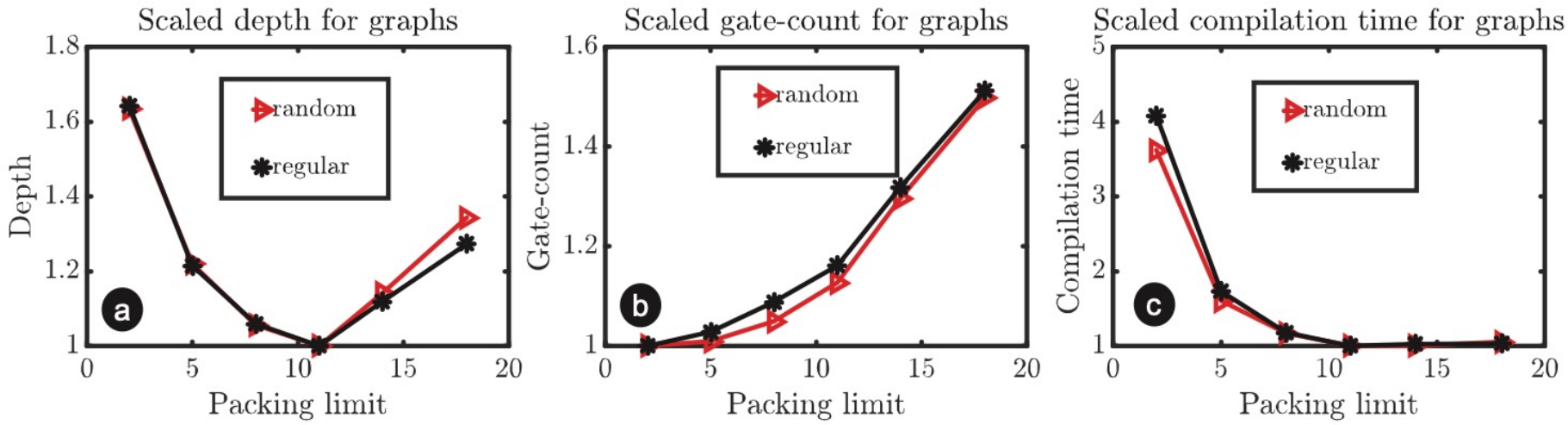


Impact of Packing Density

The mean depth:
the depth tends to decrease with an increase in the packing limit at first. However, when it went beyond 11, it went up again.

The gate count:
It is found to increase with packing limit as shown in (b)

The compilation time:
Packing more operations reduces the total number of circuit layers as well as the number of required SWAPs or permutation layers, so the compilation time decreases.



05 CONCLUSION

The author present four novel methodologies to compile QAOA circuits.

The methodologies can be integrated with any conventional compiler to improve the quality of the compiled QAOA- circuits.

The author validate performance improvement through experiments on real quantum devices from IBM. And demonstrate up to ≈53.0% reduction in circuit-depth, ≈23% reduction in gate- counts and ≈45% reduction in compilation time over a NAIVE approach. We also demonstrate up to ≈25.8%.

06 CONTRIBUTIONS

Presented a novel Qubit Allocation and Initial Mapping (QAIM).

Proposed a greedy heuristic for Instruction Parallelization (IP) to reduce the circuit depth and execution time.

Proposed an Incremental Compilation (IC) technique that reduces the need for added SWAP operations utilizing the dynamic changes

Proposed a Variation-aware Incremental Compilation (VIC) technique that enhances compiled circuit success probability

Present detailed comparative analysis of the proposed methodologies in terms of circuit quality metrics.

THANKS | Q&A

UC RIVERSIDE