Greedy Reliability-Aware Qubit Routing

Kevin Volkel

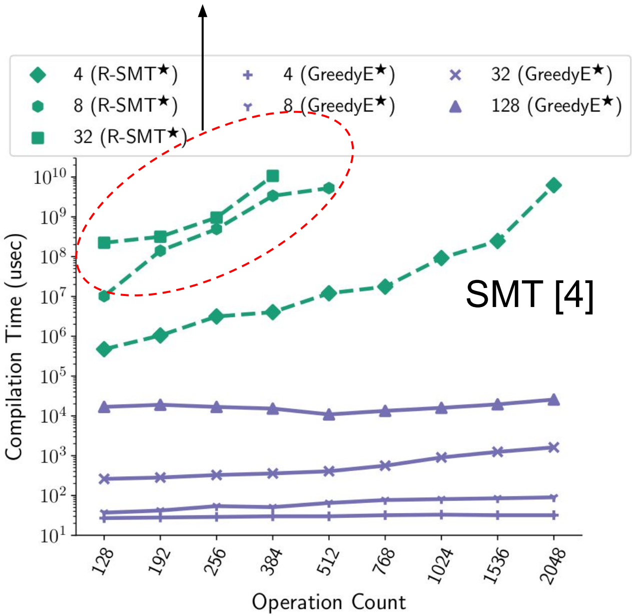
Outline

- Motivation
  - Computational efficiency need for greedy approach
  - Gaps in current reliability routing work
- Introduction to mapping and routing
  - Intro to Sabre(Swap)
- Modifying greedy score heuristics for reliability
  - Heuristic cost function formulations
  - Reliability matrix calculations
  - Ensuring convergence
  - Including single qubit ops and decoherence
- Evaluation
  - Program dependent pass selection
  - Machine dependent pass selection

The Need for Greedy Reliability Routing

- The problem of finding the optimal SWAPs for qubit routing is NP-complete
  - Existing A* state space search methods and SMT solvers are not scalable even at reasonable program sizes
    - A* searches are O(exp(N)) time complexity
    - SMT solvers → 2.7 hours for 32 qubit programs
  - Sabre Swap instead has $O(N^{2.5})$ time complexity
    - Can complete 35k gate programs in 31 seconds
    - Achieves comparable or better gate counts compared to A* searches
    - Readily available in Qiskit open source

10 s - 2.8 hours, O(exp(ops))



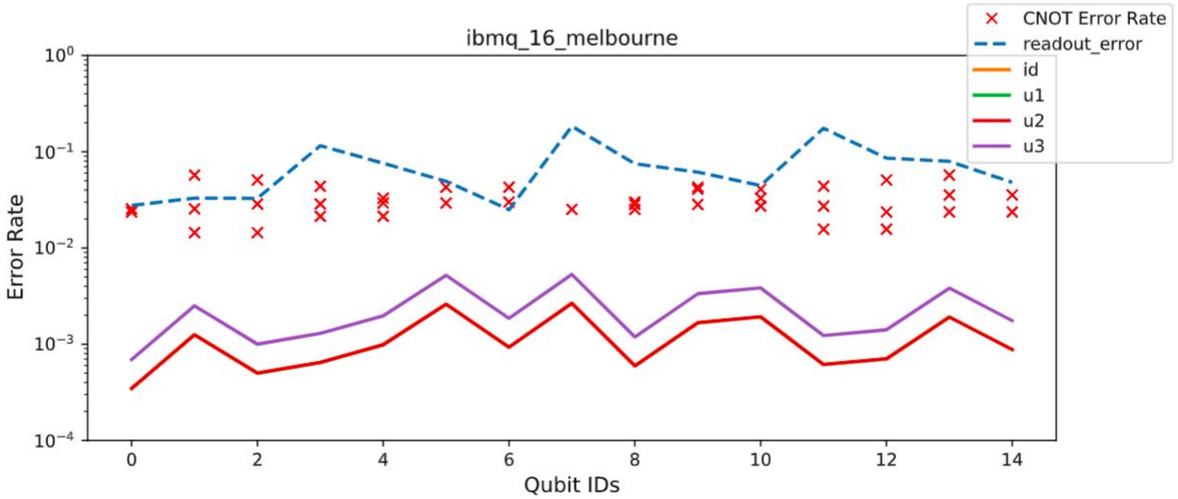
SMT [4]

[1] Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers,
[4] An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures,
[6] Qubit allocation

Gaps in Existing Noise Aware Router Work

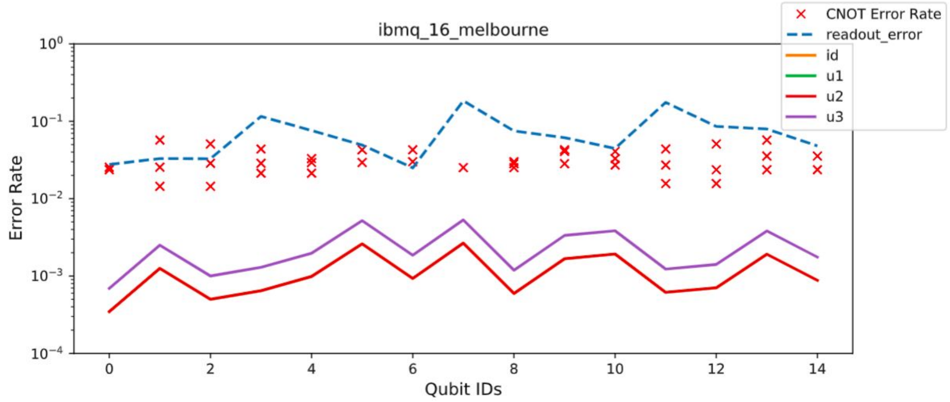
- Most approaches only consider readout and CNOT error rates
  - Single qubit operations typically ignored
    - Although ~1 order magnitude more reliable, probability of success scales $O(\varepsilon^t)$: $\varepsilon < 1$
    - Also exhibits high variability: Melbourne → min = $7\text{x}10^{-4}$ max =0.004

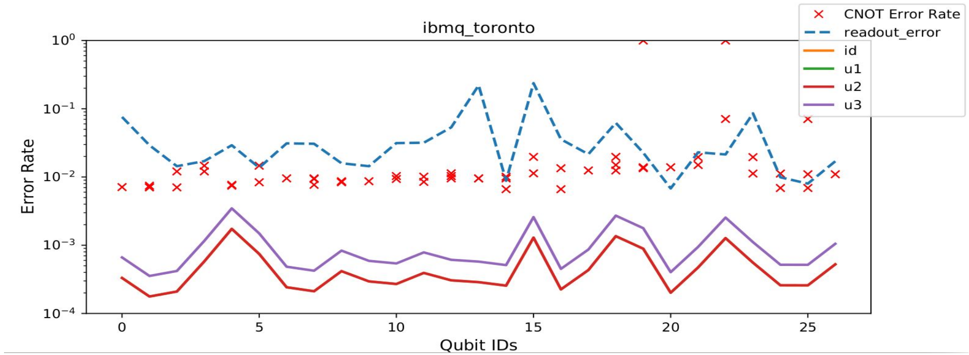
$P(\text{success,min}) = (1-7\text{x}10^{-4})^{100} = 93\%$ $P(\text{success,max}) = (1-0.004)^{100} = 67\%$



Gaps in Existing Noise Aware Router Work (Cont.)

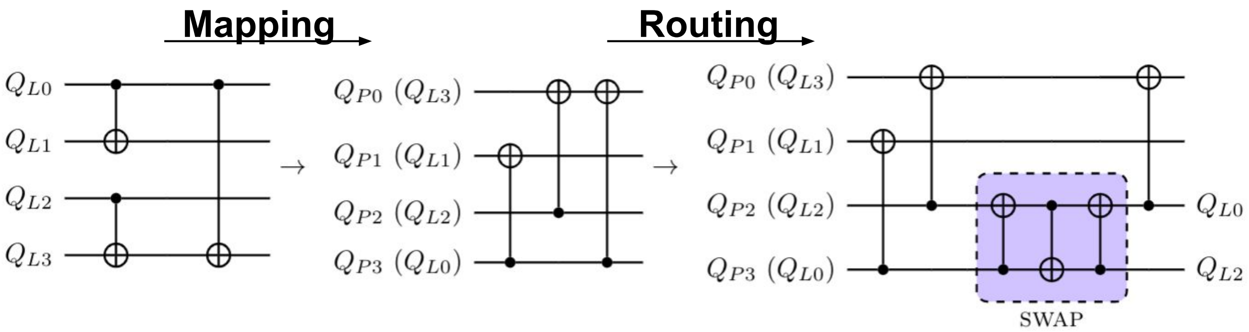
- Current works do not consider evaluation of multiple quantum machines
  - Error patterns vary machine to machine
  - Other characteristics vary as well
    - $T_{1,avg}$ (Mel) = 56 us
    - $T_{1,avg}$ (Tor) = 94 us
    - Tor. avg CNOT error rate = 4.8%
    - Mel. avg CNOT error rate = 3.2%
- What does this mean for choice of reliability routing heuristic cost functions?





Introduction to Mapping and Routing

- ***Mapping*** is the is the process of picking the initial placement of physical qubits for a set of logical qubits
  - Effectively a wire reordering for the initial physical circuit
- ***Routing*** ensures all quantum states are in appropriate physical locations at the moment of multi-qubit gates
  - This is the main focus of this work, *Qiskit* already has a noise aware ***mapper***

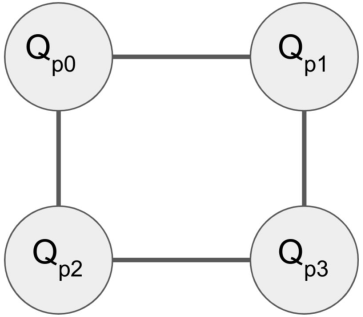


Logical Circuit

Initial Physical Circuit

Final Physical Circuit

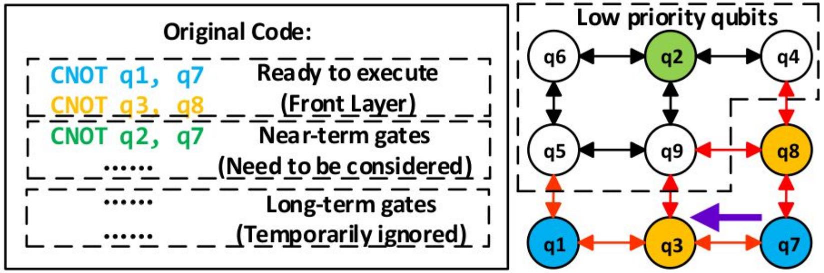
Coupling Graph

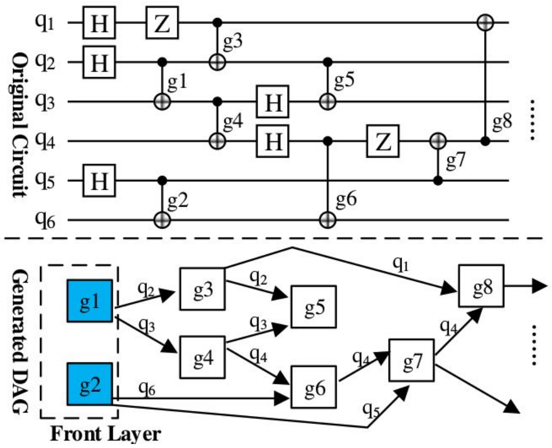


Introduction to Sabre Swap

- Greedy strategy for developing a mapping *and* routing
  - Uses three passes of the router to develop an initial mapping and a final routed circuit
  - Implemented in Qiskit as 2 individual modules to allow configurability of the mapping passes with different routers
  - Greedy since it picks SWAPs iteratively where each minimizes the overall distance of qubits that need to be routed

- **General idea:** Construct a topological list of 2-q gates, determine low/high priority qubit sets, search all SWAPs around high prio qubits, pick best SWAP for some cost





[2] Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices

Sabre Algorithm

- Consists of 3 main tasks
  - 1. Search for finished front layer gates
    - Checks to see if the gates in *F* are neighbors on coupling graph *G*
    - Single qubit gates are automatically executable
    - Creates an execution list
  - 2. Use execution list to derive next gates of *F*
    - Executable list gates serve as start points for finding the next gates to route
    - All gate's qubits must be finished routing up to this point
    - *F* iteratively checked until no more executions available
  - 3. Find the best SWAP to apply to the layout
    - Searches all possible candidates based on gates in *F*
    - SWAP that minimizes the heuristic cost is chosen
    - Go back and check if anything can execute now

Algorithm 1: *SabreSwap* [6]

```
Input: Front Layer F, Current Mapping π, Distance Matrix D,
Circuit DAG,Circuit New_DAG, Chip Coupling Graph G(V,E)
Output: Inserted SWAPS, Final Map π_f
/* assume F filled with independent gates from the front of the DAG
   */
while F ≠ ∅ do                                              1
    execution_list = ∅;
    for gate g in F do
        if π satisfies g.qubits then
            execution_list.append(g);
        end
    end                                                               2
    if execution_list ≠ ∅ then
        for gate e in execution_list do
            /* Get dependent gates                                   */
            successors = DAG.get_successors(e);
            F.remove(e);
            New_DAG.apply(e)
            for gate s in successors do
                if s.dependencies resolved then
                    /* Gate s is next to be routed                   */
                    F.append(s)
                end
            end
        end
        /* Go back and check if more can be executed      3         */
        Continue;
    else
        /* Noting to execute, need to route                          */
        score_list = [ ];
        /* Get viable SWAPs of F                                     */
        SWAP_candidates=Obtain_SWAPs(F,G);
        for SWAP in SWAP_candidates do
            π_temp = π.copy()
            π_temp.update(SWAP)
            score_list[SWAP] = Heuristic_Cost(F,SWAP,D,π_temp, E)
        end
        best_SWAP=score_list.index_of(min(score_list))
        New_DAG.apply(best_SWAP)
    end
end
```

[2] Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices

Base Sabre Cost Heuristic

- Built around three main components
  - 1. A Decay factor that discourages the use of the same logical qubit in a serialized manner
    - Small factor (1.001) used to slightly bias away from a previously used qubit

$$Heuristic\_Cost(F, SWAP, D, \pi, E) = max(decay(SWAP.q_1), decay(SWAP.q_2))\{ \frac{1}{|F|} \sum_{gate \in F} D[\pi(gate.q_1)][\pi(gate.q_2)] + \frac{W}{|E|} \sum_{gate \in E} D[\pi(gate.q_1)][\pi(gate.q_2)]\}$$

[2] Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices

Base Sabre Cost Heuristic

- Built around three main components
  - 1. A Decay factor that discourages the use of the same logical qubit in a serialized manner
    - Small factor (1.001) used to slightly bias away from a previously used qubit
  - 2. A front layer score which sums up the physical distances between all logical qubits to be routed using the pre-calculated matrix *D*
    - Main term used to pick a SWAP that best benefits all gates in *F*

$$Heuristic\_Cost(F, SWAP, D, \pi, E) = max(decay(SWAP.q_1), decay(SWAP.q_2))\{ \frac{1}{|F|} \sum_{gate \in F} D[\pi(gate.q_1)][\pi(gate.q_2)] + \frac{W}{|E|} \sum_{gate \in E} D[\pi(gate.q_1)][\pi(gate.q_2)]\}$$

[2] Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices

Base Sabre Cost Heuristic

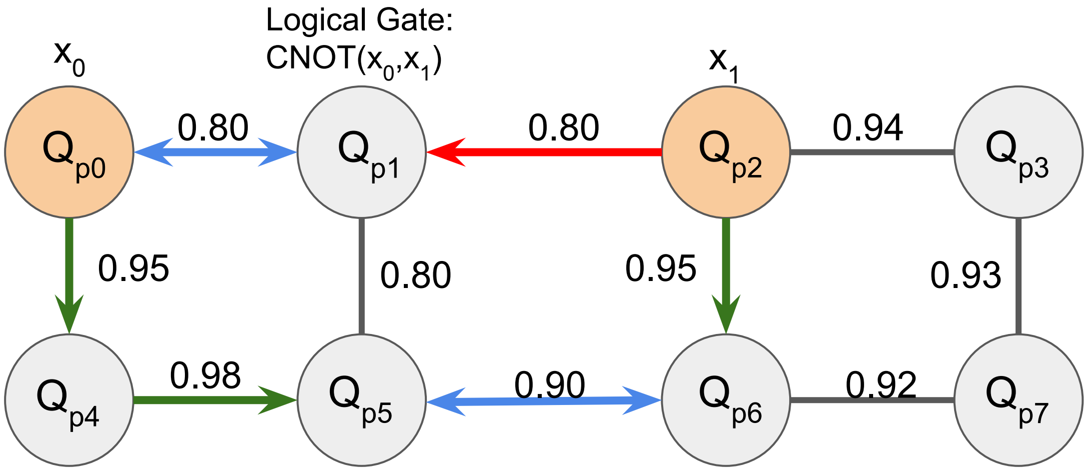
- Built around three main components
  - 1. A Decay factor that discourages the use of the same logical qubit in a serialized manner
    - Small factor (1.001) used to slightly bias away from a previously used qubit
  - 2. A front layer score which sums up the physical distances between all logical qubits to be routed using the pre-calculated matrix *D*
    - Main term used to pick a SWAP that best benefits all gates in *F*
  - A extended layer score that considers the logical qubits succeeding *F* that will need to be routed in the future
    - Helps pick SWAPs that will help reduce work to be done when gates in $E \rightarrow F$

$$Heuristic\_Cost(F, SWAP, D, \pi, E) = max(decay(SWAP.q_1), decay(SWAP.q_2))\{ \boxed{\frac{1}{|F|}\sum_{gate \in F} D[\pi(gate.q_1)][\pi(gate.q_2)]} + \frac{W}{|E|}\sum_{gate \in E} D[\pi(gate.q_1)][\pi(gate.q_2)]\}$$

[2] Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices

Heuristics for Reliability Routing

- Sabre focuses on finding the SWAPs that are the best from a geometric perspective
  - The overall objective is minimizing gates and depth, not maximizing success rate
- Maximizing success rate involves choosing the path of SWAPs with the minimum error rate
  - The minimum error rate path may not be the shortest
  - Most successful path not just required, but need most successful ***combination***
    - Leveraging this can lead to divergence if not careful



Sabre:
SWAP(Q_{p2},Q_{p1})
CNOT(Q_{p1} ,Q_{p0})
Success rate = $0.8^3 * 0.8 = 40.9\%$

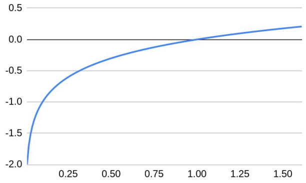
Reliability Aware:
SWAP(Q_{p2},Q_{p6})
SWAP(Q_{p0},Q_{p4})
SWAP(Q_{p4},Q_{p5})
CNOT(Q_{p5} ,Q_{p6})
Success rate = $0.98 * 0.95^6 * 0.90 = 64.8\%$

Formulating Probability of Success as a Distance

- Instead of pushing the layout to shorter distances, push to higher success rates
- SWAP path success probabilities are maximized when qubits are neighbors
  - Convergence guaranteed: shorter geometric distances still more favorable
  - Switch *D* for a matrix *R* that reflects the probability of success when moving between 2 qubits
- ***Catch:*** Program success is multiplicative instead of additive like distance
  - Leveraging algorithms like Dijkstra and Floyd-Warshall to construct *R* not useful for multiplication
  - Use negative-log-probabilities instead of direct probabilities

$Path_1$ =P(success) = A*B
$Path_2$ = P(success) = C*D
P(x)>P(y)

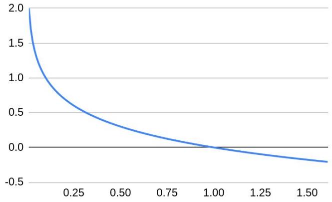
log(A*B) = log(A)+log(B)
log(C*D) = log(C)+log(D)
log(A*B)>log(C*D)



This is still a maximum distance problem

-log(A*B) = -log(A)+-log(B)
-log(A*B) = -log(A)+-log(B)
-log(A*B)<-log(C*D)

Highest probability path is now minimum distance problem, weights all positive



Calculating Reliability Matrix R

- Create a reliability graph from the chip coupling graph *G* and the chip noise information *N*
  - Vertices are still physical qubits
  - Edge weights are negative-log-probabilities of SWAP gates
- Use reliability graph and dijkstra to calculate entries for *R* and *P*
  - R[i] [j] = cost of highest reliability path between physical qubit *i* and physical qubit *j*
  - P[i] [j] = path of physical qubits on most reliable path

Algorithm 2: Constructing Reliability Matrix R and Path Matrix P

```
Input: Chip Coupling Graph G(V, E), Chip Noise Information N
Output: Reliability Matrix R, Shortest Path Matrix P
reliability_graph=∅
/* Let any vertex in G and reliability_graph be equivalent to a
   physical qubit.                                                  */
for v in G do
    if v not in reliability_graph then
        reliability_graph.add_vertex(v);
    end
    for v_n in v.neighbors do
        if edge (v,v_n) in reliability_graph then
            Continue;
        end
        CNOT_error_rate= N.gate(v, v_n);
        neg_log_prob_SWAP = −3 · log(1 − CNOT_error_rate);
        /* Add edge with weight = neq_log_prob_SWAP                 */
        reliability_graph.add_edge(v,v_n,neg_log_prob_SWAP);
    end
end
/* Have graph with edges representing -log success rates of SWAPs,
   now create R and P                                               */
R = [ ][ ]
for v_s in reliability_graph do
    for v_d ≠ v_s in reliability_graph do
        R[v_s][v_d] = reliability_graph.dijkstra(v_s,v_d).length();
        P[v_s][v_d] = reliability_graph.dijkstra(v_s,v_d).path();
    end
end
```

Handling Path Asymmetry

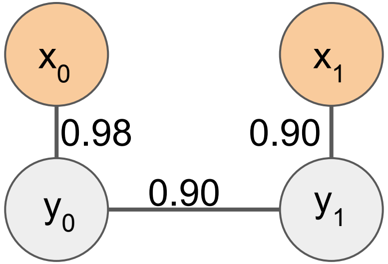
- Sabre naturally adds asymmetry by not considering cost of link taken
  - Leads to unwanted swaps taken, largest cost SWAP tends to get taken
- Could change heuristic to add in SWAP cost, but *R* is symmetric
  - Using asymmetric *R* that downgrades large cost links can cause oscillation
- Use algorithm that detects movement down best paths and changes orientation of score

Sabre Asymmetry

$\text{SWAP}(x_0,y_0)$ Cost $= 0.9^3 * 0.9^3 = 53\%$

$\text{SWAP}(x_1,y_1)$ Cost $= 0.98^3 * 0.9^3 = 69\%$

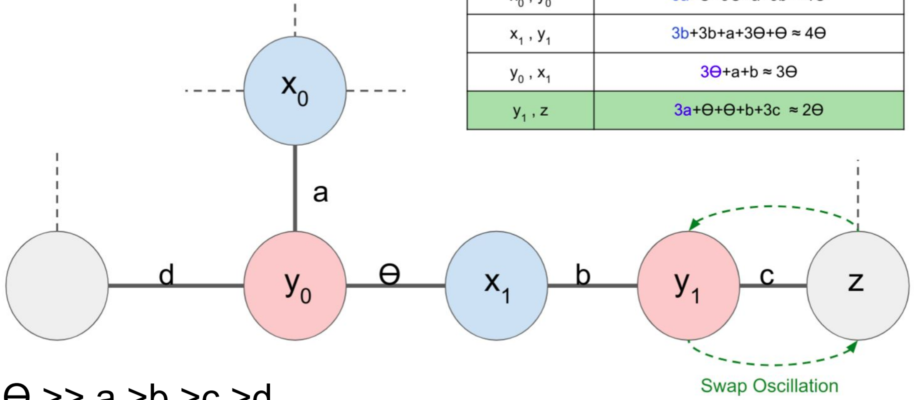
$\text{SWAP}(x_1,y_1)$ taken although it uses a higher cost link. 0.9 vs a 0.98 success rate



Asymmetric Matrix

Blue = Cost of SWAP   Black = remaining path cost

SWAP	Cost
x_0 , y_0	$3a+\Theta+3\Theta+a+3b \approx 4\Theta$
x_1 , y_1	$3b+3b+a+3\Theta+\Theta \approx 4\Theta$
y_0 , x_1	$3\Theta+a+b \approx 3\Theta$
y_1 , z	$3a+\Theta+\Theta+b+3c \approx 2\Theta$

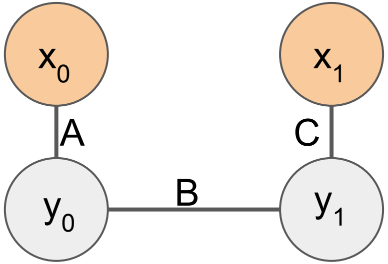


$\Theta >> a > b > c > d$

Asymmetry Algorithm

- Calculates an initial cost for distance between qubits for each gate
- Checks if gate qubits are involved in swap
    - Check if SWAP moves along the best path
- If moving down best path add in subtractive term
    - Otherwise don't modify cost
- Use path matrix to calculate subtraction term, add cost of actual SWAP used

$SWAP(x_0,y_0) = (B+C) - C + A$
(final cost)

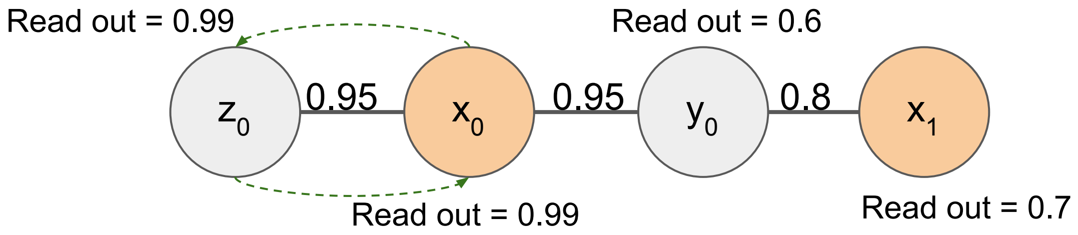


Algorithm 3: Cost function algorithm to adjust for the cost of the SWAP applied.

```
Input: Front Layer F, Mapping after SWAP π,
Mapping Before SWAP π_i, Reliability Matrix R, Path Matrix P
Output: Cost of SWAP C
/* NOTE: negative index indicates python-type reverse indexing     */
C = 0;
for gate g in F do
    C = C + R[π(g.q1)][π(g.q2)];
    for logical qubit q in g do
        if q not in SWAP.logical_qubits then
            /* Search for gate g that SWAP works on                */
            Continue;
        end
        source = q;
        destination = logical qubit in g not q;
        original_cost = R[π_i(source)][π_i(destination)];
        current_cost = R[π(source)][π(destination)];
        if current_cost > original_cost then
            /* Only consider SWAPs that move logical qubits closer */
            Continue;
        end
        /* get physical qubit path between qubits in g             */
        path = P[π(source)][π(dest)];
        next_to_last_qubit = path[-2];
        last_qubit = path[-1];
        unused_swap_cost = R[next_to_last_qubit][last_qubit];
        adjustment = R[π(SWAP.q1)][π(SWAP.q2)] - unused_swap_cost;
        C = C + adjustment;
    end
end
```

Accounting for Single Qubit Operations

- Read out error rates can be on the same order of magnitude as CNOT error rates
  - Unlike SWAP scores held in *R*, read out/single qubit error rates do not provide any information on physical positioning.
  - Cannot let single qubit operations dominate the cost in a way convergence cannot be achieved
- **Main Idea**: Collect all SWAPs that result in CNOTs being executable, *then* adjust the score of the SWAP using gates that will immediately execute there
  - SWAPs that fulfill routing requirements always taken: so single-qubit ops cannot lead to divergence
  - Does constrain possible SWAP space that can be used



SWAP(x_0,z_0) Cost = $0.95^3 * 0.95^3 * 0.8^3 * 0.99^2$ = 42%

SWAP(x_0,y_0) Cost = $0.95^3 * 0.6 * 0.7$ = 36%

SWAP(x_1,y_0) Cost = $0.8^3 * 0.6 * 0.99$ = 30%

Single Qubit Reliability Algorithm

1. Search list of SWAPs for all SWAPs that allow gates in *F* to be executed
   a. All gates in *F* executable for each *s* are collected
2. If no SWAPs lead to a gate in *F* executing, return base scores
3. Evaluate each swap that gets *filtered*, and add in the costs of executing the CNOT gate, single qubit gates, and measurements
   a. Algorithm proceeds along qubits that have ops that keep being satisfied
   b. Implementation maintains live and dead qubits
      i. Live: 2-q gate that cannot execute not found yet
      ii. Dead: Reached a 2-q gate not executable with π

Algorithm 4: Pass on SWAPs after initial scoring to take into account single qubit reliability

```
Input: Mapping Before SWAP π, List of SWAPs L, Front Layer F
Circuit DAG, Chip Coupling Graph G, Gate Reliability Table T
Output: SWAPs with adjust costs
filtered_swaps = ∅
for swap s in L do
    /* copy layout and apply s, put new layout in π_s        1  */
    π_s= π.apply(s);
    for gate g in F do
        phys_distance=G.distance(π_s(g.q1),π(g.q2))
        if phys_distance is 1 then
            /* collect swaps result in g's qubits being physical
               neighbors                                           */
            if s not in filtered_swaps then
                filtered_swaps.append(s);
            end
            /* collect all gates that execute on this swap          */
            filtered_swaps[s].executable_gates.append(g);
        end
    end
end
if filtered_swaps == ∅ then
    /* return swaps w/o changing their scores                2  */
    return L;
end
for swap s in filtered_swaps do
    executed_list = ∅;
    π_s= π.apply(s);                                         3
    for gate e in s.executable_gates do
        if e in execute_list then
            Continue;
        end
        executed_list.append(e)
        s.cost = s.cost + T[π_s(e.qubits)];
        successors = DAG.get_successors(e);
        for gate n in successors do
            if π_s satisfies n.qubits and n not in executed_list and
              n.predecessor in executed_list then
                s.cost = s.cost + T[π_s(n.qubits)];
                executed_list.append(n);
            end
        end
    end
    return filtered_swaps;
end
```

Factoring in Decoherence

- Decoherence is used to limited the amount of serialization on inserted SWAPs
  - Used as the reliability analog to the decay factor used in Sabre's base heuristic
- Uses qubit operation count list to estimate critical path length in DAG
  - Running time cost kept for all gates "executed" on a qubit
  - List entry is incremented for every operation on a qubit
- P(success) from decoherence modeled as $e^{-t/min(T1,T2)}$ $\rightarrow -\log(e^{-t/min(T1,T2)}) = t/min(T1,T2)$
- If SWAP is placed on qubit with most operations → add SWAP time with base time t_c
  - Else use the base time of the qubit with most operations, t_c
  - Similar considerations to single-q ops are made in the implementation to ensure convergence

$$Cost(SWAP) = HeuristicCost(SWAP) + \begin{cases} \frac{t_c}{min(T_1,T_2)} \\ \frac{t_c+3*t_{CNOT}}{min(T_1,T_2)} \end{cases}$$

Benchmarks and Evaluated Passes

- 3 program types: Adder, BV, QFT
  - 2 instances of BV and QFT, 6 qubits and 8 qubits → 5 total programs
  - 3 different communication patterns
- 3 different machines: Toronto, Manhattan, Melbourne
- 14 different pass combinations applied to each program and machine
- All passes evaluated within the same calibration window for a machine

Benchmarks

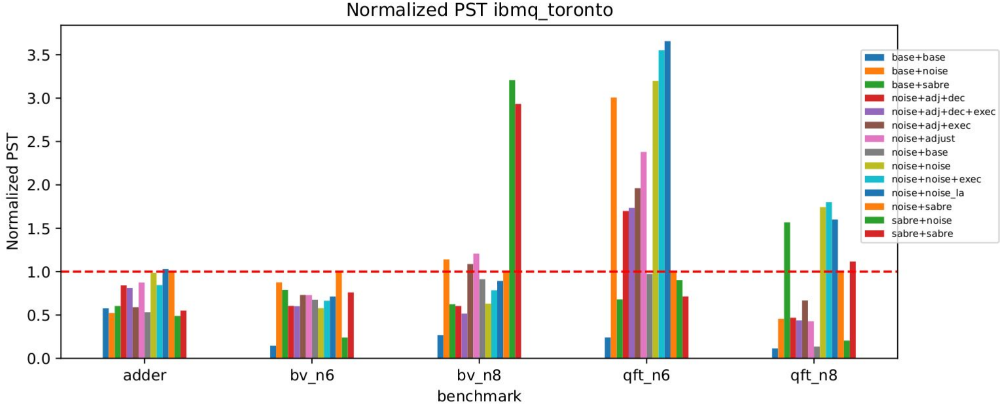
Benchmark	Input	Output	N	Communication	Pattern
Adder	$\|0001\rangle : \|1111\rangle$	$\|10000\rangle$	9	Short range	Nearest Neighbor
qft_n8	$\|0\rangle_8$	$\|0\rangle_8$	8	All distances	Multi-target Hubs
qft_n6	$\|0\rangle_6$	$\|0\rangle_6$	6	All distances	Multi-target Hubs
bv_n8	String = $\{1\}^7$	$\|1\rangle_7$	8	All distances	Single-Target Hub
bv_n6	String = $\{1\}^5$	$\|1\rangle_5$	6	All distances	Single-Target Hub

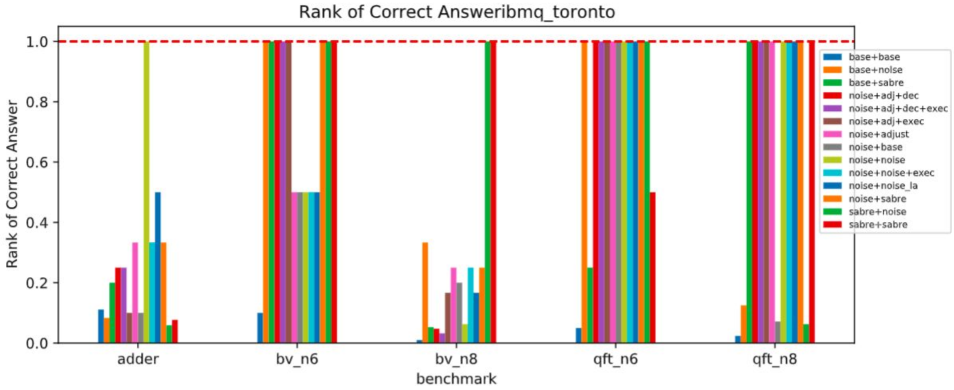
passes
noise+adj+dec
noise+noise+exec
noise+noise_la
sabre+sabre
noise+sabre
base+noise
noise+adjust
base+sabre
noise+adj+dec+exec
noise+adj+exec
sabre+noise
noise+noise
noise+base
base+base

Relative Program Success Probabilities

- All best performing passes with respect to PST take into account reliability information in some capacity
  - No clear best pass across all benchmarks
  - Studied later using average ranks
  - Qiskit pass combo is only the best for 1 benchmark (*bv_n6*)

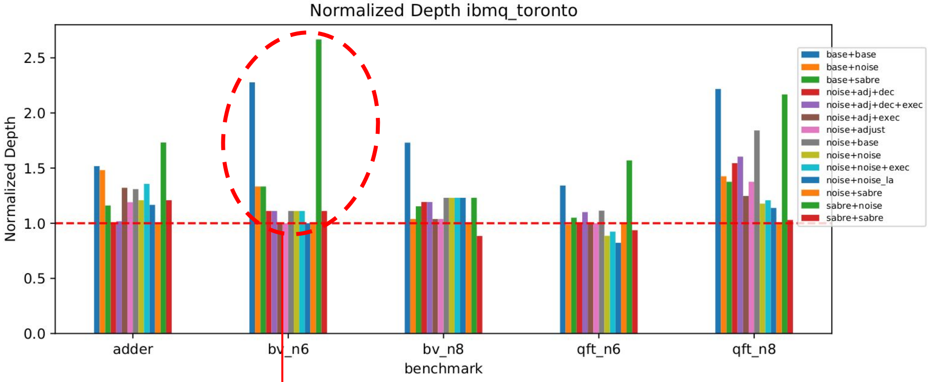
- Correct answer for *adder* is the most prominent only after a noise adaptive router pass is used

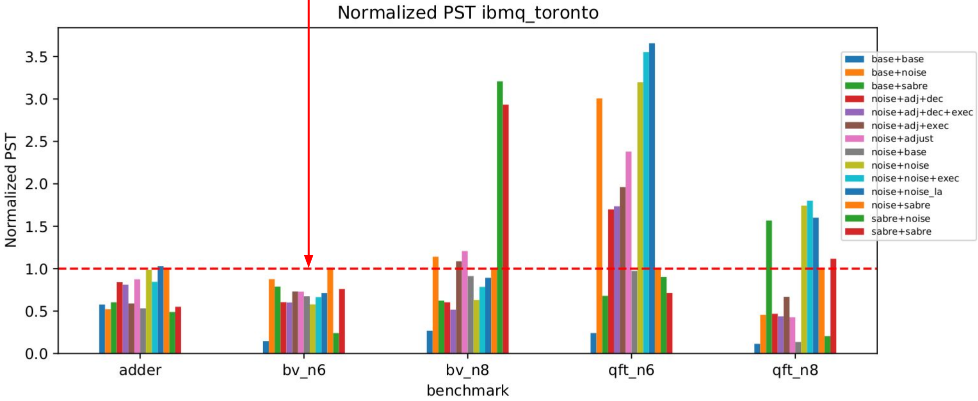




Impact of Circuit Depth

- Depth of the circuit after the compiler passes was found to correlate with achieved program success
  - All 1x depths for *bv_n6* depths have <1x program success rates
- Correlation with depth seems to be weaker as program depth increases
  - *qft_n8* ~100-200 layers
  - *adder* ~200-300 layers
  - *bv_n6* ~ 20 - 40 layers
  - *Adder qft_n8* best passes have >1x depth
  - Larger logical depth circuits can trade depth for more reliability, additional SWAP depth not as impactful





Machine Dependent Selection

- Manhattan and Melbourne seemed to have stronger correlation between depth and PST
- Man. and Mel both prefer *noise+adj+dec*
  - This pass accounts for decoherence effects
    - Mel & Man have ~50 us avg T_1
    - Toronto has ~100 us avg T_1 → decoherence less important
- On average, the best passes account for more than just SWAP reliability
  - 3 best passes on average not available in Qiskit

Average Pass PST Rank (Machines)

passes	Man.	Mel.	Avg Mel-Man	Tor.	Total Avg
noise+adj+dec	10.2	10	10.10	6.2	9.13
noise+noise+exec	10.4	7.4	8.90	9.8	9.13
noise+noise_la	8	8.4	8.20	11	8.9
sabre+sabre	10.8	7.4	9.10	8.2	8.88
noise+sabre	7.4	8	7.70	10.2	8.33
base+noise	5	10.6	7.80	8.6	8
noise+adjust	10.6	4.2	7.40	9.2	7.85
base+sabre	7	8.6	7.80	7.2	7.65
noise+adj+dec+exec	7	9.2	8.10	5.4	7.43
noise+adj+exec	8.2	5.2	6.70	8.6	7.18
sabre+noise	5.6	9.2	7.40	4.8	6.75
noise+noise	5.8	5.8	5.80	9	6.6
noise+base	6.8	6	6.40	5	6.05
base+base	2.2	5	3.60	1.8	3.15

Program Dependent Selection

- Performance of pass depends on algorithm
  - *Noise+base* works well for *bv* since the simple *bv* interactions can be embedded easily in a reliable set of paths
    - Clearly not a reasonable strategy for more complex communication patterns like *qft*
  - Simple localized communication patterns like *adder* seem to prefer a lookahead approach which may be able to leverage the simplicity to inject helpful information about future routings
  - QFT prefers a pass that accounts for both routing and single qubit reliability
    - qft_n8 has ~100 single qubit operations

Average Pass PST Rank (Benchmarks)

passes	adder	bv	qft
noise+noise+exec	6.67	7.50	12.17
noise+noise	7.33	4.17	9.33
sabre+sabre	6.00	9.83	9.17
base+noise	6.00	8.50	8.67
noise+noise_la	10.33	9.17	8.50
noise+adj+dec	9.33	9.00	8.33
noise+adj+exec	8.00	6.00	8.33
base+sabre	7.33	7.33	8.00
noise+sabre	9.33	8.83	7.83
noise+adj+dec+exec	6.33	7.17	7.67
noise+adjust	8.33	8.67	7.17
sabre+noise	9.00	6.50	5.33
noise+base	4.33	9.50	3.17
base+base	6.67	2.83	1.33

Conclusion

- The passes presented in this project provide improvements in reliability over the existing mapping and routing existing in Qiskit
- The best pass for program success varies by program and machine
  - Machines with lower T_1 are more sensitive to decoherence, prefer passes that try to control depth
  - Programs with simple qubit communication allow lookahead passes to make accurate assumptions on future reliability impacts
- In general reliability routing algorithms should consider all reliability factors including single qubit operations and decoherence

References

[1] A. Zulehner, A. Paler, and R. Wille, “An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, Jul. 2019, doi: 10.1109/TCAD.2018.2846658.

[2] 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*, New York, NY, USA, Apr. 2019, pp. 1001–1014, doi: 10.1145/3297858.3304023.

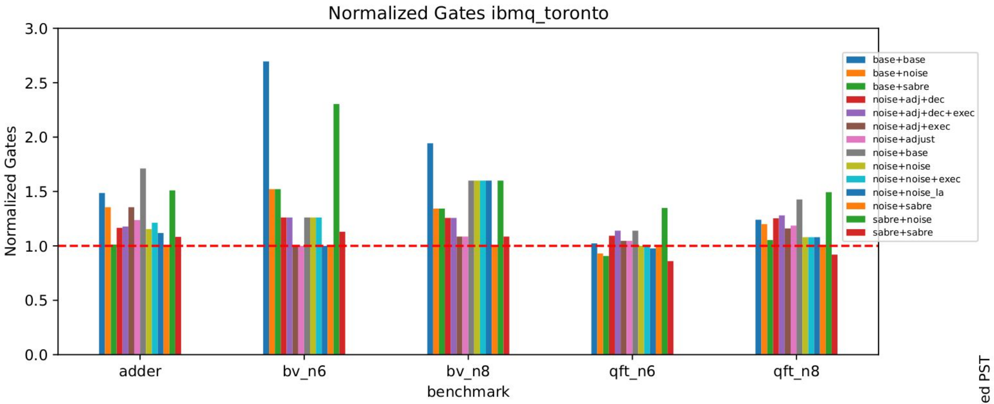
[3] S. S. Tannu and M. K. Qureshi, “Not All Qubits Are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers,” in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, Apr. 2019, pp. 987–999, doi: 10.1145/3297858.3304007.

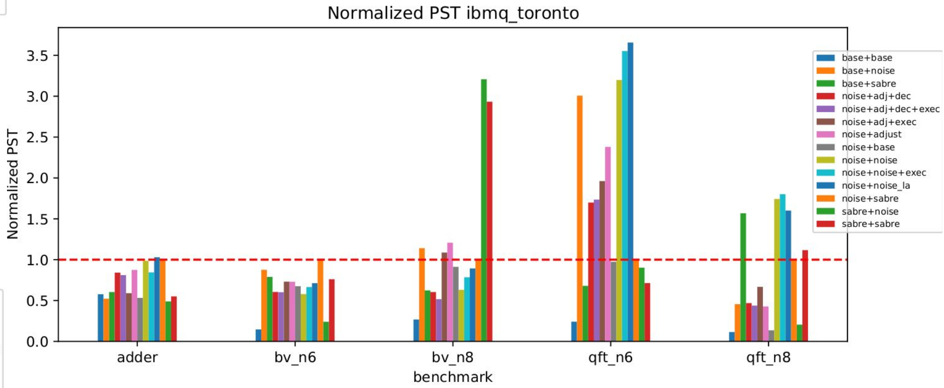
[4] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, “Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers,” in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, Apr. 2019, pp. 1015–1029, doi: 10.1145/3297858.3304075.

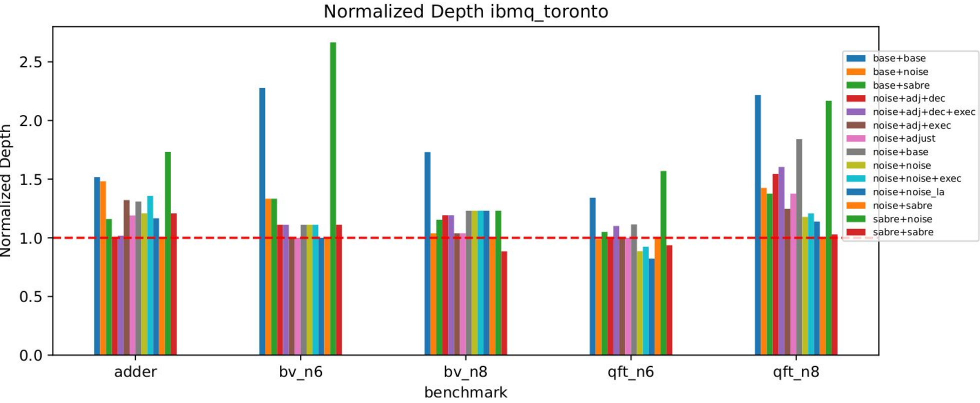
[5] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, “Full-stack, real-system quantum computer studies: architectural comparisons and design insights,” in *Proceedings of the 46th International Symposium on Computer Architecture*, Phoenix Arizona, Jun. 2019, pp. 527–540, doi: 10.1145/3307650.3322273.

[6] M. Y. Siraichi, V. F. dos Santos, S. Collange, and F. M. Q. Pereira, “Qubit allocation,” in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, New York, NY, USA, Feb. 2018, pp. 113–125, doi: 10.1145/3168822.

Backup Slides Tor. Gates vs Depth vs PST







Backup Slides Man. Gates vs Depth vs PST