



NORTHWESTERN UNIVERSITY

Computer Science Department

Technical Report
Number: NU-CS-2025-48

December, 2025

Title

Quantum compilation for hybrid modular architectures

Authors

Mingyoung Jessica Jeng

Abstract

As quantum computing technology matures, the pursuit of performance and scalability has led to the widespread adoption of modular quantum architectures in industry. We expect that the next stage of technological evolution will integrate multiple qubit modalities into these systems, producing hybrid, modular quantum architectures. A combination of qubit modalities could result in devices that combines the performance capabilities of their constituent technologies while mitigating their respective weaknesses. However, the complexity of hybrid, modular quantum devices, coupled with their growing size, presents an imminent scalability challenge for quantum compilation. Contemporary compilation methods are not well-suited to hybrid modular architectures — in particular, existing qubit allocation methods are often unable to contend with inter-module links, which don't necessarily support a universal basis gate set. Furthermore, existing methods of logical-to-physical qubit placement, swap insertion (routing), unitary synthesis, and/or optimization, are typically not designed for qubit links of significantly varying latency or fidelity.

In this work, we propose SEQC, a hierarchical parallelized compilation pipeline optimized for modular quantum systems, including several novel methods for qubit placement, qubit routing,

and circuit optimization. SEQC attains a 9.3–32.3% average increase in circuit fidelity (49.99–63.36% max), depending on the chiplet size and topology. Additionally, owing to its ability to parallelize compilation, SEQC achieves 1.34–3.27× faster compilation on average (3.37–6.74× max) over a chiplet-unaware Qiskit baseline.

Keywords

Quantum computing, quantum compilation, modular architectures

Acknowledgements

Thank you to **Nikos Hardavellas** and **Kaitlin N. Smith** for their expert guidance.

I would also like to thank the other students who also made significant contributions to this effort.

- (1) **Nikola Vuk Maruszewski** designed and implemented an earlier variant of the circuit splitting algorithm by formulating it as a constrained optimization problem, solved by Gurobi, and a distance-based precursor to the current splitting algorithm we use. Both of these algorithms were eventually abandoned, as they were fully subsumed by the STA-based algorithm we currently use. Nino also designed and implemented ANN-based splitting, which is included in this work as an alternative, but it is outperformed on all metrics by the STA-based algorithm. Finally, Nino contributed to writing the corresponding sections in the final draft.
- (2) **Michael Gavrincea** helped design an initial subcircuit-to-module mapping algorithm for the Gurobi-based approach, which was eventually subsumed by the one we currently use.
- (3) **Connor Selna** contributed to the Related Work section.

Table of Contents

ABSTRACT	3
Acknowledgements	5
Table of Contents	6
List of Tables	8
List of Figures	9
Chapter 1. Introduction	11
Chapter 2. Background and Motivation	17
Chapter 3. Compilation for Hybrid Modular Quantum Systems	20
3.1. Module-awareness via Peephole Correction	22
3.2. The Stratify-Elaborate Quantum Compiler	23
Chapter 4. Methodology	34
4.1. Benchmark Suite	35
4.2. Simulated Quantum Device Specifications	37
Chapter 5. Experimental Results	39
Chapter 6. Discussion	47

Chapter 7. Related Work

49

Chapter 8. Conclusion

53

Bibliography

54

List of Tables

4.1	Specifications of Simulated Modular Backend.	38
-----	--	----

List of Figures

3.1	Peephole correction for inter-module gates that are unimplementable on hardware (i.e., non-SWAP, shown in red).	22
3.2	High-level diagram of SEQC. Elements in red reflect modifications from existing compiler designs.	23
3.3	Step-by-step example of inter-module compilation in the Stratification stage.	24
3.4	Step-by-step example of intra-module compilation in the Elaboration stage. Gates that cannot be implemented on hardware are shown in red.	31
4.1	Examples of benchmark applications for 5-qubit circuits.	36
4.2	Modeled topologies.	37
5.1	Main circuit performance metrics: inter-module gate count and estimated fidelity, 10-qubit heavy-hex backend.	40
5.2	Main circuit performance metrics: inter-module gate count and estimated fidelity, 120-qubit grid backend.	41
5.3	Total gate count, circuit depth, and per-shot execution time, 10-qubit heavy-hex backend (lower is better).	42

5.4	Total gate count, circuit depth, and per-shot execution time, 120-qubit grid backend (lower is better).	42
5.5	Compilation performance metrics: compilation time, recurring compilation time, and peak memory utilization, 10-qubit heavy-hex backend (lower is better).	45
5.6	Compilation performance metrics: compilation time, recurring compilation time, and peak memory utilization, 120-qubit grid backend (lower is better).	46

CHAPTER 1

Introduction

Classical computer systems, through the decades of their existence, have become increasingly distributed. Physical, technological, and economic constraints have prevented single “monolithic” systems from scaling to the point that they could meet the demand for high performance and yet remain practical. Today, distributed architectures are prevalent, and their latest renditions, from cloud computing to chiplet-based digital processors, are ubiquitous.

We postulate that quantum computing is on a similar path. While improvements in superconducting quantum hardware have led to the debut of processors with 1000+ qubits [22], practical implementations of quantum computation will require millions of physical qubits [50]. The number of qubits required by a quantum algorithm, the depth of the algorithm (i.e., execution time or gate count on the critical path), the auxiliary and syndrome qubits required to reach sufficiently low error rates, and the size of qubits with all control hardware, all suggest that many more qubits are required than are likely to fit in a single die [70]. The overhead for error correction, cooling, dilution, control systems, I/O lines, challenges associated with verification and adequate chip thermalization, and realistic resources such as lossy waveguides, limited qubit fabrication yields as qubit capacity grows [62], and finite chip sizes, make the prospect of a single “monolithic” quantum processor very expensive or entirely unrealistic by current standards. These constraints force large-scale quantum systems to adopt a physically distributed architecture.

We are already observing signs of this shift. Recent developments in quantum chip linking, including flip-chip architectures [24] and low-loss coaxial cables [47], suggest that modular designs are the most viable for scaling [62]. Many contemporary or upcoming leading quantum systems adopt chiplet-based modular quantum processors, for example using high-bandwidth quantum links between nearest-neighbor quantum chiplets (e.g., carrier-chip couplers in Rigetti Aspen-M [59, 24], or m-couplers in IBM Crossbill [23, 40]), or lower-fidelity, lower-bandwidth, but longer-distance flexible coupling of discrete chips (e.g., l-couplers in IBM Flamingo [23, 40]), or a combination of the above (e.g., c-, m-, and l-couplers in IBM Starling [40]), or multi-chip connectivity through tunable couplers and routing chips in Rigetti Cepheus [67, 20]. Most major companies have set their sights at modular designs to meet quantum scaling targets in practical ways, and modular quantum processors dominate their latest roadmaps [41, 25, 56].

At the same time, no single qubit technology simultaneously offers long coherence, fast gates, high connectivity, and scalable, mature fabrication. Superconducting qubits offer very fast gate latencies and mature fabrication, but low coherence times and mostly local connectivity. Trapped ions have excellent coherence but slow speeds, and all-to-all connectivity but only for the few ions located at the C-cut sapphire window. Neutral atoms have excellent scalability and arrangeable connectivity, but do not reach the gate speeds and manufacturing robustness of superconducting qubits, or the coherence times of trapped ions. Hybrid, modular quantum architectures, comprising QPUs based on different qubit modalities, provide a path to combine the complementary strengths of each platform, enabling performance impossible for any single modality. For example, superconducting qubits can serve as fast computational QPUs, while ions can provide memory

QPU, and Rydberg atoms can provide ancillae and distillation functionality. Such heterogeneity mirrors the classical transition from monolithic CPUs to systems composed of CPUs, GPUs, NPUs, and specialized accelerators. As such, hybrid modular quantum systems are increasingly recognized as a pragmatic design path for long-term universal quantum computers [14, 68, 63, 73]. Efforts to realize hybrid quantum systems have predominantly leveraged quantum state transfer, i.e., SWAPs, between heterogeneous qubit technologies [33, 11, 77, 29]. While universal quantum operations, e.g., CNOT or CZ operations, that interface two different quantum technologies have been demonstrated [44, 73, 10, 13, 75], these often have to compromise on gate fidelity, and have yet to be realized for all combinations of common qubit technologies, e.g., between superconducting and trapped-ion qubits. Thus, hybrid quantum systems may exhibit interconnect heterogeneity, where inter-module links may not be able to support universal gate sets, while intra-module couplers do.

Unfortunately, current compilation infrastructure does not support quantum interconnect heterogeneity. The mapping of a quantum program’s logical quantum gates into the native gates supported by the underlying quantum processor, and the mapping of logical qubits to the physical qubits of the quantum hardware, largely determine the number and type of native quantum gates required for the computation, the circuit depth, and its execution time—long programs may not complete successfully as qubits decohere. Different mappings can result in vastly different compiled quantum circuits, with diverse characteristics along all these axes. These details of efficiency can make or break an algorithm in today’s noisy intermediate-scale quantum systems (NISQ) era, so quantum compilers aggressively optimize for all of them. To make matters more challenging, each qubit,

coupler, and gate have diverse error profiles that are highly variable both spatially and temporally [64, 45, 15]. Unlike classical compilation that is done only once for a given architecture, quantum programs must be recompiled every time before execution¹, as the ideal physical qubits to target change between runs. So, quantum compilers today are faced with the daunting task of optimizing quantum programs across multiple dimensions with often conflicting demands, in a continuously changing environment. Coupled with the hardness of synthesizing unitaries (which is exponential in the number of qubits) and the need for quick and frequent compilation, modern quantum compilers have no option other than to rely on heuristics to perform the task. These heuristics result in a compilation complexity that is quadratic to the number of qubits in the quantum processor (experimentally observed in Figure 5.5a). Hardware modularity, though, also presents an opportunity. Inspired by classical compilation, in this paper we leverage hardware modularity to achieve compilation modularity.

Compilation in classical systems is typically performed independently and in parallel for each source file, producing one object file per source. The individual object files are then linked together to construct the executable. We propose a compilation framework for modular quantum processors that works in a similar fashion: it **stratifies**, i.e., splits, the source quantum circuit into subcircuits small enough to fit in each module, and maps subcircuits to modules, and then *in parallel* **elaborates** each subcircuit and compiles it for its target module. This **Stratify-Elaborate Quantum Compiler (SEQC)** stratifies a source program only once for a given module architecture, and performs only the elaboration step recurrently before each execution.

¹The term “execution” here includes all shots required to produce a final result for the user. Different executions of the same circuit can span qubit calibration windows.

As the qubit capacity n of quantum processors grows exponentially, the $O(n^2)$ compilation latency of contemporary compilers rises even faster. SEQC replaces the recurrent $Oa(n^2)$ compilation step with several parallel $O(k^2)$ elaboration steps for k -qubit modules. The stratification step is $O(n^2)$, so the end-to-end complexity remains the same, but stratification is performed only once. The recurrent compilation in SEQC is only $O(k^2)$, and as the number of qubits k per module is expected to remain relatively stable, or grow at a much slower pace in the foreseeable future [41], SEQC’s recurrent compilation latency will barely grow as processors scale.

Additionally, as SEQC is cognizant of hardware modularity, it can stratify the source quantum circuit into subcircuits and map them to modules while minimizing cross-module communication. As we show in this paper, SEQC produces circuits with shorter execution times and significantly fewer inter-module gates compared to today’s stock compilers, leading to higher execution fidelity. More importantly, as the number of qubits in a processor grows, SEQC achieves even higher performance in these figures of merit.

In summary, the contributions of this paper are as follows:

- We make stock compilers aware of modularity through a “peephole” correction pass, thereby allowing them to correctly compile circuits for modular architectures with limited cross-module gate support.
- We design and implement SEQC, a Stratify-Elaborate Quantum Compiler for modular architectures. SEQC performs compilation in two stages, with the first stage (stratification, or module splitting) performed only once for a given architecture, and the second stage (elaboration, or module compilation) performed in parallel for each module. Only this second stage needs to be performed before

each execution, and thus SEQC’s compilation time is largely unaffected by the growth of qubit counts in future quantum processors.

- We design and implement in SEQC several novel methods for qubit placement, qubit routing, and circuit optimization. In particular, we classify swaps into symbiotic, commensalistic, and parasitic, based on how swap placement affects inter-module gates, and develop algorithms to promote symbiotic and avoid parasitic swap placement; we develop qubit “pinning” techniques to allow elaboration without disrupting module connectivity; and we adapt Simulated Annealing and Spatio-Temporal Aware Qubit Allocation (STA) techniques to aid in circuit stratification.
- We evaluate SEQC on a modular quantum system consisting of small-scale (10-qubit heavy-hex) modules, and find that it compiles circuits with 9.3% higher circuit fidelity on average (up to 49.99%), while consistently achieving faster compilation times ($3.27\times$ on average, up to $6.74\times$) compared to a module-unaware Qiskit baseline. When evaluated on a system with large-scale (120-qubit grid topology) modules, SEQC achieves 23.3% higher fidelity on average (63.36% max), while compiling $1.34\times$ faster on average (up to $3.37\times$). More importantly, SEQC outperforms Qiskit even when the latter is afforded significant allowances (no universal gate restrictions, and much lower inter-module gate errors).

CHAPTER 2

Background and Motivation

Compilation is often divided into distinct qubit allocation (layout and routing), basis translation, and optimization stages.

2.0.0.1. Qubit Allocation. Qubit allocation addresses the topology constraints of quantum devices, where physical qubits on a device have limited connectivity to other qubits on the device. It is critical that after this stage, which often requires the modification or addition of circuit gates, the resulting optimized, technology-dependent circuit is functionally equivalent to the original, technology-independent algorithm [61]. As qubit allocation is NP-complete [60], it is typically divided into two substages, *layout* and *routing*, to make it more tractable.

Logical-to-physical qubit *layout*, also known as qubit placement, involves deriving an injective map from the virtual qubits of the quantum circuit to the physical qubits of the quantum device [27, 35], cognizant of its limited connectivity. In other words, the qubit layout stage decides the initial position of logical qubits on the device.

Qubit *routing* operates from an initial qubit layout and inserts SWAP operations to align a quantum circuit's gates to a given qubit topology. Consider, for example, the state-of-the-art in qubit allocation: the SABRE heuristic search algorithm [35]. In SABRE, the distinction of the layout and routing stages motivates an iterative method for solving qubit allocation: an initial layout is used to perform qubit routing, which, in turn, is used to generate a new initial layout. Because qubit routing remains NP-hard [27], the problem is

often simplified by assuming qubit links are roughly equivalent, particularly with respect to average fidelity and two-qubit gate duration. This assumption can manifest implicitly when a given device topology is represented as an unweighted, undirected graph. For monolithic architectures, one could debate the marginal tradeoff between performance and solution quality. However, for modular architectures, the cross-module connections are expected to be far worse compared to intra-module connections [62, 44, 77, 29, 75]. Thus, as we will show, the current simplified model applied on modular architectures leads to worse solutions than a more physically accurate one.

2.0.0.2. Basis Translation. Basis translation converts the gates of an input quantum circuit into gates that are physically supported on hardware. Basis translation can be performed efficiently given the target gate set is universal [46], and the quantum circuit contains a fixed, finite number of qubits [17]. Unfortunately, modular quantum architectures depend upon specialized links to connect otherwise-independent modules. In the general case, the cross-module links can only be used for data movement (such as SWAPs) and not computation, meaning these links may not support universal operations [47]. Thus, compilers that depend on the expectation of universality among qubit links [66], and do not expect these *heterogenous, non-universal* basis gate sets, can generate incompatible outputs when targeting modular architectures.

2.0.0.3. Optimization. Quantum circuit optimization seeks to prune extraneous operations by finding opportunities to combine, eliminate, and parallelize quantum gates [43]. Optimization is highly time and resource intensive, with the problem scaling exponentially to the number of qubits and circuit depth [32]. As a result, partitioning, the division of a quantum circuit into *blocks* of quantum gates, is a common tactic among optimization

techniques [32, 72, 74]. One major consideration in this process is demarcation, i.e., where and when to draw the line between blocks. Namely, one forfeits all optimization opportunities along boundaries of demarcation, so they must be carefully chosen to minimize the negative impact on the output solution. Modular architectures are naturally synergistic with partitioning, since cross-module connections present a physically motivated boundary between operations.

CHAPTER 3

Compilation for Hybrid Modular Quantum Systems

While SEQC can be applied on a modular quantum system comprising hybrid QPUs, in this paper we consider the simpler case of homogeneous modules as a starting point. The reasons behind this decision are simple. First, the primary goal of this paper is to introduce and evaluate the impact of the circuit stratification compilation stage with STA and Simulated Annealing we developed, along with qubit pinning, and symbiotic swap insertion. The presence of diverse-modality QPUs with varying capabilities would dilute these results, and make it hard to separate their effect from the effects of heterogeneity. At the same time, it would make comparisons to the existing state-of-the-art hard, as, to the best of our knowledge, there are no compilers that can target multiple modalities simultaneously for different parts of the same input circuit. Thus, in order to isolate the impact of our techniques, and facilitate comparisons with existing tools, in this paper we describe and evaluate SEQC assuming homogeneous modules of the same technology. When extending SEQC to a hybrid system, one only needs to execute different elaboration stages, one for each QPU modality, but the stratification stage and the other techniques we develop will remain largely unaffected. Thus, for the remainder of this paper, we will be focusing on homogeneous QPUs based on transmons, and we leave qubit-specific stratification optimizations, and the implementation of elaboration stages for other qubit species for future work.

Modular quantum architectures introduce new challenges to quantum compilation that impact the correctness of solutions and the speed of compilation. As discussed previously in §2, our model of modular quantum architectures considers the possibility that inter-module links can be *non-universal*. This is representative of hardware manufacturing constraints [47], where only a restricted set of operations, e.g., state transfers or SWAPs, may be supported over long-distance couplers (inter-module links). However, many compiler designs (including designs for modular architectures, see §7) experience errors or crashes when presented with non-universal, heterogeneous, basis gate sets. For example, the Qiskit compiler [28] is explicit in its compatibility for only universal basis gate sets [66]. While it is possible to create a Qiskit `Backend` that supports heterogeneous basis gate sets, the compiler itself does not take them into account. Specifically, the Qiskit routing stage [78] (an implementation of SABRE) will freely place gates on incompatible physical links, and there are no methods in the compiler to prevent or fix consequent errors. Moreover, compilers are called to target increasingly larger quantum devices, enabled by modularity. To address the mounting pressure of ever-growing compilation workloads, these compilers must be cognizant of classical hardware limitations and tradeoffs between solution quality and compilation latency.

To ameliorate these issues, we present a family of compilers optimized for modular quantum architectures that support module links with non-universal basis gate sets, and leverage modular compilation strategies. In Section 3.1, we propose a peephole compilation pass that can be inserted into *any* compiler to add support for non-universal basis gate sets. This would allow any compiler to be compatible with module-based devices without significant changes to their underlying implementations. Next, in Section 3.2, we

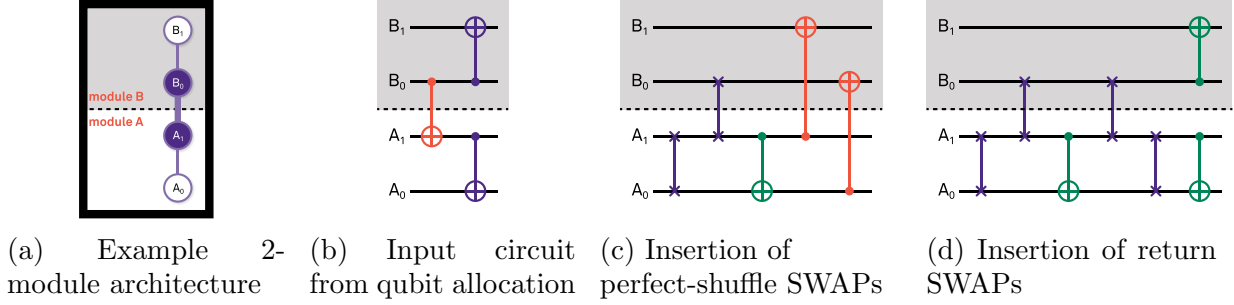


Figure 3.1. Peephole correction for inter-module gates that are unimplementable on hardware (i.e., non-SWAP, shown in red).

introduce the stratify-elaborate quantum compiler (SEQC), which advocates for a hierarchical two-stage compilation that produces both better circuits for execution and faster compilation times.

3.1. Module-awareness via Peephole Correction

The design of the peephole correction pass was inspired by the “peephole” optimizations commonly used in classical compilation [3]. Peephole passes operate within a nominal peephole window and substitute its contents with a logically equivalent replacement. Peephole correction (Figure 3.1) operates on a circuit generated by qubit allocation, where it detects when non-SWAP gates are placed on non-universal inter-module links (Figure 3.1b), and employs a cyclic qubit rotation, i.e., a perfect shuffle permutation, to move the aforementioned gates to nearby intra-module links (Figure 3.1c). Additional “return” SWAPs are then inserted to return the qubits to their initial locations (Figure 3.1d). Overall, for every inter-module gate in the original circuit generated by SABRE, the peephole pass inserts 4 additional SWAP gates — 2 inter-module and 2 intra-module. Ultimately, this degree of overhead is inevitable when the stock, unmodified compiler is unaware of module-specific device constraints.

For example, skipping the return SWAP operations would appear to save 2 SWAPs, and halve the cost per inter-module gate. However, it would also introduce a net qubit permutation that propagates downstream in the circuit. From the perspective of modular compilation, this creates serial dependencies that prevent parallel execution of the pass. From a routing perspective, the resultant qubit permutations disrupt the post-routing layout of qubits, causing destructive routing errors that can only be resolved by an entirely new routing pass. Overall, such a hypothetical pass would result in substantially worse output circuits compared to the peephole method.

3.2. The Stratify-Elaborate Quantum Compiler

We incorporate the hierarchical elements of module architectures into SEQC by adopting a two-stage procedure (Figure 3.2). The first, **stratification stage**, represents a one-time overhead to elevate a quantum circuit to the module level. The second, **elaboration stage**, represents a recurring cost to complete the compilation of a quantum circuit given the most up-to-date backend properties.

3.2.1. Stratification Stage

In this stage, we map a quantum circuit to the inter-module device topology (Figure 3.3a in our running example) by stratifying it into smaller, interconnected *subcircuits*. This can be

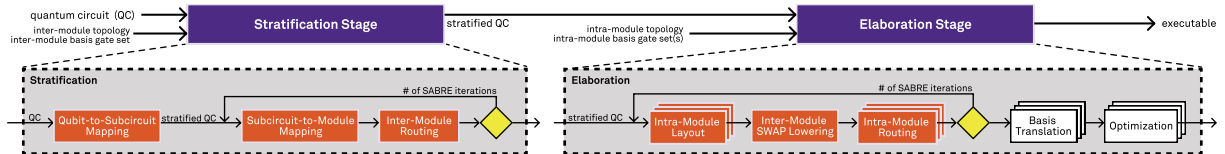


Figure 3.2. High-level diagram of SEQC. Elements in red reflect modifications from existing compiler designs.

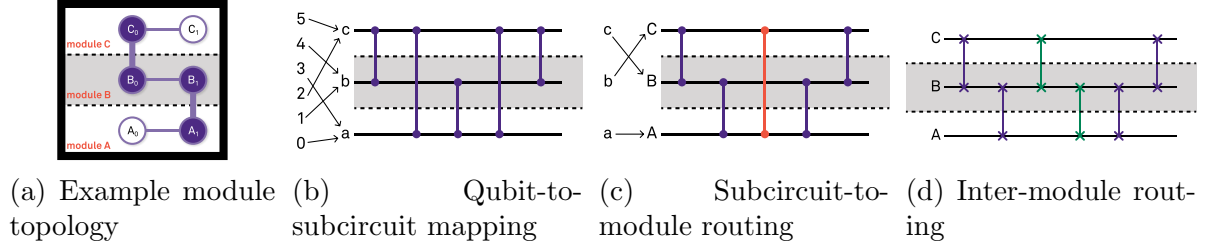


Figure 3.3. Step-by-step example of inter-module compilation in the Stratification stage.

accomplished with the following intermediate steps (Figures 3.3b–3.3d). First, **qubit-to-subcircuit mapping** designates which logical qubits belong to which logical subcircuits. Then, as part of **module allocation**, we define a mapping of logical subcircuits to physical modules on the target device and route inter-module gates to conform with constraints of module connectivity.

3.2.1.1. Qubit-to-Subcircuit Mapping. Qubit-to-subcircuit mapping produces a *stratified* quantum circuit, where logical qubits (numbers in Figure 3.3b) are grouped into subcircuits (lowercase letters). An optimal qubit-to-subcircuit mapping minimizes the number of connections between subcircuits, thereby reducing the number of inter-module SWAPs needed to transfer data between modules during execution. To perform this task, we investigate two methods: Simulated Annealing and Spatio-Temporal Aware Qubit Allocation (STA) [48].

Simulated annealing is an efficient stochastic optimization technique shown to work well on NP-complete problems, such as place-and-route in VLSI algorithms [58]. It is an iterative method to optimize a state against a given cost function. In each iteration, the state is “kicked,” or randomly perturbed, and the new state is greedily accepted if there is a reduction in cost. However, the algorithm may randomly also accept worse solutions

to escape from local minima wells. The runtime of simulated annealing is statically determined using constant tuning parameters, thus the overall complexity is determined by the complexity of the cost and kick functions, which are typically polynomial. Additionally, since each run of a simulated annealing is randomized and independent, it is possible to run multiple trials in parallel with different initial random seeds, then pick the best result. This improves the chances of finding a (near-)optimal solution. Our simulated annealing method (Algorithm 1) starts with an initial qubit-to-subcircuit mapping, which we minimize against a cost function, defined as the number of inter-module gates given a certain layout. By minimizing the number of inter-module gates, we implicitly reduce the inter-module connectivity. The kick function then swaps random qubits in the layout, ensuring that each SWAP is between qubits in different subcircuits. In doing so, we hope to move strongly interacting qubits close together, minimizing the number of cross-module gates. As input, our algorithm takes the input circuit $Circ$, the module count N_C , the module size N_Q , the starting temperature T_0 , the number of changes to do per degree of temperature α , and the cooling factor β . The last three are constant tuning parameters across runs.

The STA algorithm is a qubit allocation algorithm designed for trapped-ion quantum systems. These devices often feature modular designs, where one device is composed of “traps” arranged in a (typically) line or ring topology, where each trap contains multiple qubits (ions). Qubit allocation algorithms for modular trapped-ion systems are incentivized to minimize the number of operations between modules due to associated qubit shuttling and relocation overheads [48]. When compiling for modular transmon architectures, we are similarly incentivized to minimize the number of inter-module interactions.

Algorithm 1 Qubit-to-Subcircuit Mapping with Simulated Annealing

```

1: procedure COST(Circ, L)
2:    $C \leftarrow 0$  ▷ Count
3:   for  $g \in \text{TwoQubitGates}(\textit{Circ})$  do
4:      $q_0, q_1 \leftarrow \text{Qubits}(g)$ 
5:      $s_0, s_1 \leftarrow L[q_0], L[q_1]$ 
6:     Increment  $C$  if  $s_0 \neq s_1$ 
7:   return  $C$ 
8: procedure KICK( $D, N_C, N_Q, L$ )
9:    $L' \leftarrow \text{COPY}(L)$ 
10:  for  $i = 1, \dots, D$  do
11:     $s_0 \leftarrow \text{RANDCHOICE}(\{1, \dots, N_C\})$ 
12:     $s_1 \leftarrow \text{RANDCHOICE}(\{1, \dots, N_C\} - \{s_0\})$ 
13:     $q_0, q_1 \leftarrow \text{RANDCHOICE}(L'[s_0]), \text{RANDCHOICE}(L'[s_1])$ 
14:     $L'[s_0], L'[s_1] \leftarrow (L'[s_0] \cup \{q_1\}) - \{q_0\}, (L'[s_1] \cup \{q_0\}) - \{q_1\}$  ▷ Swap  $q_0$  and  $q_1$ 
15:  return  $L'$ 
16: procedure Q2SMAPPINGAN(Circ,  $N_C, N_Q, T_0, \alpha, \beta$ )
17:    $L \leftarrow \text{Initial Layout}$  ▷ Current Solution
18:    $R \leftarrow L$  ▷ Best Solution
19:    $T \leftarrow T_0$  ▷ Annealing "Temperature"
20:   while  $T > 1$  do
21:      $D \leftarrow \text{CEIL}(T \times \alpha)$  ▷ Number of changes to make
22:      $L' \leftarrow \text{KICK}(D, N_C, N_Q, L)$ 
23:      $\Delta \leftarrow \text{COST}(\textit{Circ}, L') - \text{COST}(\textit{Circ}, L)$ 
24:      $k \leftarrow \text{RAND}(0, 1)$  ▷ Random number in  $[0, 1) \subset \mathbb{R}$ 
25:      $L \leftarrow L'$  if  $(\Delta < 0)$  or  $(k > \exp(\Delta/T))$ 
26:      $R \leftarrow L'$  if  $(\text{COST}(\textit{Circ}, L') < \text{COST}(\textit{Circ}, R))$ 
27:      $T \leftarrow T \times \beta$  ▷ Decrease temperature
28:   return  $R$ 

```

Thus, we adapted a component of the STA algorithm (Algorithm 2 in Ovide *et al.* [48]) that assigns qubits to ion traps to perform qubit-to-subcircuit mapping and generate an initial subcircuit-to-module mapping (Algorithm 2).

The original STA algorithm prioritizes mapping qubits that frequently interact with other qubits (quantified by R in Algorithm 2), or have a large degree of direct or indirect interactions (quantified heuristically as the depth-weighted sum T of interactions between every pair of qubits). These are used to derive a sequence of qubit pairs, from which STA

Algorithm 2 Qubit-to-Subcircuit Mapping with STA

```

1: procedure COMPUTERATIOS( $C$ )
2:    $R \leftarrow$  empty map
3:   for  $q \in \text{QUBITS}(C)$  do
4:      $R[q] \leftarrow \frac{\text{INTERACTIONCNT}(q,C)}{\text{NUMQUBITS}(C)}$   $\triangleright$  fraction of qubits  $q$  interacts with
5:   return  $R$ 
6: procedure COMPUTEST( $C, S$ )
7:    $T \leftarrow$  empty map
8:   for  $(q_1, q_2) \in \text{ALLPAIRS}(\text{QUBITS}(C))$  do
9:     for  $(i, s) \in \text{ENUMERATE}(S)$  do
10:       $\text{Add } 2^{-i} \text{ to } T[(q_1, q_2)]$  if  $q_1$  and  $q_2$  interact in  $s$ 
11:   return  $T$ 
12: procedure MAPQUBIT( $q_1, T, R, L$ )
13:    $P \leftarrow$  first pair in  $T$  containing  $q_1$ ;  $q_2 \leftarrow$  other qubit in  $P$ 
14:   if  $\text{INDEX}(q_2) < \text{INDEX}(q_1)$  then
15:      $\text{MAPQUBIT}(q_2, T)$   $\triangleright$  Map  $q_2$  if it comes first
16:   if  $q_1 \in R$  or  $q_2 \in R$  then
17:      $\text{PLACEQUBITS}(q_1, q_2, L)$ 
18:      $R.\text{REMOVE}(q_1, q_2)$ ;  $T.\text{REMOVE}(P)$   $\triangleright$  Remove placed qubits
19: procedure Q2SMAPPINGSTA( $C$ )  $\triangleright C$ : Quantum Circuit
20:    $S \leftarrow \text{LAYERS}(C)$ ;  $L \leftarrow$  initial layout
21:    $R \leftarrow \text{COMPUTERATIOS}(C)$ ;  $T \leftarrow \text{COMPUTEST}(S)$ 
22:   while  $R$  is not empty do
23:      $\text{MAPQUBIT}(R[0], T, R, L)$ 
24:   return  $L$ 

```

constructs a qubit mapping. Namely, for each iteration of the STA algorithm, a pair of qubits (q_a, q_b) is used to insert at least one qubit into the mapping, based on the shortest topological distance between physical qubits. If one of the qubits in the pair is already mapped, say q_a , then STA also recursively maps any qubits that are more strongly connected to q_a than q_b is. This is done in order to keep strongly connected qubits physically close together.

3.2.1.2. Module Allocation. Module allocation describes the process of transforming a stratified circuit to be compatible with the inter-module topology of a modular device. In Figure [3.2](#), we decompose module allocation into subcircuit-to-module mapping and

Algorithm 3 Inter-Module Routing

```

1: procedure FINDSWAPS( $Q, E$ )
2:   return FINDSYMBIOTICSWAPS( $Q, E$ ) if non-empty
3:   return FINDCOMMENSALISTICSWAPS( $Q, E$ ) if non-empty
4:   return FINDPARASITICSWAPS( $Q, E$ )
5: procedure VALIDGATES( $Q, L$ )
6:   Initialize  $V_{virt}$  and  $V_{phys}$  to empty sets
7:   for  $g \in Q$  do
8:      $H \leftarrow \{L[q] | q \in \text{QUBITS}(g)\}$   $\triangleright$  Get module info
9:     if  $|H| = 1$  then  $\triangleright$  All qubits on same module
10:     $\quad$  Add  $g$  to  $V_{virt}$  and  $(g, H)$  to  $V_{phys}$ 
11:   return  $V_{virt}, V_{phys}$ 
12: procedure STEP( $Q, R, L, D$ )  $\triangleright$  Advances the “wavefront”
13:    $V_{virt}, V_{phys} \leftarrow \text{VALIDGATES}(Q, L)$ 
14:    $R \leftarrow R + V_{phys}$ 
15:   for  $g \in V_{virt}$  do
16:     Remove  $g$  from  $Q$ 
17:     for  $c \in \text{CHILDREN}(g, D)$  do
18:       Add  $c$  to  $Q$  if  $p \in R$  for all  $p \in \text{PARENTS}(c, D)$ 
19: procedure HEURISTIC( $S, R, Circ$ )
20:    $L' \leftarrow \text{SWAP}(\text{COPY}(L), S)$ ;  $C \leftarrow 0$ 
21:   for  $g \in \text{TWOQUBITGATES}(Circ)$  do
22:     if  $g \notin R$  then
23:        $q_0, q_1 \leftarrow \text{QUBITS}(g)$ 
24:        $s_0, s_1 \leftarrow L[q_0], L[q_1]$ 
25:       Increment  $C$  if  $s_0 \neq s_1$ 
26:   return  $C$ 
27: procedure INTERMODULEROUTING( $Circ, L, T$ )  $\triangleright L$  is from Algorithm 2,  $T$  is the module topology.
28:    $D \leftarrow \text{CIRCUITToDAG}(Circ)$ 
29:    $E \leftarrow$  All-pairs distances from module topology  $T$ 
30:    $Q \leftarrow \text{FIRSTLAYER}(DAG)$ ;  $R \leftarrow$  empty list
31:   STEP( $Q, R, L, D$ )
32:   while  $Q$  is not empty do
33:      $S \leftarrow \text{FINDSWAPS}(Q, E)$   $\triangleright$  SWAP Candidates
34:      $S \leftarrow \arg \min_{s \in S} \text{HEURISTIC}(s, R, Circ)$ 
35:     SWAP( $L, S$ )
36:     STEP( $Q, R, L, D$ )
37:    $D' \leftarrow$  DAG built in topological order from  $R$   $\triangleright R$  is ordered
38:   return DAGToCIRCUIT( $D'$ )

```

inter-module routing, analogous to how qubit allocation can be decomposed into qubit layout and routing. Accordingly, subcircuit-to-module mapping (Figure 3.3c), defines an initial placement of logical subcircuits (lowercase letters) to physical modules (uppercase letters). Subsequently, inter-module routing (Figure 3.3d) inserts inter-module SWAPs to account for limited cross-module gate connectivity.

In our design, we use our qubit-to-subcircuit mapping to provide an initial subcircuit-to-module mapping. Then, we extend and modify the SABRE algorithm [35] to apply to modular architectures, placing iterations of inter-module routing alongside randomized qubit permutations to inform a final subcircuit-to-module mapping. We did explore a design that extended other components of STA to perform subcircuit-to-module mapping on transmon module systems, but it underperformed this iterative approach, and had increased compilation time, hence it was abandoned.

Inter-module routing seeks to consolidate the qubits of (non-SWAP) inter-module gates to a single module by inserting inter-module SWAPs into the stratified circuit. For a single two-qubit, inter-module gate, we represent inter-module routing as a shortest-path problem, where the two qubits of the gate correspond to module endpoints in the unweighted inter-module connectivity graph. From the shortest path, we can derive an optimal sequence of SWAPs for routing, where the unweighted path length, or module distance, indicates the length of the SWAP chain. However, in the context of a stratified circuit, composed of multiple inter-module gates, it becomes possible to interleave SWAP chains (i.e., efficiently reorder their execution) to reduce the depth and/or total number of inter-module SWAPs. Like qubit routing, we would expect it to be impractical to derive

an optimal solution for this more complex picture of inter-module routing at compile-time. Thus, we employ a similar SWAP selection scheme as SABRE [35], where inserted SWAPs are chosen from a pool of SWAP candidates according to a cost heuristic. For our inter-module routing algorithm, we categorize SWAPs from our candidate pool into three tiers, based on a concept of the “efficiency” of each SWAP, as shown in Algorithm 3.

An inter-module SWAP is considered **symbiotic** if it directly affects two inter-module gates in a “beneficial” manner. In other words, each qubit of a symbiotic SWAP corresponds to a different inter-module gate, where the insertion of the SWAP reduces the module distance score of both gates. In theory, this outcome represents the most efficient possible insertion for a single SWAP, and thus should be the highest-priority case for our greedy inter-module routing cost heuristic. Next, **commensalistic** SWAPs are those that do not “harm,” or increase that module distance, of any directly impacted gates. These SWAPs typically occur when an idle qubit can be used as an ancilla qubit to facilitate the routing of an inter-module gate without affecting other inter-module gates. In certain device topologies, a SWAP acting on the qubits of two inter-module gates can also exhibit commensalistic behavior. Finally, **parasitic** SWAPs act on the qubits of two inter-module gates, where one gate benefits while the other is harmed. Since this variety of SWAPs is actively counterproductive towards a gate, it should be, and likely will be, avoided in most circumstances. However, given a particularly onerous circuit, it may be necessary in such rare instances to break out of a deadlock. For example, clearing one gate may free physical qubits to help route others.

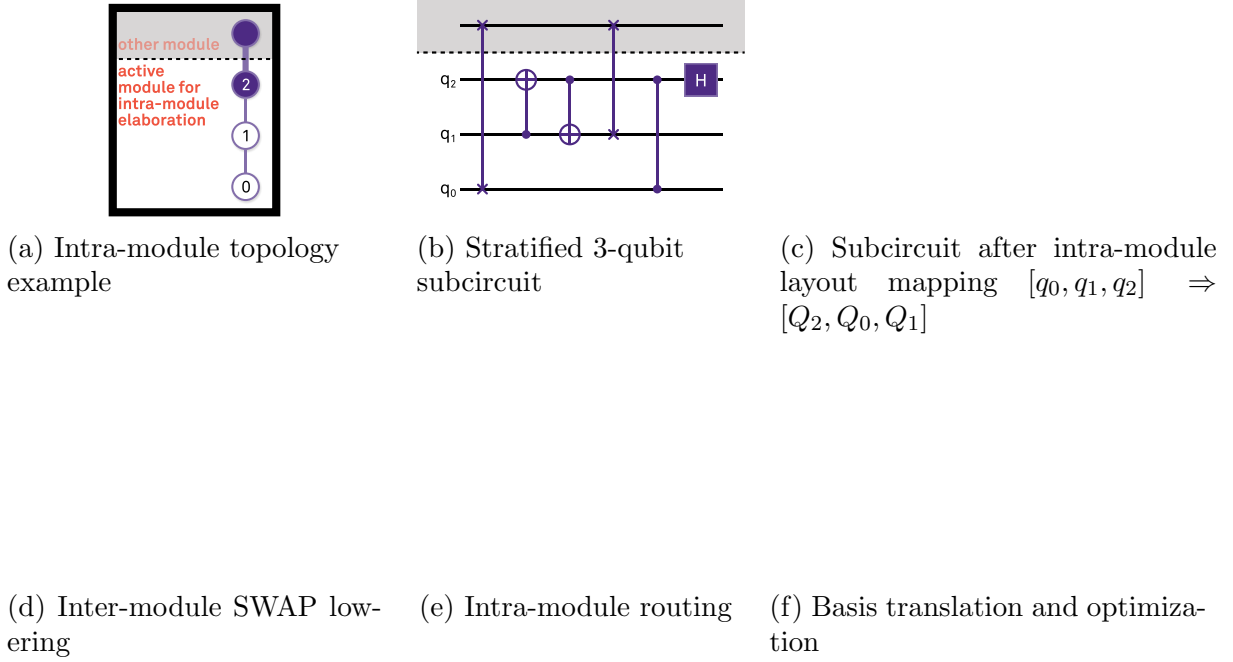


Figure 3.4. Step-by-step example of intra-module compilation in the Elaboration stage. Gates that cannot be implemented on hardware are shown in red.

3.2.2. Elaboration Stage

Following the stratification stage, each module can be compiled almost completely independently from one another. More precisely, the compilation process within each module is largely identical to that of a monolithic quantum architecture. The only exception is the presence of inter-module SWAPs, which must be lowered from the module level to the qubit level. Once this assignment is made, these SWAPs must remain fixed, which we call “**qubit pinning**” and immutable to subsequent compilation stages. Despite this overhead, restricting the task of compilation to a narrower domain of qubits facilitates both parallelization (since each subcircuit can be processed in parallel) and problem size reduction (since there are fewer qubits to consider) of the most computationally intensive compilation stages, namely routing and optimization. To accomplish these tasks, we

divide the elaboration stage into two substages: **qubit allocation** and **parallel basis translation and optimization**. Figure 3.4 illustrates this with an example of elaboration for a single module (Figure 3.4a) and its corresponding subcircuit (Figure 3.4b).

3.2.2.1. Qubit Allocation. We employ a modified version of SABRE to perform qubit allocation in the elaboration stage. Specifically, we parallelize the SABRE routing algorithm and add constraints to maintain correctness when applied to a stratified quantum circuit with inter-module gates. We also explored a layout pass that used portions of the STA algorithm, but the marginal benefits to solution quality were not worth the $5 - 6\times$ slower elaboration time.

The placement of inter-module gates is a non-parallelizable task (due to cross-module interactions) that strongly influences the layout and routing passes, which could otherwise be fully parallelized across modules. To account for this, we introduce a serial stage (Figure 3.4d) between the parallel layout and routing passes that greedily places gates on the nearest valid inter-module edge. We choose a greedy policy because inter-module gates have more restrictions on valid placements compared to intra-module gates, while also experiencing substantially higher error rates [62]. Once the positions of the inter-module gates are settled, they are *pinned*, imposing an additional constraint on qubit routing. To softly encourage this behavior in SABRE, we extend its cost heuristic to weigh the device topology by fidelity. This causes higher error links to assume higher costs, discouraging SABRE from moving the high-error inter-module gates. As a result, SEQC achieves better solutions in the common case. However, there is nothing to prevent SABRE from incorrectly modifying any inter-module SWAPs. Thus, we need to impose hard restrictions to ensure correctness. To address this we establish two disjoint gate

sets: one gate set that operates strictly within a module (the traditional gate set for a monolithic processor), and another that is the only gate set permitted to operate across modules (and, specifically for our example implementation, only implements inter-module SWAPs). By restricting each gate set to its own disjoint part of the overall device topology, we ensure correctness across all cases.

3.2.2.2. Parallel Translation and Optimization. The remaining compilation stages of basis translation and circuit optimization can be easily parallelized at the module granularity (Figure [3.4f](#)). Basis translation, in particular, is embarrassingly parallel: gates can be translated independently from each other. Meanwhile, as discussed in [§2](#), optimization algorithms already employ partitioning and parallelization to create tractable subproblems from intractably large circuits, facilitating a smaller problem space, lower memory usage, and faster performance. In modular quantum architectures, we physically motivate partitions through the natural module boundaries, enabling us to reap the same resource and performance benefits of partitioning without compromising on solution quality.

CHAPTER 4

Methodology

We evaluate quantum compilers based on the quality of their solutions, their speed of compilation, and their ability to scale to large circuits/devices. To experimentally profile these characteristics, we benchmark a compiler on a suite of quantum circuits, detailed in §4.1, targeting a mock module-based quantum backend, as specified in §4.2. In total, we evaluate four compilers—a reference compiler, a peephole-augmented compiler, and two versions of SEQC—on simulated backends ranging from 1 to 100 modules. All compilers are implemented in the Qiskit SDK [28] v1.2.4 and Python 3.12.6, and timing results are taken on a machine with a 128-core Ampere Altra Max ARM64 processor and 256 GB of DDR4 RAM. For our reference compiler, we use the one built into Qiskit, parameterized with `optimization_level=3`. Because Qiskit is not natively module-aware, we run these experiments on a backend with *universal* inter-module links (§4.2). To construct the peephole compiler we augment the reference Qiskit compiler by inserting our peephole pass between its `routing` and `basis_translation` stages, thus making it module-aware. Our two SEQC versions differ in the technique they use for qubit-to-subcircuit mapping. Our first SEQC implementation leverages simulated annealing, initialized with a starting temperature $T_0 = 200$, a rate of change of $0.005T$ per round, $\frac{T}{50}$ permutations per round, and a total of 10 trials per CPU core. Meanwhile, our second SEQC implementation leverages STA for subcircuit-to-qubit mapping and subcircuit-to-module mapping.

We aggregate results in a manner similar to SPEC benchmarks [7], by taking the geometric mean of the relative performance ratio against a baseline technique across a benchmark suite of quantum circuits. The resultant score is then derived for each of the evaluated metrics on multiple backends of varying numbers of modules.

4.1. Benchmark Suite

We leverage circuits from Supermarq [65] that are representative of practical, real-world applications for quantum computing, while also being feasible to compile for a large number of qubits. In particular, our suite comprises the following quantum circuits.

Bit Code. Bit codes are often used to detect and correct for bit-flip errors in quantum error correction applications. For an n -qubit benchmark, $k = \lfloor \frac{n+1}{2} \rfloor$ data qubits are prepared in an initial state of $|1010\dots10\rangle$ and fed through two rounds of error correction. The structure of the bit code circuit (Figure 4.1a) invites many opportunities for gate-level parallelism during compilation. However, because each logical qubit is connected with its two neighbors with CNOTs, a compiler with an ill-suited topology may struggle to exploit that parallelism due to the number of necessary inserted SWAPs.

Phase Code. Like the bit code, phase codes are often used to detect and correct for phase-flip errors in quantum error correction applications. For an n -qubit benchmark, $k = \lfloor \frac{n+1}{2} \rfloor$ data qubits are prepared in an initial state of $|1010\dots10\rangle$ and fed through two rounds of error correction. To the compiler, the main difference compared to the bit code circuit is the presence of additional single-qubit gates (Figure 4.1b). This may showcase a compiler’s ability to do noise-aware mapping by differentiating between the relative cost (duration, fidelity) between single- and two-qubit gates.

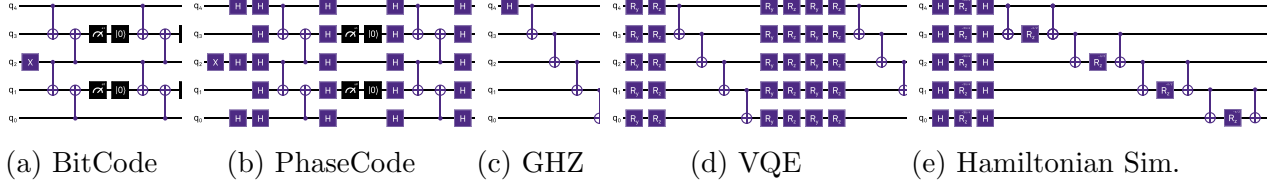


Figure 4.1. Examples of benchmark applications for 5-qubit circuits.

Greenberger–Horne–Zeiling (GHZ). Entanglement is one of the fundamental properties of quantum states and a major source of quantum advantage. The GHZ state represents a maximally entangled n -qubit state of $\frac{1}{\sqrt{2}} (|0\rangle^{\otimes n} + |1\rangle^{\otimes n})$. The algorithmic circuit structure is essentially a single chain of CNOT gates (Figure 4.1c). From a compilation perspective, this represents an exercise in qubit allocation, because the logical circuit is entirely serial. An n -qubit GHZ circuit that fully utilizes a k -module device should necessitate $\lfloor \frac{n}{k} \rfloor - 1$ CNOT operations to extend between modules. In turn, this will necessitate at least $2 (\lfloor \frac{n}{k} \rfloor - 1)$ inter-module SWAPs to transfer qubits in/out of a module.

VQE. The variational quantum eigensolver (VQE) [52] is another hybrid quantum-classical variational algorithm that is often used in chemistry applications. In our experiments, we choose a 2-layer ansatz whose parameters are again randomly generated (Figure 4.1d). The resultant circuit structure features two CNOT chains separated by single-qubit operations. The CNOT chains can be interleaved, which in conjunction with the single-qubit gates, creates many opportunities for parallelism (resembling pipelining in classical computing). Moreover, from a qubit routing perspective, the second CNOT chain forces the compiler to consider how qubit permutations in one circuit layer propagate to future layers.

Hamiltonian Simulation. Quantum devices are naturally well-suited to simulating Hamiltonians. The **Supermarq** benchmark selects the 1D Transverse Field Ising Model

(TFIM) system to model, which is relevant for phase transitions in magnetic materials. The circuit for Hamiltonian Simulation (Figure 4.1e) resembles a slightly more complex GHZ circuit, including how it is completely serial. Compared to GHZ, this circuit should test the compiler’s ability to lookahead during routing to detect back-to-back gates on the same two-qubit pair, while also emphasizing parallelism originating from the efficient placement of SWAP gates.

4.2. Simulated Quantum Device Specifications

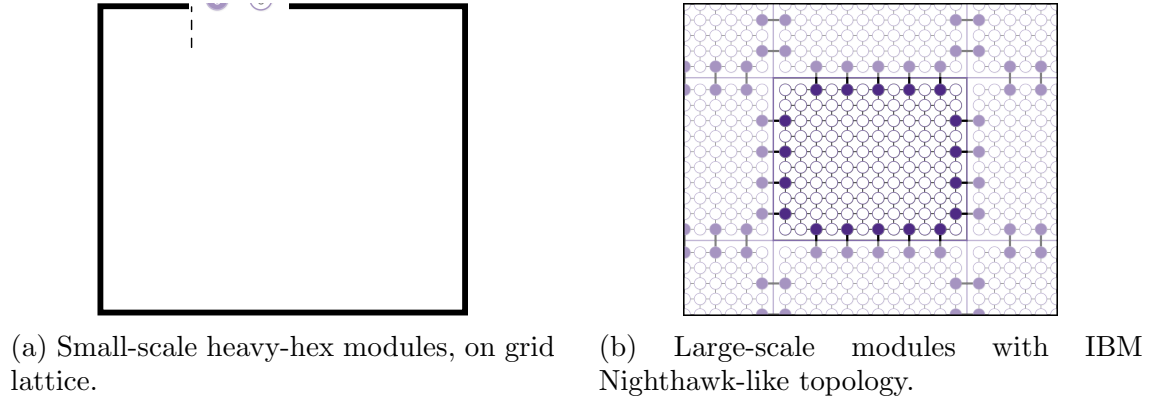


Figure 4.2. Modeled topologies.

We model two types of simulated devices differing in their inter-module topology and module size. To isolate the impact of architecture modularity, we evaluated on symmetrical modules based on a transmon qubit technology. One variant, denoted the **heavy-hex backend** (Figure 4.2a), is generated to conform to the modular architecture proposed in Smith *et al.* [62], and features a 10-qubit, heavy-hexagon topology [9]. The other, denoted the **grid backend** (Figure 4.2b), uses modules with a 120-qubit grid topology, based on the upcoming IBM Nighthawk processor [41, 26]. Both simulated devices use a grid lattice topology of symmetric modules. The hardware specifications of

(a) Qubit Properties		(b) Instruction Properties		
		Instruction	Duration (ns)	Error
T1	[2]	X [2]	25.0	0.109%
T2	[2]	SX [2]	25.0	0.109%
Frequency	[2]	$R_z(\phi)$ [2]	0.0	0.000%
		CZ (intra-module) [2]	34.0	0.605%
		CZ (inter-module) [†]	126.0	2.420%
		SWAP (inter-module) [‡]	702.4	10.23%
		Reset [2]	500.0	0.186%
		Measure [2]	500.0	0.196%

[†]Universal gate set only. [‡]Non-universal gate set only.

Table 4.1. Specifications of Simulated Modular Backend.

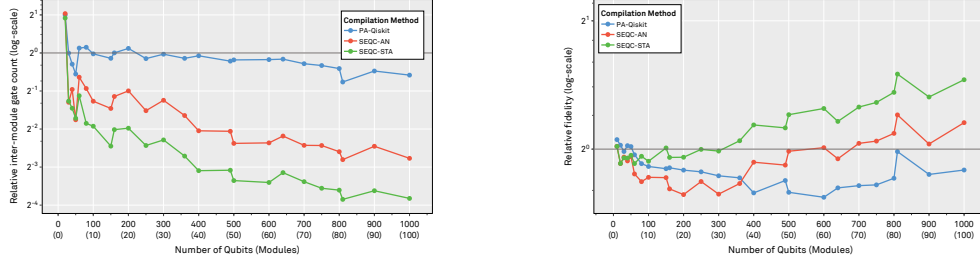
the qubits and intra-module connections (detailed in Table 4.1) are sourced from Acharya *et al.* [2]. To generate the expected fidelity of the inter-module SWAPs, we perform a simulated random benchmarking trial and calculate the error rate of a SWAP gate with these backend specifications. Following Smith *et al.* [62], we conservatively model *inter*-module SWAPs with $4\times$ the error rate and $4\times$ the duration of an *intra*-module SWAP. Finally, note that for each valid module count, there may exist multiple valid module topologies. In this work, we generate our backends to use the “most-square” topology for a given module count. For example, a 12-module machine would be a 3×4 grid.

CHAPTER 5

Experimental Results

We evaluate the Qiskit reference compiler, the peephole-augmented Qiskit (PA-Qiskit) compiler, SEQC using simulated annealing (SEQC-AN), and SEQC using STA (SEQC-STA) on metrics relating to the quality of the compiled circuits, on both the 10-qubit heavy-hex backend, and the 120-qubit grid backend. The Qiskit-based and SEQC compilers prioritize different heuristics in compilation. We examine how well each compiler is able to minimize its preferred heuristics, and how well that translates to practical solution quality. To demonstrate scalability, we analyze compilation time, specifically the recurring and non-recurring compilation times for repeating compilations of a single application, as well as memory requirements for the compilation. Relative metrics are normalized over the Qiskit reference compiler.

5.0.0.1. Number of inter-module gates. The primary heuristic we minimize in SEQC is the number of inter-module gates in a quantum circuit. This is because we expect that the inter-module gates of a modular architecture contribute the most to error and execution time relative to other gates (at least by a factor of $4\times$ [62]). As a result, the number of inter-module gates should represent a strong proxy metric for the quality of a compiler solution, and thus minimizing the number of inter-module gates should result in higher-fidelity, faster-executing quantum circuits. Accordingly, SEQC consistently produces results that have significantly fewer inter-module gates compared to Qiskit-based compilers (Figures 5.1a and 5.2a). For example, compilation with SEQC-STA results in 82.43%



(a) Relative inter-module gate count (lower is better).

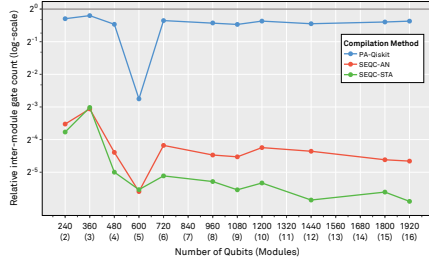
(b) Relative estimated fidelity (ESP) (higher is better).

Figure 5.1. Main circuit performance metrics: inter-module gate count and estimated fidelity, 10-qubit heavy-hex backend.

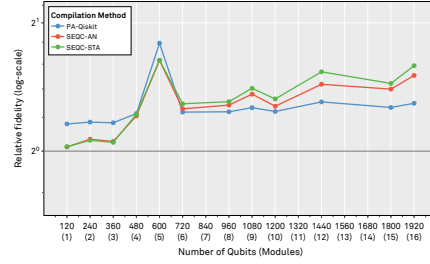
fewer inter-module gates on average compared to Qiskit (up to 92.94%). on the heavy-hex backend. Meanwhile, compilation with SEQC-STA results in 96.31% fewer inter-module gates on average compared to Qiskit (up to 98.32%) on the grid backend. Compared to PA-Qiskit, compilation with SEQC-STA results in 64.92% fewer inter-module gates on average, up to 78.62%, on the heavy-hex backend. Compared to PA-Qiskit, compilation with SEQC-STA results in 94.31% fewer inter-module gates on average, up to 97.83% on the grid backend.

It is worth mentioning that the inter-module gates in the universal backend would be CZ gates, while the non-universal backend would be SWAP gates. For every inter-module (CZ) gate in Qiskit, PA-Qiskit should have 2 inter-module (SWAP) gates. However, we observe PA-Qiskit produces circuits with a roughly comparable number of inter-module gates to Qiskit. In fact, as size increases, PA-Qiskit tends to result in fewer inter-module gates than Qiskit. This outcome may indicate the presence of optimizations that are removing extraneous SWAPs.

5.0.0.2. Fidelity. Practical applications of quantum computing require some guarantee of program output correctness (measurements). Thus, the highest priority for quantum



(a) Relative inter-module gate count (lower is better).



(b) Relative estimated fidelity (ESP) (higher is better).

Figure 5.2. Main circuit performance metrics: inter-module gate count and estimated fidelity, 120-qubit grid backend.

circuit compilation is to maximize fidelity. We estimate fidelity through estimated success probability (ESP) [55]. Specifically, we track the per-qubit ESP of a quantum circuit and report an averaged ESP for each benchmark from the geometric mean of the circuit measurements. Figures 5.1b and 5.2b presents the relative estimated fidelity of each compiler compared to the Qiskit compiler with respect to circuit size. There is a noticeable benefit in fidelity associated with the SEQC compiler, particularly with SEQC-STA. On the heavy-hex backend, we observe an average improvement of 22.13% over PA-Qiskit, and up to 62.87%. On the grid backend, SEQC-compiled circuits achieve 3.51% higher fidelity on average (22.50% max) over PA-Qiskit.

SEQC similarly outperforms Qiskit by a wide margin. It is important to emphasize here that SEQC outperforms Qiskit **even when the latter is afforded significant allowances** (no gate restrictions, and much lower inter-module gate errors). The Qiskit trial backend supports universal inter-module links, in contrast to the SEQC backend, and with only a 2.42% error for inter-module CZs, compared to 10.23% error for SEQC’s inter-module SWAPs. All these allowances should translate to higher fidelity solutions for Qiskit. Nevertheless, even though much more constrained, SEQC-STA demonstrates an

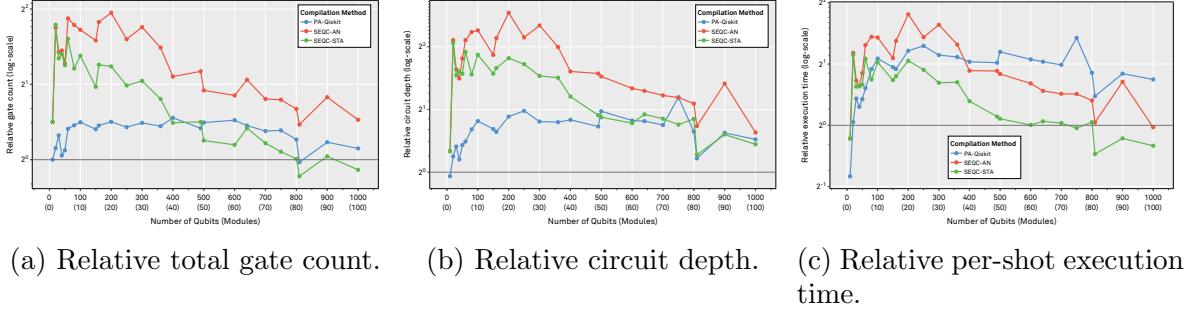


Figure 5.3. Total gate count, circuit depth, and per-shot execution time, 10-qubit heavy-hex backend (lower is better).

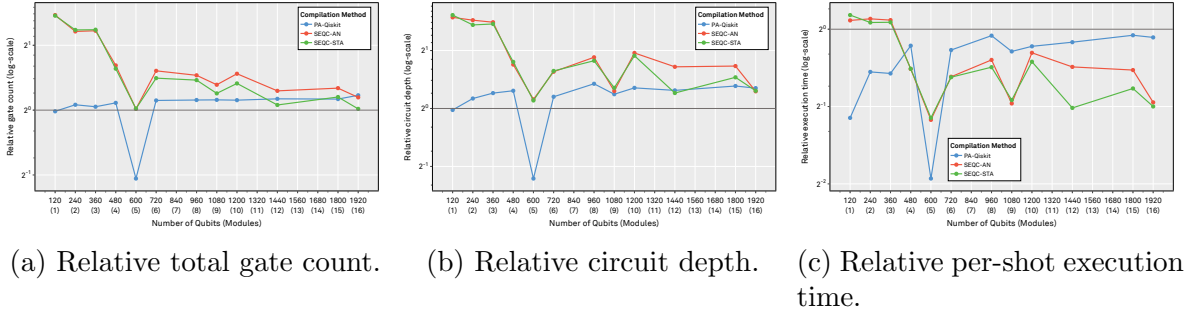


Figure 5.4. Total gate count, circuit depth, and per-shot execution time, 120-qubit grid backend (lower is better).

average improvement of 9.30%, up to a 49.99%, on the heavy-hex backend, and 32.30% (63.36% max) on the grid backend.

5.0.0.3. Gate count and circuit depth. Gate count and circuit depth are static, backend-agnostic metrics that can serve as indicators for a quantum circuit’s expected fidelity and/or execution time, where circuit depth is the critical path of a quantum circuit in terms of individual gates [46]. SABRE [78, 35], moreover, explicitly minimizes gate count by incorporating it into its cost functions. We observe that SEQC typically produces results with higher gate count (Figures 5.3a and 5.4a) and circuit depth (Figures 5.3b and 5.4b) compared to Qiskit and PA-Qiskit. This is due to SEQC prioritizing the

reduction of the number of *inter-module* gates over total gate count or circuit depth. Nevertheless, SEQC-STA, in particular, is able to demonstrate competitive results in these metrics for larger circuits.

SEQC-STA has comparable or slightly better results compared to PA-Qiskit at 500 qubits (50 modules) and above for both metrics on the heavy-hex backend. For the 810 and 1000 qubit (81 and 100 module) trials on the heavy-hex backend, SEQC-STA achieves 14.11% and 8.87% gate count reduction over Qiskit, respectively, **despite the latter using universal inter-module links**. For the grid backend, we observe similar behavior to the heavy-hex backend, where SEQC produces increasingly competitive results as the circuit size increases. However, we did not perform any trials large enough to witness the point where SEQC-STA would begin to produce better results than the Qiskit baselines

5.0.0.4. Estimated execution time. The per-shot execution time of a quantum circuit is determined by the critical path of gate operations weighted by gate duration. In addition to performance concerns, execution time indirectly affects fidelity on NISQ-era devices due to decoherence. In other words, it is imperative that quantum circuits execute within the decoherence time to ensure high-fidelity outcomes. [Figures 5.3c](#) and [5.4c](#) presents the estimated per-shot execution time of our proposed compilers relative to the reference as a function of the number of qubits per circuit. As explained previously, we approximate the duration of an inter-module SWAP as $4\times$ the duration of an intra-module SWAP [\[62\]](#). Under this model, we observe that the SEQC compilers consistently outperform PA-Qiskit in execution by 24.19% on average (63.98% max) on the heavy-hex backend. It is notable that, **despite the gate set affordances given to the Qiskit compiler**, SEQC-STA achieves parity in execution time starting from 600 qubits (60 modules). For 810

qubits (81 modules) and above, SEQC-STA actually compiles faster-executing quantum circuits. Meanwhile, on the grid backend, SEQC-STA-compiled circuits execute up to 151.29% faster over PA-Qiskit. Circuit execution time for SEQC-STA-compiled circuits is also faster than the Qiskit baseline from 480 qubits (4 modules) onwards by 28.04% on average, up to 54.82%, with SEQC-SA trailing close behind.

5.0.0.5. Compilation time. Quantum devices feature rapidly fluctuating error characteristics that can change hourly, or even on a run-to-run basis [64, 45, 15]. Thus, across the entire execution lifetime of an application (quantum circuit), regular re-compilation is necessary to generate the highest-quality circuits with the most up-to-date characteristics possible for every execution. Traditional quantum compilers, like Qiskit and PA-Qiskit, approach compilation *tabula rasa*, where each new compilation restarts from the original algorithmic circuit. It is challenging to speed up repeated recompilations without fundamental changes to their design, as noise-aware routing and synthesis algorithms, which leverage updated error characteristics, usually function from a device-agnostic starting point. In contrast, SEQC’s stratification stage enables stateful retention of previous compilations. It is agnostic to the ever-changing error characteristics of a quantum device, and instead only depends on the device’s static topology. Therefore, a stratified quantum circuit can be perennially reused over multiple rounds of compilation, and thus SEQC only repeats the elaboration stage.

Nevertheless, although stratification is a one-time cost for the SEQC compiler, it must be scalable in order to be practical. In Figures 5.5a and 5.6a, we plot the non-recurring compilation time of our evaluated compilers. For our SEQC compilers, we plot stratification time and stratification + elaboration time. These can be compared

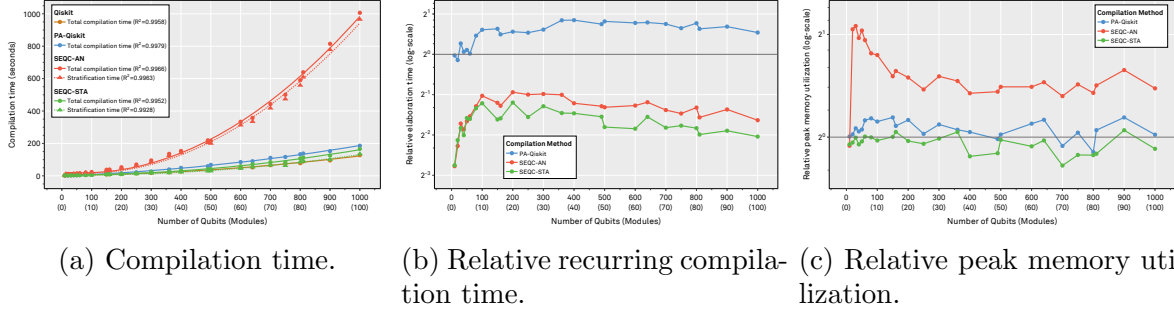
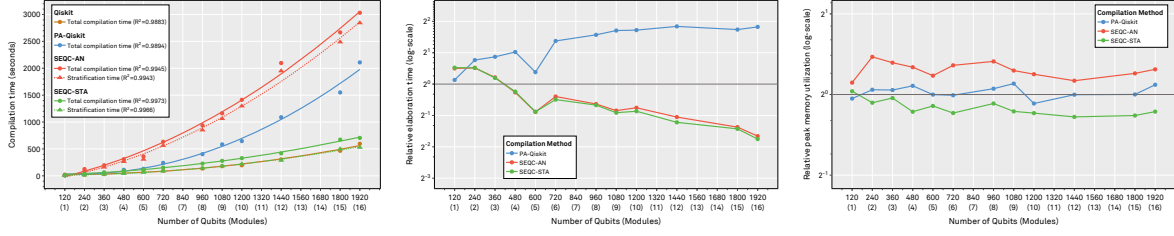


Figure 5.5. Compilation performance metrics: compilation time, recurring compilation time, and peak memory utilization, 10-qubit heavy-hex backend (lower is better).

to the total solve time for the Qiskit and PA-Qiskit compilers. We demonstrate that all compilation times obey a quadratic trajectory ($R^2 \geq 0.9883$).

Figures 5.5b and 5.6b show the recurring compilation time. Specifically, this refers to the elaboration time of SEQC and the full compilation time for Qiskit and PA-Qiskit. Overall, SEQC compilers have a large speedup over Qiskit-based compilers, owing to the use of parallelism and modular compilation strategies. Compared to Qiskit on the heavy-hex backend, SEQC-STA achieves $3.27\times$ speedup on average (up to $6.74\times$), and compared to PA-Qiskit on the heavy-hex backend, $4.69\times$ average speedup ($6.61\times$ max). Similarly on the grid backend, SEQC-STA achieves $1.34\times$ speedup on average (up to $3.37\times$) over Qiskit, and $2.52\times$ speedup on average (up to $11.93\times$) over PA-Qiskit.

5.0.0.6. Memory Utilization. Figures 5.5c and 5.6c presents the relative peak memory utilization of the evaluated compilers. Compared to Qiskit, SEQC-SA exhibits 55.68% higher memory usage on average (117.71% max) on the heavy-hex backend and 23.71% higher memory usage on average (38.24% max) on the grid backend. This could likely be ameliorated by reducing the number of annealing trials, at the expense of circuit quality. SEQC-STA tends to use less memory compared to the Qiskit variants, by 4.22% on average



(a) Compilation time. (b) Relative recurring compilation time. (c) Relative peak memory utilization.

Figure 5.6. Compilation performance metrics: compilation time, recurring compilation time, and peak memory utilization, 120-qubit grid backend (lower is better).

(17.47% max) on the heavy-hex backend and 10.69% on average (17.51% max) on the grid backend. It is possible that the process of stratification and parallelized elaboration is responsible for the conservation of memory usage, presuming the memory requirements of quantum compilation scales super-linearly with the circuit size. This points to the scalability benefits of the SEQC framework in general.

CHAPTER 6

Discussion

These results demonstrate a flaw with gate count and circuit depth as heuristics: their lack of noise-awareness. Circuit depth and gate count only consider the number of gates (in the critical path and in total) irrespective of each one’s unique underlying properties. It is expected that two-qubit gates will be significantly more noisy than single-qubit gates, and inter-module gates should be significantly more noisy than intra-module gates [62]. Failure to account for duration and error differences between gates or types of gates in quantum compilation heuristics will inevitably result in sub-optimal solutions when the discrepancies are extreme, e.g., in modular architectures. Conversely, the hierarchical structure of the SEQC framework with the stratification and elaboration stages imparts some degree of implicit noise-awareness, where minimizing the number of inter-module gates takes precedence over intra-module/total gates. As a result, SEQC can better optimize for the practically significant metrics of circuit fidelity and execution time.

Overall, we observe that SEQC yields circuits with more desirable characteristics when executing on modular quantum architectures, while at the same time it achieves faster compilation times. Also, as the number of qubits increases, the performance improvement of SEQC widens compared to the baseline, especially for the most important metrics of fidelity, compiled circuit execution time, and number of inter-module gates. These trends hold promise in allowing for faster and better compilation for future large-scale, hybrid modular quantum devices than we can achieve with today’s frameworks.

It is also important to note that SEQC is equally applicable to NISQ as well as future quantum systems. The stratification stage can decompose a large circuit into subcircuits that will be mapped to QPUs at the logical level, and then the elaboration stage can compile for each QPU together with an error correction method for that QPU. The two stages are distinct, and stratification operates more on the logical level, while the elaboration can be tasked with lowering from the logical to the physical qubits. Applicability to NISQ systems is still important, as even with technological breakthroughs, the transition to Fault Tolerant Quantum Computing (FTQC) will be gradual, not instantaneous. The hardware that exists now will continue improving and will be widely used for years, as there is practical scientific and industrial value in NISQ, even without the need to wait for fully error-corrected systems [53, 30, 57, 1]. Moreover, early fault-tolerant demonstrations will run on NISQ hardware, and will heavily rely on NISQ-era methods, as only few qubits can be protected. Thus, it remains critical to design architectures and compilers that serve NISQ hardware, both to extract near-term value, and to pave the engineering path toward full FTQC.

CHAPTER 7

Related Work

The state-of-the-art in distributed quantum computing (DQC) [71, 42, 39, 8, 16, 12] requires the generation and distribution of EPR pairs as the main entanglement resource over long-distance optical links, and the deployment of emitter-emitter, emitter-scatterer, or scatterer-scatterer photon-mediated entanglement schemes. However, modular architectures have less restrictive distance constraints, and they can often support other hardware technologies, such as microwave links [21]. As a result, SEQC can model inter-module allocation topologically (as SWAP chains) [34], similar to compilation for monolithic (single-die) quantum processors. An analogy can be drawn to classical compilation between chiplet-based classical systems and distributed classical systems, where diverse compilation strategies are required due to very different hardware technologies and distances between computation units.

Compilation for modular transmon systems shares many similar constraints with compilation for quantum charge-coupled devices (QCCDs). Namely, these devices features modules, in the form of ion traps, and support movement between traps, via ion shuttling mechanisms [48]. However, there are some key differences between the hardware technologies that make it difficult to compare SEQC with QCCD compilers on equal footing. Notably, qubits in superconducting devices needs to be topologically adjacent to each another to support two-qubit operations. However, QCCD devices are less restrictive, and support two-qubit operations so long as qubits are in the same traps / interaction

region. As a result, QCCD compilers [69, 31] can employ simpler routing strategies, or forgo them entirely, compared to modular transmon compilers.

MECH [76] entangles ancillary qubits and distributes them throughout a module to build fast network-on-chip routing channels (a "*communication highway*"). MECH includes a compiler that requires, and optimizes for, the "highway" design. Unfortunately, the highway does not generalize to non-universal inter-module links. Future work to extend SEQC with MECH's methodology may enable combining MECH's techniques with SEQC's allocation procedures to gain the benefits of each. Lin *et al.* [37] present a compilation technique that utilizes calibration data to determine potentially nonstandard 2Q gates that can perform standard operations along bases more compatible with their respective physical states and improve fidelity. Bandic *et al.* [5] utilize the relationship between quantum circuit structure and mapping efficiency to optimize the architecture of monolithic and modular quantum processors, on the path to application-specific hardware optimization. Escofet *et al.* [19] develop a characterization technique for arbitrary quantum circuits, which provides theoretical upper/lower optimization bounds, enabling the optimality assessment of qubit mapping solutions. These techniques are orthogonal to SEQC, and could work synergistically with it.

CutQC [36] partitions large quantum circuits into smaller subcircuits that fit on a device, executes them one at a time, and then merges the individual results through classical post-processing. Piveteau and Sutter [54] similarly allow subcircuits to execute independently, and use *quasiprobability simulation* to simulate inter-module link behavior where the subcircuits connect, reaching fixed accuracy with less simulation time. SEQC's stratification stage was inspired by CutQC, but unlike these works SEQC does not perform any

classical “stitching” or simulation. We initially formulated qubit-to-subcircuit mapping as a constraint optimization problem, similar to CutQC, but existing numerical solvers were impractically slow for circuits larger than 20 qubits, corroborating the experience of Baker *et al.* [4]. Instead, we opted to use the approximate, but practical, solutions we currently employ in SEQC.

Bandic *et al.* [6] map circuit partitioning to min-cut and use the QUBO model with quantum solvers to generate a set of cuts intended to minimize swaps between partitions. QUBO employs quantum optimization algorithms on quantum annealers to accelerate the solution process and remain practical, as the search space for transmission qubits is 2^n . Instead, SEQC takes a “best-effort” approach that works well in practice, and does not require acceleration through quantum means.

Liu *et al.* [38] partition a circuit into blocks that span an identical number of qubits, perform permutation-aware synthesis on each block in parallel (solving for all possible qubit topologies), and finally use an augmented SABRE algorithm to map the blocks to the target device and route between blocks. SEQC takes a top-down approach instead, where SWAP permutations between modules are settled prior to any compilation tasks performed for each module (block). As a result, SEQC only needs to solve for one qubit layout per block, rather than all permutations. Hungarian Qubit Assignment (HQA) [18] extends the Hungarian algorithm with the ability to adjust a qubit’s cost matrix according to behavior multiple time slices ahead. Pastor *et al.* [51] use deep reinforcement learning to partition quantum circuits for execution across multiple modules. Both works aim to reduce inter-module swaps, but assume that all inter-module communications have

the same cost, i.e., modules are fully connected, which is both impractical for large-scale systems and misses optimization opportunities. SEQC, in contrast, can work with arbitrary module topologies.

Ovide *et al.* [49] leverage both quantum and classical methods for inter-module communication. While it advocates for a two-level approach to quantum circuit mapping, it leaves the actual problem of qubit placement and routing unsolved, which SEQC addresses with its algorithm.

CHAPTER 8

Conclusion

The increasing complexity and scale of modular quantum devices challenges the scalability of quantum compilers, while their lack of module awareness impedes their ability to produce high-quality results. The compiler we propose, SEQC, implements several novel methods for qubit placement, qubit routing, and circuit optimization, with both hardware modularity and compilation parallelization in mind. While SEQC can be applied on a modular quantum system comprising hybrid QPUs, in this paper we consider the simpler case of homogeneous modules as a starting point, in order to isolate the impact of our techniques from the effects of heterogeneity and provide comparison points with existing infrastructures. SEQC achieves on average 9.3-32.3% higher circuit fidelity (up to 49.99-63.36%), depending on the module size and topology, attaining an average 1.34-3.27 $\times$ faster compilation (up to 3.37-6.74 $\times$) compared to a module-unaware Qiskit baseline.

Bibliography

- [1] Dmitry A. Abanin et al. 2025. Observation of constructive interference at the edge of quantum ergodicity. *Nature*, 646, 8086, 825–830. DOI: [10.1038/s41586-025-09526-6](https://doi.org/10.1038/s41586-025-09526-6).
- [2] Rajeev Acharya et al. 2023. Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614, 7949, (Feb. 2023), 676–681. DOI: [10.1038/s41586-022-05434-1](https://doi.org/10.1038/s41586-022-05434-1).
- [3] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. 2006. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA. ISBN: 0321486811.
- [4] Jonathan M. Baker, Casey Duckering, Alexander Hoover, and Frederic T. Chong. 2020. Time-sliced quantum circuit partitioning for modular architectures. In *Proceedings of the 17th ACM International Conference on Computing Frontiers (CF '20)*. Association for Computing Machinery, Catania, Sicily, Italy, 98–107. ISBN: 9781450379564. DOI: [10.1145/3387902.3392617](https://doi.org/10.1145/3387902.3392617).
- [5] Medina Bandic, Pablo le Henaff, Anabel Ovide, Pau Escofet, Sahar Ben Rached, Santiago Rodrigo, Hans van Someren, Sergi Abadal, Eduard Alarcon, Carmen G. Almudever, et al. [n. d.] Profiling quantum circuits for their efficient execution on single-and multi-core architectures. *Quantum Science and Technology*.
- [6] Medina Bandic, Luise Prielinger, Jonas Nüßlein, Anabel Ovide, Santiago Rodrigo, Sergi Abadal, Hans van Someren, Gayane Vardoyan, Eduard Alarcon, Carmen G Almudever, et al. 2023. Mapping quantum circuits to modular architectures with QUBO. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 1. IEEE, 790–801.

- [7] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. 2018. Spec cpu2017: next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering* (ICPE '18). Association for Computing Machinery, Berlin, Germany, 41–42. ISBN: 9781450356299. DOI: [10.1145/3185768.3185771](https://doi.org/10.1145/3185768.3185771).
- [8] Felix Burt, Kuan-Cheng Chen, and Kin K. Leung. 2024. Generalised circuit partitioning for distributed quantum computing. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 02, 173–178. DOI: [10.1109/QCE60285.2024.10273](https://doi.org/10.1109/QCE60285.2024.10273).
- [9] Christopher Chamberland, Guanyu Zhu, Theodore J. Yoder, Jared B. Hertzberg, and Andrew W. Cross. 2020. Topological and subsystem codes on low-degree graphs with flag qubits. *Phys. Rev. X*, 10, (Jan. 2020), 011022, 1, (Jan. 2020). DOI: [10.1103/PhysRevX.10.011022](https://doi.org/10.1103/PhysRevX.10.011022).
- [10] Cleaven Chia, Ding Huang, Victor Leong, Jian Feng Kong, and Kuan Eng Johnson Goh. 2024. Hybrid quantum systems with artificial atoms in solid state. *Advanced Quantum Technologies*, 7, 5, 2300461. eprint: <https://advanced.onlinelibrary.wiley.com/doi/pdf/10.1002/qute.202300461>, DOI: <https://doi.org/10.1002/qute.202300461>.
- [11] A. A. Clerk, K. W. Lehnert, P. Bertet, J. R. Petta, and Y. Nakamura. 2020. Hybrid quantum systems with circuit quantum electrodynamics. *Nature Physics*, 16, 3, (Mar. 2020), 257–267. DOI: [10.1038/s41567-020-0797-9](https://doi.org/10.1038/s41567-020-0797-9).
- [12] Daniele Cuomo, Marcello Caleffi, Kevin Krsulich, Filippo Tramonto, Gabriele Agliardi, Enrico Prati, and Angela Sara Cacciapuoti. 2023. Optimized compiler for distributed quantum computing. *ACM Transactions on Quantum Computing*, 4, 2, Article 15, (Feb. 2023), 29 pages. DOI: [10.1145/3579367](https://doi.org/10.1145/3579367).
- [13] Nikos Daniilidis, Dylan J Gorman, Lin Tian, and Hartmut Häffner. 2013. Quantum information processing with trapped electrons and superconducting electronics. *New Journal of Physics*, 15, 7, (July 2013), 073017. DOI: [10.1088/1367-2630/15/7/073017](https://doi.org/10.1088/1367-2630/15/7/073017).
- [14] DARPA. 2025. Heterogeneous architectures for quantum (harq). Available online at: <https://sam.gov/opp/967cd8f4c3554b448d7ba3325ed99de2/view>, (Aug. 2025).

- [15] Samudra Dasgupta and Travis S Humble. 2021. Stability of noisy quantum computing devices. *arXiv preprint arXiv:2105.09472*.
- [16] Marc G. Davis, Joaquin Chung, Dirk Englund, and Rajkumar Kettimuthu. 2023. Towards Distributed Quantum Computing by Qubit and Gate Graph Partitioning Techniques. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE Computer Society, Los Alamitos, CA, USA, (Sept. 2023), 161–167. DOI: [10.1109/QCE57702.2023.00026](https://doi.org/10.1109/QCE57702.2023.00026).
- [17] Christopher M. Dawson and Michael A. Nielsen. 2006. The Solovay-Kitaev algorithm. *Quantum Info. Comput.*, 6, 1, (Jan. 2006), 81–95.
- [18] Pau Escofet, Anabel Ovide, Carmen G Almudever, Eduard Alarcón, and Sergi Abadal. 2023. Hungarian qubit assignment for optimized mapping of quantum circuits on multi-core architectures. *IEEE Computer Architecture Letters*.
- [19] Pau Escofet, Anabel Ovide, Medina Bandic, Luise Prielinger, Hans van Someren, Sebastian Feld, Eduard Alarcón, Sergi Abadal, and Carmen G. Almudéver. 2024. Revisiting the mapping of quantum circuits: entering the multi-core era. (2024). arXiv: [2403.17205 \[quant-ph\]](https://arxiv.org/abs/2403.17205).
- [20] Mark Field, Angela Q. Chen, Ben Scharmann, Eyob A. Sete, Feyza Oruc, Kim Vu, Valentin Kosenko, Joshua Y. Mutus, Stefano Poletto, and Andrew Bestwick. 2024. Modular superconducting-qubit architecture with a multichip tunable coupler. *Phys. Rev. Appl.*, 21, (May 2024), 054063, 5, (May 2024). DOI: [10.1103/PhysRevApplied.21.054063](https://doi.org/10.1103/PhysRevApplied.21.054063).
- [21] Mark Field, Angela Q. Chen, Ben Scharmann, Eyob A. Sete, Feyza Oruc, Kim Vu, Valentin Kosenko, Joshua Y. Mutus, Stefano Poletto, and Andrew Bestwick. 2024. Modular superconducting-qubit architecture with a multichip tunable coupler. *Phys. Rev. Appl.*, 21, (May 2024), 054063, 5, (May 2024). DOI: [10.1103/PhysRevApplied.21.054063](https://doi.org/10.1103/PhysRevApplied.21.054063).
- [22] Jay Gambetta. 2023. The hardware and software for the era of quantum utility is here. [Accessed 04-14-2024]. IBM, (Dec. 2023). <https://www.ibm.com/quantum/blog/quantum-roadmap-2033>.
- [23] Jay Gambetta and Ryan Mandelbaum. 2024. IBM quantum delivers on performance challenge made two years ago. IBM Quantum Developer Conference (<https://www.ibm.com/quantum/blog/qdc-2024>). (2024).

- [24] Alysson Gold, JP Paquette, Anna Stockklauser, Matthew J Reagor, M Sohaib Alam, Andrew Bestwick, Nicolas Didier, Ani Nersisyan, Feyza Oruc, Armin Razavi, et al. 2021. Entanglement across separate silicon dies in a modular superconducting qubit device. *npj Quantum Information*, 7, 1, 142.
- [25] HPCWire. 2024. IonQ plots path to commercial (quantum) advantage. Available online at: <https://www.hpcwire.com/2024/07/02/ionq-plots-path-to-commercial-quantum-advantage/>. (July 2024).
- [26] Economist Impact. 2025. The superposition of impact and hype: building a quantum-powered future, one qubit at a time. Available online at: https://youtu.be/vvJvhoRPTgo?si=HGlxiHfK_PON0ZfM&t=927 (May 2025).
- [27] Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto. 2023. Algorithmic theory of qubit routing. In *Algorithms and Data Structures*. Pat Morin and Subhash Suri, (Eds.) Springer Nature Switzerland, Cham, 533–546. ISBN: 978-3-031-38906-1.
- [28] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. 2024. Quantum computing with Qiskit. (2024). arXiv: [2405.08810 \[quant-ph\]](https://arxiv.org/abs/2405.08810). DOI: [10.48550/arXiv.2405.08810](https://doi.org/10.48550/arXiv.2405.08810).
- [29] D. Kielpinski, D. Kafri, M. J. Woolley, G. J. Milburn, and J. M. Taylor. 2012. Quantum interface between an electrical circuit and a single atom. *Phys. Rev. Lett.*, 108, (Mar. 2012), 130504, 13, (Mar. 2012). DOI: [10.1103/PhysRevLett.108.130504](https://doi.org/10.1103/PhysRevLett.108.130504).
- [30] Youngseok Kim, Andrew Eddins, Sajant Anand, Ken Xuan Wei, Ewout van den Berg, Sami Rosenblatt, Hasan Nayfeh, Yantao Wu, Michael Zaletel, Kristan Temme, and Abhinav Kandala. 2023. Evidence for the utility of quantum computing before fault tolerance. *Nature*, 618, 7965, 500–505. DOI: [10.1038/s41586-023-06096-3](https://doi.org/10.1038/s41586-023-06096-3).

- [31] Fabian Kreppel, Christian Melzer, Diego Olvera Millán, Janis Wagner, Janine Hilder, Ulrich Poschinger, Ferdinand Schmidt-Kaler, and André Brinkmann. 2023. Quantum Circuit Compiler for a Shuttling-Based Trapped-Ion Quantum Computer. *Quantum*, 7, (Nov. 2023), 1176. DOI: [10.22331/q-2023-11-08-1176](https://doi.org/10.22331/q-2023-11-08-1176).
- [32] Alon Kukliansky, Ed Younis, Lukasz Cincio, and Costin Iancu. 2023. Qfactor: a domain-specific optimizer for quantum circuit instantiation. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 01, 814–824. DOI: [10.1109/QCE57702.2023.00096](https://doi.org/10.1109/QCE57702.2023.00096).
- [33] Gershon Kurizki, Patrice Bertet, Yuimaru Kubo, Klaus Mølmer, David Petrosyan, Peter Rabl, and Jörg Schmiedmayer. 2015. Quantum technologies with hybrid systems. *Proceedings of the National Academy of Sciences*, 112, 13, 3866–3873. eprint: <https://www.pnas.org/doi/pdf/10.1073/pnas.1419326112> DOI: [10.1073/pnas.1419326112](https://doi.org/10.1073/pnas.1419326112).
- [34] Nicholas LaRacuente, Kaitlin N. Smith, Poolad Imany, Kevin L. Silverman, and Frederic T. Chong. 2025. Modeling Short-Range Microwave Networks to Scale Superconducting Quantum Computation. *Quantum*, 9, (Jan. 2025), 1581. DOI: [10.22331/q-2025-01-08-1581](https://doi.org/10.22331/q-2025-01-08-1581).
- [35] Gushu Li, Yufei Ding, and Yuan Xie. 2019. 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 (ASPLOS '19)*. Association for Computing Machinery, Providence, RI, USA, 1001–1014. ISBN: 9781450362405. DOI: [10.1145/3297858.3304023](https://doi.org/10.1145/3297858.3304023).
- [36] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the qubit mapping problem for nisc-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*. Association for Computing Machinery, Providence, RI, USA, 1001–1014. ISBN: 9781450362405. DOI: [10.1145/3297858.3304023](https://doi.org/10.1145/3297858.3304023).
- [37] Sophia Fuhui Lin, Sara Sussman, Casey Duckering, Pranav S. Mundada, Jonathan M. Baker, Rohan S. Kumar, Andrew A. Houck, and Frederic T. Chong. 2023. Let each quantum bit choose its basis gates. In *Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture*

- (MICRO '22). IEEE Press, Chicago, Illinois, USA, 1042–1058. ISBN: 9781665462723. DOI: [10.1109/MICRO56248.2022.00075](https://doi.org/10.1109/MICRO56248.2022.00075).
- [38] Ji Liu, Ed Younis, Mathias Weiden, Paul Hovland, John Kubitowicz, and Costin Iancu. 2023. Tackling the qubit mapping problem with permutation-aware synthesis. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 01, 745–756. DOI: [10.1109/QCE57702.2023.00090](https://doi.org/10.1109/QCE57702.2023.00090).
- [39] Lei Liu. 2025. Larqcut: a new cutting and mapping approach for large-sized quantum circuits in distributed quantum computing (dq) environments. *ACM Trans. Archit. Code Optim.*, (Apr. 2025). Just Accepted. DOI: [10.1145/3730585](https://doi.org/10.1145/3730585).
- [40] Ryan Mandelbaum, Antonio D. Córcoles, and Jay Gambetta. 2024. Ibm’s big bet on the quantum-centric supercomputer: recent advances point the way to useful classical-quantum hybrids. *IEEE Spectrum*, 61, 9, 24–33. DOI: [10.1109/MSPEC.2024.10669253](https://doi.org/10.1109/MSPEC.2024.10669253).
- [41] Ryan Mandelbaum, Jay Gambetta, Jerry Chow, Tushar Mittal, Theodore J. Yoder, Andrew Cross, and Matthias Steffen. 2025. How IBM will build the world’s first large-scale, fault-tolerant quantum computer. Available online at: <https://www.ibm.com/quantum/blog/large-scale-ftqc/> (June 2025).
- [42] Yingling Mao, Yu Liu, and Yuanyuan Yang. 2023. Qubit allocation for distributed quantum computing. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, 1–10. DOI: [10.1109/INFOCOM53939.2023.10228915](https://doi.org/10.1109/INFOCOM53939.2023.10228915).
- [43] Dmitri Maslov, Gerhard W. Dueck, D. Michael Miller, and Camille Negrevergne. 2008. Quantum circuit simplification and level compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27, 3, 436–444. DOI: [10.1109/TCAD.2007.911334](https://doi.org/10.1109/TCAD.2007.911334).
- [44] David C. McKay, Ravi Naik, Philip Reinhold, Lev S. Bishop, and David I. Schuster. 2015. High-contrast qubit interactions using multimode cavity qed. *Phys. Rev. Lett.*, 114, (Feb. 2015), 080501, 8, (Feb. 2015). DOI: [10.1103/PhysRevLett.114.080501](https://doi.org/10.1103/PhysRevLett.114.080501).

- [45] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. 2019. 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 (ASPLOS '19)*. Association for Computing Machinery, Providence, RI, USA, 1015–1029. ISBN: 9781450362405. DOI: [10.1145/3297858.3304075](https://doi.org/10.1145/3297858.3304075).
- [46] Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press. DOI: <https://doi.org/10.1017/CB09780511976667>.
- [47] Jingjing Niu, Libo Zhang, Yang Liu, Jiawei Qiu, Wenhui Huang, Jiaxiang Huang, Hao Jia, Jiawei Liu, Ziyu Tao, Weiwei Wei, et al. 2023. Low-loss interconnects for modular superconducting quantum processors. *Nature Electronics*, 6, 3, 235–241.
- [48] Anabel Ovide, Daniele Cuomo, and Carmen G. Almudever. 2024. Scaling and assigning resources on ion trap qccd architectures. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 01, 959–970. DOI: [10.1109/QCE60285.2024.00115](https://doi.org/10.1109/QCE60285.2024.00115).
- [49] Anabel Ovide, Santiago Rodrigo, Medina Bandic, Hans Van Someren, Sebastian Feld, Sergi Abadal, Eduard Alarcon, and Carmen G Almudever. 2023. Mapping quantum algorithms to multi-core quantum computing architectures. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.
- [50] Alexandru Paler, Daniel Herr, and Simon J Devitt. 2019. Really small shoe boxes: on realistic quantum resource estimation. *Computer*, 52, 6, 27–37.
- [51] Arnau Pastor, Pau Escofet, Sahar Ben Rached, Eduard Alarcón, Pere Barlet-Ros, and Sergi Abadal. 2024. Circuit partitioning for multi-core quantum architectures with deep reinforcement learning. *arXiv preprint arXiv:2401.17976*.
- [52] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O’Brien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5, 1, (July 2014), 4213. DOI: [10.1038/ncomms5213](https://doi.org/10.1038/ncomms5213).

- [53] Samuele Piccinelli, Alberto Baiardi, Max Rossmannek, Almudena Carrera Vazquez, Francesco Tacchino, Stefano Mensa, Edoardo Altamura, Ali Alavi, Mario Motta, Javier Robledo-Moreno, William Kirby, Kunal Sharma, Antonio Mezzacapo, and Ivano Tavernelli. 2025. Quantum chemistry with provable convergence via randomized sample-based quantum diagonalization. (2025). <https://arxiv.org/abs/2508.02578> arXiv: 2508.02578 [quant-ph].
- [54] Christophe Piveteau and David Sutter. 2024. Circuit knitting with classical communication. *IEEE Transactions on Information Theory*, 70, 4, 2734–2745. DOI: [10.1109/TIT.2023.3310797](https://doi.org/10.1109/TIT.2023.3310797).
- [55] Fang Qi, Kaitlin N. Smith, Travis LeCompte, Nian-feng Tzeng, Xu Yuan, Frederic T. Chong, and Lu Peng. 2024. Quantum Vulnerability Analysis to Guide Robust Quantum Computing System Design. *IEEE Transactions on Quantum Engineering*, 5, 01, (Jan. 2024), 1–11. DOI: [10.1109/TQE.2023.3343625](https://doi.org/10.1109/TQE.2023.3343625).
- [56] Rigetti. 2024. Investor presentation. Available online at: <https://investors.rigetti.com/statistic-files/fbac3801-223f-4f0f-a207-47d25084a1d7>. (Nov. 2024).
- [57] Javier Robledo-Moreno, Mario Motta, Holger Haas, Ali Javadi-Abhari, Petar Jurcevic, William Kirby, Simon Martiel, Kunal Sharma, Sandeep Sharma, Tomonori Shirakawa, Iskandar Sitdikov, Rong-Yang Sun, Kevin J. Sung, Maika Takita, Minh C. Tran, Seiji Yunoki, and Antonio Mezzacapo. 2025. Chemistry beyond the scale of exact diagonalization on a quantum-centric supercomputer. *Science Advances*, 11, 25, eadu9991. DOI: [10.1126/sciadv.adu9991](https://doi.org/10.1126/sciadv.adu9991).
- [58] R.A. Rutenbar. 1989. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine*, 5, 1, 19–26. DOI: [10.1109/101.17235](https://doi.org/10.1109/101.17235).
- [59] SiliconAngle. 2021. Rigetti debuts multichip quantum processor with 80 qubits. Available online at: <https://siliconangle.com/2021/06/29/rigetti-looks-scale-quantum-computing-modular-processor-architecture/>. (June 2021).
- [60] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintao Pereira. 2018. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization (CGO 2018)*. Association for Computing Machinery, Vienna, Austria, 113–125. ISBN: 9781450356176. DOI: [10.1145/3168822](https://doi.org/10.1145/3168822).

- [61] Kaitlin N Smith and Mitchell A Thornton. 2019. A quantum computational compiler and design tool for technology-specific targets. In *Proceedings of the 46th International Symposium on Computer Architecture*, 579–588.
- [62] Kaitlin N. Smith, Gokul Subramanian Ravi, Jonathan M. Baker, and Frederic T. Chong. 2022. Scaling superconducting quantum computers with chiplet architectures. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 1092–1109. DOI: [10.1109/MICRO56248.2022.00078](https://doi.org/10.1109/MICRO56248.2022.00078)
- [63] Samuel Stein, Sara Sussman, Teague Tomesh, Charles Guinn, Esin Tureci, Sophia Fuhui Lin, Wei Tang, James Ang, Srivatsan Chakram, Ang Li, Margaret Martonosi, Fred Chong, Andrew A. Houck, Isaac L. Chuang, and Michael Demarco. 2023. Hetarch: heterogeneous microarchitectures for superconducting quantum systems. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23)*. Association for Computing Machinery, Toronto, ON, Canada, 539–554. ISBN: 9798400703294. DOI: [10.1145/3613424.3614300](https://doi.org/10.1145/3613424.3614300).
- [64] Swamit S. Tannu and Moinuddin K. Qureshi. 2019. 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 (ASPLOS '19)*. Association for Computing Machinery, Providence, RI, USA, 987–999. ISBN: 9781450362405. DOI: [10.1145/3297858.3304007](https://doi.org/10.1145/3297858.3304007)
- [65] T. Tomesh, P. Gokhale, V. Omole, G. Ravi, K. N. Smith, J. Viszlai, X. Wu, N. Hardavellas, M. R. Martonosi, and F. T. Chong. 2022. Supermarq: a scalable quantum benchmark suite. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE Computer Society, Los Alamitos, CA, USA, (Apr. 2022), 587–603. DOI: [10.1109/HPCA53966.2022.00050](https://doi.org/10.1109/HPCA53966.2022.00050)
- [66] [n. d.] Translation Errors — IBM Quantum Documentation — docs.quantum.ibm.com. [Accessed 2025-02-20]. <https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.passes.BasisTranslator#translation-errors%7D>.
- [67] Trueman, Charlotte. 2025. Rigetti makes its 36-qubit cepheus-1-36q quantum computer available in the cloud. Available online at: <https://www.datacenterdynamics.com/en/news/rigetti-m>

- [akes-its-36-qubit-cepheus-1-36q-quantum-computer-available-in-the-cloud/](#) (Aug. 2025).
- [68] University of Illinois Urbana-Champaign. 2020. Hybrid quantum architectures and networks (hqan). Available online at: <https://hqan.illinois.edu/>. (Sept. 2020).
- [69] Suryansh Upadhyay, Abdullah Ash Saki, Rasit Onur Topaloglu, and Swaroop Ghosh. 2022. A shuttle-efficient qubit mapper for trapped-ion quantum computers. In *Proceedings of the Great Lakes Symposium on VLSI 2022* (GLSVLSI '22). Association for Computing Machinery, Irvine, CA, USA, 305–308. ISBN: 9781450393225. DOI: [10.1145/3526241.3530366](https://doi.org/10.1145/3526241.3530366).
- [70] Rodney Van Meter, Thaddeus D. Ladd, Austin G. Fowler, and Yoshihisa Yamamoto. 2010. Distributed quantum computation architecture using semiconductor nanophotonics. *International Journal of Quantum Information*, 08, 01n02, 295–323. eprint: <https://doi.org/10.1142/S0219749910006435>. DOI: [10.1142/S0219749910006435](https://doi.org/10.1142/S0219749910006435).
- [71] Anbang Wu, Hezi Zhang, Gushu Li, Alireza Shabani, Yuan Xie, and Yufei Ding. 2022. Autocomm: a framework for enabling efficient communication in distributed quantum programs. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 1027–1041. DOI: [10.1109/MICRO56248.2022.00074](https://doi.org/10.1109/MICRO56248.2022.00074).
- [72] Xin-Chuan Wu, Marc Grau Davis, Frederic T. Chong, and Costin Iancu. 2021. Reoptimization of quantum circuits via hierarchical synthesis. In *2021 International Conference on Rebooting Computing (ICRC)*, 35–46. DOI: [10.1109/ICRC53822.2021.00016](https://doi.org/10.1109/ICRC53822.2021.00016).
- [73] Ze-Liang Xiang, Sahel Ashhab, J. Q. You, and Franco Nori. 2013. Hybrid quantum circuits: superconducting circuits interacting with other quantum systems. *Rev. Mod. Phys.*, 85, (Apr. 2013), 623–653, 2, (Apr. 2013). DOI: [10.1103/RevModPhys.85.623](https://doi.org/10.1103/RevModPhys.85.623).
- [74] Ed Younis, Costin C Iancu, Wim Lavrijsen, Marc Davis, Ethan Smith, and USDOE. 2021. Berkeley quantum synthesis toolkit (bqskit) v1. (Apr. 2021). DOI: [10.11578/dc.20210603.2](https://doi.org/10.11578/dc.20210603.2).
- [75] Deshui Yu, María Martínez Valado, Christoph Hufnagel, Leong Chuan Kwek, Luigi Amico, and Rainer Dumke. 2016. Quantum state transmission in a superconducting charge qubit-atom hybrid. *Scientific Reports*, 6, 1, (Dec. 2016), 38356. DOI: [10.1038/srep38356](https://doi.org/10.1038/srep38356).

- [76] Hezi Zhang, Keyi Yin, Anbang Wu, Hassan Shapourian, Alireza Shabani, and Yufei Ding. 2024. Mech: multi-entry communication highway for superconducting quantum chiplets. (2024). arXiv: [2305.05149 \[quant-ph\]](#).
- [77] Jian Zhou, Yong Hu, Zhang-qi Yin, Z. D. Wang, Shi-Liang Zhu, and Zheng-Yuan Xue. 2014. High fidelity quantum state transfer in electromechanical systems with intermediate coupling. *Scientific Reports*, 4, 1, (Aug. 2014), 6237. DOI: [10.1038/srep06237](#).
- [78] Henry Zou, Matthew Treinish, Kevin Hartman, Alexander Ivrii, and Jake Lishman. 2024. Lightsabre: a lightweight and enhanced sabre algorithm. (2024). <https://arxiv.org/abs/2409.08368> arXiv: [2409.08368 \[quant-ph\]](#).