Extensions to QStack: Virtual Qubit Routing and SuperMarQ Benchmarks

Ahmed Darwish
Chair of Distributed Systems and Operating Systems
Department of Computer Science
School of Computation, Information, and Technology
Technical University of Munich

Abstract

There remains a need for better software for quantum computing that is able to achieve the full potential of the commercially-present NISQ hardware, despite its inherent flaws and imperfections, and the lack of sufficient error correction. Such software needs to be able to transform quantum circuits, schedule jobs on the quantum hardware, and allocate resources such that noise is minimized while maximizing utilization. And so, during my Guided Research, I contributed to the design and development of QStack, a software library for noise-aware circuit execution, as well as its sub-component, the Quantum Virtual Machine (QVM), which utilizes gate virtualization techniques to mitigate noise and fit large quantum circuits unto smaller-sized hardware. I migrated the peer-reviewed SuperMarQ benchmarking suite to the Qiskit quantum framework, such that it is compatible with our library, and implemented the first version of a user configuration class for jobs submitted to the system. I also developed a heuristic for intra-chip gate virtualization, that is able to perform better than Qiskit's trivial routing algorithm. An empirical experiment demonstrates the effectiveness of the heuristic. In this report, I shall summarize my contributions, and discuss the experiments that I ran.

1 Introduction

Achieving the theorized quantum advantage is a multi-faceted challenge that does not stop at building hardware of sufficient scale, but also suppressing the noise of such hardware to the point it is not hindering reliable results anymore. While companies such as IBM already have a roadmap [1] to scale up their quantum chips, the existing software is still not realizing the full potential of the existing hardware, and not sufficiently ready for the upcoming scaling up in quantum resources. By the latter statement I mean the scarcity of high-quality software that is able to utilize all the quantum chips available to the user to run some quantum program even if the program does not fit on one of the those chips, but would require some notion of distributed computing to break down this program into smaller subprograms that would then run on the smaller chips, aggregating their outputs at a later stage to produce the output of the original program. Different works of literature have proposed several techniques for realizing this notion, such as circuit cutting [2], which splits a program along the operations of some qubit, and gate virtualization [3], which replaces two-qubit gates with an equivalent set of single-qubit gates.

During the guided research project, I contributed to a proposed system architecture for quantum computing that tackles both challenges subsequently. The architecture, denoted QStack, first analyzes the given program and determines whether it needs to be broken down into smaller programs, an operation called "cutting", thus addressing the problem of scalability. In the current iteration of the system gate virtualization is used, but the system design is compatible with any technique that uses a similar break-then-aggregate methodology. The aggregation unit is called "knitting". Afterwards, the resulting programs are passed on to the noise-aware circuit-to-hardware mapper and scheduler, which automatically chooses the quantum chip that would optimally execute the program with the least noise. This architecture is an improvement over the conventional compilation pipeline [4], which simply translates high-level quantum programs to low-level native quantum instructions. It is worth mentioning that the methodology of the our stack makes it also compatible with simulators, and therefore could interface with a heterogeneous pool of quantum backends.

The remainder of this report will be structured as follows. Section 2 discusses work already done in the QVM component and elaborates further on the inner workings of gate virtualization. In addition, transpilation of quantum circuits is briefly introduced. Finally, the SuperMarQ benchmarks added to our system are introduced and discussed in more detail. In Section 3, I describe the work that I have carried out for the QVM sub-component. An exploratory experiment that investigated the effect of entanglement of the effectiveness of gate virtualization is described, followed by the introduction

of a new usage of QVM; Virtual Qubit Routing. Due to some organizational restructuring that occurred during the project, I did not get the opportunity to contribute to QOS further, and so it shall be granted no further space of this report. Section 4 contains a continuation of my contributions but within the general scope of QStack, starting with an overview of the specification of the user's jobs to the system, and a brief illustration of the differences between Google's Cirq[1] and IBM's Qiskit[2] that I faced when migrating the SuperMarQ benchmarks [5]. In section 5, I discuss gaps in our work that can be addressed in future work.

2 Background

2.1 Circuit Decomposition using Gate Virtualization

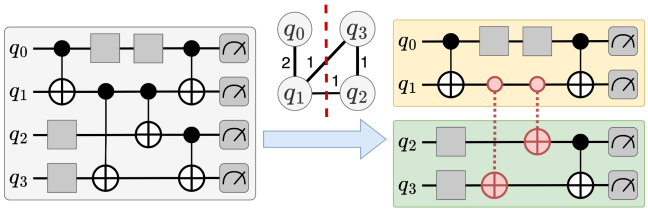

Figure 1: An example of circuit decomposition using Gate Virtualiztion. Assuming only 2-qubit chips are present, the original 4-qubit circuit can be bifurcated by virtualizing two CNOT gates into equally-sized fragments that can then run on the available hardware.

In his Bachelor's thesis, Nathaniel Tornow implemented a circuit decomposition component that is able to break down a large circuit; one that does not fit on any of the available quantum hardware, into smaller fragments that are then executable on the limited-size chips [6]. The execution results of the fragments are then aggregated as described in the work introducing gate virtualization [3]. The resulting number of the fragments and the size of each is dependent on the quantum backends available to the system. Moreover, a user can forcibly fragment a circuit if the quantum chip that could contain the circuit is known to be unreliably noisy for the specific application the user is intending, and so they may choose to rely on smaller, less noisy quantum chips instead. Circuit decomposition is also applicable to quantum simulators that run on classical hardware. Pure statevector simulators can only run circuits with a number of qubits in the order of tens, and even then, it could take up to 2 hours to run a quantum circuit on high-performance hardware [7]; consuming both memory and compute time. Through gate virtualization, a simulation can be reduced in size such that it is executable on more modest hardware, or executed in a shorter period of time. An example of circuit decomposition can be seen in Figure 1.

The reduction in quantum resources that gate virtualization is offering does not come for free. For the case of gate virtualization, each virtualized circuit requires the execution of six instantiations of the circuit to be faithfully represented, resulting in a complexity of $O(6^k)$. An instantiation is a variation of the circuit that has specific single-qubit gates in place of the virtualized gate, that is meant to represent the respective degree-of-freedom of quantum information relayed by the original two-qubit gate. Other techniques, such as wire cutting [2], require only four instantiations to be run; a complexity of $O(4^k)$, but provide a smaller reduction in quantum resources. It has been demonstrated that, for specific cases, fewer instantiations can be run [8], while still faithfully representing the original circuit. However, those techniques are still not introduced into QVM.

2.2 Transpilation of Quantum Circuits and Qubit Routing

Before a circuit can be sent to a quantum backend provided by IBM through its Qiskit framework, it needs to be transpiled against that specific backend [9]. During the transpilation process, passes of different procedures go through the circuit, either modifying or analyzing it for the subsequent passes.

An example of such passes is the translation pass, which breaks the input circuits gates into the native operations of the specified backend, or an approximation thereof if the original operation is not directly supported by the backend. A standard subsequent step would then to optimize the circuit by simplifying operations, or removing them from the circuit if their effect is counteracted by an equivalent operation later on. At this stage, it would be reasonable to start a routing pass, which assigns the logical qubits to physical ones on the quantum chip.

Qubit Routing [10] is the procedure of assigning the logical qubits of the quantum circuit to the physical qubits present on the chip, while inserting qubit swapping operations into the circuit that bring two qubits adjacent to each other, such that a two-qubit operation can be applied to them. A single swap operation is typically broken down to three consecutive CNOT two-qubit operations, which introduces more noise to the execution of the circuit. Therefore, the routing algorithm has to introduce as few swap operations as possible. Qiskit provides several routing passes to choose from, with increasing levels of complexity, which range from the minimal-effort `BasicSwap` pass, to the Binary Integer Programming-based `BIPMapping` pass.

[1] https://quantumai.google/cirq
[2] https://qiskit.org/

2.3 SuperMarQ Benchmarking Suite

Benchmarking suites for NISQ quantum computers typically suffer from being too specialized, superficiality of the benchmarking tasks, and unscalable design. The SuperMarQ suite [5] attempts to address this issue by offering a set of architecture-agnostic application-level tasks that are applicable to any number of qubits. The tasks provided by the suite range from fundamental quantum computing tasks that examine the basic performance of the system and Quantum Error Correction routines to applications in quantum chemistry and simulation. Namely, the 8 benchmarks provided by SuperMarQ are:

- **GHZ-state Preparation** This benchmark tests the ability of the system to create a maximally-entangled set of qubits.
- **Mermin-Bell experiment** This benchmark quantifies the “quantumness" of the system’s performance, and its ability to achieve purely quantum states.
- **Phase Code and Bit Code routines** These two separate benchmarks test the ability of the system to apply the two error-correcting routines to counteract the errors inherent in it.
- **Quantum Approximate Optimization Algorithm (QAOA) for MaxCut** The QAOA algorithm is the current standard for solving optimization problems on quantum computers, and so serves as a good proxy for the application domain of optimization. The suite provides two variants of the circuit, using two different circuits to solve the MaxCut problem. Since QAOA is a variational algorithm, the circuit is first optimized through classical simulation, before the circuit is run on the quantum hardware with the converged parameters. The performance of the system is then evaluated based on how much it diverges from the correct solution due to its operational errors.
- **The Variational Quantum Eigensolver (VQE)** VQE is a commonly used algorithm for quantum chemistry that is able to find the smallest eigenvalue of a given problem, typically a hamiltonian. Similar to the QAOA benchmarks, the circuit is variational, and so its optimal parameters are also first retrieved through classical simulation, and the quality of the system is measured by the amount of divergence from the classically-obtained best solution.
- **Hamiltonian Simulation** This benchmark is critical since Hamilitonian Simulation is one of the most unique applications of quantum computing. Again, this benchmark quantifies the quality of the system by comparing its output to a classically obtained minimum.

3 Contributions to QVM

3.1 Effect of Entanglement on Virtualization Effectiveness

Multi-qubit gates could be used to entangle two or more qubits, to varying degrees. During the project, we were interested in investigating whether the amount of entanglement could affect the effectiveness of virtualization. We required a simple empirical proof that entanglement did not require any special consideration in our work, and that the effectiveness of the virtualization is agnostic to its presence.

The experiment consisted of a simple two-qubit circuit using two parameterized single-qubit gates (RY gate followed by RX gate) to control the amount of entanglement between the two qubits. Since the existing QVM implementation uses Qiskit as the main quantum framework, the circuit was implemented in Qiskit as well. The circuit is shown in Figure 2.

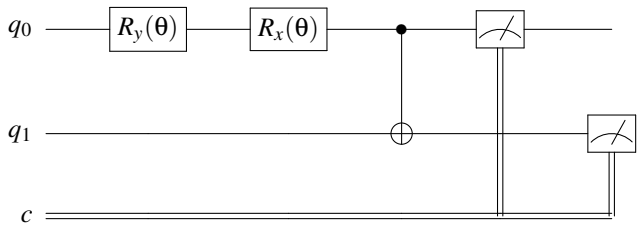

Figure 2: Qubit-Entangling Circuit

Entanglement of Formation [11] is used as the metric for quantifying the entanglement between the two qubits. It ranges from 0 for uncorrelated qubits to 1 for maximally entangled qubits. This spectrum of entanglement can be covered by tuning the parameters of the parameterized gates from 0 to $\frac{\pi}{2}$. In this experiment, a sweep of the parameters with steps of $\frac{\pi}{50}$, has been used, resulting in a total of 26 readings.

To measure any effects of entanglement on the virtualization, state fidelity [12] of the virtualized circuit is computed against the resulting state of the original circuit without any virtualization.

For executing the circuits, the AerSimulator backend from Qiskit (v0.22) has been used, set to the default settings. The metrics have been computed using the `quantum_info` module from Qiskit. In total, two experiments have been run; where for one all the required instantiations of the virtualized circuit are executed and used for the aggregation, and for the other, only two instantiations of the six has been run.

The results of this investigation are illustrated in Figure 3. The vertical axis represents both the entanglement of formation and the state fidelity values. The blue line illustrates the increase in the amount of entanglement as the parameter of the RX and the RY gates is increased from 0 to $\frac{\pi}{2}$. The red

and the green lines illustrate the change, or rather the lack thereof, of fidelity as the entanglement increases. It is clear from the graph that the fidelity is unaffected by the entanglement, regardless of the number of circuit instantiations that have been executed. In conclusion, no special handling of entangled qubits of any degree needed to be employed.

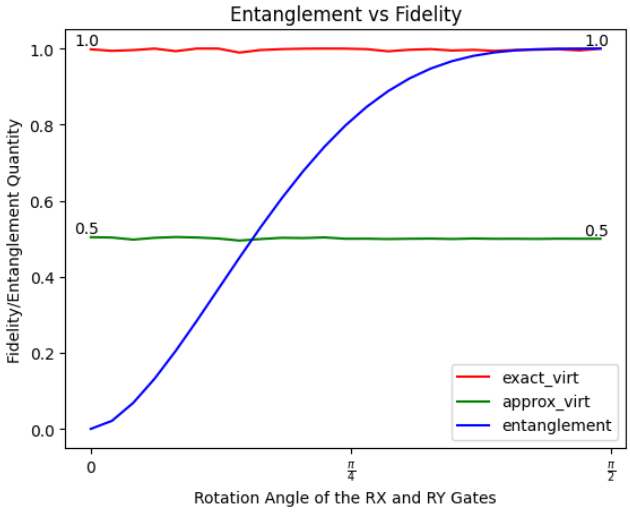

Figure 3: Effect of entanglement on fidelity of the final state of the virtualized circuit. It can be seen that the fidelity remains unaffected no matter the amount of entanglement.

3.2 The Virtual Qubit Router

Although scalability has been the main discussion point when introducing circuit cutting in the previous section, it is not the sole benefit that could be begot through this technique. Circuit cutting techniques targeting gates instead of wires, such as the one introduced in [3], could be exploited to remove the most noise-inducing two-qubit gates present in a circuit that has been assigned a quantum chip. This is very beneficial given how two-qubit gates could result in the introduction of Swap operations to be applied, building up more noise during execution. And so, a new component is introduced to QVM, denoted the Virtual Qubit Router, that does just that.

The virtual qubit routing routine (Figure 4) is a novel routing technique that aims at the removal of two-qubit operations on distant qubits altogether, obviating the need to insert Swap operations.

As mentioned before, gate virtualizations incurrs an exponential overhead. And so one still needs to be as conservative as possible when virtualizing gates, unless the user specifies a different threshold in their submitted job, as I shall clarify further in section 4.

The virtual router utilizes heuristic that attempts to provide an improvement over the `BasicSwap` pass from Qiskit. To

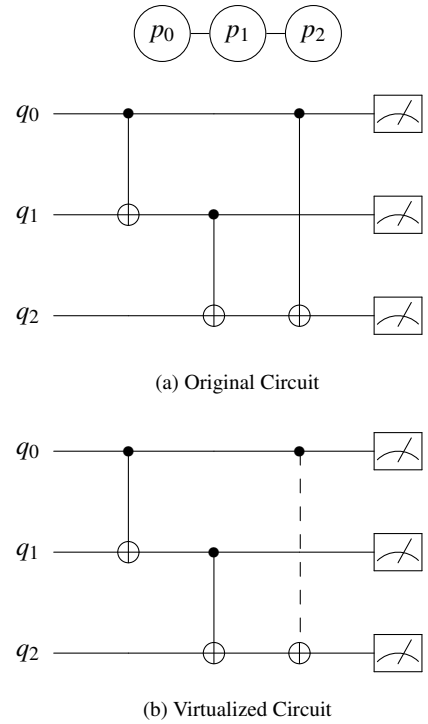

Figure 4: An example of virtual routing for a chip with a linear qubit topology shown at the top

emphasize the added logic, the original logic of the algorithm is first clarified:

1. A trivial allocation of qubits is computed. A trivial allocation is where a logical qubit is assigned a physical qubit sharing the same index.
2. For each layer in the quantum circuit, two-qubit operations are analyzed, checking whether the two respective qubits of the operation are not adjacent.
3. If it is the case that the qubits are not adjacent, a shortest undirected path between the two qubits on the connectivity graph of of the chip is computed, and a chain of Swap operations is added along it.
4. Once all layers are analyzed, the routing terminates.

In the example from Figure 4, the trivial layout would map qubit q_0 to p_0, qubit q_1 to p_1, and qubit q_2 to p_2.

Introducing virtualization to the pass is simple. Instead of adding Swap operations at the detection of distant qubits, we simply virtualize the concerned two-qubit operation. However, virtualizing all such operations can lead to a detrimental overhead. Therefore, a more guided approach would be to allow the user to configure how much intra-chip virtualization they are permitting. For example, the user may only allow two-qubit operations on qubit pairs a minimum amount of distance apart to be virtualized, and so avoiding the introduction

of any overhead. We also consider the threshold that is going to be mentioned in section 4.1, where we will present a parameter a user can override to specify the maximum number of virtualizations they are allowing. In such case, we give a higher priority to the operations applied to the qubits furthest apart, virtualizing every operation we meet until we hit the user's specified limit. Once the limit is reached, the original `BasicSwap` pass proceeds as usual, adding the necessary Swap operations.

The modified pass, while very simple, is not without its drawbacks. The first drawback manifests itself in the way the pass can be configured. As mentioned in the previous paragraph, the user can provide a maximum count of allowed virtualizations. This could result in an undesirable situation if some pair, such as qubits 0 and 3 in Figure 5, has 3 operations among them but the system is only allowed 2 more virtualizations. In such case, the user will end up with 36 circuits to run, and still end up with two Swap operations to bring the distant qubits together. The second drawback of the modification is its ineffectiveness against densely connected circuits. If, for instance, a distant qubit pair has 10 operations among them, but only 5 operations in total with their adjacent qubits, it would be roughly more beneficial to swap these qubits closer to each other, and then virtualize those 5 operations instead, if they happened to be pushed further from their original neighbors. It should be noted that this is mainly due to the Step 1 of the `BasicSwap` pass, which can result in an initial inefficient allocation of qubits. Integrating our approach of virtualize-not-swap into the more advanced passes present in Qiskit is bound to mitigate the effect of this issue, as I shall discuss as future work in section 5.

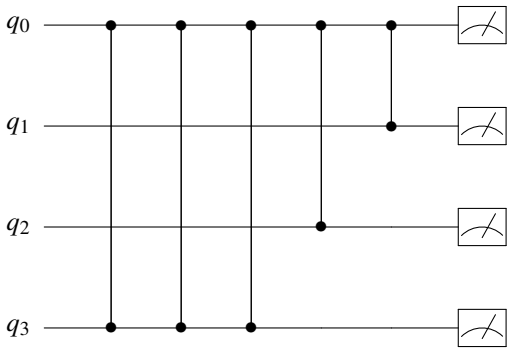

Figure 5: A pathological example of a circuit that could lead to undesired behavior by the virtual qubit router if misconfigured

To demonstrate the effect of virtualization on the ultimate number of Swap operations present in the circuit after transpilation, a simple experiment was run, where a densely connected circuit of 7 qubits (shown in Figure 6) is ran through three different routing passes: the original `BasicSwap`, the modified `BasicSwap`, and `SabreSwap`, which is the default routing pass used by Qiskit's `transpile` function and is based on the SABRE heuristic [13], a stochastic bidirectional search heuristic. For the modified algorithm, a maximum of 3 virtualizations was allowed. The circuit was transpiled against IBM's Nairobi chip's configuration through the `FakeNairobi` class from Qiskit, which provides access to the metadata of the chip without needed to establish a connection to the cloud; which is sufficient for the experiment's purposes. The qubit connectivity graph of the chip can be seen in Figure 7. To compare the performance of the three passes, the number of CNOTs present in the transpiled circuit is used as a proxy for the number of Swap operations that were present before their decomposition to the backend's native operations. This does not interfere with the count of unrelated CNOT operations that may have been initially present in the circuit, as this number is simply an independent constant that is shared between the three experiments. The results of the three comparisons are shown in Table 1.

From the results shown in the table, it could be concluded that the modification can result in a significant decrease (about %48) in the number of CNOTs after a circuit is transpiled. Although the number of CNOTs is still higher than that of the `SabreSwap` pass, it can be argued that the modified algorithm provides a much smaller transpilation overhead when compared to the expensive runtime of `SabreSwap`, which utilizes several threads for the different seeds used to run the different search runs.

`BasicSwap`	Modified `BasicSwap`	`SabreSwap`
133	69	49

Table 1: Post-transpilation number of CNOTs

4 Contributions to QStack

4.1 Job Specification

The main design philosophy for the user's job configuration is focused around two main degrees of freedom. The first revolves around giving the user the ability to set certain thresholds and limits for the different components of the system, either lying in QVM or QOS. The second revolves around giving the user the ability to override the different policies and default functionalities of the system. The latter is important to facilitate experimentations with modifications to the system through a standard API for the different components.

With the fundamental philosophy described, I shall now explain the first realization of this philosophy, currently residing in the code (example configuration in Listing 1). Configuration parameters serving the same degree of freedom are grouped together. Apart from the user's circuit, a job configuration can also include:

- First DoF Parameters (Thresholds and Constraints):

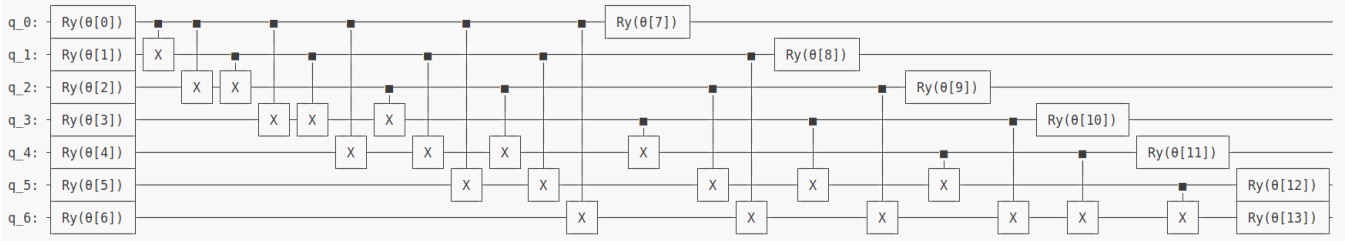

Figure 6: Circuit Used for the Virtual Routing Experiment

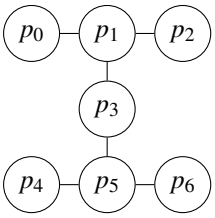

Figure 7: Qubit connectivity graph of the Nairobi chip

  - **Virtualization Technique** This parameter specifies whether the default exact virtualization should be used, or rather the cheaper approximation virtualization. It can also be set to “None", disabling virtualization altogether.
  - **Maximum Number of Virtualizations** As the name suggests, this parameter specifies the maximum number of virtualizations a user would tolerate.
  - **Permission to Use Simulators** If a user is not limiting themselves to running their programs on real quantum hardware, they can give the system permission to use a simulator, if available. This is, however, disabled by default.
  - **Hardware Error Constraints** The user can specify both soft and hard requirements for the execution of their jobs by specifying thresholds for the different calibration parameters of a quantum chip, such as T1 and T2 times, gate errors, or readout error. A boolean flag attached to the threshold’s value determines whether a constraint is hard.

- Second DoF Parameters (Overriding Policies):
  - **Circuit Cutting Policy** This parameter specifies the manner with which the two-qubit operations will be prioritized during the virtualization procedure. While the two already-implemented policies can be chosen through this parameter, the user can provide their own callable function as long as it follows the expected method signature.
  - **Gate Equivalences** As mentioned before, a virtualized gate can be reproduced by running a specific number of equivalent single-gate operations. This parameter allows users to override the gate equivalences that are already implemented in the code, or specify new equivalences for two-qubit operations for which no equivalence is implemented.

```
user_job = UserConfiguration(
    circuit=user_circuit,
    virtualization_technique="Exact",
    maximum_virtualizations=2,
    hardware_constraints={
        "CNOTError": (0.01, True),
        "ReadoutError": (0.025, False)
    }
)
```

Listing 1: Sample UserConfiguration object

4.2 System Evaluation Benchmarks

The original implementation of the SuperMarQ benchmarks [14] is in Cirq, which is incompatible with our Qiskit-based code, and therefore they had to be adapted. While sharing some similarities in the way a quantum gate can be applied, Qiskit and Cirq use different abstractions to represent their circuits. An example is shown in Listing 2. In addition, the two frameworks use different default Endianness, and this fact has been a root to several errors during the migration.

```
num_qubits = 4

# Cirq
qubits = cirq.LineQubit.range(num_qubits)
circuit = cirq.Circuit()
circuit.append(cirq.X(qubits[0]))
circuit.append(
    cirq.measure(
        *qubits,
        key="meas_all"
    )
```

```
)

# Qiskit
circuit = qiskit
            .circuit
            .QuantumCircuit(num_qubits)
circuit.x(0)
circuit.measure_all()
```

Listing 2: A comparison between Cirq and Qiskit abstractions of quantum circuits for a single-qubit circuit with a Pauli-X gate and measurement

5 Conclusion and Future Work

During this guided research, I contributed to QStack, a novel operating system stack for scalable quantum computing that incorporates circuit cutting and backend-aware scheduling, which gives it a competitive edge over the existing systems.

I initially contributed to the general code of QStack, implementing the initial version of the `UserConfiguration` class, which defined the freedoms the user had in using our system, and their ability in extending the system by incorporating their own virtualization functionalities, customizing the system as they see fit. I also adapted the SuperMarQ benchmark suite to Qiskit such that it is useful to us as the system continues to evolve.

Afterwards, I specialized in the QVM subcomponent of the system, running an experiment to investigate a hypothesis we had about the relationship between entanglement quantity in a circuit and its effect on gate virtualization's performance. Later on, I implemented an extension of Qiskit's `BasicSwap` routing pass that considers virtualization an option when faced with a two-qubit operation operating on distant qubits. A simple empirical experiment on a densely-connected circuit proved that the modification was effective in reducing the number of Swap operations during the routing procedure, but still not as effecting as Qiskit's best general performer, the `SabreSwap` pass.

There is a lot of room for improvement and exploration that could set about a more flexible system architecture that can be applied to more domains of quantum computing. For example, circuit cutting could be included in QVM as a second virtualization paradigm that a user can choose from. A hybrid paradigm incorporating both circuit cutting and gate virtualization can also be brought about; one that combines the best of both worlds.

For the virtual qubit router, gate virtualization can be augmented into the other routing passes present in Qiskit. For instance, gate virtualization can be incorporated into the Binary Integer Programming formulation of the `BIPMapping`, or include as an action in the search policy of the `LookaheadSwap` or `SabreSwap`. Ultimately, a novel routing pass should be able to achieve a balance between Swap operations and virtualizations.

References

[1] IBM Quantum Roadmap to build quantum-centric supercomputers. https://research.ibm.com/blog/ibm-quantum-roadmap-2025, Aug 2022.

[2] Wei Tang, Teague Tomesh, Martin Suchara, Jeffrey Larson, and Margaret Martonosi. Cutqc: Using small quantum computers for large quantum circuit evaluations. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 473–486, New York, NY, USA, 2021. Association for Computing Machinery.

[3] Kosuke Mitarai and Keisuke Fujii. Constructing a virtual two-qubit gate by sampling single-qubit operations. *New Journal of Physics*, 23(2):023021, 2021.

[4] X. Fu, L. Riesebos, L. Lao, C. G. Almudever, F. Sebastiano, R. Versluis, E. Charbon, and K. Bertels. A heterogeneous quantum computer architecture. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '16, page 323–330, New York, NY, USA, 2016. Association for Computing Machinery.

[5] T. Tomesh, P. Gokhale, V. Omole, G. Ravi, K. N. Smith, J. Viszlai, X. Wu, N. Hardavellas, M. R. Martonosi, and F. T. Chong. Supermarq: A scalable quantum benchmark suite. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 587–603, Los Alamitos, CA, USA, apr 2022. IEEE Computer Society.

[6] Nathaniel Tornow. Dqs: A framework for efficient distributed simulation of large quantum circuits. 2022.

[7] Amazon Braket supported devices. https://docs.aws.amazon.com/braket/latest/developerguide/braket-devices.html#braket-simulator-sv1, 2023.

[8] Daniel T. Chen, Ethan H. Hansen, Xinpeng Li, Vinooth Kulkarni, Vipin Chaudhary, Bin Ren, Qiang Guan, Sanmukh Kuppannagari, Ji Liu, and Shuai Xu. Efficient quantum circuit cutting by neglecting basis elements, 2023.

[9] Qiskit Transpiler API Reference. https://qiskit.org/documentation/apidoc/transpiler.html, 2023.

[10] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. On the qubit routing problem. 2019.

[11] William K. Wootters. Entanglement of formation of an arbitrary state of two qubits. *Phys. Rev. Lett.*, 80:2245–2248, Mar 1998.

[12] Richard Jozsa. Fidelity for mixed quantum states. *Journal of Modern Optics*, 41(12):2315–2323, 1994.

[13] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices, 2019.

[14] SuperMarq's Code. https://github.com/SupertechLabs/client-superstaq/tree/main/supermarq-benchmarks, May 2023.