QUANTUMOS 2024

November 3 - Austin, Texas

NSF WORKSHOP ON QUANTUM OPERATING SYSTEMS AND REAL-TIME CONTROL

National Science Foundation Award #2435033

Speakers

Harry Buhrman (Amsterdam)

Laura Caune (Riverlane)

Nicolas Delfosse (IonQ)

Florian Huber (QuEra)

Costin Iancu (LBNL)

Christina Lee (PennyLane)

Diego Riste (IBM)

Rob Schoelkopf (Yale)

Ruben Verresen (UChicago)

Bart van der Vecht (Delft)

Xiaodi Wu (UMD)

Hengyun -Harry- Zhou (Harvard)

Organizers

Yongshan Ding (Co-Chair), Yale

Yunong Shi (Co-Chair), AWS

Zheng Zhang (Co-Chair), Rutgers

Steve Flammia, Virginia Tech

Steven Girvin, Yale

Blake Johnson, IBM

About the Workshop

As quantum hardware scales up and becomes increasingly heterogeneous and distributed, a quantum OS will be responsible for executing error-correcting kernels (e.g., decoders), allocating systems resources (e.g. entanglement, magic states, etc), managing shared quantum memory (e.g., storage, and qRAM), and scheduling batch/concurrent programs and applications. Quantum OS is the essential tool that guarantees to sustain precision control of large quantum systems and manage quantum resources for practical QC applications. This workshop aims to explore how the emerging quantum systems enable and expand novel Computer Systems Research (CSR) opportunities and promote cross-disciplinary collaborations.

QuantumInstitute.yale.edu/QuantumOS

NSF WORKSHOP ON QUANTUM OPERATING SYSTEMS AND REAL-TIME CONTROL

Agenda

8 am - Check in - Breakfast [Salon H]

8 am - **Welcome and Introduction by the organizers** [Salon J]

8:40 am - **Keynote 1 - Hardware/Architecture: Rob Schoelkopf (Yale)**
Post-NISQ Quantum Computing: Real-time Control and Error Detection with Superconducting Qubits
9:10 am - **Talk 1: Florian Huber (QuEra)**
Control System Stack for Neutral Atom Quantum Computing
9:40 am - **Talk 2: Ruben Verresen (UChicago)**
Quantum States of Matter from Adaptive Circuits: Theory and Experiment
10:10 am - **Talk 3: Xiaodi Wu (UMD)**
RISC-Q: A Generator for Real-time Quantum Control System-on-Chip (SoCs) compatible with RISC-V

10:40 am - Coffee

11 am - **Talk 4: Costin Iancu (LBNL)**
Resource Management to Support Design Automation for Quantum Computing
11:30 am - **Talk 5: Christina Lee (PennyLane)**
PennyLane: A full-stack software ecosystem for quantum computing research

12 pm - Lunch [Salon H]

1:30 pm - **Keynote 2 - Algorithms/Applications: Harry Buhrman (Amsterdam)** [Salon J]
Parallelizing quantum circuits: trading time for space
2 pm - **Talk 6: Nicolas Delfosse (IonQ)**
Low-cost noise reduction for Clifford circuits
2:30 pm - **Talk 7: Diego Riste (IBM)**
Combining superconducting QPUs via real-time classical communication
3 pm - **Talk 8: Laura Caune (Riverlane)**
Quantum error correction experiments decoded in real time and with low-latency response on a superconducting quantum computer

3:30 pm - Coffee

4:00 pm - **Talk 9: Bart van der Vecht (Delft)**
Design and demonstration of an operating system for executing applications on quantum network nodes
4:30 pm - **Talk 10: Hengyun -Harry- Zhou (Harvard)**
Fast quantum interconnects via constant-rate entanglement distillation

5 pm - **Parallel 1: Poster Session** (for accepted submissions)
5 pm - **Parallel 2: Breakout Discussion and Q&A** (for invited participants)

6 pm - SOSP Conference Reception [Salon H]

Accepted Abstracts

By order of submissions

Book of abstracts available online at
QuantumInstitute.yale.edu/QuantumOS

QuReplica: quantum state "replication" for DQC and its runtime scheduling framework
Yuhang Gan, Runzhou Tao, Ruilin Zhou, and Chen Qian
University of California at Santa Cruz, University of Maryland

Design and demonstration of an operating system for executing applications on quantum network nodes
Bart van der Vecht, Mariagrazia Iuliano, Carlo Delle Donne, and Stephanie Wehner
QuTech, Kavli Institute of Nanoscience, Delft University of Technology

Generation of long-range entanglement enhanced by error detection
Haoran Liao, Gavin Hartnett, Ashish Kakkar, Pranav Mundada, Michael Biercuk, and Yuval Baum
Q-CTRL, Los Angeles & Sydney Australia

Quantum Noise Effects on QAOA for Reconfigurable Microgrid Networks
Betis Baheri, Yan Li, Qiang Guan, and Wei Xu
Kent State University, The Pennsylvania State University, Brookhaven National Laboratory

Quantum error correction experiments decoded in real time and with low-latency response on a superconducting quantum computer
Laura Caune, Luka Skoric, Nick Blunt, Archibald Ruban, Jimmy McDaniel, Joseph A. Valery, Andrew D. Patterson, Alexander V. Gramolin, Joonas Majaniemi, Kenton M. Barnes, Tomasz Bialas, Okan Bugdaycı, Ophelia Crawford, György P. Gehér, Hari Krovi, Elisha Matekole, Canberk Topal, Stefano Poletto, Michael Bryant, Kalan Snyder, Neil I. Gillespie, Glenn Jones, Kauser Johar, Earl Campbell, and Alexander D. Hill
Riverlane

Protocols and Applications of Quantum Stack Memory
Leonard Li, Lingjun Xiong, and Yuan Liu
North Carolina State University

Fast quantum interconnects via constant-rate entanglement distillation
Hengyun Zhou, Christopher Pattison, Gefen Baranes, Pablo Bonilla Ataides, and Mikhail Lukin
QuEra Computing Inc., Harvard University, MIT, Caltech

QuantumOS Support for Quantum Trusted Execution Environments
Theodoros Trochatos and Jakub Szefer
Yale University

Design and Implementation of the Quantum Cloud Simulation Framework
Waylon Luo, Betis Baheri, Bo Fang, and Qiang Guan
Kent State University, Pacific Northwest National Laboratory

LEGO: QEC Decoding System Architecture for Dynamic Circuits
Yue Wu, Namitha Liyanage, and Lin Zhong
Yale University

A Case for OS-Managed Resource Pools in Fault-Tolerant Quantum Computers
Suhas Vittal and Moinuddin Qureshi
Georgia Institute of Technology

A Scalable Quantum Circuit Knitting Framework via Parallel and Hardware-Efficient Circuit Cutting
Xiangyu Ren, Mengyu Zhang, and Antonio Barbalace
The University of Edinburgh UK, Tencent Quantum Lab Shenzhen China

Understanding Real Time Decoding for Photonic Quantum Computers
Avinash Kumar, Eneet Kaur, and Poulami Das
The University of Texas at Austin, CISCO Research

Pauli Check Sandwiching for Quantum Characterization and Error Mitigation during Runtime
Joshua Gao, Ji Liu, Alvin Gonzales, Zain Saleem, Nikos Hardavellas, and Kaitlin Smith
Virginia Tech, Argonne National Laboratory, Northwestern University

Quantum Tape and Stack Data Structures
Ulrik de Muelenaere and Peter M. Kogge
University of Notre Dame

A Formalization of Measurement Commuting Unitaries
Ulrik de Muelenaere, Sinan Pehlivanoglu, Amr Sabry, and Peter M. Kogge
University of Notre Dame, Indiana University Bloomington

Characterizing Equivalences Between Shallow Quantum Circuit Models
Ben Foxman and Akshat Yaparla
Yale University, Columbia University

Quantum Control of an Oscillator with a Kerr-cat Qubit
A. Ding, B. Brock, A. Eickbusc A. Koottandavida, N. Frattini, R. Cortiñas, V. Joshi, S. de Graaf, B. Chapman, S. Ganjam, L. Frunzio, R. Schoelkopf, M. Devoret
Yale University

Architecting a quantum operating system: microkernel, message passing and supercomputing
Alexandru Paler
Aalto University

Meetings of possible interest to the QuantumOS community

HELGOLAND 2025

100 Years of Quantum Mechanics

June 9–14, 2025

Helgoland Island – Germany

Program & Organization Committees

Časlav Brukner – IQOQI Vienna
Steve Girvin – Yale Quantum Institute
Jack Harris – Yale Quantum Institute
Florian Marquardt – Max Planck Institute for the Science of Light
Florian Carle – Yale Quantum Institute
Katharina Kißner – Max Planck Institute for the Science of Light
Gesine Murphy – Max Planck Institute for the Science of Light

helgoland2025.org

NSF NQVL ERASE

Erasure Qubits and Dynamic Circuits for Quantum Advantage

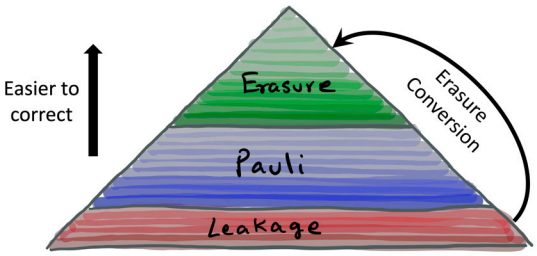
Town Halls

February, April, and August 2025

New Haven, CT – United State

The ERASE project will establish a National Quantum Virtual Laboratory (NQVL) to develop an innovative quantum computing platform based on dual-resonator 'erasure flag' qubits that enhance the error detection and correction required to achieve practical quantum computing.

ERASE.yale.edu (coming soon)



QEC 2025

7th International Conference on Quantum Error Correction

August 11–15, 2025

New Haven, CT – United State

Program & Organization Committees

Stephen Bartlett – The University of Sydney
Kenneth Brown – Duke University
Earl Campbell – University of Sheffield & Riverlane
Steve Flammia – Virginia Tech
Steve Girvin – Yale Quantum Institute
Amy Badner – Yale Quantum Institute
Florian Carle – Yale Quantum Institute
Aleksander Kubica – Yale Quantum Institute
Shruti Puri – Yale Quantum Institute

quantuminstitute.yale.edu/qec25

QuReplica: quantum state "replication" for DQC and its runtime scheduling framework

Yuhang Gan*
University of California, Santa Cruz
Santa Cruz, CA, USA
ygan11@ucsc.edu

Runzhou Tao*
University of Maryland, College Park
College Park, MD, USA
runzhou.tao@columbia.edu

Ruilin Zhou
University of California, Santa Cruz
Santa Cruz, CA, USA
rzhou39@ucsc.edu

Chen Qian
University of California, Santa Cruz
Santa Cruz, CA, USA
cqian12@ucsc.edu

1 Introduction

Scalability is one of the most important factors in determining the success of quantum computing. It is commonly believed that a usable quantum computing system requires millions of computing qubits to perform practical computational tasks (e.g., factoring [12]). Distributed computing, a paradigm that has been proven to be highly successful on classical computing systems over the past two decades, has become one of the most promising solutions for scaling up quantum computing systems and has attracted increasing attention [6, 16]. By forming a Distributed Quantum Computing (DQC) system with multiple quantum processing units(QPU) of slightly weaker computational power, it is possible to greatly expand the scale of the whole system.

However, similar to classical distributed computing, data transmission, i.e., the communication between nodes, is a critical bottleneck in DQC as well [1, 14]. It is common in distributed systems that a piece of data needs to be shared among different nodes and may be modified by any node at any time [16, 17]. Hence, the protocols and transmission methods for data transfer need to be carefully designed during the computation process to ensure the correctness of the data while minimizing the overhead of system communication [7].

In classical distributed computing, systems can set up replicas of the data in different computing nodes to reduce the data transmission process. The replica has demonstrated its powerful ability in failure recovery, load balancing, and enhancing locality to accelerate the computation [6, 8, 11, 16]. In quantum computing, due to the no-cloning theorem, we can not copy quantum data. Instead, entangled quantum states can be used across different nodes as a communication resource for a particular qubit's data transmission. By consuming entanglement pairs, besides teleportation, the generalized GHZ state establishment via cat-entangler [15] between two or more qubits can be achieved to *share* the state information of one qubit on one QPU with other QPUs for performing non-local multi-qubit gates.

Multiple works have been proposed to use quantum entanglements to share data between multiple QPUs in DQC [5, 13–15]. However, they have several drawbacks that limit their scalability. First, most approaches focus on minimizing communication overhead by mapping quantum programs to QPUs at the compilation stage, preventing runtime adaptations and limiting flexibility. While some consider system state during mapping, static compiling-time optimization is ill-suited for the dynamic nature of quantum computing systems, leading to suboptimal performance. As quantum programs grow in size and complexity, allocating resources based on the system's pre-run state without runtime adaption becomes impractical. Furthermore, scheduling programs statically requires all programs in a batch to run together, even though they may have different runtime and resource needs, reducing scalability.

Second, due to concerns about the consistency of state sharing of qubits across QPUs, most of the existing work employs a passive ad hoc scheduling of communication resources: state sharing by creating multi-qubits GHZ state is performed only when needed and is deactivated as soon as it is completed to ensure program correctness. However, due to the non-locality of entanglement, we do not need to always cancel all GHZ state replicas of a qubit when encountering an arbitrary gate performing on this qubit. An example is, for a GHZ state, $\alpha|00...0\rangle + \beta|11...1\rangle$, when we perform a Z gate on the first qubit, the phase of this GHZ state will "globally" change to $\alpha|00...0\rangle - \beta|11...1\rangle$, the state information is still consistent between the original qubit and its all replica qubits. Actually, this applies to all phase gates. Besides, research about using the GHZ state to share state information among different compute nodes in DQC is under-explored and is limited to the most basic 2-qubit scenarios [14].

Third, current work also lacks a discussion of multi-tenancy scenarios for distributed quantum computing systems. Existing works on multi-tenant scenarios are mostly limited to allocating physical qubits on a single QPU to different programs, named quantum multi-programming [4, 9, 10].

In this work, we propose QuReplica, a proactive data-sharing framework combining dynamic runtime scheduling and static compilation using the generalized GHZ state in multi-tenant DQC scenarios. In the static compilation stage, we annotate the program using flags to reveal potential opportunities for setting replicas of some qubits by creating a GHZ state of a qubit via cat-entangler [15] to reduce communication overhead among QPUs while ensuring computation correctness. Then, based on these annotations, the runtime scheduler dynamically creates and recycles replicas of

*Both authors contributed equally to this research.

Conference'17, July 2017, Washington, DC, USA
2024. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

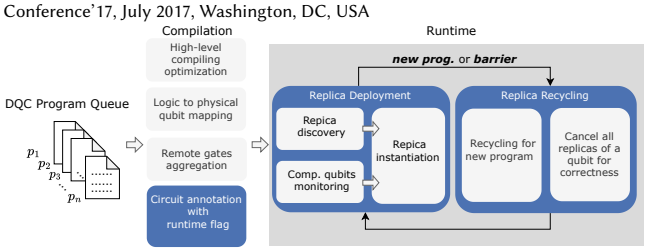


Figure 1: Overview of QuReplica workflow

qubits according to the system's real-time state during program executions. Instead of trying to complete all optimizations at the compilation stage, we first brought dynamic *runtime* resource management and scheduling into the multi-tenant DQC system, which could be a good starting point for exploring the future quantum operating system (QuOS).

2 QuReplica and its runtime scheduling

The goal of QuReplica is to allow dynamic runtime scheduling for state sharing of some qubits to reduce the overall communication overhead in multi-tenant DQC. QuReplica consists of three major components as shown in Fig. 1: 1) Circuit annotation with replica flags after normal compilation optimization and mapping steps to reveal potential opportunities of setting replicas of a qubit to reduce communication overhead by adding some annotation flags; 2) A replica deployment module that creates replica qubits when possible; And 3) A replica recycling module that cancels some or all replica qubits to release computing qubits for new joining programs to get better initial mapping or guarantee computing correctness.

In the rest of this section, we will use a simplified version of Quantum Fourier Transform(QFT) based algorithm as a running example to give a high-level overview of QuReplica. Quantum Fourier Transform (QFT) is the basic functional block of many important quantum algorithms [2, 3]. Fig. 2 shows a simplified QFT-base algorithm circuit. It is evident that q_m is the control qubit for many two-qubit gates involving several other qubits (e.g., q_0, q_1, q_2, q_3) and may be affected by some phase gates. In a DQC setup, these qubits will be distributed across different QPUs. Fig. 3 shows a possible condition that runs the circuit in Fig. 2 on a DQC system with three QPUs (A, B, C) in line topology. q_0, q_1, q_2, q_3 and q_m are deployed on QPU A, B, C separately and q_{c0}, q_{c1}, q_{c2} are communication qubits belongs to each QPU node.

At the compilation stage, after obtaining the initial qubit mapping, we identify all remote gates and need to determine which qubits should have replicas on other QPUs. When annotating the DQC program, two main questions arise: 1) For which qubits to set up replicas, on which QPUs, and when; and 2) When to recycle the replica qubits to maintain program correctness and system performance, in what quantities. Replicas are needed for all remote gates, except those handled by teleportation, with the key challenge being when to set them up. The worst case is an ad-hoc setup, which eliminates dynamic runtime scheduling and relies on static compilation results, introducing fully remote gate latency. Setting up in advance can hide this overhead during previous gate execution. During annotation, we divide the circuit of each qubit into segments using certain gates (e.g., H, R_x, R_y) that will break the GHZ state as segmentation points. Each segment has a earliest setup time and a latest recycling time for setting replica qubits, marked by a "barrier" flag at the end.

For the q_m shown in Fig. 3, since it can only be subjected to phase gates during a period of time, which has no effect on the GHZ state, when adding the annotation, we set the earliest replica set up time of this segmentation of q_m to be "after the first H-gate", and the latest recycling time to be "before the second H-gate" " and insert a "barrier" flag before the second H gate and add corresponding cat-disentangler circuit to cancel the replicas.

At runtime, the replica deployment module maintains one priority queue on each QPU. Based on the priority of each replica fragment (determined by the gain and urgency of the replica) and the availability of computing qubits on QPUs, it utilizes the available computing qubits to set the replica qubits required by a remote gate when the communication qubits are idle. This asynchronous establishment method can fully utilize the communication qubits and reduce the competition between different remote gates for communication resources, compared with using the remote gate to establish the replica instantly only when it is needed.

In Fig. 3, since the qubits that are to use the q_m state information are distributed in QPU A and B, we need to use communication qubits to build the replica of q_m on A and B via cat-entangler. We first use q_{c1}, q_{c2} to share the state information of q_m to QPU B. We assume that q^r_{m1} is idle at this time, and then we build the replica qubit of q_m on QPU B by applying a swap gate. Subsequently, for QPU A, since the qubit q_x of the previously running program occupies the computing qubit of QPU A during the initial mapping, for q_0 and q_1, without the use of runtime scheduling, and based on the compilation result alone, we can only fully occupy q_{c0} to share q_m's state to complete the remote gate that blocking the other qubits from using q_{c0}. However, through the replica deployment module's monitoring of runtime computing qubits, we can find that the computing qubit occupied by q_x is released in time to be used to build q_m's replica qubit q^r_{m2} on QPU A ahead of time, asynchronously using q_{c0}, thus avoiding resource competition blocking.

The third part of the QuReplica, the replica recycling module, is responsible for unsetting the replica qubits to ensure the correctness of the program and the overall performance of the system. Both q_m as well as q^r_{m1}, q^r_{m2} pause when execution reaches a previously set barrier flag, wait for each other until synchronization is reached, and via cat-disentangler, all replica qubits are canceled before the H gate is applied to q_m. By setting the barrier to synchronize qubits and their replica qubits, we avoid the inconsistency between the original qubit q_m and the replica qubits q^r_{m1}, q^r_{m2} and

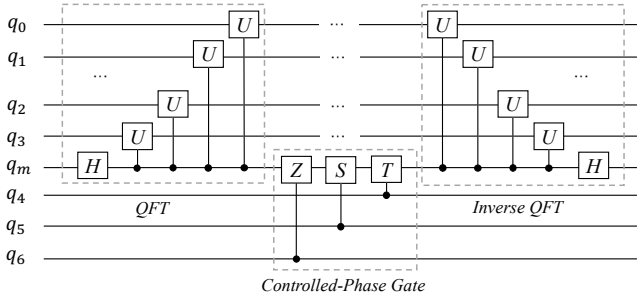


Figure 2: QFT-based algorithm example

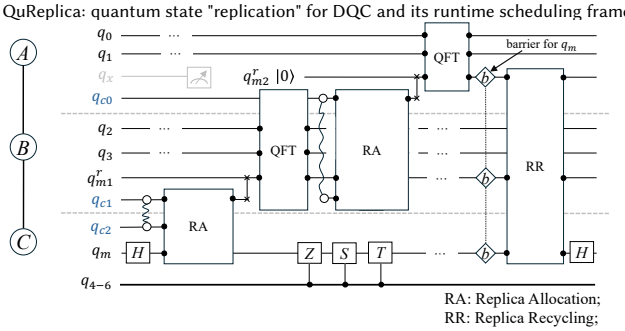


Figure 3: QFT-based algorithm example instance in DQC system. The left figure is the DQC topology with QPU A, B, and C; The right figure is a real instance of the circuit in Fig. 2 running on the DQC system. q_x is a logical qubit of another program, q^r_{m1}, q^r_{m2} are the replica qubits of q_m.

ensure the correctness of the program. Besides, if canceling some replica qubits can improve the initial mapping for a new program and the benefit outweighs maintaining them, we can actively cancel the corresponding qubits to enhance system performance.

References

[1] David Barral, F Javier Cardama, Guillermo Díaz, Daniel Faílde, Iago F Llovo, Mariamo Mussa Juane, Jorge Vázquez-Pérez, Juan Villasuso, César Piñeiro, Natalia Costas, et al. 2024. Review of distributed quantum computing. from single qpu to high performance quantum computing. *arXiv preprint arXiv:2404.01265* (2024).

[2] Francesco Bova, Avi Goldfarb, and Roger G Melko. 2021. Commercial applications of quantum computing. *EPJ quantum technology* 8, 1 (2021), 2.

[3] Don Coppersmith. 2002. An approximate Fourier transform useful in quantum factoring. *arXiv preprint quant-ph/0201067* (2002).

[4] Poulami Das, Swamit S Tannu, Prashant J Nair, and Moinuddin Qureshi. 2019. A case for multi-programming quantum computers. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 291–303.

[5] Davide Ferrari, Angela Sara Cacciapuoti, Michele Amoretti, and Marcello Caleffi. 2021. Compiler design for distributed quantum computing. *IEEE Transactions on Quantum Engineering* 2 (2021), 1–20.

[6] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*. 29–43.

[7] Leslie Lamport. 2001. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)* (2001), 51–58.

[8] Eliezer Levy and Abraham Silberschatz. 1990. Distributed file systems: Concepts and examples. *ACM Computing Surveys (CSUR)* 22, 4 (1990), 321–374.

[9] Lei Liu and Xinglei Dou. 2021. Qucloud: A new qubit mapping mechanism for multi-programming quantum computing in cloud environment. In *2021 IEEE International symposium on high-performance computer architecture (HPCA)*. IEEE, 167–178.

[10] Siyuan Niu and Aida Todri-Sanial. 2023. Enabling multi-programming mechanism for quantum computing in the NISQ era. *Quantum* 7 (2023), 925.

[11] Gang Peng. 2004. CDN: Content distribution network. *arXiv preprint cs/0411069* (2004).

[12] Peter W Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 124–134.

[13] Anbang Wu, Yufei Ding, and Ang Li. 2023. Qucomm: Optimizing collective communication for distributed quantum computing. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 479–493.

[14] 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)*. IEEE, 1027–1041.

[15] Anocha Yimsiriwattana and Samuel J Lomonaco Jr. 2004. Distributed quantum computing: A distributed Shor algorithm. In *Quantum Information and Computation II*, Vol. 5436. SPIE, 360–372.

[16] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A {Fault-Tolerant} abstraction for {In-Memory} cluster computing. In *9th USENIX symposium on networked systems design and implementation (NSDI 12)*. 15–28.

[17] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the twenty-fourth ACM symposium on operating systems principles*. 423–438.

Design and demonstration of an operating system for executing applications on quantum network nodes*

Carlo Delle Donne[1,2], Mariagrazia Iuliano[1], Bart van der Vecht[1,2], and Stephanie Wehner[1,2,†]

[1] *QuTech and Kavli Institute of Nanoscience, Delft University of Technology*
[2] *Quantum Computer Science, Department of Software Technology, Delft University of Technology*
[†] For full co-author list see full paper*

1 Introduction

The goal of future quantum networks is to enable new internet applications that are impossible to achieve using solely classical communication[1, 2, 3]. Up to now, demonstrations of quantum network applications[4, 5, 6] and functionalities[7, 8, 9, 10, 11, 12] on quantum processors have been performed in ad-hoc software that was specific to the experimental setup, programmed to perform one single task (the application experiment) directly into low-level control devices using expertise in experimental physics. Here, we report on the design and implementation of the first architecture capable of executing quantum network applications on quantum processors in platform-independent high-level software. We demonstrate the architecture's capability to execute applications in high-level software, by implementing it as a quantum network operating system - **QNodeOS** - and executing test programs including a delegated computation from a client to a server[13] on two quantum network nodes based on nitrogen-vacancy (NV) centers in diamond[14, 15]. We show how our architecture allows us to maximize the use of quantum network hardware, by multitasking different applications on a quantum network for the first time. Our architecture can be used to execute programs on any quantum processor platform corresponding to our system model, which we illustrate by demonstrating an additional driver for QNodeOS for a trapped-ion quantum network node based on a single $^{40}Ca^{+}$atom[16]. Our architecture lays the groundwork for computer science research in the domain of quantum network programming, and paves the way for the development of software that can bring quantum network technology to society.

2 Design Considerations and Challenges

Interactive Classical-Quantum Execution: Quantum network program instructions can be divided into classical blocks (including message-passing over the network) and quantum blocks (gates, measurements, remote entanglement generation) and these blocks are highly interdependent. The execution of quantum network applications hence requires a continuing interaction between the quantum and classical parts of the execution.

Different Hardware Platforms: Interfacing with different hardware platforms presents technological challenges: currently, a clear line between software and hardware has not been defined, and the low-level control of present-day quantum processor hardware has been built to conduct physics experiments. Early microarchitectures [17, 18] and operating systems [19, 20] for quantum computing do not address the execution of quantum *network* applications.

Timescales: A quantum network node must operate at vastly different timescales. For nodes separated by many kilometers, the duration of network operations is in the millisecond (ms) regime, and some applications [2] need significant local classical processing (ms). In contrast, execution time of local quantum operations is in the regime of microseconds (μs), and the low-level control (including timing synchronization between nodes to generate entanglement [21]) requires nanosecond (ns) precision.

Memory Lifetimes: Present-day quantum network nodes have short coherence times, posing a technological challenge to ensure operations are executed within the timeframe allowed by the quantum memory.

Scheduling Local and Network Operations: Heralded entanglement generation requires agreement between neighboring network nodes to trigger entanglement generation in precise time-bins [22], organized into a network schedule [23] that dictates when nodes make entanglement. It is a technological challenge to manage the interdependencies between the schedule of local operations and of networked operations, since in all current quantum node implementations [24, 25], entanglement generation cannot be performed simultaneously with local quantum operations [24, 26].

Multitasking: When executing quantum network applications, one node is typically idle while waiting for the other node before it can continue execution. A fundamental chal-

* Full paper: https://doi.org/10.48550/arXiv.2407.18306

lenge is how system utility can be increased by multitasking [27, 28], that is, allowing concurrent execution of several programs at once to make use of idle times. There is hence need for managing state and resources for multiple independent programs, including processes, quantum memory management, and entanglement requests.

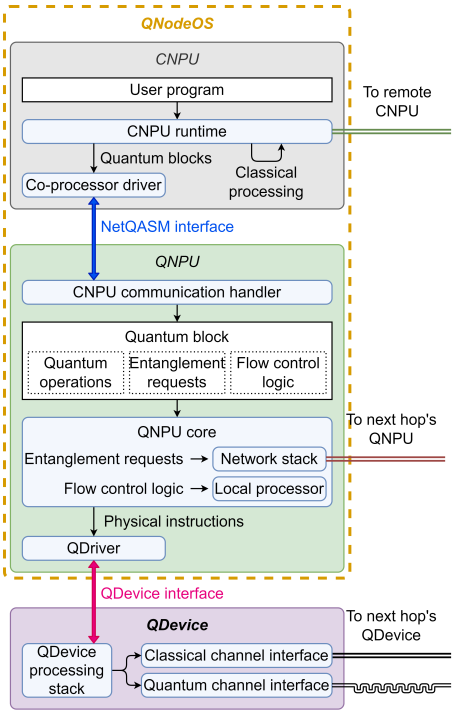


Figure 1: **QNodeOS architecture. (a)** QNodeOS consists of a Classical Network Processing Unit (CNPU) and a Quantum Network Processing Unit (QNPU, classical system). QNodeOS controls a QDevice (quantum hardware and low-level classical control).

3 Architecture

We present our QNodeOS architecture, which is logically divided into three main components (Figure 1): The CNPU for execution of classical code blocks; the QNPU for governing execution of quantum code blocks; The CNPU and QNPU together form QNodeOS and control the QDevice, which executes quantum operations (gates, measurements, entanglement generation at the physical layer [22]) on the quantum hardware. This logical division allows for realizing different timing granularities, addressing the challenge of different timescales.

We introduce a QDriver realizing a hardware abstraction layer (HAL) for any hardware corresponding to our minimal QDevice system model. The QDriver is responsible for translating quantum operations, expressed in NetQASM [29], into platform dependent (streams of) physical instructions to the underlying QDevice. We realize a QDriver for the trapped-ion system of [30, 31], and one for NV centers in diamond based on the system of [7, 24, 32].

Because of the interactive nature of programs, the architecture needs to be able to dynamically handle both classical and quantum blocks, even if not known at runtime. Therefore, our QNPU is continuously ready to receive new quantum blocks from the CNPU, and the QDevice can continuously receive and respond to physical instructions from the QNPU.

QNodeOS uses a QNPU scheduler that allows interleaving the execution of different processes directly on the QNPU without incurring delays on the timescale of the CNPU (ms), addressing the challenge of short coherence times. In our implementation, we use a priority based non-preemptive scheduler [33], due to limited quantum memory lifetimes, which make it undesirable to pre-empt and temporarily store quantum states while halting the execution. A network process, realized as a kernel process, handles entanglement requests submitted by user processes. It uses the network stack [22, 34], including a network schedule that can be determined by a time-division multiple access (TDMA) controller [23] to coordinate entanglement generation with the rest of the network. After interacting with the QDevice it eventually returns entangled qubits to user processes.

To increase utility, QNodeOS allows multiple programs to be run concurrently. Similar to classical memory management systems [35], a quantum memory management unit (QMMU) on the QNPU manages qubit allocations from processes, and translates virtual qubit addresses in quantum blocks to physical addresses in the QDevice. Entanglement generation between different pairs of processes at remote nodes are distinguished by Entanglement Request (ER) sockets, inspired by classical sockets.

4 Demonstrations

We validate our architecture by implementing QNodeOS on a two-node (client and server) setup of NV centers using one qubit per node. We show the first successful execution of an arbitrary (not preloaded) execution of a quantum network application in high-level software on quantum processors. We also validate QNodeOS's multitasking capability by the first concurrent execution of two quantum applications on a quantum network: a Delegated Quantum Computation (DQC) application, and a single-node local gate tomography (LGT) application on the client. We observe interleaved execution of DQC quantum blocks and LGT quantum blocks on the client node. We verify that interleaving different programs does not decrease the quantum result (fidelity) of the applications. We further test multitasking by scaling up the number of programs executed concurrently, up to 5 DQC and 5 LGT programs at the same time. The interleaved execution of blocks of different programs increases quantum device utilization compared to the same scenario but with multitasking disabled.

References

[1] H. J. Kimble. "The Quantum Internet". In: *Nature* 453.7198 (2008), pp. 1023–1030. DOI: 10.1038/nature07127.

[2] S. Wehner, D. Elkouss, and R. Hanson. "Quantum Internet: A Vision for the Road Ahead". In: *Science* 362.6412 (2018), pp. 1–9. DOI: 10.1126/science.aam9288.

[3] R. van Meter. *Quantum Networking*. John Wiley and Sons, Ltd, 2014. DOI: 10.1002/9781118648919.

[4] S. Barz, E. Kashefi, A. Broadbent, J. F. Fitzsimons, A. Zeilinger, and P. Walther. "Demonstration of Blind Quantum Computing". In: *Science* 335.6066 (2012), pp. 303–308. DOI: 10.1126/science.1214707.

[5] P. Drmota, D. Nadlinger, D. Main, B. Nichol, E. Ainley, D. Leichtle, A. Mantri, E. Kashefi, R. Srinivas, G. Araneda, et al. "Verifiable blind quantum computing with trapped ions and single photons". In: *Physical Review Letters* 132.15 (2024). Publisher: APS, p. 150604. DOI: 10.1103/PhysRevLett.132.150604.

[6] D. Nadlinger. "Device-independent key distribution between trapped-ion quantum network nodes". PhD thesis. University of Oxford, 2022.

[7] S. Hermans, M. Pompili, H. Beukers, S. Baier, J. Borregaard, and R. Hanson. "Qubit teleportation between non-neighbouring nodes in a quantum network". In: *Nature* 605.7911 (2022), pp. 663–668. DOI: 10.1038/s41586-022-04697-y.

[8] M. Iuliano, M.-C. Slater, A. J. Stolk, M. J. Weaver, T. Chakraborty, E. Loukiantchenko, G. C. d. Amaral, N. Alfasi, M. O. Sholkina, W. Tittel, et al. "Qubit teleportation between a memory-compatible photonic time-bin qubit and a solid-state quantum network node". In: *arXiv preprint arXiv:2403.18581* (2024). DOI: 10.48550/arXiv.2403.18581.

[9] D. Matsukevich, P. Maunz, D. Hayes, L.-M. Duan, and C. Monroe. "Quantum teleportation between distant matter qubits". In: *Science* 323.5913 (2009). Publisher: American Association for the Advancement of Science, pp. 486–489. DOI: 10.1126/science.1167209.

[10] S. Langenfeld, S. Welte, L. Hartung, S. Daiss, P. Thomas, O. Morin, E. Distante, and G. Rempe. "Quantum teleportation between remote qubit memories with only a single photon as a resource". In: *Physical Review Letters* 126.13 (2021). Publisher: APS, p. 130502. DOI: 10.1103/PhysRevLett.126.130502.

[11] W. Pfaff, B. J. Hensen, H. Bernien, S. B. van Dam, M. S. Blok, T. H. Taminiau, M. J. Tiggelman, R. N. Schouten, M. Markham, D. J. Twitchen, et al. "Unconditional quantum teleportation between distant solid-state quantum bits". In: *Science* 345.6196 (2014). Publisher: American Association for the Advancement of Science, pp. 532–535. DOI: 10.1126/science.1253512.

[12] K. S. Chou, J. Z. Blumoff, C. S. Wang, P. C. Reinhold, C. J. Axline, Y. Y. Gao, L. Frunzio, M. Devoret, L. Jiang, and R. Schoelkopf. "Deterministic teleportation of a quantum gate between two logical qubits". In: *Nature* 561.7723 (2018). Publisher: Nature Publishing Group UK London, pp. 368–373. DOI: 10.1038/s41586-018-0470-y.

[13] A. Broadbent, J. Fitzsimons, and E. Kashefi. "Universal Blind Quantum Computation". In: *FOCS*. IEEE, 2009, pp. 517–526. DOI: 10.1109/FOCS.2009.36.

[14] M. W. Doherty, N. B. Manson, P. Delaney, F. Jelezko, J. Wrachtrup, and L. C. Hollenberg. "The nitrogen-vacancy colour centre in diamond". In: *Physics Reports* 528 (2013). DOI: 10.1016/j.physrep.2013.02.001.

[15] L. Childress and R. Hanson. "Diamond NV centers for quantum computing and quantum networks". In: *MRS bulletin* 38.2 (2013), pp. 134–138. DOI: 10.1557/mrs.2013.20.

[16] D. Fioretto. "Towards a flexible source for indistinguishable photons based on trapped ions and cavities". PhD thesis. University of Innsbruck, 2020.

[17] K. Bertels, A. Sarkar, T. Hubregtsen, M. Serrao, A. A. Mouedenne, A. Yadav, A. Krol, I. Ashraf, and C. G. Almudever. "Quantum computer architecture toward full-stack quantum accelerators". In: *IEEE Transactions on Quantum Engineering* 1 (2020). Publisher: IEEE, pp. 1–17. DOI: 10.1109/TQE.2020.2981074.

[18] X. Fu, L. Riesebos, M. A. Rol, J. van Straten, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, V. Newsum, K. K. L. Loh, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. "eQASM: An Executable Quantum Instruction Set Architecture". In: *HPCA*. IEEE, 2019, pp. 224–237. DOI: 10.1109/HPCA.2019.00040.

[19] E. Giortamis, F. Romão, N. Tornow, and P. Bhatotia. "QOS: A Quantum Operating System". In: *arXiv preprint arXiv:2406.19120* (2024). DOI: 10.48550/arXiv.2406.19120.

[20] W. Kong, J. Wang, Y. Han, Y. Wu, Y. Zhang, M. Dou, Y. Fang, and G. Guo. "Origin Pilot: a Quantum Operating System for Effecient Usage of Quantum Resources". 2021. arXiv: 2105.10730. URL: https://arxiv.org/abs/2105.10730.

[21] P. C. Humphreys, N. Kalb, J. P. J. Morits, R. N. Schouten, R. F. L. Vermeulen, D. J. Twitchen, M. Markham, and R. Hanson. "Deterministic Delivery of Remote Entanglement on a Quantum Network". In: *Nature* 558.7709 (2018), pp. 268–273. DOI: 10.1038/s41586-018-0200-5.

[22] A. Dahlberg, M. Skrzypczyk, T. Coopmans, L. Wubben, F. Rozpędek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, J. de Oliveira Filho, R. Hanson, and S. Wehner. "A Link Layer Protocol for Quantum Networks". In: *SIGCOMM*. ACM, 2019, pp. 159–173. DOI: 10.1145/3341302.3342070.

[23] M. Skrzypczyk and S. Wehner. "An Architecture for Meeting Quality-of-Service Requirements in Multi-User Quantum Networks". 2021. arXiv: 2111.13124.

[24] M. Pompili, S. L. N. Hermans, S. Baier, H. K. C. Beukers, P. C. Humphreys, R. N. Schouten, R. F. L. Vermeulen, M. J. Tiggelman, L. dos Santos Martins, B. Dirkse, S. Wehner, and R. Hanson. "Realization of a Multinode Quantum Network of Remote Solid-State Qubits". In: *Science* 372.6539 (2021), pp. 259–264. DOI: 10.1126/science.abg1919.

[25] P. Drmota, D. Main, D. Nadlinger, B. Nichol, M. Weber, E. Ainley, A. Agrawal, R. Srinivas, G. Araneda, C. Ballance, et al. "Robust quantum memory in a trapped-ion quantum network node". In: *Physical Review Letters* 130.9 (2023). Publisher: APS, p. 090803. DOI: 10.1103/PhysRevLett.130.090803.

[26] V. Krutyanskiy, M. Meraner, J. Schupp, V. Krcmarsky, H. Hainzer, and B. P. Lanyon. "Light-matter entanglement over 50 km of optical fibre". In: *npj Quantum Information* 5.1 (2019). Publisher: Nature Publishing Group UK London, p. 72. DOI: 10.1038/s41534-019-0186-3.

[27] J. D. McCullough, K. H. Speierman, and F. W. Zurcher. "A design for a multiple user multiprocessing system". In: *Proceedings of the November 30–December 1, 1965, fall joint computer conference, part I*. 1965, pp. 611–617. DOI: 10.1145/1463891.1463957.

[28] J. B. Dennis. "Segmentation and the design of multiprogrammed computer systems". In: *Journal of the ACM (JACM)* 12.4 (1965). Publisher: ACM New York, NY, USA, pp. 589–602. DOI: 10.1145/321296.321310.

[29] A. Dahlberg, B. van der Vecht, C. Delle Donne, M. Skrzypczyk, I. te Raa, W. Kozlowski, and S. Wehner. "NetQASM—A Low-Level Instruction Set Architecture for Hybrid Quantum–Classical Programs in a Quantum Internet". In: *Quantum Science and Technology* 7.3 (2022), p. 035023. DOI: 10.1088/2058-9565/ac753f.

[30] M. Teller, V. Messerer, K. Schüppert, Y. Zou, D. A. Fioretto, M. Galli, P. C. Holz, J. Reichel, and T. E. Northup. "Integrating a fiber cavity into a wheel trap for strong ion–cavity coupling". In: *AVS Quantum Science* 5.1 (2023). DOI: 10.1116/5.0121534.

[31] M. Teller, D. A. Fioretto, P. C. Holz, P. Schindler, V. Messerer, K. Schüppert, Y. Zou, R. Blatt, J. Chiaverini, J. Sage, et al. "Heating of a trapped ion induced by dielectric materials". In: *Physical Review Letters* 126.23 (2021), p. 230505. DOI: 10.1103/PhysRevLett.126.230505.

[32] M. Pompili, C. Delle Donne, I. te Raa, B. van der Vecht, M. Skrzypczyk, G. M. Ferreira, L. de Kluijver, A. J. Stolk, S. L. N. Hermans, P. Pawełczak, W. Kozlowski, R. Hanson, and S. Wehner. "Experimental Demonstration of Entanglement Delivery Using a Quantum Network Stack". In: *npj Quantum Information* 8.1 (2022), p. 121. DOI: 10.1038/s41534-022-00631-2.

[33] C. L. Liu and J. W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". In: *J. ACM* 20.1 (1973), pp. 46–61. DOI: 10.1145/321738.32174.

[34] W. Kozlowski, A. Dahlberg, and S. Wehner. "Designing a Quantum Network Protocol". In: *CoNEXT*. ACM, 2020, pp. 1–16. DOI: 10.1145/3386367.3431293.

[35] J. L. Peterson and A. Silberschatz. *Operating system concepts*. Addison-Wesley Longman Publishing Co., Inc., 1985. DOI: 10.5555/3526.

Generation of long-range entanglement enhanced by error detection

Abstract submission to NSF workshop on quantum operating systems and real-time control

Haoran Liao*[1], Gavin S. Hartnett[1], Ashish Kakkar[1], Pranav S. Mundada[1], Michael J. Biercuk[1], and Yuval Baum[1]

[1] *Q-CTRL, Los Angeles, CA USA and Sydney, NSW Australia*

1 Extended abstract

Over the recent years, QEC research across the quantum computing community led to a great number of impressive demonstrations [Acharya et al., 2023, Gupta et al., 2024, Bluvstein et al., 2024, Acharya et al., 2024]. This includes validation of most of the underlying primitives of the QEC protocol, such as developing efficient encodings, the ability to repeatedly identify errors and to perform active corrections, and even the ability to deliver net improvements to logical qubit lifetime or the quality of logical operations. However, implementing QEC remains costly, and limited to small scales due to rather limited number of physical qubits available currently. While QEC is expected to deliver some net benefits, such benefits are not likely to be competitive in the near term over alternate techniques that do not require significant overhead. In light of these, primitives (sub-routines) of QEC may be borrowed and applied in low-overhead, physical levels to improve system performance in the near term. In this work, we focus on near-term implementations of two core primitives of QEC—error detection and gate teleportation.

In the first part of the paper, we demonstrate the utility of error detection by performing sparse parity checks of local stabilizers on the generation of Greenberger-Horne-Zeilinger (GHZ) states. The core idea behind this method [Mooney et al., 2021] is to utilize a small number of ancilla qubits (flags) to indicate the presence of an error. Such error detection methods leverage local symmetries among the physical qubits. While in the context of QEC, such symmetries arise naturally as part of the physical-to-logical encoding, such symmetries can also arise on the physical level, for example, as a symmetry of the state we wish to generate or the Hamiltonian we wish to simulate. The flags, checking the parity of data qubits, are chosen to minimize the number of required entangling gates. While the method was used by Mooney *et al.* [Mooney et al., 2021] to generate GHZ states exhibiting genuine multi-partite entanglement (GME) as large as 25

*haoran.liao@q-ctrl.com

qubits, we show that the combination of an effective error suppression pipeline [Mundada et al., 2023, Hartnett et al., 2024] with such a limited error detection scheme allows to observe GME for GHZ states of up to 70 qubits, verified in terms of multiple-quantum coherence (MQC) fidelity. These results demonstrate the effectiveness of hardware-efficient error detection schemes, where the benefit gains due to the detection of bitflips and amplitude-damping errors, greatly exceed the quality degradation due to added overhead, enabling the largest GHZ state generation on any quantum processors reported to date.

In the second part of the paper, we tackle the challenge of generating entanglement at a distance, in particular, the generation of long-range CNOT gates. Long-range entangling operations are highly desirable for QEC protocols on low-connectivity device topologies, particularly for non-local QEC codes, such as quantum low-density parity-check (qLDPC) codes [Bravyi et al., 2024, Berthusen et al., 2024]. Moreover, the availability of non-local operations may shorten the path towards large-scale implementation of algorithms such as quantum Fourier transform and fermionic simulations [Holmes et al., 2020]. Engineering long-range entangling operations with high fidelity remains a significant and open problem, especially in the push towards QEC.

We introduce a novel protocol for generating a long-range CNOT gate, partially based on local operations and classical communications (LOCC) (mid-circuit measurement feedforward). The protocol we designed follows three principle guidelines: (1) It provides a trade-off between measurements and two-qubit gates, which better suits near-term hardware; (2) it enables efficient error detection, which allows for erroneous shots (samples) to be discarded; (3) it is more amenable to readout error mitigation in post-processing if allowed. We demonstrate our protocol to implement a long-range CNOT gate between two qubits separated by tens of physical qubits in the quantum device. We achieve $\gtrsim 90\%$ average gate fidelity of the long-range CNOT across up to 20 qubits in shot-by-shot experiments, and up to 30 qubits in post-processing with additional readout error mitigation. Our protocol for long-range CNOT gate significantly outperforms a recently proposed dynamic-circuit (mid-circuit measurement feedforward) protocol [Bäumer et al., 2024], which in turn significantly outperforms the baseline SWAP protocol. In addition, we provide an alternative, first-principle derivation of the dynamic-circuit protocol for long-range CNOT, which unifies it into a more general framework for teleporting control-U gates. This general framework also include our novel protocol for long-range CNOT mentioned above.

References

[Acharya et al., 2024] Acharya, R., Aghababaie-Beni, L., Aleiner, I., Andersen, T. I., Ansmann, M., Arute, F., Arya, K., Asfaw, A., Astrakhantsev, N., Atalaya, J., et al. (2024). Quantum error correction below the surface code threshold. *arXiv preprint arXiv:2408.13687*.

[Acharya et al., 2023] Acharya, R., Aleiner, I., Allen, R., Andersen, T. I., Ansmann, M., Arute, F., Arya, K., Asfaw, A., Atalaya, J., Babbush, R., et al. (2023). Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614(7949):676–681.

[Bäumer et al., 2024] Bäumer, E., Tripathi, V., Wang, D. S., Rall, P., Chen, E. H., Majumder, S., Seif, A., and Minev, Z. K. (2024). Efficient long-range entanglement using dynamic circuits. *PRX Quantum*, 5(3):030339.

[Berthusen et al., 2024] Berthusen, N., Devulapalli, D., Schoute, E., Childs, A. M., Gullans, M. J., Gorshkov, A. V., and Gottesman, D. (2024). Toward a 2d local implementation of quantum ldpc codes. *arXiv:2404.17676*.

[Bluvstein et al., 2024] Bluvstein, D., Evered, S. J., Geim, A. A., Li, S. H., Zhou, H., Manovitz, T., Ebadi, S., Cain, M., Kalinowski, M., Hangleiter, D., et al. (2024). Logical quantum processor based on reconfigurable atom arrays. *Nature*, 626(7997):58–65.

[Bravyi et al., 2024] Bravyi, S., Cross, A. W., Gambetta, J. M., Maslov, D., Rall, P., and Yoder, T. J. (2024). High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627(8005):778–782.

[Gupta et al., 2024] Gupta, R. S., Sundaresan, N., Alexander, T., Wood, C. J., Merkel, S. T., Healy, M. B., Hillenbrand, M., Jochym-O'Connor, T., Wootton, J. R., Yoder, T. J., et al. (2024). Encoding a magic state with beyond break-even fidelity. *Nature*, 625(7994):259–263.

[Hartnett et al., 2024] Hartnett, G. S., Barbosa, A., Mundada, P. S., Hush, M., Biercuk, M. J., and Baum, Y. (2024). Learning to rank quantum circuits for hardware-optimized performance enhancement. *arXiv:2404.06535*.

[Holmes et al., 2020] Holmes, A., Johri, S., Guerreschi, G. G., Clarke, J. S., and Matsuura, A. Y. (2020). Impact of qubit connectivity on quantum algorithm performance. *Quantum Science and Technology*, 5(2):025009.

[Mooney et al., 2021] Mooney, G. J., White, G. A., Hill, C. D., and Hollenberg, L. C. (2021). Generation and verification of 27-qubit greenberger-horne-zeilinger states in a superconducting quantum computer. *Journal of Physics Communications*, 5(9):095004.

[Mundada et al., 2023] Mundada, P. S., Barbosa, A., Maity, S., Wang, Y., Merkh, T., Stace, T., Nielson, F., Carvalho, A. R., Hush, M., Biercuk, M. J., et al. (2023). Experimental benchmarking of an automated deterministic error-suppression workflow for quantum algorithms. *Physical Review Applied*, 20(2):024034.

Quantum Noise Effects on QAOA for Reconfigurable Microgrid Networks

Betis Baheri*, Yan Li†, Wei Xu‡, and Qiang Guan*

*Kent State University, Kent, OH USA
† The Pennsylvania State University, PA, USA
‡ Brookhaven National Laboratory, NY, USA

***Abstract*—The Quantum Approximate Optimization Algorithm (QAOA) has gained attention as a potential solution for combinatorial optimization problems on near-term noisy quantum computers. In the context of networked microgrids (NMs), QAOA is used to optimize maximum power exchange by framing it as a Max-Cut problem. While promising, the effects of quantum noise on QAOA's performance on real Noisy Intermediate-Scale Quantum (NISQ) devices remain largely unexplored. This study thoroughly analyzes the impact of different types of quantum noise on QAOA's effectiveness in NMs, comparing results with idealized simulations. The findings open new avenues for research, emphasizing the need to improve the reliability, scalability, and error mitigation of quantum algorithms in practical settings.**

I. Introduction

In recent years, microgrids have become a recognized solution for improving energy sustainability [7]. A key challenge in networked microgrids (NMs), particularly reconfigurable ones, is optimizing maximum power exchange, which quantifies the dependence of each microgrid on its neighboring systems. This problem can be modeled as a Max-Cut problem [4], with the goal of maximizing power exchange efficiency by identifying the optimal cut value.

Two main approaches exist for solving the Max-Cut problem: relaxation and heuristic methods. Relaxation methods simplify the problem constraints, with algorithms like the Geomans-Williamson (GW) [5] achieving an expected cutting ratio of 0.878 by mapping discrete variables into vectors. The rank-two relaxation algorithm [2], by contrast, projects the discrete variables onto the unit circle in R^2, demonstrating superior approximation quality relative to the GW algorithm. Extensions of these techniques, including the rank-k relaxation method, among others, are documented as striving for improved approximations without surpassing the GW algorithm's efficacy. Heuristic methods, such as the simple iterative (SI) algorithm [8], divisive strategies for dense graphs, and Lagrangian relaxation [1], offer effective alternatives but often require high computational resources.

The Quantum Approximate Optimization Algorithm (QAOA) [6] has been shown to be adaptable for optimizing power exchange in NMs by approximating a Max-Cut solution. Research indicates that QAOA's performance can be significantly improved by optimizing the algorithm's angle parameters and normalizing edge weights [3], [9], and its efficacy increases with more layers (p-values). However, the selection of classical optimizers remains a challenge.

This study explores how noise in Noisy Intermediate-Scale Quantum (NISQ) devices affects QAOA's ability to solve the Max-Cut problem in NMs. By analyzing noise sources such as relaxation, dephasing, and gate errors, the research reveals that while increasing p-values generally enhances QAOA's performance, noise constrains these gains. The findings highlight the need for further investigation into QAOA's scalability, reliability, and error mitigation strategies when implemented on real quantum devices.

II. QAOA Noise Model

We used a noise model for quantum computations to simulate QAOA under quantum noise, accounting for gate errors, relaxation, and dephasing through depolarizing, amplitude damping, and phase damping channels. The input density matrix undergoes ideal gate operations before noise effects are applied, with the operator-sum method used to capture the impact of noise.

In real quantum computers, noise, especially in the ZZ-interaction (involving CNOT and RZ gates), affects QAOA by altering qubit amplitudes and introducing phase errors. These inaccuracies disrupt the solution space. Decoherence also causes errors when the Hamiltonian's operation exceeds qubit coherence times. We simulated a network microgrid power exchange problem (p=1) with both noiseless and noisy qubits to assess these effects.

III. Result of RNMs with QAOA

To evaluate the robustness and effectiveness of the Quantum Approximate Optimization Algorithm (QAOA) in networked microgrids (NMs), this study began by replicating previous experimental results from [6]. Initially, QAOA was tested without any noise models to establish a baseline for comparison.

The study advanced by incorporating a gate noise model focused on Pauli-Z operators to examine the effects of quantum noise on QAOA's performance. Using the quantum qt package, the model enabled a detailed analysis of how noise alters algorithmic results. The comparative analysis, shown

in Figure 1, highlights the algorithm's sensitivity to noise-induced deviations.

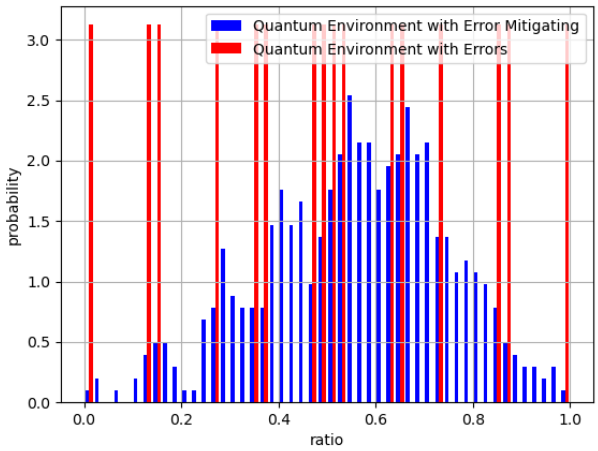


Fig. 1: Quantum Environment W/WO Error Mitigation, The red bars shows the final probability w.r.t ratio without error mitigation and blue lines shows the error mitigated result

In response to the challenges posed by quantum noise, the study explored noise mitigation strategies using Qiskit API version 0.46. This phase was crucial for assessing the effectiveness of noise reduction techniques in quantum computing applications for networked microgrids (NMs). Figure 2 visually compares QAOA's performance before and after noise mitigation, demonstrating the potential for improving the algorithm's resilience to noise.

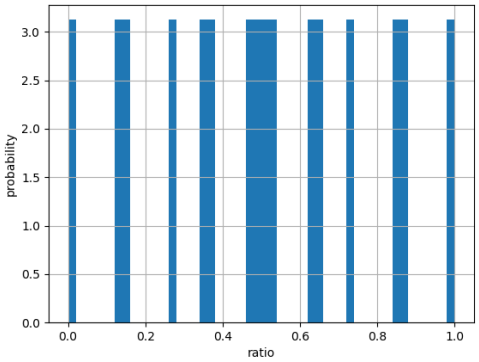
However, QAOA remains highly sensitive to even small noise variations, particularly in NMs, as evidenced by significant changes in the algorithm's output. This highlights the critical need for precise noise modeling and mitigation strategies.

In the final part of the study, QAOA was implemented in the Qiskit environment to compare the quasi-probability distribution of a two-qubit test case with the ideal probability distribution. Figures 2a and 2b illustrate the differences between the ideal and noise-mitigated contexts, emphasizing both the impact of quantum noise and the effectiveness of mitigation techniques in narrowing the gap between theoretical expectations and real-world challenges.

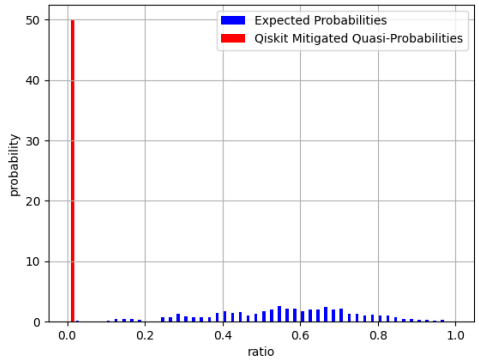
This study, from replicating experimental results to applying innovative noise mitigation techniques, makes a valuable contribution to the discussion on QAOA's resilience to quantum noise. By analyzing the algorithm's performance across various contexts, it highlights the key relationship between design, noise modeling, and mitigation strategies, offering insights for improving QAOA's robustness and effectiveness in quantum computing applications for network microgrids.

IV. Conclusion

This study provides a detailed analysis of the Quantum Approximate Optimization Algorithm (QAOA) for solving



(a) Probabilities w.r.t. ratio Noisy Environment



(b) Quasi-Probabilities of Mitigated Noisy Environment

Fig. 2: Comparision of expected solution Probabilities to Qiskit Mitigated Quasi-Probabilities

the MaxCut problem in Networked Microgrids (NMs) under realistic noise conditions in superconducting qubits. Results show that even minimal quantum noise can significantly impact the expected cut value. Basic noise mitigation techniques using the Qiskit framework yielded promising results, but further research is needed to explore advanced mitigation strategies and assess the scalability and effectiveness of QAOA in practical quantum computing environments.

References

[1] H. Alperin and I. Nowak, "Lagrangian smoothing heuristics for max-cut," *Journal of Heuristics*, vol. 11, no. 5, pp. 447–463, Dec 2005. [Online]. Available: https://doi.org/10.1007/s10732-005-3603-z

[2] S. Burer, R. D. C. Monteiro, and Y. Zhang, "Rank-two relaxation heuristics for max-cut and other binary quadratic programs," *SIAM Journal on Optimization*, vol. 12, no. 2, pp. 503–521, 2002. [Online]. Available: https://doi.org/10.1137/S1052623400382467

[3] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," 2014.

[4] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete problems," in *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '74. New York, NY, USA: Association for Computing Machinery, 1974, p. 47–63. [Online]. Available: https://doi.org/10.1145/800119.803884

[5] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite

programming," *J. ACM*, vol. 42, no. 6, p. 1115–1145, nov 1995. [Online]. Available: https://doi.org/10.1145/227683.227684

[6] H. Jing, Y. Wang, and Y. Li, "Interoperation analysis of reconfigurable networked microgrids through quantum approximate optimization algorithm," in *2022 IEEE Power & Energy Society General Meeting (PESGM)*, 2022, pp. 1–5.

[7] Y. Li, P. Zhang, and P. B. Luh, "Formal analysis of networked microgrids dynamics," *IEEE Transactions on Power Systems*, vol. 33, no. 3, pp. 3418–3427, 2018.

[8] S. Shao, D. Zhang, and W. Zhang, "A simple iterative algorithm for maxcut," 2023.

[9] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, "Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices," *Phys. Rev. X*, vol. 10, p. 021067, Jun 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.10.021067

Quantum error correction experiments decoded in real time and with low-latency response on a superconducting quantum computer

Laura Caune, Luka Skoric, Nick S. Blunt, Archibald Ruban, Jimmy McDaniel, Joseph A. Valery, Andrew D. Patterson, Alexander V. Gramolin, Joonas Majaniemi, Kenton M. Barnes, Tomasz Bialas, Okan Buğdaycı, Ophelia Crawford, György P. Gehér, Hari Krovi, Elisha Matekole, Canberk Topal, Stefano Poletto, Michael Bryant, Kalan Snyder, Neil I. Gillespie, Glenn Jones, Kauser Johar, Earl T. Campbell, Alexander D. Hill

Quantum error correction (QEC) will be essential to perform quantum computations that require hundreds of qubits and over a billion operations[1–4], which are susceptible to errors. QEC schemes repeatedly generate data characterising quantum errors[5–7] and use this data in classical algorithms, known as *decoders* to identify errors that occurred during quantum computation. Leading proposals for implementing fault-tolerant non-Clifford gates, for example magic state injection[8–10], require *logical branching* – logical operation conditional on a corrected observable, which is computed by combining the measured value of the observable and the logical correction returned by the decoder. Since logical branching happens during circuit execution and depends on the decoding result, this places throughput and latency requirements on the decoding process. In this work, we demonstrate fast, low-latency *full-decoding response time* by decoding with real-time FPGA decoder[11] integrated into Rigetti's superconducting qubit device Ankaa-2™ control system.

The rate at which the decoder processes data (the throughput) needs to be greater than the rate at which data is generated, which on superconducting devices can be as fast as one data extraction round per 1 µs[12–15]. This is necessary to avoid the backlog problem[5], i.e. an exponential slowdown of the quantum computation due to amassing a growing backlog of data at each decoding iteration. A second key parameter to optimize is the full-decoding response time, which is the time between the final data extraction round and the application of a logical conditional gate. It includes the decoding time as well as the communication and control latency times to send the data to and from the decoder. The full-decoding response time can be a significant bottleneck for operations involving logical conditional gates. As a result, it directly affects the logical clock rate (i.e. the inverse of the time required to perform a single logical non-Clifford gate), another QEC metric that determines the execution speed of fault-tolerant algorithms. Reducing both the decoding time and the communication and control latency to minimize the full-decoding response time is key to ensure that complex quantum algorithms are executed within reasonable times. For example, when estimating that 2048 bit RSA integers can be factored in 8 hours using 20 million noisy superconducting qubits, the authors assume a full-decoding response time within 10 µs[1]. Real-time decoding has been experimentally demonstrated but with either full-decoding response time far exceeding the 10 µs goal[16;17] or using an unscalable lookup table approach[16;18–20].

Here, we present a real-time decoded *stability experiment*[21] on 8 qubits (*stability*-8) with logical branching that measures the full-decoding response time. We decode with a scalabale FPGA implementation of a Collision Clustering decoder[11], integrated into Rigetti's Ankaa-2™ superconducting device's control system. Surface code operations such as lattice surgery[22;23], logical qubit patch movement[10] or the logical Hadamard gate[24;25] all involve measuring products of stabilizers. In these examples, incorrectly measuring the stabilizer product can introduce a logical error. For instance, in lattice surgery, this stabilizer product determines the logical branching decisions when performing non-Clifford gates, with a logical error meaning the wrong branch is followed. The stability experiment[21] is designed to test the ability to protect against these logical errors and in this work acts as a smaller scale simulation of logical branching experiments.

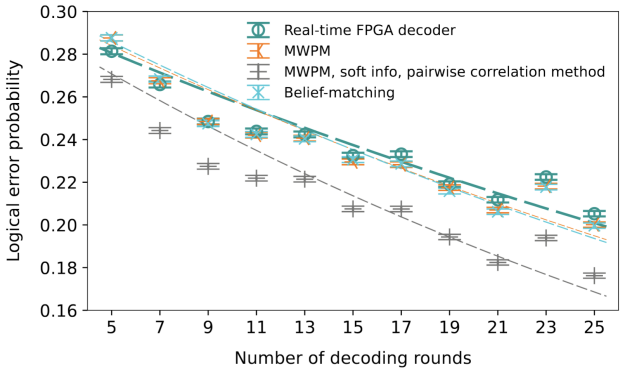


Figure 1: Logical error probabilities for the stability-8 experiment executed on the Ankaa-2™ device as a function of number of decoding rounds.

In Fig. 1, we show that on Ankaa-2™ using the real-time FPGA decoder the physical error rates are low enough to demonstrate logical error probability suppression with an increasing number of decoding rounds, which is the signature of a successful stability experiment. The logical error probabilities decrease from around $(28.1 \pm 0.1)\%$ at 5 decoding rounds to around $(20.5 \pm 00.1)\%$ at 25 decoding rounds. We achieve the lowest logical error probability, $(17.6 \pm 0.1)\%$, with minimum-weight perfect matching (MWPM) decoder leveraging soft information analysis and a decoding graph constructed with the pairwise correlation method[26–30].

Fig. 2 shows that our mean decoding time per round ranges from 0.44 µs when decoding 5 rounds to 0.79 µs when decoding 25 rounds. These values remain below the 1 µs threshold for data generation on a superconducting qubit device, providing strong evidence that the backlog problem will be avoided when operated as a streaming decoder[31–33].

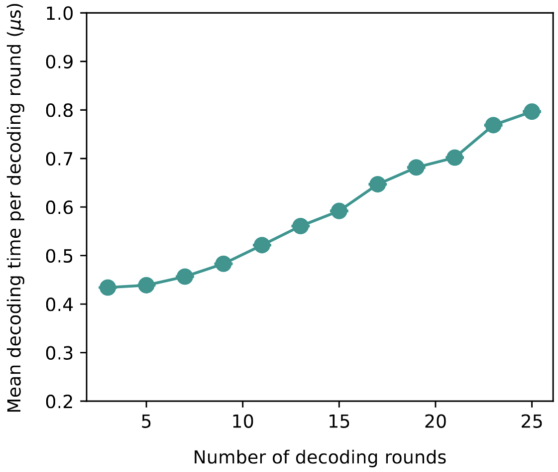


Figure 2: Mean decoding time per decoding round for the stability-8 experiment, decoded using the real-time FPGA decoder run at 156.25 MHz frequency.

We also perform a circuit that conditionally applies a physical X gate based on the outcome of decoding the stability experiment. This allows us to measure the full-decoding response time as a function of the number of QEC rounds (see Fig. 3). For a 9 measurements rounds experiment, we find the full-decoding response time to be 9.6 µs, including 6.5 µs decoding time and 3.1 µs communication and control latencies. We note that while the majority of the full-decoding response time can be attributed to the decoding time for a larger number of rounds, there is a significant additional latency incurred by the control system delays. 1.4 µs is the time spent waiting for the final readout results to reach the decoder, with the remaining 1-1.7 µs consisting of additional control system logic: collecting the measurements into packets to be sent to the decoder, sending them via the wishbone bus, receiving results and performing the conditional logic. Thanks to the FPGA implementation of our fast decoder and its integration into the Ankaa-2™ system, in the small-distance studies reported in this work we manage to keep the full-decoding response time within the order of d µs, where d is the code distance. Maintaining this condition will be crucial when scaling up d, as it will ensure that this response time will not be a critical limiting factor for the logical clock speed when implementing lattice surgery operations[32].

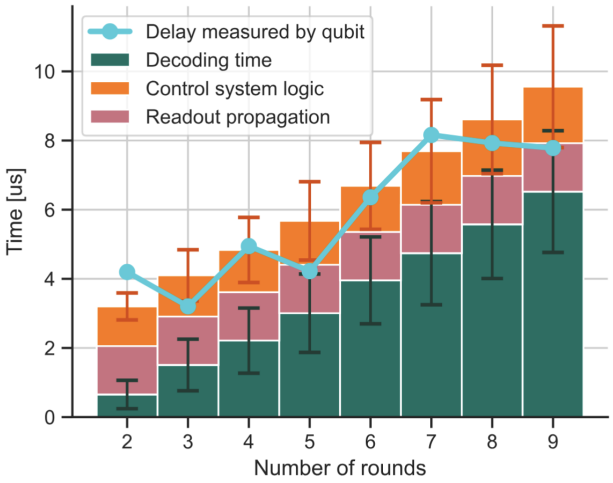


Figure 3: Full-decoding response time measurements as a function of number of rounds.

In the future, fault-tolerant quantum algorithms that have the potential to outperform classical algorithms will require codes utilizing a large number of operations[1–4;10]. To support this, the FPGA decoder should be updated to be a streaming decoder[31–33]. To improve the accuracy of decoding results, we expect future FPGA implementations to enable updates to the decoding graph in real-time, allowing for soft information or leakage-aware decoding. Such advances to decoding and improvements to the device combined with the throughput and latency developed in this work, will unlock the next generation of experiments that go beyond purely keeping logical qubits alive and into demonstrating building blocks of fault-tolerant computation, such as lattice surgery and magic state teleportation.

References

[1] C. Gidney and M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits," Publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften.

[2] M. Reiher, N. Wiebe, K. M. Svore, D. Wecker, and M. Troyer, Proceedings of the National Academy of Sciences **114**, 7555 (2017), publisher: Proceedings of the National Academy of Sciences.

[3] N. S. Blunt, J. Camps, O. Crawford, R. Izsák, S. Leontica, A. Mirani, A. E. Moylett, S. A. Scivier, C. Sünderhauf, P. Schopf, J. M. Taylor, and N. Holzmann, Journal of Chemical Theory and Computation **18**, 7001 (2022), publisher: American Chemical Society.

[4] J. Lee, D. W. Berry, C. Gidney, W. J. Huggins, J. R. McClean, N. Wiebe, and R. Babbush, PRX Quantum **2**, 030305 (2021), publisher: American Physical Society.

[5] B. M. Terhal, Reviews of Modern Physics **87**, 307 (2015), publisher: American Physical Society.

[6] D. Gottesman, "Stabilizer Codes and Quantum Error Correction," (1997), arXiv:quant-ph/9705052.

[7] E. T. Campbell, B. M. Terhal, and C. Vuillot, Nature **549**, 172 (2017), publisher: Nature Publishing Group.

[8] S. Bravyi and J. Haah, Physical Review A **86**, 052329 (2012), publisher: American Physical Society.

[9] S. Bravyi and A. Kitaev, Physical Review A **71**, 022316 (2005), publisher: American Physical Society.

[10] D. Litinski, Quantum **3**, 128 (2019), arXiv:1808.02892 [cond-mat, physics:quant-ph].

[11] B. Barber, K. M. Barnes, T. Bialas, O. Buğdaycı, E. T. Campbell, N. I. Gillespie, K. Johar, R. Rajan, A. W. Richardson, L. Skoric, C. Topal, M. L. Turner, and A. B. Ziad, "A real-time, scalable, fast and highly resource efficient decoder for a quantum computer," (2023), arXiv:2309.05558 [quant-ph].

[12] F. Battistel, C. Chamberland, K. Johar, R. W. J. Overwater, F. Sebastiano, L. Skoric, Y. Ueno, and M. Usman, Nano Futures **7**, 032003 (2023), publisher: IOP Publishing.

[13] J. P. G. van Dijk, E. Charbon, and F. Sebastiano, Microprocessors and Microsystems **66**, 90 (2019).

[14] E. Jeffrey, D. Sank, J. Y. Mutus, T. C. White, J. Kelly, R. Barends, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, A. Megrant, P. J. J. O'Malley, C. Neill, P. Roushan, A. Vainsencher, J. Wenner, A. N. Cleland, and J. M. Martinis, Physical Review Letters **112**, 190504 (2014), publisher: American Physical Society.

[15] R. Acharya, I. Aleiner, R. Allen, T. I. Andersen, M. Ansmann, F. Arute, K. Arya, A. Asfaw, J. Atalaya, R. Babbush, D. Bacon, J. C. Bardin, J. Basso, A. Bengtsson, S. Boixo, G. Bortoli, A. Bourassa, J. Bovaird, L. Brill, M. Broughton, B. B. Buckley, D. A. Buell, T. Burger, B. Burkett, N. Bushnell, Y. Chen, Z. Chen, B. Chiaro, J. Cogan, R. Collins, P. Conner, W. Courtney, A. L. Crook, B. Curtin, D. M. Debroy, A. Del Toro Barba, S. Demura, A. Dunsworth, D. Eppens, C. Erickson, L. Faoro, E. Farhi, R. Fatemi, L. Flores Burgos, E. Forati, A. G. Fowler, B. Foxen, W. Giang, C. Gidney, D. Gilboa, M. Giustina, A. Grajales Dau, J. A. Gross, S. Habegger, M. C. Hamilton, M. P. Harrigan, S. D. Harrington, O. Higgott, J. Hilton, M. Hoffmann, S. Hong, T. Huang, A. Huff, W. J. Huggins, L. B. Ioffe, S. V. Isakov, J. Iveland, E. Jeffrey, Z. Jiang, C. Jones, P. Juhas, D. Kafri, K. Kechedzhi, J. Kelly, T. Khattar, M. Khezri, M. Kieferová, S. Kim, A. Kitaev, P. V. Klimov, A. R. Klots, A. N. Korotkov, F. Kostritsa, J. M. Kreikebaum, D. Landhuis, P. Laptev, K.-M. Lau, L. Laws, J. Lee, K. Lee, B. J. Lester, A. Lill, W. Liu, A. Locharla, E. Lucero, F. D. Malone, J. Marshall, O. Martin, J. R. McClean, T. McCourt, M. McEwen, A. Megrant, B. Meurer Costa, X. Mi, K. C. Miao, M. Mohseni, S. Montazeri, A. Morvan, E. Mount, W. Mruczkiewicz, O. Naaman, M. Neeley, C. Neill, A. Nersisyan, H. Neven, M. Newman, J. H. Ng, A. Nguyen, M. Nguyen, M. Y. Niu, T. E. O'Brien, A. Opremcak, J. Platt, A. Petukhov, R. Potter, L. P. Pryadko, C. Quintana, P. Roushan, N. C. Rubin, N. Saei, D. Sank, K. Sankaragomathi, K. J. Satzinger, H. F. Schurkus, C. Schuster, M. J. Shearn, A. Shorter, V. Shvarts, J. Skruzny, V. Smelyanskiy, W. C. Smith, G. Sterling, D. Strain, M. Szalay, A. Torres, G. Vidal, B. Villalonga, C. Vollgraff Heidweiller, T. White, C. Xing, Z. J. Yao, P. Yeh, J. Yoo, G. Young, A. Zalcman, Y. Zhang, N. Zhu, and Google Quantum AI, Nature **614**, 676 (2023).

[16] C. Ryan-Anderson, J. Bohnet, K. Lee, D. Gresh, A. Hankin, J. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. Brown, T. Gatterman, S. Halit, K. Gilmore, J. Gerber, B. Neyenhuis, D. Hayes, and R. Stutz, Physical Review X **11**, 041058 (2021), publisher: American Physical Society.

[17] R. Acharya, L. Aghababaie-Beni, I. Aleiner, T. I. Andersen, M. Ansmann, F. Arute, K. Arya, A. Asfaw, N. Astrakhantsev, J. Atalaya, R. Babbush, D. Bacon, B. Ballard, J. C. Bardin, J. Bausch, A. Bengtsson, A. Bilmes, S. Blackwell, S. Boixo, G. Bortoli, A. Bourassa, J. Bovaird, L. Brill, M. Broughton, D. A. Browne, B. Buchea, B. B. Buckley, D. A. Buell, T. Burger, B. Burkett, N. Bushnell, A. Cabrera, J. Campero, H.-S. Chang, Y. Chen, Z. Chen, B. Chiaro, D. Chik, C. Chou, J. Claes, A. Y. Cleland, J. Cogan, R. Collins, P. Conner, W. Courtney, A. L. Crook, B. Curtin, S. Das, A. Davies, L. De Lorenzo, D. M. Debroy, S. Demura, M. Devoret, A. Di Paolo, P. Donohoe, I. Drozdov, A. Dunsworth, C. Earle, T. Edlich, A. Eickbusch, A. M. Elbag, M. Elzouka, C. Erickson, L. Faoro, E. Farhi, V. S. Ferreira, L. F. Burgos, E. Forati, A. G. Fowler, B. Foxen, S. Ganjam, G. Garcia, R. Gasca, E. Genois, W. Giang, C. Gidney, D. Gilboa, R. Gosula, A. G. Dau, D. Graumann, A. Greene, J. A. Gross, S. Habegger, J. Hall, M. C. Hamilton, M. Hansen, M. P. Harrigan, S. D. Harrington, F. J. H. Heras, S. Heslin, P. Heu, O. Higgott, G. Hill, J. Hilton, G. Holland, S. Hong, H.-Y. Huang, A. Huff, W. J. Huggins, L. B. Ioffe, S. V. Isakov, J. Iveland, E. Jeffrey, Z. Jiang, C. Jones, S. Jordan, C. Joshi, P. Juhas, D. Kafri, H. Kang, A. H. Karamlou, K. Kechedzhi, J. Kelly, T. Khaire, T. Khattar, M. Khezri, S. Kim, P. V. Klimov, A. R. Klots, B. Kobrin, P. Kohli, A. N. Korotkov, F. Kostritsa, R. Kothari, B. Kozlovskii, J. M. Kreikebaum, V. D. Kurilovich, N. Lacroix, D. Landhuis, T. Lange-Dei, B. W. Langley, P. Laptev, K.-M. Lau, L. L. Guevel, J. Ledford, K. Lee, Y. D. Lensky, S. Leon, B. J. Lester, W. Y. Li, Y. Li, A. T. Lill, W. Liu, W. P. Livingston, A. Locharla, E. Lucero, D. Lundahl, A. Lunt, S. Madhuk, F. D. Malone, A. Maloney, S. Mandrá,

L. S. Martin, S. Martin, O. Martin, C. Maxfield, J. R. McClean, M. McEwen, S. Meeks, A. Megrant, X. Mi, K. C. Miao, A. Mieszala, R. Molavi, S. Molina, S. Montazeri, A. Morvan, R. Movassagh, W. Mruczkiewicz, O. Naaman, M. Neeley, C. Neill, A. Nersisyan, H. Neven, M. Newman, J. H. Ng, A. Nguyen, M. Nguyen, C.-H. Ni, T. E. O'Brien, W. D. Oliver, A. Opremcak, K. Ottosson, A. Petukhov, A. Pizzuto, J. Platt, R. Potter, O. Pritchard, L. P. Pryadko, C. Quintana, G. Ramachandran, M. J. Reagor, D. M. Rhodes, G. Roberts, E. Rosenberg, E. Rosenfeld, P. Roushan, N. C. Rubin, N. Saei, D. Sank, K. Sankaragomathi, K. J. Satzinger, H. F. Schurkus, C. Schuster, A. W. Senior, M. J. Shearn, A. Shorter, N. Shutty, V. Shvarts, S. Singh, V. Sivak, J. Skruzny, S. Small, V. Smelyanskiy, W. C. Smith, R. D. Somma, S. Springer, G. Sterling, D. Strain, J. Suchard, A. Szasz, A. Sztein, D. Thor, A. Torres, M. M. Torunbalci, A. Vaishnav, J. Vargas, S. Vdovichev, G. Vidal, B. Villalonga, C. V. Heidweiller, S. Waltman, S. X. Wang, B. Ware, K. Weber, T. White, K. Wong, B. W. K. Woo, C. Xing, Z. J. Yao, P. Yeh, B. Ying, J. Yoo, N. Yosri, G. Young, A. Zalcman, Y. Zhang, N. Zhu, and N. Zobrist, "Quantum error correction below the surface code threshold," 2408.13687 [quant-ph] .

[18] L. Egan, D. M. Debroy, C. Noel, A. Risinger, D. Zhu, D. Biswas, M. Newman, M. Li, K. R. Brown, M. Cetina, and C. Monroe, Nature **598**, 281 (2021), publisher: Nature Publishing Group.

[19] D. Ristè, L. C. G. Govia, B. Donovan, S. D. Fallek, W. D. Kalfus, M. Brink, N. T. Bronn, and T. A. Ohki, npj Quantum Information **6**, 1 (2020), publisher: Nature Publishing Group.

[20] D. Bluvstein, S. J. Evered, A. A. Geim, S. H. Li, H. Zhou, T. Manovitz, S. Ebadi, M. Cain, M. Kalinowski, D. Hangleiter, J. P. Bonilla Ataides, N. Maskara, I. Cong, X. Gao, P. Sales Rodriguez, T. Karolyshyn, G. Semeghini, M. J. Gullans, M. Greiner, V. Vuletić, and M. D. Lukin, Nature **626**, 58, publisher: Nature Publishing Group.

[21] C. Gidney, Quantum **6**, 786 (2022), arXiv:2204.13834 [quant-ph].

[22] D. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, New Journal of Physics **14**, 123011 (2012), arXiv:1111.4022 [quant-ph].

[23] A. Erhard, H. Poulsen Nautrup, M. Meth, L. Postler, R. Stricker, M. Stadler, V. Negnevitsky, M. Ringbauer, P. Schindler, H. J. Briegel, R. Blatt, N. Friis, and T. Monz, Nature **589**, 220 (2021), publisher: Nature Publishing Group.

[24] H. Bombin, C. Dawson, R. V. Mishmash, N. Nickerson, F. Pastawski, and S. Roberts, PRX Quantum **4**, 020303 (2023), arXiv:2112.12160 [quant-ph].

[25] G. P. Gehér, C. McLauchlan, E. T. Campbell, A. E. Moylett, and O. Crawford, Quantum **8**, 1394 (2024), publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften.

[26] E. H. Chen, T. J. Yoder, Y. Kim, N. Sundaresan, S. Srinivasan, M. Li, A. D. Córcoles, A. W. Cross, and M. Takita, Physical Review Letters **128**, 110504 (), publisher: American Physical Society.

[27] S. T. Spitz, B. Tarasinski, C. W. Beenakker, and T. E. O'Brien, Advanced Quantum Technologies **1**, 1800012 (2018).

[28] Z. Chen, K. J. Satzinger, J. Atalaya, A. N. Korotkov, A. Dunsworth, D. Sank, C. Quintana, M. McEwen, R. Barends, P. V. Klimov, S. Hong, C. Jones, A. Petukhov, D. Kafri, S. Demura, B. Burkett, C. Gidney, A. G. Fowler, A. Paler, H. Putterman, I. Aleiner, F. Arute, K. Arya, R. Babbush, J. C. Bardin, A. Bengtsson, A. Bourassa, M. Broughton, B. B. Buckley, D. A. Buell, N. Bushnell, B. Chiaro, R. Collins, W. Courtney, A. R. Derk, D. Eppens, C. Erickson, E. Farhi, B. Foxen, M. Giustina, A. Greene, J. A. Gross, M. P. Harrigan, S. D. Harrington, J. Hilton, A. Ho, T. Huang, W. J. Huggins, L. B. Ioffe, S. V. Isakov, E. Jeffrey, Z. Jiang, K. Kechedzhi, S. Kim, A. Kitaev, F. Kostritsa, D. Landhuis, P. Laptev, E. Lucero, O. Martin, J. R. McClean, T. McCourt, X. Mi, K. C. Miao, M. Mohseni, S. Montazeri, W. Mruczkiewicz, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Newman, M. Y. Niu, T. E. O'Brien, A. Opremcak, E. Ostby, B. Pató, N. Redd, P. Roushan, N. C. Rubin, V. Shvarts, D. Strain, M. Szalay, M. D. Trevithick, B. Villalonga, T. White, Z. J. Yao, P. Yeh, J. Yoo, A. Zalcman, H. Neven, S. Boixo, V. Smelyanskiy, Y. Chen, A. Megrant, J. Kelly, and Google Quantum AI, Nature **595**, 383 (), publisher: Nature Publishing Group.

[29] H. Ali, J. Marques, O. Crawford, J. Majaniemi, M. Serra-Peralta, D. Byfield, B. Varbanov, B. M. Terhal, L. DiCarlo, and E. T. Campbell, "Reducing the error rate of a superconducting logical qubit using analog readout information," (2024), arXiv:2403.00706 [cond-mat, physics:quant-ph].

[30] C. A. Pattison, M. E. Beverland, M. P. da Silva, and N. Delfosse, arXiv preprint arXiv:2107.13589 (2021).

[31] L. Skoric, D. E. Browne, K. M. Barnes, N. I. Gillespie, and E. T. Campbell, Nature Communications **14**, 7040 (2023), publisher: Nature Publishing Group.

[32] H. Bombín, C. Dawson, Y.-H. Liu, N. Nickerson, F. Pastawski, and S. Roberts, "Modular decoding: Parallelizable real-time decoding for quantum computers," 2303.04846 .

[33] X. Tan, F. Zhang, R. Chao, Y. Shi, and J. Chen, "Scalable Surface-Code Decoders with Parallelization in Time," .

Protocols and Applications of Quantum Stack Memory

Leonard Li
North Carolina State University
Raleigh, United States
sli74@ncsu.edu

Lingjun Xiong
North Carolina State University
Raleigh, United States
lxiong4@ncsu.edu

Yuan Liu
North Carolina State University
Raleigh, United States
q_yuanliu@ncsu.edu

Introduction. Quantum memory is an indispensable component for quantum computers. Prior art on quantum memory has focused on quantum random-access memory [1, 4, 7, 8], the quantum analog of classical random-access memory (RAM). Despite decades of efforts, efficient and reliable protocols for realizing QRAMs are still challenging. This is mostly due to the stringent requirement on the feature of *random*-accessing.

It is worth noting that useful computation may not necessarily require fully random-access to the memory, because data usage can often exhibit spatial and temporal locality [2] which tends to relax the requirement on "random" access. In classical computers, such temporal locality of data usage is well captured by the so-called *stack memory*, where the last bit of data that was stored has to be retrieved first, i.e., "last-in-first-out". Despite the limited memory-access ability in stack memory, they can be easy to realize physically. Envisioning similar temporal locality in quantum data access models, a quantum stack memory can be of paramount importance in these cases. More importantly, because memory access only need to be "last-in-first-out", it is reasonable to believe that Q-stack can be easier to realize and possess smaller overhead physically than QRAM.

At the physical level, a minimum requirement for realizing compact quantum memory is the existence of Hilbert space with large dimensions. Bosonic modes, or quantum harmonic oscillators, are ubiquitous in nature, and their infinite-dimensional nature [5] serves as natural hardware-efficient quantum memories. However, writing/reading quantum data in/out such oscillator memory is challenging, especially in a qubit-by-qubit fashion. Protocols exist to use oscillators as quantum associative memory [3], but no proposal for quantum stack memory that can employ temporal locality has been designed using oscillators.

In this work, we develop protocols for realizing quantum stack memory and quantum queue for hybrid oscillator-qubit quantum processors. We provide physical level realizations of the push and pop operation using native instruction gate sets, and analyze the performance of these new quantum memory protocol under noise. We conclude by a discussion on the utility of these new quantum memories for quantum data processing and quantum learning models.

New Quantum Memory: Quantum Stack. In computer science, a **stack** is an abstract data type that functions as a collection of elements with two primary operations: 1) **Push**: Adds an element to the collection; 2) **Pop**: Removes the most recently added element. Stacks operate on a Last-In-First-Out (LIFO) principle, meaning the last element added is the first to be removed. This abstract data structure supports address-independent operations, making it a fundamental tool for a variety of applications, including Depth-First Search, Base Conversion and Bracket Matching.

The **Quantum Stack** mirrors the operations of a classical stack but offers unique advantages in pushing and popping quantum data instead of classical ones. To demonstrate how our Q-stack works clearly, a schematic is shown in Fig. 1, where unitary operation U_{pop} (U_{push}) is designed to pop (push) one qubit of data out of (into) the Q-stack. A quantum harmonic oscillator is used to store the quantum data, while a qubit is coupled to the oscillator to facilitate easy quantum control of the oscillator.

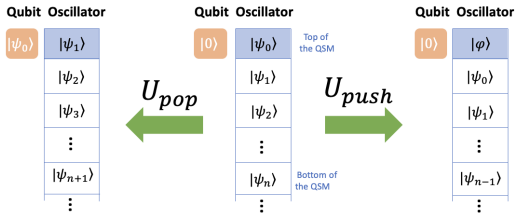


Figure 1: Schematic of the operation of a quantum stack memory based on oscillator-qubit systems, showing the action of the push and pop operations.

Physical Realization of Q-Stack from Correction-free Quantum Teleportation. Our idea for realizing quantum stack is to teleport an external qubit of states into a superposition of Fock levels of the oscillator. For Fock level $|i\rangle$, define the shift operator $S_n|i\rangle = |i + 2^n\rangle$. Define the logical zero and one that store n qubits of information recursively as

$$|\bar{0}\rangle_n = \alpha_n|\bar{0}\rangle_{n-1} + \beta_n|\bar{1}\rangle_{n-1},\ |\bar{1}\rangle_n = S_n|\bar{0}\rangle_n. \quad (1)$$

The entanglement generate unitary U_n can be synthesized by using U_n

$$U_n|0\rangle_{\text{qubit}} \otimes |i\rangle = \frac{1}{\sqrt{2}}(|0\rangle_{\text{qubit}} \otimes |i\rangle + |1\rangle_{\text{qubit}} \otimes |i + 2^{n-1}\rangle), \quad (2)$$

or equivalently on the logical states, $U_n|0\rangle|\bar{0}\rangle_{n-1} = \frac{1}{\sqrt{2}}\big[|0\rangle|\bar{0}\rangle_{n-1} + |1\rangle|\bar{1}\rangle_{n-1}\big]$. Using U_n, it can be shown that the quantum circuit in Fig. 2 can be used to teleport an external qubit of data $|\phi_n\rangle_e$ into the oscillator.

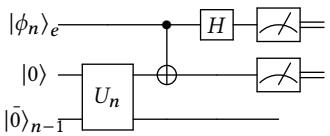


Figure 2: Quantum circuit which teleports an external qubit $|\phi_n\rangle_e$ into the bosonic oscillator without the Pauli correction operation, resulting in an error version of $|\phi_n\rangle_e$ stored in the oscillator where the four possible errors are stored as a 2-bit classical bitstring obtained from the measurement.

The entangling gate U_n is not a native gate on oscillator-qubit systems. However, it can be realized by using the following sequence of native gates in the instruction sets of hybrid oscillator-qubit quantum processors

$$U_{n,\text{approx}} = D(\alpha_n)\text{SQR}(\vec{\theta}_n, \vec{\phi}_n)D(\beta_n)\text{SQR}(\vec{\delta}_n, \vec{\eta}_n)D(\gamma_n), \quad (3)$$

where $D(\alpha) \equiv e^{\alpha a^\dagger - \alpha^* a}$ is the displacement gate, and $\text{SQR}(\vec{\theta}, \vec{\varphi}) = \sum_{n=0}^{N_{\max}} R_{\varphi_n}(\theta_n) \otimes |n\rangle\langle n|$ is the photon-number-Selective Qubit Rotation (SQR) gate [6] for $R_\varphi(\theta) = \exp\left(-i\frac{\theta}{2}\sigma_\varphi\right)$ and $\sigma_\varphi = \sigma_x \cos\varphi + \sigma_y \sin\varphi$. $\{\alpha_n, \beta_n, \gamma_n\} \in \mathbb{C}$ are complex numbers representing the displacement amount, while $\{\vec{\theta}_n, \vec{\phi}_n, \vec{\delta}_n, \vec{\eta}_n\} \in [0, 2\pi)$ are four vectors of phase angles parametrizing the SQR gate. It can be shown numerically that Eq. (3) can realize U_n with high precision.

From Quantum Stack to Quantum Queue. Build on top of Q-stack, we demonstrate that more complicated quantum memory access pattern such as quantum queue can be realized by using two quantum stacks.

A **queue** is an abstract data type in computer science characterized by its First-In-First-Out (FIFO) operation. It represents a collection of entities organized in a sequence, where entities are added to one end and removed from the opposite end. The end where elements are added is known as the **back** or **rear**, while the end where elements are removed is the **head** or **front**. This structure mirrors how people line up for goods or services.

Queues support two primary operations: 1) **Enqueue**: Add an entity to the back of the queue; 2) **Dequeue**: Remove an entity from the front of the queue. Queues are pivotal in various applications due to their address-independent operations and their natural fit for sequential data processing. Notable uses include Breadth-First Search, Message Queues, Task Scheduling and a key feature of our work: Buffer Management.

Quantum Queue: Advantages and Implementation. The **Quantum Queue** builds upon the classical queue's properties, incorporating all the advantages of the Quantum Stack over the classical stack. By implementing a quantum version of the queue, we achieve a significant milestone: the quantum realization of two of computer science's most classic data structures.

By leveraging the Quantum Stack, we have devised a method to implement a Quantum Queue without introducing any additional physical structures. This implementation incurs an overhead cost of precisely twice that of the Quantum Stack, as it utilizes two quantum stacks to form a quantum queue. Each data throughput operation involves exactly 2 **push** and 2 **pop** actions, as illustrated in Figure 3. A formal algorithm for this implementation can be found in Algorithm 1. For brevity, we omit the proof of its correctness.

Applications and Discussions. Leveraging Q-stack and Q-queue can enable many interesting quantum data processing and learning tasks. For example, quantum queue allows asynchronization between quantum data acquisition and processing. Storage of multi-qubit of data into a single oscillator mode also opens the potential for quantum single-instruction-multi-data processing. Our work opens a new avenue for quantum memory design that will play an important role in quantum computer system design and applications.

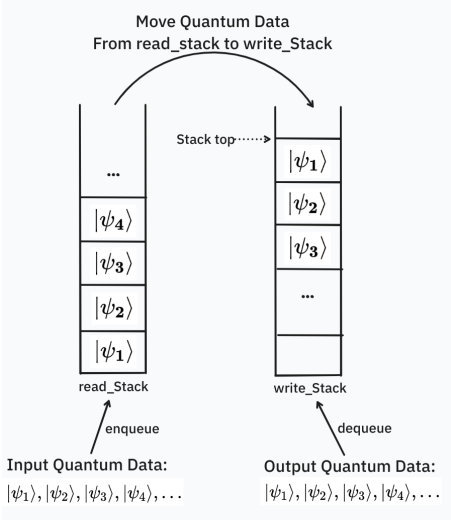


Figure 3: Stack to Queue Illustration

Algorithm 1 Queue Operations with Two Stacks

```
QuantumStack read_stack, write_stack;
function enQueue(data)
    write_stack.push(data)
end function
function deQueue
    if read_stack is empty then
        while write_stack not empty do
            tmp_data = write_stack.pop()
            read_stack.push(tmp_data)
        end while
    end if
  return read_stack.pop()
end function
```

REFERENCES

[1] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. 2008. Quantum random access memory. *Physical review letters* 100, 16 (2008), 160501.

[2] John L Hennessy and David A Patterson. 2011. *Computer architecture: a quantitative approach.* Elsevier.

[3] Adrià Labay-Mora, Roberta Zambrini, and Gian Luca Giorgi. 2023. Quantum Associative Memory with a Single Driven-Dissipative Nonlinear Oscillator. *Physical Review Letters* 130, 19 (2023), 190602.

[4] Chenxu Liu, Meng Wang, Samuel A Stein, Yufei Ding, and Ang Li. 2023. Quantum Memory: A Missing Piece in Quantum Computing Units. *arXiv preprint arXiv:2309.14432* (2023).

[5] Yuan Liu, Jasmine Sinanan-Singh, Matthew T. Kearney, Gabriel Mintzer, and Isaac L. Chuang. 2021. Constructing qudits from infinite-dimensional oscillators by coupling to qubits. *Phys. Rev. A* 104 (Sep 2021), 032605. Issue 3. https://doi.org/10.1103/PhysRevA.104.032605

[6] Yuan Liu, Shraddha Singh, Kevin C Smith, Eleanor Crane, John M Martyn, Alec Eickbusch, Alexander Schuckert, Richard D Li, Jasmine Sinanan-Singh, Micheline B Soley, et al. 2024. Hybrid Oscillator-Qubit Quantum Processors: Instruction Set Architectures, Abstract Machine Models, and Applications. *arXiv preprint arXiv:2407.10381* (2024). https://arxiv.org/abs/2407.10381

[7] DK Weiss, Shruti Puri, and SM Girvin. 2023. QRAM architectures using superconducting cavities. *arXiv preprint arXiv:2310.08288* (2023).

[8] Shifan Xu, Connor T Hann, Ben Foxman, Steven M Girvin, and Yongshan Ding. 2023. Systems Architecture for Quantum Random Access Memory. *arXiv preprint arXiv:2306.03242* (2023).

Fast quantum interconnects via constant-rate entanglement distillation

C. A. Pattison*, G. Baranes*, J. P. Bonilla Ataides, M. D. Lukin, and H. Zhou
(Dated: September 30, 2024. QuEra Computing Inc., Harvard University, MIT, Caltech)

Background

Distributed quantum computing allows the modular construction of large-scale quantum computers and enables new protocols for verifiably-secure quantum computation [1–7]. However, such applications place stringent demands on the fidelity and rate of entanglement generation, which are not met by existing methods for quantum interconnects. This is particularly challenging in the regime of large-scale, fault-tolerant distributed quantum computing, which requires very low logical error rates and fast logical clock speeds. For example, state-of-the-art demonstrations of interconnecting quantum nodes only achieve logical entangling fidelities in the high 90s and entangling rates of a few hundred pairs per second [8–14]. The entanglement fidelity can be improved through the use of various one-way or two-way entanglement distillation schemes [15, 16], but this leads to a further degradation in the effective logical entangling rate. With typical target gate operation fidelities for large-scale algorithms in the 10^{-12} range, the standard 4-to-1 recursive distillation scheme would require almost a hundred physical Bell pairs (per logical Bell pair) of percent-level infidelity. Alternative schemes relying on lattice surgery have also been developed [17, 18], but would again require hundreds to thousands of physical Bell pairs per logical Bell pair in similar parameter regimes. The combination of these factors suggest that current schemes and physical hardware will result in logical entangling rates of a few Hz, much slower than the QEC cycle times of 1 μs to 1 ms in various state-of-the-art hardware systems [19–27]. It is thus highly desirable to develop methods that use the communication link much more sparingly, thereby achieving higher logical entangling rates while maintaining high target logical fidelities.

Constant-Overhead Entanglement Distillation

In this work, we develop entanglement distillation methods that achieve constant communication rate, and show that they yield practical protocols with very low communication overhead. To achieve constant communication rate, we adapt a technique originating in fault-tolerant computation [28], where careful selection of the concatenation sequence permits a concatenated code family to achieve constant rate. We rigorously analyze the error rates and success probabilities at each step of the protocol, thereby providing theoretical guarantees of the distillation overhead and performance.

We work in the setting where two-way classical communication is permitted [16], thereby allowing the use of error detection and post-selection instead of error correction. This crucial usage of real-time error information, and feed-forward selection of distillation instances that pass the error detection, allows one to use input physical Bell pairs with much lower starting fidelity, and can achieve higher encoding rates. Moreover, it bypasses the decoding problem for random quantum codes, which may be computationally challenging. To conserve communication bandwidth and ensure that errors primarily arise from noisy physical Bell pair generation between different nodes, we inject each physical qubit state into a logical qubit (e.g. a surface code) using standard high-fidelity state injection techniques [29–31], which add only a negligible amount of noise in the process when the network is much noisier. This allows us to bound and ignore local operation errors in the process, thereby using the communication channel more efficiently.

Empirical Overhead and Order-Of-Magnitude Improvement Over Current Schemes

Following the proof of asymptotic constant rate, we examine and optimize our scheme numerically, directly refining the code sequence over the best possible parameters for small classical and quantum codes [32]. We find sequences of codes that require as few as 6 physical Bell pairs per logical Bell pair at 1% physical Bell pair error rates, almost an order of magnitude reduction compared to

conventional schemes (see Tab. I), with only a small increase in the required memory footprint. Our scheme also allows a continuous tradeoff between communication overhead and network buffer memory footprint, enabling the optimization of these parameters in a wide range of physical settings.

To optimize system resource allocation during the establishment of high-fidelity logical entanglement, we optimize the unencoding circuit and pipeline the different levels of entanglement distillation to minimize resource usage while providing high throughput. We find that in a wide range of parameter regimes of interest, particularly considering the fact that many local operations can typically be performed in parallel within a module, the distillation procedure itself does not cause a bottleneck, and our scheme compares favorably to other distillation schemes or lattice-surgery-based interconnect schemes. These methods are also directly compatible with recent advances in intra-module operations based on transversal gates [20, 33, 34].

Finally, we apply our results to distributed quantum computation, analyzing how the efficacy of quantum computation varies depending on the algorithm and quantum interconnect characteristic. We characterize the algorithmic requirements of intercore communication with the parameter β, which we define to be the average number of intercore logical entanglement operations per core required for each intracore logical circuit layer, and estimate this parameter for a variety of algorithms such as ripple-carry adders and random quantum circuits. Assuming one layer of intramodule logical operation takes time t_l, while an intermodule logical operation takes time $t_e\alpha$, we find that when $\beta t_e\alpha \geq t_l$, the network communication becomes the limiting factor. For typical operation parameters and conventional approaches to quantum interconnects, we find that the network communication is indeed limiting. However, the use of our constant-rate distillation scheme significantly alleviates this issue, leading to a better balance between intracore and intercore operations.

Taken together, our work provides over an order of magnitude performance improvement to the communication overhead for fault-tolerant distributed quantum computing, as well as the flexibility to adapt to a variety of settings with different communication rates and buffer memory sizes. We therefore believe that our results will be a key building block to future large-scale quantum computers.

Network error rate	0.1%	1%	5%	10%	15%
Distillation input error rate	0.35%	1.25%	5.2%	10.2%	15.2%
BDSW-2EPP scheme - overhead	64.2	66.8	295.3	348.5	1694
BDSW-2EPP scheme with Y basis - overhead	32.3	33.1	148.2	173.6	416.2
Lattice surgery - overhead	1,089	1,369	5,329	22,201	142,129
Constant-overhead distillation (buffer = 30) - overhead	4.7	6.2	14	33	63
Constant-overhead distillation (buffer = 50) - overhead	3.3	6.2	14	≤ 26	≤ 52
Constant-overhead distillation (buffer = 100) - overhead	2.5	4.4	≤ 11	≤ 23	≤ 46

TABLE I: Physical bell pair overhead per logical gate required for different schemes for distributed quantum computation. The values in the table are calculated using the gate error rate $p_{\text{gate}} = 0.1\%$. For all distillation schemes, we first use state injection, transforming the network error rate into the distillation input error rate. For lattice surgery, we instead directly operate a lattice surgery operation between two surface code patches across the link. Our scheme (bottom three rows) shows an order of magnitude overhead reduction compared to existing schemes.

[1] H. J. Kimble, Nature **453**, 1023 (2008), ISSN 1476-4687, URL http://dx.doi.org/10.1038/nature07127.

[2] C. H. Bennett and G. Brassard, Workshop on the theory and application of cryptographic techniques (1984).

[3] N. Gisin, G. Ribordy, W. Tittel, and H. Zbinden, Reviews of Modern Physics **74**, 145 (2002), ISSN 00346861, URL https://journals.aps.org/rmp/abstract/10.1103/RevModPhys.74.145.

[4] H. K. Lo, M. Curty, and K. Tamaki, Nature Photonics 2014 8:8 **8**, 595 (2014), ISSN 1749-4893, URL https://www.nature.com/articles/nphoton.2014.149.

[5] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L. M. Duan, and J. Kim, Physical Review A - Atomic, Molecular, and Optical Physics **89** (2012), URL http://arxiv.org/abs/1208.0391http://dx.doi.org/10.1103/PhysRevA.89.022317.

[6] H. Buhrman and H. Röhrig, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **2747**, 1 (2003), ISSN 16113349, URL https://link.springer.com/chapter/10.1007/978-3-540-45138-9_1.

[7] J. F. Fitzsimons, npj Quantum Information 2017 3:1 **3**, 1 (2017), ISSN 2056-6387, URL https://www.nature.com/articles/s41534-017-0025-3.

[8] L. J. Stephenson, D. P. Nadlinger, B. C. Nichol, S. An, P. Drmota, T. G. Ballance, K. Thirumalai, J. F. Goodwin, D. M. Lucas, and C. J. Ballance, Physical Review Letters **124**, 110501 (2020), ISSN 10797114, URL https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.124.110501.

[9] B. Jing, X. J. Wang, Y. Yu, P. F. Sun, Y. Jiang, S. J. Yang, W. H. Jiang, X. Y. Luo, J. Zhang, X. Jiang, et al., Nature Photonics 2019 13:3 **13**, 210 (2019), ISSN 1749-4893, URL https://www.nature.com/articles/s41566-018-0342-x.

[10] S. Ritter, C. Nölleke, C. Hahn, A. Reiserer, A. Neuzner, M. Uphoff, M. Mücke, E. Figueroa, J. Bochmann, and G. Rempe, Nature 2012 484:7393 **484**, 195 (2012), ISSN 1476-4687, URL https://www.nature.com/articles/nature11023.

[11] D. Hucul, I. V. Inlek, G. Vittorini, C. Crocker, S. Debnath, S. M. Clark, and C. Monroe, Nature Physics 2014 11:1 **11**, 37 (2014), ISSN 1745-2481, URL https://www.nature.com/articles/nphys3150.

[12] C. M. Knaut, A. Suleymanzade, Y.-C. Wei, D. R. Assumpcao, P.-J. Stas, Y. Q. Huan, B. Machielse, E. N. Knall, M. Sutula, G. Baranes, et al., arXiv preprint arXiv:2310.01316 (2023), URL https://arxiv.org/abs/2310.01316v1.

[13] M. Pompili, S. L. Hermans, S. Baier, H. K. Beukers, P. C. Humphreys, R. N. Schouten, R. F. Vermeulen, M. J. Tiggelman, L. dos Santos Martins, B. Dirkse, et al., Science **372**, 259 (2021), ISSN 10959203, URL https://www.science.org/doi/10.1126/science.abg1919.

[14] S. Meesala, D. Lake, S. Wood, P. Chiappina, C. Zhong, A. D. Beyer, M. D. Shaw, L. Jiang, and O. Painter, arXiv preprint arXiv:2312.13559 (2023), URL https://arxiv.org/abs/2312.13559v2.

[15] C. H. Bennett, G. Brassard, S. Popescu, B. Schumacher, J. A. Smolin, and W. K. Wootters, Physical Review Letters **76**, 722 (1996), ISSN 10797114, URL https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.76.722.

[16] C. H. Bennett, D. P. DiVincenzo, J. A. Smolin, and W. K. Wootters, Physical Review A **54**, 3824 (1996).

[17] A. G. Fowler, D. S. Wang, C. D. Hill, T. D. Ladd, R. V. Meter, and L. C. L. Hollenberg, Physical Review Letters **104**, 180503 (2010), URL https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.104.180503.

[18] J. Ramette, J. Sinclair, N. P. Breuckmann, and V. Vuleti´cvuleti´c, arXiv preprint arXiv:2302.01296 (2023), URL https://arxiv.org/abs/2302.01296v1.

[19] R. Acharya, I. Aleiner, R. Allen, T. I. Andersen, M. Ansmann, F. Arute, K. Arya, A. Asfaw, J. Atalaya, R. Babbush, et al., arXiv preprint arXiv:2207.06431 (2022), URL https://arxiv.org/abs/2207.06431v2http://arxiv.org/abs/2207.06431.

[20] D. Bluvstein, S. J. Evered, A. A. Geim, S. H. Li, H. Zhou, T. Manovitz, S. Ebadi, M. Cain, M. Kalinowski, D. Hangleiter, et al., Nature **626**, 58 (2024), ISSN 0028-0836, URL https://www.nature.com/articles/s41586-023-06927-3.

[21] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. P. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown, et al., Physical Review X **11** (2021), URL https:

//arxiv.org/abs/2107.07505v1.
[22] L. Egan, D. M. Debroy, C. Noel, A. Risinger, D. Zhu, D. Biswas, M. Newman, M. Li, K. R. Brown, M. Cetina, et al., Nature 2021 598:7880 **598**, 281 (2021), ISSN 1476-4687, URL https://www.nature.com/articles/s41586-021-03928-y.
[23] M. H. Abobeih, Y. Wang, J. Randall, S. J. H. Loenen, C. E. Bradley, M. Markham, D. J. Twitchen, B. M. Terhal, and T. H. Taminiau, Nature 2022 pp. 1–1 (2022), ISSN 1476-4687, URL https://www.nature.com/articles/s41586-022-04819-6.
[24] L. Postler, F. Butt, I. Pogorelov, C. D. Marciniak, S. Heußen, R. Blatt, P. Schindler, M. Rispler, M. Müller, and T. Monz, arXiv preprint arXiv:2312.09745 (2023), URL https://arxiv.org/abs/2312.09745v1.
[25] S. Krinner, N. Lacroix, A. Remm, A. Di Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann, et al., Nature 2022 605:7911 **605**, 669 (2022), ISSN 1476-4687, URL https://www.nature.com/articles/s41586-022-04566-8.
[26] V. V. Sivak, A. Eickbusch, B. Royer, S. Singh, I. Tsioutsios, S. Ganjam, A. Miano, B. L. Brock, A. Z. Ding, L. Frunzio, et al., arXiv preprint arXiv:2211.09116 (2022), URL https://arxiv.org/abs/2211.09116v1.
[27] Z. Ni, S. Li, X. Deng, Y. Cai, L. Zhang, W. Wang, Z.-B. Yang, H. Yu, F. Yan, S. Liu, et al., arXiv preprint arXiv:2211.09319 (2022), ISSN 14764687, URL https://arxiv.org/abs/2211.09319v1.
[28] H. Yamasaki and M. Koashi, arXiv preprint arXiv:2207.08826 (2022), URL https://arxiv.org/abs/2207.08826v2.
[29] Y. Li, New Journal of Physics **17**, 023037 (2015), ISSN 1367-2630, URL https://arxiv.org/abs/1410.7808v1https://iopscience.iop.org/article/10.1088/1367-2630/17/2/023037.
[30] L. Lao and B. Criger, ACM International Conference Proceeding Series pp. 113–120 (2022), URL https://dl.acm.org/doi/10.1145/3528416.3530237.
[31] C. Gidney, arXiv preprint arXiv:2302.12292 (2023), URL https://arxiv.org/abs/2302.12292v1.
[32] M. Grassl, *Bounds on the minimum distance of linear codes and quantum codes*, Online available at http://www.codetables.de (2007), accessed on 2023-12-19.
[33] M. Cain, C. Zhao, H. Zhou, N. Meister, J. Pablo, B. Ataides, A. Jaffe, D. Bluvstein, and M. D. Lukin, arXiv preprint arXiv:2403.03272 (2024), URL https://arxiv.org/abs/2403.03272v1.
[34] H. Zhou, C. Zhao, M. Cain, D. Bluvstein, C. Duckering, H.-Y. Hu, S.-T. Wang, A. Kubica, and M. D. Lukin, arXiv preprint arXiv:2406.17653 (2024), URL https://arxiv.org/abs/2406.17653v1.

Quantum Operating System Support for Quantum Trusted Execution Environments

Theodoros Trochatos
Yale University
New Haven, Connecticut, USA
theodoros.trochatos@yale.edu

Jakub Szefer
Yale University
New Haven, Connecticut, USA
jakub.szefer@yale.edu

Abstract

With the growing reliance on cloud-based quantum computing, ensuring the confidentiality and integrity of quantum computations is paramount. Quantum Trusted Execution Environments (QTEEs) have been proposed to protect users' quantum circuits when they are submitted to remote cloud-based quantum computers. However, deployment of QTEEs necessitates a Quantum Operating Systems (QOS) that can support QTEEs hardware and operation. This work introduces the first architecture for a QOS to support and enable essential steps required for secure quantum task execution on cloud platforms.

1 Introduction

Current cloud computing platforms such as IBM Quantum [2] or Amazon Braket [1] provide users access to quantum computers. However, today, cloud providers have full visibility and control over users' submitted quantum circuits. This can expose users to the risk of proprietary quantum algorithms being reverse-engineered or stolen by the providers. Even if the cloud providers are trusted, there are other threats such as malicious insiders [3, 5] who can gain access to the users' circuits and steal the information. To mitigate risks of security attacks in cloud-based quantum computation, Quantum Trusted Execution Environments (QTEEs) have been proposed [3–5]. Their goal is in general to prevent the cloud provider from knowing the operation executed by the user or the results. To achieve, this QTEEs apply some sort of obfuscation on the software level, the obfuscated or protected circuit is sent to the cloud provider, who cannot understand it, and only in the trusted hardware of the quantum processor can the circuit be de-obfuscated and executed. Currently, the missing piece of the technology is the Quantum Operating System (QOS) which can manage the user circuits and the QTEEs hardware. The QOS must manage secure loading of quantum circuits, execution, and transmission of computation results back to the users.

2 Design of QOS Support for QTEEs

This work outlines the mechanisms needed for securely loading quantum circuits, establishing a trusted environment for their execution, and returning the results to the users.

2.1 Existing QTEEs

There are already a number of QTEEs that have been proposed in the literature, which are briefly introduced below.

QC-TEE The QC-TEE work [5] introduced the idea of adding obfuscation to quantum circuits. While digital representation of the quantum circuits can be encrypted, the circuits are eventually transformed into analog pulses before execution on quantum hardware, analog pulses cannot be encrypted – but cloud provider can spy and attack these pulses. QC-TEE introduced hardware modifications to remove the dummy obfuscation pulses before they reach qubits. Encrypted metadata was used to allow QTEE hardware to determine which are the dummy obfuscation pulses.

SoteriaQ The SoteriaQ work [4] expanded the ideas of QC-TEE [5] and outlined detailed architecture of the circuit obfuscation. Encrypted metadata was again used to allow QTEE hardware to determine which are the dummy obfuscation pulses.

CASQUE The CASQUE work [3] introduced a new idea of extending obfuscation by swapping pulses between different control and drive channels. In the user's circuit, after transpilation, the control pulses would be swapped between different channels. On the quantum computer end, CASQUE introduced hardware modifications so that the pulses could be swapped back into the correct channels before they reach qubits. Encrypted metadata was used to allow CASQUE hardware to determine how to un-swap the control pulses.

2.2 Life-cycle of Quantum Circuit in QTEE

Regardless of the QTEE type, the lifecycle of a circuit in a QTEE follows three phases: I) secure loading of quantum circuits, II) execution on the quantum computing hardware, and III) transmission of computation results back to the users. Figure 1 outlines the life-cycle of a quantum circuit as handled by QOS.

Phase I: Secure Loading of Quantum Circuits On the user-end, the quantum circuit is obfuscated according to the target QTEE, and encrypted metadata is attached to the circuit. The obfuscated circuit and encrypted metadata are securely sent to the cloud provider, by an encrypted network connection. Upon decryption of the network packets, the circuit and encrypted metadata, needs to be safely stored

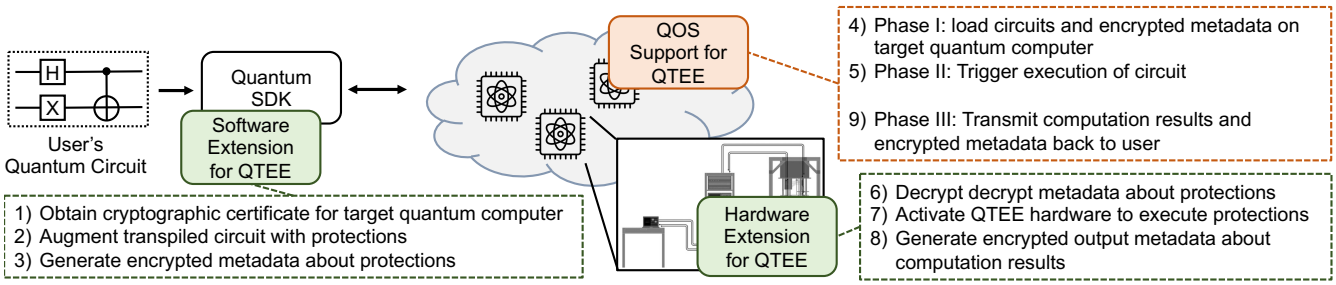


Figure 1. Lifecycle of quantum circuits and QOS support needed for QTEEs.

by QOS while awaiting execution. *The QOS needs to support classical, secure networking to receive users' circuits and their encrypted metadata. The QOS needs to track of the circuit and encrypted metadata once received. When storing them on the cloud, the circuit and encrypted metadata need to be associated with each other. Since the obfuscation method and encrypted metadata is specific to a particular quantum computer, the QOS scheduling also needs to be augmented to keep track of which quantum computer the circuit can execute on.*

Phase II: Secure Execution of Quantum Circuits When the circuit is ready to execute, the transpiled circuit is loaded onto the quantum controller, which, for example, in case of superconducting qubit quantum computer, generates the analog pulses that drive the qubits. In case of QC-TEE (5], SoteriaQ (4], and CASQUE (3], these pulses contain some form of obfuscation. Thus, in parallel the encrypted metadata has to be sent to the quantum computer, so it can decrypt it and operate on the input pluses according to the metadata. For example, for (4], some pulses are attenuated based on the metadata, while for (3], channels on which pulses are supposed to execute are swapped. *The QOS needs to ensure that the obfuscated circuits of the user are loaded in parallel to the encrypted metadata on the target quantum computer.*

Phase III: Transmission of Computation Results Back to the User For each shot of a circuit, it is measured and results returned to the user. Both QC-TEE (5] and SoteriaQ (4] proposed to randomly insert `X` gates at the end of the circuit to randomize the output. In parallel, the modified quantum computer hardware generates (and encrypts) its own metadata that can be used by the users to know which qubits' outputs were flipped by the `X`, so the users can recover the correct output. To support these operations, the QOS needs to keep track of the (encrypted) output metadata and transmit it back to the user along with circuit outputs. The transmission back to the user should use secure, classical networking. *The QOS needs to ensure that the circuit outputs and the output metadata are associated until they are returned to the user. The QOS needs to support classical, secure networking to send back users' results and their encrypted output metadata.*

3 Analysis of QOS Support for QTEE

The QOS modifications to support QTEE are minimal, and can be realized with no overhead on the computation (beyond the overheads of the specific QTEE hardware). Scheduling will be impacted by the QTEEs need that the circuit protection is specific to each quantum computer (because of the unique cryptographic keys needed for the encrypted metadata). QOS scheduler cannot move a circuit to a different quantum computer, since each circuit targets a specific backend. This is not a problem in the current NISQ era, as all circuits are transpiled to a specific back end. But in error corrected quantum computers, where a circuit can execute on different quantum computer backends, this will be a new constraint that the QOS needs to manage.

4 Conclusion

The role of the Quantum Operating System (QOS) in supporting Quantum Execution Environments (QTEEs) is crucial to the security of cloud-based quantum computing. By facilitating secure loading of quantum circuits, execution on the quantum computing hardware, and transmission of computation results back to the users, the QOS can enable robust protection for sensitive quantum computations without compromising performance.

Acknowledgements

This work was supported in part by NSF grant 2245344.

References

[1] Amazon Braket. https://aws.amazon.com/braket/.
[2] IBM Quantum. https://quantum.ibm.com/.
[3] Theodoros Trochatos, Sanjay Deshpande, Chuanqi Xu, Yao Lu, Yongshan Ding, and Jakub Szefer. Dynamic pulse switching for protection of quantum computation on untrusted clouds. In *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 404–414, 2024.
[4] Theodoros Trochatos, Chuanqi Xu, Sanjay Deshpande, Yao Lu, Yongshan Ding, and Jakub Szefer. Hardware architecture for a quantum computer trusted execution environment, 2023.
[5] Theodoros Trochatos, Chuanqi Xu, Sanjay Deshpande, Yao Lu, Yongshan Ding, and Jakub Szefer. A quantum computer trusted execution environment. *IEEE Computer Architecture Letters*, 22(2):177–180, 2023.

Design and Implementation of the Quantum Cloud Simulation Framework

Waylon Luo[1], Betis Baheri[1], Bo Fang[2], and Qiang Guan[1]

[1]Kent State University
[2]Pacific Northwest National Laboratory

***Abstract*—The increasing demand for quantum computing has exposed inefficiencies in current quantum cloud platforms, where users often face extensive wait times due to a lack of adequate simulation frameworks. Classical cloud simulators cannot model quantum cloud services due to the differences in properties of quantum tasks from that of classical tasks. To address this gap, we present QCloudSim, a specialized simulation framework designed to model the quantum clouds at an administrative level. QCloudSim offers access to the qubit configurations of quantum devices from leading providers such as IBM, Google, and Amazon. By utilizing a coarse-grained model, the framework simplifies complex quantum bits (qubits) interactions to avoid unnecessary computational overhead. Furthermore, QCloudSim supports parallel simulation through mpi4py, enhancing scalability and reducing execution times. That allows researchers and quantum cloud administrators to experiment with default and custom configurations, offering an efficient, flexible, and scalable platform for advancing quantum cloud research.**

I. Introduction

Quantum Computing (QC) is an emerging field with significant potential to solve problems [1] considered computationally infeasible for classical computers, with diverse applications in pharmacological discovery [2], financial analysis [3], optimization challenges [4], and machine learning [5], [6]. As quantum cloud systems grow in complexity, quantum cloud computing simulations become critical for research, development, and decision-making, assessing performance, cost efficiency, scalability, and security.

To address these challenges, we introduce QCloudSim, a discrete event simulation framework specifically designed to quantify quantum task scheduling and allocation policies based on the properties of tasks, configurations, and quantum devices. Recognizing that the configurations of quantum devices and allocation policy influence the fidelity and execution time of quantum tasks, QCloudSim enables researchers to assess their algorithms across different quantum bit (qubit) topologies without the need for direct experimentation on physical quantum devices [7]. The goal of quantum computing experiments is to understand and predict the behavior of real-world quantum computers and their future performance with mathematical models [8]. In such case, a simulation framework is an essential tool in designing and testing new qubit mapping mechanisms [9] on a large-scale quantum cloud environment.

QCloudSim offers researchers and quantum cloud administrators an efficient and scalable platform for quantum cloud simulations. It also allows them to experiment with default configurations or implement custom scheduling and allocation policies for quantum devices from leading quantum cloud providers.

II. System Design

The architecture layers of QCloudSim is illustrated in Figure 1. The framework is primarily implemented in Python. QCloudSim is designed by extending an open-source discrete-event simulation library for Python, SimPy, as a chain of discrete events, such as job arrivals, processes, yields, and device maintenance. SimPy manages concurrency and synchronization mechanisms to coordinate multiple processes in scalable simulations. QCloudSim layer contains the necessary modules to simulate a quantum cloud environment and manages to instantiate core entities such as QCloud, QDevice, Broker, and QJobs. The user's codes layer is where the framework allows users to define and implement their own task scheduling and allocation policies without needing to alter the core components. For large-scale parallel simulations, mpi4py (Message Passing Interface for Python) is integrated at the top-most layer of the framework. MPI distributes computational workloads across multiple processes, each simulating a different quantum device. This parallel execution improves efficiency and reduces wait times by enabling simultaneous simulations.

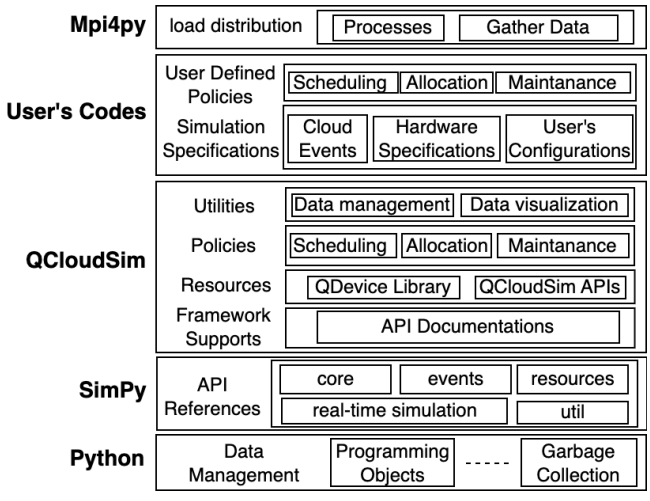


Figure 1: The architecture layers

III. Entities

The main component of QCloudSim is the SimPy simulation environment, where processes are defined and executed.

These processes represent entities or activities within the simulated system. It is necessary for users to instantiate core entities, including the quantum cloud (QCloud), quantum devices (QDevice), and Broker. Additionally, users are required to configure parameters such as simulation duration, task scheduling, allocation policies, job intervals, and maintenance schedules. A QCloud is an entity that contains at least one QDevice and a Broker. A QDevice contains machine profile, qubit topology, and maintenance schedule. Quantum jobs (QJobs) or tasks are generated by the job generator and passed to Broker. The Broker is responsible for assigning and allocating QJobs on QDevices. QJobs has properties of quantum tasks that are required to evaluate and estimate execution time and allocation. Figure 2 shows the core components of QCloudSim and the relations between entities.

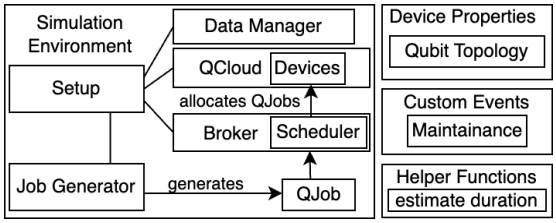


Figure 2: The core components of the simulation framework

IV. Use Cases

This section discusses QCloudSim's scalability and potential use cases. To maintain consistency in performance data, all simulations are conducted on a single Linux server. The system hosting QCloudSim uses an x86_64 hardware architecture, supporting both 32-bit and 64-bit CPU operation modes. It features a 48-bit address space for both physical and virtual addresses. The system has 32 CPUs and an AMD EPYC 7313 processor with 16 cores.

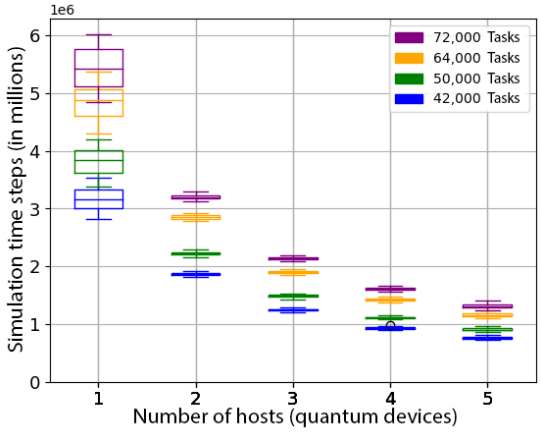


Figure 3: Sim-time to complete a fixed number of QJobs with varying available resources.

In the first demonstration, a fixed number of tasks (QJobs) is assigned to varying numbers of quantum computers, ranging from one to five devices. The simulation time steps are recorded for each experiment. Figure 3 shows that the simulation time steps decrease in logarithmic scales with increasing numbers of quantum devices. This suggests that the system scales efficiently with the addition of more quantum devices, reducing the overall time required for simulations as more devices are used to process a fixed number of tasks.

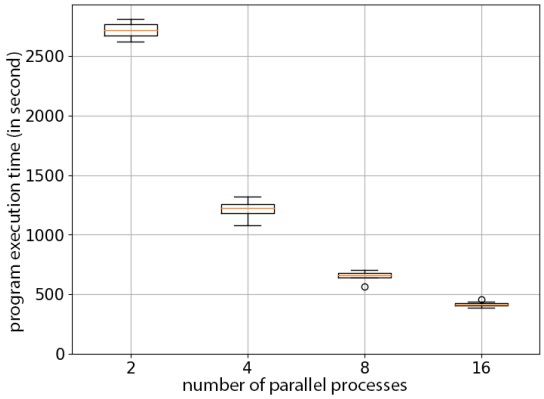


Figure 4: Comparison of execution times for simulations with varying numbers of processors.

MPI is implemented to execute independent simulations in parallel to mitigate the lengthy program execution time for large-scale simulations. A series of quantum cloud simulations is conducted using various number of parallel processes. As shown in Figure 4, increasing the number of processors exponentially reduces simulation time. In that way, the implementation of MPI significantly accelerates program execution.

Utilizing job traces and machine characteristics from real quantum clouds [10], QCloudSim can serve service providers as digital twin of their quantum clouds. This approach facilitates the development of larger quantum computers, as it enables QCloudSim users to model and configure quantum clouds by adjusting simulation parameters and device configurations.

V. Conclusion

QCloudSim addresses a significant and critical gap in the field of quantum cloud research by providing a specialized simulation framework dedicated to modeling quantum cloud systems within a controlled and reproducible environment. By abstracting the complex internal mechanisms of quantum devices and providing flexible configuration options, QCloudSim enables researchers to conduct thorough evaluations of quantum cloud systems without the need for costly and time-consuming experiments on physical hardware. Furthermore, the framework incorporates parallel processing capabilities through MPI, which significantly enhances both scalability and overall simulation performance. Ultimately, QCloudSim creates new opportunities for experimentation, exploration, and development in quantum cloud computing, thereby fostering the creation of more efficient and optimized quantum cloud services and contributing substantially to the advancement of the broader quantum computing ecosystem.

REFERENCES

[1] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018. [Online]. Available: http://dx.doi.org/10.22331/q-2018-08-06-79

[2] M. Zinner, F. Dahlhausen, P. Boehme, J. Ehlers, L. Bieske, and L. Fehring, "Quantum computing's potential for drug discovery: Early stage industry dynamics," *Drug Discovery Today*, vol. 26, no. 7, pp. 1680–1688, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1359644621002750

[3] P. Griffin and R. Sampat, "Quantum computing for supply chain finance," in *2021 IEEE International Conference on Services Computing (SCC)*, 2021, pp. 456–459.

[4] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn, A. Kandala, A. Mezzacapo, P. Müller, W. Riess, G. Salis, J. Smolin, I. Tavernelli, and K. Temme, "Quantum optimization using variational algorithms on near-term quantum devices," *Quantum Science and Technology*, vol. 3, no. 3, p. 030503, jun 2018. [Online]. Available: https://dx.doi.org/10.1088/2058-9565/aab822

[5] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, Oct 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1666-5

[6] E. P. DeBenedictis, "A future with quantum machine learning," *Computer*, vol. 51, no. 2, pp. 68–71, 2018.

[7] E. Altman, K. R. Brown, G. Carleo, L. D. Carr, E. Demler, C. Chin, B. DeMarco, S. E. Economou, M. A. Eriksson, K.-M. C. Fu, M. Greiner, K. R. Hazzard, R. G. Hulet, A. J. Kollár, B. L. Lev, M. D. Lukin, R. Ma, X. Mi, S. Misra, C. Monroe, K. Murch, Z. Nazario, K.-K. Ni, A. C. Potter, P. Roushan, M. Saffman, M. Schleier-Smith, I. Siddiqi, R. Simmonds, M. Singh, I. Spielman, K. Temme, D. S. Weiss, J. Vučković, V. Vuletić, J. Ye, and M. Zwierlein, "Quantum simulators: Architectures and opportunities," *PRX Quantum*, vol. 2, p. 017003, Feb 2021. [Online]. Available: https://link.aps.org/doi/10.1103/PRXQuantum.2.017003

[8] A. Hashim, L. B. Nguyen, N. Goss, B. Marinelli, R. K. Naik, T. Chistolini, J. Hines, J. Marceaux, Y. Kim, P. Gokhale *et al.*, "A practical introduction to benchmarking and characterization of quantum computers," *arXiv preprint arXiv:2408.12064*, 2024.

[9] L. Liu and X. Dou, "Qucloud: A new qubit mapping mechanism for multi-programming quantum computing in cloud environment," *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 167–178, 2021.

[10] G. S. Ravi, K. N. Smith, P. Gokhale, and F. T. Chong, "Quantum computing in the cloud: Analyzing job and machine characteristics," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, 2021, pp. 39–50.

Lego: QEC Decoding System Architecture for Dynamic Circuits

Yue Wu, Namitha Liyanage and Lin Zhong
Department of Computer Science, Yale University, New Haven, CT

1 Introduction

Quantum error correction (QEC) is a critical component of FTQC; the QEC decoding system is an important part of Classical Computing for Quantum or C4Q. Recent years have seen fast development in real-time QEC decoders [1–11]. Existing efforts to build real-time decoders have yet to achieve a critical milestone: decoding dynamic logical circuits with error-corrected readout and feed forward. Achieving this requires significant engineering effort to adapt and reconfigure the decoders during runtime, depending on the branching of the logical circuit.

We present a QEC decoding system architecture called Lego, with the ambitious goal of supporting dynamic logical operations. Lego employs a novel abstraction called the *decoding block* to describe the decoding problem of a dynamic logical circuit. Moreover, decoding blocks can be combined with three other ideas to improve the efficiency, accuracy and latency of the decoding system. First, they provide data and task parallelisms when combined with *fusion-based decoding* [6]. Second, they can exploit the pipeline parallelism inside multi-stage decoders. Finally, they serve as basic units of work for computational resource management.

Using decoding blocks, Lego can be easily reconfigured to support all QEC settings and to easily accommodate innovations in three interdependent fields: code [12, 13], logical operations [14–17] and qubit hardware [18–20]. In contrast, existing decoders are highly specialized to a specific QEC setting, which leads to redundant research and engineering efforts, slows down innovation, and further fragments the nascent quantum computing industry.

2 Lego Decoding System Architecture

We place the proposed Lego decoding system inside the classical computing part of FTQC as illustrated by Figure 1, with well-defined, technology-agnostic interfaces. The input to the quantum computer is a logical circuit specified using a programming language such as OpenQASM. Like a classical software program, the logical circuit consists of operations and conditionals. Due to the use of conditionals, the exact sequence of operations is only known at run-time. *The physical controller* directly talks to quantum hardware, generating control signals and receiving measurements, i.e., physical readouts, and sending them to the decoding system to compute the logical readout. *The logical controller* follows the logical circuit: it receives the logical readout from the decoding system and informs the physical controller and decoding system what logical operation is the next so that the latter can perform the operation and its QEC decoding,

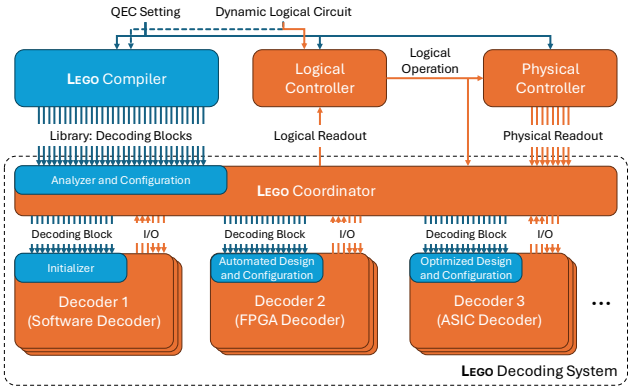


Figure 1. Lego decoding system and its interaction with other classical components of an FTQC. The orange and blue blocks represent online and offline components, respectively.

respectively. The *compiler* takes both the QEC setting and, optionally, the dynamic logical circuit (user program) as input, and generates decoding blocks that can be *merged* into any possible decoding graph of the logical circuit.

QEC decoding systems commonly take physical readout as input and output logical readout; Lego takes two additional inputs: (1) the logical operation being executed by the computer, which is known at runtime; and (2) decoding blocks for all logical operations, which are generated offline.

Lego itself is a specialized computer. Hardware-wise, it includes a collection of decoders of varying degrees of specialization. Software-wise, the coordinator functions as a resource manager or operating system that schedules/maps decoding blocks to decoders, potentially with support from the compiler.

3 Decoding Graph and Decoding Block

We introduced the notion of decoding graph in [21] in which an edge represents an error source and a vertex a detector [22], an XOR of a set of stabilizer measurements. Because an error source may impact more than two detector readings, edges can be hyperedges and the graph can be a hypergraph. Our key insight is that a decoding graph can precisely describe the decoding problem of many important classes of qubit codes [23, 24]. Many existing QEC decoders accept a decoding graph [6, 7, 25–28] as input. For a static logical circuit that does not contain any conditionals, one can statically generate the entire decoding graph for its decoding problem because there is a single execution path. For a dynamic logical circuit, there can be many possible execution paths, each with a different decoding problem. As the actual execution path is only known at runtime, its decoding graph

can only be constructed at runtime, raising a challenge to real-time decoding. Lego solves it with the abstraction of the decoding block.

Decoding Blocks: When a quantum computer executes a logical operation, the operation contributes to sources of errors (edges) and detector readings (vertices). These vertices and edges form a decoding graph $G_1 = (V_1, E_1)$ that describes the decoding problem contributed by this logical operation. The next logical operation in the execution may contribute another decoding graph $G_2 = (V_2, E_2)$. We call $B = V_1 \cap V_2$ the combination boundary between G_1 and G_2. B is non-empty if the two operations operate on the same qubit during the same QEC cycle. The decoding system must combine G_1 and G_2 at this boundary. For a dynamic logical circuit, the decoding system must combine such decoding graphs from logical operations at runtime, adding decoding latency.

We introduce a new abstraction called *decoding block*, or simply *block*, for each logical operation in a dynamic logical circuit. Let $O_i, i = 1, 2..., n$ denote the ith operation. $G_i = (V_i, E_i)$ denote the decoding graph it contributes. Its combination boundary with that of O_j is therefore $B_{ij} = V_i \cap V_j$. The block for O_i is defined as a tuple: $(G_i, B_i = \{B_{ij} = V_i \cap V_j, i \neq j\})$. That is, it includes O_i's decoding graph and all its combination boundaries with other operations in the same execution. An example is shown in Figure 2.

We say two blocks are of the same type if they have identical decoding graphs. Logical operations of the same type operating on the same set of qubits will produce blocks of the same type, no matter where they appear in the logical circuit. Blocks of the same type can share the same decoder, providing an opportunity for runtime optimization.

Generating Blocks: Importantly, given a logical circuit, all its blocks can be generated statically, i.e., offline. G_i can be generated based on the QEC setting, including the physical layout of logical qubits. To generate B_i, a compiler must enumerate all execution paths and derive the combination boundaries for the operation in each path. The compiler can change the granularity of the blocks, making trade-offs between a large number of small decoding blocks and a small number of large decoding blocks or finding an optimal combination of small and large. For example, the compiler can generate a single block for a series of operations in the same execution path. This block will have a large decoding graph but fewer combination boundaries for the decoding system to handle at runtime. We note that in general, small blocks provide finer granularity for computational parallelism and scheduling flexibility, at the cost of more runtime overhead due to combination.

Decoders for Blocks: In Lego, blocks are basic units of work and decoders are basic units of computational resource. A decoder can be completely programmable like a CPU core or FPGA. In this case, it can support any block as long as the necessary program is available. A decoder can also be specialized to support a specific decoding graph and therefore, a specific type of blocks. We can also imagine a more general decoder that can support decoding graphs of certain properties and therefore, support more types of blocks. What types of decoders to develop and include in Lego is an important task for the designer.

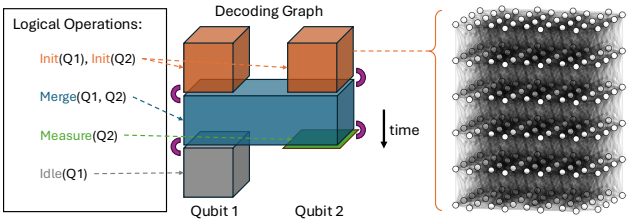


Figure 2. The QEC decoding problem of a logical circuit (left) can be described by a decoding graph (middle) as a combination of decoding blocks (right).

4 Decoding System Design

In this section, we elaborate on the potential design opportunities brought by decoding blocks.

Fusion-based Decoding: Decoding blocks conveniently support fusion-based decoding [6] as their decoding graphs can serve as the partitions with their combination boundaries being the fusion boundaries. With fusion-based decoding, each decoding block can be decoded independently, in parallel, and their results are then used to find a solution for the combined decoding graph efficiently without loss of accuracy. Fusion-based decoding was first supported for the MWPM decoder as a parallelization technique [6] and was recently generalized to the Union-Find decoder [29] and MWPF decoder [30]. We hypothesize that other QEC decoders can be adapted for fusion-based decoding. We also note that window decoding [12] can be used in place of the fusion operation [6] in fusion-based decoding, albeit less efficiently with redundant computation, less accuracy, and restricted boundary conditions [31–34].

Efficient Multi-Stage Decoding: Decoding blocks also support multi-stage decoding efficiently and flexibly. Multi-stage decoding combines multiple decoders by passing the output of one to the input of another, for better accuracy [35, 36] or reduced bandwidth [37]. However, multi-stage decoders are not suitable for low-latency decoding because a later stage cannot start until all earlier stages finish. Decoding blocks support pipeline parallelism inside multi-stage decoding with adaptive delayed scheduling.

Coordinator: Given a logical operation, the coordinator creates a decoding task, from its decoding block and assigns it to one of the many decoders. Because the quantum computer could execute multiple logical operations concurrently and not all decoders can execute all decoding blocks, the coordinator resembles the operating system of a heterogeneous classical computer. On the other hand, QEC decoding brings its own set of challenges due to the tight latency requirement.

Moreover, as the compiler generates the decoding blocks (and their granularity), it can inform and even collaborate with the coordinator for optimized resource management.

References

[1] Yosuke Ueno, Masaaki Kondo, Masamitsu Tanaka, Yasunari Suzuki, and Yutaka Tabuchi. QECOOL: On-line quantum error correction with a superconducting decoder for surface code. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021.

[2] Yosuke Ueno, Masaaki Kondo, Masamitsu Tanaka, Yasunari Suzuki, and Yutaka Tabuchi. QULATIS: A quantum error correction methodology toward lattice surgery. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022.

[3] Poulami Das, Aditya Locharla, and Cody Jones. LILLIPUT: a lightweight low-latency lookup-table decoder for near-term quantum error correction. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022.

[4] Ramon WJ Overwater, Masoud Babaie, and Fabio Sebastiano. Neural-network decoders for quantum error correction using surface codes: A space exploration of the hardware cost-performance tradeoffs. *IEEE Transactions on Quantum Engineering*, 3:1–19, 2022.

[5] Namitha Liyanage, Yue Wu, Alexander Deters, and Lin Zhong. Scalable quantum error correction for surface codes using fpga. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2023.

[6] Yue Wu and Lin Zhong. Fusion Blossom: Fast MWPM decoders for QEC. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2023.

[7] Oscar Higgott and Craig Gidney. Sparse Blossom: correcting a million errors per core second with minimum-weight matching. *arXiv preprint arXiv:2303.15933*, 2023.

[8] Ben Barber, Kenton M. Barnes, Tomasz Bialas, Okan Buğdaycı, Earl T. Campbell, Neil I. Gillespie, Kauser Johar, Ram Rajan, Adam W. Richardson, Luka Skoric, Canberk Topal, Mark L. Turner, and Abbas B. Ziad. A real-time, scalable, fast and highly resource efficient decoder for a quantum computer, 2023.

[9] Suhas Vittal, Poulami Das, and Moinuddin Qureshi. Astrea: Accurate quantum error-decoding via practical minimum-weight perfect-matching. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery.

[10] Narges Alavisamani, Suhas Vittal, Ramin Ayanzadeh, Poulami Das, and Moinuddin Qureshi. Promatch: Extending the reach of real-time quantum error correction with adaptive predecoding. *arXiv preprint arXiv:2404.03136*, 2024.

[11] Namitha Liyanage, Yue Wu, Siona Tagare, and Lin Zhong. Fpga-based distributed union-find decoder for surface codes. 2024.

[12] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[13] Sergey Bravyi, Andrew W Cross, Jay M Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627(8005):778–782, 2024.

[14] A Yu Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003.

[15] Daniel Litinski. Magic state distillation: Not as costly as you think. *Quantum*, 3:205, 2019.

[16] Hengyun Zhou, Chen Zhao, Madelyn Cain, Dolev Bluvstein, Casey Duckering, Hong-Ye Hu, Sheng-Tao Wang, Aleksander Kubica, and Mikhail D Lukin. Algorithmic fault tolerance for fast quantum computing. *arXiv preprint arXiv:2406.17653*, 2024.

[17] Craig Gidney, Noah Shutty, and Cody Jones. Magic state cultivation: growing t states as cheap as cnot gates. *arXiv preprint arXiv:2409.17595*, 2024.

[18] Dolev Bluvstein, Simon J Evered, Alexandra A Geim, Sophie H Li, Hengyun Zhou, Tom Manovitz, Sepehr Ebadi, Madelyn Cain, Marcin Kalinowski, Dominik Hangleiter, et al. Logical quantum processor based on reconfigurable atom arrays. *Nature*, 626(7997):58–65, 2024.

[19] Rajeev Acharya, Laleh Aghababaie-Beni, Igor Aleiner, Trond I Andersen, Markus Ansmann, Frank Arute, Kunal Arya, Abraham Asfaw, Nikita Astrakhantsev, Juan Atalaya, et al. Quantum error correction below the surface code threshold. *arXiv preprint arXiv:2408.13687*, 2024.

[20] Ben W Reichardt, David Aasen, Rui Chao, Alex Chernoguzov, Wim van Dam, John P Gaebler, Dan Gresh, Dominic Lucchetti, Michael Mills, Steven A Moses, et al. Demonstration of quantum computation and error correction with a tesseract code. *arXiv preprint arXiv:2409.04628*, 2024.

[21] Yue Wu, Namitha Liyanage, and Lin Zhong. An interpretation of union-find decoder on weighted graphs. *arXiv preprint arXiv:2211.03288*, 2022.

[22] Craig Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, 2021.

[23] David Kribs, Raymond Laflamme, and David Poulin. Unified and generalized approach to quantum error correction. *Physical review letters*, 94(18):180501, 2005.

[24] Matthew B Hastings and Jeongwan Haah. Dynamically generated logical qubits. *Quantum*, 5:564, 2021.

[25] David Poulin and Yeojin Chung. On the iterative decoding of sparse quantum codes. *arXiv preprint arXiv:0801.1241*, 2008.

[26] Nicolas Delfosse, Vivien Londe, and Michael E Beverland. Toward a union-find decoder for quantum LDPC codes. *IEEE Transactions on Information Theory*, 2022.

[27] Yue Wu, Lin Zhong, and Shruti Puri. Hypergraph minimum-weight parity factor decoder for qec. *Bulletin of the American Physical Society*, 2024.

[28] Nicolas Delfosse and Naomi H Nickerson. Almost-linear time decoding algorithm for topological codes. *Quantum*, 2021.

[29] Namitha Liyanage, Yue Wu, Emmet Houghton, and Lin Zhong. Multi-fpga system for quantum error correction with lattice surgery. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2024.

[30] Liu Yang, Yue Wu, and Lin Zhong. Parallel minimum-weight parity factor decoding for quantum error correction. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2024.

[31] Aravind R Iyengar, Marco Papaleo, Paul H Siegel, Jack Keil Wolf, Alessandro Vanelli-Coralli, and Giovanni E Corazza. Windowed decoding of protograph-based ldpc convolutional codes over erasure channels. *IEEE Transactions on Information Theory*, 58(4):2303–2320, 2011.

[32] Xinyu Tan, Fang Zhang, Rui Chao, Yaoyun Shi, and Jianxin Chen. Scalable surface code decoders with parallelization in time. *PRX Quantum*, 2022.

[33] Luka Skoric, Dan E Browne, Kenton M Barnes, Neil I Gillespie, and Earl T Campbell. Parallel window decoding enables scalable fault tolerant quantum computation. *Nature Communications*, 2023.

[34] Héctor Bombín, Chris Dawson, Ye-Hua Liu, Naomi Nickerson, Fernando Pastawski, and Sam Roberts. Modular decoding: parallelizable real-time decoding for quantum computers. *arXiv preprint arXiv:2303.04846*, 2023.

[35] Oscar Higgott, Thomas C Bohdanowicz, Aleksander Kubica, Steven T Flammia, and Earl T Campbell. Fragile boundaries of tailored surface codes and improved decoding of circuit-level noise. *arXiv preprint arXiv:2203.04948*, 2022.

[36] Cody Jones. Improved accuracy for decoding surface codes with matching synthesis. *arXiv preprint arXiv:2408.12135*, 2024.

[37] Nicolas Delfosse. Hierarchical decoding to reduce hardware requirements for quantum computing. *arXiv preprint arXiv:2001.11427*, 2020.

A Case for OS-Managed Resource Pools in Fault-Tolerant Quantum Computers

Suhas Vittal
suhaskvittal@gatech.edu
Georgia Institute of Technology
Atlanta, GA, USA

Moinuddin Qureshi
moin@gatech.edu
Georgia Institute of Technology
Atlanta, GA, USA

Abstract

Quantum error correction is the most promising approach for realizing quantum advantage. Unfortunately, the surface code, the most promising error correction code, is inefficient. Thus, recent work has proposed a register-memory paradigm, where surface codes as used for compute, and efficient QLDPC codes are used as a main memory. In this abstract, we examine the resource demands of quantum applications under this paradigm and argue for shared resource pools.

1 Introduction

Quantum error correction remains the most promising path forward for realizing promising applications and achieving quantum advantage [2, 9, 13]. Currently, virtually all proposals for *Fault-Tolerant Quantum Computers (FTQCs)*, which use error correction to handle errors during program execution, use the *surface code*, which is widely considered to be the most promising error correction code [5, 6, 8]. However, for many practical applications, FTQCs using the surface code are expected to require millions of physical qubits [7].

Thus, *Quantum Low Density Parity Check (QLDPC)* codes have emerged as more efficient alternative to the surface code. Unlike the surface code, which encodes a single logical qubit no matter how many physical qubits are used, QLDPC codes encode multiple logical qubits into a *logical block*: larger logical blocks will have more logical qubits and higher code distances. Thus, QLDPC codes are orders of magnitude more efficient than the surface code. However, it remains unclear how to perform basic quantum gates on single logical qubits within a logical block.

Thus, recent research has argued for operating QLDPC codes as a *main memory*, where surface code *registers* retrieve program qubits from the memory and perform computation [1, 3, 4, 14]. Under this paradigm, a quantum program executing on an FTQC will split its program memory between surface code registers and QLDPC memory. However, to perform logical operations on and between program qubits, the FTQC must also provide the program (1) routing space to execute CX gates between program qubits and (2) magic state factories, which produce magic states that are consumed to perform T gates. Ideally, quantum programs should receive as many resources as possible to avoid slowdowns, which can occur due to register file misses or insufficient magic state production. Unfortunately, resources will be scarce for the next few decades. Ideally, applications should be given minimal resources that enable them to run without much performance degradation. In this abstract, we explore the resource demands of FTQC applications.

2 Application Resource Demands

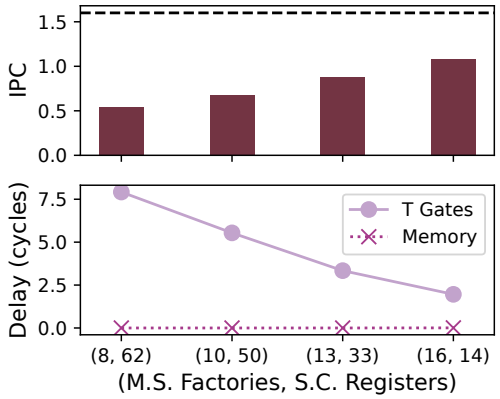


Figure 1. Performance of `shor_n62` with different configs.

In this section, we study two applications, a 62-qubit factoring benchmark (`shor_n62`) and 177-qubit binary-welded tree benchmark (`bwt_n177`), and the tradeoffs between allocating resources towards magic state production[1] and surface code registers. To evaluate the impact of different configurations, we use an event-driven simulator to measure (1) *IPC*[2] and (2) operation delay for 10M gates.

2.1 Results

shor_n62: Figure 1 shows the the performance and operation delays of `shor_n62` for different configurations. We observe that for the $(16, 14)$ configuration, where about 23% of the program memory is readily accessible, `shor_n62` experiences practically no memory access delay. We attribute this to the application having many R_Z gates, which unroll into 100s or 1000s of single-qubit gates, most of which are T gates. Thus, the performance of `shor_n62` mostly depends on magic state production rather than register file size. We term such applications as *magic-state-bound*.

bwt_n177: Figure 2 shows the performance and T gate latency of `bwt_n177` for different configurations. Unlike

[1] Our evaluations assume each magic state must be distilled twice.
[2] *IPC* is a commonly-used metric in architecture research that describes the number of instruction/gates completed per (logical) cycle.

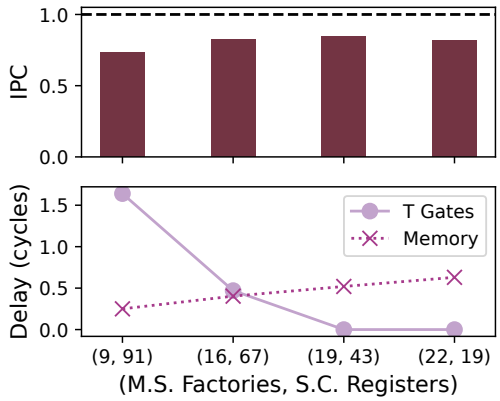


Figure 2. Performance of `bwt_n177` with different configs.

`shor_n62`, the performance of `bwt_n177` saturates at $IPC = 0.8$, which is about a 25% slowdown relative to an idealized $IPC = 1$. We observe that performance plateau is due to memory accesses. For instance, in the $(22, 19)$ configuration, there is no T gate delay, but there are memory access delays (about 0.6 cycles per gate). Hence, the only way to improve the performance of `bwt_n177` is by increasing the register file size: we term such applications as *memory-bound*.

3 Shared Resource Management

The above evaluations demonstrate that quantum applications can have different application requirements: they are either magic-state-bound or memory-bound. Magic-state-bound applications do not need many registers, whereas memory-bound applications do not have extremely high magic state production. Given these extremes, a shared resource pool is the best way to optimize resource usage across multiple applications. Shared resource pools are beneficial for two reasons:

1. QLDPC memory blocks often have *internal fragmentation* or unused logical qubits, which is more severe for denser codes. A shared memory pool minimizes the impacts of fragmentation.
2. All applications require magic states, and thus, having shared magic state factories can enable proportional resource allocation depending on application needs.

The alternative to a shared resource pool is having users request private application resources. Unfortunately, this inevitably results in *resource stranding*, or unused and inaccessible resources, as users typically request more resources than they need for their applications. Resource stranding is a significant source of inefficiency in classical datacenters [11], and is harmful to FTQCs as resources are already scarce.

3.1 A Case for OS-Managed Shared Resources

But *who* should manage the shared resources? Historically, quantum programs have been *compiler-driven*, in that the compiler determines what gates are executed, when they are executed, and how they are executed [10, 12, 15]. Thus, the compiler would be the natural choice. Unfortunately, existing compilers for FTQC programs are rather slow and are limited to programs with a few thousands of gates [12, 15]. Figure 3 shows the compilation latency for QFT benchmarks for a recently published surface code compiler [15]. These trends in compilation latency suggest that adding the complex task of arbitrating resources between multiple programs might overwhelm existing compilers.

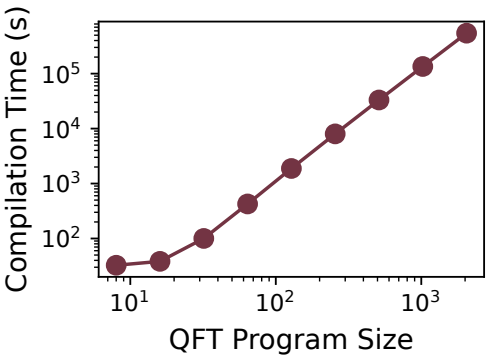


Figure 3. QFT program compilation latency.

Thus, we argue that this shared resource pool should be managed by a *kernel* (OS) running on the quantum control processor, which dictates program execution. An OS-managed resource pool is beneficial for the compiler, as the compiler can request resources from the pool instead of managing the pool itself, and for the vendor, who can ensure the OS meets users' *Quality of Service (QoS)*.

We propose a straightforward OS design with three main components; we discuss each component below. (1) a routing service, which manages routing space, (2) a memory manager, which manages the surface code register file and QLDPC main memory, and (3) a magic state daemon, which continously produces magic states. We discuss the high-level design of each component below.

Routing Service. The routing service allocates routing space for programs on demand.

Memory Manager. The memory manager retrieves program qubits for applications. If a request misses in the register file, the program qubit, retrieved from QLDPC memory, is swapped with a *victim* qubit in the register file. Consequently, the memory hierarchy is exclusive.

Magic State Daemon. The magic state daemon produces magic states constantly and writes them to a buffer. If the buffer becomes full, factories are deallocated to make space for more distilled magic states. Once the buffer empties sufficiently, factories are reallocated.

4 Conclusion

Quantum error correction is the most promising method of realizing quantum advantage. Still, due to the overheads

of the surface code, it is expected to require tens of millions of physical qubits. To reduce these overheads, recent work has proposed a hybrid where surface codes are used as registers, and QLDPC codes are used as memory [14]. In this abstract, we discuss the challenges with adequately allocating resources to quantum programs under this model: we identify that programs are either *magic-state-bound* or *memory-bound*. We further argue for OS-managed shared resource pools and present a basic skeleton for such an OS.

References

[1] Sergey Bravyi, Andrew W Cross, Jay M Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *arXiv preprint arXiv:2308.07915*, 2023.

[2] Andrew M. Childs, Dmitri Maslov, Yunseong Nam, Neil J. Ross, and Yuan Su. Toward the first quantum simulation with quantum speedup. *Proceedings of the National Academy of Sciences*, 115(38):9456–9461, sep 2018.

[3] Lawrence Z Cohen, Isaac H Kim, Stephen D Bartlett, and Benjamin J Brown. Low-overhead fault-tolerant quantum computing using long-range connectivity. *Science Advances*, 8(20):eabn1717, 2022.

[4] Alexander Cowtan. Ssip: automated surgery with quantum ldpc codes. *arXiv preprint arXiv:2407.09423*, 2024.

[5] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, Sep 2002.

[6] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[7] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, apr 2021.

[8] A Yu Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191, dec 1997.

[9] Ian D. Kivlichan, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Wei Sun, Zhang Jiang, Nicholas Rubin, Austin Fowler, Alá n Aspuru-Guzik, Hartmut Neven, and Ryan Babbush. Improved fault-tolerant quantum simulation of condensed-phase correlated electrons via trotterization. *Quantum*, 4:296, jul 2020.

[10] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, pages 1001–1014, 2019.

[11] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. Pond: Cxl-based memory pooling systems for cloud platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, page 574–587, New York, NY, USA, 2023. Association for Computing Machinery.

[12] Abtin Molavi, Amanda Xu, Swamit Tannu, and Aws Albarghouthi. Compilation for surface code quantum computers. *arXiv preprint arXiv:2311.18042*, 2023.

[13] Peter W Shor. Scheme for reducing decoherence in quantum computer memory. *Physical review A*, 52(4):R2493, 1995.

[14] Joshua Viszlai, Willers Yang, Sophia Fuhui Lin, Junyu Liu, Natalia Nottingham, Jonathan M Baker, and Frederic T Chong. Matching generalized-bicycle codes to neutral atoms for low-overhead fault-tolerance. *arXiv preprint arXiv:2311.16980*, 2023.

[15] George Watkins, Hoang Minh Nguyen, Keelan Watkins, Steven Pearce, Hoi-Kwan Lau, and Alexandru Paler. A high performance compiler for very large scale surface code computations. *Quantum*, 8:1354, 2024.

A Scalable Quantum Circuit Knitting Framework via Parallel and Hardware-Efficient Circuit Cutting

Xiangyu Ren[1], Mengyu Zhang[2], Antonio Barbalace[1]
[1]The University of Edinburgh, Edinburgh, UK
{xiangyu.ren,abarbala}@ed.ac.uk
[2]Tencent Quantum Lab, Shenzhen, China
mengyuzhang@tencent.com

1 Introduction

Quantum computing has the potential to solve many classically intractable problems. However, the limited number of available qubits in current Noisy Intermediate Scale Quantum (NISQ) devices hinders quantum computing from realizing its initially anticipated computational power.

Circuit knitting emerges as a promising technique to overcome the limitation of the few physical qubits in near-term quantum hardware by cutting large quantum circuits into smaller subcircuits. These subcircuits can be executed on different quantum computers that are connected by classical communication channels. This is suitable for quantum computing data centers or modular quantum computer solutions, while subcircuits can be distributed on heterogeneous quantum hardware and executed simultaneously, and benefit from the acceleration of parallel quantum computing.

Although current circuit knitting frameworks have been proven to extend the scalability of quantum computers, there exists challenges that prevent circuit knitting technique from being feasible to execute large scale quantum program on real hardware: 1) Exponentially high sampling overhead for each subcircuit; 2) Exponential post-processing computation for recombining result of subcircuits; 3) Incompatible to the qubit layout in current quantum computers.

To address the above challenges, we design a scalable circuit knitting framework that reduces the overheads by cutting a group of non-local quantum gates in parallel. Also, it leverages the qubit layout information of heterogeneous quantum computers, providing a circuit cutting solution that reduces the compilation overhead when subcircuits are executed on specific hardware. In addition, we utilize the tensor network contraction approach to accelerate classical post-processing. The detail of our designs are elaborated in the following sections after a brief background section.

2 Background

Quantum circuit knitting [2, 5, 7, 8] has been put forward as a technique to solve this problem by running large circuits with more qubits than physically available. The theory behind circuit knitting is quasiprobability simulation [6]: the expected value of the original circuit is obtained by sampling each smaller part of it that we define as subcircuit. Based on the quasiprobability simulation, previous works provide several approaches to partition the quantum circuit into smaller ones. Specifically, this is realized by decomposing a *qubit wire* or a *2-qubit gate* into a set of *single qubit operations*, followed by classical postprocessing to reconstruct the expectation value of the original circuit.

Fig. 1 shows a generalization of previous circuit knitting frameworks, e.g., CutQC [11]. We have our input circuit as OpenQASM

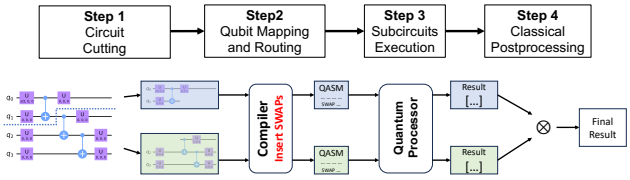


Figure 1: Generalization of previous knitting frameworks.

format. First, the original circuit is cut along qubit wires (wire cutting) or 2-qubit gates (gate cutting, not in Figure) to form separated subcircuits. Then, subcircuits are individually compiled and executed on quantum processors. Finally, classical postprocessing: individual execution results are reconstructed into the result of the original circuit.

3 Parallel Circuit Cutting

One of the primary overheads for circuit knitting is the sampling overhead for subcircuits. Sampling overhead is the number of shots each subcircuit needs to run in order to grantee the accuracy of circuit knitting. Generally, the sampling overhead $O(\gamma^{2n})$ scales exponential with the number of cuts n. In example, cutting a non-local CX gate has $\gamma = 3$ for *gate cut* and $\gamma = 4$ for *wire cut* [3, 8].

Recently, several theoretical works emerge with the idea of cutting multiple quantum gates in parallel [10], to minimize the overall cutting overhead of a quantum circuit. Schmitt et al. [10] proposed the parallel cutting technique that is proven theoretically to improve the sampling overhead of subcircuits over an exponential scale. For an arbitrary two-qubit unitary quantum gate U, it can be performed the KAK decomposition

$$U = (V_1 \otimes V_2)(\sum_{k=0}^{3} u_k \sigma_k \otimes \sigma_k)(V_3 \otimes V_4) \tag{1}$$

while $\sigma_0 = I$ and $\sigma_1, \sigma_2, \sigma_3$ are the Pauli matrices, and V_i are single-qubit unitaries. With such unitary U we have

$$\gamma(U) = 1 + 2\Delta_U = 1 + 2 \sum_{k \neq k'} |u_k||u'_k| \tag{2}$$

as the parameter γ of sampling overhead.

Above is the overhead for a single gate cutting, however for cutting n multiple unitary gates $U^{\otimes n}$ in parallel, the cutting overhead $\gamma(U^{\otimes n})$ could be improved over cutting these gates solely:

$$\gamma(U^{\otimes n}) = 2(1 + \Delta_U)^n - 1 < (1 + 2\Delta_U)^n = \gamma(U)^n \tag{3}$$

On the left of the equation is the overhead for parallel cutting, while the right side is overhead for single cuttings. Further theoretical details can be found in the research [10]. Hence, leveraging such

parallel gate cutting technique can reduce the overall subcircuits sampling overhead in circuit knitting.

In spite of the benefits from parallel cutting, it would be non-trivial to integrate it into the circuit knitting framework. The sampling overhead parameter γ is related with both the type and the location of the unitaries inside a circuit, hence there exists chances to optimize the original circuit via quantum circuit transformation, leading to a lower cutting overhead.

Specifically, here are the potential challenges, along with the opportunities:

(1) Compared to previous circuit knitting frameworks which only consider about cutting CX gate, our framework leveraging parallel cutting have to consider the type of quantum gate to cut. That is because in the case of parallel cutting, overall sampling overhead γ is related to the characteristic of gate. Compared to previous framework, which simply optimize the number of gate cuttings, our framework need to find the lowest-overhead cuttings by utilizing gate information.
(2) The location of gates to be cut also matters. Given the theory of parallel gate cutting in theoretical work [10], cutting a group of adjacent gates in the circuit require no extra resources, while cutting several gates that is interleaved with other local gates would induces extra ancillary qubits for gate-base teleportation. Hence it calls for a trade-off between ancillary qubit resource and sampling overhead.

To address above challenges, we propose the circuit optimization approached based on ZX-calculus [4]. We represent the original circuit in the intermediate representation form in ZX-Calculus, and reconstruct the circuit with consideration of gate type and location that minimize the sampling overhead while cutting the circuit.

4 Hardware-Aware Circuit Cutting

Circuit knitting is costly as the overhead of subcircuits sampling and classical postprocessing scale exponentially with the number of cuts [8]. Hence, to lower such overheads, previous research focuses on minimizing the number of cuts [11, 12]. Other research involving cluster-based hardware [1] also focus on minimizing the number of cuts.

Despite their efficiency in minimizing the number of cuts, the follow-on compilation may counteract the improvements. This is because during compilation a large number of SWAPs may be inserted in the qubit routing stage, which likely extends the depth of each subcircuit and undermines the fidelity.

The depth increase is caused by neglecting hardware information during circuit cutting, which makes the routing of the resulting subcircuits onto actual devices challenging. In fact, in previous works, the SWAPs insertion remains unknown till we finish the circuit cutting procedure and start compilation, which makes the final result vulnerable to mapping and routing overheads.

Hardware-aware Circuit Cutting. Instead of investigating new qubit mapping and routing methods – widely studied in recent years, we proposes to *incorporate* the routing problem into the circuit cutting [9]. This is in sharp contrast to previous works, where such steps execute sequentially, and independently. Our idea is to improve the quality of circuit cutting by considering the routing overhead on the actual hardware during cutting.

Based on this idea, we exploit the fact that graph similarity between **a subcircuit's qubit interaction graph** and **hardware layout** can well predict the number of SWAPs for qubit routing. Setting graph similarity as an optimization objective, we can iteratively improve the quality of circuit cutting. These steps are briefly detailed below.

(1) **Circuit Cutting:** Our algorithm co-optimizes two objectives: 1) minimizing the number of gate cuts, 2) maximizing the subcircuit graph similarity to the hardware-layout of a quantum processor. Graph similarity serves as a "heuristic clue" to reduce SWAPs before the qubit mapping & routing step. After dividing logical qubits into different subcircuits, we insert those local single-qubit operations for replacement using Qiskit.
(2) **Compilation and Execution:** Each subcircuit is compiled through qubit mapping & routing, where SWAPs insertion are actually performed. Then subcircuits are individually executed on one or more quantum processor. This step is unchanged in our framework, easily integrating in existent frameworks.
(3) **Classical Postprocessing:** Finally, we also implement an efficient module for reconstructing results during classical postprocessing. The classical postprocessing can be represented by *tensor network contraction*, and a contraction tree optimizer is utilized to find the optimal sequence for contraction.

Apart from quantum circuit knitting, our design is also beneficial for cutting circuits in other distributed quantum computing scenarios [13], e.g. chiplet architecture [14].

5 Accelerating Classical Postprocessing

Optimizing Recombination Protocol. Classical postprocessing reconstructs the result of the original circuit by merging subcircuits result using Kronecker products. Specifically, the basic protocol to merge results of two subcircuits is defined in Fig. 2. Note that the third term in the protocol will expand into 2×4 sub-terms (each corresponds to a Kronecker product) in respect to each value combination of the coefficients $\alpha_1\alpha_2$. Due to our observation that each coefficient α_i is only related to one side of the result in these sub-terms, we can split the combined coefficient $\alpha_1\alpha_2$ on each side, and the following Kronecker product would do recombination for them. Thus, on each side of the result, the coefficient will calculate with subcircuit results ahead of Kronecker product, shrinking 4 sub-terms into one. Our preprocessing on the coefficients improve the number of Kronecker products of each merging from 10 ($= 2+2\times4$) to 4, shown in Fig. 2.

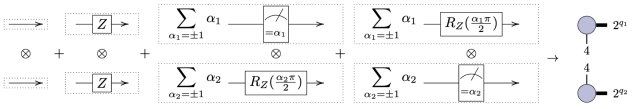


Figure 2: Classical postprocessing: merging two subcircuits result using tensor contraction. The notation on the right is a tensor network notation, representing the contraction of these two subcircuit result.

Acceleration with Tensor Contraction. Furthermore, the merging operation in Fig. 2 is identical to the operation of tensor contraction. Kronecker product is a special version of tensor product when we group the q indices each with the 2-dimension, into one single index with 2^q-dimension (q=num of qubits). Assume that in both

upper and lower side of Fig. 2, the 4 result vectors (each with 2^{q_1} or 2^{q_2}-dimension) can be seen as tensor in 4×2^q shape, the whole merging operation is abstracted into the tensor contraction on the right of Fig. 2. Correspondingly, if there are n gate cuts between two subcircuits, the result tensor of each side would be in shape $4_1 \times \cdots \times 4_n \times 2^q$, who has n legs with 4-dimension and one leg with 2^q-dimension in tensor network notation. With such abstraction, we can translate the classical postprocessing problem into a tensor network to searching a better solution.

6 Conclusion

We propose a framework that leverages parallel gate cutting, hardware-aware cutting and tensor network contraction-based postprocessing, trying to address the corresponding overhead – sampling overhead, compilation overhead and classical postprocessing overhead, in current circuit knitting framework.

References

[1] Jonathan M. Baker, Casey Duckering, Alexander Hoover, and Frederic T. Chong. Time-sliced quantum circuit partitioning for modular architectures. CF '20, page 98–107. ACM, 2020.

[2] Sergey Bravyi, Graeme Smith, and John A. Smolin. Trading classical and quantum computational resources. *Phys. Rev. X*, 6:021043, Jun 2016.

[3] Lukas Brenner, Christophe Piveteau, and David Sutter. Optimal wire cutting with classical communication. *arXiv preprint arXiv:2302.03366*, 2023.

[4] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John Van De Wetering. Graph-theoretic simplification of quantum circuits with the zx-calculus. *Quantum*, 4:279, 2020.

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

[6] Hakop Pashayan, Joel J. Wallman, and Stephen D. Bartlett. Estimating outcome probabilities of quantum circuits using quasiprobabilities. *Phys. Rev. Lett.*, 115:070501, Aug 2015.

[7] Tianyi Peng, Aram W. Harrow, Maris Ozols, and Xiaodi Wu. Simulating large quantum circuits on a small quantum computer. *Phys. Rev. Lett.*, 125:150504, Oct 2020.

[8] Christophe Piveteau and David Sutter. Circuit knitting with classical communication. *IEEE Transactions on Information Theory*, pages 1–1, 2023.

[9] Xiangyu Ren, Mengyu Zhang, and Antonio Barbalace. A hardware-aware gate cutting framework for practical quantum circuit knitting. *arXiv preprint arXiv:2409.03870*, 2024.

[10] Lukas Schmitt, Christophe Piveteau, and David Sutter. Cutting circuits with multiple two-qubit unitaries. *arXiv preprint arXiv:2312.11638*, 2023.

[11] Wei Tang, Teague Tomesh, Martin Suchara, Jeffrey Larson, and Margaret Martonosi. CutQC: using small quantum computers for large quantum circuit evaluations. ASPLOS '21, page 473–486. ACM, 2021.

[12] Teague Tomesh, Zain H. Saleem, Michael A. Perlin, Pranav Gokhale, Martin Suchara, and Margaret Martonosi. Divide and conquer for combinatorial optimization and distributed quantum computation, 2021.

[13] Anbang Wu, Hezi Zhang, Gushu Li, Alireza Shabani, Yuan Xie, and Yufei Ding. Autocomm: A framework for enabling efficient communication in distributed quantum programs. MICRO-55, pages 1027–1041, 2022.

[14] Hezi Zhang, Keyi Yin, Anbang Wu, Hassan Shapourian, Alireza Shabani, and Yufei Ding. Compilation for quantum computing on chiplets. *arXiv preprint arXiv:2305.05149*, 2023.

NSF Workshop on Quantum Operating Systems and Real-Time Control, 2024

Understanding Real Time Decoding for Photonic Quantum Computers

Avinash Kumar[1], Eneet Kaur[2], and Poulami Das[1]

[1]The University of Texas at Austin
[2]CISCO Research

1. Problem: Photonic Quantum Computers Require Fault-Tolerance

Quantum computers promise substantial speedup over conventional machines for many crucial applications [16, 19]. Photonic qubits are emerging as a promising hardware technology due to their great scalability, long coherence times and easy integration with quantum networks. There has been rapid advancements in this domain over the last few years, both at device [14, 17, 21, 24] and software levels [22, 23].

Photonic systems comprise of a device called *Resource State Generator (RSG)* that generates photons continuously every clock cycle. Unlike the circuit based quantum computing model (similar to the machines being built by Google or IBM) which uses multi-qubit gates for entangling qubits, photon entanglement happens via projective measurement in X-X or Z-Z basis. Photons can be entangled in space or in time, as shown in Figure 1. Projective measurements for entangling photons are also called *fusions* and these operations are inherently probabilistic and error-prone. The typical failure rate of a fusion ranges between 25-50% [4, 7, 9]. Moreover, to facilitate operations in time domain, entangled photons are held in *delay lines* where they are vulnerable to losses. Such large error-rates necessitate *quantum error correction (QEC)*.

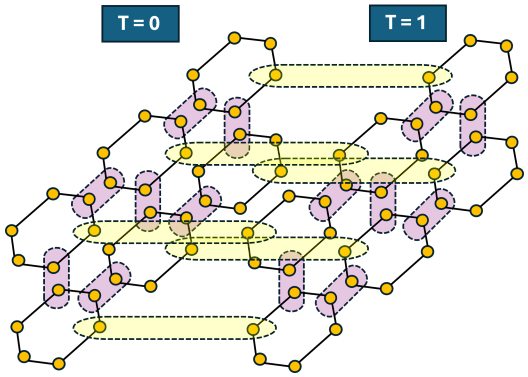


Figure 1: In each clock cycle, six photons are generated by each of six RSGs arranged in a 2x3 grid, producing a total of 36 photons. Photons can be entangled in both space and time.

2. Takeaways of This Paper

We introduce the unique challenges associated with QEC on photonic systems from an architecture perspective. We discuss how publicly available frameworks for surface codes [10] can be modified to study QEC for photonic quantum systems. We open source this framework [10] and finally, discuss open problems and how this tool can be used for future research.

3. How Does QEC work for Photonic Systems?

We consider fusion based quantum computing (FBQC) [3] paradigm due to its inherent fault tolerant characteristics. The central principle of FBQC is to construct *fusion networks*, which defines the configuration of fusion measurements to be made between qubits of different *resource states*. Resource states refer to the collection of qubits generated by an RSG in each clock cycle, similar to the hexagons in Figure 1. FBQC uses these arrays of RSGs and classical processors to construct the fusion network, as shown in Figure 2. This is in stark contrast to surface codes [8, 12] that are primarily studied on superconducting systems, where entanglements on a fixed grid lattice of qubits produce the parity information for QEC.

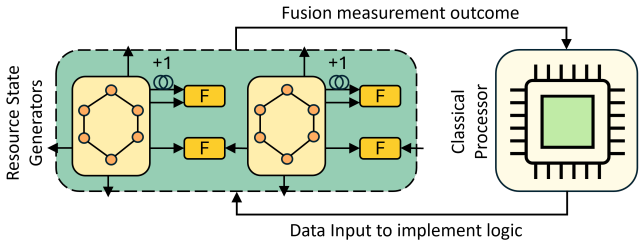


Figure 2: There's a feedback loop between the RSGs and the classical processor, which adjusts the device settings for the RSG array based on fusion outcomes.

We discuss QEC on FBQCs using a 6-ring fusion network [3]. The fusion network comprises of unit cells with two resource states per unit cell, as shown in Figure 3(a). These unit cells are then combined with each other (can be in space or time) to make a larger fusion network as shown in Figure 3(b). The topology of the unit cell when combined with other unit cells leads to a fusion happening on the shared face. For example, if we consider unit cells ❶ and ❸, there is a possible fusion (let's call this fusion-Ⓐ) between the photon on the upper resource state in ❶ and the photon in the lower resource state in ❸. A fusion is also observed between a photon in the upper resource state in ❶ which lies along the edge of the unit cell and the photon in the lower resource state in ❷, which is similarly positioned along the edge of the unit cell (let's call this fusion–Ⓑ). Note that the topology of the unit cell only allows for one fusion between non-adjacent cubes due to the design of the lattice. The fusion networks provides the *parity information* or a *syndrome graph*, as shown in Figure 3(c), that can be used to identify errors via a *decoder*. Typically, *minimum weight perfect matching (MWPM)* [5, 10, 13] is considered a promising algo-

rithm for decoding these codes, similar to surface codes, due to their polynomial time complexity. The direct application of MWPM poses challenges due to the huge volume of parity information that must be decoded in contrast to surface codes on superconducting qubits. Also note the difference in error-rates- superconducting qubits have an average error-rate of 1% on current systems [2], whereas fusions fail with an order of magnitude higher error-rate.

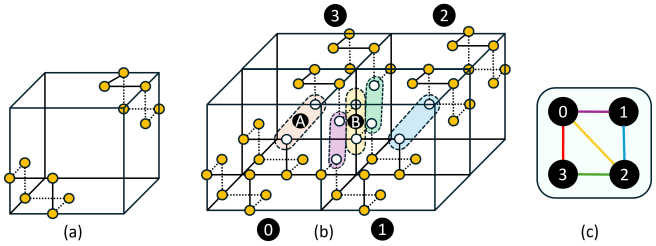


Figure 3: (a) A unit cell with two resource states (6-qubit hexagonal ring). (b) Unit cells combine to form larger fusion networks (c) The syndrome graph of the fusion network (similar to surface codes); NOTE: a parity check can have four or more edges depending on the fusions involved in the check.

To each unit cell there is a parity node in the syndrome graph. If we observe that syndrome bits 0 and 1 light up, then this would indicate that an error has occured with fusion I. This is analogous to surface codes where we have checks defined throughout the code and when the neighboring checks light up, this indicates that there occured an error on the data qubit between the syndrome qubits.

Each check in the syndrome graph (Figure 3 (c)) is representative of one whole unit cell, these checks are formed by combining the surviving stabilizers in the unit cell after the fusion process has been completed [3]. If there are no fusion failures associated with the fusions involving the unit cell, then the product of surviving stabilizers yields a +1, in the presence of a fusion failure, the surviving stabilizers yield a -1, this mechanism is similar to the X and Z checks defined for surface codes which capture parity information for phase and bit-flip errors respectively. MWPM (Pymatching) cannot be directly applied in syndrome graphs where an edge represents more than one fusion (or data qubit) such as in 4-star fusion networks [3], a solution for this is to introduce *virtual nodes and edges* in the graph to represent the additional connections.

4. Key Results and Contributions

An effective QEC algorithm for photonic systems needs to operate within the time-frame required to produce the next layer of resource states (approximately 10-100 ms [11,15,18]). While our initial impression was that this time frame is significantly long for us to rely on software solutions, we quickly identified that this is not the case because the complexity of MWPM grows with the number of 1s or failure bits in the syndrome graph (also represented by the Hamming weight).

Our evaluation involves testing the performance of MWPM [10] for different lattice sizes with an assumed fusion failure rate of 40%. Figure 4 shows that MWPM performs well for small photonic systems, with average decoding latency ∼1ms. However, as we start scaling the system, decoding time increases by manifold as shown in Figures 5. This is because as we scale the system size, the parity information that needs to be decoded for QEC scales exponentially. Existing hardware decoders using MWPM [1,6,20] are incapable of addressing this issue because the Hamming weights that can be handled by these decoders are orders of magnitude lower than what is required to be dealt with in photonic quantum computers.

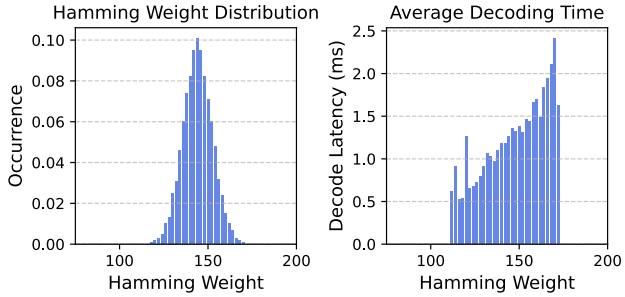


Figure 4: Hamming weight and average decoding time taken by MWPM for a 12x12x2 photonic system (Illustrating 144 resource states and parity collection done over two clock cycles).

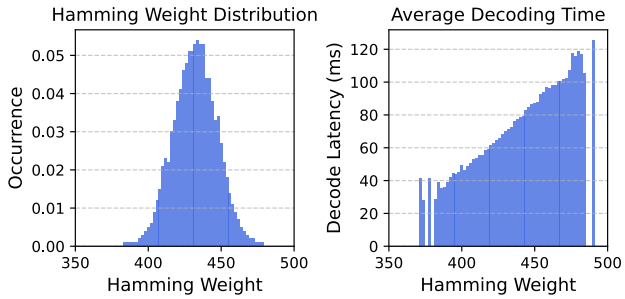


Figure 5: Hamming Weight and Corresponding average time taken by MWPM to decode the syndrome for a 12x12x6 Lattice.

Overall, this paper makes the following contributions:

1. We discuss the inherent challenges associated with photonic systems that necessitate quantum error correction.

2. We provide the first open source framework, inspired by Google's pymatching library, to study the performance of software Minimum Weight Perfect Matching decoder for photonic systems: https://github.com/avinkumarUT/MWPM_Photonics.

3. We show that software decoding is too slow for reasonably sized lattices even though photonic quantum systems can tolerate upto several orders of magnitude higher decoding latency than the competing superconducting qubit technology.

In the future, we hope this infrastructure attracts more effort from the computer architecture and quantum computing community to explore robust scalable solutions for real time decoding in photonic systems.

References

[1] Narges Alavisamani, Suhas Vittal, Ramin Ayanzadeh, Poulami Das, and Moinuddin Qureshi. Promatch: Extending the reach of real-time quantum error correction with adaptive predecoding. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 818–833, 2024.

[2] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[3] Sara Bartolucci, Patrick Birchall, Hector Bombin, Hugo Cable, Chris Dawson, Mercedes Gimeno-Segovia, Eric Johnston, Konrad Kieling, Naomi Nickerson, Mihir Pant, et al. Fusion-based quantum computation. *Nature Communications*, 14(1):912, 2023.

[4] Daniel E Browne and Terry Rudolph. Resource-efficient linear optical quantum computation. *Physical Review Letters*, 95(1):010501, 2005.

[5] William Cook and Andre Rohe. Computing minimum-weight perfect matchings. *INFORMS journal on computing*, 11(2):138–148, 1999.

[6] Poulami Das, Aditya Locharla, and Cody Jones. Lilliput: a lightweight low-latency lookup-table decoder for near-term quantum error correction. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 541–553, 2022.

[7] Fabian Ewert and Peter van Loock. 3/4-efficient bell measurement with passive linear optics and unentangled ancillae. *Physical review letters*, 113(14):140403, 2014.

[8] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A—Atomic, Molecular, and Optical Physics*, 86(3):032324, 2012.

[9] Warren P Grice. Arbitrarily complete bell-state measurement using only linear optical elements. *Physical Review A—Atomic, Molecular, and Optical Physics*, 84(4):042331, 2011.

[10] Oscar Higgott and Craig Gidney. Sparse blossom: correcting a million errors per core second with minimum-weight matching. *arXiv preprint arXiv:2303.15933*, 2023.

[11] D Istrati, Y Pilnyak, JC Loredo, C Antón, N Somaschi, P Hilaire, H Ollivier, M Esmann, L Cohen, L Vidro, et al. Sequential generation of linear cluster states from a single photon emitter. *Nature communications*, 11(1):5501, 2020.

[12] A Yu Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191, 1997.

[13] Vladimir Kolmogorov. Blossom v: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1:43–67, 2009.

[14] Mikkel V Larsen, Xueshi Guo, Casper R Breum, Jonas S Neergaard-Nielsen, and Ulrik L Andersen. Deterministic multi-mode gates on a scalable photonic quantum computing platform. *Nature Physics*, 17(9):1018–1023, 2021.

[15] Jin-Peng Li, Jian Qin, Ang Chen, Zhao-Chen Duan, Ying Yu, YongHeng Huo, Sven Hofling, Chao-Yang Lu, Kai Chen, and Jian-Wei Pan. Multiphoton graph states from a solid-state single-photon source. *ACS Photonics*, 7(7):1603–1610, 2020.

[16] Seth Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–1078, 1996.

[17] Lars S Madsen, Fabian Laudenbach, Mohsen Falamarzi Askarani, Fabien Rortais, Trevor Vincent, Jacob FF Bulmer, Filippo M Miatto, Leonhard Neuhaus, Lukas G Helt, Matthew J Collins, et al. Quantum computational advantage with a programmable photonic processor. *Nature*, 606(7912):75–81, 2022.

[18] Evan Meyer-Scott, Nidhin Prasannan, Ish Dhand, Christof Eigner, Viktor Quiring, Sonja Barkhofen, Benjamin Brecht, Martin B Plenio, and Christine Silberhorn. Scalable generation of multiphoton entangled states by active feed-forward and multiplexing. *Physical Review Letters*, 129(15):150501, 2022.

[19] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[20] Suhas Vittal, Poulami Das, and Moinuddin Qureshi. Astrea: Accurate quantum error-decoding via practical minimum-weight perfect-matching. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–16, 2023.

[21] Jianwei Wang, Fabio Sciarrino, Anthony Laing, and Mark G Thompson. Integrated photonic quantum technologies. *Nature Photonics*, 14(5):273–284, 2020.

[22] Hezi Zhang, Jixuan Ruan, Hassan Shapourian, Ramana Rao Kompella, and Yufei Ding. Oneperc: A randomness-aware compiler for photonic quantum computing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 738–754, 2024.

[23] Hezi Zhang, Anbang Wu, Yuke Wang, Gushu Li, Hassan Shapourian, Alireza Shabani, and Yufei Ding. Oneq: A compilation framework for photonic one-way quantum computation. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023.

[24] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, et al. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020.

Pauli Check Sandwiching for Quantum Characterization and Error Mitigation during Runtime

Joshua Gao[1], Ji Liu[2], Alvin Gonzales[2], Zain H. Saleem[2], Nikos Hardavellas[3], Kaitlin N. Smith[3*]
[1]*Department of Computer Science, Virginia Tech, Blacksburg, VA, USA*
[2]*Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, USA*
[3]*Department of Computer Science, Northwestern University, Evanston, IL, USA*
[*]*kns@northwestern.edu*

***Abstract*—This work presents a novel quantum system characterization and error mitigation framework that applies Pauli check sandwiching (PCS). We motivate our work with prior art in software optimizations for quantum programs like noise-adaptive mapping and multi-programming, and we introduce the concept of PCS while emphasizing design considerations for its practical use. We show that by carefully embedding Pauli checks within a target application (i.e. a quantum circuit), we can learn quantum system noise profiles. Further, PCS combined with multi-programming unlocks non-trivial fidelity improvements.**

I. Introduction and Motivation

While the quantum compute stack has improved, existing quantum system noise still varies cross-chip (Fig. 1) and hovers above error correction thresholds. Much work has been dedicated to improving quantum program success. For example, noise-adaptive mapping is a state-of-art quantum circuit optimization technique used to place and route quantum programs on quantum hardware [5]. Noise-aware schemes, however, depend on accurate calibration data from the quantum computer (QC) provider or from benchmarking. Even in the case that a snapshot of machine properties were available, QC noise is observed to fluctuate over time in ways that are challenging to capture with analytical models.

In our work, we seek alternative methods to determine the noise profile of a QC when its properties are unknown. It is desirable to develop characterization techniques with minimal overheads in terms of quantum processor unit (QPU) runtime. We are inspired by prior art in multi-programming [1] and ensemble distributions [6] to explore how parallelization during runtime can be employed for 1) learning quantum noise / hardware characterization, 2) fidelity improvements in program outcomes, and 3) reductions in runtime latency. This work presents a novel framework that combines the power of multi-programming with Pauli check sandwiching (PCS) for QC characterization and error mitigation. Multi-programming is a technique for parallelized quantum circuit execution while PCS is an error mitigation strategy that utilizes post-selection to reduce errors in quantum circuit outcomes. Our techniques intelligently embed PCS into quantum applications. These programs are then simultaneously executed across a QPU for noise profile characterization. Resulting characterization data adaptively mitigates noisy outcomes within individual thread outcomes. Knowledge of the QPU's best regions creates a weighted ensemble result with non-trivial gain, found to be up to a 25% fidelity improvement in our reported results.

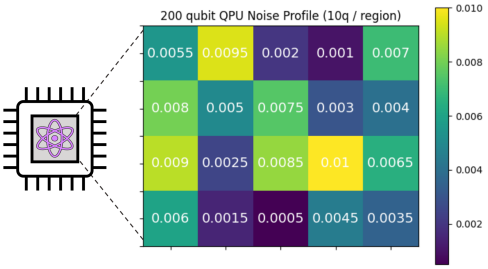


Fig. 1. Example QPU with 20 regions (10 qubits/region) of unique 1q error rates, $p \in [0.0005, 0.01]$, and 2q errors, $2p$. PCS-based characterization with multi-programmed circuits learns the QPU's unknown noise properties.

II. Pauli Check Sandwiching

We rely on a technique called 'Pauli Check Sandwiching' (PCS) to detect and mitigate errors [2]–[4]. In our work, we also find that PCS is a powerful tool for guiding QC characterization. As seen in Fig. 2, PCS surrounds a payload circuit, U, with controlled Pauli operator checks. Errors on U can be detected on an ancilla through phase kickback. It is important that the relationship $R_1 U L_1 = U$ holds so the introduced checks do not disturb the original payload circuit semantics.

Recognizing that all errors can be decomposed into Pauli operators is a key concept in understanding PCS. By cleverly constructing Pauli sandwich checks dependent on U, anti-commutativity relations between the errors and checks can reveal errors in the ancilla qubits. Measuring 1 in any ancillas indicates that phase kickback occurred, meaning anti-commutation between the Paulis, and thus, a present error. As part of the PCS protocol, we discard this error-corrupted shot, leading to an increase in final distribution fidelity of U.

Practical application of PCS requires consideration of trade-offs. First, determining the Pauli unitary used in the check increases in complexity with the circuit. Second, Pauli check logic must be decomposed into the supported gate set of the QC. Third, the overhead of adding PCS into the circuit must not outweigh the corrective benefit in terms of gate error. We consider these constraints in our solution.

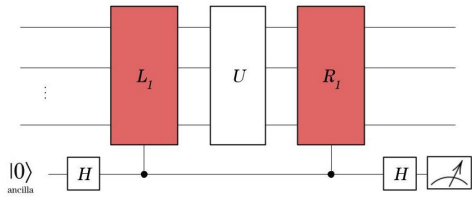


Fig. 2. General PCS circuit layout. The red unitaries represent the Pauli checks that sandwich the main payload circuit. Measurement of the ancillas provide detection of errors that occurred in the payload circuit.

III. QC Characterization and Error Mitigation with PCS

With the PCS protocol, we can 1) learn the noise profile of the QC and 2) increase circuit fidelity by eliminating the states with errors. In our work, we assume that the number of qubits on chip is larger than the number of logical qubits required for an application. Further, we assume no prior knowledge of the noise profile on-chip, and we attempt to maximize hardware utilization. We inject PCS into our target application, but only single-qubit checks on edge qubits are employed to minimize PCS gate count overhead. To maximally utilize hardware via multi-programming, a QC with n physical qubits holds

$$threads = \lfloor n/(q_{algorithm} + q_{ancilla}) \rfloor \tag{1}$$

instances of a circuit running in parallel. In this equation, q indicates qubits, both algorithm and ancilla, that are in the logical quantum circuit.

Assuming the same shot budget in each each local thread, values of the checks (i.e. PCS ancilla measurements of 1) are used during post processing to filter error-corrupted outcomes from each local thread's measurement distribution. This process creates a vector of error-mitigated counts for each thread, c_i. Circuits that encounter more noise will discard a greater count of shots. Discarded shot count for each local thread is represented with the scalar r_i. The percentage of discarded measurement outcomes for each thread,

$$d_i = \frac{r_i}{shots_i}, \tag{2}$$

gives insight into the underlying noise profile of the QC when different regions are compared. Each local d_i is then used as a weight that scales c_i to create s_i:

$$s_i = c_i * \frac{min(d)}{d_i}. \tag{3}$$

In Eqn. 3, $min(d)$ represents the minimum percentage of discarded shots from all the parallel threads. A cumulative distribution, S, is created from the sum of the s_i vectors from the PCS-protected local threads,

$$S = \sum s_i. \tag{4}$$

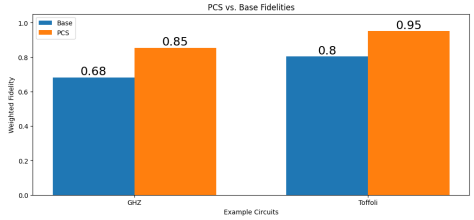


Fig. 3. Differences between fidelity measurements of a circuit with no PCS (base) vs. PCS protection. 10,000 shots were used for both circuits, and fidelity improvement was determined to be 25% (left) and 18.75% (right).

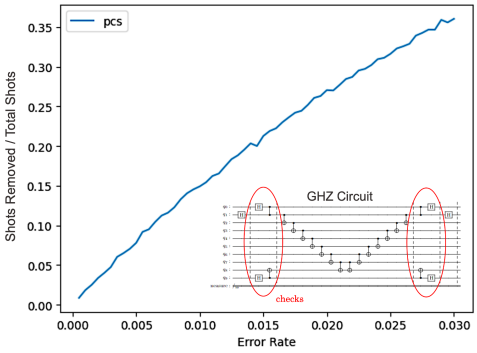


Fig. 4. Shots removed / total shots as error rates increase. 10,000 shots on a GHZ circuit were executed for every error rate (60 total).

IV. Experimental Results

We simulated the fidelity rates of PCS in both a GHZ / sensing circuit and Toffoli circuit to concretely measure the improvement. We expand the Fig. 1 model QC and study a system with 60 10-qubit regions where single-qubit depolarizing noise $p \in [0.0005, 0.03]$ and two-qubit gates are $2p$. We find that PCS not only mitigates error in individual circuit distributions – check data can also be used to intelligently combine the results of multiple threads cross-QPU, demonstrating measurable noise reductions in final outcomes. Fig. 3 shows our results, detailing a multiple percentage point increase in fidelity when multi-programmed applications are protected with PCS and a cumulative distribution is created from local results weighted by check data. Even with only the edge qubits protected, a measurable benefit results. Fig. 4 documents the relationship between discarded shots and error rate, illustrating PCS's potential for QC error characterization. We believe this suggests that PCS has additional promise to guide efficient qubit mapping.

V. Acknowledgments

JL, AG, and ZHS acknowledge support by the Q-NEXT Center. NH was partially supported by NSF CCF-2119069.

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable

worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. http://energy.gov/downloads/doe-public-access-plan.

REFERENCES

[1] Poulami Das, Swamit S Tannu, Prashant J Nair, and Moinuddin Qureshi. A case for multi-programming quantum computers. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 291–303, 2019.

[2] Alvin Gonzales, Ruslan Shaydulin, Zain H Saleem, and Martin Suchara. Quantum error mitigation by pauli check sandwiching. *Scientific Reports*, 13(1):2122, 2023.

[3] Quinn Langfitt, Ji Liu, Benchen Huang, Alvin Gonzales, Kaitlin N Smith, Nikos Hardavellas, and Zain H Saleem. Pauli check extrapolation for quantum error mitigation. *arXiv preprint arXiv:2406.14759*, 2024.

[4] P. Li, J. Liu, A. Gonzales, Z. Saleem, H. Zhou, and P. Hovland. Qutracer: Mitigating quantum gate and measurement errors by tracing subsets of qubits. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 103–117, 2024.

[5] Prakash Murali, Jonathan M Baker, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, pages 1015–1029, 2019.

[6] Swamit S Tannu and Moinuddin Qureshi. Ensemble of diverse mappings: Improving reliability of quantum computers by orchestrating dissimilar mistakes. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 253–265, 2019.

Quantum Tape and Stack Data Structures

Ulrik de Muelenaere and Peter M. Kogge
University of Notre Dame
Notre Dame, IN, USA
udemuele@nd.edu, kogge@nd.edu

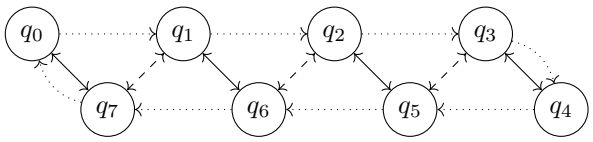


Fig. 1. A quantum tape of $n = 8$ qubits. The qubits that occupy the first $\lceil n/2 \rceil$ positions of the tape are shown in the top row, with the remaining qubits in the bottom row. To shift every qubit one position to the right, as indicated by the dotted arrows, requires two layers of SWAP gates: those shown as solid arrows, followed by the dashed arrows.

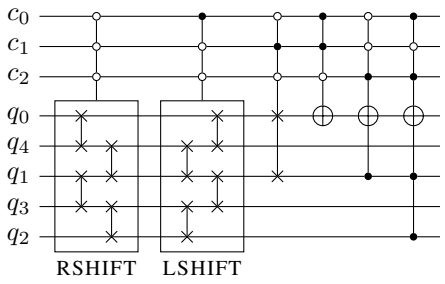


Fig. 2. A quantum circuit to apply a single operation encoded in the opcode register c_i, to the 5-qubit tape q_i. The tape qubits are shown in physical rather than logical order, but indexed in logical order.

I. Introduction

Quantum data structures, that is, data structures in quantum superposition, have been studied in the context of higher-level quantum programming languages [1], [2]. However, in the near term, we see value in data structures with a straightforward implementation as quantum circuits, which can be used directly in the circuit model without overhead.

In this work, we propose one such data structure: a quantum tape, akin to a classical Turing machine tape. By adopting a specific convention, this can also be used as a stack data structure.

We explore the use of quantum tapes to evaluate a sequence of instructions—a program—encoded in a quantum state. A potential application of this is using Grover's algorithm to search for programs with a desired property.

II. Quantum Tapes and Stacks

We first introduce the *quantum tape* data structure. Like the tape in a Turing machine, a "head" points at a single qubit and can be moved left or right, or equivalently, the entire tape can be shifted right or left. Unlike a Turing machine tape, a quantum tape is finite and forms a loop.

Fig. 1 pictures a quantum tape, with the logical order of the qubits indicated by dotted arrows. The figure shows how the RSHIFT operation, which shifts every qubit one position to the right, can be implemented as two layers of SWAP gates. The inverse operation, LSHIFT, simply reverses the two layers of SWAPs. The tape only requires linear connectivity between qubits in the hardware, in order to execute the required SWAPs. Adjacent qubits in the tape need not be physically adjacent, as the first $\lceil n/2 \rceil$ qubits of the tape are physically interleaved with the remaining $\lfloor n/2 \rfloor$ qubits. This arrangement works for tapes of even or odd size.

A quantum tape can be used as a stack by using the following conventions. We consider qubit q_0 to be the top of the stack. The RSHIFT operation is used to push $|0\rangle$ onto the stack, as long as the maximum capacity n is not exceeded, and the tape was initialized to $|0\rangle^{\otimes n}$. Pushing any other state $U|0\rangle$ can be achieved by RSHIFT followed by the unitary operation U.

The pop operation on a classical stack is not reversible, as it discards the top element. For a quantum stack, since the push operation pushes a $|0\rangle$ onto the stack, its inverse would be to pop $|0\rangle$ from the stack. Under this convention, the programmer must ensure that q_0 is uncomputed to the known state $|0\rangle$ before the LSHIFT operation is applied.

If the decision of whether to push or pop is classical, then although individual values are quantum, the number of such values held within the stack is entirely classical. The same effect could be achieved by renumbering the qubits in the classical control system. If the decision is made by quantum control, however, the data structure itself can be in a superposition, so a quantum register is needed if one desires to keep track of the number of elements in the stack.

It is straightforward to extend this to a stack or tape of k-qubit registers instead of a tape of single qubits. Depending on the connectivity of the hardware and the requirements of the program, one could either use k separate tapes, or consider every group of k qubits in the tape to be a single item, and execute LSHIFT or RSHIFT operations in multiples of k.

III. Program Encoding and Search

We construct a quantum circuit that can execute instructions encoded in a quantum state. Previous designs for such quantum stored-program protocols focus on applying an arbitrary unitary operator $\exp(-i\theta B)$, where either the parameter θ [3], [4] or the Hermitian operator B [5] is encoded in a quantum

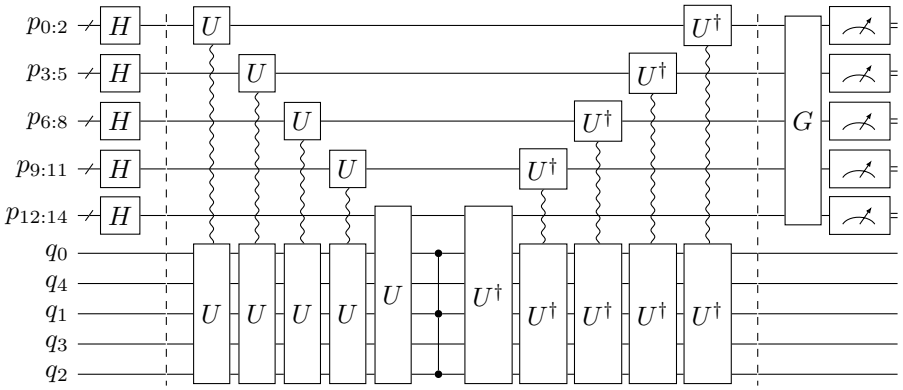


Fig. 3. Grover's algorithm circuit (with one iteration) to find a sequence of 5 instructions that set the top three elements of the stack to $|1\rangle$. The program register p_i consists of 5 3-qubit opcodes. U is the single-operation circuit shown in Fig. 2, and G is the Grover diffusion operator. The part between the barriers is the oracle function which executes the instructions encoded in the program register, flips the phase if the final state is correct, then executes the program in reverse to uncompute the state.

state. Both these approaches require multiple redundant copies of the encoded instruction[1], either because executing the stored instruction is stochastic and may have to be repeated if it fails [3], [4], or it is approximate with error inversely proportional to the number of copies [5]. In either case, requiring multiple independent encoded instructions reduces the ability to truly use these instructions as quantum data, due to the no-cloning theorem. In contrast, we select a finite set of operators without parameters, each represented by an opcode. We could still approximate any unitary if this set is universal, e.g. with a set of Clifford group generators and the T gate.

The opcode can be encoded in the computation basis in a quantum register. In our experiment, $|0\rangle$ is RSHIFT (or "push"), $|1\rangle$ is LSHIFT (or "pop"), $|2\rangle$ is SWAP, $|3\rangle$ is X, $|4\rangle$ is CNOT, $|5\rangle$ is CCNOT, and any other state results in a no-op. This set of operations is inspired by a classical stack machine, and is sufficient to compute any classical reversible function. Fig. 2 shows a circuit to evaluate a single operation, controlled by the opcode register. A k-qubit operation is applied to the first k qubits on the tape, similar to a stack machine that would operate on the top of the stack. Note that unlike the classical stack machine, the opcode register could be in a superposition state, in which case the tape will end up in a superposition after applying this circuit.

This shows that we can encode an instruction as quantum data, and evaluate it using a unitary circuit. The technique easily generalizes to encoding and evaluating a sequence of instructions, i.e. a program (without control flow, although conditional execution is possible using the controlled operations included in the instruction set). This can be useful if one needs to evaluate a program as part of an oracle for a quantum algorithm.

For example, one potential application is using Grover's algorithm [6] to speed up a search through the space of all possible instruction sequences. We demonstrate this by constructing a circuit, shown in Fig. 3, to search for programs that set the top three elements of the stack to $|1\rangle$, from all possible 5-instruction programs. We simulated this circuit using Qiskit. As expected, the most common measurement results (probability 2.7×10^{-4} each) correspond to the 12 programs like $(X, \text{RSHIFT}, X, \text{RSHIFT}, X)$ or $(X, \text{SWAP}, \text{CNOT}, \text{RSHIFT}, \text{CCNOT})$, while all other results have probability 3.0×10^{-5}. Using the optimal number of Grover iterations, $\frac{\pi}{4}\sqrt{2^{5\times 3}/12} \approx 41$, increases this difference, giving a 99.97% probability to measure one of the desired programs (ignoring noise).

[1]This can be redundant in space, by encoding multiple copies of the state into different quantum registers, or in time, by re-encoding the same register multiple times.

IV. Future Work

In the program search experiment, the program register takes on a state which is a superposition of programs. This appears to be a restricted case of what is known as *quantum control* [2], [7] or *quantum alternation* [8]: control flow in a quantum program that depends on quantum state. This topic is not yet fully understood, in particular regarding a physical realization, and we hope to explore how these superpositions of instruction sequences relate to the more general concept.

We also see the potential to use quantum tapes to implement quantum oracles for various classical algorithms that rely on a stack, such as pushdown automota to parse context-free grammars, or algorithms based on depth-first search.

References

[1] C. Yuan and M. Carbin, "Tower: Data structures in quantum superposition," *Proceedings of the ACM on Programming Languages*, vol. 6, no. OOPSLA2, pp. 259–288, Oct. 2022, doi:10.1145/3563297.

[2] A. Sabry, B. Valiron, and J. K. Vizzotto, "From symmetric pattern-matching to quantum control," in *Foundations of Software Science and Computation Structures*, C. Baier and U. Dal Lago, Eds. Cham: Springer International Publishing, 2018, vol. 10803, pp. 348–364, doi: 10.1007/978-3-319-89366-2_19, arXiv:1804.00952 [cs].

[3] G. Vidal, L. Masanes, and J. I. Cirac, "Storing quantum dynamics in quantum states: Stochastic programmable gate for U(1) operations," *Physical Review Letters*, vol. 88, no. 4, p. 047905, Jan. 2002, doi: 10.1103/PhysRevLett.88.047905.

[4] J. Kim, Y. Cheong, J.-S. Lee, and S. Lee, "Storing unitary operators in quantum states," *Physical Review A*, vol. 65, no. 1, p. 012302, Dec. 2001, doi:10.1103/PhysRevA.65.012302.

[5] M. Kjaergaard, M. E. Schwartz, A. Greene, G. O. Samach, A. Bengtsson, M. O'Keeffe, C. M. McNally, J. Braumüller, D. K. Kim, P. Krantz, M. Marvian, A. Melville, B. M. Niedzielski, Y. Sung, R. Winik, J. Yoder, D. Rosenberg, K. Obenland, S. Lloyd, T. P. Orlando, I. Marvian, S. Gustavsson, and W. D. Oliver, "Programming a quantum computer with quantum instructions," Dec. 2020, arXiv:2001.08838 [quant-ph].

[6] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing - STOC '96*. Philadelphia, Pennsylvania, United States: ACM Press, 1996, pp. 212–219, doi:10.1145/237814.237866.

[7] B. Valiron, "Semantics of quantum programming languages: Classical control, quantum control," *Journal of Logical and Algebraic Methods in Programming*, vol. 128, p. 100790, Aug. 2022, doi:10.1016/j.jlamp.2022.100790.

[8] C. Bădescu and P. Panangaden, "Quantum alternation: Prospects and problems," *Electronic Proceedings in Theoretical Computer Science*, vol. 195, pp. 33–42, Nov. 2015, doi:10.4204/EPTCS.195.3, arXiv:1511.01567 [quant-ph].

A Formalization of Measurement Commuting Unitaries

Ulrik de Muelenaere[1], Sinan Pehlivanoglu[2], Amr Sabry[2], Peter Kogge[1]

[1]University of Notre Dame
[2]Indiana University Bloomington

I. Introduction

Nielsen and Chuang [1] state the deferred measurement principle as follows: "Measurements can always be moved from an intermediate stage of a quantum circuit to the end of the circuit; if the measurement results are used at any stage of the circuit then the classically controlled operations can be replaced by conditional quantum operations." This makes it explicit that measurements can be moved over the control (qu)bits of controlled operations, but leaves it implicit that the same works for other classical operations, which can be rewritten as reversible operations, which then have a straightforward mapping to unitary quantum operators.

Dynamic compilation and execution of hybrid quantum algorithms heavily motivates the need to further understand measurement commutation, more specifically, understanding under what conditions a measurement may be moved *earlier* in a quantum circuit. The need for hybrid algorithms commonly arise from the lack of scale and fidelity concerns in the NISQ era. Larger algorithms that can not be practically embedded into a single quantum routine are divided into a sequence of classical and quantum subroutines. However, each switch between a classical and a quantum subroutine comes with a cost in the form of job scheduling, availability, classical communication overhead and bandwidth, that are not often considered. According to the results from IBM [2], a quantum subroutine that takes less than 3 seconds, can spend upwards of 4 minutes in a job queue. As a result, an iterative variational quantum algorithm with multiple iterations of a quantum subroutine can end up spending orders of magnitude more time waiting in a queue than executing. Better understanding how a hybrid routine can be "cut" would allow compilers to choose the most efficient sequence to be executed. The hybrid divide that balances classical and quantum trade-offs might very well be different from what the programmer expected.

The deferred measurement principle, being a statement about circuit equivalence, can always be applied in reverse, so that measurements can be moved before controls, or before unitaries which are equivalent to classical reversible operations. However, it remains unclear what quantum operations will commute with a measurement. We aim to formalize the rules for measurement commutation.

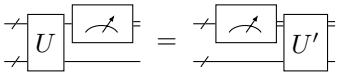


Fig. 1. We say that a pair of operators (U, U') commutes with a measurement when this circuit equivalence holds.

II. Results

When discussing the commutation of a unitary operator with a measurement, the pre-measurement operator U cannot always be equal to the post-measurement operator U', if only because U' usually operates on classical data while U operates on quantum data, as shown in Fig. 1. Therefore it is not a commutation in the strict sense, and we use the term *measurement commutation* to describe it.

Formally, we say that a pair of unitary operators (U, U') commute with projective measurement defined by observable $M = \sum_m mP_m$ (or simply that (U, U') measurement-commute, when the measurement is clear from context) if and only if the final states are equal, i.e.

$$\sum_m P_m U \rho U^\dagger P_m^\dagger = \sum_m U' P_m \rho P_m^\dagger U'^\dagger, \tag{1}$$

for any density operator ρ.

The class of unitary operators that measurement-commute consists of eigenvalue permutations, which we define below.

Let E be the set of eigenvalues of observable M, let d_m denote the degeneracy degree of eigenvalue $m \in E$, and let $\{|mi\rangle \mid m \in E, i = 1, \ldots, d_m\}$ be a basis for the Hilbert space such that the measurement operators are $P_m = \sum_{i=1}^{d_m} |mi\rangle \langle mi|$.

We say that an operator U is an *eigenvalue permutation* with respect to the observable M if it can be written as

$$U = \sum_{m \in E} \sum_{i,j=1}^{d_m} \langle mi|U_m|mj\rangle \, |\phi(m)i\rangle \langle mj| \,, \tag{2}$$

where $\phi : E \to E$ is a permutation of the eigenvalues with the property that $d_{\phi(m)} = d_m$, and for every $m \in E$, U_m is a unitary operator on the subspace spanned by $\{|mi\rangle \mid i = 1, \ldots, d_m\}$.

If each eigenvalue $m \in E$ has the same degree of degeneracy $d_m = d$, then an eigenvalue permutation has the form

$$U = \sum_{m \in E} |\phi(m)\rangle \langle m| \otimes U_m, \tag{3}$$

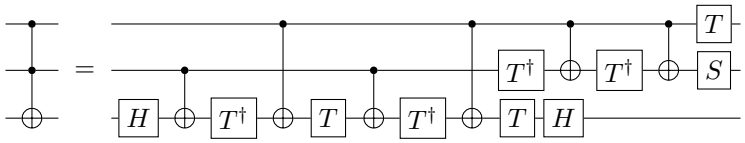


Fig. 2. A decomposition of the Toffoli gate.

which shows that eigenvalue permutations are represented by block matrices, where the overall structure is a permutation matrix on eigenvalues (preserving degeneracy), and the nonzero blocks are the unitaries U_m.

For example, the block matrix

$$U = \begin{pmatrix} U_{00} & 0 & 0 & 0 \\ 0 & 0 & U_{10} & 0 \\ 0 & 0 & 0 & U_{11} \\ 0 & U_{01} & 0 & 0 \end{pmatrix} \tag{4}$$

is an eigenvalue permutation with respect to a computational basis measurement on the first two qubits. However, it is not an eigenvalue permutation with respect to a measurement of only the first qubit, because when considered as a 2×2 block matrix

$$U = \left(\begin{array}{cc|cc} U_{00} & 0 & 0 & 0 \\ 0 & 0 & U_{10} & 0 \\ \hline 0 & 0 & 0 & U_{11} \\ 0 & U_{01} & 0 & 0 \end{array}\right), \tag{5}$$

the overall structure is not a permutation, and the blocks are not unitary.

Our main result follows.

Theorem 1. *A pair of unitary operators (U, U') commute with a projective measurement with observable M, if and only if both are eigenvalue permutations with respect to M, that differ only in a phase $\theta_m \in \mathbb{R}$ per eigenvalue, i.e.*

$$U = \sum_{m \in E} e^{i\theta_m} \sum_{i,j=1}^{d_m} \langle mi|U_m|mj\rangle \, |\phi(m)i\rangle \langle mj| \,, \tag{6a}$$

$$U' = \sum_{m \in E} \sum_{i,j=1}^{d_m} \langle mi|U_m|mj\rangle \, |\phi(m)i\rangle \langle mj| \,. \tag{6b}$$

At least two known results are special cases of Theorem 1:

- A controlled-U operator commutes with a computational basis measurement of any of its control qubits [1, Exercise 4.35].
- A pair of unitaries (U, U') of the form

$$U = \sum_{m \in E} e^{i\theta_m} |\phi(m)\rangle \langle m| \,, \quad U' = \sum_{m \in E} |\phi(m)\rangle \langle m| \tag{7}$$

commute with a measurement of the entire system. In this case, U' is any permutation, i.e. any classical reversible operator, and $U = U'D$, where D is a diagonal operator with entries given by $e^{i\theta_m}$.

III. Future Work

We point out that measurement-commuting unitaries may not be compromised of a single gate. Bian and Selinger formalized the generators for 2-qubit [3] and n-qubit [4] Clifford+T operators. Since the Clifford group is defined as the group of unitaries that normalize Pauli operators, any block or gate that is conjugate under the members of the Clifford group, per our remark about permutation matrices, will measurement-commute as a block. One might also note that the rotation gate $R_x(\frac{\pi}{4}) = HTH$ does not commute with a measurement in the computational basis. However, the Toffoli gate, a permutation matrix in the Z basis, which can be decomposed as in Fig. 2 to contain a similar block of T gates conjugate under the Hadamard, does commute. It isn't immediately obvious to us why this block measurement-commutes, when it contains gates and blocks that do not. We hope to investigate and formalize *block commutation* in future work.

Van den Nest [5] distinguishes between *strong* and *weak* classical simulation of a quantum circuit: the former requires computing the probability distribution of measurement outcomes, while the latter merely requires sampling from said distribution. The final result of [5] is a class of n-qubit circuits that can be efficiently simulated in the weak sense, when applied to the input $|0\rangle^{\otimes n}$ and measured in the computation basis. These circuits consist of a single layer of arbitrary single-qubit rotations, followed by an arbitrary block U of Toffoli and diagonal gates. These unitaries U are exactly those of the form given in (7), and the given simulation method is equivalent to commuting the measurement before U, while removing the diagonal gates to leave the U' given in (7). In other words, Theorem 1 leads to an alternative proof for the result from [5]. This suggests that, using this theorem, we can find a more general class of circuits that can be efficiently simulated, in the weak sense, using a similar technique.

References

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 10th ed. Cambridge: Cambridge University Press, 2010.

[2] P. D. Nation, A. A. Saki, S. Brandhofer, L. Bello, S. Garion, M. Treinish, and A. Javadi-Abhari, "Benchmarking the performance of quantum computing software," 2024, arXiv:2409.08844 [quant-ph].

[3] X. Bian and P. Selinger, "Generators and relations for 2-qubit Clifford+T operators," *Electronic Proceedings in Theoretical Computer Science*, vol. 394, p. 13–28, Nov. 2023, doi:10.4204/eptcs.394.2.

[4] P. Selinger, "Generators and relations for n-qubit Clifford operators," *Logical Methods in Computer Science*, vol. Volume 11, Issue 2, Jun. 2015, doi:10.2168/lmcs-11(2:10)2015.

[5] M. Van den Nest, "Classical simulation of quantum computation, the Gottesman-Knill theorem, and slightly beyond," *Quantum Information & Computation*, vol. 10, no. 3, pp. 258–271, Mar. 2010, arXiv:0811.0898 [quant-ph].

Characterizing Equivalences Between Shallow Quantum Circuit Models

Ben Foxman, Akshat Yaparla

1 Overview

Quantum computers leverage principles of quantum mechanics, such as superposition and entanglement, to perform computational tasks that can be intractable for classical computers. Quantum computational models, while distinctly different from their classical counterparts, draw significant inspiration from them. The quantum circuit model, for instance, is a generalization of classical circuits where local unitary operations replace traditional logic gates. This model is by far the most widely used framework for quantum computing.

Some quantum computational models, however, have no classical analogues, making them particularly intriguing. The "feedforward" model [5] is one such example, where classical measurement outcomes control future gate applications. This model enables separate analysis of classical computational costs during quantum circuit execution.

While both models are universal for quantum computation, the feedforward model has surprising advantages when the quantum computer is restricted to constant-depth computation. The feedforward model enables implementations of Clifford Circuits, PARITY functions, and "Grover-like" oracles[3]—all impossible in constant-depth quantum circuits—by leveraging classical post-processing of measurement outcomes. The constant-depth regime is particularly important for NISQ (Noisy Intermediate Scale Quantum) devices, where low gate fidelities impose heavy restrictions of the depth of the circuit.

Although depth-limited, certain NISQ platforms can leverage the benefits of the feedforward model. For instance, some superconducting devices already support real-time feedforward, which can be used to apply non-local gates [7, 8]. However, such devices often have limited connectivity, i.e. qubits can only interact with their neighbors [1]. Conversely, other near-term platforms, such as those using neutral atoms, allow multi-qubit gates such as `FANOUT`s (via global Rydberg pulses) [9] but are less conducive to feedforward implementation.

Motivated by these experimental setups, we investigate the relationship between two low-depth models of quantum computation. The first model, which we call $\mathsf{QNC}^0_{\mathsf{IM}}$, denotes the class of constant depth quantum circuits with feedforward, where unitaries preceding a set of measurement outcomes $\mathcal{M}$ may be parameterized by an arbitrary TC^0 function of $\mathcal{M}$. We will also use $\mathsf{QNC}^0_{\mathsf{IM}}(G)$ when the interactions on the quantum computer are constrained by some graph topology G. We note that this model is very similar to the "LAQCC" model defined in [3].

The second model, QNC^0_f, has been well-studied in the literature. It consists of constant-depth quantum circuits over single qubit gates and unbounded `FANOUT` gates. It has been shown that the inclusion of `FANOUT` gates adds a surprising amount of power to the constant-depth circuit model, capable of implementing functions such as Mod q and Threshold [4]. We will also consider restricted variants of QNC^0_f, for example where the `FANOUT`s gates are constrained to have some structure (i.e. they cannot act on arbitrary qubits).

In fact, we can show that these two models are equivalent in a tight sense: any circuit family over one model can be translated into a circuit family other the other (see Figure 1), i.e.

$$\mathsf{QNC}^0_{\mathsf{IM}} = \mathsf{QNC}^0_f \tag{1}$$

We give a high-level overview of the proof (along with more formal definitions) in the following section. We are also able to show a version of this equivalence for some of the more restricted models mentioned earlier. For example, $\mathsf{QNC}^0_{\mathsf{IM}}(\mathbb{Z}^2) = \mathsf{QNC}^0_f$ (two-qubit nearest-neighbor interactions), and QNC^0_f remains unchanged even when `FANOUT` gates are restricted to nearest-neighbor gates on a line. In the process of these proofs, we analyze the fine-grained space and time costs required to map between these models (Section 3).

Finally, we aim to further investigate the number of measurement rounds required in feedforward applications. It is known that Clifford circuits can be implemented using only one round of measurement, even when constrained to two-dimensional nearest-neighbor gates [3]. However, non-Clifford circuits may require more rounds of measurements, with the specific number of rounds highly dependent on the circuit in question [6]. Through Equation 1, reducing the number of measurement rounds corresponds to decreasing the `FANOUT` costs in circuits.

Overall, our complexity-theoretic analysis allows us to rigorously characterize, and unify, practically-motivated models of quantum computation. As quantum hardware continues to progress, our results will help inform algorithm design and hardware development, bridging the gap between theoretical understanding and practical implementations.

2 Proof Outline

We now provide some of the formalism behind our definitions, and outline the proof strategy. We first define $\mathsf{QNC}^0_{\mathsf{IM}}$:

Definition 1 ($\mathsf{QNC}^0_{\mathsf{IM}}$). *Families of quantum circuits on n qubits with the following structure: Initialize at most $p(n)$ many ancilla qubits in the all-zero state (in addition to the n-qubit input, p is some polynomial).*

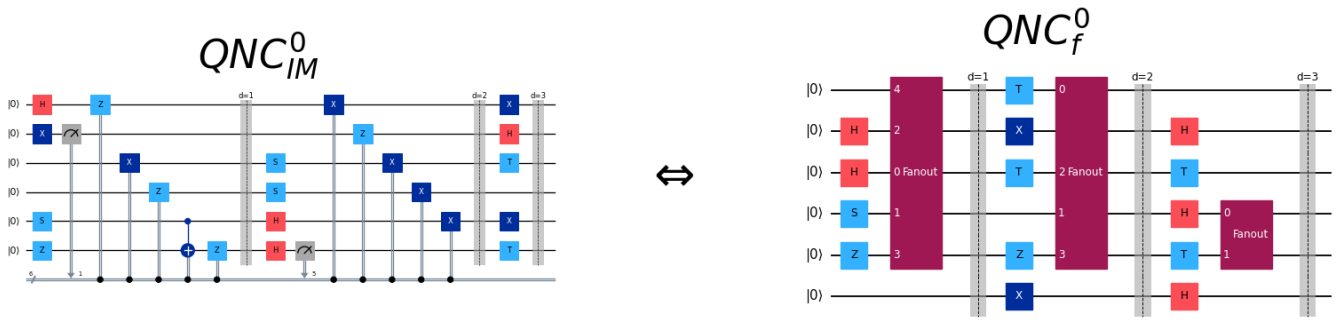


Figure 1: Examples of circuits from the two main classes we study. The bottom line of the $\mathsf{QNC}^0_{\mathsf{IM}}$ circuit represents a classical memory, where subsequent quantum gates are controlled on bits in the memory (two lines). Dotted lines represent barriers between layers in the circuits.

1. *Apply a constant depth, polynomial size quantum circuit over some fixed universal gate set $\mathcal{G}$ consisting of 1 and 2 qubit gates.*
2. *Measure a subset of the qubits in the Z-basis, tracing out the measured qubits, i.e. a destructive measurement.*
3. *Repeat for $O(1)$ iterations.*

Moreover, gates in the ith iteration may be parametrized by the parity of a subset of the measurement outcomes in iterations $1, 2, \ldots, i-1$.

Now, we outline the proof. First, to show $\mathsf{QNC}^0_f \subseteq \mathsf{QNC}^0_{\mathsf{IM}}$, we use a gadget that uses a mid-circuit measurements and classical computation that simulates the function of the `FANOUT` gate. For the gadget to simulate the `FANOUT` gate, that the mixed state after the gate and after the gadget are applied are the same. Since `FANOUT` is a unitary operator, we just have to show that the pure state after `FANOUT` and the gadget are applied are identical, with the measured qubits traced out. For a `FANOUT` acting on n qubits, we require an ancilla register of $n-1$ qubits that will eventually be measured. Denote the ith ancilla qubit as $|a_i\rangle$. In addition, denote the ith input register to `FANOUT` as $|q_i\rangle$. We initialize $|q_1\rangle \leftarrow |\phi\rangle$, while all other registers are initialized to $|0\rangle$. The gadget can then be constructed as follows.

1. Apply a Hadamard transform on registers $|q_2\rangle$ through $|q_n\rangle$
2. For $i \in [n-1]$ Apply a CNOT gate on the ancilla qubit $|a_i\rangle = |0\rangle$ with $|q_i\rangle$ as a control.
3. For $i \in [n-1]$ Apply a CNOT gate on the ancilla qubit $|a_i\rangle = |0\rangle$ with $|q_{i+1}\rangle$ as a control.
4. Measure the ancilla register $\{|a_1\rangle, \ldots, |a_{n-1}\rangle\}$ into the classical register $\{c_1, \ldots, c_{n-1}\}$.
5. Using an $\mathsf{AC}^0[\oplus]$ circuit, compute $n-2$ parities
$$p_i = \bigoplus_{k=1}^{i} c_k \text{ for } 2 \le i \le n-1$$
6. Apply X^{c_1} on qubit $|q_2\rangle$ Using the classical parities p_i computed in the previous step, apply X^{p_i} on qubit $|q_{i+1}\rangle$ for all $2 \le i \le n-1$.

Let the state before the gadget is applied by $|\psi\rangle = (a|0\rangle + b|1\rangle)|0^{n-1}\rangle$, where $a|0\rangle + b|1\rangle$ is the qubit in the first register we wish to `FANOUT`. When the measured qubits of the gadget are traced out, the state obtained is $|\psi'\rangle = a|0^n\rangle + b|1^n\rangle$.

For the reverse direction, we treat the classical memory used to store measurement information as an additional quantum register. Then, for all allowed classical gates, we can construct an equivalent quantum gate. For example, we can use a `TOFFOLI` gate to simulate classical `AND` and `OR` gates up to depth 3, and all unbounded `PARITY` gates can be simulated with a `FANOUT` gate in depth 3. Then, for all unitaries that are controlled by classical data, we can create equivalent unitaries that are controlled by the quantum memory meant to simulate classical ones. This new QNC^0_f circuit simulates the mid-circuit measurement circuit before and after all classical and quantum operations are applied.

3 Further Remarks

Finally, we add a few remarks regarding the fine-grained equivalences mentioned earlier, and consider two restricted versions our our main models—locality constraints on the feedforward circuit (for example, $\mathsf{QNC}^0_{\mathsf{IM}}(\mathbb{Z}^2)$) and a restricted action of the `FANOUT` gates in QNC^0_f. Surprisingly, we can demonstrate that these models recover the full power of feedforward (or `FANOUT`) computation. When imposing locality constraints, we can demonstrate that arbitrary layers of `FANOUT` gates can be performed in parallel, using a version of the Bell-State routing algorithm described in [2]. When restricting the action of `FANOUT` gates (for example, only allowing FANOUTs to be performed on horizontal or vertical stretches of qubits), we must be strategic about the locations of the ancillas used as inputs in the unitary formulation of the classical feedforward circuit. The equivalence of these models under such restrictions further highlights the robustness of the feedforward computational model.

References

[1] M. AbuGhanem. *IBM Quantum Computers: Evolution, Performance, and Future Directions*. 2024. arXiv: 2410.00916 [quant-ph]. URL: https://arxiv.org/abs/2410.00916.

[2] Michael Beverland, Vadym Kliuchnikov, and Eddie Schoute. "Surface Code Compilation via Edge-Disjoint Paths". In: *PRX Quantum* 3.2 (May 2022). ISSN: 2691-3399. DOI: 10.1103/prxquantum.3.020342. URL: http://dx.doi.org/10.1103/PRXQuantum.3.020342.

[3] Harry Buhrman et al. *State preparation by shallow circuits using feed forward*. 2024. arXiv: 2307.14840 [quant-ph]. URL: https://arxiv.org/abs/2307.14840.

[4] Peter Hoyer and Robert Spalek. In: *Theory of Computing* 1.1 (2005), pp. 81–103. ISSN: 1557-2862. DOI: 10.4086/toc.2005.v001a005. URL: http://dx.doi.org/10.4086/toc.2005.v001a005.

[5] Richard Jozsa. *An introduction to measurement based quantum computation*. 2005. arXiv: quant-ph/0508124 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/0508124.

[6] Robert Raussendorf and Hans Briegel. *Computational model underlying the one-way quantum computer*. 2002. arXiv: quant-ph/0108067 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/0108067.

[7] Yongxin Song et al. *Realization of Constant-Depth Fan-Out with Real-Time Feedforward on a Superconducting Quantum Processor*. 2024. arXiv: 2409.06989 [quant-ph]. URL: https://arxiv.org/abs/2409.06989.

[8] L. Steffen et al. "Deterministic quantum teleportation with feed-forward in a solid state system". In: *Nature* 500.7462 (Aug. 2013), pp. 319–322. ISSN: 1476-4687. DOI: 10.1038/nature12422. URL: http://dx.doi.org/10.1038/nature12422.

[9] Karen Wintersperger et al. "Neutral atom quantum computing hardware: performance and end-user perspective". In: *EPJ Quantum Technology* 10.1 (Aug. 2023). ISSN: 2196-0763. DOI: 10.1140/epjqt/s40507-023-00190-1. URL: http://dx.doi.org/10.1140/epjqt/s40507-023-00190-1.

Quantum Control of an Oscillator with a Kerr-cat Qubit

Andy Z. Ding[1],[*] Benjamin L. Brock[1],[†] Alec Eickbusch,[‡] Akshay Koottandavida, Nicholas E. Frattini,[§] Rodrigo G. Cortiñas, Vidul R. Joshi, Stijn J. de Graaf, Benjamin J. Chapman,[¶] Suhas Ganjam,[‡] Luigi Frunzio, Robert J. Schoelkopf, and Michel H. Devoret[**]

Departments of Applied Physics and Physics, Yale University, New Haven, CT, USA

(Dated: October 22, 2024)

A major obstacle to scaling up quantum computers is noise, which causes logical errors and prevents the reliable execution of quantum algorithms. Quantum error correction (QEC) provides a path toward fault-tolerant operations [1–4], but this typically comes at the cost of significant resource overhead, often requiring hundreds of physical qubits per logical qubit [5–9]. Bosonic codes offer a hardware-efficient alternative to multi-qubit QEC codes by redundantly encoding quantum information in the large Hilbert space of a harmonic oscillator [10–15]. These codes have been employed to achieve landmark experimental demonstrations, including beyond break-even QEC of quantum memories [16–18] and fully-autonomous QEC protocols [19–21]. However, these experiments rely on an ancilla qubit to control the oscillator and perform quantum error correction, such that errors on the ancilla can propagate to the logical qubit, limiting its lifetime.

In circuit quantum electrodynamics [22, 23], these realizations of bosonic codes typically use a microwave cavity as the oscillator and a transmon as the ancilla. The two are coupled dispersively, and since this dispersive interaction is transparent to transmon phase-flip errors, these QEC protocols are usually only sensitive to transmon bit-flip errors. Although fault-tolerant error syndrome measurements have been experimentally demonstrated using a transmon ancilla [24], another approach to achieving fault-tolerance is to use a biased-noise qubit as an ancilla for bosonic codes [25]. Ideally, the error channel of such an ancilla should be dominated by phase flips, with a negligible rate of bit flips compared to other rates in the system.

The Kerr-cat qubit (KCQ) has the potential to be exactly such an ancilla due to its promise of an exponential noise bias [26]. Recent experimental realizations of KCQs have reached a strong noise bias of about 1000 [27, 28] (corresponding to a bit-flip lifetime of $\sim$ 1 ms and phase-flip lifetime of ~ 1 μs), and although this is not exponentially large, there are many possible methods for further improvement [29–33]. With a strong noise bias and fast single-qubit gates [28, 34], the only remaining ingredient for using the KCQ as an ancilla for bosonic codes is an entangling operation between the KCQ and an oscillator that enables the measurement of error syndromes.

In this work, we experimentally demonstrate a coherent parametrically-driven conditional displacement (CD) gate between a KCQ and a high-quality-factor microwave cavity, where the cavity is displaced in one of two directions depending on the state of the KCQ. Combined with single-qubit gates on the KCQ, this CD gate enables universal quantum control of the cavity [35]. We use this CD gate to measure the decoherence of the cavity in the presence of the KCQ and discover excess cavity dephasing due to heating of the KCQ into excited states, an effect that was not previously predicted. However, by engineering frequency-selective dissipation to counteract this heating [29], we are able to eliminate this dephasing up to the precision of our measurements. This lack of dephasing indicates that the two systems do not entangle unless we are actively driving their interaction. The Kerr-cat control of a cavity can be applied for fault-tolerant syndrome measurements of bosonic codes [25], in particular the Gottesman-Kitaev-Preskill code [11] whose error syndromes can be mapped to an ancilla via CD gates [36–38].

[1] These authors contributed equally to this work.
[*] zhenghao.ding@yale.edu
[†] benjamin.brock@yale.edu
[‡] Present address: Google Quantum AI, Santa Barbara, CA
[§] Present address: Nord Quantique, Sherbrooke, QC J1J 2E2, Canada
[¶] Present address: Microsoft Azure Quantum
[**] michel.devoret@yale.edu

[1] P. Shor, Proceedings of 37th Conference on Foundations of Computer Science , 56 (1996).
[2] E. Knill, R. Laflamme, and W. H. Zurek, Science **279**, 342 (1998).
[3] D. Aharonov and M. Ben-Or, SIAM Journal on Computing **38**, 1207 (2008).
[4] A. Y. Kitaev, Annals of Physics **303**, 2 (2003).
[5] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Physical Review A - Atomic, Molecular, and Optical Physics **86**, 032324 (2012).
[6] I. D. Kivlichan, C. Gidney, D. W. Berry, N. Wiebe, J. McClean, W. Sun, Z. Jiang, N. Rubin, A. Fowler, A. Aspuru-Guzik, H. Neven, and R. Babbush, Quantum **4**, 296 (2020).
[7] J. Lee, D. W. Berry, C. Gidney, W. J. Huggins, J. R. McClean, N. Wiebe, and R. Babbush, PRX Quantum **2**, 030305 (2021).
[8] D. Gottesman, Quantum Information and Computation **14**, 1338 (2014).
[9] N. P. Breuckmann and J. N. Eberhardt, PRX Quantum **2**, 040101 (2021).
[10] I. L. Chuang, D. W. Leung, and Y. Yamamoto, Phys. Rev. A **56**, 1114 (1997).
[11] D. Gottesman, A. Kitaev, and J. Preskill, Physical Review A **64**, 10.1103/PhysRevA.64.012310 (2001).
[12] M. H. Michael, M. Silveri, R. T. Brierley, V. V. Albert, J. Salmilehto, L. Jiang, and S. M. Girvin, Phys. Rev. X **6**, 031006 (2016).
[13] M. Mirrahimi, Z. Leghtas, V. V. Albert, S. Touzard, R. J. Schoelkopf, L. Jiang, and M. H. Devoret, New Journal of Physics **16**, 045014 (2014).
[14] A. Joshi, K. Noh, and Y. Y. Gao, Quantum Science and Technology **6**, 033001 (2021).
[15] W. Cai, Y. Ma, W. Wang, C. L. Zou, and L. Sun, Fundamental Research **1**, 50 (2021).
[16] N. Ofek, A. Petrenko, R. Heeres, P. Reinhold, Z. Leghtas, B. Vlastakis, Y. Liu, L. Frunzio, S. M. Girvin, L. Jiang, M. Mirrahimi, M. H. Devoret, and R. J. Schoelkopf, Nature **536**, 441 (2016).
[17] V. V. Sivak, A. Eickbusch, B. Royer, S. Singh, I. Tsioutsios, S. Ganjam, A. Miano, B. L. Brock, A. Z. Ding, L. Frunzio, S. M. Girvin, R. J. Schoelkopf, and M. H. Devoret, Nature **616**, 50 (2023).
[18] Z. Ni, S. Li, X. Deng, Y. Cai, L. Zhang, W. Wang, Z. B. Yang, H. Yu, F. Yan, S. Liu, C. L. Zou, L. Sun, S. B. Zheng, Y. Xu, and D. Yu, Nature 2023 616:7955 **616**, 56 (2023).
[19] J. M. Gertler, B. Baker, J. Li, S. Shirol, J. Koch, and C. Wang, Nature 2021 590:7845 **590**, 243 (2021).
[20] B. de Neeve, T. L. Nguyen, T. Behrle, and J. P. Home, Nature Physics 2022 18:3 **18**, 296 (2022).
[21] D. Lachance-Quirion, M.-A. Lemonde, J. O. Simoneau, L. St-Jean, P. Lemieux, S. Turcotte, W. Wright, A. Lacroix, J. Fréchette-Viens, R. Shillito, F. Hopfmueller, M. Tremblay, N. E. Frattini, J. Camirand Lemyre, and P. St-Jean, Phys. Rev. Lett. **132**, 150607 (2024).
[22] A. Blais, R.-S. Huang, A. Wallraff, S. M. Girvin, and R. J. Schoelkopf, Phys. Rev. A **69**, 062320 (2004).
[23] A. Blais, A. L. Grimsmo, S. M. Girvin, and A. Wallraff, Rev. Mod. Phys. **93**, 025005 (2021).
[24] S. Rosenblum, P. Reinhold, M. Mirrahimi, L. Jiang, L. Frunzio, and R. J. Schoelkopf, Science **361**, 266 (2018).
[25] S. Puri, A. Grimm, P. Campagne-Ibarcq, A. Eickbusch, K. Noh, G. Roberts, L. Jiang, M. Mirrahimi, M. H. Devoret, and S. M. Girvin, Physical Review X **9**, 041009 (2019).
[26] S. Puri, S. Boutin, and A. Blais, npj Quantum Information **3**, 1 (2017).
[27] N. E. Frattini, R. G. Cortiñas, J. Venkatraman, X. Xiao, Q. Su, C. U. Lei, B. J. Chapman, V. R. Joshi, S. M. Girvin, R. J. Schoelkopf, S. Puri, and M. H. Devoret, The squeezed kerr oscillator: spectral kissing and phase-flip robustness (2022), arXiv:2209.03934 [quant-ph].
[28] A. Hajr, B. Qing, K. Wang, G. Koolstra, Z. Pedramrazi, Z. Kang, L. Chen, L. B. Nguyen, C. Junger, N. Goss, I. Huang, B. Bhandari, N. E. Frattini, S. Puri, J. Dressel, A. N. Jordan, D. Santiago, and I. Siddiqi, High-coherence kerr-cat qubit in 2d architecture (2024), arXiv:2404.16697 [quant-ph].
[29] H. Putterman, J. Iverson, Q. Xu, L. Jiang, O. Painter, F. G. Brandão, and K. Noh, Physical Review Letters **128**, 110502 (2022).
[30] B. Bhandari, I. Huang, A. Hajr, K. Yanik, B. Qing, K. Wang, D. I. Santiago, J. Dressel, I. Siddiqi, and A. N. Jordan, Symmetrically threaded squids as next generation kerr-cat qubits (2024), arXiv:2405.11375 [quant-ph].
[31] J. Venkatraman, R. G. Cortiñas, N. E. Frattini, X. Xiao, and M. H. Devoret, Proceedings of the National Academy of Sciences of the United States of America **121**, 10.1073/PNAS.2311241121 (2024).
[32] L. Gravina, F. Minganti, and V. Savona, PRX Quantum **4**, 020337 (2023).
[33] D. Ruiz, R. Gautier, J. Guillaud, and M. Mirrahimi, Phys. Rev. A **107**, 042407 (2023).
[34] A. Grimm, N. E. Frattini, S. Puri, S. O. Mundhada, S. Touzard, M. Mirrahimi, S. M. Girvin, S. Shankar, and M. H. Devoret, Nature **584**, 205 (2020).
[35] A. Eickbusch, V. Sivak, A. Z. Ding, S. S. Elder, S. R. Jha, J. Venkatraman, B. Royer, S. M. Girvin, R. J. Schoelkopf, and M. H. Devoret, Nature Physics **18**, 1464 (2022).
[36] B. M. Terhal and D. Weigand, Phys. Rev. A **93**, 012315 (2016).
[37] P. Campagne-Ibarcq, A. Eickbusch, S. Touzard, E. Zalys-Geller, N. E. Frattini, V. V. Sivak, P. Reinhold, S. Puri, S. Shankar, R. J. Schoelkopf, L. Frunzio, M. Mirrahimi, and M. H. Devoret, Nature **584**, 368 (2020).
[38] C. Flühmann and J. P. Home, Physical Review Letters **125**, 043602 (2020).

Architecting a quantum operating system: microkernel, message passing and supercomputing

Alexandru Paler
Aalto University, Finland
alexandru.paler@aalto.fi

Abstract

Quantum computers require an operating system (QCOS). A QCOS is classic software running on classic hardware, responsible for preparing, starting, and managing quantum computations. In the following, we discuss why a QCOS should be architected as follows: 1) using a microkernel; 2) the software components are working in an aggregated, non-stacked manner and communicate by message passing; 3) the components are executed by default on supercomputers, unless there are very good reasons not to.

1 Introduction

Technology roadmaps (e.g. [1, 10]), envision the early 2030s as the moment when large scale quantum computers might be operating. To have a QCOS ready by that deadline, its components have to be as small as possible, well defined, working and integrated[1]. Therefore, this is not to open a Tanenbaum – Torvalds-like debate, but to use parts of the experience of NASA's Apollo programme when building the first QCOS. In a nutshell, the Apollo approach [8] is: 1) meeting a fixed deadline means to work backward to identify the points by which sub-systems have to be ready and integrated; 2) given a choice between two technologically workable ways to do something, take the better, proven and more expensive way. We argue that the better, proven and more expensive way are supercomputers running a QCOS built on top of a microkernel.

Practical quantum computations will require millions of qubits (e.g. [5, 7]), such that distributed quantum computers have been proposed [13]. At the same time, it has been shown that a QCOS could use Grover's algorithm to speed up classical OS functions like scheduling [4]. Although any OS could benefit from quadratic speedups, qubits are such a scarce resource that it would be better to use them for quantum chemistry, for example.

A supercomputer would enable the extremely scalable control of millions of physical qubits, and support the fault-tolerance of the QCOS. Nevertheless, for some tasks, such as the decoding of quantum error-correcting codes (e.g. [2, 12]), the communication latency between a quantum computer and supercomputer might be too large.

[1]This is a shorter and updated version of the manuscript published in [9]

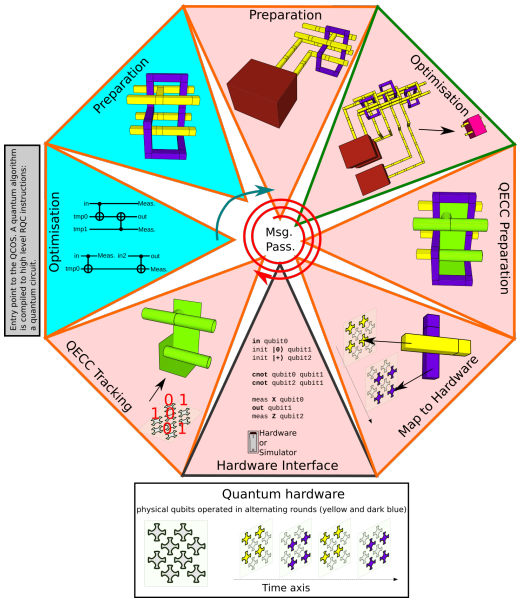


Figure 1. Architecture and interaction diagram of the QCOS. Each triangle is a QCOS component. The arrows indicate a non-strict execution order of the components. To execute a quantum algorithm, the first part of the QCOS is offline (blue arrow and triangles), after which online preparations, optimisations etc. are performed in a loop (red spiral and triangles). The component pictograms represent elements of quantum circuits protected by the braided surface quantum error-correcting code [14].

2 An aggregated architecture

Our QCOS has an aggregated architecture, meaning that there is no top-down work-flow, and all components operate at the same level (Figure 1). At the same time, the quantum computer control software has been proposed having layered, stacked and densely interacting software components (e.g. [6]). This is a trait of *almost-monolithic architectures*. Monolithic kernels do not exhibit high levels of fault-tolerance (e.g. a driver crash can panic the kernel and stop the entire system).

The fault-tolerance of the QCOS plays a significant role in evaluating the total reliability of an arbitrary quantum computation. In general, it is common for systems to fail even when every component is correct and seems secure [3]. It would be ironic for the quantum computing pessimists

to be right, but not because the quantum technology or the QCOS would not be scalable per se, but because the QCOS is unknowingly faulty.

Our proposal is to depart from monolithic architectures and to use distributed microkernels (so-called splitkernels [11]) in order to increase the QCOS fault-tolerance. The simplest QCOS architecture will have a loosely coupled and distributed architecture, which will include components for circuit compilation, optimisation, error-correction, decoding etc. These components are running on top of the microkernel.

A microkernel assumes that the QCOS is light and the components are applications, whereas in a splitkernel the components are parts of the kernel, and have high execution priority. A splitkernel for classical OS has been proposed by [11], and the overall system fault-tolerance has increased, while the performance did not drop significantly – we increase the fault-tolerance by using micro- and split-kernels at the same time.

3 Message-passing components

The QCOS will be started each time a quantum algorithm is executed. The QCOS components and the splitkernel should use message-passing (e.g. MPI), which is a natural communication strategy in supercomputing.

In Figure 1 there is a QCOS boot phase (offline, blue triangles), and a QCOS execution phase (online, red triangles). During boot, the quantum algorithm is compiled and optimised in the form of a quantum circuit. The next step is to prepare the error-corrected quantum circuit offline. Offline compilation, optimisation and error-correction are very complex and may take a lot of time. The booting phase is an abstraction of the entire procedure until the actual execution of the quantum algorithm is started.

The unique character of the QCOS is given by its online components, which are used to take corrective measures in a real-time, low-latency and resilient manner in order to maintain computational fault-tolerance.

The advance being proposed here is the scalable and resilient feedback-loop between the high level quantum algorithm and the quantum computer. During the loop's execution, the circuits are now compiled and optimized based on the stream of results received from the computer. After compilation, the error-corrected representation of the computations is translated to quantum hardware instructions. The instructions are sent through a hardware interface to the quantum computer. The QCOS schedules instructions into discrete rounds. For example, in Figure 1, there are yellow and magenta rounds, and green crosses represent the hardware qubits.

Real-time error decoding is a topic of intense research, because the QCOS has to face immense data rates generated by the millions of qubits[2]. Quantum measurements are probabilistic, and instruction execution generates a probabilistic binary measurement result. The feedback for correcting the computations is formed by the measurement results collected through the hardware interface. Error-correction tracking uses the feedback stream: error syndromes are computed, and the execution of the quantum computation is dynamically adapted in case computational corrections are needed.

Corrections imply online compilation and optimisation of sub-circuits. The mapping component is to communicate to the optimisation component that further work is necessary to fit the computation on the available qubits. Overall, the functionality of the QCOS has to be perfectly timed, because the quantum hardware cannot wait for an optimisation stuck in local minima.

4 Running the QCOS on a supercomputer

The aggregated QCOS should be easily executed on a supercomputer. The QCOS problem is difficult, but not extravagantly complex in the sense of computer science theory, and throwing more hardware at it to solve it should be fine [8]. The advantages of using a supercomputer are that the nodes communicate through very high speed links, message passing is natively supported, and supernodes could be used for each QCOS component. Almost surely these supercomputers would be able to process at the necessary throughput.

The cost of a supercomputer QCOS does not seem exaggerated when considering that the most powerful ones are around $400 million. It is reasonable to assume that the million-qubit quantum computer will cost more than the supercomputer needed to control it. In fact, nobody has built a simple aggregated QCOS starting from all the freely available software for compiling, optimising and error-correcting quantum circuits. The impossibility of controlling a large scale quantum computer could be rebutted by a supercomputer aggregated QCOS capable of handling the most horrific worst-case scenarios of error-correction.

References

[1] [n. d.]. Quantinuum accelerates the path to Universal Fully Fault-Tolerant Quantum Computing; supports Microsoft's AI and quantum-powered compute platform and "the path to a Quantum Supercomputer" — quantinuum.com. https://www.quantinuum.com/blog/quantinuum-accelerates-the-path-to-universal-fault-tolerant-quantum-computing-supports-microsofts-ai-and-quantum-powered-compute-platform-and-the-path-to-a-quantum-supercomputer. [Accessed 11-10-2024].

[2] Francesco Battistel, Christopher Chamberland, Kauser Johar, Ramon WJ Overwater, Fabio Sebastiano, Luka Skoric, Yosuke Ueno, and Muhammad Usman. 2023. Real-time decoding for fault-tolerant quantum computing: Progress, challenges and outlook. *Nano Futures* 7, 3 (2023), 032003.

[3] Steven M Bellovin and Peter G Neumann. 2018. The big picture. *Commun. ACM* 61, 11 (2018), 24–26.

[4] Keith A Britt, Fahd A Mohiyaddin, and Travis S Humble. 2017. Quantum Accelerators for High-Performance Computing Systems. In *Rebooting Computing (ICRC), 2017 IEEE International Conference on*. IEEE, 1–7.

[5] Earl T Campbell, Barbara M Terhal, and Christophe Vuillot. 2017. Roads towards fault-tolerant universal quantum computation. *Nature* 549, 7671 (2017), 172.

[6] Xiang Fu, MA Rol, CC Bultink, J van Someren, Nader Khammassi, Imran Ashraf, RFL Vermeulen, JC de Sterke, WJ Vlothuizen, RN Schouten, et al. 2018. A microarchitecture for a superconducting quantum processor. *IEEE Micro* 38, 3 (2018), 40–47.

[7] Craig Gidney and Martin Ekerå. 2021. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum* 5 (2021), 433.

[8] Thomas Haigh. 2018. Hey Google, what's a moonshot?: how Silicon Valley mocks Apollo. *Commun. ACM* 62, 1 (2018), 24–30.

[9] Alexandru Paler. 2020. Aggregated control of quantum computations: When stacked architectures are too good to be practical soon. *Computer* 53, 8 (2020), 74–78.

[10] Riverlane. [n. d.]. Introducing Riverlane's Quantum Error Correction roadmap - Riverlane — riverlane.com. https://www.riverlane.com/blog/introducing-riverlane-s-quantum-error-correction-roadmap. [Accessed 11-10-2024].

[11] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A Disseminated, Distributed {OS} for Hardware Resource Disaggregation. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 69–87.

[12] Luka Skoric, Dan E Browne, Kenton M Barnes, Neil I Gillespie, and Earl T Campbell. 2023. Parallel window decoding enables scalable fault tolerant quantum computation. *Nature Communications* 14, 1 (2023), 7040.

[13] Rodney Van Meter and Simon J Devitt. 2016. The path to scalable distributed quantum computing. *Computer* 49, 9 (2016), 31–42.

[14] Rodney Van Meter and Clare Horsman. 2013. A blueprint for building a quantum computer. *Commun. ACM* 56, 10 (2013), 84–93.