Taming the Instruction Bandwidth of Quantum Computers via Hardware-Managed Error Correction

Swamit S. Tannu
Georgia Institute of Technology
swamit@gatech.edu

Zachary A. Myers
Stanford University
zamyers@stanford.edu

Prashant J. Nair
Georgia Institute of Technology
pnair6@gatech.edu

Douglas M. Carmean
Microsoft Research
dcarmean@microsoft.com

Moinuddin K. Qureshi
Georgia Institute of Technology
moin@ece.gatech.edu

ABSTRACT

A quantum computer consists of quantum bits (qubits) and a control processor that acts as an interface between the programmer and the qubits. As qubits are very sensitive to noise, they rely on continuous error correction to maintain the correct state. Current proposals rely on software-managed error correction and require large instruction bandwidth, which must scale in proportion to the number of qubits. While such a design may be reasonable for small-scale quantum computers, we show that instruction bandwidth tends to become a critical bottleneck for scaling quantum computers.

In this paper, we show that 99.999% of the instructions in the instruction stream of a typical quantum workload stem from error correction. Using this observation, we propose *QuEST (Quantum Error-Correction Substrate*), an architecture that delegates the task of quantum error correction to the hardware. QuEST uses a dedicated programmable micro-coded engine to continuously replay the instruction stream associated with error correction. The instruction bandwidth requirement of QuEST scales in proportion to the number of active qubits (typically << 0.1%) rather than the total number of qubits. We analyze the effectiveness of QuEST with area and thermal constraints and propose a scalable microarchitecture using typical Quantum Error Correction Code (QECC) execution patterns. Our evaluations show that QuEST reduces instruction bandwidth demand of several key workloads by five orders of magnitude while ensuring deterministic instruction delivery. Apart from error correction, we also observe a large instruction bandwidth requirement for fault tolerant quantum instructions (magic state distillation). We extend QuEST to manage these instructions in hardware and provide additional reduction in bandwidth. With QuEST, we reduce the total instruction bandwidth by eight orders of magnitude.

CCS CONCEPTS

• **Computer systems organization → Quantum computing**;

KEYWORDS

Quantum Control Processor, Quantum Error Correction

MICRO-50, October 14–18, 2017, Cambridge, MA, USA

ACM ISBN 978-1-4503-4952-9/17/10…$15.00
https://doi.org/10.1145/3123939.3123940

ACM Reference format:
Swamit S. Tannu, Zachary A. Myers, Prashant J. Nair, Douglas M. Carmean, and Moinuddin K. Qureshi. 2017. Taming the Instruction Bandwidth of Quantum Computers via Hardware-Managed Error Correction. In *Proceedings of MICRO-50, Cambridge, MA, USA, October 14–18, 2017,* 13 pages.
https://doi.org/10.1145/3123939.3123940

1 INTRODUCTION

A *Quantum Computer* can solve fundamentally difficult problems which are unsolvable with conventional computers. It is envisioned as an accelerator that can accelerate specific class of problems. These problems range from breaking existing public-key encryption to designing better catalysts for reducing carbon emissions [17, 42]. Because of the potential for achieving substantial speedups, both industry and academia are actively developing a prototype quantum computer. To this end, industry leaders like Google and IBM have demonstrated quantum computers with nine and sixteen quantum bits (qubits), respectively. To enable scalability, Google and IBM are using *superconducting qubit* technology to fabricate the quantum substrate. Scaling the number of qubits is vital as it will enable the development of practical quantum applications. To this end, several industry labs are in the process of building 50 to 100 qubit machines [9, 44]. As we move into the regime of hundreds of qubits, quantum computers will face critical system software and architectural-level bottlenecks that will hinder their scalability [3, 16, 18, 24]. In this paper, we identify the instruction delivery to the qubits as a critical bottleneck that prevents the scalability of superconducting quantum computers. This paper aims to provide low-cost solutions for mitigating the instruction delivery bottleneck and enabling practical and scalable quantum computers.

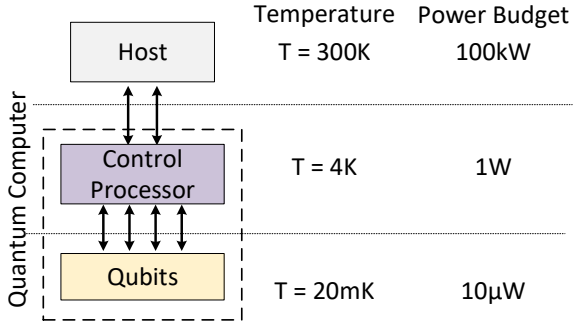

Figure 1: Organization of a superconducting quantum computer. Control processor operating at 4 Kelvin manipulate the qubits operating at 20 milliKelvin [24]).

Figure 1 shows an organization of a superconducting quantum computer. It consists of a control processor that is connected to a quantum substrate that is composed of qubits. The control processor is a conventional computer which acts as an interface between a programmer and qubits. The control processor can execute quantum algorithms by manipulating and measuring states of qubits. Unfortunately, qubits are highly susceptible to thermal noise and frequently turn erroneous, thereby causing them to lose their quantum information. To shield qubits against thermal noise, they are placed inside a refrigerator that operates at ultra-low temperatures (around 20 milli-Kelvin). Even at these ultra-low temperatures, qubits can only retain their data for few microseconds and experience error rates as high as 10^{-3}. To make matters worse, data loss from a qubit cannot be prevented by reading and refreshing their state (like DRAM) as reading a qubit destroys the information associated with the qubit. To improve reliability, theorists developed *Quantum Error Correction Codes (QECC)* that enable fault-tolerant quantum computing by mitigating erroneous qubits [41]. Currently, the QECC protocol is embedded in the execution model and is implemented by using a set of quantum instructions.[1]

The implementation of QECC also needs to be programmable as the field of quantum error correction is currently an active area of research. For programmability and flexibility, current proposals for quantum computer designs manage QECC at the software level. Based on this design, several prior works have recommended the programmer or the compiler insert the QECC instructions directly into the program's regular instruction stream. This mixed instruction stream is then provided to the control processor [2, 16]. For data integrity, the stream of QECC instructions needs to be executed across all qubits even when there are no quantum operations scheduled on a qubit. Such continuous and parallel execution of QECC results in an instruction bandwidth that scales linearly with the number of qubits. To estimate the increase in bandwidth as the size of the quantum computer is scaled, we analyzed the bandwidth requirements of Shor's factoring algorithm with different input sizes (ranging from 128 bit to 1024 bits). The results of this analysis are shown in Figure 2. As per our analysis, factoring a 1024 bit number requires an extremely high instruction bandwidth ($100TB/s$) as it requires millions of qubits.[2] Unfortunately, it is impractical to sustain this instruction bandwidth within the tight power budgets that dictate the operation of the quantum substrate and control processor operate [24].

If QECC instructions are delayed due to the lack of available instruction bandwidth, it can lead to uncorrectable errors. Unlike conventional systems, where one can cache instructions and sustain a large instruction bandwidth, instruction caching cannot be used for QECC instructions. This is because any non-determinism (from events like cache misses and tag-lookups) in the QECC instruction delivery will delay the execution of QECC. Even small delay (~100ns) in the execution of QECC can result in uncorrectable errors and the loss of quantum states. To make matters worse, as dictated by the *No-Cloning Theorem* [53], the state of a qubit cannot be copied. Due to this limitation, quantum applications cannot be checkpointed and recovered in case of uncorrectable errors. Therefore, the bandwidth required for deterministic execution of the instruction will hinder the scalability of quantum computers.

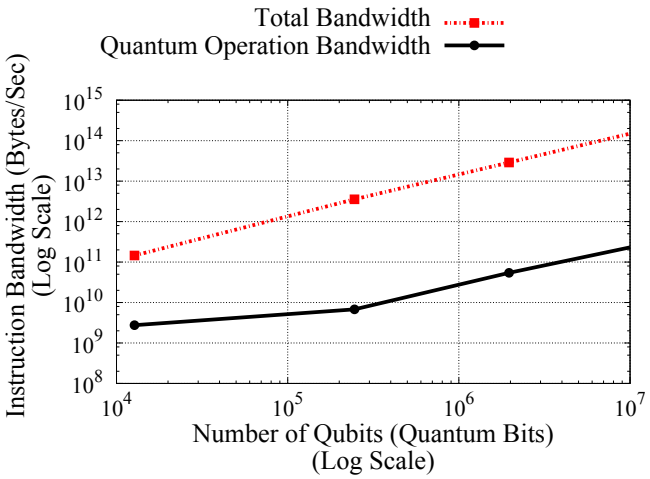

Figure 2: Instruction bandwidth for a superconducting quantum computer with increasing number of qubits.

To better manage instruction bandwidth, we propose *QuEST (Quantum Error-Correction Substrate)*, an architecture for the control processor that delegates the task of quantum error correction from software to the hardware and reduces the QECC overheads in the instruction stream. QuEST is based on three insights. First, QECC instructions form a significant majority of the instruction bandwidth (99.99%). Second, QECC instructions are independent of the quantum application control flow, and they can be executed without any global synchronization. Third, QECC instructions have a deterministic control flow and relatively small instruction footprints. QuEST manages instruction bandwidth by distributing the instruction delivery across simple *Micro-coded Control Engines (MCEs)*. Each MCE addresses a subset of the quantum substrate and manages QECC instructions using a tiny programmable microcode. This enables the MCEs to execute quantum error correction instructions without requiring any software coordination and global synchronization.

To maximize the number of qubits serviced by each MCE, we exploit underlying reuse patterns in QECC operations. By using this additional optimization, each MCE can support about 90× more qubits than the unoptimized design. By managing QECC instructions using micro-codes, QuEST decouples the delivery of QECC instructions and regular instructions, permitting non-determinism and enabling instruction caching for fault-tolerant logical instructions. By managing QECC in hardware near the quantum substrate, QuEST reduces the required instruction bandwidth by five orders of magnitude. Moreover, QuEST enables the non-deterministic delivery of the fault-tolerant (FT) logical instructions by decoupling the issuing of QECC instruction and FT instructions. This can be further leveraged to reduce the instruction bandwidth by three orders of magnitude by caching fault-tolerant logical instructions.

The control processor operates at a temperature of 4K, and it is placed close to the quantum substrate to maximize the connectivity between qubits and the control processor. At a temperature of 4K, conventional CMOS is not a feasible option as control processors

[1] We assume the gate compute model and use quantum gates and instructions interchangeably.

[2] The bandwidth bloat and the number of physical qubits required are calculated using the model described in appendix-M of [14]. It uses superconducting qubits that run surface codes to mitigate physical error rates of 10^{-4} per error correction cycle.

have small power budget [24]. To enable efficient designs within the small power budget, alternative technologies like Josephson-junction (JJ) based quantum control logic are proposed [20, 33, 46]. This is because JJ based logic is 1000x more power efficient and orders of magnitude more reliable than CMOS devices. In this paper, we assume the MCEs are fabricated using JJ based technology and that the MCEs use the surface code as QECC. This paper makes the following contributions:

- We identify instruction bandwidth as a major challenge that limits the scalability of superconducting quantum computers and show that the Quantum Error Correction instructions dominate the instruction bandwidth. As the number of qubits scale, we show that the software managed error correction is not viable as it leads to a linear relationship between instruction bandwidth and the number of qubits (Section 3).
- We propose QuEST, a quantum control processor architecture where QECC is handled by using a programmable microcoded engine hardware. We utilize the QECC microcode to continuously replay the QECC instructions without any software intervention (Section 4).
- We propose a scalable microarchitecture for the microcode engine using insights from the quantum execution model and QECC instruction locality. By doing this, QuEST reduces the instruction bandwidth by five orders of magnitude.
- We decouple the delivery of QECC instructions and fault-tolerant instructions to enable instruction caching for fault-tolerant logical instructions. This enables QuEST to provide eight orders of magnitude instruction bandwidth savings (Section 5).

2 BACKGROUND

In this section, we briefly discuss the fundamentals of quantum computing, the organization of a quantum computer, and the implementation of quantum instructions.

2.1 What is Quantum Computing?

Unlike conventional computers that use binary bits to encode information, quantum computers encode information using two-level quantum systems called qubits. These two levels can be denoted as vectors: $|0\rangle = \begin{pmatrix}1 & 0\end{pmatrix}^T$ and $|1\rangle = \begin{pmatrix}0 & 1\end{pmatrix}^T$. Unlike a classical bit, a qubit can exist in any state which is a superposition of the two levels. Mathematically, it can be expressed as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $|\psi\rangle$ is an arbitrary state of a qubit. When a qubit is measured or read, the quantum state collapses into a single classical bit that is either 0 or 1 with the probabilities α^2 and β^2 respectively. The measurement of a qubit is irreversible and it destroys the quantum information associate with a qubit.

Quantum instructions manipulate the state of a qubit by changing the magnitude and phase of the probability amplitudes α and β. For example, applying a X-gate flips α and β i.e $X|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$. Theoretically, an arbitrarily large number of quantum instructions are possible because even a small change in probability amplitudes produces a valid quantum state. However, most useful quantum instructions can be generated with a small instruction set of universal gates ($X, Z, H, S, T, CNOT$). Apart from these universal gates, the Prep_0 and Prep_1 instructions initialize qubits in either the $|0\rangle$ or $|1\rangle$ state, while the *Meas* instruction measures a qubit.

2.2 Organization of Quantum Computer

Figure 3 shows the organization of the quantum computer that is considered in our study. We assume the accelerator model for the quantum computer, where the quantum computer is connected to a traditional host which offloads quantum executables onto the quantum computer. The quantum computer is kept inside a cryogenic refrigerator with multiple thermal domains. A 77K domain holds cryogenic DRAM that keeps the instruction working set of the quantum application. A 4K domain holds a control processor. A 20 mK domain contains the quantum substrate. DRAM is used to contain the working instruction set of the quantum application as the instruction footprint for quantum algorithms is typically large (10s GB) [28, 29]. Additionally, conventional memories such as DRAM are empirically found to continue operating at 77K [19].

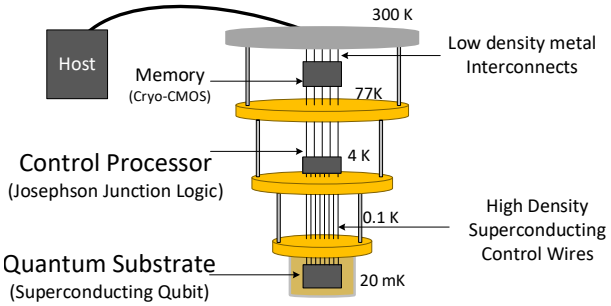

Figure 3: Organization of a scalable quantum computer.

We assume that the control processor operates at 4K and is constructed with JJ technology [36, 46]. In JJ technology, a Josephson junction serves as a switch to construct Boolean gates. Compared to CMOS gates, JJ based gates are 1000x more power efficient when clocked at 10 GHz [20]. Furthermore, JJ technology is extremely reliable at 4K and has a demonstrated bit error rate of 10^{-30} [20].

We assume that the control signals to the quantum substrate are routed using flexible superconducting interconnects, which provide high-density wires at very low leakage and thermal losses [43, 51]. This proposed interconnect technology can support more than million control wires at 20mK without violating the thermal budget. This paper focuses on quantum substrates built with superconducting qubit technologies that operate at 100MHz.

2.3 Implementing Quantum Instructions

The implementation of the quantum instructions depends on the qubit technology. For superconducting qubits, the state of a qubit can be manipulated by applying an electric signal at microwave frequencies. The application of the microwave signal on the qubit is equivalent to executing a quantum instruction on that qubit. Existing superconducting quantum computers utilize point to point connections between the qubits and a microwave generator, so every qubit device demands a dedicated waveform generator which limits the scalability.

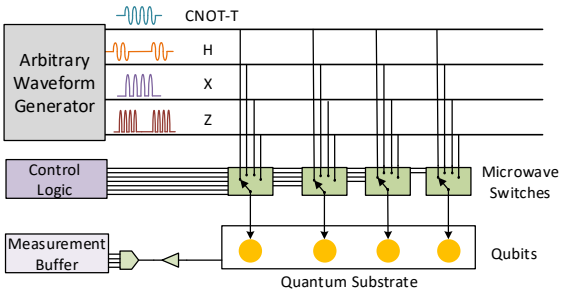

Figure 4: A Quantum Execution Unit with waveform generator and microwave switch. A quantum instruction provides the select signal to the microwave switches.

A recent work by Hornibrook et al. [24] demonstrates a scalable quantum execution unit called the Primeline Multiplexing Architecture. The primeline architecture uses a matrix of microwave switches to multiplex the outputs of a small number of expensive, arbitrary waveform generators (AWG) as shown in the Figure 4. The AWGs generate control pulses capable of manipulating and measuring qubits. The control pulses correspond to specific types of quantum instructions. The AWG continuously passes the pulses to the switch matrix through an analog bus called the *prime-line bus*. The switch matrix acts as an analog multiplexer that is controlled by select bits from a digital address bus. The select bits on the bus determine which waveform is directed to which qubit in a given cycle. The quantum instructions in this architecture are simply the appropriate bits in the select line of the switch matrix. In this paper, we assume that the quantum execution unit is based on the primeline architecture.

3 QUANTUM ERROR CORRECTION

In this section, we discuss the need for error correction in quantum computing and how this dramatically increases the instruction bandwidth for quantum workloads.

3.1 Need for Quantum Error Correction

Qubits are sensitive to noise and can lose their state information through interactions with their environment. This loss of state is called *Decoherence*. For superconducting qubits, coherence time is up to 20μS [4] and the probability of decoherence increases exponentially as time passes. Experiments indicate the average error rate of 10^{-3} per 100nS for a superconducting qubit device [38]. In addition to decoherence effects, quantum instructions have low fidelity and can produce erroneous states. To make quantum computing feasible, *Quantum Error Correction Code* (QECC) was developed to mitigate these errors [41]. At a high level, a QECC is composed of a continuous loop of quantum instructions that correct memory errors and the use of redundant data encoding to enable reliable quantum instructions.

3.2 QECC Implementation

QECC detects and corrects errors by generating error syndromes. To generate error syndromes, the QECC algorithm entangles extra 'ancillary' qubits with data qubits. The algorithm then measures the ancillary qubits in the error syndrome to detect errors. The measurement destroys the entangled information of the ancillary qubits while leaving the data qubits unaffected. After that, the ancillary qubits and data qubits are re-entangled to produce a new error syndrome for the next QECC cycle as shown in Figure 5. This cycle is performed continuously to avoid non-correctable errors.

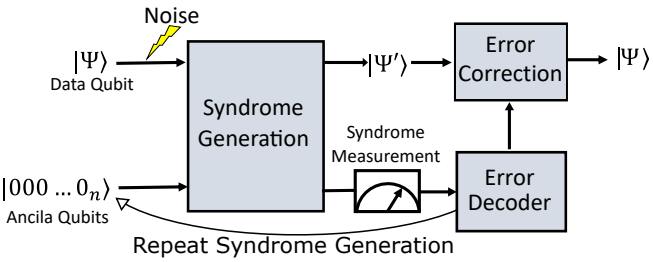

Figure 5: Quantum error correction instructions continuously generate and measure syndromes by entangling and measuring the ancillary qubits.

QECC improves the reliability of instructions by grouping several fragile physical qubits into a single 'logical' qubit. By using this abstraction, fault tolerant quantum instructions can be built that amortize low instruction fidelity. In a proposed quantum programming model, a programmer composes a quantum application with fault tolerant logical qubits and instructions [16]. A compiler then expands each logical instruction into a group of parallel physical instructions and streams these instructions to physical data and ancillary qubits. The choice of QECC determines the underlying implementation of error syndromes and logical qubits. In this paper, we utilize state of the art Surface Code based error correction schemes. A brief design overview for surface code protocol is described in Appendix A.

3.3 QECC Dominates Instruction Stream

Current quantum programming models manage QECC at the software level. This provides the programmer with the flexibility to choose the type of QECC they want, including pre-compiled error correction software libraries [16]. However, managing QECC at the software level requires the host to transmit all the QECC instructions to the control processor through the same channels as program instructions. Since QECC issues continuous parallel instructions, its required bandwidth instruction grows significantly, and the specification for the control processor becomes impractical.

For example, let's assume a superconducting qubit that operates at 100 MHz with byte sized quantum instructions. To maintain data integrity, each physical qubit needs to receive the QECC instructions close to its operating rate and thus requires 100 MB/s of instruction bandwidth. At small scale, this linear relation between qubits and instruction bandwidth may be tolerable. At large scale, a large instruction bandwidth can hinder the design of large-scale quantum computers. For example, a quantum computer with 100,000 qubits will require 10TB/s of instruction bandwidth within a tight temperature and power budget. Worse, unlike conventional systems, the bandwidth is not only a performance bottleneck but a

correctness bottleneck. If QECC instructions are not issued at the desired rate, the physical qubits experience a delay in the QECC cycles thereby increasing their error rates.

To understand the severity of the QECC overhead, we estimated the bandwidth overhead of seven quantum workloads with QuRE [47] by calculating the ratio of QECC instructions to the actual workload instructions. Figure 6 shows the ratio of QECC instructions to the useful instructions for various workloads. Almost 99.999% bandwidth is dedicated to the QECC instructions.

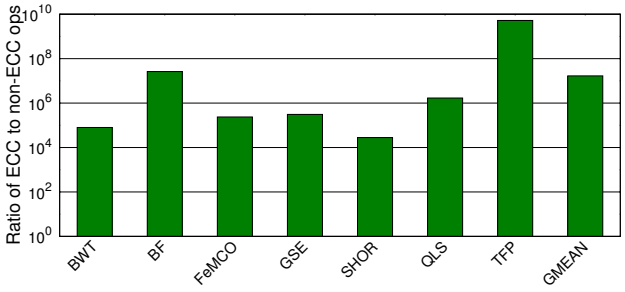

Figure 6: Ratio of QECC instructions to regular instructions in quantum workloads. QECC requires an instruction overhead of 4 to 9 orders of magnitude.

3.4 Need for Deterministic Instruction Supply

Unfortunately, traditional methods of managing high instruction bandwidth do not translate to quantum computing. For instance, conventional computers cache instructions to manage high instruction bandwidth. However, these instruction caches introduce non-deterministic latencies (due to cache misses and tag lookup) that are unacceptable when executing QECC instructions as it increases the error rate. To minimize the complexity of QECC, it is essential to have a deterministic supply of instructions from the control processor to each of the qubits.

4 QUEST: OVERVIEW AND DESIGN

To manage the instruction bandwidth of quantum computers, we propose a control processor architecture called *QuEST (Quantum Error-Correction Substrate)*.

4.1 Insight

We design QuEST based on four key insights. First, with conventional software-managed error correction, QECC instructions contribute more than 99.999% of the total instructions in the instruction stream of a quantum workload. Second, it is necessary to deliver QECC instructions in a deterministic fashion to ensure the reliable operation of the quantum substrate. Third, QECC instructions have high parallelism and can be performed independently without any global synchronization. Fourth, QECC instructions are simple enough that microcode tables can store and deliver micro-operations without sacrificing determinism and programmability.

4.2 Control Processor Organization

In QuEST, the control processor is divided into an array of dedicated *Microcode Control Engines (MCEs)* placed at 4K temperature and a master-controller. These are all connected via a global data and instruction bus as shown in Figure 7. Each MCE manages a tiled subsection of the quantum substrate and executes logical and QECC instructions for that tile. Each MCE is solely responsible for issuing QECC instructions for its tile. These tiled subsections are composed of a fixed number of predefined qubits and are independent of each other. The master-controller orchestrates all logical operations by dispatching the logical instructions to the appropriate MCEs.[3]

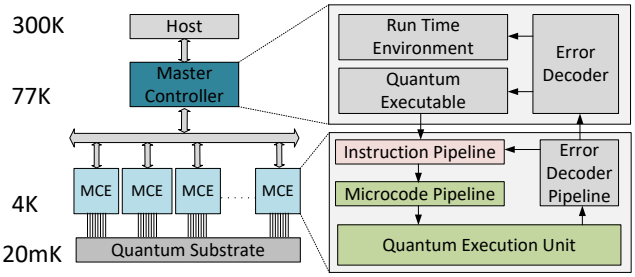

Figure 7: QuEST uses independent hardware units called as Microcoded Control Engines (MCE), to deliver instructions to a fixed qubit group.

An MCE has four functional blocks: instruction pipeline, microcode pipeline, quantum execution unit, and error decoder pipeline, as shown in Figure 7. The instruction pipeline (IP) delivers logical instructions and translates them into physical instructions. The microcode pipeline (MP) feeds these physical instructions to the quantum execution unit. The quantum execution unit is based on a prime line architecture. It consists of an AWG and a microwave switch array controlled by physical instructions. The error decoder pipeline is a part of a two-level error decoding scheme. The error decoder collects the syndrome measurement data and performs a limited local error decoding with a lookup table to correct frequently occurring isolated single-qubit errors. Additionally, there is a single global decoder in the master-controller that decodes more complex error patterns [50]. The *global bus* between the MCE and master controller carries both logical instructions and error syndrome data. The master controller delivers logical instructions to MCE using a packet switched network. In this paper, we focus on solving the instruction bandwidth problem by optimizing the instruction and microcode pipeline.

4.3 Microcode Pipeline

The microcode pipeline (MP) executes all physical instructions translated from logical and QECC instructions. The MP consists of a microcode memory and an address decoder as shown in Figure 8a. To guarantee that each qubit receives an instruction per cycle, the physical instruction is designed similar to a very long instruction word (VLIW) and composed of a μop per qubit. These instructions are executed in lockstep for all qubits to ensure parallel QECC operation. The parallel constraint stems from the need to construct a continuous lattice of error syndromes as discussed in Appendix A. The microcode pipeline architecture ensures the complete control

[3]Master-controller is assumed to be placed at a higher temperature (77K) and is fabricated using CMOS technology as it requires a complex architecture to handle the run-time environment and the error correction decoder.

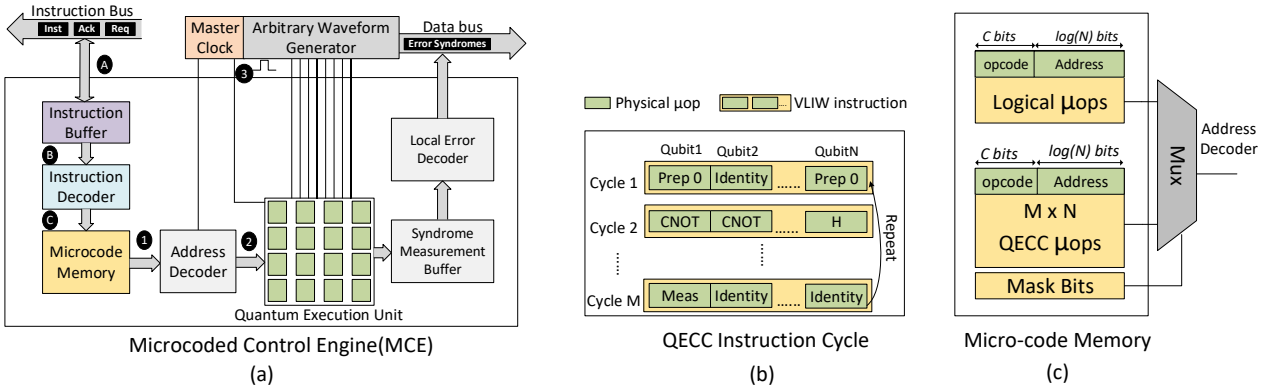

Figure 8: (a) An overview of MCE microarchitecture. (b) QECC instruction cycle. QECC cycle repeats after M instructions. (c) Microcode memory organization with QECC-μop table, Logical-μop table and mask table. QECC μop contains fully decoded μops used to execute QECC cycle. Where each μop has op-code and address bits.

over all the quantum operations at any point of time so that no qubit remains idle.

Inside the MCE, a physical instruction is executed in three steps as shown in Figure 8a. In step ❶, the μop corresponding to an instruction moves from the microcode memory to the address decoder. In step ❷, the decoder delivers the μop to the corresponding microwave switch, and μop is latched onto it. The μop corresponds to the specific waveform as discussed in Section 2.3. Steps ❶ and ❷ are repeated until all the microwave switches in the execution unit are latched with the corresponding μop. In step ❸, a master clock is applied to both the AWG and all the microwave switches which activate the switches and results in parallel synchronized execution of quantum instructions.

This execution model guarantees a complete control over all qubits at any point in time. The timings for all the instructions are therefore predetermined by design and are managed by the master clock. These steps are pipelined to improve the throughput and minimize the delay between two quantum instructions. When a microwave switch is active and passing microwave signals from the AWG to the qubits, a μops corresponding to next instructions can be latched onto the microwave switches.

4.4 Managing QECC Bandwidth

The primary objective of QuEST is to deliver QECC instructions without intervention from the master controller. To achieve this goal, we partitioned the microcode memory into a QECC-μop memory, a logical-μop memory and a mask table as shown in Figure 8c. QECC-μop table contains the fully decoded QECC-μops which can be directly latched into Quantum Execution Unit by accessing the QECC-μop table. Figure 8b shows a QECC instruction cycle, where all the qubits are assigned an instruction.

The mask table holds mask-bits that decide if the μop for a given qubit should be fetched from logical μop or QECC-μop memory. If there are no logical instructions in logical-μop table then the MP replays QECC-μops continuously to generate error syndromes.[4]

The size of the microcode table and the design of the QECC protocols depend on the number of instructions in the QECC cycle and the size of instructions. Both the number and size of the instructions depend on the type of QECC used. For our architecture, we use a "simulacrum" of existing surface code QECC cycle (syndrome generation circuit depth) that is approximately 9 to 14 instructions long [14, 50]. The last instruction in this QECC cycle measures the syndrome qubits. In our design, the resulting measurement is fed into the error decoder pipeline which resolves errors and reports them to the master controller.

Our architecture makes the QECC portion of microcode table programmable, so the choice of QECC is flexible. However, even with a different error correction scheme, the number of instructions (quantum circuit depth) in a syndrome generation cycle is expected to be small as the cycle size and error rate (due to decoherence and instruction fidelity) are coupled. We reasonably assume that number of QECC instruction in a QECC cycle (syndrome generation circuit depth) will be short and that it is possible to supply QECC instructions from a reasonably sized microcode. A detailed feasibility analysis and design optimization are discussed in the next section. However, it will be shown that designing a microcode table using JJ technology faces some challenges.

4.5 Microcode Architecture

The main limitation of JJ technology is the lack of a dense memory. Existing JJ fabrication technology has limited JJ density (10^6 to 10^8 JJs per cm^2) which is almost 100× to 10000× smaller than state of the art CMOS technology [21, 49]. Additionally, the high complexity of JJ-based memories also limits the memory capacity as a memory element requires a large number of JJs. The limited memory capacity also has a direct impact on our design since this limits the capacity of our microcode memory. This places a ceiling on the number of qubits serviced per MCE which restricts the overall scalability of our quantum computer. While it is difficult to quantify the available memory capacity for the state of the art JJ process technology due to limited data, conservative estimations of memory capacity from older fabrication processes suggest a memory density of ~4Kb/cm^2 [49]. Optimistic estimations of memory capacity using state of the art technology suggest a capacity of ~400Kb/cm^2 [11, 12]. Limited memory capacity is not the only design challenge. The bandwidth of the microcode memory also limits the number of qubits per MCE, especially for the lockstep

[4]The function of mask table is elaborated in Section 5.1 which describes the execution of logical instructions.

VLIW execution model. This model assumes a microcode table that delivers instructions to all the qubits in one cycle (each cycle is 10ns-20ns long).

However, we can reduce the capacity needed for QECC-microcode with the following insights: In a quantum execution unit, physical instructions are executed in lock step. All μops are first latched onto all microwave switches, and then the quantum instruction is executed. This process does not require random access to the μops as all qubits are addressed every execution cycle. Additionally, the order of latching the instructions is not important so the address bits can be omitted from the μops. This means the QECC microcode memory can be designed as FIFO. This reduces the capacity required per instruction, so the total QECC microcode capacity will scale with $O(N)$ instead of $O(Nlog_2(N))$. This improves the scalability by 3 to 4 times as shown in the Figure 10.

To reduce the capacity requirement further, we can utilize the underlying instructional similarity in the surface code. The basic unit of a surface code is the set of spatially repeating syndrome generation instructions that are executed on all the physical qubits in order to generate spatially periodic syndromes as shown in Figure 9. The spatial granularity of the repeating instructions is a unit cell of 25 qubits [14]. By replaying the unit cell op-codes spatially, one can generate the complete sequence of instructions for a surface code error correction cycle. A small set of instructions corresponding to the unit-cell can be stored in a microcode memory and replayed using a state machine. This optimization effectively manages QECC instructions with a fixed memory capacity such that QECC microcode capacity scales with $O(1)$ which improves the scalability as shown in Figure 10.

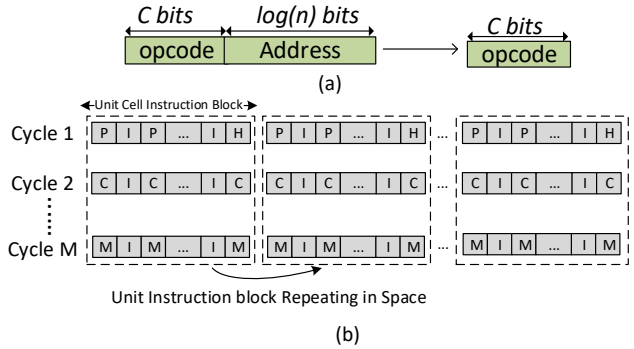

Figure 9: (a) In FIFO optimization, address bits for a μop are dropped. (b) QECC instructions have a repeating unit instruction block. Only unit block instructions can generate instructions for all of the qubits.

The mask memory requires a memory capacity of N bits per MCE, as each qubit requires a mask bit to select between QECC instructions or logical instructions, so its capacity scales on the order of $O(N)$. However, for the surface code, the storage can be reduced if a coalesced mask bit is used for a small group of qubits. For surface codes, all the logical instructions have the operational granularity of d^2 physical qubits where d is the code distance.[5] For N physical qubits, only N/d^2 mask bits are used. As the shape and dimension of the mask are pre-determined, coalesced mask-bits can be used to mask QECC instructions. The proposed capacity optimization is based on the locality and usage patterns of QECC instructions. Although the design is specific to surface codes, the underlying insights can be applied to other classes of quantum error correction codes such as planner codes, color codes and triangle codes [27, 50, 54]. For example, the 4×4 unit cell can be used for the class of surface code discussed in [50].

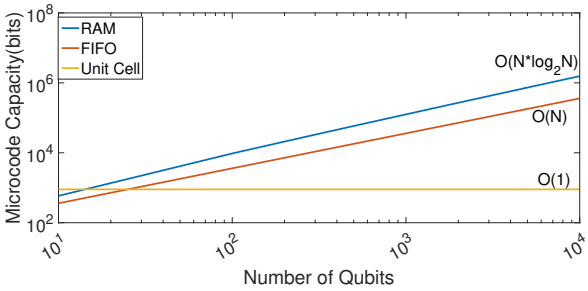

Figure 10: Trends in required memory capacity vs number of qubits serviced for microcode designs.

In QuEST, all the contents of microcode memory (both mask and QECC microcode) are streamed out every cycle which creates a large bandwidth demand. This bandwidth requirement for microcode memory is $\frac{N \times InstructionSize}{InstructionLatency}$ where the instruction latency is roughly 10 ns. The proposed FIFO optimization reduces the raw bandwidth demand from $O(Nlog_2N)$ to $O(N)$ since it reduces the instruction size. However, the unit cell optimization does not reduce the bandwidth directly since we need bandwidth to replay QECC instructions. However, the capacity reduction from the unit cell optimization indirectly affects the output microcode bandwidth. A smaller capacity microcode has a smaller read latency, i.e. it can deliver more op-codes to Quantum Execution Units per unit time.

To find the efficacy of our optimizations, we calculated the number of qubits serviced per MCE for each optimization for one-channel, two-channel and four-channel memory configurations for a fixed memory capacity of 4Kb.[6] The one channel memory is a 4Kb memory array with one read port. The two channel is a 2×2Kb memory with two independent memory arrays with a read port each. The four channel memory is 4×1 Kb memory with four independent memory arrays. Figure 11 shows the results of this experiment. The baseline design assumes no capacity optimization. QECC instructions are managed by software and buffered in a JJ based memory with a conventional encoding of op-code and address bits. In the baseline design, the number of serviced qubits is limited by the microcode capacity since the capacity scales with $O(N \times log_2(N))$. A 4Kb microcode memory can only hold 48 qubits worth of QECC instructions and addition of channels does not affect the number of serviceable qubits.

For FIFO optimization, microcode memory capacity scales as $O(c \times N)$ where c is the op-code size. For a 4 bit op-code, the number of serviced qubits is limited to 120 due to the memory capacity. For Unit cell optimization, a number of serviced qubits is decoupled from the memory capacity since the number of QECC instructions

[5] A code with the distance of d can detect $\frac{d-1}{2}$ errors in a single error correction cycle.

[6] 4Kb memory requires about 170,000 JJs over 1 cm^2 and consumes about 10μW [11].

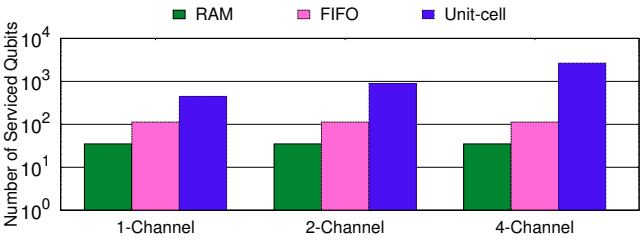

Figure 11: Number of qubits serviced per MCE for microcode designs with a fixed 4Kb microcode memory.

for a unit cell easily fit into a 4Kb microcode memory. The number of serviced qubits now only depends on the memory bandwidth as shown in Figure 11 where adding more bandwidth improves the number of serviced qubits. For a microcode design with unit cell optimization, a linear increase in channels provides a super-linear increase in bandwidth due to the small capacity of the microcode bank. For a one channel 4Kb, the memory access latency is three cycles [11]. For a four-channel 1Kb memory configuration, the read latency decreases to 2 cycles, and the bandwidth improves by 6x.

5 INSTRUCTION PIPELINE

In this section, we briefly overview how logical instructions are executed in QuEST. Later we discuss a source of logical bandwidth bloat and how QuEST can be extended to solve it.

5.1 Executing Logical Instructions

Surface codes group physical qubits into a block and disable the execution of quantum error correction within that block to create a logical qubit. Figure 12a shows a section of the surface code composed of both data and ancillary qubits (for a description of surface codes see Appendix A). The surface code section in Figure 12a does not contain any logical qubits. To create a logical qubit, the syndrome generation and measurement is turned-off. Figure 12b shows logical qubits Q1 and Q2 that were created by turning off (or masking) error correction for all the ancilla qubits inside the area and on the perimeter of the highlighted squares.

A single logical qubit can be created with two masked squares briefly connected during initialization (Fowler et. al [14] describes this process in detail). The logical qubit with code distance of d can detect $\frac{d-1}{2}$ errors every error correction cycle and requires at-least d qubits on the perimeter of each square mask. The distance between two masks should also be d data qubits. This requires total $12.5 \times d^2$ physical qubits per logical qubit (as per Appendix-M of [14]). Broadly, there are two categories of logical instructions that manipulate logical qubits: transverse instructions and mask instructions. Transverse instructions, like SIMD instructions, are applied on all physical qubits inside a logical qubit. For example, to prepare a logical qubit in the $|0\rangle$ state, a transverse $Prep_{Logical}$ instruction is applied to all physical qubits inside the logical qubit and initializes all the physical qubits to $|0\rangle$. Mask instructions move, expand and contract the boundary of logical qubits by changing the QECC-mask. Figure 12c shows an intermediate braiding step for the logical CNOT operation where the boundary of the logical qubit-Q1 is expanded and braided along the boundary of the logical qubit-Q2 by altering the bits in the mask-table.

The Instruction Pipeline (IP) supports transverse and mask instruction in hardware. This pipeline consists of an instruction buffer, an instruction decoder, and a logical μops table as shown in Figure 8a. The IP executes logical instructions in three steps. Step (A), the MCE receives packets of logical instructions from the master-controller and places them in the instruction buffer.[7] Step (B), the decoder decodes the instruction and, in step (C), places the resultant μops into the logical microcode. To define a logical qubit and execute a mask instruction, the instruction decoder sets bits in the mask table to disable QECC μops for a defined boundary. Note that the interleaving of QECC in logical instructions is fixed and the part of Quantum Error Correction protocol design.

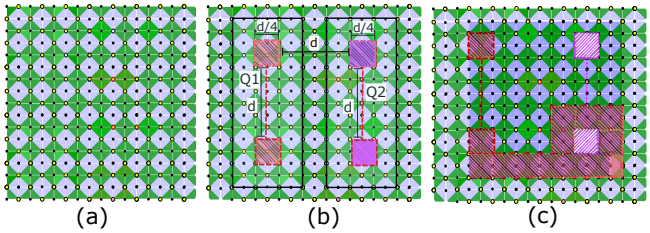

Figure 12: (a) Surface code substrate with data and ancilla qubits. (b) Logical qubit Q1 and Q2 created by masking error correction instructions over group of qubits. (c) State of the logical qubits can be manipulated by growing or shrinking their boundaries and braiding.

5.2 The Logical Bandwidth Problem

Most quantum workloads execute only two to three logical instructions in parallel [18, 28] hence the required logical instruction bandwidth is expected to be small. However, workload analysis shows that required logical instruction bandwidth is substantial even though logical instruction level parallelism is small. This high demand stems from just one class of complex logical instruction called T-gates or T-instructions. The T-gate rotates the qubit by $\pi/4$ about the Z-axis. It is one of the gates required to form a universal set of quantum gates. Its use is essential to perform arbitrary rotations [29]. The fault-tolerant logical T-gate operation requires ancillary qubits in a specific state known as the magic state. Unfortunately, magic states can only be generated by a family of recursive algorithms, called distillation processes, that consume many execution cycles. Magic state distillation quickly bottlenecks almost all analyzed workloads as these workloads require magic states for T-gates roughly every third instructions. Overall, T-gate instructions constitute 25% to 30% of the instruction stream. One solution to avoid stalling T-gate instructions is to execute many parallel instances of the distillation algorithm [25]. Each instance of distillation is called a T-factory. T-factories require a large portion of the instruction bandwidth as they require continuous execution of parallel instructions. Figure 13 shows the T-factory bandwidth overhead. The overhead is the ratio of distillation instructions to

[7] Note that the arbitrary rotations are not translated at the MCE. They are either decomposed at run-time (by the master controller) or at compile time (by the Host). The Instruction Pipeline (IP) then translates only the logical Clifford operations (CNOT, H, X, Z) into corresponding physical instructions or μops.

algorithmic instructions. Like QECC instructions, the demand for the distillation instructions is deterministic and continuous.

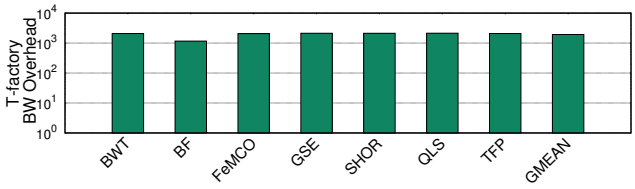

Figure 13: The ratio of T-factory instructions to total quantum application logical instructions in the instruction stream of quantum workloads.

5.3 Enabling Logical Instruction Caching

As QuEST decouples the logical and QECC instructions, we can extend the MCEs to cache logical instructions and reduce logical instruction bandwidth without impacting reliability. Additionally, the distillation algorithms are recursive and have a deterministic control flow. We can leverage this observation and use the instruction buffer as an instruction cache. The size of instruction cache depends on the specific distillation algorithm used. A typical distillation algorithm has 100 to 200 logical instructions. We evaluated the feasibility of adding the cache to the logical buffer while assuming our quantum ISA was similar to the one proposed by Balensiefer et al. [2] and that quantum instruction size was fixed at two bytes. We found that we could architect an instruction buffer as a software-managed cache and reduce the global bandwidth further by three orders of magnitude.

6 METHODOLOGY

In this section, we briefly describe the quantum workloads and the tools used in our evaluations of QuEST architecture.

6.1 Quantum Workloads

Current prototypes of quantum computers are small and unsuitable for executing realistic quantum workloads because they require several hundred thousand to millions of qubits. For our studies, we assumed our quantum substrate has a sufficient number of qubits to solve these realistic quantum workloads. We used six different quantum workloads drawn from a ScaffCC workload suite [26] and recent quantum chemistry applications [17]. A brief description of these workloads is as follows:

- Binary Welded Tree (BWT) [7]: The benchmark utilizes quantum random walk algorithm to find a path between an entry and exit node of a binary welded tree.
- Boolean (BF) [1]: This benchmark uses the quantum algorithm to provide the best strategy for the game of hex.
- Ground State Estimation (GSE): This quantum chemistry application uses the phase estimation algorithm to compute the ground state of the Fe_2S_2 molecule which is essential to understand photosynthesis.
- Ground State Estimation of (FeMoCo) [17]: This quantum chemistry application computes the ground state of the active site of a molecule which acts as a catalyst in biological nitrogen fixation. This application can increase the efficiency of fertilizer production.
- Quantum Linear System (QLS) [31]: QLS finds the solution to a linear system Ax=b. It shows a polynomial speedups compared to classical algorithms and can help domains such as machine learning and optimization.
- Shor's Algorithm (SHOR) [42]: Shor's factoring algorithm can factor numbers in polynomial time. Prime factorization is a classically hard problem and it scales exponentially. The exponential scaling of the factorization is the foundation of RSA encryption standard. However, with a quantum computer running the Shor's algorithm, it is possible to break RSA and other public key encryption systems.
- Triangle Finding Problem(TFP) [32]: This benchmark finds a triangle within a dense undirected graph. It is parameterized by the number of nodes n in the graph.

6.2 Evaluation Framework

We evaluated the total bandwidth requirement with modified *QuRE-Toolbox* and Scaffold [26]. QuRE is a resource estimation tool for quantum algorithms [47]. It uses analytical models to evaluate execution time and the total number of physical qubits and physical instructions for a particular algorithm.[8] It requires technology parameters such as quantum instruction latencies and error rates, along with the design parameters of a quantum error correction code. For this work, we assumed the substrate was made of superconducting qubits with the technology parameters tabulated in Table 1. The experimental instruction latencies are based on real qubit devices described in [50]. $Projected_D$ are projected gate latencies proposed by DiVincenzo that are widely used in QECC studies [10]. For modeling microcode memory we use technology parameters described in [11, 12].

Note that the logical qubits described in the Figure 12c cannot guarantee the braiding of multiple logical qubits in parallel. Parallel braiding of multiple logical qubits is essential for parallel CNOT operations. However, our evaluations utilize 2-D layout of logical qubits and employ a qubit definition that is used by QuRE. Such a representation uses extra physical qubits to allow multiple parallel CNOTs and braiding operations without moving the logical qubits. This definition of a logical qubit assumes $7d$x$3d$ patch of physical qubits per logical qubit and requires 6x more qubits as compared to the optimal logical qubit described in Figure 12. This leads only to 0.1% increase in the estimated number of physical qubits and the instruction bandwidth for the workloads used in our studies.

Table 1: Technology Parameters

Parameter	Description	$Experimental_S$	$Projected_F$	$Projected_D$
t_{prep}	state preparation	$1\mu s$	$40ns$	$40ns$
t_1	single-qubit	$25ns$	$10ns$	$5ns$
t_{meas}	measurement	$1\mu s$	$35ns$	$35ns$
t_{CNOT}	CNOT	$100ns$	$80ns$	$20ns$
T_{ecc}	one round	$2.42\mu s$	$405ns$	$165ns$

[8] QuRE assumes zero overhead on the movement of the logical qubits to evaluate the execution time which may not be true in realistic quantum machines. This assumption leads to underestimation of the execution time of a quantum application and overestimation the logical instruction bandwidth. However, we are interested in the fraction of the ECC bandwidth to logical bandwidth. This ratio is largely independent of the execution time and is dependent only on the fraction of QECC instructions in the instruction stream.

7 EVALUATIONS

In this section, we evaluate the total bandwidth savings of QuEST over the baseline design that manages the error correction in software. We also examine how Quantum Error Correction protocol design impacts the throughput of the MCEs.

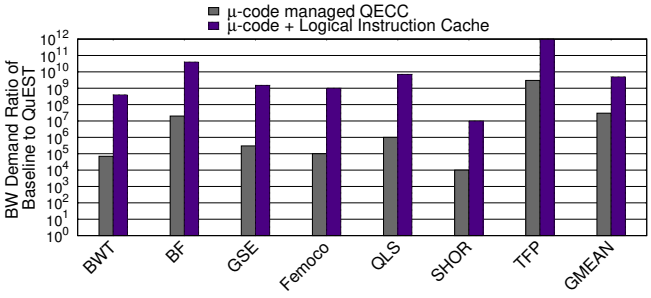

Figure 14: Global bandwidth savings with QuEST.

We use QuRE to evaluate the reduction in global instruction bandwidth by calculating the total instruction bandwidth with and without QECC overheads. In the baseline implementation, the compiler generates the physical instruction stream corresponding to the logical instructions, including the magic-state distillation instructions and QECC instructions. In QuEST, the architecture manages error correction using dedicated MCEs. Now, only the application's logical instructions and the master-controller's synchronization tokens contribute to the global bandwidth. QuEST also manages magic state instructions locally at the MCEs with software managed logical instruction caches. Synchronization tokens manage instruction cache and facilitate logical qubit movement and complex logical instructions across MCEs.[9]

We evaluated QuEST for the quantum workloads described in Section 6.1 over three different set of qubit technology parameters and two different error-syndrome designs: Shor-syndrome and Steane-syndrome. The qubit parameters are specified in table 1. Each error-syndrome design requires a different number of instructions in each QECC cycle. The Shor syndrome based design needs 14 instructions per qubit to generate and measure the syndrome. The Steane syndrome based design requires fewer instructions that the shor-syndrome and only needs nine instructions per qubit to generate and measure the syndrome. For all our evaluations we assumed an error rate of 10^{-4}.

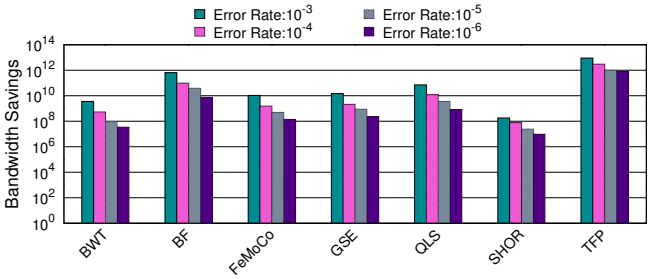

Figure 15: Sensitivity of global bandwidth savings for variable qubit error rates.

[9]Logical instructions across MCEs are not evaluated in this paper

Figure 14 shows the global bandwidth savings for quantum workloads run on QuEST with $Projected_D$ technology and Steane-syndrome. These results represent all configurations as both the technology parameters and the syndrome design have little impact on bandwidth savings with a coefficient of variation (standard deviation divided by the mean) of 0.0002% between configurations. Managing QECC instruction in the MCEs reduces the instruction bandwidth by at least five orders of magnitude. The addition of logical instruction caches to the MCEs reduces the instruction bandwidth by another three orders of magnitude. Overall, the QuEST architecture reduces the instruction bandwidth by almost eight orders of magnitude.[10]

Figure 15 shows bandwidth savings for three different qubit error rates. A reduced error rate lowers the total number of physical qubits needed to run a workload. In turn, this reduces total instruction bandwidth required in the baseline design. Most of this bandwidth reduction results from the reduction in quantum error correction bandwidth bloat. The bandwidth overheard of magic state distillation remains constant for different error rates because of the overhead of distillation scales with the number of T-factories and the number of T-factories scales sub-linearly ($C^{log|log(e_r)|}$) with the error-rate (e_r).

Table 2: QECC microcode design

Syndrome	No. Instructions	Optimal μCode Configuration	No. JJs	Power
Steane	148	4 Channel = 1Kb x 4	170048	2.1μW
Shor	300	2 Channel = 2Kb x 2	168264	1.1μW
SC-17	136	8 Channel = 512b x 8	163472	5.6 μW
SC-13	147	4 Channel = 1Kb x 4	170048	2.1μW

The technology parameters and syndrome design have significant impacts on the MCEs' throughput (the number of qubits serviced per MCE). Syndrome design decides the unit cell size and number of sub-cycles (circuit depth) in a QECC cycle. Technology parameters, like the single qubit instruction latency, decide the desired peak bandwidth of the microcode memory. Table 2 shows the optimal microcode memory configurations, a corresponding number of JJs and total power dissipation for different syndrome designs. Figure 16 shows MCE throughput for three different technologies and four syndrome designs. SC-17 and SC-13 are optimized codes with 17 and 13 qubit unit-cell respectively [50].

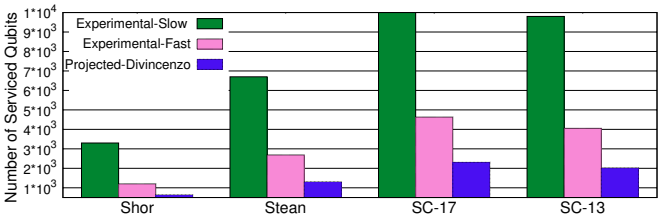

Figure 16: MCE throughput with syndrome designs.

[10]The instruction bandwidth bloat from QECC, is directly proportional to the number of physical qubits in a quantum computer. Improvements in the magic state distillation algorithms and the QECC protocols may reduce the number of physical qubits and the bandwidth bloat. Similarly better magic state distillation algorithms may increase the number of magic states delivered per T-factory and the instruction bandwidth requirement for the T-factories [15]. We expect the results reported in this study to change depending on the theoretical and experimental improvements in the future.

8 RELATED WORK

Quantum architecture is an active area of research, and we direct readers to these studies [24, 34, 35, 52] to understand the quantum computing system stack. Balensiefer et al. [2] identified the overhead of error correction for the ion-trap computers and proposed a framework to understand hardware constraints on the system reliability. Several other works highlighted the impact of quantum error correction on the design of microarchitecture and proposed solutions to optimize for area and performance overheads [37]. Thakkar et al. [48] focused on the organization of compute qubits and memory qubits. The authors observed the temporal locality and reuse patterns in the quantum workloads and proposed a trade-off between reliability and latency to optimize the area and execution time. Researchers have also proposed distributed SIMD architectures and associated scheduling policies to support quantum operations on a distributed ion-trap based quantum substrate [18, 28, 29]. Prior works have also explored the microarchitectural trade-offs for *eSHe* qubits and silicon qubits [6, 8].

Outside of technology constraints, quantum workloads and programming also present a unique set of challenges as highlighted by Patil et al. [39] and Schuchman et al. [40]. A large body of work has also focused on compiler and programming-language level problems for quantum computers. One of the major challenges in this space is handling the extremely large executables. Work in [26] demonstrated a tool-chain which enables quantum compilation.

9 IMPACT OF TECHNOLOGY ON QUEST

We focus on superconducting quantum computers running defect based surface code to highlight the instruction bandwidth problem. To mitigate bandwidth bloat, we envision a distributed control processor that handles QECC instructions using microcode by leveraging the deterministic and asynchronous instruction execution of QECC instructions. We optimize the capacity and bandwidth of the microcode table by exploiting the underlying unit-cell structure in surface codes. These insights are fundamental, they are not limited to any one qubit technology or a specific QECC and can be extended for other error correction protocols and qubit technologies. For example, QuEST can work with concatenation codes where the first level (inner code) is handled by microcode and higher level (outer code) concatenations can be handled by software. The hardware/software interface for quantum error correction is still an open problem. QuEST proposes solutions to overcome this problem by highlighting the benefits of hardware managed Quantum Error Correction primitives.

Cryogenic control is one of the dominant constraints for not only superconducting qubit technologies but also for other promising quantum technologies [45]. Quantum dots, cryogenic NV-centers and fully microwave controlled ion-traps [30] will also have to deal with the challenges of cryogenic control. To this end, both industry and academia have highlighted the need to investigate cryogenic control to enable scalable quantum computers [3, 5, 22–24]. Nonetheless, the effectiveness of QuEST is not limited to quantum computers which require cryogenic environments. The distributed control processor with MCE-units can reduce the overall cost even at room temperature and the reduction in instruction bandwidth from handling the QECC with MCEs would still be substantial.

10 CONCLUSION

As quantum computers grow from a few qubits to potentially thousands of qubits, they will face several systems and architectural challenges to scalability. Now is the time for architects to identify bottlenecks for large-scale quantum computers and propose practical solutions as these solutions have the potential to enable the future of quantum computers. In this paper, we identified instruction bandwidth as one of the key bottlenecks to scaling. As the quantum substrate relies on a deterministic instruction delivery, the inability to supply the necessary instruction bandwidth will result not only in slowdowns but also limit the computer's function and scalability. Practical solutions that reduce the demand for instruction bandwidth allow larger scale quantum computers. To that end, this paper proposes QuEST, a control processor microarchitecture, which manages QECC bandwidth with dedicated QECC microcode. Our solution reduces instruction bandwidth by five orders of magnitude for key quantum workloads. We extend QuEST to mitigate the bloat of logical instruction bandwidth caused by magic state distillation and show an overall bandwidth reduction of eight orders of magnitude.

While we focus on the instruction bandwidth delivery in this paper, the concepts in our paper can be used to make quantum control hardware more efficient by delegating the repeatedly executing tasks to the hardware. In a sense, the quantum computing domain is the middle of establishing the right dynamic-static interface, something that conventional computing went through about four decades ago. Our future work is to continue to explore the trade-offs in the dynamic-static interface in the quantum domain.

APPENDIX A

Surface code (SC) is a quantum error correction code which promises high reliability. In this section, we will provide a brief overview of the surface codes to highlight the impact of QECC on control processor design.

A.1 Surface Code Design

The surface code is implemented on a two-dimensional array of physical qubits, as shown in Figure 18. The qubits are of two types: data qubits (a, b, c, d, e, f) and ancillary qubits (p, q). The ancillary qubits are used to generate error syndromes, and data qubits are used to encode the quantum information. There are two types of syndromes, X-syndrome which detect bit-flip errors and Z-syndrome which detect phase-flip errors.

The Figure 18 shows an architecture for 5x5 surface code *unit cell* with circuits for X-syndrome and Z-syndrome. Quantum circuits are set of quantum instructions applied in specific order. For example, in the X-syndrome circuit, first, an identity instruction is applied on the qubit p which does not change the state of the qubit. Then a preparation gate is used to initializes p in state $|0\rangle$. Then, four CNOT gates are applied serially such that the ancillary qubit:p is a target qubit and neighboring four data qubits: a, b, c, d are control qubits as shown in Figure 18b. To implement the surface code on large qubit lattice, instructions patterns of a unit cells are simply repeated across multiple physical qubits.

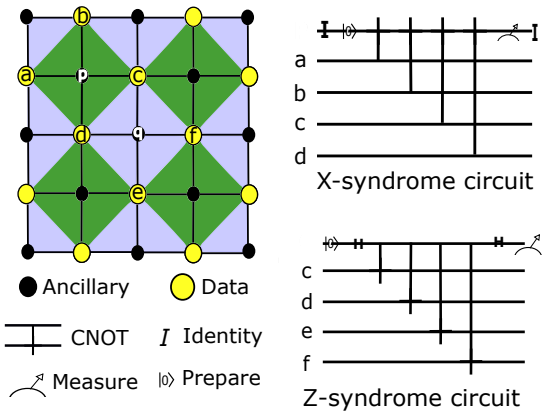

Figure 17: Surface code X-syndrome and Z-syndromes applied on an unit cell of 5x5 qubits. Unit cell is a patch of data qubits (yellow circles) and ancillary qubits (black circles) that repeats in space.

To understand how X-syndrome detects the bit-flip errors, we need to understand the function of CNOT instruction. The CNOT instruction operates on two qubits: control and target, it flips the target qubit if the control bit is $|1\rangle$, else target is unchanged. Note that the control qubit is always unchanged when CNOT is applied. In case of x-syndrome generation, we prepared the p qubit in $|0\rangle$, and let's assume all the data qubits (a, b, c, d) are $|1\rangle$. This means, when we apply $CNOT(a, p)$, $CNOT(b, p)$, $CNOT(c, p)$, and $CNOT(d, p)$ each gate would flip the ancillary qubit p and ultimately have no effect on the state of p. However, if there is one bit-flip error on any of the data qubits then p would flip, and error can be detected once we measure p. This can be extended to a case where states of the data-qubits are arbitrary. As long as a, b, c, d have identical quantum states; ancillary qubit would not flip its state. Similarly, a phase-flip can be detected by generating and measuring Z-syndrome. When an isolated single bit flip or phase flip error occurs, syndrome measurements can be used to point the errors. In Figure 18, ❶ shows scenario when the data qubit have a phase(Z) error which creates an unbalanced Z-syndromes, and Z error can be detected by measuring ancillary qubits. In the case of bit-flip error ❷, ancillary qubit in an X-syndrome would flip, and error can be detected. However, error detection becomes tricky in the case of long chains of X and Z errors, as shown in ❸ in Figure 18b. Symmetric chains of X-errors about measurement qubit causes flip in adjacent error syndromes of erroneous qubits. To resolve this error we need to look at all the neighboring syndromes. The decoding of error becomes, even more, trickier if there are more than two errors in the same syndrome, or if the error chain is long ❹.

A.2 Error Decoder

Error decoder is used to resolve complex error patterns and provide protection against a large class of error patterns. To facilitate the error decoding, syndrome measurement is recorded every time it changes its value. These records can be represented as a classical data structure which stores the changes in syndrome measurement in space and time. This data structure is used as part of error correction decoder which can identify the error positions by observing

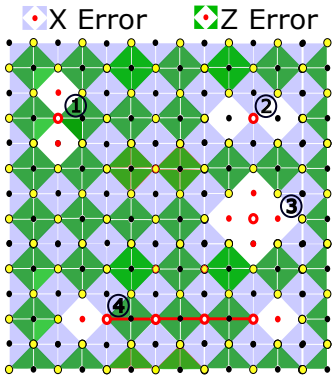

Figure 18: Surface code detects the isolated and chains of X and Z errors. It detect the chain of errors using minimum weight perfect matching decoder [14].

changes in syndrome over a window of space and time. Pairs of flipped syndromes are connected to generate a weighted graph. To find the exact locations of the errors, the minimum weight matching algorithm is run on the graph [13]. It is important to note that surface code performs decoding with classical information since the syndromes are measured every time, and the measurements are stored in a weighted graph. Once errors are resolved, they can be corrected by executing quantum instructions. As both X and Z errors are unitary, we can simply maintain the log for all the errors and correct them before measuring a qubit. This decouples the error decoding and error correction from the quantum execution (for details please refer [14]).

ACKNOWLEDGEMENTS

We thank Alan Geller, Burton Smith, Nathan Wiebe, Bob Krick, Chia-Chen Chou, Mohammad Arjomand, Gururaj Shaileshwar, and Vinson Young for technical discussions and comments. We would like to thank anonymous reviewers for the feedback and comments. This work was supported in part by NSF grant 1526798 and the Center for Future Architecture Research (C-FAR), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

REFERENCES

[1] Andris Ambainis, Andrew M Childs, Ben W Reichardt, Robert Špalek, and Shengyu Zhang. 2010. Any AND-OR formula of size N can be evaluated in time N^1/2+o(1) on a quantum computer. *SIAM J. Comput.* 39, 6 (2010), 2513–2530.

[2] Steven Balensiefer, Lucas Kregor-Stickles, and Mark Oskin. 2005. An evaluation framework and instruction set architecture for ion-trap based quantum micro-architectures. In *ACM SIGARCH Computer Architecture News*, Vol. 33. IEEE Computer Society, 186–196.

[3] Douglas M. Carmean. 2017. Quantum and Cryo and DNA, oh my! sights along the new yellow brick road. ISCA 2016 Keynote, http://dcarmean.azurewebsites.net/ISCA2016.pdf. (2017). [Online; accessed 7-June-2017].

[4] J. Chang, M. R. Vissers, A. D. Corcoles, M. Sandberg, J. Gao, David W. Abraham, Jerry M. Chow, Jay M. Gambetta, M. B. Rothwell, G. A. Keefe, Matthias Steffen, and D. P. Pappas. 2013. Improved superconducting qubit coherence using titanium nitride. Appl. Phys. Lett. 103, 012602 (2013). (2013). https://doi.org/10.1063/1.4813269 arXiv:arXiv:1303.4071

[5] E. Charbon, F. Sebastiano, A. Vladimirescu, H. Homulle, S. Visser, L. Song, and R. M. Incandela. 2016. Cryo-CMOS for quantum computing. In *2016 IEEE International Electron Devices Meeting (IEDM)*. 13.5.1–13.5.4. https://doi.org/10.1109/IEDM.2016.7838410

[6] Eric Chi, Stephen A. Lyon, and Margaret Martonosi. 2007. Tailoring Quantum Architectures to Implementation Style: A Quantum Computer for Mobile and Persistent Qubits. *SIGARCH Comput. Archit. News* 35, 2 (June 2007), 198–209. https://doi.org/10.1145/1273440.1250687

[7] Andrew M Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A Spielman. 2003. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. ACM, 59–68.

[8] Dean Copsey, Mark Oskin, Francois Impens, Tzvetan Metodiev, Andrew Cross, Frederic T Chong, Isaac L Chuang, and John Kubiatowicz. 2003. Toward a scalable, silicon-based quantum computing architecture. *IEEE Journal of selected topics in quantum electronics* 9, 6 (2003), 1552–1569.

[9] International Business Machines Corporation. 2017. Universal Quantum Computer Development at IBM:. http://research.ibm.com/ibm-q/research/. (2017). [Online; accessed 3-April-2017].

[10] David P DiVincenzo et al. 2000. The physical implementation of quantum computation. *arXiv preprint quant-ph/0002077* (2000).

[11] Mikhail Dorojevets and Zuoting Chen. 2015. Fast pipelined storage for high-performance energy-efficient computing with superconductor technology. In *Emerging Technologies for a Smarter World (CEWIT), 2015 12th International Conference & Expo on*. IEEE, 1–6.

[12] Mikhail Dorojevets, Zuoting Chen, Christopher L Ayala, and Artur K Kasperek. 2015. Towards 32-bit Energy-Efficient Superconductor RQL Processors: The Cell-Level Design and Analysis of Key Processing and On-Chip Storage Units. *IEEE Transactions on Applied Superconductivity* 25, 3 (2015), 1–8.

[13] Austin G Fowler. 2015. Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $O(1)$ parallel time. *Quantum Information & Computation* 15, 1-2 (2015), 145–158.

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

[15] Jeongwan Haah, Matthew B. Hastings, D. Poulin, and D. Wecker. 2017. Magic State Distillation with Low Space Overhead and Optimal Asymptotic Input Count. (2017). arXiv:arXiv:1703.07847

[16] Thomas Häner, Damian S Steiger, Krysta Svore, and Matthias Troyer. 2016. A software methodology for compiling quantum programs. *arXiv preprint arXiv:1604.01401* (2016).

[17] Matthew B. Hastings, Dave Wecker, Bela Bauer, and Matthias Troyer. 2015. Improving Quantum Algorithms for Quantum Chemistry. *Quantum Info. Comput.* 15, 1-2 (Jan. 2015), 1–21. http://dl.acm.org/citation.cfm?id=2685188.2685189

[18] Jeff Heckey, Shruti Patil, Ali JavadiAbhari, Adam Holmes, Daniel Kudrow, Kenneth R Brown, Diana Franklin, Frederic T Chong, and Margaret Martonosi. 2015. Compiler management of communication and parallelism for quantum computation. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 445–456.

[19] W. H. Henkels, N. C. C. Lu, W. Hwang, T. V. Rajeevakumar, R. L. Franch, K. A. Jenkins, T. J. Bucelot, D. F. Heidel, and M. J. Immediato. 1989. A low temperature 12 ns DRAM. In *VLSI Technology, Systems and Applications, 1989. Proceedings of Technical Papers. 1989 International Symposium on*.

[20] Quentin P Herr, Anna Y Herr, Oliver T Oberg, and Alexander G Ioannidis. 2011. Ultra-low-power superconductor logic. *Journal of applied physics* (2011).

[21] D. S. Holmes, A. L. Ripple, and M. A. Manheimer. 2013. Energy-Efficient Superconducting Computing x2014;Power Budgets and Requirements. *IEEE Transactions on Applied Superconductivity* 23, 3 (June 2013), 1701610–1701610. https://doi.org/10.1109/TASC.2013.2244634

[22] Harald Homulle, Stefan Visser, Bishnu Patra, Giorgio Ferrari, Enrico Prati, Carmen G. Almudéver, Koen Bertels, Fabio Sebastiano, and Edoardo Charbon. 2016. CryoCMOS Hardware Technology a Classical Infrastructure for a Scalable Quantum Computer. In *Proceedings of the ACM International Conference on Computing Frontiers (CF '16)*. ACM, New York, NY, USA, 282–287. https://doi.org/10.1145/2903150.2906828

[23] Harald Homulle, Stefan Visser, Bishnu Patra, Giorgio Ferrari, Enrico Prati, Fabio Sebastiano, and Edoardo Charbon. 2017. A reconfigurable cryogenic platform for the classical control of quantum processors. *Review of Scientific Instruments* 88, 4 (2017), 045103.

[24] J. M. Hornibrook, J. I. Colless, I. D. Conway Lamb, S. J. Pauka, H. Lu, A. C. Gossard, J. D. Watson, G. C. Gardner, S. Fallahi, M. J. Manfra, and D. J. Reilly. 2015. Cryogenic Control Architecture for Large-Scale Quantum Computing. *Phys. Rev. Applied* 3 (Feb 2015), 024010. Issue 2.

[25] Nemanja Isailovic, Mark Whitney, Yatish Patel, and John Kubiatowicz. 2008. Running a quantum circuit at the speed of data. In *ACM SIGARCH Computer Architecture News*, Vol. 36. IEEE Computer Society, 177–188.

[26] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T Chong, and Margaret Martonosi. 2015. ScaffCC: Scalable compilation and analysis of quantum programs. *Parallel Comput.* 45 (2015), 2–17.

[27] Cody Jones, Peter Brooks, and Jim Harrington. 2016. Gauge color codes in two dimensions. *Phys. Rev. A* 93 (May 2016), 052332. Issue 5. https://doi.org/10.1103/PhysRevA.93.052332

[28] Daniel Kudrow, Kenneth Bier, Zhaoxia Deng, Diana Franklin, and Frederic T Chong. 2013. Dynamic Machine-Code Generation for Quantum Rotations. *GSWC 2013* (2013), 23.

[29] Daniel Kudrow, Kenneth Bier, Zhaoxia Deng, Diana Franklin, Yu Tomita, Kenneth R Brown, and Frederic T Chong. 2013. Quantum rotations: a case study in static and dynamic machine-code generation for quantum computers. In *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 166–176.

[30] Bjoern Lekitsch, Sebastian Weidt, Austin G Fowler, Klaus Mølmer, Simon J Devitt, Christof Wunderlich, and Winfried K Hensinger. 2017. Blueprint for a microwave trapped ion quantum computer. *Science Advances* 3, 2 (2017), e1601540.

[31] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. 2013. Quantum algorithms for supervised and unsupervised machine learning. (2013). arXiv:arXiv:1307.0411

[32] Frédéric Magniez, Miklos Santha, and Mario Szegedy. 2007. Quantum algorithms for the triangle problem. *SIAM J. Comput.* 37, 2 (2007), 413–424.

[33] J. Medford, M. Stoutimore, Q. Herr, O. Naaman, H. Hearne, J. Strand, A. Przybysz, A. Pesetski, and J. Przybysz. 2015. Demonstrated control of a Transmon using a Reciprocal Quantum Logic digital circuit - Part 2. In *APS Meeting Abstracts*.

[34] Rodney Van Meter and Mark Oskin. 2006. Architectural Implications of Quantum Computing Technologies. *J. Emerg. Technol. Comput. Syst.* 2, 1 (Jan. 2006), 31–63. https://doi.org/10.1145/1126257.1126259

[35] Tzvetan S Metodi, Arvin I Faruque, and Frederic T Chong. 2011. Quantum Computing for Computer Architects. *Synthesis Lectures on Computer Architecture* 6, 1 (2011), 1–203.

[36] O. Naaman, M. O. Abutaleb, C. Kirby, and M. Rennie. 2016. On-chip Josephson junction microwave switch. *Applied Physics Letters* 108, 11, Article 112601 (2016). https://doi.org/10.1063/1.4943602

[37] Mark Oskin, Frederic T Chong, and Isaac L Chuang. 2002. A practical architecture for reliable quantum computers. *Computer* 35, 1 (2002), 79–87.

[38] PJJ OâĂŹMalley, J Kelly, R Barends, B Campbell, Y Chen, Z Chen, B Chiaro, A Dunsworth, AG Fowler, I-C Hoi, et al. 2015. Qubit metrology of ultralow phase noise using randomized benchmarking. *Physical Review Applied* 3, 4 (2015), 044009.

[39] Shruti Patil, Ali JavadiAbhari, Chen-Fu Chiang, Jeff Heckey, Margaret Martonosi, and Frederic T Chong. 2014. Characterizing the performance effect of trials and rotations in applications that use Quantum Phase Estimation. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on*. IEEE, 181–190.

[40] Ethan Schuchman and TN Vijaykumar. 2006. A program transformation and architecture support for quantum uncomputation. *ACM SIGOPS Operating Systems Review* 40, 5 (2006), 252–263.

[41] Peter W. Shor. 1995. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A* 52 (Oct 1995), R2493–R2496. Issue 4. https://doi.org/10.1103/PhysRevA.52.R2493

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

[43] Sung-min Sim, Yeonsu Lee, Hye-lim Kang, Kwon-Yong Shin, Sang-Ho Lee, and Jung-Mu Kim. 2016. Transmission line printed using silver nanoparticle ink on FR-4 and polyimide substrates. *Micro and Nano Systems Letters* 4, 1 (2016), 7.

[44] Tom Simonite. 2015. Google Quantum Dream Machine. https://www.technologyreview.com/s/544421/googles-quantum-dream-machine/?state=email-verified. (2015). [Online; accessed 2-November-2016].

[45] Marshall Stoneham. 2009. Trend: Is a room-temperature, solid-state quantum computer mere fantasy? *Physics* 2 (2009), 34.

[46] M. Stoutimore, J. Medford, Q. Herr, O. Naaman, H. Hearne, J. Strand, A. Przybysz, A. Pesetski, and J. Przybysz. 2015. Demonstrated control of a Transmon using a Reciprocal Quantum Logic digital circuit - Part 1. In *APS Meeting Abstracts*.

[47] M. Suchara, J. Kubiatowicz, A. Faruque, F. T. Chong, C. Y. Lai, and G. Paz. 2013. QuRE: The Quantum Resource Estimator toolbox. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*. 419–426. https://doi.org/10.1109/ICCD.2013.6657074

[48] Darshan D Thaker, Tzvetan S Metodi, Andrew W Cross, Isaac L Chuang, and Frederic T Chong. 2006. Quantum memory hierarchies: Efficient designs to match available parallelism in quantum computing. In *ACM SIGARCH Computer Architecture News*, Vol. 34. IEEE Computer Society, 378–390.

[49] Sergey K Tolpygo, Vladimir Bolkhovsky, TJ Weir, Alex Wynn, DE Oates, LM Johnson, and MA Gouker. 2016. Advanced Fabrication Processes for Superconducting Very Large-Scale Integrated Circuits. *IEEE Transactions on Applied Superconductivity* 26, 3 (2016), 1–10.

[50] Yu Tomita and Krysta M. Svore. 2014. Low-distance surface codes under realistic quantum noise. *Phys. Rev. A* 90 (Dec 2014), 062320. Issue 6. https://doi.org/10.1103/PhysRevA.90.062320

[51] David B Tuckerman, Michael C Hamilton, David J Reilly, Rujun Bai, George A Hernandez, John M Hornibrook, John A Sellers, and Charles D Ellis. 2016. Flexible superconducting Nb transmission lines on thin film polyimide for quantum computing applications. *Superconductor Science and Technology* (2016).

[52] Rodney Van Meter and Clare Horsman. 2013. A Blueprint for Building a Quantum Computer. *Commun. ACM* 56, 10 (Oct. 2013), 84–93.

[53] W. K. Wootters and W. H. Zurek. 1982. A single quantum cannot be cloned. *Nature* 299, 5886 (28 Oct 1982), 802–803. https://doi.org/10.1038/299802a0

[54] Theodore J Yoder and Isaac H Kim. 2016. The surface code with a twist. *arXiv preprint arXiv:1612.04795* (2016).