An Evaluation Framework and Instruction Set Architecture for Ion-Trap based Quantum Micro-architectures.

Steven Balensiefer, Lucas Kregor-Stickles, and Mark Oskin
Department of Computer Science and Engineering
University of Washington
{alaska, lucasks, oskin}@cs.washington.edu

Abstract:

The theoretical study of quantum computation has yielded efficient algorithms for some traditionally hard problems. Correspondingly, experimental work on the underlying physical implementation technology has progressed steadily. However, almost no work has yet been done which explores the architecture design space of large scale quantum computing systems. In this paper, we present a set of tools that enable the quantitative evaluation of architectures for quantum computers.

The infrastructure we created comprises a complete compilation and simulation system for computers containing thousands of quantum bits. We begin by compiling complete algorithms into a quantum instruction set. This ISA enables the simple manipulation of quantum state. Another tool we developed automatically transforms quantum software into an equivalent, fault-tolerant version required to operate on real quantum devices. Next, our infrastructure transforms the ISA into a set of low-level micro architecture specific control operations. In the future, these operations can be used to directly control a quantum computer. For now, our simulation framework quickly uses them to determine the reliability of the application for the target micro architecture.

Finally, we propose a simple, regular architecture for ion-trap based quantum computers. Using our software infrastructure, we evaluate the design trade offs of this micro architecture.

1 Introduction

Experimental research into quantum computing technologies has been progressing at a steadily. Demonstrations of bulk-spin NMR computers [1], ion-trap based designs [2, 3, 4], and optical cavity wells [5, 6] for quantum computation have been performed. The next step in this area is to scale up from experimental quantum computers consisting of a handful of quantum bits to large scale quantum computing systems. Clearly many technological hurdles still exist, and one of the most basic is the architectural design of these systems.

Why worry about the architecture of a quantum computer now? The most promising technologies are at least five years from demonstrations of a dozen qubits or more, and large scale systems are not even seriously on the drawing board. Architects, however, can make significant contributions by: (1) identifying the serious practical difficulties that will arise from the physical structure of these devices and (2) finding solutions to these and other challenges with the technology.

Identifying the challenges these systems face allows device physicists and quantum theorists to start exploring potential solutions. By understanding the challenges facing the practical implementation of these technologies, architects can find solutions through the proper organization of the structure of these devices. Collectively, what this means is computer architects have the potential to hasten the development of a large scale quantum computer sooner rather than later by identifying and solving scalability problems early.

Where to begin with quantum architecture research? Similar to classical architecture, one begins with the applications. Surprisingly, even though it will be some time before a quantum computer is built the application that computer will execute is already well known: *error correction*. Quantum technologies will operate with error rates far higher than classical machines. Experimental error rates of $1E{-}3$ per bit operation have been measured in NMR systems [1]. Technological advances are expected to lower these rates dramatically, but reaching $1E{-}10$ - $1E{-}12$ is considered highly aggressive. There is only one way to manage these errors in a quantum computer: utilizing software error correction on a well-designed quantum computer architecture.

Our research efforts have been devoted to developing these architectures. To conduct this research in a quantitative fashion, we developed an infrastructure consisting of compilation and modeling tools. This paper will spend significant time describing these software artifacts (Sections 3- 7) because the methodology for applying architectural principles to quantum computers is one of the primary contributions of this work. All existing work on quantum architectures has produced either hand-designed circuits without considerations for scalability [7] or analytical models for performance and reliability that are unable to scale to systems large enough to solve real-world problems [8].

We rely on appropriate technological abstractions and careful design of the ISA, scheduler, and simulator to construct an infrastructure that scales (linearly) to thousands of qubits and billions of time steps. Briefly, our tool chain is the following:

0-7695-2270-X/05/$20.00 (C) 2005 IEEE

- A compiler from an existing high-level language which enables the manipulation of quantum bits to an instruction set architecture we developed for quantum computers.
- An error correction compiler that automatically transforms a quantum ISA assembly text into equivalent fault tolerant versions.
- A device scheduler that maps an assembly source into a set of device specific primitive operations for controlling a quantum micro-architecture.
- A simulator that models the reliability of the quantum bits in a quantum computer; performance and reliability metrics for an application running on a targeted microarchitecture can be obtained by using this simulator.

Our tools enable researchers to explore architectural tradeoffs directly. Instead of using high-level models or mathematical equations to calculate execution time and reliability our infrastructure provides the proper compilation, scheduling, and simulation tools to compute these results precisely.

Using these tools, in Section 8 we evaluate a few quantum micro-architectures as they perform error correction steps. We find that the realistic constraints exposed by execution on a microarchitecture significantly decrease the acceptable error rates. Idealized theoretical models set the critical threshold – above which sustainable quantum computation is not possible [9] – at approximately $1E{-}4$, but a threshold which accounts for the constraints of the proposed microarchitecture is closer to $1E{-}9$. Our results indicate that more than 4/5 of this difference can be accounted for by resource contention and the impact of ion movement and turning in a real system. Since architects excel at the exploitation of locality and the minimization of resource contention, this suggests that through intelligent design, architects have the potential to have a major impact on the accuracy of quantum computation thus allowing us to achieve a scalable quantum computer sooner rather than later.

The remainder of this paper is structured in a logical progression. In Section 2 we describe the abstractions we use to make quantum architectures accessible. Section 3 presents an overview of our software infrastructures. The ISA we developed is described in Section 4. Sections 6, 5 and 7 elaborate on the design of our device scheduler, error correction compiler, and simulator. In Section 8 we present the result from our exploration of a simple tile-based quantum microarchitecture. In Section 9 we describe where to go next with this work and in Section 10 conclude.

2 Technology abstraction

The science of architecture is the optimization of the hardware / software interface. The nuts and bolts of it is examining applications, working with the realistic constraints of the technology, and developing software infrastructures and hardware designs. Research into architectures for quantum computers is no different. To design architectures, a reasonable abstraction for the underlying technology and understanding of the software applications is required. In this section, we describe a basic set of abstractions for ion trap based quantum computing technology. We will discuss the application characteristics further in Section 4.

We focus our attention on ion trap based designs because they appear to be the most promising in terms of a near term ability to deliver a system with 10's to 100's of qubits. The cost of these systems will not be insignificant with estimates in the hundreds of millions of dollars to develop a single prototype. Proper engineering of their architectural design ahead of time will be required to maximize their scientific and national infrastructure value.

For architectural design, we focus on three circuit components: ions, traps and wires, as depicted in Figure 1. Ions are the entities that realize qubits. The excitation state of the outer electron on a ^{9}BE+ ion is the actual quantum property used to realize a qubit [3, 4]. A trap is a device that uses classical support circuitry and lasers to perform quantum operations on ions. This gives it a multi-purpose, ALU-like functionality. Quantum operations can only be performed on ions that are located in traps. Inside of the trap, any arbitrary single qubit operation and a limited number of two qubit operations including CNOT and controlled rotation can be performed. For the two qubit operations, both ions must be located in the same trap. Wires are just two sided structures within the design in which ions can move. Wires can contain corners but care must be taken when moving ions in anything other than a straight line. Ions must move adiabatically (read: slowly) around corners or an unrepairable amount of noise will be introduced.

While the precise timing of all operations is obviously not known yet – it is technology specific and will change as the systems evolve, the relative timing between them, observed from [3, 4], is roughly: moving 1 unit within a wire is $1/10^{th}$ a time step; performing a single qubit operation, 1 time step; performing a two qubit operation, 10 time steps; turning a corner including getting into and out of a trap, 100 time steps. Architects should think of the single-qubit operations as the "clock cycle" of the machine. The classical analogy is that these operations are simple and fast, like an addition. Measurement and two-qubit operations are slow, just like complex classical functions such as divide. Later in Section 5 we will present statistics for the relative instruction mix between single/two qubit operations and measurement.

The basis unit for these time steps is $\approx 1us$. For single- and two-qubit operations, this will not change, as it is a fundamental property of the ions [3, 4] used to realize qubits. For movement and turning, it is a function of the technology, and as this develops, they may become faster. Moving,

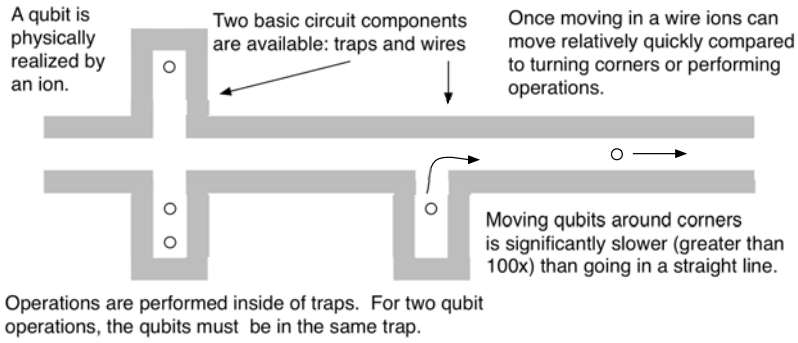


Figure 1: Technological abstraction of ion trap based quantum computers.

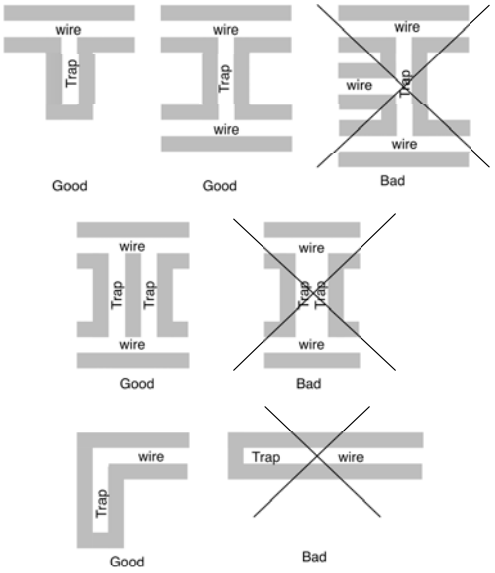


Figure 2: Basic design rules for ion trap systems.

and in particularly turning, induces noise (from heat) on the qubit and must be performed slowly so that the state does not decohere. Current experimental work aims for turning to be 50-300 times slower than single-qubit operations [3]. Since controlling noise is so important for quantum architectures we do not use a highly aggressive turning time for our simulations. We do, however, explore the impact of this parameter on performance in Section 8.

Our review of current ion-trap based designs [3, 7] suggests a few simple design rules that must be observed by architects. These rules are the quantum analog of VLSI design rules:

- Ion traps may only abut one or two wires
- Ion traps may not share any sides
- Ion traps may not abut the end of a wire

These rules are depicted in Figure 2. They serve as an additional level of abstraction by removing the need to consider the exact sizing and space tolerances for layouts. Later, in Section 8, we will explore a simple regular architecture that observes these design constraints.

3 Software overview

To evaluate complex conventional systems, architects utilize a variety of software tools. Starting with a (hopefully) representative set of applications, they compile and execute them on sophisticated simulation infrastructures that model different points in the design space. To properly study large scale, quantum computers we created a corresponding infrastructure. This infrastructure is comprised of four major components: a source compiler, an error correction compiler, a device scheduler, and a simulator. In this section, we describe what these tools do and how they are used. In the next few sections we elaborate more on how they work. Figure 3 contains a pictorial overview of the flow of information through the tools.

Source compiler: To describe quantum algorithms, we utilize the existing QCL [10] work. The QCL toolkit provides an interpreter for a fairly straightforward imperative programming language that includes data types and operation primitives for quantum operations. We did not extend this work significantly except to make minor changes to perform loop unrolling and output instructions in the instruction set described in Section 4.

To allow for aggressive code optimization we require the input to applications at compile time. The resulting assembly output from the compiler contains only the operations required to perform the algorithm on the provided input data. This may seem limiting, but two related reasons motivate this design choice. First, our expectation is that the time required for a quantum computer to execute an algorithm will be significantly longer than the time required to optimize resource usage for a particular algorithm/input dataset combination.

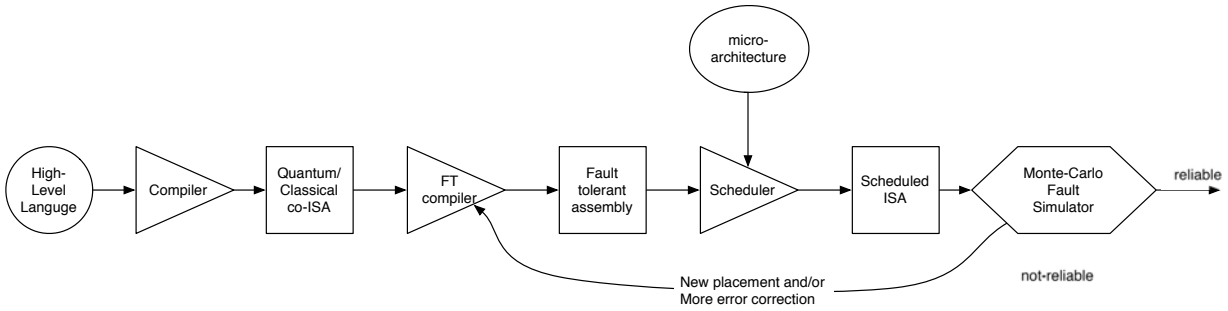


Figure 3: Quantum architecture research infrastructure. This set of tools enables architects to start with a high level language description of an algorithm and a microarchitecture and compile, add fault tolerant steps, schedule for the architecture, and simulate the speed and reliability of the algorithm.

Stated another way, there will be sufficient gains in execution time to spend significant time "up front" optimizing resource usage. The second reason is that error correction incurs a high overhead, suggesting that it should be applied as minimally as possible. More general representations require more general computation and hence more error correction, while an executable targeted to only a single input dataset can be optimized aggressively for just that dataset. Similar findings have been reported in classical computing, where dynamic optimizers aggressively tailor executables, folding in constants, etc [11, 12, 13].

Error correction compiler: The output of our source compiler is an assembly text. This assembly assumes an idealized machine – one with no errors. This is not true at all – quantum computers will have error rates between $1E{-}6$ and $1E{-}10$ per operation. To counteract this, researchers discovered and explored many different types of quantum error correction [14, 15, 16, 17]. For our purpose, we selected the 7 qubit Steane code [14] and the recursive construction process described in [9]. The error correction compiler inputs the assembly text that assumed an ideal computer and an "error correction strength" level and outputs another assembly text that is the same algorithm except with fault tolerant constructs included. This output text is considerably larger – potentially by several orders of magnitude – but is required to coax the right answer from an otherwise noisy quantum device.

Scheduler: The next step in the tool chain is to schedule the resources of the quantum computer. For classical computing devices, the schedule is implicit in the executable – the semantics of von Neumann machines are sequential. For quantum computers, sequential semantics are maintained, but the importance of exploiting parallelism increases dramatically. Ignoring parallelism in a von Neumann machine results in a longer execution time, but the computed result does not change. In a quantum computer, ignoring parallelism could result in a wrong answer. Thus our scheduler takes in an assembly text and a description of the microarchitecture of the quantum computer and creates a parallel schedule of operations that should be performed on the actual microarchitecture.

Simulator: Once the application is scheduled onto the physical resources of the machine, the next step in the tool chain is to decide whether or not the application will actually work. Too little error correction or a poor schedule will produce noise instead of the correct answer. The purpose of this step is to determine how reliable the scheduled application will be on the device. If the simulator determines the schedule will be reliable then we are done. The end results are two facts: how fast the algorithm executed on the microarchitecture and how reliable the result was. If the result is determined to be unreliable, the user has to back up two steps and add more error correction or model a different microarchitecture that might perform better.

A schedule that shows a high rate of reliability under simulation is detailed enough to control the physical computer during the execution of the algorithm and dataset. This step is beyond the scope of this work, but basically, it involves translating the schedule using a fairly straightforward mapping between operation steps and the pulses that control the actual quantum computer.

4 Instruction set architecture

The design of an instruction set architecture (ISA) encompasses many different pieces. The most fundamental is the execution model, which describes how a machine will process a group of instructions. Next are the resources available in the machine, typically memories and their interface. Finally, there are the actual instructions themselves. Figure 4 describes the ISA we designed.

The ISA we describe here is a "high-level ISA" which is not directly executable by any quantum computer. These ISAs have also been referred to as "virtual ISAs" [18] and linear intermediate representations. The purpose of this ISA

is to provide a workable representation of an application. By workable, we mean that it has relatively straightforward semantics, tools can process it largely piecemeal operation-by-operation, and it can be translated in a direct way to the actual control sequences a quantum computer requires.

Execution model: We base our ISA on the von Neumann execution model. This means that conceptually, the quantum computer can be thought to fetch, decode, and execute the primitive operations one-by-one. This design choice is motivated by an additional restriction we place on execution: Quantum programs may not contain branches. All loops must be fully unrolled, and all conditionals must be converted into predicates. (see Figure 5).

The reason we chose to restrict applications in this way is that it enables our software infrastructure to provide developers with concrete reliability results. Since there are no branches in the compiled binary, every instruction must be scheduled onto a quantum micro-architecture. Once scheduled, our simulator can provide a very precise answer to the question, "Will it work?"

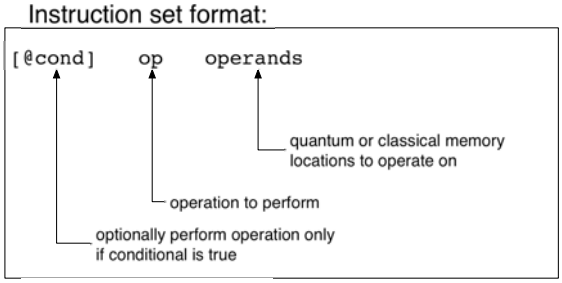
If branches were part of the ISA, then answering that question would no longer be possible. The schedule of low-level operations could vary significantly from execution-run to execution-run based upon branch outcomes, which in quantum software, depends largely on random noise the system experiences and corrects for. These variances make it far more difficult to predict reliably whether or not the schedule will actually compute correctly.

Resources: In our high-level ISA, we assume an infinite number of quantum and classical memory locations are available. Memory is split into two segments, a quantum segment and a classical segment. Quantum bits (qubits) are referred to as $qName1, qName2, ...$, while classical bits are referred to as $cName1, cName2, ...$ Since this is a high-level ISA, there is no need to restrict the name of bits to simple numerical addresses as a simple compilation pass prior to scheduling can assign device specific addresses and resolve any false dependencies caused by name reuse. We do not use a hierarchical memory (i.e. there is no distinction between memory and registers).

Operations: The instruction set we have devised operates on both classical and quantum data. The classical operations are fairly ordinary and encompass a straightforward set of opcodes (logic, arithmetic, etc). For brevity, we do not describe them in detail because they are your typical three operand RISC-like ISA: $cOutput = cInput1\ op\ cInput2$.

The quantum opcodes are summarized in Figure 4. These operations provide a fairly basic, yet complete set of operations for manipulating quantum state. There are many things to note about this instruction set. First, all quantum operations (except measurement) are, by definition, re-

Instruction set format:



Quantum Operations:

h	qN	Basic quantum primitives such as Hadamard (H), invert (X), invert phase (Z), arbitrary rotation (R), and phase gate (S)
x	qN	
z	qN	
rot	qN,real	
s	qN	
v	qN,qC,real	rotate qN about X axis, conditional on qC, by real
cnot	qN,qC	flip qN conditional on qC
swap	qN1,qN2	swap qN1 and qN2
toffoli	qN,qC1,qC2	flip qN conditional on qC1, qC2
measure	cT, qN	measure qN place result in cT

Pseudo-operations:

.exchange	qN1,qN2	move qN1,qN2 together. (No operation)
.new	qN | cN	allocate/deallocate a new quantum or classical bit under name N.
.free	qN | cN	

Figure 4: The instruction set architecture for quantum computers

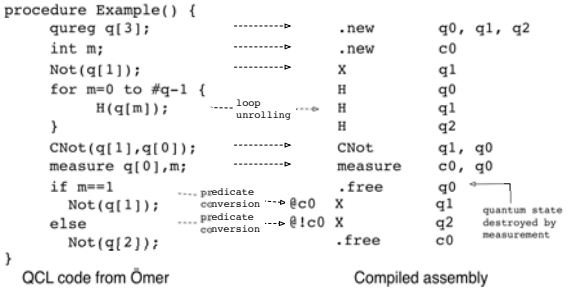


Figure 5: The compiler, based on QCL [10], transforms QCL source text into assembly.

versible. This is a constraint imposed by the quantum computing model itself. One implication of this is there is no distinction between input and output operands. Instead, operations transform all of their operands. Second, this ISA is a balance between high-level primitives, such as multi-qubit complex operations, and low-level device-specific controls. Our guiding principle has been to design an ISA with primitive enough operations that a clear $1 : N$ mapping exists between the operations and device specific control sequences, but high-level enough that the tool infrastructure could manipulate a useful block of related work.

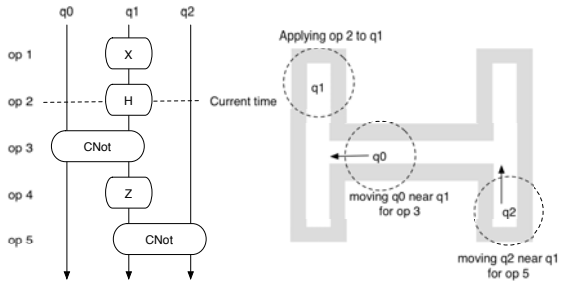


Figure 6: The goal of scheduling is to transform a program source (left) into a sequence of primitive operations that move and manipulate ions in a quantum computer (right). A main requirement is that the scheduler operate in $O(instructions)$ time because the number of instructions is in the billions for a complete fault tolerant run of Shor's algorithm.

5 Error correction compiler

After compilation to an assembly text, the next step is to transform the application into a fault-tolerant version. Fault tolerant quantum computing is done in pretty much the same way it has been done for decades in the classical domain – through redundancy. Each quantum bit is encoded into a *logical* qubit. Logical qubits utilize several physical qubits to store a coded version of the quantum state. Each operation on the original qubit is transformed into an equivalent set of operations on the qubits that make up the logical qubit. The major drawback in all of these schemes is the increase in the number of qubits required.

In our system, we utilize the 7-qubit Steane code [14] and employ the recursive error correction constructions described in [9]. We do this with an assembly source to source translator. This translator converts a compiled quantum application into an equivalent assembly source file which contains the embedded error correction operations.

The precise fault tolerant constructions are not a contribution of our work. We base them on prior work [14, 9, 19] and refer the interested reader there. However, to the best of our knowledge, our tool is the first to apply them automatically to an application, accounting for all of the required ancilla preparation work and at multiple strength levels (0,1, and 2 levels of error correction). Because of this, we have calculated some useful statistical properties about its output.

The results are shown in Figure 5. Architects should take note of the overhead in both time and space introduced by the error correction processes. The critical path for a single-qubit operation with one layer of error correction (EC1) is 31 operations long. Only 1 of those is devoted to actually performing the operation on the logical qubit. The rest are devoted to the fault tolerant correction step. More realistically, not all operations will be conducted in parallel, the overhead will be substantially higher, and stronger levels of error correction will be required. This sizable overhead is one of the reasons we can design quantum computing architectures now – Amdahl's Law [20] suggests quantum computers are going to spend all of their time error correcting!

6 Device scheduler

Once we have a source assembly file with the error correction compiled in, the next step is produce a schedule for those operations on an ion trap computer micro-architecture. Figure 6 depicts the overall goal: given a source assembly text (represented in graph form on the left), the scheduler produces the parallel sequence of low-level operations (right). In this section, we describe the scheduling process.

6.1 Input

The scheduler takes three pieces inputs: the source assembly text to be scheduled, a description of the architecture to schedule the source on, and a description of the technology parameters and constraints. The source text has been previously described (Section 4).

The architecture description is a low-level description of the ion trap layout. As a classical analogy, this is at the same level as a VLSI layout produced with tools such as Magic [21]. The description includes the precise X/Y coordinates of ion-traps, the operations each trap can perform, and their interconnection wiring.

The technology parameters provided to the scheduler contain timing information for all device-specific operations. This includes the timing of all operations (X, H, CNOT, etc) and the timing for moving ions around the computer. Specific movement parameters are included for moving ions through wires, into and out of wires, and for turning corners.

6.2 Scheduling algorithm

The ability to process billions of operations was paramount in designing the scheduler. Therefore, we chose to trade-off optimality for speed. One of the major costs in scheduling is determining the route an ion should take to travel between traps that do not abut the same wire. This problem has paral-

	% cnot	% measurement	min time	max time	min space	max space
no EC	-	-	1		1	1
EC 1	56.37%	10.31%	31	447	16	96
EC 2	56.74%	9.44%	211	217529	58	12456

Table 1: **Properties of error correction:** In this table, we present results from our error correction compiler and scheduler. The first two columns indicate the percentage of two-qubit CNOT gates and measurement operations. The next two columns min/max time indicate the time (in ops) required to perform the error correction process. Minimum time refers to doing things maximally parallel (no architectural constraints), while maximum refers to doing all operations sequentially. Minimum/max space refers to number of physical qubits required to perform the operation. Minimum space comes from doing all operations sequentially (scheduled perfectly), while maximum space comes from doing as many operations in parallel as possible, reducing execution time but increasing resource requirements.

lels in the routing of signals between logic units within FPGAs, so we adapted the PathFinder algorithm [22] to create a collection of efficient paths from source to destination in the micro-architecture. The biggest change is that we compute 5-10 paths on the first movement between a source-destination pair. This computation only occurs once, and subsequent movements simply pick the best path from those stored.

The next step is to parse the source assembly and schedule the operations. For this we employ a variant of list-scheduling [23]. First, the source text is parsed to completion and a graph representation is produced. We process this graph in reverse order, starting from the leaves, and proceeding to the root(s). By applying an earliest-possible greedy approach in reverse, we approximate latest-possible scheduling if run forward in time. From the view of the simulator, qubits allocated only when absolutely necessary, allowing reuse of "scratch" bits and attempting to minimize the time that qubits must stay coherent.

At any given time point, the scheduler maintains a list of operations that can be scheduled and attempts to allocate the physical resources of the machine for the required number of time steps. In the case of operations, this means simply holding onto the ion trap for that time. For movement, it means referring to the pre-computed path data structure and choosing the path that with no conflicts for the time required. Operations that cannot be scheduled due to resource conflicts are simply delayed and another scheduling attempt is made at the next opportune time step.

7 Simulator

The scheduler produces an exact set of command sequences for controlling a quantum computer. One can directly read the tail end of this schedule to determine the running time of the application. Of critical importance, however, is whether or not the qubits will contain correct values. Noise (decoherence) could have corrupted them so much that the schedule will not produce any meaningful result from a quantum device.

In all other quantum research projects, a precise physical level simulator of the device is used to determine the reliability. In our study, however, we are interested in computers with hundreds to thousands of qubits. Since the running time of precise simulation is exponential in the number of entangled qubits, the number of qubits that can be simulated in reasonable time with current technology (clusters of machines, days of time) is in the low 30's [24]. Clearly, this approach will not work for 100 - 100,000 qubits.

Instead, we make the observation that if you do not care about simulating the precise state of a quantum computer, Monte Carlo simulation can be used to produce an expected reliability for the device. With Monte Carlo simulation, the expected probability of a phenomenon is determined by performing an action several times and calculating what percentage of the time the phenomenon in question occurs.

In our case, the phenomenon in question is the introduction of error into an ion's quantum state. To perform our simulation, we start with a base error rate for each step of computation. This base error rate represents the probability that an error occurs in an ion at each time-step. We introduce an error in the ion when our pseudo-random number generator [25] produces a result less than this base error.

Within the simulation, errors are propagated based on the dependencies of the computation. Once an ion is in error, it stays in error and introduces error on any other ions it interacts with. The only exception to this rule is when error correction is applied. Our simulation framework models the effect of the error correction added prior to scheduling. Once an error correction is completed, the simulator examines the qubits of the logical code word. If only one qubit is in error, then the simulator assumes the error correction process fixed that single qubit error. If two or more qubits are in error, it propagates the error and assumes all qubits of that code word are now in error (the upper bound of the effect of error correction on a terminally broken code word).

Naturally, the effectiveness of Monte Carlo simulation depends on the randomness of the pseudo-random number generator used. For this purpose, we have selected a random number generator based on bit-rotation and addition which is considered particularly well suited to Monte Carlo simu-

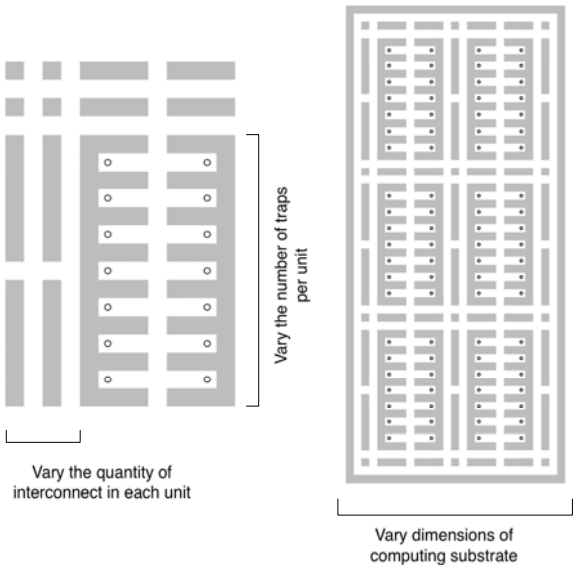


Figure 7: Basic tile structure (left) and substrate micro architecture (right). The key parameters: ion trap cells, connectivity, and substrate size are varied in this study.

lation [25].

The simulator can be used to quickly ($O(n)$ in the number of scheduled operations) determine whether or not a schedule and micro-architecture will operate accurately. Since the problems targeted by quantum computers are in the complexity class NP (a super-set of NP-complete), despite the fact that solutions to these problems are hard to generate, verification in possible in polynomial time. This means that accuracy need only be around 90% (since incorrect answers can be quickly detected and the process re-run if necessary). The implication is that the reliability of the system can be determined with a mere 100 trials on our simulator.

In addition to a quick test of the reliability of a quantum program and micro-architecture pair, the simulator can also execute an arbitrary number of trials to achieve a fine-grained understanding of the rate of error. In the next section of this paper, we use this technique to explore variations on a canonical quantum micro-architecture and measure their runtime performance and critical thresholds [9].

8 Micro-architecture exploration

In this section, we use our infrastructure to explore basic micro-architectural trade-offs. We begin by first validating the simulation model. Next, we use the tools to explore trap width versus wiring density in a simple quantum micro-architecture. Finally, we conclude by exploring the differences between quantum computing theory and practice, which highlights both the challenges for future technology development, and the importance of architecture to this discipline.

8.1 Validation

Since these tools are the first of their kind and our simulation methodology is a novel approach to modeling reliability in quantum systems, some form of validation is desired. To do this, we produced a single fault-tolerant error correction sequence. This was scheduled onto an architecture and processed by our simulator. The parameters the simulator used to model error were changed such that moving ions around the micro-architecture occurred in zero time, ions that were not being operated on had zero chance of decohering, and CNOT instructions required the same amount of time as single-qubit gates. These parameters match the theoretical model of quantum computing that is used in the literature. Doing this, we found the critical threshold – the maximum error per operation for sustainable fault tolerant computation, to be $4E - 4$. This is exactly in line with what one would expect from the theoretical estimate previously calculated [9, 19].

8.2 Exploration

A basic design of a quantum micro-architecture is depicted in Figure 7. The concept is to use a substrate of identical tiles. This design has two basic micro-architectural knobs to vary: the number of ion traps in a tile and the amount of wiring between tiles.

To explore the effects of these two parameters on execution time, we mapped the error-correction (level 1) process onto varying substrates using our scheduler. We chose layouts that provided 150 traps total and organized the tiles to be as square as possible.

The scheduler is non-deterministic (being based on a synthesis of PathFinder and list scheduler), so results vary slightly between runs. Therefore we execute each test 8 times and average the results. The overall results are shown in Figure 8.

The results show three interesting trends. First, except for the smallest design, small numbers of traps per tile are favored. Too few traps and scheduling becomes more difficult. Ions must move into and out of regions too often, increasing execution time. With too many traps, the conflicts over the single wire into the tile begins to counter the increased potential for intra-tile movement. With larger trap numbers, the ions must also move further, leading to longer execution times.

Second, beyond 2 traps / tile, a single surrounding wire (which is 2 wires between ion trap complexes; Figure 7) is less efficient than having more interconnect. However, moving from 2-3 surrounding wires provides no real savings. Looking carefully at each trap complex configuration, there is a corresponding ideal interconnect width: 2 traps / 1 wire, 3 traps / 2 wires, 5 traps / 3 wires. This pairing arises from the schedulers ability to exploit trap resources

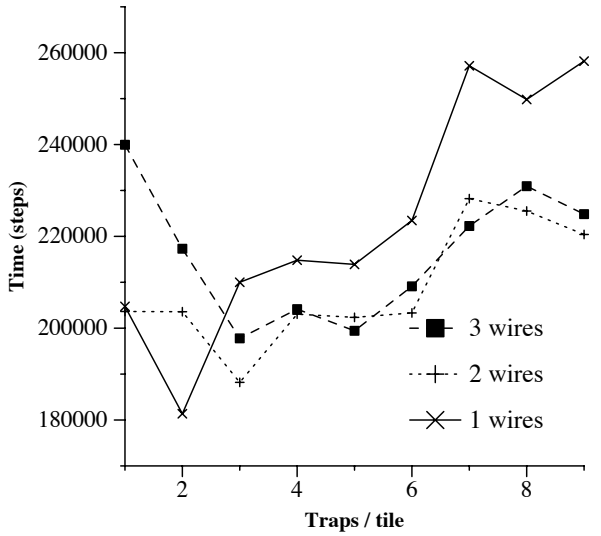


Figure 8: Performance of error-correction on various trap configurations, from designs built from tiles that are 1 trap high and 1 wire in between, to 9 traps and 3 wires.

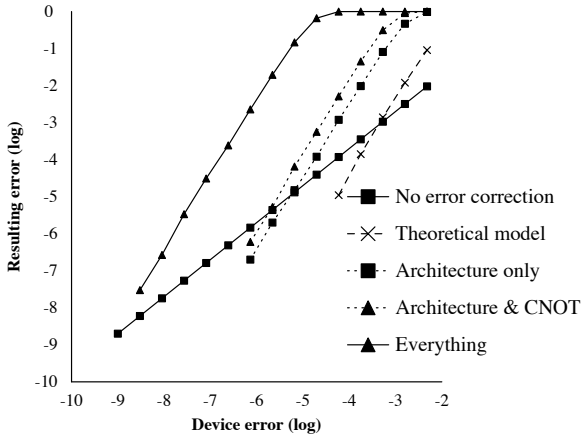


Figure 9: Observed error rates for different technology and architectural assumptions.

and wire-resources, and it makes intuitive sense that the order of these (more traps - more wires) is aligned up until 5 traps. Beyond 5, the resources are not exploited well by the scheduler and simply increase delay.

The final observation is that 2 traps / tile and a surrounding wire per tile performs best on average for this single application. For our scheduler algorithm and the error correction process, this design point minimizes overall length of travel for ions and balances trap complex size against interconnect size.

8.3 Dealing with architectural reality in quantum computers

We conclude our study by examining the critical threshold – the error rate above which error correction processes will not work. In the past [9], theorists have estimated this threshold using an overly idealized model of computation that did not account for the actual microarchitecture of the machine. Using our tools, we can account for this.

Figure 9 plots the reliability of fault-tolerant operations as various technology and architectural features are progressively accounted for. The x-axis of this graph is the rate of error for a single-qubit gate. The y-axis of this graph depicts the rate of error for the qubit measured by our simulator. The straight-line depicts the rate of error for a non-encoded non-fault tolerant single qubit operation. The x-axis points at which the other curves cross this line are their critical-thresholds.

The first line (farthest to the right) is the theoretical quantum computing model. In this model, there is no accounting for architecture or technology implications – such as movement, turns, the difference between single- and two- qubit gates, and the reality that a quantum state naturally decoheres with time, even if no operation is performed on it. This line crosses the non-fault tolerant line where prior literature [9, 19] estimates it should.

The next line over, *architecture only* alters the model to begin to consider the implications of having to perform error correction in a real micro-architecture. For this calculation, the impact of decoherence from having to wait for resources to become available is introduced.

Next comes the *architecture and cnot* line. This line depicts the effects of the micro-architecture and accounts for the fact that CNOT gates require an order of magnitude more time to operate than single qubit gates.

The final result, *everything accounted for*, is one of the main results of our work. In this trial, we introduce the full impact of movement and turns. We found that when operating on an actual micro-architecture and accounting for all of the implications of scheduling, resource conflicts, the cost of moves and turns and single versus two-qubit gates, the true threshold lies at $\approx 1E-9$. This is lower than the theoretical calculation by *5* orders of magnitude.

An important observation from this data is that of these 5 orders of magnitude in difference between the theoretical model and the actual implementation, *3* of these are the result of movement and turning, ≈ 1.5 are the result of basic resource contention and only $\approx 1/2$ is the result of the increased cost of binary operations such as CNOT.

The implication of this is that improving the accuracy of individual quantum operations will only have a minimal impact on the overall accuracy of quantum computation. Instead, our work indicates that physicists should focus on reducing the error rate and improving the execution time for turns, while architects can make a major contribution by de-

signing micro-architectures and schedulers that capitalize on locality to decrease the need for movement and allow for the efficient utilization and placement of resources to decrease contention. In this way, architects can raise the *practical* threshold. Otherwise, it really will be later rather than sooner before quantum computing is a reality.

9 Future work

There is much work left to do on quantum architectures. Right now, and for the foreseeable future, the goal of this work should be to reduce the critical threshold. What we have presented in this paper is a set of tools and architectural analyses that show the real threshold is $\approx 1E-9$. At least two and perhaps as much as three orders of magnitude of this threshold, however, are due to the micro-architecture and tool chain infrastructure. We will elaborate below on ways to reduce this threshold:

Better error correction processes: Our current infrastructure utilizes the error correction steps described in [19]. More complex, but parallel steps are known [26]. Changing the front end of the tool chain to utilize these alternative constructions could reduce by about 1/3 the minimum-time component in Table 5. This is at the expense of more complex ancilla.

Dynamically adding teleportation-channels: In [27] the authors describe an alternative way to move quantum state around a large micro-architecture. Exploiting these teleportation channels instead of direct movement where appropriate could further parallelize the operations involved in moving quantum state about.

Better micro-architectures: For this paper we did not extensively study micro-architecture designs. Our goal was more on the front end in creating all of the tools required to really study micro-architectures. Thus, the very next step seems to be to design architectures that are better able to exploit parallelism within the error correction processes.

Smarter scheduling: Our current scheduler is essentially a greedy algorithm with a bounded window. Perfect scheduling is NP-hard. There is a middle ground. Right now the scheduler is micro-architecture agnostic. It can schedule any set of quantum algorithms onto any micro-architecture. Making the scheduler more micro-architecture and error-code aware seems a rich area for performance gains. For example, qubits are often operated on in repetitive ways. Having efficient (perhaps hand-done) schedules for these common-case operations that the scheduler could draw upon to create a larger application schedule seems a viable approach.

Hierarchical simulation: Currently, our simulator is pessimistic. It is akin to an automated "counting" simulator (used to count point of failure). If the actual device had the technology characteristics specified it would be more reliable when executing the application. How much more reliable is not yet known, but it is speculated that it is perhaps as much as an order of magnitude. The simulator can be made more precise by integrating a precise device-level physics simulator and grouping operations into large units. These units can be modeled precisely using the device simulator and then their reliability parameters integrated using the counting approach of our existing framework.

10 Conclusion

In this paper, we described our work in designing an instruction set architecture, compiler, device scheduler and simulator for ion trap based quantum computers. Many design choices in each of these components were made to make them scale to real application sizes. Among them: the tools compile-in the input dataset and fully unroll all loops so that the scheduler and simulator can provide concrete results; the error correction compiler automatically transforms arbitrary input programs into fault tolerant versions; the scheduler combines techniques from FPGA/CAD synthesis and traditional processor compilers; finally, the simulator efficiently models errors instead of quantum state in order to quickly provide reliability information.

Using these tools, architects can design and quantitatively evaluate large scale architectures. In the past, quantum researchers have had to make careful analytical models for system reliability and performance. Now, they can evaluate these systems directly by compiling applications for them, scheduling them for performance, and simulating them for reliability. We did this for a few tile based designs and found that a balanced design of 2 traps to 1 interconnect wire laid out in a substrate performed best. We also found that the critical threshold is in fact five orders of magnitude lower than previously found by theoretical models alone. In addition, we determined that much of the difference between the theoretical, and practical breaking point can be attributed to problems that computer architects are particularly well suited to solve.

Acknowledgments: This work is supported in part by the DARPA QuIST Program (AFRL-F30602-01-2-0521), NSF Nanoscale Program (CCF-0210373) and NSF CAREER grants (CCF-0133188). Additional support is provided by the A. P. Sloan foundation. We would like to thank our anonymous reviewers for their constructive feedback.

References

[1] I. L. Chuang, N. Gershenfeld, M. G. Kubinec, and D. W. Leung, "Bulk quantum computation with nuclear-magnetic-resonance: theory and experiment," *Proc. R. Soc. London A*, vol. 454, no. 1969, pp. 447–467, 1998.

[2] C. Monroe, D. M. Meekhof, B. E. King, W. M. Itano, and D. J. Wineland, "Demonstration of a fundamental quantum logic gate," *Phys. Rev. Lett.*, vol. 75, p. 4714, 1995.

[3] M. A. Rowe, A. Ben-Kish, B. DeMarco, D. Leibfried, V. Meyer, J. Beall, J. Britton, J. Hughes, W. M. Itano, B. Jelenkovic, C. Langer, T. Rosenband, and D. J. Wineland, "Transport of quantum states and separation of ions in a dual rf ion trap," *Quantum Information and Computation*, vol. 2, pp. 251–271, 2002.

[4] D. J. Wineland, M. Barrett, J. Britton, J. Chiaverini, B. L. DeMarco, W. M. Itano, B. M. Jelenkovic, C. Langer, D. Leibfried, V. Meyer, T. Rosenband, and T. Schaetz, "Quantum information processing with trapped ions," *Phil. Trans. Royal Soc. London A*, vol. 361, pp. 1349–1361, 2003.

[5] G. T. Foster, L. A. Orozco1, H. M. Castro-Beltran, and H. J. Carmichael, "Quantum state reduction and conditional time evolution of wave-particle correlations in cavity qed," *Phys. Rev. Lett.*, vol. 85, pp. 3149–3152, Oct 2000.

[6] P. Domokos, J. M. Raimond, M. Brune, and S. Haroche, "Simple cavity-qed two-bit universal quantum logic gate: The principle and expected performances," *Phys. Rev. A*, vol. 52, no. 5, pp. 3554–3559, 1995.

[7] D. Kielpinsky, C. Monroe, and D. Wineland, "Architecture for a large-scale ion trap quantum computer," *Nature*, vol. 417, p. 709, 2002.

[8] T. Metodiev, A. Cross, D. Thaker, K. Brown, D. Copsey, F. T. Chong, and I.L.Chuang, "Preliminary results on simulating a scalable fault tolerant ion-trap system for quantum computation," in *3rd Workshop on Non-Silicon Computing*, June 2004.

[9] D. Aharonov, *Noisy Quantum Computation*. PhD thesis, The Hebrew Univesity, Jerusalem, 1999.

[10] B. Ömer, "Quantum programming in qcl," Master's thesis, Technical University of Vienna, 2000.

[11] M. U. Mock, C. Chambers, and S. J. Eggers, "Calpa: A tool for automating selective dynamic compilation.," in *In Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-33)*, pp. 291–302, Dec 2000.

[12] E. Feigin, *A Case for Automatic Run-Time Code Optimization*. PhD thesis, Harvard College, Div. of Eng. and Applied Sciences, 1999.

[13] J. Auslander, M. Philipose, C. Chambers, S. J. Eggers, and B. N. Bershad, "Fast, effective dynamic compilation," in *SIGPLAN Conference on Programming Language Design and Implementation*, pp. 149–159, 1996.

[14] A. Steane, "Error correcting codes in quantum theory," *Phys. Rev. Lett.*, vol. 77, 1996.

[15] P. Shor, "Scheme for reducing decoherence in a quantum computer memory," *Phys. Rev. A*, vol. 52, no. 2493, 1995.

[16] C. H. Bennett, D. P. Vincenzo, J. A. Smolin, and W. K. Wootters, "Mixed state entanglement and quantum error correction," *Phys. Rev. A*, vol. 54, no. 5, pp. 3824–3851, 1996.

[17] R. Laflamme, C. Miquel, J.-P. Paz, and W. H. Zurek, "Perfect quantum error correction code," *Phys. Rev. Lett.*, vol. 77, p. 198, 1996. arXive e-print quant-ph/9602019.

[18] V. Adve, C. Lattner, M. Brukman, A. Shukla, and B. Gaeke, "LLVA: A Low-level Virtual Instruction Set Architecture," in *Proceedings of the 36th annual ACM/IEEE international symposium on Microarchitecture (MICRO-36)*, (San Diego, California), Dec 2003.

[19] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge, UK: Cambridge University Press, 2000.

[20] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc. AFIPS Conf.*, (Reston,Virginia), pp. 483–485, 1967.

[21] G. S. Taylor, J. K. Ousterhout, G. T. Hamachi, R. N. Mayo, and W. S. Scott, "Magic: A vlsi layout system.," in *Proceedings of the 21th Design Automation Conference*, pp. 152–159, 1984.

[22] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for FPGAs," in *Proceedings of ACM Symp. on Field-Programmable Gate Arrays*, pp. 111–117, 1995.

[23] T. Yang and A. Gerasoulis, "List scheduling with and without communication delays," *Parallel Comput.*, vol. 19, no. 12, pp. 1321–1344, 1993.

[24] H. Rosé, T. Asselmeyer-Maluga, M. Kolbe, F. Nieh$^{\text{f}}$orster, and A. Schramm, "The fraunhofer quantum computing portal - www.qc.fraunhofer.de - a web-based simulator of quantum computing processes," tech. rep., Fraunhofer Institute for Computer Architecture and Software Technology, Berlin, 2003.

[25] A. Fog, "Chaotic random number generators with random cycle lengths." www.agner.org/random/theory, Nov 2001.

[26] A. Steane, "Active stabilisation, quantum computation and quantum state synthesis," *quant-ph/9611027*, 1996.

[27] M. Oskin, F. Chong, and I. Chuang, "A practical architecture for reliable quantum computers," in *Proc. International Symposium on Computer Architecture (ISCA 2001)*, (New York), ACM Press, 2001.