A Combinatorial Noise Model for Quantum Computer Simulation

Eric Chi, Stephen A. Lyon, Margaret Martonosi
Dept. of Electrical Engineering, Princeton University
{echi,lyon,mrm}@princeton.edu

May 18, 2007

Abstract

Quantum computers (QCs) have many potential hardware implementations ranging from solid-state silicon-based structures to electron-spin qubits on liquid helium. However, all QCs must contend with gate infidelity and qubit state decoherence over time. Quantum error correcting codes (QECCs) have been developed to protect program qubit states from such noise. Previously, Monte Carlo noise simulators have been developed to model the effectiveness of QECCs in combating decoherence. The downside to this random sampling approach is that it may take days or weeks to produce enough samples for an accurate measurement.

We present an alternative noise modeling approach that performs combinatorial analysis rather than random sampling. This model tracks the progression of the most likely error states of the quantum program through its course of execution. This approach has the potential for enormous speedups versus the previous Monte Carlo methodology. We have found speedups with the combinatorial model on the order of 100X-1,000X over the Monte Carlo approach when analyzing applications utilizing the [[7,1,3]] QECC. The combinatorial noise model has significant memory requirements, and we analyze its scaling properties relative to the size of the quantum program. Due to its speedup, this noise model is a valuable alternative to traditional Monte Carlo simulation.

1 Introduction

Quantum computing is a promising new computing paradigm that leverages some unique properties of quantum mechanics. Quantum algorithms such as Shor's factoring algorithm [9] offer an exponential speedup over the best known classical approach and have important applications in defeating public-key cryptography. Quantum computing hardware is only in early stages of development, and many different implementation styles are being pursued. These include ion-traps [5], the Kane model [4], and electron-spins on helium (eSHe) [6]. Although these hardware implementations differ vastly in terms of the mobility and robustness of quantum bits (qubits), they all must contend with noise and decoherence adversely affecting the state of the program qubits.

Quantum error correction and fault-tolerant protocols have been developed to combat decoherence in quantum computers (QCs) [7, 8, 10, 11, 12]. These approaches encode *logical* program qubits into code blocks and perform all program operations on encoded blocks so as to correct random errors and prevent such errors from propagating wildly. For example, the [[7,1,3]] quantum error correction code (QECC) encodes one logical qubit into a block of seven *physical* qubits. Every logical operation is followed by an error recovery phase that extracts error syndromes from a logical qubit code block and performs corrective procedures if necessary.

Noise modeling is critical in the development of QC architectures. When integrated into an architectural simulator, it permits evaluations of QECCs, error recovery techniques, and microarchitecture noise tolerances. The simplest noise model applies the Monte Carlo (MC) simulation approach, which randomly samples possible error scenarios. This methodology has been applied to previous studies including [1, 13]. The downside to MC simulation is its long runtime. When used to measure the crash rate of a quantum application, the MC simulator is attempting to measure the frequency of a failure event that occurs very infrequently (on the order of 1 in 100,000) with effective error correction. The simulator must run on the order of a billion iterations to obtain sufficiently many samples for an accurate measurement. For example, a small quantum program with only two logical qubits required nearly eight days of runtime on the MC simulator to achieve three significant digits of accuracy.

We propose an alternative combinatorial noise model that is deterministic and tracks the most probable error scenarios during program execution. We have built and evaluated this combinatorial noise model, and we will show that this approach matches the MC model's accuracy with much faster simulation times.

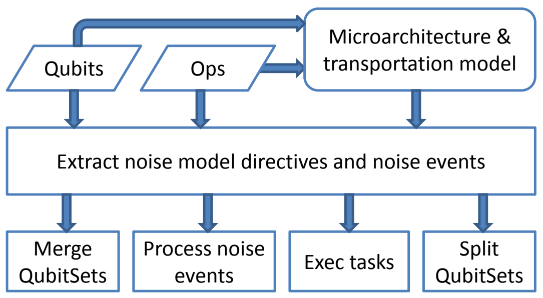
Qubits
Ops
Microarchitecture & transportation model
Extract noise model directives and noise events
Merge QubitSets
Process noise events
Exec tasks
Split QubitSets

Figure 1: The operational flow of the combinatorial noise simulator. The quantum program is defined as a set of physical qubits and operations. This program information is fed into a microarchitecture model that manages resources and calculates timing. All of this information is fed into the noise model. The noise model processes directives annotated in the operations to merge or split QubitSets and to calculate interesting event probabilities. It also creates a set of noise events for every qubit and every operation to model decoherence and operational errors.

This paper proceeds as follows: Section 2 describes our combinatorial noise model; Section 3 describes our experimental methodology; Section 4 presents our results; Section 5 presents related work; and Section 6 concludes.

2 A combinatorial approach to noise simulation

2.1 Overview

The operational flow of our noise simulator is depicted in Figure 1. The noise model works in conjunction with a microarchitecture simulator that models the quantum processor; it manages memory, transportation, and execution resources and calculates timing through program simulation. A quantum program consists of a set of operations acting on a set of physical qubits. The operations are organized into a sequence of VLIW instruction bundles, so a different set of independent operations are presented to the processor every cycle. All this information is fed to the noise model.

The error state of an individual qubit is associated with a Pauli operator: I, X, Y, or Z. The I state indicates the error-free state. The X, Z, and Y states indicate the presence of a bit-flip, phase-flip, or both bit- and phase-flip errors, respectively. Each error state is associated with a probability value, and a qubit may be in a superposition of multiple error states.

The simulated physical qubits are partitioned into QubitSets that track correlated errors among the qubits in a set. Error states in a QubitSet are represented by Pauli strings: sequences of Pauli operators that define the combined error state of multiple qubits. The QubitSet maps its Pauli strings to probability values in a hash table: the QubitSet's *error map*. All the noise model's operations interact with these QubitSets by manipulating their error maps.

2.2 Processing noise events

As a quantum program is simulated, the qubits evolve to carry errors with some probability. Errors may arise from decoherence over time, from transporting qubits, or from faulty operations. Every cycle, these potential error scenarios are quantized into *noise events* for processing by the noise model. A cycle of execution involves processing a VLIW bundle of operations. A noise event is created for each operation in this bundle and represents the chance of a faulty operation. One or more noise events are also created each cycle for every physical qubit. Decoherence is modeled as a noise event with the probability proportional to the time length of the cycle and dependent on whether the qubit was undergoing an operation or merely sitting idly in memory. Transportation decoherence is also modeled as a separate per-qubit noise event proportional to time spent travelling.

A noise event targets either one or two qubits (to support two-qubit operations), and it affects the QubitSet associated with the target qubit(s). In the case of two-qubit noise events, both qubits should belong to the same QubitSet to track correlated errors. The noise event is stochastic; it leaves the QubitSet unchanged with probability $1-f$, the probability that the event does not trigger an error. The noise event generates new error states with total probability f divided evenly into the three error types (X, Y, and Z) for one-qubit events or into fifteen error types (IX, IY, IZ, XI, XX, XY, XZ, YI, YX, YY, YZ, ZI, ZX, ZY, and ZZ) for two-qubit events following the example of [13]. The noise model implements the noise event by creating a new error map and expanding every

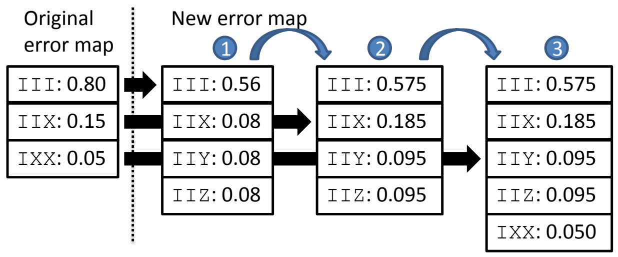

Figure 2: A noise event with probability 0.3 acts on bit 0 of a 3-bit QubitSet. Assume that the event branch threshold is 0.1. The noise event creates a new error map and (1) initially expands the error-free III state into four states. (2) The *IIX* state is also expanded with new error scenarios and added to the new error map. (3) The *IXX* state falls below the event branch threshold so is appended to the new error map without expansion.

error state in the original error map into four or sixteen new error states for the new error map.

This noise event process would lead to exponential growth in the number of entries in the error map. To avoid this exponential growth, we only apply a noise event to a particular error state if that state's probability value is greater than some specified *event branch threshold*. With an appropriately selected threshold, this approach avoids the creation of minute error states that are relatively insignificant. The noise event process including the application of the branch threshold is illustrated in Figure 2.

Two-qubit operations introduce an additional complication in that errors may propagate between the two operand qubits. We follow the example set by [13] in defining error propagation properties. Error propagations are implemented in our noise model as simple transformations of error states. For example, given a 3-qubit state *XII*, a CNot operation on the first two bits may propagate the bit-flip error, transforming the state to *XXI*.

2.3 Merging and splitting QubitSets

Two-qubit noise events require their qubits to belong to the same QubitSet to enable tracking of correlated errors between those qubits. This requires the ability for QubitSets to *merge* during runtime. The merging procedure is straightforward: first, a new error map is created for the merged QubitSet. Every error state in the first error map is multiplied with every error state in the second error map to create new error states for the merged error map. Again, to avoid an enormous expansion of the state space, a *merge branch threshold* is applied during the merge process so that a new merged state is created only if its resultant probability is greater than the merge branch threshold.

We have two approaches to handling merged error states that fall below the merge branch threshold: the *preservation* approach and the *lossy* approach. The preservation approach takes the less probable of the two states to be merged and converts that state to the error-free state. This way, the error information in the more probable error state is retained in the resultant merged state. The lossy approach simply discards the low-probability merged error states. These two approaches will yield near-similar results when the merge threshold is set appropriately. If the merge threshold is too large, the preservation approach will overestimate the success rate while the lossy approach will underestimate the success rate. This is because the less probable an error state is, the more errors it is likely to have (noise events are assumed to be improbable). The preservation approach converts low probability states with higher error weights (the number of bits in a Pauli string bearing errors) to lower error-weight states, reinforcing their probabilities. These lower weight error states are more likely to be correctable by QECCs, so the resultant success rate is inflated. Conversely, the lossy approach discards error states and may lead to an undercounting of successful error states. Figure 3 illustrates merging QubitSets with these two approaches.

After qubits have been measured, it is no longer necessary to track their error states. A QubitSet may then *split* to remove the measured qubits and reduce the size of its state space. Splitting a QubitSet partitions a QubitSet into two smaller QubitSets and is the inverse procedure of merging. No thresholds are necessary for this procedure as the state space never expands while splitting.

Directives to the noise model for merging and splitting QubitSets are manually embedded into the operations comprising the quantum program. The annotation of these operations may be automated in the future as the rules for merging and splitting are straightforward based on dependency analysis. Two-qubit noise events or any sort of analysis on multiple qubits require that the qubits reside in the same QubitSet and implicitly requires merg-

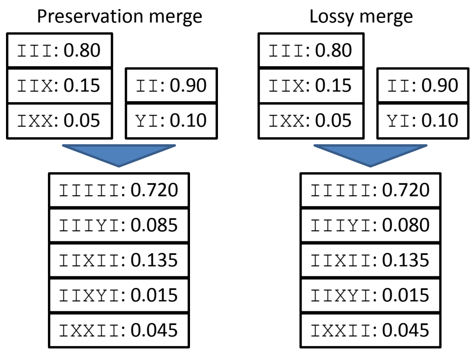

Figure 3: Two QubitSets merge with a merge threshold of 0.01. The merged error state $IXXYI$ has a probability of 0.005, which is below the merge threshold. The preservation approach converts this merged state to $IIIYI$ and avoids creating a new error state with minuscule probability. On the other hand, the lossy approach simply discards this error state.

ing QubitSets when necessary. Following the final measurement of a subset of qubits in a QubitSet, the noise model may split those qubits from the QubitSet to improve performance on subsequent operations acting on that QubitSet.

2.4 Special tasks

We have described how the noise model evolves through program execution by processing noise events and manipulating error maps in QubitSets. This subsection describes the remaining functions of the noise model; these include modeling error correction effects and measuring event likelihoods.

During program execution, the noise model may be directed to calculate interesting event probabilities such as the success rate in preparing ancilla blocks without errors or the probability of extracting a correct syndrome. These probabilities are easily measured by iterating through the QubitSet's error map and summing the probabilities of those error states that match the desired event. For example, to measure the overall success rate of a program, all the program qubits should be in the same QubitSet, and the success rate is the sum of all error state probabilities in which the program qubits are free of errors or have sustained a correctable number of errors (this is a QECC-dependent value).

movement speed	100 μm/μs
1-bit op time	1 μs
2-bit op time	1 ms
memory decay constant	1×10^5 s
operation decay constant	5×10^3 s
transportation decay constant	2.5×10^4 s
1-qubit op error rate	1×10^{-6}
2-qubit op error rate	1×10^{-4}
measurement error rate	1×10^{-4}
reset error rate	1×10^{-6}

Table 1: Timing and noise parameters for the eSHe architecture described in [2].

We have described noise events as the primary means for modifying error states. However, the software error correction operations also impact the qubits' error states. We implement the error correction effects as special noise model tasks that are associated with specific operations. These tasks include ancilla verification, syndrome extraction, and data correction. The overall correction routine is described in [13]. These tasks work similarly to noise events: a new error map is created; the simulator iterates through the old error map entries from which new error map entries are created with some appropriate transformation.

3 Simulation methodology

This section describes our experimental setup. We programmed both our proposed combinatorial noise model and a simple MC noise model in Java. For concreteness, we adopted the eSHe QC architecture [2] for noise and timing parameters (Table 1), but the modeling approach is applicable to any architecture.

The MC noise model that we compare our approach to is simple and follows the methodology defined in [13]. We use the Mersenne Twister pseudo-random number generator (RNG) that is bundled with RngPack 1.1a [3]. This RNG has an exceptionally long period of $2^{19937} - 1$.

Our simulated quantum applications focus on the error recovery procedures that dominate every program. We adopt the [[7,1,3]] QECC for this paper. Error recovery consists of two phases: one to detect and correct bit-flip errors, and the other, phase-flip errors. Each of these phases consists of three syndrome extractions using specially prepared and verified ancilla blocks. If a majority syndrome exists and indicates an error, correction procedures are applied. The [[7,1,3]] QECC is only capable of correcting general errors of weight one per code block. If more than one qubit in a block has an error, then the block is considered to be uncorrectable, and the QC is considered to have crashed. The combinatorial noise simulator tallies up all the error states that do not crash, and the MC simulator tallies the number of iterations that do not crash.

We do not focus on higher-level program behavior for this paper. Logical qubits are entangled with each other through logical CNots, and every logical operation is followed by error recovery. The final cycle of program execution is concluded with measurement of the program qubits.

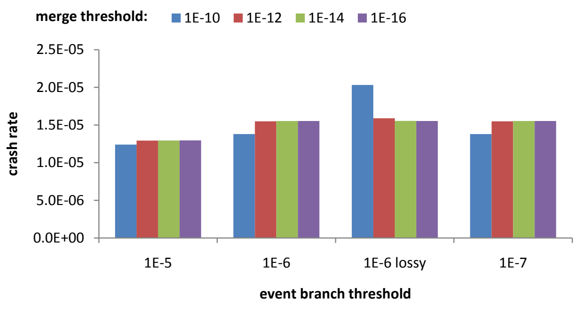

Figure 4: A parameter analysis of the combinatorial noise model. We measure the crash rate of a simple, 2-logical qubit application with various threshold parameters. Unless otherwise noted, merges take the preservation approach (rather than lossy approach) when error states fall below the merge threshold. Larger thresholds lead to an underreporting (or over-reporting for lossy merges) of crash probabilities. Preservation merges are more accurate than lossy merges, but the distinction dissipates as the merge threshold is reduced.

4 Results

4.1 Threshold parameter exploration

We begin our results section by highlighting the impact of the threshold parameters for the combinatorial noise model. This first application is particularly simple and involves two logical qubits. Following an initial logical CNot, error recovery is performed on both logical qubits. A second logical CNot is performed, followed by measurement. We perform a parameter analysis by varying the event branch threshold from 10^{-5} to 10^{-7} and the merge threshold from 10^{-10} to 10^{-16}. The results are illustrated in Figure 4.

We find that an event threshold of 10^{-6} or smaller leads to accurate results for this application. The merge threshold then dominates accuracy with a threshold of 10^{-12} yielding good results; smaller merge thresholds yield subtly better accuracy. We default to the preservation merge approach when the error states fall below the merge threshold. However, we illustrate the results of lossy merges for the 10^{-6} event threshold. As the gap between the preservation and lossy merge results diminishes, the user may have greater confidence that the merge threshold is not impacting the final result.

4.2 Scalability with program size

While the combinatorial model is faster than the MC model, scalability is a concern. The basic two-logical qubit program used in the last subsection consumed several gigabytes of memory for the combinatorial model compared to hundreds of megabytes for the MC simulator. This subsection analyzes how the program runtime and error map sizes scale with program size.

We extend our basic quantum program to support an arbitrary number of logical qubits. Given N logical qubits, our scalability program applies $N-1$ logical CNots and recoveries spread over $\log_2 N$ phases. All the logical qubits merge into the same QubitSet by the end of the program, so this should give us an indication of how the error map scales with larger programs.

Figure 5 plots how the combinatorial model's runtime scales with the program size. The event and merge threshold parameters are kept constant here at 10^{-6} and 10^{-12}. The runtime appears to scale polynomially with the program size. Garbage collection overhead increases as memory becomes more scarce for a program size greater than 7 logical qubits. We ran these tests with a Java virtual machine (VM) heap size of 11 GB.

To estimate the memory requirements for the simula-

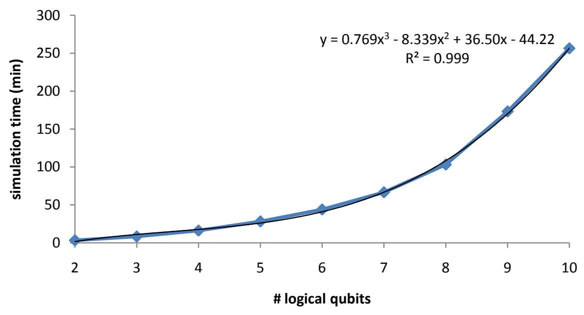

Figure 5: Simulation time for the combinatorial noise model scales at a cubic rate with respect to the number of logical qubits in the program. Memory scarcity impacts runtimes as the program size increases beyond seven logical qubits.

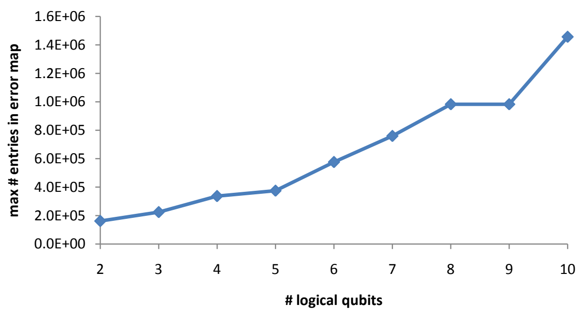

Figure 6: The maximum error map size scales more or less linearly as we scale the number of logical qubits in the application.

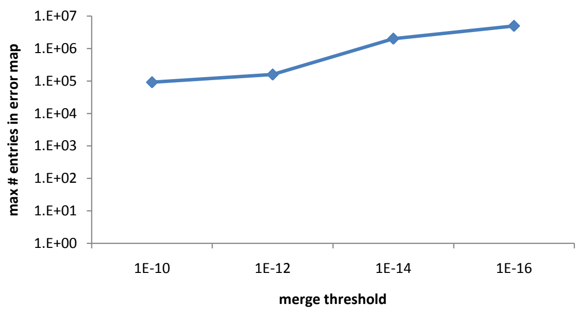

Figure 7: Maximum error map size vs merge branch threshold. There is a large jump between 10^{-12} and 10^{-14}. These numbers are from the basic quantum program with two logical qubits.

tor, we experimented with restricting the Java VM heap size until insufficient memory was available to complete simulation. We estimate that about 1 kB of memory is required for every entry in the error map. This 1 kB footprint includes memory for the Pauli string describing the error state and the double for its probability, as well as the duplicate entry in a shadow map. Because the error map is rebuilt for every noise event, a shadow map is utilized to avoid constantly reallocating hash table memory. We have profiled the runtime memory usage of the noise simulator and found the Pauli strings to be consuming the largest memory footprint. Packing the Pauli strings to two bits per qubit would lead to better memory efficiency with the error map entry footprint dropping to 64-128 bytes.

Figure 6 plots how the error map size scales with increasing program size. This scaling appears to be roughly linear, so improving the memory efficiency of our Pauli strings would likely increase the capacity of the noise model by a factor of ten. Figure 7 plots how the error map size scales with various merge thresholds. This gives an indication of the cost associated with greater accuracy, which may be necessary for larger applications.

Because of the hefty memory requirements, the combinatorial noise model is best suited to analyzing smaller software routines from which larger application behavior may be derived. We expect this model to be able to handle applications with QubitSets as large as 100-200 physical qubits with our memory capacity of 12 GB. To give some perspective on the number of physical qubits, the program with nine logical qubits has a maximum QubitSet size of 105 physical qubits. Greater memory capacity will, of course, increase the program size capability, and distributing the simulator across multiple computers may be an effective method to accomplish that.

4.3 Comparison with Monte Carlo simulation

Figure 8 plots the measured crash rate using the MC model with varying numbers of iterations. As expected, the accuracy increases with increased sampling, and the longest MC run measured a crash rate of 1.56×10^{-5}, which is only 0.22% off from the combinatorial model's result. However, this level of accuracy for the MC model comes at a great time cost: that simulation took 7.9 days

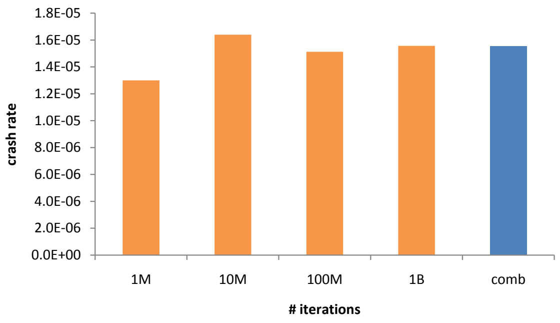

Figure 8: Increasing the number of iterations for MC simulation increases the accuracy of the predicted crash rate. With one billion iterations, the MC model measures 1.56×10^{-5}, just 0.22% off from the combinatorial model's result.

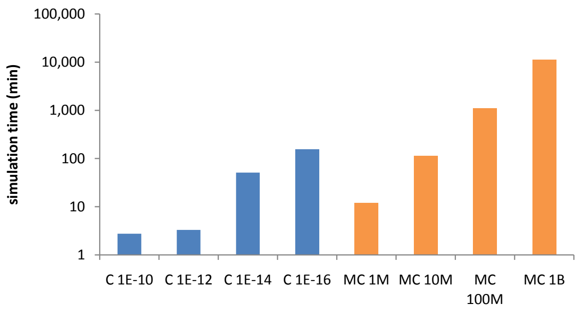

Figure 9: Simulation times for the combinatorial and MC noise models. The combinatorial model uses an event branch threshold of 10^{-6} and varies the merge threshold. The MC model varies the number of iterations. Comparing the combinatorial 10^{-12} merge threshold (smaller merge thresholds yield only subtle increases in accuracy) to the MC run with one billion iterations results in a speedup of over 3,400x in favor of the combinatorial model.

to execute. Figure 9 plots the simulation times for both the combinatorial and MC noise models. The combinatorial model was executed on a 12 GB, 1.8 GHz Opteron computer, and the MC model was executed on 4 GB, 2 GHz Athlon 64 X2 computers. The combinatorial model with the 10^{-6} and 10^{-12} event and merge thresholds has a speedup of over 3,400X compared to the one billion iteration MC run with a similar level of accuracy.

We tested the noise models on a variety of other applications and found similar agreement on crash rates between the two noise models but varying speedups. If we evaluate the [[7,1,3]] code using only a single syndrome extraction (rather than the three syndromes normally employed), the crash rate grows to 2.1%. For a given level of precision, the MC model requires a number of iterations proportional to the crash rate of the program. In other words, the MC model's runtime is largely proportional to the effectiveness of the QECC. When the [[7,1,3]] code is rendered ineffective with only a single syndrome extraction, the MC model completes in 55 minutes. The combinatorial model completes in 2.7 minutes, which is a speedup of 20x over the MC model. We are primarily interested in evaluating effective QECCs, and the combinatorial model will likely have much greater speedups for such applications.

5 Related work

Steane's work evaluating various QECCs [13] is most relevant to this one. In that work, Steane applied a MC simulator to evaluate the [[7,1,3]] and [[23,1,7]] QECCs. He also created a numerical analysis model to evaluate a larger set of QECCs and used his MC model to verify and calibrate his numerical model. His numerical model is similar to our combinatorial model in that he employs a branching probability tree to enumerate possible crash states and their probabilities. Our model differentiates itself by being implemented in a simulator framework that enables flexibility in analyzing a variety of programs rather than a single recovery methodology. Furthermore, our model evaluates a wider range of noise sources, including decoherence due to transportation of qubits.

Balensiefer *et al.* included a Monte Carlo noise model in their evaluation framework for ion-trap QCs [1]. Like our work, their noise model is integrated into a microarchitecture-level simulator. They applied their noise model to predict error rate thresholds: the maximum per-qubit failure rate that would be correctable via fault-tolerant error correction procedures. They found that when all architecture and current technology limitations are accounted for, the threshold lies around 10^{-9}, which is several orders of magnitude lower than the ideal threshold of 10^{-4}.

6 Conclusion

We have presented a new combinatorial noise model for QC simulation that offers the potential for speedups of the order of 100X to 1,000X compared to Monte Carlo simulation. Like MC simulation, our model has runtime parameters that offers the user trade-offs between accuracy and simulation speed enabling its use for a wide range of applications. While the scalability of our noise model is memory-limited, we estimate that it is capable of analyzing interesting problems involving hundreds of physical qubits when simulated on a computer with 12 GB of memory.

For future work, we are interested in quantifying speedups for a greater range of situations. We are also interested in applying the noise model to compare different QECCs, different approaches to applying QECCs, and methods for implementing fault-tolerant logical operations. Finally, a practical noise model such as the combinatorial model is a crucial simulation component for comparing different QC implementations.

References

[1] S. Balensiefer, L. Kregor-Stickles, and M. Oskin. An evaluation framework and instruction set architecture for ion-trap based quantum micro-architectures. In *ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 186–196, Washington, DC, USA, 2005. IEEE Computer Society.

[2] E. Chi, S. A. Lyon, and M. Martonosi. Tailoring quantum architectures to implementation style: A quantum computer for mobile and persistent qubits. In *ISCA '07*, 2007.

[3] P. Houle. Rngpack: High-quality random numbers for java. `http://www.honeylocust.com/RngPack/`, Nov. 2003.

[4] B. E. Kane. A silicon-based nuclear spin quantum computer. *Nature*, 393(6681):133–137, May 1998.

[5] D. Kielpinski, C. Monroe, and D. J. Wineland. Architecture for a large-scale ion-trap quantum computer. *Nature*, 417:709–711, June 2002.

[6] S. A. Lyon. Spin-based quantum computing using electrons on liquid helium. *Phys. Rev. A*, 74:052338, 2006.

[7] J. Preskill. Reliable quantum computers. *Proc. Roy. Soc. Lond.*, A454:385–410, 1998.

[8] P. W. Shor. Fault-tolerant quantum computation. In *IEEE Symposium on Foundations of Computer Science*, pages 56–65, 1996.

[9] P. W. Shor. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Sci. Statist. Comput.*, 26:1484, 1997.

[10] A. M. Steane. Error correcting codes in quantum theory. *Phys. Rev. Lett.*, 77(5):793–797, Jul 1996.

[11] A. M. Steane. Active stabilisation, quantum computation and quantum state synthesis. *Phys. Rev. Lett.*, 78:2252–2255, 1997.

[12] A. M. Steane. Efficient fault-tolerant quantum computing. *quant-ph/9809054*, 1998.

[13] A. M. Steane. Overhead and noise threshold of fault-tolerant quantum error correction. *Phys. Rev. A 68, 042322*, 2002.