Reversible Fault-Tolerant Logic

P. Oscar Boykin
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL, USA
boykin@ece.ufl.edu

Vwani P. Roychowdhury
Electrical Engineering Department
University of California, Los Angeles
Los Angeles, CA, USA
vwani@ee.ucla.edu

Abstract

It is now widely accepted that the CMOS technology implementing irreversible logic will hit a scaling limit beyond 2016, and that the increased power dissipation is a major limiting factor. Reversible computing can potentially require arbitrarily small amounts of energy. Recently several nano-scale devices which have the potential to scale, and which naturally perform reversible logic, have emerged. This paper addresses several fundamental issues that need to be addressed before any nano-scale reversible computing systems can be realized, including reliability and performance trade-offs and architecture optimization. Many nano-scale devices will be limited to only near neighbor interactions, requiring careful optimization of circuits. We provide efficient fault-tolerant (FT) circuits when restricted to both 2D and 1D. Finally, we compute bounds on the entropy (and hence, heat) generated by our FT circuits and provide quantitative estimates on how large can we make our circuits before we lose any advantage over irreversible computing.

1 Introduction

There are several compelling reasons for a renewed interest in reversible computing systems: First, it is now widely accepted that the CMOS technology implementing irreversible logic will hit a scaling limit beyond 2016, and the increased power dissipation is a major limiting factor. Reversible computing[2, 17, 6] can potentially require zero or very little energy. Second, several new nano-scale devices which have the potential to scale, and which naturally perform reversible logic, have emerged. This paper addresses several fundamental issues that need to be addressed before any nano-scale reversible computing systems can be realized, including:

1. *Reliability and Performance Trade-offs:* Current nano-scale logic proposals appear to provide extremely unreliable devices, requiring extensive use of fault-tolerant (FT) circuits. We provide a systematic design for reversible FT circuits, which will work reliably *even if each gate has an error probability as high as* $1/108$. We also calculate the blow up in size and gate count that would result from the use of such FT circuits.

2. *Architecture Optimization:* many of the proposed nano-scale devices will be limited to only near-neighbor interactions[12, 10, 19], requiring careful circuit optimization. We provide efficient FT circuits when restricted to both 2D and 1D. We show that for a 2D topology with near neighbor connections, the error threshold decreases only to $1/273$, and that a 1D lattice that is 27 bits wide but arbitrarily long has an error threshold only 23% less than the full 2D case.

3. *Power Dissipation:* Reversible computing systems in the presence of errors will generate heat. We compute bounds on the entropy (and hence, heat) generated by our FT circuits and provide quantitative estimates on how large can we make our circuits before we lose any advantage over irreversible computing.

Many previous works have considered gate-level fault tolerance techniques for irreversible gates[18, 13, 9, 8]. Local fault tolerance schemes for irreversible automata have also been studied [7]. Quantum computers are reversible; however, the properties of quantum errors and quantum information are sufficiently different from the classical case that fault-tolerant quantum computation[16, 14] is not directly applicable to the traditional reversible classical computing model, which is the subject of this work.

This paper is organized into three main parts. In Section 2, we give our model of noisy reversible gates and give a circuit fault-tolerant error-recovery and compute the overhead that the circuit requires both in time and space. We show that, as long as the error rate is below a threshold, the circuit can be made reliable. In Section 3, we apply the results of Section 2 to locally connected models where bits can only operate on their nearest neighbors. We consider the local problem in $1D$ and $2D$. In Section 4, we compute how much entropy and heat is dissipated in the error-recovery process, and we see that the entropy saving aspect of reversible computing is lost in our scheme once the error rate gets close to the threshold.

2 Reversible majority multiplexing

In standard irreversible computing, we often imagine that functions are represented by circuits of wires and gates at fixed positions. We can think of the bits as moving through the wires to different gates. In reversible computing, since gates have identical numbers of input bits and output bits, we have a choice: we can picture the bits moving through wires to gates at fixed locations, or picture the bits as fixed locations in space and the reversible function as a sequence of reversible gates *applied on the bits.* This is commonly represented as the gate array notation where space is on the y-axis and time is on the x-axis, and operations are boxes or symbols that connect the bits they are applied to (see, for instance Figure 1). This model is realizable in many nanocomputing proposals[12, 10, 19][1]

The error model is a simple independent gate failure model: at each application, a gate will randomize all the bits it is applied to with probability g. While this model does not directly address correlated failures, it will apply as long as the probability that k out of G gates fail is *less than* $\binom{G}{k}g^k(1-g)^{G-k}$. For most of this paper, we will assume that this error rate applies to any three-bit gate, and that we have access to no fault-free operations. Our goal is to make larger modules of T reversible gates with a module error rate which is independent of T, and is in general much smaller than $1-(1-g)^T \approx gT$ (which is what we could expect without any fault tolerance). In fact, we show that by using $O(T\log^{4.75} T)$ gates instead of just T, we obtain an error rate that is constant in T.

Rather than explicitly deal with error correction codes, the best gate-level, fault-tolerant schemes for classical computing are those based on Von-Neumann multiplexing[18]. In this case, each bit is copied to a second bit, a random permutation is applied, and finally, a NAND gate is applied to each pair of bits. These approaches make use of the irreversible NAND gate. Schemes such as this can result in fault-tolerant computation as long as the gate error rate is less than about 11%[18].

[1]It should be noted that, since all quantum computers are reversible, this work is particularly applicable to classical processing on quantum hardware.

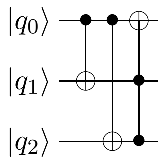


Figure 1. The reversible Majority gate constructed from two controlled-not gates and one Toffoli gate. The horizontal lines represent bits. Each vertical line connecting horizontal lines represents a gate. If every bit connected by a filled dark circle on a particular vertical line has the value one, the bit on the horizontal line with the $\oplus$ is flipped. Time flows from left to right.

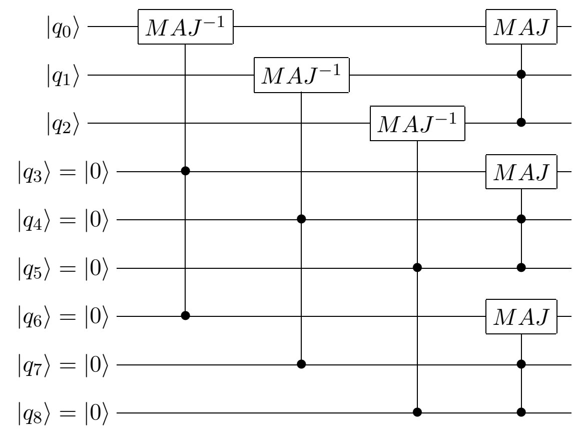
Input	Output
000	000
001	001
010	010
011	111
100	011
101	110
110	101
111	100

Table 1. The truth table for the reversible MAJ gate. Note that each input has a unique output, and that the first bit of the output is the majority of the input bits. This gate is obtained by flipping the second two bits if the first bit is 1, and then flipping the first bit if the second two bits are 1.

Rather than base our multiplexing scheme on NAND or some derivative, we base ours on a reversible extension of the majority gate (MAJ), depicted in terms of CNOT and Toffoli in Figure 1[2]. The reversible majority gate (MAJ), has a truth table given in Table 1.

We claim that the circuit in Figure 2 is in fact a

[2]We note that variants of the MAJ gate have found application in algorithmic cooling[5] and reversible addition[4]; thus MAJ appears to be a valuable gate for reversible and quantum computers.



$|q_0\rangle$
$|q_1\rangle$
$|q_2\rangle$
$|q_3\rangle = |0\rangle$
$|q_4\rangle = |0\rangle$
$|q_5\rangle = |0\rangle$
$|q_6\rangle = |0\rangle$
$|q_7\rangle = |0\rangle$
$|q_8\rangle = |0\rangle$
MAJ^{-1}
MAJ^{-1}
MAJ^{-1}
MAJ
MAJ
MAJ

Figure 2. A reversible multiplexing scheme based on the 3-bit repetition code. The output bits are those bit positions 0,3,6. The rest may be discarded. This circuit is the error-recovery circuit referred to as E_L in Figure 3

fault-tolerant error correction circuit. To see this, consider the code space of 000 representing logical zero (0_L) and 111 (1_L) representing logical one. Consider Figure 2 as having two phases, encoding and decoding. The first three MAJ^{-1} gates are the encoding gates and the last three MAJ gates are decoding gates. After the encoding gates the bits should all have the same value (i.e. 000000000 or 111111111 if the input was 0_L or 1_L, respectively). Decoding puts the majority value of each block of three bits into the three output bits. Clearly, if there are no errors, the circuit should output exactly what was input, due to the symmetry of the code under permutations. Now we simply observe that, if any single error occurs, it will change at most one bit in each of the final decoder blocks. Since the decoders return the majority result in their output bit, a single bit flip will not change the majority result. If one of the final MAJ gates has an error, it will only effect one bit in the output, and that can be repaired in the next error-recovery cycle. Thus we have a fault-tolerant error-recovery because we can tolerate errors in our recovery gates.

Since the codewords in this system are repetition code words, we can use any universal, reversible set of gates for computation directly on the repetition codewords. After each gate operation, we apply our error-recovery circuit from Figure 2. Now that we have a circuit, we can ask: for what error rates will this circuit perform as expected?

2.1 Concatenation

In order to suppress the probability of error, we code one bit as three bits in a repetition code. But why stop there? We could also code each of the three bits into another repetition code, resulting in nine bits, and so forth. This method is called concatenation.

We say a physical bit is level 0, a three-bit code 000 or 111 is level 1, a nine bit code is level 2, a 3^L bit code is level L. A bit encoded at level L is made up of three bits encoded at level $L-1$. A gate at level 0 is a physical gate to which we have access (in our case, MAJ). To implement a logical gate at level 1, we apply the gate at level 0 to each of the three bits in the code for level 1. To implement a 3-bit gate at level L, we apply the gate at level $L-1$ on the logical bits in the code, and then correct any errors we may have caused in each of the bits. This is depicted in Figure 3.

2.2 The threshold for fault-tolerant computation

Throughout this work, we consider the gate error rate g as the error rate for a 3-bit operation. Thus, we assume that we can reset three bits with one initialization operation. The error-recovery circuit depicted in Figure 2 requires us to initialize six bits (two 3-bit initialization operations), apply three MAJ^{-1} gates, and three MAJ gates for a total of eight gate operations (six if initialization can be assumed to be far more accurate than our gates). As previously shown, as long as there is no more than one error in all of these operations, the final result will not be an error. We say that the error-recovery circuit requires E gates.

In addition to the error-recovery, we also have the logical gate which we want to apply on the data. To apply our logical gate, we need to operate on each of the three bits in the code, which gives us three more gates which can go wrong.

A particular bit will be correct unless there are two or more errors. Thus, if each operation has an error rate g and there are $G = 3 + E$ operations acting on each encoded bit (some act on more than one encoded bit), the bit error rate P_{bit} is:

$$\begin{aligned} P_{bit} &\leq \sum_{k=2}^{G} \binom{G}{k} g^k (1-g)^{G-k} \\ &\leq \binom{G}{2} g^2 \end{aligned}$$

The probability that a gate has no error is at least as large as the probability that none of the bits are in error if each were considered independently:

$$1 - g_{logical} \geq (1 - P_{bit})^3$$

The above is true because the right hand side triple counts the case where the logical gate (the gate applied to all three bits) fails. We note that the above bound is a convenient bound, but a tighter bound will result in an improved error threshold. We can use the above to see that:

$$\begin{aligned} g_{logical} &\leq 1 - (1 - P_{bit})^3 \\ &\leq 3P_{bit} \\ &\leq 3\binom{G}{2}g^2 \end{aligned} \tag{1}$$

Thus if we want $g_{logical} < g$, it is sufficient to have $g < \frac{1}{3\binom{G}{2}}$, which we call the threshold. In our cases of $G = 11$ $(3 + (E = 8))$ and $G = 9$ $(3 + (E = 6))$, we get threshold results of $\rho = 1/165$ and $\rho = 1/108$, respectively.

The above only shows that we can decrease the logical error rate by applying the error-recovery circuit; it does not show that we can make it as small as we like. In order to push the error rate to lower values, we *concatenate* our bits recursively. At the lowest logical level, $L = 0$, each bit is represented by a physical bit. At all other levels, each bit at level L is represented by 3-bits at level $L - 1$. Thus, at level L we are actually using 3^L physical bits. We only pay attention to the error rates at the largest level, but after each gate at level L we do an error-recovery at level L, which in turn applies gates at level $L - 1$, and so on, until we reach the bottom level. This recursive structure is represented in figure 3.

Thus, if the error probability at level k is g_k, $g_0 = g$, and we have G total operations to perform, Equation 1 tells us that:

$$g_{k+1} \leq 3\binom{G}{2}g_k^2$$

To solve equations like the one above, we introduce $\kappa = \log 3\binom{G}{2}$, and $r_k = \log g_k$:

$$\begin{aligned} g_{k+1} &\leq 3\binom{G}{2}g_k^2 \\ \log g_{k+1} &\leq \log 3\binom{G}{2} + 2\log g_k \\ r_{k+1} &\leq \kappa + 2r_k \\ (r_{k+1} + \kappa) &\leq 2(r_k + \kappa) \\ r_k &\leq 2^k(r_1 + \kappa) - \kappa \\ g_k &\leq \frac{(3\binom{G}{2}g_0)^{2^k}}{3\binom{G}{2}} \end{aligned}$$

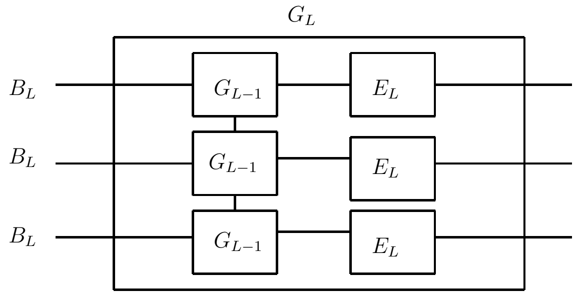


Figure 3. A gate at concatenation level L. We apply the gate three times at level $L - 1$ and then do an error correction cycle. G_{L-1} is the gate operation at the lower logical level, E_L is the error-recovery circuit (Figure 2) on logical level L (using only gates at level $L - 1$).

$$= \rho\left(\frac{g}{\rho}\right)^{2^k} \tag{2}$$

Assuming $G = 9$, $3\binom{G}{2} = 108$; thus, if the lowest level gate error is $g_0 < 1/108$, we can correct all the errors with high probability if we use enough levels of concatenation. Throughout this paper we use ρ for threshold values. Thus, if we define $\rho = 1/3\binom{G}{2}$, and if $g_0 < \rho$ we are sure to be able to correct all errors in the limit of large L.

2.3 Circuit blowup

In this section, we consider how much larger a module made of T perfect gates will be when constructed from the FT techniques of this paper, such that the final module will have a constant probability of error irrespective of T.

To implement a 3-bit gate at level L, we must correct 3-bits at level $L - 1$. Starting from Figure 3, we can see that if Γ_k is the number of gates required for one complete error correction and gate operation on levels k and lower, and E is the number of gates required to do error-recovery, then we have that:

$$\begin{aligned} \Gamma_k &= 3\Gamma_{k-1} + 3E\Gamma_{k-1} \\ &= 3(1 + E)\Gamma_{k-1} \\ &= (3(1 + E))^k \end{aligned}$$

Using $G = 3 + E$, we have:

$$\Gamma_k = (3(G - 2))^k$$

So there is an exponential blow-up in gate count with concatenation depth. The bit size blowup is just as easy to calculate: at each level, we use 9 bits of the level below:

$$\begin{aligned} S_{k+1} &= 9S_k \\ S_k &= 9^k \end{aligned}$$

How deep do we need to concatenate? If the module we want to simulate has T gates, we need $g_L \leq 1/T$ in order to have less than on average one error in the FT module. Hence, by bounding Equation 2:

$$\begin{aligned} \rho(g/\rho)^{2^L} &\leq \frac{1}{T} \\ L &\geq \log_2 \frac{\log T\rho}{\log \rho/g} \end{aligned} \quad (3)$$

If we use the minimum valid value for L, the gate blow-up factor is $\Gamma_L = O(G^L)$, and is poly-log in T:

$$\begin{aligned} (3(G-2))^L &= 2^{L \log_2 3(G-2)} \\ &= \left(\frac{\log T\rho}{\log \rho/g}\right)^{\log_2 3(G-2)} \end{aligned}$$

As is the size blow-up factor:

$$\begin{aligned} S_L = 9^L &= 2^{L \log_2 9} \\ &= \left(\frac{\log T\rho}{\log \rho/g}\right)^{\log_2 9} \\ &\approx \left(\frac{\log T\rho}{\log \rho/g}\right)^{3.17} \end{aligned}$$

For $G = 11$, we have $(3(G-2))^L = O((\log T)^{4.75})$ and $S_L = O((\log T)^{3.17})$.

Suppose we have $g = \rho/10$ with $G = 9$ and $\rho \approx 10^{-2}$. Without any error correction, modules larger than $1,000$ gates will almost certainly be faulty. If we want to make a module of $T = 10^6$, we need $L = \log_2((\log_2 10^4)/\log_2 10) = 2$, so we can make an accurate module with 10^6 gates, using 2 levels of concatenation, if $g = \rho/10$. This means that, rather than using one gate, we will need to replace each with $(3(G-2))^2 = 441$ gates and replace each bit with $3^2 = 81$ bits. However, we are able to construct a much larger module: 10^6 logical gates rather than $1,000$ logical gates.

We have shown that we can take noisy gates and create modules of bounded noise with only a poly-log overhead factor. Once we have modules with bounded noise, higher level fault tolerance techniques may be applied.

3 Local reversible fault-tolerant schemes

In this section, we consider the problem of an array of bits on which we may operate with noisy, reversible gates. We assume that we may only operate on at most three neighboring bits at a time. When it is necessary to operate on pairs of remote bits, we must first move them close together by a series of SWAP operations and then operate. This introduces extra overhead into every logical operation. Since we are assuming that each operation is noisy we expect this to reduce the error threshold. We will first consider a 2D array and see that the 2D array only requires extra SWAP operations to operate on three *logical* bits, and no extra SWAP operations to do error-recovery. As such, the threshold is not much lower than the result of Section 2.2. Later, we consider a 1D array; in this case, our error correction circuit will require many SWAP operations. We can expect this to lower the threshold much more than the 2D case.

3.1 A local 2-dimensional fault-tolerant system

In Figure 2, we presented a fault-tolerant error correction circuit which assumes that any pair of bits may be operated on; notice, however, that during the recovery process, only certain bits interacted. In practice, many systems may allow only local operations. In this section we consider a 2D lattice of bits. At each time step any adjacent pairs (or triples) of bits may interact.

If we put the circuit from Figure 2 on the lattice in Figure 4, we see that all the bits that interact in the recovery circuit are already near one another in the lattice. The only additional complication we need to consider is the difficulty of bringing three logical bits near one another in order to do a logical operation.

To operate on logical bits, we must interleave the bits, operate locally, and then uninterleave. But in which direction do we interleave? There are two directions: parallel and perpendicular to the logical bit line (see Figure 4). Interleaving three logical bits parallel to the logical line requires nine SWAP gates. Interleaving three logical bits perpendicular to the logic line requires 12 SWAP gates. However, both schemes use at most six SWAPs on a given logical bit. If we combine two SWAPs into one three bit gate, which we call $SWAP_3$, depicted in Figure 5, (and then only count three bit gates in the threshold) we only use three $SWAP_3$ gates.

Thus, a full cycle now requires six additional gate operations: three to interleave, three to uninterleave the bits[3]. Thus our total gate count is 14 if we ignore bit initialization, and 16 if we include bit initialization. As such the threshold using only local operations in

[3]The error-recovery circuit in Figure 2 actually rotates the logical bit line, but as long as all bits are recovered at the same time, this rotation is uniform throughout the circuit and can be ignored

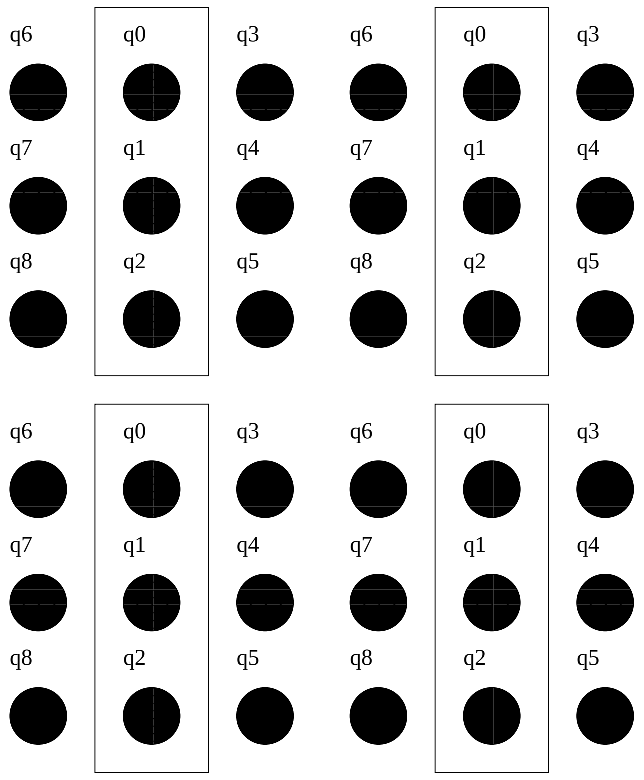


Figure 4. Layout of bits on a 2D lattice. Boxes denote the locations of the logical bits; the other bits are ancillary bits. To bring logical bits close to one another, we can either move them perpendicular to the logical line (q0,q1,q2) by swapping them past the ancillary bits in between two logical lines (q3,q4,q5 for one bit and q6,q7,q8 for the other), or we can move them parallel to the logical line, in which case the two logical bits are adjacent to each other in a line and must be interleaved (q0,q1,q2 with the next q0,q1,q2 just below it in the above figure).

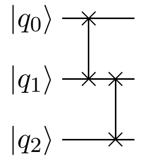


Figure 5. A $SWAP_3$ gate, which is composed of two swaps on three bits. Since this gate only acts on three bits, we assume that its error rate is at most g, the error of any 3-bit gate.

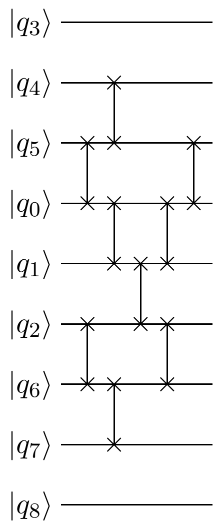


Figure 6. Interleaving three codewords which are linearly adjacent.

$2D$ becomes $\rho_2 = 1/3\binom{14}{2} = 1/273$ and $\rho_2 = 1/3\binom{16}{2} = 1/360$ respectively. Clearly, if we can initialize bits with error probability very much lower than $1/273$, they can be ignored in the threshold calculation, and we can assume that the gate error rate only needs to reach the larger threshold, which is approximately 0.4%.

3.2 A local 1-dimensional fault-tolerant system

Figure 7 uses only nearest-neighbor operations on a 1D array. The error correction circuit requires six MAJ gates, nine SWAPs, and six initializations. Instead of counting nine SWAPs, we count four $SWAP_3$ gates and one SWAP. Instead of counting six initializations we count two 3-bit initializations. This gives a total of 11 gates or 13 gates, with or without initialization, respectively.

We want to balance the number of SWAPs applied to each codeword so that no codeword is corrupted

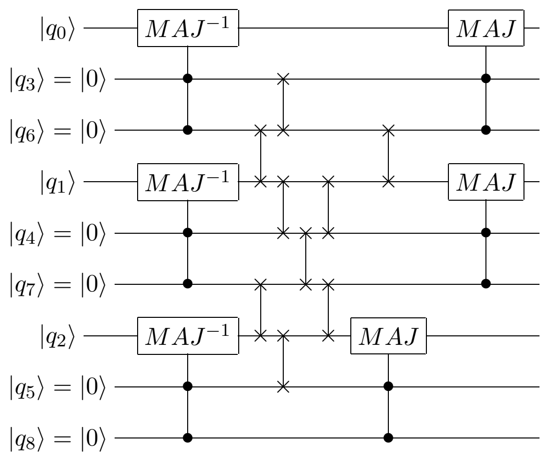

$|q_0\rangle$
$|q_3\rangle = |0\rangle$
$|q_6\rangle = |0\rangle$
$|q_1\rangle$
$|q_4\rangle = |0\rangle$
$|q_7\rangle = |0\rangle$
$|q_2\rangle$
$|q_5\rangle = |0\rangle$
$|q_8\rangle = |0\rangle$
MAJ^{-1}
MAJ^{-1}
MAJ^{-1}
MAJ
MAJ
MAJ

Figure 7. A fault-tolerant error-recovery circuit in one dimension with only local gate operations. This may be thought of as Figure 2 plus Figure 6 handling the the interleaving of the bits.

more than the other two. As such, we interleave by bringing the two outer codewords close to the middle codeword. In order to interleave three logical bits $b0, b1, b2$, we move the last bit in $b0$ just above the last bit in $b1$. Then we move the second bit in $b0$ just above the second bit in $b1$. Next, we move the first bit in $b0$ just above the first bit in $b1$. Subsequently, we do a similar operation on $b2$.

Interleaving $b0$ and $b1$ requires $8 + 7 + 6$ SWAPs (8 for the last bit, 7 for the second bit, 6 for the first bit). Interleaving $b2$ requires $10 + 8 + 6$ SWAPs (10 for the first bit, 8 for the second, and 6 for the last). This gives a total of 45 SWAPs; however, at most 24 act on a single bit. If, instead of counting SWAPs, we count $SWAP_3$ gates, we have only 12 $SWAP_3$ gates acting on each codeword to interleave.

Thus, a full operation is 12 $SWAP_3$ on each codeword to interleave, the gate operation, which touches each of the three bits in each codeword, and finally 12 $SWAP_3$ gates to uninterleave, for a total of 27 gates, in addition to the error-recovery cycle. The error-recovery cycle requires 13 gates (11 if we neglect initialization), for a total of 40 gates. This yields a threshold of $\rho_1 = 1/3\binom{40}{2} = 1/2340$ (or $\rho_1 = 1/2109$ if bit initialization is much more accurate than ρ_1). Thus, we find that ρ_1 is about an order of magnitude worse in the $1D$ case than it is in the $2D$ case. In the next section, we will see how using a few levels of $2D$ at the lowest level can recapture most of the advantage that $2D$ offers in the threshold.

3.3 Concatenating different thresholds

If a particular fault-tolerant scheme has a threshold ρ_2 and the elementary gates have an error rate of g, Equation 2 tells us that, after k levels of concatenation, the resulting error rate is less than:

$$g_k = \rho_2 \left(\frac{g}{\rho_2}\right)^{2^k}$$

If, after k levels of this scheme, we concatenate with $L-k$ levels of a scheme with threshold ρ_1, we have:

$$\begin{aligned} g_L &= \rho_1 \left(\frac{g_k}{\rho_1}\right)^{2^{L-k}} \\ &= \rho_1 \left(\frac{\rho_2 \left(\frac{g}{\rho_2}\right)^{2^k}}{\rho_1}\right)^{2^{L-k}} \\ &= \rho_1 \left(\frac{g}{\left(\frac{\rho_1}{\rho_2}\right)^{\frac{1}{2^k}} \rho_2}\right)^{2^L} \\ &= \rho_1 \left(\frac{g}{\rho(k)}\right)^{2^L} \end{aligned}$$

with $\rho(k) = \rho_2 \left(\frac{\rho_1}{\rho_2}\right)^{\frac{1}{2^k}}$. Thus, if we use k levels of a scheme with a lower threshold, we can get most of the advantages of a lower threshold, even with a small, finite number of concatenations. Table 2 shows that, after a few levels of $2D$ concatenation, the threshold approaches the $2D$ case studied in Section 3.1. In particular, a linear array nine bits wide has a threshold 60% as large as the full $2D$ case, and an array 27 bits wide has a threshold 77% as large as the full $2D$ case. This underscores the fact that most of the benefits of a $2D$ structure accrue *in the first few levels of concatenation.*

k	Width	$\rho(k)/\rho_2$
0	1	0.13
1	3	0.36
2	9	0.60
3	27	0.77
4	81	0.88
5	243	0.94

Table 2. If we concatenate k levels of 2D circuits with $L-k$ levels of 1D circuits, we see that the threshold rapidly approaches the 2D case. If we are allowed 27 lines rather than just one, we can get a threshold which is only 23% smaller than 2D case.

4 Entropy dissipation

Reversible computing has been proposed as a method to reduce power consumption of computing devices. In some quantum systems, reversible logic is all that is available, and irreversible devices must be simulated from reversible ones (by discarding or resetting bits). However, a Toffoli gate can simulate an irreversible NAND gate by dissipating at most $3/2$ bits of entropy per cycle[4]. Similarly, due to the universality of NAND, we can use NAND gates to build a functionally equivalent gate to Toffoli at a entropic expense of only a few bits. Hence, once our encoded gates dissipate $3/2$ bits of entropy per operation, we can say that we have actually used faulty, reversible gates to build fault-free *irreversible* logic.

[4]The value of $3/2$ bits is in fact optimal (assuming equally likely inputs and using only reversible logic), and may be achieved using the MAJ^{-1} gate.

It has been shown that if a reversible computer has errors, there must be a supply of fresh zero bits in order to remove entropy from the computer[1]. Here, we estimate how much entropy per gate must be dissipated during fault-tolerant operation of a noisy reversible computer. We note that when n bits have $n \times H$ bits of entropy, it is not necessary to replace them with n zero-entropy bits; instead, reversible cooling[15, 3, 5] schemes can ensure than we only need to replace $n \times H$ of them with zero-entropy bits. Thus, asymptotically, we need to calculate the expected amount of entropy in the ancillary bits, and that will correspond to the number of bit resets we will need in our system.

Landauer pointed out that where there is irreversibility in computing, there must be heat dissipation[11]. Thus, by computing the amount of entropy dissipated, we know that the heat dissipated is:

$$\Delta E \geq k_b t \Delta H$$

where ΔH is the amount of entropy dissipated, k_b is Boltzmann's constant, and t is the temperature.

We represent the number of gates at level $L-1$ needed to simulate a gate at level L as $\tilde{G}$. The value of $\tilde{G}$ will depend on the model we are working with, e.g. non-locally connected vs. locally connected. Following the calculations in Section 2.3, and due to subadditivity of entropy, if H_L is the entropy generated by one gate operation at level L, we can see that:

$$\begin{aligned} H_L &\leq \tilde{G} H_{L-1} \\ &= \tilde{G}^{L-1} H_1 \end{aligned}$$

Not every error is distinguishable, so assuming each error can be distinguished provides an upper bound. When a gate has an error, we assume it outputs totally random bits; thus, with probability $1-g$, the output is correct, and with probability g the output is one of eight equally likely outputs. Calculating the entropy of such a scenario yields $H(\frac{7g}{8}) + \frac{7g}{8}\log 7$, and we find:

$$\begin{aligned} H_1 &\leq \tilde{G}\left(H(\frac{7g}{8}) + \frac{7g}{8}\log 7\right) \\ &\leq \tilde{G}\left(2\sqrt{\frac{7g}{8}} + \frac{7g}{8}\log 7\right) \\ &\leq \tilde{G}\left(2\sqrt{\frac{7}{8}} + \frac{7}{8}\log 7\right)\sqrt{g} \end{aligned}$$

If we define $\kappa = \left(2\sqrt{\frac{7}{8}} + \frac{7}{8}\log 7\right)$, then we have:

$$H_L \leq \tilde{G}^L \kappa \sqrt{g}$$

To obtain a lower bound, we recall that we assume independent errors. From Figure 3 we can see that errors in the recovery process are independent of errors in different logical bits. After each gate at level $L > 0$, we do error-recovery, and this is where the entropy is removed by means of bit resets. If there are E gates in the recovery process, we know that:

$$\begin{aligned} H_L &\geq 3EH_{L-1} \\ &= (3E)^{L-1}H_1 \end{aligned}$$

We note that every gate touches at least one ancillary bit. With probability g the physical gate fails. When the gate fails, that bit will be flipped with probability $1/2$; thus, the entropy of all the ancillary bits is at least:

$$H_1 \geq H(g/2) \geq g$$

So we have:

$$H_L \geq (3E)^{L-1}g$$

Putting both the upper and lower bounds together:

$$g(3E)^{L-1} \leq H_L \leq \tilde{G}^L \kappa \sqrt{g}$$

If we want to have $O(1)$ bits of entropy per gate, by bounding the left side of the above equation, we must have:

$$\begin{aligned} g(3E)^{L-1} &\leq 1 \\ \log g + (L-1)\log 3E &\leq 0 \\ L &\leq \frac{\log\frac{1}{g}}{\log 3E} + 1 \end{aligned}$$

For example, if $g = 10^{-2}$, and $E = 11$, we have $L \leq 2.3$.

We see that the entropy-saving aspect of reversible computing is indeed highly sensitive to error. Both the upper and lower bounds of entropy per gate are exponential in L for fixed g. At the same time, we see that, even if there is some small finite error with $g \ll 1$, the entropic savings relative to irreversible computing may be obtained by using $O(\log\frac{1}{g})$ levels of error correction.

5 Conclusion

We have given a method of producing fault-tolerant reversible circuits. We also considered this problem in which only local communications are allowed, which we believe will be very valuable for quantum computing systems that need to perform some classical processing without having to resort to quantum measurements[19]. We also note that the circuits and threshold values presented here represent an *lower* bound on the threshold for reversible, fault-tolerant logic. There may exist improved schemes which could improve the threshold values; however, the circuits here provide an existence proof.

While we showed how to make fault-tolerant, reversible circuits, we also saw that, when the error rate is near the threshold, there is considerable cost (in bits, gates, and entropy) to the error correction procedure. While it was already known that, in principle, any noisy reversible computer must dissipate entropy[1], our circuits provide a useful upper bound on how much entropy must be released in the computing process with noisy reversible gates.

The authors would like to thank Michael Frank for many helpful comments, and Thomas Szkopek for pointing out an error in an earlier draft.

References

[1] D. Aharonov, M. Ben-Or, R. Impaglazzo, and N. Nisan. Limitations of noisy reversible computation. http://arxiv.org/abs/quant-ph/9611028.

[2] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 6:525–532, 1973.

[3] P. O. Boykin, T. Mor, V. Roychowdhury, F. Vatan, and R. Vrijen. Algorithmic cooling and scalable nmr quantum computers. *Proceedings of the National Academy of Sciences*, 99(6):3388–3393, 2002.

[4] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton. A new quantum ripple-carry addition circuit.

[5] J. M. Fernandez, S. Lloyd, T. Mor, and V. Roychowdhury. Algorithmic cooling of spins: A practicable method for increasing polarization.

[6] E. Fredkin and T. Toffoli. Conservative logic. *Int. J. Theor. Phys.*, 21:219–253, 1982.

[7] P. Gács. Reliable computation with cellular automata. *Journal of Computer and System Sciences*, 32:15–78, 1986.

[8] J. B. Gao, Y. Qi, and J. A. B. Fortes. Bifurcations and fundamental error bounds for fault-tolerant computations. *IEEE Tran. Nanotechnology.* (in press).

[9] J. Han and P. Jonker. A system architecture solution for unreliable nanoelectronic devices. *IEEE Transactions on Nanotechnology*, 1(4):201–208, 2002.

[10] B. Kane. A silicon-based nuclear spin quantum computer. *Nature*, 393:133–137, 1998.

[11] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.

[12] S. Lloyd. A potentially realizable quantum computer. *Science*, 261:1569–1571, Sept. 1993.

[13] K. Nikolić, A. Sadek, and M. Forshaw. Fault-tolerant techniques for nanocomputers. *Nanotechnology*, 13(3):357–362, 2002.

[14] J. Preskill. Reliable quantum computers. *Proceedings of the Royal Society*, 454(1969):385–410, 1998.

[15] L. J. Schulman and U. V. Vazirani. Molecular scale heat engines and scalable quantum computation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. ACM Press, 1999.

[16] P. Shor. Fault-tolerant quantum computation. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*. IEEE Press, 1996.

[17] T. Toffoli. Reversible computing. In *Automata, Languages and Programming*, pages 632–644. Springer-Verlag, 1980.

[18] J. von Neumann. Probabilistic logics. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 329–378. Princeton University Press, 1956.

[19] R. Vrijen, E. Yablonovitch, K. Wang, H. W. Jiang, A. Balandin, V. Roychowdhury, T. Mor, and D. DiVincenzo. Electron-spin-resonance transistors for quantum computing in silicon-germanium heterostructures. *Phys. Rev. A*, 62:012306, 2000.