Notes on reversible computation

Warren D. Smith
wds@research.NJ.NEC.COM

date = September 29, 1998

Abstract — **Erasing a bit requires the dissipation of $k_B T \ln 2$ worth of energy, if T is the temperature of the heat bath surrounding one's computer and $k_B \approx 1.381 \times 10^{-23}$ Joules/Kelvin is Boltzmann's constant. Is it possible to approach this limit? Is it possible even to blow past it by simply *refusing* to erase bits?**

Keywords —

CONTENTS

1 INTRODUCTION

1.1 Why erasing a bit costs at least $k_B T \ln 2$ energy

Suppose you have a computer (some physical system) and it contains a bit (2-state system). You "erase" the bit, thus reducing the number of states the computer could have been in, by a factor of 2.

This reduces the entropy of the computer by an additive amount $k_B \ln 2$. (Thermodynamic entropy of a system that could have been in W states, is, for the present purposes, $k_B \ln W$. The scaling factor relating thermodynamic and information theoretic entropy is $k_B \ln 2$ per bit.)

If we assume the "second law of thermodynamics" that the entropy of the entire universe cannot decrease, then the bit erasure must have raised the entropy of the rest of the universe by at least $k_B \ln 2$. If the rest of the universe happened to be a "heat bath" (e.g. large volume containing an ideal gas; or any other large system) at temperature T, then this would necessarily have required raising the energy of that bath by an amount $k_B T \ln 2$.

This is a consequence of the definition of "temperature" of a system $T = \partial E / \partial S$ where S is its entropy and E its energy.

Remark. This definition implies that energy will flow from hot to cold objects, since the reverse would decrease entropy[1].

Remark. If you try to refrigerate your computer to decrease the cost of erasing a bit, that won't work, because the best possible "Carnot efficiency" of a heat pump is T_1/T_2. In other words, if you heat up the refrigerated heat bath by adding energy $k_B T_1 \ln 2$ to it, then your refrigerator will have to get rid of that heat energy by exporting it to the outside world at temperature $T_2 > T_1$. Even if you have a perfect Carnot refrigerator, that will cost you

$$\frac{T_2}{T_1} k_B T_1 \ln 2 = k_B T_2 \ln 2. \tag{1}$$

Remark. m-state "bits" would require $k_B T \ln m$ per erasure.

1.2 Maxwell's demon

"Maxwell's Demon" is a very smart, very small red fellow who controls a tiny door between two containers of gas. Whenever a gas molecule is going to hit the right side of the door, the Demon opens the door. The left container of gas therefore acquires higher pressure than the right container, and the Demon takes advantage of this by generating power with a gas turbine connecting the two containers. The Demon, meanwhile, does not seem to require any energy to open and shut the (massless) door. So: in this scenario, the random thermal motion of the gas molecules is gradually depleted and converted into ordered, easily usable, energy. This contradicts the laws of thermodynamics?!

Some people concluded from this paradox that the laws of thermodynamics need not hold in the presence of very

[1] Also, energy will flow from objects at more negative temperature, to objects of positive, or less negative, temperature.

small intelligent beings. Others thought that the act of measurement (involved in realizing that a gas molecule is approaching on one side and not the other) would somehow consume power and save us.

It took until the early 19980s before the problem was finally resolved by Bennett [5] [18] [11]. Bennett followed the lead of L.Szilard [25] by considering an ultrasimplified Maxwell Demon scenario in which there was only one gas molecule. To show that the "measurement" (i.e. deciding which container the molecule was in) did not necessarily consume energy, Bennett devised a classical measurement device which plainly consumed an arbitrarily small amount of energy because, from the point of view of the gas molecule, it was reversible (i.e. after the measurement, the gas molecule sees exactly the same shape gas container as before). The only energy consuming step (and also the only entropy exporting step) in the entire Maxwell Demon engine cycle, in Bennett's analysis, was the step in which the Demon erased its own mind!

Specifically: a complete cycle of Bennett's version of the Demonic engine is as follows.

1. Demon uses Bennett's measurement device to find out which container the gas molecule is in. Cost 0.
2. Demon contracts empty container to zero volume with piston. Cost 0 (massless piston).
3. Demon slides open door between containers. Cost 0 (massless door).
4. Demon extracts energy from piston as empty container expands back to full volume. The gas molecule slows down but it is restored to expected energy $3k_BT/2$ by heat leaking in from the external world at temperature T. The expansion of the accessible volume by a factor of 2 causes our 1-molecule gas to gain $\Delta S = k_B \ln 2$ entropy. The total amount of work extracted by the Demon's heat engine (i.e., piston) was therefore at most $T\Delta S$, i.e. $\leq k_BT \ln 2$.
5. Demon closes door. Cost 0. The gas molecule is now in a random one of two containers, we do not know which.
6. Demon must therefore *erase the bit in its mind* representing its knowledge of where the gas molecule is. Energy Cost: $k_BT \ln 2$.
7. Back to step 1!

1.3 Reversible computers – the dream

The bit-erasure principle was popularized by R.Landauer [17] [10] but may have been known to earlier authors. This led those investigating fundamental physical limits on power consumption to ask:

1. **Logical question** Can we avoid erasing bits entirely in the course of arbitrary computation?
2. **Physical/engineering question** And if so, will the laws of physics really allow us to compute without consuming power?

1.4 Where digital technology stands in the year 1998

As of 1998, here are some claimed performance levels of CMOS (complementary metal oxide semiconductor) VLSI digital logic.

- NEC economical 0.35 micron line width CMOS gate arrays (μPD65958) with up to 1.6×10^6 gates and 708 IO pads per chip. $9.4 times 10^{-11}$ second gate propagation delays. Consumes 1.8×10^{-13} Joules per gate toggling operation.
- LSI logic 0.13 micron line width CMOS "G12" gate arrays with up to 2.6×10^7 gates on a 2cm-square chip. With 1.8 volt power supply, achieves 2.4×10^{-11} second switching times and consumes 1.15×10^{-14} Joules per gate toggle. This may be reduced to 7.1×10^{-15} Joules per gate toggle in a version with power supply voltage of 1 volt.
- Mitsubishi 0.25 micron line width CMOS gate arrays with 3.5×10^6 gates per chip. With a 2.5 volt power supply, runs with gate delays of 8.5×10^{-11} second switching times and consumes 9×10^{-14} Joules per gate toggle.

The LSI logic number is about $2.8 \times 10^5 k_BT$, assuming $T = 300$ Kelvin.

Thus, energy consumption per operation in presently available technologies is still far above Landauer's limit of $k_BT \ln 2 \approx 0.69 k_BT$ per bit-erasing operation.

Exceeding k_BT is not necessarily a bad thing – since it allows these operations to be extremely reliable. But only $100k_BT$ would suffice to reduce error probabilities to e^{-100} (i.e. below 10^{-43}) per operation.

2 Reversible computers can emulate irreversible ones

Definition. A "(locally) reversible Turing machine" is a Turing machine whose previous tape and head state may be uniquely deduced from the current state, and indeed a different "time reversed" Turing machine exists which would "undo" the first machine's computation by going backwards starting from its final state.

Note, most Turing machines are not reversible since they could overwrite a tape square, or perform some many-to-1 transition, and then there would be no way to recover.

It was first pointed out by LeCerf, but did not "stay discovered" until it was rediscovered by Bennett [6], that a reversible Turing machine can emulate an ordinary Turing machine, although a considerable price apparently must be paid in terms of slowdown and memory consumption.

Indeed, Bennett observed that there is a simple way to convert any computer program into a reversible version. We assume all variables are initially zero (for a Turing machine, all tape squares after some point are initially blank). At each point in the program where the computer overwrites a variable

```
var = something;
```

you substitute

```
stack[stack_index] = var;
increment stack_index;
var = something;
```

and then the reversed-time program would have at this point

```
decrement stack_index;
var = stack[stack_index];
```

Thus, any program is converted into a reversible program, and on a RAM, the slowdown is only by a constant factor. *But* we have to store an unboundedly growing number of "garbage bits" (here stored in an unboundedly growing "stack"), whose number could grow as fast as linearly with time.

Objection: It seems ridiculous that by actually doing the same computation plus *more* work, we could save power. Also, don't we have to erase those garbage bits someday, so the savings are illusory?

Response: After the computation is complete, we copy the output bits into an extra memory to preserve them, and then we run the program in reverse. This gets rid of all the garbage bits and brings the computer back to its initial state. The *only* irreversible steps were the copying of the output bits (and similarly reading the input bits at the beginning)!

One can also devise a "universal reversible Turing machine" which has as primitive operations things like "swap two adjacent tape squares" and program Bennett's emulation procedure for it. The slowdown may be by a linear factor, i.e. a program running for time T on an irreversible Turing machine may require reversible time of order T^2.

Thus, at least as far as the Landauer bit-erasure principle and Turing machines are concerned, it is possible to do unboundedly huge computations with bounded energy consumption.

2.1 Reversal-friendly algorithms exist

There may be a large class of "nice" algorithms which generate very little "garbage" and thus may be made reversible with much less than the usual overhead. For example, there are entire computer languages (so-called "functional languages," e.g. "Haskell") which are supposed to be useful despite the fact that they do not allow overwriting a variable. For another example, consider the "summation problem" of computing

$$\sum_{j=1}^{N} f(j). \tag{2}$$

This may be solved by the following algorithm:

1. $j = 1$; $S = 0$;
2. Compute $x = f(j)$.
3. Add x to S.
4. Uncompute $x = f(j)$.
5. Increment j.
6. If $j \le N$ then go back to step 2.
7. Output S.

There is a sense in which all the steps (except for the first and last) of this algorithm are reversible. That is evidenced by the following reversed algorithm

1. Initially $j = N + 1$ and $S = \sum_{j=1}^{N} f(j)$.
2. Compute $x = f(j)$.
3. Subtract x from S.
4. Uncompute $x = f(j)$.
5. Decrement j.
6. If $j > 0$ then go back to step 2.

In fact, a reversible adder/subtractor "circuit" may be built out of Toffoli gates (discussed later) and a reversible integer comparator (to detect if $a > b$, and then proceed, only costing energy in the exceptional case when $a \le b$) too... so it seems that the summation problem is a particularly nice algorithm and will generate only $O(1)$ words of garbage no matter how many terms are to be summed.

2.2 Garbage compaction

Still, what about non-nice algorithms? Have we no choice but to generate enoumous amounts of garbage (with current 500 MHZ computer speeds, that could be 5×10^8 words of garbage per second filling up your memory *fast*!). It turns out, as was realized by Bennett [8] and perfected Levine & Sherman [19] that it is possible to trade off time versus space (i.e. garbage bits) when emulating irreversible computations with reversible computers. We can store much less garbage if we are willing to spend only a little more time.

Example. We'll now show how to simulate a time-T run of an irreversible RAM with a reversible one running for time $O(T^{\log_2 3})$ while only generating $O(\log T)$ words of garbage. The picture below shows the particular case $T = 8$.

Levine and Sherman showed that the same idea, but with "segment" boundaries defined by the amount of garbage they generate rather than runtime, and allowing "branching factors" other than 2 in the conceptual "tree," would yield claim I in the following result. (Claim II is really just a rewriting of claim I, and claim III is just the naive emulation.)

Theorem 1 (Levine-Sherman reversible emulation time-garbage tradeoff) *Let $\epsilon > 0$.*
I: Any program running on an irreversible RAM for T steps and consuming S words of storage, may be emulated by a

program running on an reversible RAM for $O(T^{1+\epsilon}S^{-\epsilon})$ steps and utilizing $O(S + S\ln\frac{T}{S}]\epsilon 2^{1/\epsilon})$ words of storage.
II: If $T \geq 2S$, the emulation may be performed in time $O(T)$ and space $O(T^{\epsilon}S^{1-\epsilon})$ for any $\epsilon \geq 1.2/\ln\frac{T}{S}$.
III: If $T \leq 2S$, the emulation may be performed in time $O(T)$ and space $O(S)$.

Notice that the number of "excess garbage" words grows at worst only logarithmically, not linearly, with T in claim I... although watch out for the nasty behavior of that $\epsilon 2^{1/\epsilon}$ constant as $\epsilon \to 0$.

Stage	Action	Segments with stored final states							
0	Start								
1	Do segment 1	1							
2	Do segment 2	1	2						
3	Undo segment 1		2						
4	Do segment 3		2	3					
5	Do segment 4		2	3	4				
6	Undo segment 3		2		4				
7	(Re)do seg. 1	1	2		4				
8	Undo segment 2	1			4				
9	Undo segment 1				4				
10	Do segment 5				4	5			
11	Do segment 6				4	5	6		
12	Undo segment 5				4		6		
13	Do segment 7				4		6	7	
14	Do segment 8				4		6	7	8
15	Undo segment 7				4		6		8
16	Do segment 5				4	5	6		8
17	Undo segment 6				4	5			8
18	Undo segment 5				4				8
19	Do segment 1	1			4				8
20	Do segment 2	1	2	3	4				8
21	Undo segment 1		2						8
22	Do segment 3		2						8
23	Undo segment 4		2						8
24	Undo segment 3		2						8
25	Do segment 1	1	2						8
26	Undo segment 2	1							8
27	Undo segment 1								8

Bennett's emulation method emulates 8-step computation in 27 reversible steps. Stored garbage: 4.

It is *not known* that any of these emulation algorithms and time-space tradeoff bounds are optimal.

3 Are zero power reversible computers allowed by the laws of physics?

3.1 Bennett's "Brownian" computer

There is no doubt that something like the following "enzymatic" computational device indeed will perform arbitrary computations with zero power. However, an N-step reversible program will typically require time equivalent to order N^2 steps to run to completion on a Brownian computer!

The idea is simple. Imagine your computer's "memory tape" is stored in some chemical, e.g. DNA. The fundamental set of reversible operations on that data are performed by enzyme-catalyzed chemical reactions. Enyzme catalysis typically enhances reaction speeds by huge factors (millions) versus uncatalyzed reactions, so assume (or hope) that the undesired side reactions will occur at negligible rates, whereas the desired reactions corresponding to permitted Turing machine steps, will happen frequently.

Now the Turing machine state after t steps will then be in a chemical equilibrium with the Turing machine state after $t+1$ steps. Assuming these two states have equal energies, this will be a 50-50 equilibrium. The DNA tape will thus perform a random walk forward and backward in time. After order N^2 steps, it becomes reasonably likely that the random walk has managed to go forward in time for N steps.

If we assume that the transition to the final "halt" state of the computation involves an exothermic chemical reaction (dissipating order k_BT) then we pay order k_BT energy cost to complete the entire computation, no matter how many steps it required.

For many purposes, this quadratic slowdown is going to be unacceptable. However, we can bias the random walk by making every chemical step a little exothermic. We call this a "Thermodynamically biased Universal Reversible Turing Machine (TURTM)."

Specifically, suppose energy E is dissipated in a forward step. Then the probability p of making a forward step rather than a backward one obeys

$$\frac{p}{1-p} = \exp(\frac{2E}{k_BT}). \tag{3}$$

In this case, it will become exponentially likely (for c large) that the calculation has been completed after t steps where

$$(2p-1)t > N + c\sqrt{(1-p)pt}. \tag{4}$$

Incidentally, it is impossible to expend energy less than E on some steps and more than E on others in some clever way designed to still use average energy E per step but to increase the rate of forward progress. This is because the expected forward progress per step is

$$2p - 1 = \tanh(\frac{2E}{k_BT}). \tag{5}$$

which is a *concave-down* function of E, $E \geq 0$. Hence

Theorem 2 (Biased random walk) *There is a tradeoff between energy-usage and speed. Expending energy of order k_BTN will assure completion of an N-step TURTM calculation in order N basic time units (unless one is exponentially unlucky). On the other hand, one may expend a factor of N^{β} less energy, $0 \leq \beta \leq 1$, in which case the calculation will almost certainly take about N^{β} times longer.*

Incidentally, the DNA copying and checking machinery inside living cells appears to *be* a (non-universal) TRTM which dissipates 20 to 100 units of k_BT or so per step (the energy is supplied in the form of ATP, GTP, CTP and TTP nucleotide triphosphates, four kinds of chemical "fuels").

3.2 Ballistic reversible computers: classical models

The trouble with Brownian computers is that they randomly walk forwards and backwards, slowing down the computation. The next class of reversible computer designs, called "ballistic" computers, always walk unidirectionally and will run N-step reversible programs in $O(N)$ steps.

But known ballistic computers are very vulnerable to noise and errors in the initial state.

The classical reversible ballistic computers of Fredkin, Toffoli and Margolus are surveyed in the accompanying note [22], so we won't go into much detail about them here. Toffoli observed that the "Toffoli controlled crossover gate," a 3-input, 3-output box whose first input bit a_{in}, which is simply transmitted unchanged to the first output a_{out}, controls whether the other two inputs were transmitted unchanged, or exchanged, i.e. whether $\{b_{\text{out}} = b_{\text{in}},\ c_{\text{out}} = c_{\text{in}}\}$ or whether $\{b_{\text{out}} = c_{\text{in}},\ c_{\text{out}} = b_{\text{in}}\}$ is "reversible," and that circuits may be built out of these boxes which will emulate any Boolean circuit. Fredkin [13] devised a beautifully simple toy model of perfect rigid billiard balls bouncing around on a planar table equipped with fixed "mirrors" and showed that this model could perform an arbitrary sequence of Boolean operations – in particular they could implement a Toffoli gate. Margolus [21] showed how to turn Fredkin's model into a simple "reversible 2D cellular automaton (CA)" also capable of performing an arbitrary sequence of Boolean operations. It had been apparent immediately after Bennett and LeCerf's discovery of reversible universal Turing machines that 1D reversible CAs existed, with universal computational power (because you can easily device a reversible CA transition rule to emulate any reversible Turing machine transition function). But Margolus's 2D automaton seemed easier to understand than these constructions.

Critics then pointed out that Fredkin's Billiard balls were unstable to small perturbations – small errors in their initial positions and velocities would be magnified exponentially with time. This error growth would either destroy the computer (if uncorrected), or if corrected, the correction process could be interpreted as entropy exportation and hence would be subject to a bound proportional to k_BT. Indeed, the bound turned out to be almost twice as large as $k_BT \ln 2$ per bounce, so that the Fredkin reversible computer appeared in reality to need *more* power than a conventional irreversible implementation of the same sequence of Boolean operations!

However, as I [22] pointed out, it is possible to modify the billiard ball model so that small errors will *not* grow exponentially with time, and then the objections of the critics are mooted.

This made it appear that it *is* possible to do near-reversible computation in classical mechanics, consuming Qk_BT per logical operation in the analogous conventional circuit, where the "technological factor" Q may be made very small. The only significant difference between the reversible and conventional versions is the large number of "garbage" bits which need to be preserved in the former. This requirement could reduce circuit density, and hence reduce speed, in the worst case by provably unboundedly large factors.

3.3 Ballistic reversible computers: Quantum models

Now, let us consider quantum mechanics. Can we devise toy models of reversible computers, in quantum mechanics? The answer, as was pointed out by Feynman [12] and Benioff [4] is yes.

For concreteness, let us consider a computation-universal CA. This is a grid (either 1D or 2D, or even 3D), each vertex of which can be in some fixed finite set of logical states. Not surprisingly, we will make these logical states correspond to a fixed finite set of quantum mechanical states. Now at each time step, the states are supposed to mutate according to some fixed transition function (which says what a state mutates to as a function of the old state and the old states of its neighbors). This transition function is supposed to be reversible in the sense that the reverse-time evolution is described by another such cellular automaton rule. We are going to show how to write down a "local" quantum Hamiltonian function such that the logical time-evolution demanded by the CA rule, is going to be forced by the quantum mechanical time evolution demanded by Schrödinger's equation.

Perhaps the best way to illustrate the technique is by example. Consider the Toffoli gate. The operation of this gate is determined by the "truth table"

```
inputs   outputs
abc      abc
000=0    000=0
001=1    001=1
010=2    010=2
011=3    011=3
100=4    100=4
101=5    110=6
110=6    101=5
111=7    111=7.
```

Thus, the Toffoli gate abstractly performs a permutation on an 8-element state set. The permutation matrix (which, of course, is a unitary matrix, since all permutations are unitary) is

$$U = \begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1 \end{pmatrix}. \tag{6}$$

Now, let our Hamiltonian operator for this 8-state system (hence, the Hamiltonian "operator" in this case is just an 8×8 Hermitan matrix) be called H. Schrödinger time evolution demands that the wavefunction Ψ (which, for our particular 8-state space, is just an 8-element complex vector with unit norm) evolve with time according to

$$\Psi(t) = e^{-itH/\hbar}\Psi(0). \tag{7}$$

Therefore, if

$$H = \frac{\pi\hbar}{2}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & +1 & 0 \\ 0 & 0 & 0 & 0 & 0 & +1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{8}$$

then the Schrödinger time evolution from $t = 0$ to $t = 1$ will force the Toffoli gate time evolution. We may similarly (by taking a matrix logarithm scaled by $i\hbar$) devise Hamiltonians corresponding to any permutation matrix.

Thus, at least on paper, a suitable time-independent Hamiltonian exists which will cause any RCA to do its thing at 1 RCA step per second, with no power needed. Whether any such finite dimensional Hamiltonian matrix is actually realizable as a physical system, is an issue the authors in this area have preferred to evade... but surely it may be approximated.

Now, really, if we were to build such a physical system, there might be "manufacturing errors" – causing the Hamiltonian matrix entries to be slightly different from the designed values (and possibly these errors would have some fixed random spatial distribution along the CA grid) and "thermal noise" – causing the Hamiltonian to be perturbed by random time-dependent thermal terms.

How would these errors affect the time evolution of the quantum system? Well, mathematically, the situation is fairly similar to "Anderson localization" of unbound electrons in a potential which includes a spatially random term (this corresponds to the manufacturing errors, with the direction of electron conduction corresponding to CA time) and "electron phonon scattering" (this corresponds to the thermal errors, again with the direction of electron conduction corresponding to CA time).

In these fairly well understood physical phenomena, electrons, which would like to propagate smoothly along as plane waves periodic in space and time, are unfortunately scattered by nonuniformities in the crystal lattice (due either to thermal noise or to random crystal defects and impurities), reducing conductivity to a finite value, and in some cases even converting the material into an insulator.

Similarly, our RCA would like to smoothly time-evolve forward in time at a rate of 1 CA step per second, but "scattering" off manufacturing and thermal errors causes nonzero "reflection amplitudes" creating waves going *backward* in time and depleting the forward waves. Ultimately, waves propagate both forward and backward, randomly reflecting every so often, and we get something more like a Brownian computer than a ballistic computer, with consequent enormous slowdown: squaring of the number of computational steps required due to "diffusion" in computational "time," rather than constant forward progress. If the analogy to perfect Anderson localization [1] [14] holds, in fact the waves will end up "localized" and be unable even to "diffuse," leading to a total computer shutdown.

I claim there has not yet been a decent treatment of these analogies and their consequences.

It has been argued [32] that errors in Ψ due to manufacturing errors in this type of quantum CA model will grow only quadratically with time in the sense that

$$\begin{aligned} & 1 - \langle \Psi_{\text{correct}}(t) | \Psi_{\text{errors}}(t) \rangle \\ &= 1 - \langle \Psi(0) | e^{iHt} e^{-i(H+\delta)t} | \Psi(0) \rangle \\ &\leq \| \delta H - H\delta \| t^2 \end{aligned} \tag{9}$$

– this may be shown as a consequence of error bounds in the "Baker Campbell Haussdorf formula" for matrix exponentials [23] and clarifies some vagueness in [32]. That is, the probability of getting the wrong logical state in a measurement, grows quadratically with time. Thermal errors presumably also grow quadratically or more slowly.

Furthermore, errors in $\Psi(0)$ will not be amplified at all by Schrödinger time evolution, since that evolution is unitary.

All this non-exponential error growth (analogous to achieving "zero Lyapunov exponent" in the classical models) *suggests*, but does not prove[2] that some error correction scheme might be available, which would dissipate arbitrarily less than $k_B T$ energy per CA step per CA node, and allow RCA computation to proceed despite the presence of these error mechanisms.

4 Reversible CMOS – a real world application of these ideas in a different context

As we've mentioned, present day CMOS logic dissipates heat. The remarkable speed increase and area shrinkage of digital logic over the years has led to that heat dissipation becoming a serious problem. For example, the DEC alpha 21164 (64-bit) processor chip (9.3×10^6 transistors), running at 300 MHz on a 2.75 volt supply, dissipates 40 watts from chip area of 314mm^2. (Toasty!) Intel Pentium pro processors at 250MHz have similar power needs.

As we've seen, CMOS logic at present operates far above the $k_B T$ limit, so one might imagine that there is nothing useful the theory of reversible computing can say about it. But it turns out [16] there is a remarkable mathematical analogy between

```
In CMOS logic                   In ‘‘thermal logic’’
-------------                   --------------------
voltage V                       temperature T
Charge Q                        heat energy Q
```

[2] Zurek [32] has a few vague ideas in this direction.

```
Current I=dQ/dt                      heat flow J=dQ/dt
Capacitance C=dQ/dV                  heat capacity C=dQ/dt
Conductance G=1/R=I/V[drop]          K=J/T[drop]
energy differential dE=VdQ           entropy diff'l  dS=dQ/T
```

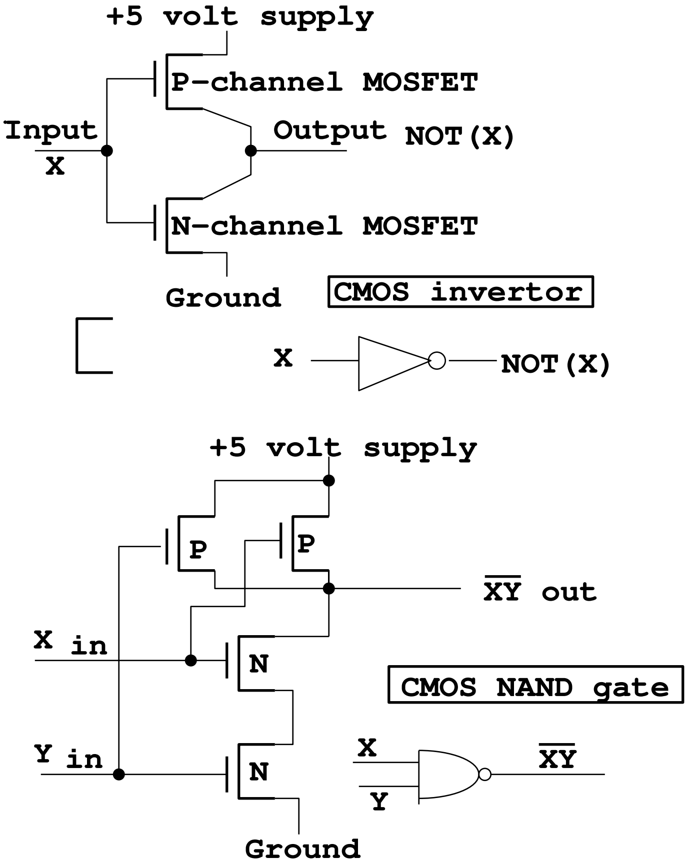

Figure 1: CMOS digital logic

In Koller et al.'s [16] edifying (albeit presumably useless) "thermal logic," "wires" are good thermal conductors while "insulators" don't transmit heat. The "DC power supply" is a heat bath at some temperature hotter than another heat bath called "ground." Thermal switches, analogous to electrical MOSFETs, work as follows. One object (the "gate") expands becuase it is heated, causing the "source" and "drain" objects to come into thermal contact. One can then construct thermal logic circuits (hot means Boolean 1, cold means boolean 0) analogous to CMOS logic circuits.

Thermal logic circuits will be reversible if they never export any entropy to the outside universe. This happens if, whenever two objects are brought into thermal contact, their temperatures are *equal*. Analogously, CMOS logic circuits will be reversible if, whenever two electrical wires (which have some capacitances) are switched into electrical contact, their voltages are the same. Otherwise, a current will flow through the resistive switch until the voltages on the two capacitors are equal, dissipating energy. You dissipate energy $CV^2/2$ if you charge a capacitor C to a voltage V through a resistor.

In conventional CMOS logic (figure 1) such energy dissipation happens during every gate toggling. For example in the CMOS NAND gate shown, switching the gate's output will charge that output (regarded as a capacitor) through a (resistive) MOSFET switch, dissipating heat. Once the capacitor is fully charged, however, current ceases to flow, and thus CMOS logic, when it is not switching state, uses (almost) no power.

The idea of "reversible" or "adiabatic" CMOS logic [2] [3] [15] [16] [24] [27] [28] [29] [30] [31] is to *avoid* turning any switches on that connect capacitors at unequal potentials. This may be accomplished by using AC power supply rails and performing the appropriate switching at precisely the right moment when the AC rail voltage is equal to the voltage of whatever you are connecting it to. For example, imagine a logical "pipeline" of logic operations one after the other.

Each output of a pipeline stage, instead of being connected directly to the input of the next stage, will be connected *through two switches* each made of a N-channel MOSFET and P-channel OSFET in parallel. (The long wire segment between the two switches is a capacitor which stores charge. The wire segments outside this region are made as short as possible so their capacitance is small.) Initially everything is at zero voltage.

To perform a logic stage, we wait until the power rails are both at zero voltage, then switch on first switches. As the power supplies *slowly* ramp up to positive and negative voltages, the outputs of each gate ramp toward their correct values. (If the ramping requires time t, the power dissipation to charge capacitance C to voltage V through a resistor R, and then discharge it again, is RC^2V^2/t, which we may make small by making t large, as opposed to instantaneous [well, with characteristic time $t \approx RC$] switching.) We could at this point trivially reverse the computation by ramping the power supplies back down again. But we don't; at the peak of the power supply sine wave, we turn the first switches off. Charge is now trapped in the wire segment between the two switches. By turning on the second switch, this output is transmitted to the input of the next stage in the pipeline.

Now, if we were to do this again, some of the wires would *not* start at zero voltage, and thus the next step would dissipate power. This is where reversibility comes in. We want first to reverse the computation to get rid of the charge on the long wires as we ramp power supplies back down. That requires the ability to compute the inverse of each logical transformation so that we can figure out which rail to connect the wire to, to carry it smoothly back to zero voltage. We do so. The wires are now once again at zero voltage, and we are ready to ramp the supplies again to perform another pipeline stage.

Note: each pipeline of logical stages comes with a reverse direction pipeline performing inverse logical stages, so we need at least twice the circuit complexity.

At the left and right ends of the pipeline we have no inverse information available and have to pay the full penalty (charging cost) of all wires we mis-predict; but this is still enough to reduce the energy consumption of an N-stage pipeline by a factor of order N. In fact in principle we could build our CMOS circuit entirely from Toffoli gates and regard each gate as a "pipeline stage," so that these losses would occur only at the inputs and outputs of the whole boolean function. In practice a "hybrid" approach is probably better where just some of the circuit stages are made adiabatic.

Ramping the power supplies may be accomplished by an inductive-capacitive "TANK" oscillator circuit, which dissipates no energy in the absence of resistance – and since all the clock/power signals flow purely through wires, not (much more resistive) MOSFETs (unless the voltage acorss them is zero, when it is OK) that criterion is reasonably satisfied.

Similarly, in the mental model of "thermal logic," we could use a "thermal oscillator" to create thermal oscillations, and the analogous thermal logic circuit would then become another amusing mental model implementing reversible and almost-dissipationless (assuming perfect thermal conductors and insulators) computing. One could in principle make the required "thermal oscillator" using a massive piston and a spring; the piston oscillates mechanically, compressing or rarifying a thermally isolated pocket of gas, whose temperature changes appropriately.

References

[1] P.W.Anderson: Absence of diffusion in certain random lattices, Phsyical Review 109,5 (1958) 1492-1505.

[2] W.C.Athas, N.Tzartzanis, L."J."Svensson, L.Peterson: A low power microprocessor based on resonant energy, IEEE J. Solid State Circ. 32,11 (1997) 1693-1701.

[3] W.C.Athas, L."J."Svensson: Reversible logic issues in Adiabatic CMOS, PhysComp 94, 111-118.

P.W.Anderson: Absence of diffusion in certain random lattices,

[4] P.Benioff: Quantum mechanical models of Turing machines which dissipate no energy, Phys. Rev. Lett. 48,23 (1982) 1581-1584.

[5] C.H.Bennett: Demons, Engines, and the Second Law, Scientific American (Nov. 1987) 108-116.

[6] C.H.Bennett: Logical reversibility of computation, IBM J. Research & Devel. 17 (1973) 525-532.

[7] C.H.Bennett: The thermodynamics of computation - a review, Int'l J. Theor. Phys. 21 (1982) 905-940.

[8] C.H.Bennett: Time/space tradeoffs for reversible computation, SIAM J.Comput. 18,4 (1989) 766-776.

[9] Charles Bennett: Notes on the History of Reversible Computation, IBM J.Research and Development 32, 1 (January 1988) 16-23.

[10] Charles H. Bennett and Rolf Landauer: The Fundamental Limits of Computation, Scientific American (July 1985) 48-56 (38-53 in some versions).

[11] P.N.Fahn: Entropy cost of information, PhysComp 94, 217-226.

[12] R.P.Feynman: Feynman lectures on computation, (ed. Anthony J.G. Hey & Robin W. Allen) Addison-Wesley 1996. An article on quantum models of reversible computation is reprinted here which originally appeared in Optics News (Feb. 1985) 11-20.

[13] E.Fredkin & T.Toffoli: Conservative logic, Int'l J. Theor. Phys. 21 (1982) 219-253.

[14] Jürg Fröhlich & Thomas Spencer: Absence of diffusion in the Anderson tight binding model for large disorder or low energy, Commun. Math. Phys. 88 (1983) 151-184.

[15] J.G.Koller & W.C.Athas: Adiabatic switching, low energy computing, and the physics of erasing and storing information, PhysComp 92, 267-270.

[16] J.G.Koller, W.C.Athas, L."J."Svenson: Thermal logic circuits, PhysComp 94, 119-127.

[17] R.Landauer: Irreversibility and heat generation in the computing process, IBM J. Res. & Devel. 5 (1961) 183-191.

[18] Harvey Leff and Andrew Rex (editors): Maxwell's Demon: Entropy, Information, Computing, Princeton University Press, 1990.

[19] R.Y.Levine & A.T.Sherman: A note on Bennett's time space tradeoff for reversible computation, SIAM J. Comput. 19,4 (1990) 673-677.

[20] K.K.Likharev: Classical and quantum limitations on energy consumption in computation, Int. J. Theor. Phys. 21 (1982) 311-326.

[21] N.Margolus: Physics-like models of computations, Physica 10D (1984) 81-95.

[22] W.D.Smith: Classical reversible computation with zero Lyapunov exponent, 1998.

[23] Masuo Suzuki: Decomposition formulas of exponential operators and Lie exponentials with some applications to quantum mechanics and statistical physics, J.Math.Phys. 26,4 (1985) 601-612.

[24] L."J."Svensson & J.G.Koller: Driving a capacitive load without dissipating fCV^2, Low power electronics (1994) 100-101.

[25] L.Szilard: On the decrease of entrpy in a thermodynamic system by the intervaention of intelligent beings, Zeitscrift für Physik 53 (1929) 840-856.

[26] T.Toffoli: Comment on 'dissipation in computation', Phys. Rev. Lett. 53, 12 (1984) 1204.

[27] N.Tzartzanis & W.C.Athas: Design and analysis of low power energy recovery adder, VLSI 1995 proceedings (5th great lakes sympos.) 66-69.

[28] N.Tzartzanis & W.C.Athas: Energy recovery for the design of high speed low power staic RAMs, IEEE Int'l sympos. Low power electronics and design (1996) 55-60.

[29] S.G.Younis & T.F.Knight Jr.: Non-dissipative rail drivers for adiabatic circuits, Adv. Res. in VLSI (1995) 404-414.

[30] S.G.Younis & T.F.Knight Jr.: Asymptotically Zero Energy Split-Level Charge Recovery Logic, International Workshop on Low Power Design (1994) 177-182.

[31] S.G.Younis & T.F.Knight Jr.: Practical implementation of charge recovery zero power CMOS, pp. 234-250 in Proc. 1993 sympos. Integr. syst., MIT press 1993.

[32] W.H.Zurek: Reversibility and stability of information processing systems, Phys. Rev. Lett. 53,4 (1984) 391-394.