A reprint from

American Scientist
the magazine of Sigma Xi, The Scientific Research Society

Reverse Engineering

Brian Hayes

Most of the machines we encounter in everyday life are one-way devices. Kitchen appliances turn bread into toast and cabbage into cole slaw, but they cannot perform the opposite transformations. Even machines that claim to be reversible adopt a very shallow notion of what it means to go backwards. The drill in my tool box has settings marked "forward" and "reverse," but no matter which I choose, I cannot undrill a hole. The gearshift in my car also has a position labeled *R*, but when I back up, the engine keeps turning in the same direction; if the car were *truly* reversed, it would suck in pollution through the tailpipe, converting it into gasoline and air.

Computers, too, are mostly irreversible machines. When a program is running, there's no way to turn it around and have it step through the same instructions in the opposite order. Even if that were feasible, the backward-running program would not uncompute an answer. Some of the individual instructions would also have to be reversed, or inverted; for example, undoing addition requires subtraction. For computers of the current generation, such reversals of logic and arithmetic are simply not possible.

In the case of toasters and gasoline engines, full reversibility is too much to ask, but no fundamental law forbids reversible computing. The theoretical possibility of a digital computer that can run forward and backward with equal ease has been recognized for more than 30 years. A few silicon prototypes have been built and shown to work. Still, up to now, reversible computing has been a toy technology, more of interest to theorists than to engineers. Only a small community of devotees believed it would ever be anything more.

backward and forward both run to need may they ,faster run to are computers If

That attitude is changing now. The reason is that reversible computing holds out the promise of dramatically lower power consumption, which is becoming an urgent need. Also, any computer based on quantum technology will necessarily be a reversible machine. As a result, forward-thinking chip designers are thinking backwards too. It's not too soon to ask what it might be like to have a reversible computer on your desk, or to write programs that can run in either direction.

Computational Fuel Economy

Why is power consumption so important, and what does it have to do with reversibility? The chain of reasoning that links these concepts is a fairly long and tangled one, but the individual steps are easy enough to follow.

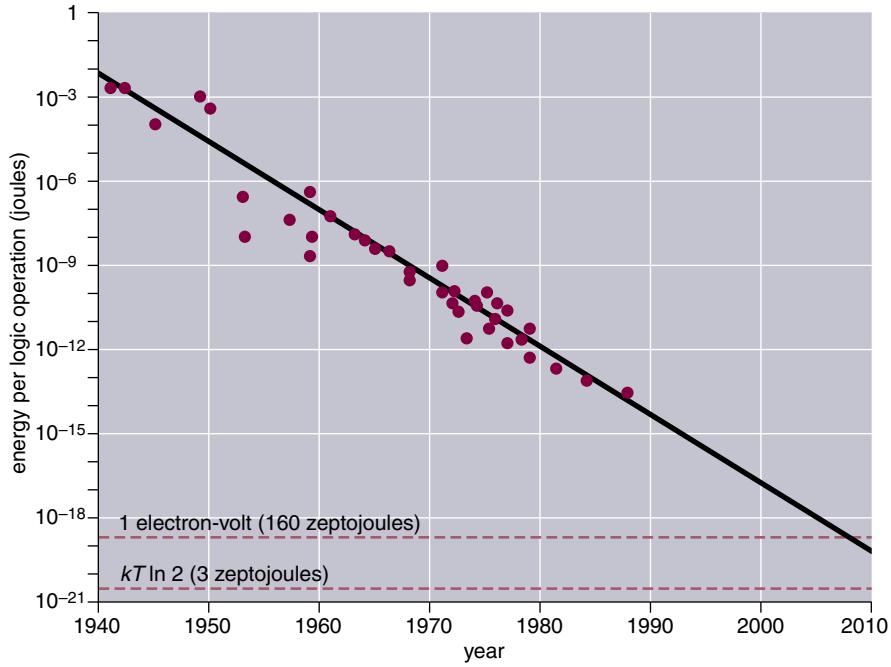
In the past two decades the performance of microprocessors has improved by a factor of 1,000, but the trends that have made computers more powerful have also made them more power-hungry. Some chips dissipate more than 100 watts and require elaborate fans, heat sinks and even liquid cooling. Designers would welcome another thousandfold gain in performance, but they cannot cope with any further increase in power density. Single chips that consume electricity by the kilowatt are just not a practical option.

Where does all the energy go? Much power is lost because neither conductors nor insulators are perfect; electrons meet resistance where they ought to pass unopposed, and they leak through materials where current ought to be blocked. Both of these problems will get more severe as silicon devices continue to shrink. Another energy drain is the need to accelerate and decelerate electric charges as signals move through the circuitry; this cost goes up along with processor speed. And it always takes energy to push electrons "uphill" against a voltage gradient.

Some of the strategies for reducing the energy demands of a computer are much like measures to improve the fuel economy of an automobile. To get better gas mileage, you make a car lighter and aerodynamically sleeker; likewise in digital circuits, you can reduce inertia by using fewer electrons to represent each bit of information, and you can cut resistive losses with better conductors. In the car, you drive slower and more smoothly; in the computer you operate at lower voltage and avoid abrupt swings in voltage. For even greater savings in an automobile, you might try a hybrid design, with a battery or a flywheel to recapture energy invested in acceleration and hill-climbing; the electronic counterpart is an experimental technology called charge recovery.

In the world of chipmaking, some of these energy-conserving measures are already well-established tools, and others are likely to be adopted soon. For example, copper is replacing aluminum in the metal interconnections on some chips to improve conductivity. The voltage levels of on-chip signals have fallen from 5 volts to as little as 1 volt. Further steps of the same general kind may well avert a silicon energy crisis for another decade or two. But

Brian Hayes is Senior Writer for American Scientist. *He is the author of* Infrastructure: A Field Guide to the Industrial Landscape, *just published by W. W. Norton, and a new weblog at the URL: http://bit-player.org. Address: 211 Dacian Avenue, Durham, NC 27701. Internet: bhayes@amsci.org*



The energy cost of computing—measured in joules per logic operation—must fall steadily if computer performance is to continue its exponential growth. The data displayed here were gathered and analyzed by Rolf Landauer in 1988; although the trend in more recent years may not have followed Landauer's projection exactly, it seems clear that energy levels will soon be approaching important landmarks. One electron-volt is the energy of a single electron at a potential of one volt. A thermodynamic limit at about 3 zeptojoules is expressed by the formula $kT \ln 2$; to get below this threshold, computers will have to be operated reversibly.

then what? If the number of operations per second is to increase by a factor of 1,000 without raising power consumption, then the average energy per operation must be reduced to a thousandth of its present value. Is that possible, even in principle? What about a millionth?

Zeptojoules

For a long time it was taken for granted that storing, manipulating and transmitting information would always necessarily dissipate some nonzero quantity of energy. Engineering prowess might lower the energy cost somewhat, but there was a threshold level, a lower limit we could never cross. A device that could compute without loss of energy was seen as the information equivalent of a perpetual-motion machine.

John von Neumann, in a 1949 lecture, set the minimum price of "an elementary act of information" at $kT \ln 2$. In this formula k is Boltzmann's constant, which is the conversion factor for expressing temperature in energy units; its numerical value is 1.4×10^{-23} joules per kelvin. T is the absolute temperature, and ln 2 is the natural logarithm of 2, a number that appears here because it corresponds to one bit of information—the amount of information needed to distinguish between two equally likely alternatives. At room temperature (300 kelvins), $kT \ln 2$ works out to about 3×10^{-21} joule, or 3 zeptojoules. This is a minuscule amount of energy; Ralph C. Merkle of the Georgia Institute of Technology estimates that it is the average kinetic energy of a single air molecule at room temperature.

Von Neumann's pronouncement was based on a thermodynamic argument. Consider a computation that answers a single *yes/no* question, where the two possible outcomes appear equally likely at the outset. Once the question has been settled, we know one bit more than we did beforehand, and so the computational process reduces the uncertainty or entropy of the computing system by one bit. But the second law of thermodynamics says that total entropy can never decrease, and so the reduction inside the computer must be compensated by an entropy increase elsewhere. Specifically, the computer must stir up at least one bit's worth of disorder in its surroundings by expelling an amount of heat equal to $kT \ln 2$. Von Neumann—along with everyone else at the time—assumed that *every* "elementary act of information" has the effect of settling at least one *yes/no* question, and thus it seemed that each step in the computer's operation inevitably dissipates at least three zeptojoules of energy.

Von Neumann's ideas on the thermodynamics of computation were widely accepted but never formally proved. In the early 1960s Rolf Landauer set out to supply such a proof and found that he couldn't. He discovered that only a certain subclass of computational events have an unavoidable three-zeptojoule cost. Ironically, these expensive operations are not those that produce information but rather those that destroy it, such as erasing a bit from a memory cell.

Landauer's work on the cost of forgetting was counterintuitive, and initially it got a frosty reception. Now the idea has been thoroughly assimilated, and it's hard to see what the controversy was all about. Erasing a memory cell amounts to ignoring its present contents—which may in fact be unknown—and resetting the cell to some standardized state (usually 0). Thus an indeterminate bit becomes a fully specified one, and the entropy of the machine is diminished accordingly. For this reason the corresponding amount of heat energy ($kT \ln 2$) has to be rejected into the environment. The consequences are even clearer if you think about erasing the entire memory of a computer, so that the system goes from a random state to a highly ordered one; this is a process of refrigeration, and so it obviously gives off heat.

An Unturing Machine

Although erasing takes energy, Landauer found that not all computer operations are subject to the three-zeptojoule stricture. The simplest counterexample is the logical NOT gate, which inverts the state of a single bit, changing 1 to 0 or 0 to 1. No information is lost in this process; the state of the bit is completely known both before and after. Thus there is no fundamental reason for a NOT gate to be dissipating energy. Present-day implementations of the NOT function may very well produce just as much heat as erasing a memory cell, but that energy loss is not required by the laws of thermodynamics.

A notable property of the NOT function is its *reversibility*. When bit A is transformed into NOT A, all that's needed to undo the action and restore the original state is a second application of the NOT function—(NOT (NOT A)) = A. The erasure of a bit is quite different; it

can't be undone because there's no way of knowing whether the previous state was a 0 or a 1.

Functions such as AND are also logically irreversible. A two-input AND gate produces an output of 1 if and only if both inputs are 1; otherwise the output is 0. Thus if the output happens to be 1, we know that both inputs were also 1, but when the output is 0, we can't distinguish between three alternative pairs of inputs (0 and 1, 1 and 0, 0 and 0). A similar analysis applies to OR. At a higher level, standard arithmetic operations such as +, –, × and ÷ also throw away information and are therefore irreversible. Take the equation 5+3=8; given the left side, it's easy to regenerate the right, but you can't go the other way.

Landauer showed that only the logically irreversible steps in a computation carry an unavoidable energy penalty. If we could compute entirely with reversible operations, there would be no lower limit on energy consumption. We could run a supercomputer on a watch battery.

The question remained: Is it possible to build a fully capable computer out of nothing but logically reversible components? Initially, Landauer himself thought the answer was no. After all, most of the familiar building blocks of computers, such as AND and OR gates, are irreversible devices. Then in 1973 Charles H. Bennett, now of IBM Research, came up with a remarkably direct demonstration that any computation by an irreversible computer can also be done reversibly.

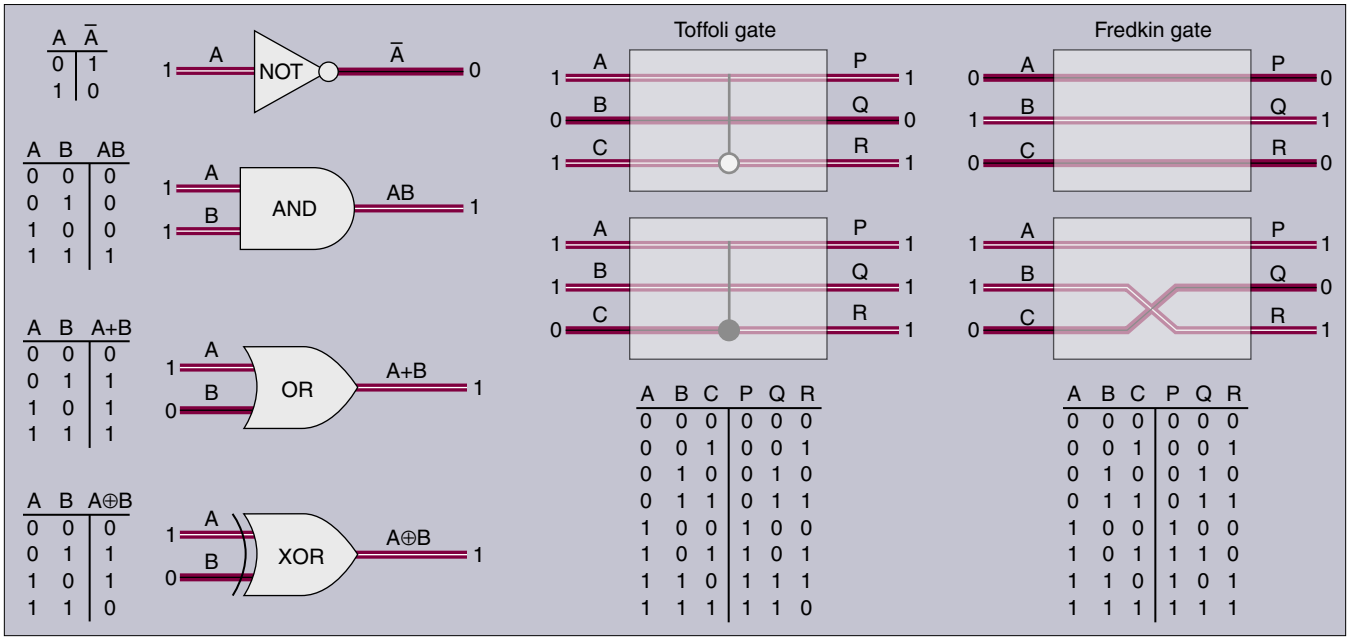
Bennett presented his idea in terms of a Turing machine, the abstract model of a computer that reads, writes and erases symbols on a tape. Erasures make the standard Turing machine irreversible, so Bennett added a second tape, called the history tape, where the machine keeps notes about erased or overwritten data. At the end of a computation, the final answer can be copied onto yet another tape for safekeeping. Then the machine is put in reverse gear, and with the help of the history tape, all the operations are undone, one by one, until the system returns to its initial condition.

The latter half of this operating cycle strikes many people as bizarrely counterproductive. How can you save energy by running the machine for twice as long, regardless of whether it's going forward or backward? Wouldn't it be better just to throw away the extra tape? Experience with ordinary machines in everyday life does not prepare us to answer these questions. When you drive a car 10 miles down the road, you can't recover the fuel you burned by backing up over the same route. But suppose the car had a history tape that kept track of all the molecules involved in the combustion process; then, in principle, you could unburn the gasoline by bringing those molecules back together in the right order. This is not a feasible scheme in automotive engineering, but in the tidier, discrete universe of digital computers, the history-tape trick does work.

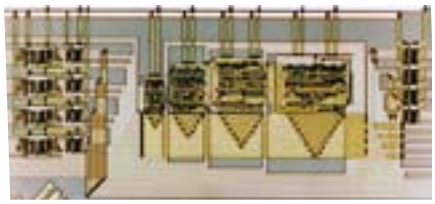
Maybe even the reverse-combustion process isn't quite as outlandish as it seems. Bennett, who began his career as a chemist, also described a biochemical model of a reversible computation. When the enzyme RNA polymerase reads off the sequence of bases in a strand of DNA and assembles a corresponding molecule of RNA, the RNA is conceptually equivalent to the history tape in Bennett's Turing machine. The same enzyme can then work in reverse to disassemble the RNA, base by base—but only for RNA that matches the DNA template. Bennett points out that this orderly reversal of the assembly process is energetically more efficient than disposing of the RNA with enzymes that degrade any such molecule.

Reversible Logic

A Turing machine is fine for reasoning about computers, but it's not an ideal model for building them. Some more-practical components of reversible



Logic gates are the conceptual building blocks of both irreversible and reversible computers. The action of each gate is defined by a truth table, which lists the gate's output for every possible combination of inputs. The most familiar gates, at left, are basic elements of Boolean logic: NOT, AND, OR and XOR. In this group, only NOT is reversible. At right, the Toffoli gate (also known as the controlled-controlled NOT gate) and the Fredkin gate (or controlled swap gate) are elements of a reversible logic. In these gates, every set of inputs corresponds to a unique output; by examining the outputs, you can tell what the input must have been, and thereby reverse the operation of the gate.



This prototype reversible circuit was designed and fabricated by Alexis De Vos and his colleagues at the University of Gent. The circuit is a four-bit adder, implemented with Toffoli gates and related devices. The area shown is a little more than half a millimeter wide. (Photograph courtesy of Alexis De Vos.)

logic were introduced in the 1980s by Edward F. Fredkin and Tommaso Toffoli, who were then working together at MIT. (Fredkin is now at Carnegie Mellon University, Toffoli at Boston University.) The components are logic gates, somewhat like AND and OR gates but designed for reversibility.

In any reversible gate the number of inputs must equal the number of outputs. Moreover, each possible set of inputs must yield a distinct set of outputs. If this were not the case—that is, if two or more input patterns had the same output—then the reverse action of the gate would be ambiguous.

The devices now known as the Fredkin gate and the Toffoli gate *(see illustration on page 109)* both have three inputs and three outputs; and, as required for reversibility, each input pattern maps to a unique output. In the Fredkin gate, one signal controls whether the other two data lines pass straight through the gate or else have their positions swapped. In the Toffoli gate, two of the signals control the third; if the first two bits are both 1, then the third bit is inverted.

Like the NOT gate, both the Fredkin and the Toffoli gates are their own inverses: No matter what the values of the three input signals, running them through two successive copies of the same gate will return the signals to their original values. Both gates are also computationally universal, meaning that a computer assembled from multiple Fredkin or Toffoli gates (and no other components) could simulate a Turing machine or any other device of equivalent computational power. Thus the gates might be considered candidates for a real reversible computer.

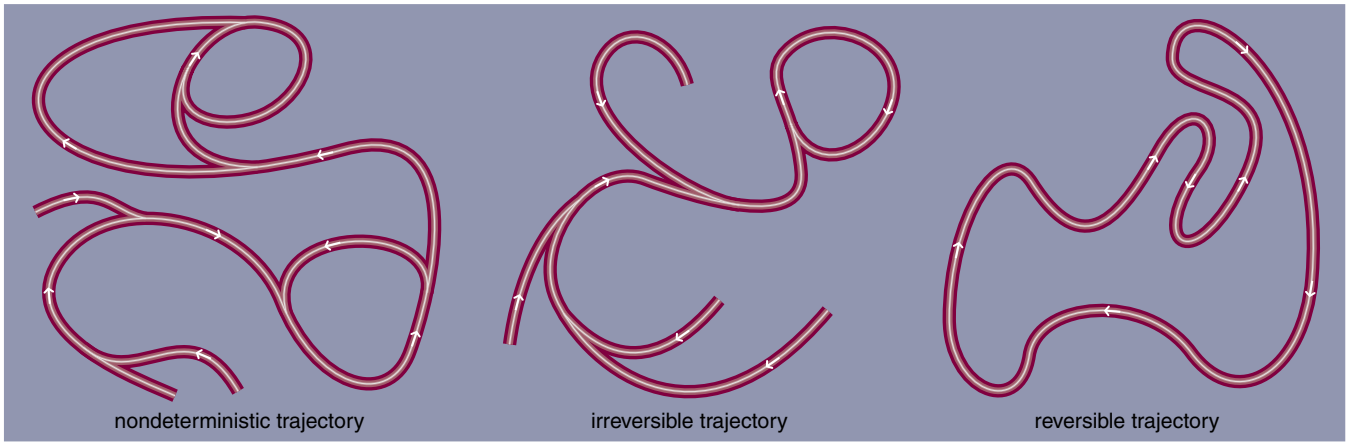
Of course logic gates are still just abstract devices; they have to be given some physical implementation with transistors or other kinds of hardware. Starting in the early 1990s, several groups have been designing and building prototypes of reversible (or nearly reversible) digital circuits. For example, at MIT a group including Michael Frank and Thomas F. Knight, Jr., fabricated a series of small but complete processor chips based on a reversible technology; Frank continues this work at Florida State University. At the University of Gent in Belgium, Alexis De Vos and his colleagues have built several reversible adders and other circuits.

It's important to note that building a computer according to a reversible logic diagram does not guarantee low-power operation. Reversibility removes the thermodynamic floor at $kT \ln 2$, but the circuit must still be designed to attain that level of energy savings. The current state of the art is far above the theoretical floor; even the most efficient chips, reversible or not, dissipate somewhere between 10,000 and 10 million times $kT \ln 2$ for each logical operation. Thus it will be some years before reversible technology can be put to the ultimate test of challenging the three-zeptojoule barrier. In the meantime, however, it turns out that some concepts derived from reversible logic are already useful in low-power circuits. One of these is charge recovery, which attempts to recycle packets of electric charge rather than let them drain to ground. Another is adiabatic switching, which avoids wasteful current surges by closing switches only after voltages have had a chance to equalize.

Back to the Future

Making a computer run backwards is not just a problem for hardware hackers and logic designers. There are also issues at higher levels of abstraction. What will it be like to write and run a program on a machine that can bounce back and forth in time?

Most of us already have some experience with a rudimentary form of reversibility—namely the *undo* command. Even this simple facility for unwinding a computation has its semantic hazards. (Should "*undo undo*" leave you two steps back, or none?) In many programs, *undo* is a bit of a sham; it doesn't turn the program around but



The evolution of a computational process can be represented as a path in an abstract space. Every point in the space corresponds to some possible state of the machine, and the coordinates of the point encode everything that can be known about the state. As the computer goes from one state to the next, it traces out a trajectory. The path of a nondeterministic computer can have both merge points (where a state has two or more predecessors) and branch points (where a state has multiple successors). It is the branch points that make this process nondeterministic; a computer that behaves in this unpredictable and inconsistent way is usually considered broken. An irreversible trajectory can have merge points but not branch points. (If the computation were reversed, the merge points would become branch points, and determinism would be lost.) A reversible computer must evolve along a simple path with neither merge points nor branch points. In a finite space, the path must be closed.

just revisits saved snapshots. Software that *really* runs in reverse has some harder problems to solve.

Consider the matter of rounding. Most numerical calculations are done with limited precision; all numbers have to fit into 32 bits, say. When that's not enough room, the least-significant digits are discarded. But dropping digits is a no-no in the world of reversible computing. If 1/3 gets converted into 0.33333, then reversing the process must somehow yield exactly 1/3 again.

Even arithmetic with exact numbers can get us into trouble, if one of those numbers happens to be zero. Division by zero is forbidden everywhere in civil society; in a reversible machine, we might also have to outlaw multiplication by zero, because it amounts to erasure, yielding a result of zero no matter what the value of the multiplicand. Personally, I rather like this idea of treating division and multiplication symmetrically, but it runs counter to hundreds of years of habit.

Henry G. Baker, in a thoughtful essay that touches on several issues of reversibility, cites another example: Newton's method for approximating square roots. In Newton's algorithm, you approach the square root of N by starting with an arbitrary guess, x, and then calculating $(x+N/x)/2$. The result of this expression becomes a new guess, and you repeat the procedure until you reach the desired accuracy. Almost any guess will converge on the true root; in other words, the algorithm "exemplifies a large class of computations in which the output appears to be independent of the input." The method also appears to be highly irreversible, since all those different inputs lead to the same output. If the computation were reversed, how could the algorithm find its way back from the one output to each of the many possible inputs? But in fact it can (at least if the initial x is greater than the square root of N). If the computation is done without rounding or truncating intermediate results, all of the information needed to reverse the process is preserved in the "insignificant" decimal places of the answers. Each initial guess can be reconstructed exactly.

Baker and others also point out that a reversible computer would be a wonderful device for debugging programs. With conventional hardware, you can set a breakpoint to stop a program when it goes off the rails, but then it's a struggle to figure out what path brought it to the trouble spot. With a reversible device, you can just back it up. Running a compiler in reverse might also be fun. Generating source code from an executable program is a handy trick (which the vendors of commercial software would no doubt want to disable).

Let's Get Physical

Landauer, who died in 1999, championed the slogan "Information is physical." He meant, first of all, that information cannot exist in the abstract but has to be embodied somehow—in electrons, photons, chalk marks, neural excitations. But the slogan can also be taken as a call to heed the laws of physics when dealing with information, just as one must with matter and energy. In this respect irreversible computers are notorious scofflaws. They make bits appear out of nothing and then disappear into the void again when they're no longer needed. They obey no conservation laws.

A reversible computer is a better-behaved device, more at home in the universe we live in. As Toffoli wrote in 1982: "Computation—whether by man or by machine—is a physical activity. If we want to compute more, faster, better, more efficiently, and more intelligently, we will have to learn more about nature. In a sense, nature has been continually computing the 'next state' of the universe for billions of years; all we have to do—and, actually, all we *can* do—is 'hitch a ride' on this huge ongoing computation, and try to discover which parts of it happen to go near to where we want."

Bibliography

Baker, Henry G. 1992. NREVERSAL of fortune—the thermodynamics of garbage collection. In *Proceedings of the International Workshop on Memory Management*, St. Malo, France, September 17–19, 1992, pp. 507–524. London: Springer Verlag. (Later version available at http://www.pipeline.com/~hbaker1/ReverseGC.html)

Bennett, C. H. 1973. Logical reversibility of computation. *IBM Journal of Research and Development* 17:525–532.

Bennett, Charles H. 1982. The thermodynamics of computation—a review. *International Journal of Theoretical Physics* 21:905–940. (Reprint at http://www.research.ibm.com/people/b/bennetc/bennettc1982666c3d53.pdf

Bennett, Charles H. 1988. Notes on the history of reversible computation. *IBM Journal of Research and Development* 32:16–23.

De Vos, Alexis, and Yvan Van Rentergem. 2005. Reversible computing: from mathematical group theory to electronical circuit experiment. *Proceedings of the Second Conference on Computing Frontiers*, Ischia, Italy, pp. 35–44. New York: ACM Press.

Frank, Michael, Carlin Vieri, M. Josephine Ammer, Nicole Love, Norman H. Margolus and Thomas F. Knight, Jr. 1998. A scalable reversible computer in silicon. In *Unconventional Models of Computation*, edited by C. S. Calude, J. Casti and M. J. Dinneen, pp. 183–200. Singapore: Springer. (Preprint available at http://www.ai.mit.edu/people/mpf/rc/flattop/newpaper/flattop.pdf/a-scalable-reversible-computer.pdf)

Frank, Michael P. 2005. Introduction to reversible computing: motivation, progress, and challenges. *Proceedings of the Second Conference on Computing Frontiers*, Ischia, Italy, pp. 385–390. New York: ACM Press.

Fredkin, Edward, and Tommaso Toffoli. 1982. Conservative logic. *International Journal of Theoretical Physics* 21:219–253.

Hey, Anthony J. G. (ed.). 1999. *Feynman and Computation: Exploring the Limits of Computers*. Reading, Mass.: Perseus Books.

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

Landauer, Rolf. 1988. Dissipation and noise immunity in computation and communication. *Nature* 335:779–784.

Landauer, Rolf. 1991. Information is physical. *Physics Today* 44:23–29.

Lecerf, Yves. 1963. Machines de Turing réversibles. *Comptes Rendus Hebdomadaires des Séances de L'académie des Sciences* 257:2597–2600. (Also available at http://www.cise.ufl.edu/~mpf/rc/Lecerf/lecerf.html)

Leeman, George B. Jr. 1986. A formal approach to undo operations in programming languages. *ACM Transactions on Programming Languages and Systems* 8:50–87.

Leff, Harvey S., and Andrew F. Rex (eds.). 2003. *Maxwell's Demon 2: Entropy, Classical and Quantum Information, Computing*. Bristol: Institute of Physics Publishing.

Lutz, Christopher, and Howard Derby. 1982. Janus: a time-reversible language. Unpublished report, California Institute of Technology. http://www.cise.ufl.edu/~mpf/rc/janus.html

Markov, Igor L. 2003. An introduction to reversible circuits. *Proceedings of the 12th International Workshop on Logic and Synthesis* (Laguna Beach, May 2003) pp. 318–319.

Mu, Shin-Cheng, Zhenjiang Hu and Masato Takeichi. 2004. An injective language for reversible computation. In *Mathematics of Program Construction 2004*, pp. 289–313. Berlin: Springer-Verlag.

Toffoli, Tommaso. 1980. Reversible computing. MIT Laboratory for Computer Science Technical Memo 151, http://pm1.bu.edu/~tt/publ/revcomp-rep.pdf.

Toffoli, Tommaso. 1982. Physics and computation. *International Journal of Theoretical Physics* 21:165–175.

Van Rentergem, Yvan, Alexis De Vos and Leo Storme. 2005. Implementing an arbitrary reversible logic gate. *Journal Of Physics A: Mathematical and General* 38:3555–3577.

Zuliani, P. 2001. Logical reversibility. *IBM Journal of Research and Development* 45:807–817.