Reversible Computing

Illustration: Parag Joshi

{Acheiving next-generation computing speed}

> BY MICHAEL P. FRANK

The computer industry is quickly approaching a fundamental brick wall. The last ~150 years of developments in the fields of thermodynamics and quantum mechanics have established, beyond all reasonable doubt, a number of incontrovertible physical facts that conspire to place firm limits on future computer performance.

First, the total information content of any physical system is finite. There are strict limits on the number of bits that may be stored in a system of given size and energy content. Atomic matter contains only a few bits of physical information per atom, and this places strict limits on bit storage densities.

Second, physical dynamics is reversible, meaning that it transforms physical states in a one-to-one fashion. As a result, information can never really be destroyed. Every clock cycle, a typical logic gate in today's processors 'overwrites' its old output with a new one. But, the information in the old output physically cannot be destroyed. In today's CPUs, at least 100,000 bits of physical information (electron states) are used (redundantly) to represent each logical bit. All this information, since it cannot be destroyed, is essentially pushed out into the environment, and the energy committed to storing this waste information (entropy) in the environment is, by definition, heat. In fact, the majority of the heat coming out of your laptop or desktop computer can be directly attributed to the physical information discarded in its CPU's logic operations.

Of course, over time, logic gates are becoming more efficient. Researchers are experimenting with molecular logic technologies that use many fewer physical bits to represent a logical bit. However, the absolute minimum we will ever attain is of course at least 1 physical bit used per logical bit encoded!

As a result, we can calculate a firm limit on the power-performance of future computers that use conventional logic techniques based on throwing away bits. Let's say you demand a laptop that uses no more than about 100 Watts of power, because otherwise it will be too hot and/or noisy to use comfortably. (Think about a 100 W light bulb, and a 1,000 W hair dryer.) It turns out that the maximum rate at which bits of physical information can be discarded within your computer is given simply by the rate of power consumption, divided by the temperature of the environment into which the machine will dump its waste heat.

Assuming you're not planning to only use your laptop while spacewalking on interstellar voyages, the typical environment temperature will be on the order of room temperature, or about 300 degrees Kelvin above absolute zero.

So, modern physics can guarantee you that your laptop will be able to discard no more than ~3.5×10^{22} bits of physical information per second-this in fact really is just the same physical quantity as (100 W ÷ 300 K), but with the units converted to different (but compatible) physical units of bits per second.

Wow, ten to the twenty-second power sounds like a huge number, but consider that today's fastest processors (e.g., the Pentium 4 Extreme) already have clock speeds of 3.2 GHz, while the

Mike designed one of the first universal DNA computers as a graduate student at MIT. He has been continuing his reversible computing research as an Assistant Professor at the University of Florida's Department of Computer and Information Science and Engineering

largest CPUs (e.g., Itanium 2) have over 220 million transistors. If we discarded the gate voltage information of all those transistors every clock cycle, it would mean that ~7×10^{17} bits were discarded each second! Even if only half the transistors were switched on average, this is already only 100,000x smaller than the limit. We can only get significantly closer by continuing to reduce the physical redundancy of our bit encodings.

> Computing 100,000 times faster will only take another 25-33 years, before today's college graduates retire

How long will it be before this 3×10^{22} bit-ops/sec limit for 100-W room-temperature conventional computing is reached?

A corollary of Moore's Law tells us that historically (and this has really remained true over the last 100 years) computer performance has doubled about every 18 months to two years. At this rate, a factor of 100,000 will only take another 25-33 years to achieve-that is, before today's college graduates will retire. If bit redundancies stop decreasing before this, performance of conventional machines will level off even sooner!

There is one, and only one, solution to this dilemma that is consistent with the most fundamental laws of physics: namely, stop throwing away so many bits when we compute. Instead, uncompute the unwanted bits. This turns out to allow us to recycle most of their energy, rather than dissipating it to waste heat! This is the idea behind reversible computing. It really is just the application of general principles of recycling to computing. I therefore sometimes also call it "green computing." OK, that's the basic principle behind reversible computing. Now, how does it work?

Ballistic Processes

A ballistic process in physics is a process in which a system proceeds 'forward' under its own momentum, with only a small fraction of the free energy in the system being converted into heat. This also means that only a small fraction of the system's extropy (compressible information) is being converted to entropy (incompressible information).

For instance, when an ICBM (Intercontinental Ballistic Missile) makes a suborbital flight between continents, it 'coasts' for most of the journey, with only a small fraction of its kinetic and potential energy being lost to friction during its trajectory through the upper atmosphere (thus the term 'ballistic'). As another example, an electron will travel ballistically in a regular structure like a silicon crystal or a carbon nanotu be, until it hits a lattice defect which deflects its path, and turns its extropy (the information describing its motion) into entropy.

The degree to which a process is ballistic can be quantified by its quality factor, or Q: this is the ratio between the amount of energy involved in carrying out the process, and the amount that is dissipated to heat. It's also the ratio between the extropy involved and the amount of entropy generated.

How high can Q be? For a perfect diamond crystal vibrating in a vacuum, it can be as large as 10^{12} or more. This means the diamond will vibrate trillions of times before its motion dampens out. We know of no fundamental reason why very high Qs cannot also be achieved in much more complex structures—such as those within a computer.

The idea in reversible computing is to construct future nanocomputers as oscillators that work ballistically, with high Q. You put in the initial data, give the computer a 'kick' to get it started, and it 'coasts' along ballistically, from one step of the computation to the next, losing only a small fraction of its energy to heat at each step. This would allow it, in theory, to perform a number of operations per unit time that is far beyond the power-over-temperature limit I mentioned earlier, limited only by the Q that can be attained in the logic mechanism.

Probably the biggest challenge for reversible computing is to design nanometer-scale electrical or electromechanical oscillators that have a high enough Q to allow reversible computers to be practical. But, we know of no fundamental reason why it cannot be done. It is 'just' an engineering problem, which we expect will someday be solved. So, for the remainder of this article, I will focus on the higher-level architectural and computer science aspects of reversible computing.

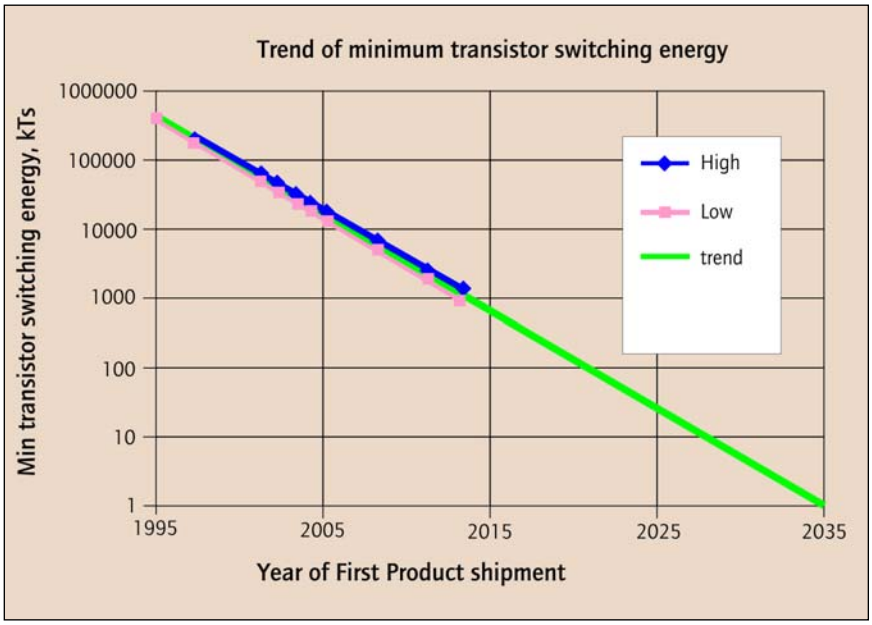

Figure 1: *Projected minimum bit energies*

Adiabatic Circuits

For a physical process to be highly ballistic implies that it is also highly adiabatic ("without flow of heat"), since at at any time heat is flowing between subsystems at different temperatures, this

means that extropy is being converted to entropy, and free energy is being reduced. A small subfield of electrical engineering called adiabatic circuits studies logic circuits that operate in a mostly-adiabatic mode.

Before		After		Pass2	
A	B	A	B	A	B
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	1	1	0
1	1	1	0	1	1

Table 1: *Reversible CNOT Transition Table*

Ordinary digital logic encodes bits using two voltage levels (high and low), and uses transistors to connect and disconnect circuit nodes to each other and to power supplies so as to twiddle the voltage levels in a way that carries out the desired computation.

>Even a NOT gate, as traditionally implemented, is irreversible, since it overwrites its last output

Adiabatic logic circuits can use the same overall approach, but two new rules must be followed: (1) never turn on a transistor when there is a voltage difference between its terminals, and (2) never turn off a transistor when there is a current passing through it. These are both irreversible events, which I call 'sparks' and 'squelches' respectively, and they each cause a certain irreducible amount of entropy to be generated.

It turns out you can still do universal computing despite these constraints, but you have to power your logic gates using trapezoidal variable-voltage power supplies driven by specially-designed resonant oscillators, and the logic design itself has to be reversible.

Reversible Logic

A physical process can be highly adiabatic only if it is mostly logically reversible. This is because of the fact I mentioned earlier: physical information cannot be destroyed, so any bits that are discarded in your logic (together with the entire retinue of physical bits used to encode them) turn into entropy.

What kind of logic doesn't discard bits? Well, logic that only reversibly transforms bits, in place. Understanding this requires thinking about logic gates a little bit differently. You probably think of ordinary Boolean logic gates such as NOT, AND, and OR, as consuming 1 or 2 inputs and producing an output. But actually, in the electronics, these gates do not consume the input bits—only measure them—and the output bit is not, strictly speaking, 'produced,' it is destructively modified, overwritten with the new information. (And the old information is kicked out to form entropy.) Even a NOT gate, as traditionally implemented, is irreversible, since it overwrites its last output.

We require a new type of logic gate, which takes the joint state of all its bits (whether you want to think of them as 'input' or 'output') and transforms them in a reversible (one-to-one) way.

If you write down the truth table for one of these gates, rather than having columns for 'input' bits and 'output' bits—which would be misleading—you should have columns for the 'before' state and 'after' state of all of its bits. (I call this a 'transition table.') In order for the gate to be reversible, there should be a one-to-one relationship between the before and after states that are used.

Perhaps the simplest-to-describe nontrivial reversible gate, which was invented by Tom Toffoli at MIT in the 1970s, is today called CNOT (short for "controlled NOT"). It operates on a pair of bits, A and B, and flips B if and only if A is 1 ('true'). You can see why this is reversible—if you perform CNOT on bits A and B, and then do it again, it undoes itself; it uncomputes the change to B. See *Table 1*.

The simplest universal reversible gate, also invented by Toffoli, is called the Toffoli gate, or CCNOT (controlled-controlled-NOT). It takes three bits—A,B and C, and performs a CNOT between B and C if and only if A is 1 . In other words, it flips C if and only if both A and B are 1 (*Table 2*). The CCNOT gate also undoes itself. If all you have are Toffoli gates, you can still build a general-purpose reversible computer that is asymptotically as efficient as any other (except quantum computers) to within a constant factor.

Before			After		
A	B	C	A	B	C
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Note change in C

Table 2: *CCNOT Transition Table.*

Although they are the easiest reversible gates to describe, the CNOT and CCNOT gates aren't actually the easiest to implement using ordinary transistors.

Before			After		
A	B	G	A	B	G
0	0	0	1	0	0
0	0	1	1	E	1
0	0	2	1	1	2
0	1	0	1	1	0
0	1	1	1	1	1
0	1	2	1	1	2
0	2	0	1	2	0
0	2	1	1	2	1
0	2	2	1	2	2

Table 3: *Transition table for the single-transistor NDRAG(A:0?1) gate.*

In fact, it turns out that even a single CMOS transistor, together with a controlling signal, can be thought of as implementing a simple type of reversible gate. There are two types of CMOS transistors, PFETs and NFETs, and I call the corresponding reversible gates the PDRAG and NDRAG gates, for reasons we will soon see. It is helpful in describ-

ing these to use three distinct voltage levels, call them 0, 1, 2.

A DRAG gate (either P or N) operates on 3 bits (really 3-level 'trits'), which I will call A, B, and G. A and B are the source/drain terminals, and G is the transistor's gate node. In a typical DRAG gate operation, some subset of the bits make unconditional, externally-driven transitions between old and new values. The rest of the bits (if any) either remain as they are, or are 'dragged' to follow the trajectory of another of the bits, depending on the type of gate.

There isn't space to give the complete rules for DRAG gates here. They can be derived from the behavior of CMOS transistors. However, as an example, *Table 3* gives the transition table for NDRAG(A:0?1), which is a notation for an NDRAG gate in which node A unconditionally goes from 0 to 1, G remains constant, and B is conditionally dragged.

Note that if B is initially 0, then it will either stay 0 if G is 0, or be 'dragged' to 1 by A (case circled) if and only if G is 2. In the next similar instance, B is dragged to 2.

> In a reversible language, we must be able to forego inherently irreversible statements

The 'E' (error) entry for B in the second row means that the adiabatic rule "no squelches" is violated as A approaches G, and as a result B will end up at an intermediate level somewhere between 0 and 1 (depending on the transistor's exact switching threshold), which is considered an error. The line through the 'before' case A=0, B=1, G=2 means there will be a sudden, non-adiabatic 'spark' in this case as B jumps from 1 to 0, after which B will be dragged up to 1. Both squelches and sparks are disallowed in a true adiabatic circuit, and so these cases must be avoided in order for this 'gate' to be used reversibly.

Note that these CMOS transistors considered as logic gates differ from traditional logic gates in that the same physical hardware can act as a different gate at different times. For example, after being used as an NDRAG(A:0?1) gate, the same NFET can later be used as an NDRAG(A:1?0) gate, which will undo any change to B, if G is still the same.

It turns out that networks of DRAG gates, constructible using ordinary CMOS transistors, are in fact sufficient for reversibly emulating arbitrary partially- to fully- reversible computers with at most constant-factor overheads. In a project at MIT in the 1990s, a number of then-graduate students (including myself) fabricated several fully-reversible processor chips using a particular style of DRAG-based logic called SCRL (for split-level charge recovery logic), to prove that it could be done.

Since reversible logic can only reversibly modify bits in-place, not overwrite them, many aspects of computer architecture are changed by it. For example, since an ordinary WRITE operation is irreversible, registers must instead be accessed using operations like CNOT to flip register bits in place, or by swapping bits in and out, or by using DRAG operations to reversibly copy information to empty bits, or to uncopy bits from a duplicate copy.

Likewise, temporary bits that are computed internally in the course of doing an operation (like the carry bits in a ripple-carry adder) have to be uncomputed later before that hardware can be reversibly reused for another operation.

The extra steps needed to uncompute old intermediate results do in general take some overhead, and this overhead cuts into the energy savings that can be achieved by using reversible computing. But, as long as sufficiently high-Q devices can be built, and the cost per device continues to decrease, the overhead is outweighed by the much larger number of simultaneous operations that can be performed in a given time interval while staying within one's power budget. I have done a detailed analysis which indicates that by the year 2050, 100-watt reversible computers could be 1,000 to 100,000 times faster than the fastest possible 100W irreversible computer, even when the overheads of reversible logic are taken into account, assuming that Qs can continue to increase.

Reversible Instruction Sets

In an optimistic scenario, as device Qs continue to increase, eventually we will reach a point where further improvements in energy efficiency require not just the logic circuits, but also the machine's instruction set itself to be reversible. This is because emulating irreversible instruction sets on reversible hardware, while possible, does introduce extra overhead. Better asymptotic efficiency can in general be achieved if we allow the CPU programmer to code a custom reversible algorithm for

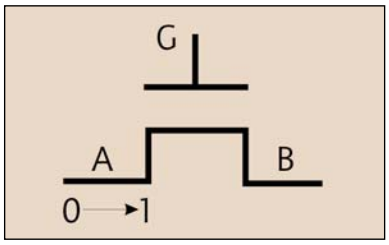

Figure 2: *An NFET as an NDRAG(A:0?1) gate..*

the problem at hand. The program itself will tell the computer exactly when and how to uncompute unwanted information, and when to throw it away, so as to make the best overall use of the machine's resources.

In the project at MIT, we also designed reversible machine instruction sets. These are different, but not too different, from ordinary irreversible instruction sets. Many common operations like NEG and ADD are already logically reversible, because they perform one-to-one transformations of their arguments in-place. Many others, like AND, are not, and have to be replaced by operations like ANDX, which exclusive-ors the result into a destination register, essentially performing a bitwise CCNOT on its 3 register arguments. Reversible memory access is done with reversible operations like EXCH (exchange) rather than read and write. And finally, ordinary branches and jumps are irreversible (because the machine forgets where it came from), and must be replaced with special paired branches, where the instruction at the branch destination can figure out where control came from, and uncompute that information (e.g., see some branch patterns in *Figure 3*).

Reversible Applications

Machine-language programs written in such reversible instruction sets have the interesting property that they can be run either forwards or in reverse. That is, after running a program, you

could hit the machine's "reverse button" and it will retrace all of its steps, and give you back the original inputs. However, this is not needed for energy recovery; that happens continuously inside the CPU, whichever direction it is running.

Reverse execution does have some minor applications which we have investigated, such as rolling back aborted transactions in databases, parallel simulations, or multi-player games, or to back up to an earlier state to deal with runtime exceptions, to repair damage caused by viruses or computer break-ins, or as an aid to program debugging. However, these applications can be implemented in other ways too, without actually using a reversible instruction-set design. The primary motivation for studying reversible instruction sets is their long-term necessity for improving the thermodynamic efficiency of computing. And also, just because they're cool!

Reversible High-Level Programming Languages

If reversible programming is necessary, we would prefer to write our programs in a high-level language. Once the requirement for reversibility becomes sufficiently stringent, automatic program translations to reversible languages performed internally within compilers cannot be expected to do the job, and we will need to give the high-level programmer the ability to directly control how the machine disposes of its unwanted information. Thus, we need a reversible high-level language.

In a reversible language, we must be able to forego inherently irreversible statements, such as destructive assignment of new values to program variables, including to members of data structures. Instead, we must make do with operations such as swapping or XORing values into desired locations.

Subroutines must clean up after themselves, and uncompute temporary values that they computed in their local variables, and in other scratch space. Automatic garbage collection inherently wastes free energy, because the garbage collector does not know how the information in the allocated memory was computed, and thus can do nothing but compress it, as much as is feasible (analogously to a trash compactor) and then eject it as entropy. If the program is designed to explicitly uncompute and deallocate its own garbage, it can often do much better.

Control flow constructs are similar to traditional ones, but are time-symmetric in form. There are IF and SWITCH variants that test conditions at the end as well as the beginning of the body, in order to reversibly re-merge the control flow, as well as loop constructs that provide conditions for loop entry as well as loop exit. Subroutines can be called either forwards or in reverse, which provides an easy way to clean up their temporary results. (Run the subroutine forwards, then do something with the result, then run it backwards.)

```
(defsub mult (m1 m2 prod)
 (for pos = 0 to 31
  (if (m1 & (1 << pos))
   (prod += (m2 << pos)))))
```

Figure 4: The R programming language

Early reversible languages included Janus, Ψ-Lisp, and R, my own contribution (*Figure 4*). Others have been invented recently within the quantum computing community, where reversible computing is needed for other reasons.

Reversible vs. Quantum Computing

Reversible computing is similar to quantum computing, and shares some technological requirements with it. However, the motivations for them are different, as are some of the requirements.

Briefly, a quantum state is a unit-length vector of complex numbers, one for each possible classical state of the system. A classical state can be represented by a quantum state vector having a 1 in that classical state, and a 0 in all the other classical states.

A quantum operation is a linear, one-to-one mapping from 'before' state vectors to 'after' state vectors. A classical reversible operation is therefore a special case of a quantum operation, in which classical input states are always mapped to classical output states. The quantum viewpoint therefore expands the range of possible operations. An input state can be mapped to a state that is 'in between' classical states, in the sense that the vector has non-zero components in more than one classical state. Apparently, allowing these in-between states opens up paths to the solution of some problems that are exponentially shorter than the shortest classical paths we know of. In particular, there is a quantum algorithm for factoring numbers (Shor's algorithm) that requires only polynomially many quantum operations to factor a number with n digits, whereas the best known classical algorithm takes an exponential number of classical operations. Similarly, simulating quantum mechanics itself can be done in polynomial time on a quantum computer, but takes exponential time using known classical algorithms.

This, then, is the primary motive for quantum computing: to obtain exponential algorithmic speedups for certain problems. Interestingly, the quantum algorithms have to be reversible in order to work properly, and in fact, they frequently use ordinary classical reversible algorithms as subroutines. So, this is another application for reversible languages: writing quantum algorithms. A few additional instructions are needed to provide the uniquely quantum operations. Unfortunately, only a very few applications are known for quantum computing so far.

However, the motive for reversible computing is different. It is to speed up future computing not just for rare applications but in all cases where power dissipation is (for whatever reason) limited, and where the application can be parallelized to take advantage of the larger number of operations that can be done in parallel in a reversible machine per unit time, given the power budget.

Like quantum computing, reversible computing requires that the internal state of the computer be well-isolated from unwanted interactions with the environment, interactions which might unpredictably scramble a bit, converting it to entropy.

However, unlike quantum computing, reversible computing does not require that all quantum states of the machine be kept stable against interactions with the environment, only that the classical states be stable. This allows certain naturally-stable quantum states called *pointer states* to be used for representing the state of the computation. These are the states that, unlike general quantum states, can be measured without disturbing them. The ability to stick with pointer states vastly simplifies the task of error correction.

So, in summary, reversible comput-

ing is both much easier and more generally useful than quantum computing, and so it deserves increased attention, relative to quantum computing. Quantum computing has received a lot of hype, more than is probably justified. In contrast, reversible computing is little known, and still relatively obscure, even though it has been around much longer.

The connection between bit erasure and entropy generation was hinted at in 1949 by von Neumann, and was proven in 1961 by Rolf Landauer of IBM. In 1973, Charles Bennett (also of IBM) proved that useful reversible computing was theoretically possible, and in 1989 he devised a general algorithm for fairly efficiently emulating irreversible computations on reversible machines. There have been few important theoretical developments since then, but a large number of more or less adiabatic circuit implementations have been proposed.

Unfortunately, present-day oscillator technologies do not yet provide high enough quality factors to allow reversible computing to be practical today for general-purpose digital logic, given its overheads. Nano-device engineers need to turn increasing attention to finding ways to solve this problem, as it offers the only hope for possibly maintaining Moore's Law for more than another two or three decades.

Conclusion

Does your parent or spouse bug you for not recycling your bottles and cans? Tell them they really should be scolding your computer! (Or perhaps Intel.)

Computers today are terribly wasteful devices. They throw away millions of bits, billions of times every second. Each bit contains ~100,000 physical bits, which all turn into entropy. Their associated energy becomes heat, which burns up your lap, runs up your electric bill, and limits your computer's performance.

The most energy-efficient computers based on ordinary irreversible logic that will ever be physically possible are 'only' ~100,000x times as efficient as today's, which means the irreversible-computer industry will definitely hit a plateau sometime within your lifetime.

Truly, the only way we might ever get around this limit is by using reversible computing, which uncomputes bits that are no longer needed, rather than overwriting them. Uncomputing bits allows their energy to be recovered and recycled for use in later operations.

In essence, a reversible computer computes by ballistically coasting along through its configuration space, analogously to a roller-coaster following a complexly-shaped track through a high-dimensional space.

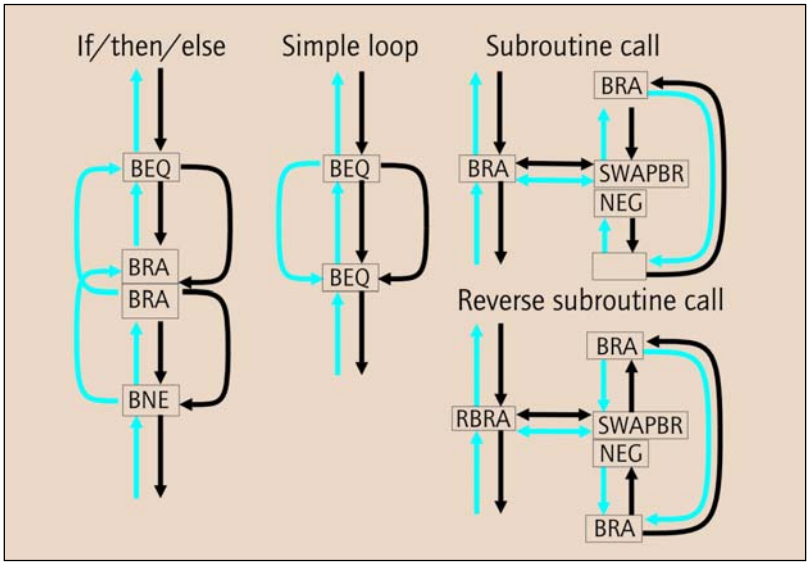

Figure 3: *Some reversible control-flow constructs. Instructions shown are from the PISA instruction set. Blue arrows show reverse paths. Note the ubiquitous use of paired branches.*

Examples of reversible logic circuits, microprocessor architectures, and programming languages have already been designed, and although they are a little bit different from familiar ones, they are similar enough that today's computer and software engineers can quickly learn to adopt such design styles, when (not if) it becomes necessary.

The ballistic quality factors of today's oscillator technologies are not quite good enough yet to make reversible computing a practical reality today, but we don't know any reason why continued engineering improvements, perhaps based on nanoscale electromechanical resonators, might not push the workable quality factors up high enough that reversible computing will eventually become practical.

In conclusion, if you're looking to get "ahead of the curve" on a future technology, you just might want to take a closer look at reversible computing. The most rock-solid principles of fundamental physics can be used to prove that reversible computing will be absolutely necessary for further progress in computer performance once bit redundancies (physical bits per logical bit encoded) stop decreasing so quickly, which must happen within 20-30 years. Many theoretical and engineering proofs-of-concept have established that reversible computing really can work, and it isn't even all that hard to understand.

Educating yourself in reversible computing today just might give you a valuable leg up when the computer industry reaches that big brick wall that's looming ahead, not far down the road, which only reversible computing can hurdle.

For further reading: Mike's Reversible Computing research website, located at *http://www.cise.ufl.edu/research/revcomp*, includes a large number of original research papers relating to reversible computing, which include extensive references into the primary literature. Lectures and recommended reading lists on reversible computing and related topics can be found on the website for Mike's course, Physical Limits of Computing, available at *http:// www.cise.ufl.edu/~mpf/physlim*.

<Dev 2.0>