A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology

James R. Heath, *et al.*

Science **280**, 1716 (1998);
DOI: 10.1126/science.280.5370.1716

A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology

James R. Heath, Philip J. Kuekes, Gregory S. Snider, R. Stanley Williams

Teramac is a massively parallel experimental computer built at Hewlett-Packard Laboratories to investigate a wide range of different computational architectures. This machine contains about 220,000 hardware defects, any one of which could prove fatal to a conventional computer, and yet it operated 100 times faster than a high-end single-processor workstation for some of its configurations. The defect-tolerant architecture of Teramac, which incorporates a high communication bandwith that enables it to easily route around defects, has significant implications for any future nanometer-scale computational paradigm. It may be feasible to chemically synthesize individual electronic components with less than a 100 percent yield, assemble them into systems with appreciable uncertainty in their connectivity, and still create a powerful and reliable data communications network. Future nanoscale computers may consist of extremely large-configuration memories that are programmed for specific tasks by a tutor that locates and tags the defects in the system.

The last 25 years have witnessed astonishing advances in the fields of microelectronics and computation. The first integrated circuit microprocessor, the Intel 4004, was able to perform roughly 5000 binary-coded decimal additions per second with a total power consumption of about 10 W (~500 additions per Joule) in 1971, whereas modern microprocessors can perform $\sim 3 \times 10^6$ additions per Joule. The 1997 National Technology Roadmap for Semiconductors (*1*) calls for an additional factor of 10^3 increase in the computational efficiency by the year 2012. If this goal is attained, then performance of the silicon-based integrated circuit will have improved by nearly seven orders of magnitude in 40 years, using energy consumed per operation as a metric, with a single manufacturing paradigm. Although complementary metal oxide semiconductor (CMOS) technology is predicted by many researchers to run into significant physical limitations shortly after 2010 (*2*), the energy cost of an addition operation will still be nowhere near any fundamental physical limit. A crude estimate of the energy required to add two 10-digit decimal numbers, based on a thermodynamic analysis of nonreversible Boolean logic steps (*3*, *4*) is $\sim 100 \cdot k \cdot T \cdot \ln(2)$, which implies that 3×10^{18} additions per Joule can be performed at room temperature without any reversible steps. Thus, there are potentially eight orders of magnitude in computational energy efficiency in a nonreversible machine available beyond the limits of CMOS technology. To achieve these further advances will require a totally different type of computational machinery, but knowing that such a system is in principle possible provides a strong incentive to hunt for it. The requirement for inventing a new technology paradigm has created exciting research opportunities for physical and biological scientists as well as for electrical engineers. Indeed, much of the current interest in interdisciplinary research in areas such as nanofabrication, self-assembly, and molecular electronics is being driven by this search for a new archetype computer.

A number of alternatives to standard Si-based CMOS devices have been proposed, including single-electron transistors (*5*), quantum cellular automata (*6*, *7*), neural networks (*8*, *9*), and molecular logic devices (*10*, *11*). A common theme that underlies many of these schemes is the push to fabricate logic devices on the nanometer-length scale. Such dimensions are more commonly associated with molecules than integrated circuits, and it is not surprising that chemically assembled (or bottom-up) configurations, rather than artificially drawn (or top-down) structures created with lithography, are expected to play an increasingly important role in the fabrication of electronic devices and circuits. We define chemical assembly as any manufacturing process whereby various electronic components, such as wires, switches, and memory elements, are chemically synthesized (a process often called "self-assembly") and then chemically connected together (by a process of "self-ordering") to form a working computer or other electronic circuit (*12*).

Several problems will arise when such an assembly is used for some computational task. Some fraction of the discrete devices will not be operational because of the statistical yields of the chemical syntheses used to make them, but it will not be feasible to test them all to select out the bad ones. In addition, the system will suffer an inevitable and possibly large amount of uncertainty in the connectivity of the devices. Given these problems, how does one communicate with the system from the outside world in a reliable and predictable way and be assured that it is performing error-free computations? Furthermore, because one goal of nanoscale technology is to provide a huge number (for example, a mole) of devices for a system, how does one impose an organization that allows the entire ensemble to operate efficiently? A self-ordering process is only likely to produce fairly regular structures with low information content, but real computers built today have great complexity imposed by human designers. A chemically assembled machine must be able to reproduce the arbitrary complexity demanded for general-purpose computation.

In engineering, the answer to low but nonzero failure rates is to design redundancy into the system. The history of integrated-circuit technology has been that wiring and interconnects have become increasingly more expensive with respect to active devices. Should nanotechnology give us extraordinarily "cheap" but occasionally defective devices, then nearly all expense will be shifted to the wires and connections. Recent research at Hewlett-Packard (HP) Laboratories with an experimental computer, code-named "Teramac", has illuminated several of these issues. Although Teramac was constructed with conventional silicon integrated-circuit technology, many of the problems associated with this machine are similar to the challenges that are faced by scientists exploring nanoscale paradigms for electronic computation. In order to keep the construction costs as low as possible, the builders of Teramac intentionally used components that were cheap but defective, and inexpensive but error-prone technologies were used to connect all the components. Because of the physical architecture chosen to implement powerful software algorithms, (*13*) Teramac could be configured into a variety of extremely capable parallel computers, even in the presence of all the defects. Thus, we define defect tol-

J. R. Heath is in the Department of Chemistry and Biochemistry, University of California at Los Angeles, Los Angeles, CA 90095–1569, USA. P. J. Kuekes, G. S. Snider, and R. S. Williams are at Hewlett-Packard Laboratories, Palo Alto, CA 94304–1392, USA.

erance as the capability of a circuit to operate as desired without physically repairing or removing random mistakes incorporated into the system during the manufacturing process (*14*). The major surprises of the Teramac project were that the compiling time for new logical configurations was linear with respect to the number of resources used and the execution time for many algorithms was surprisingly fast, given the large number of defects in the machine. The architecture of Teramac and its implementation of defect tolerance are relevant to possible future chemically assembled circuits (*15*).

Custom Configurable Architecture

The name "Teramac" takes "Tera" from 10^{12} operations per second, which is achieved by 10^6 logic elements [or gates (*4*)] operating at 10^6 Hz, and "mac" from "multiple architecture computer." It is a large Custom Configurable Computer (CCC) (*16*) that was designed for architectural exploration. The key property of reconfigurable architectures, as opposed to conventional processors, is that they can be configured by means of a software instruction set into a large variety of very different digital systems. Teramac contains 864 identical chips designed at HP labs and built specifically for Teramac. These chips, called field programmable gate arrays (FPGAs), contain a large number of very simple computing elements and a flexible communications network for routing the signals among the computing elements. Each computing element performs a six-input, one-output combinatorial logic function. The logic function is not performed with active logic elements, but rather with memory—that is, the "answers" to the logical functions (the truth tables) are stored in 64-bit Look-Up Tables (LUTs). Each LUT holds the equivalent of 10 logic gates, and there are a total of 65,536 LUTs in the machine. Thus, a total of 4 megabits (65,536 × 64 bits) of configuration memory is used to define the logic functions of all the computing elements. The system operates at a clock rate of 1 MHz, so that each computing element computes a new 1-bit output datum each microsecond, with all the 65,536 LUTs operating in parallel. The 4 megabits of configuration memory are drawn from about 30% (256) of the FPGAs. The bulk of the FPGAs are used only for communication and signal routing. It was significantly less expensive to design and manufacture a single FPGA, and then ignore the LUTs present on the chips that were used only for communication, than it would have been to produce two special-purpose chips.

In a typical microprocessor, a description of what the chip should do is first developed, and then the hardware is constructed on the basis of that logic. The general idea behind a CCC is conceptually the opposite. A generic set of wires, switches, and gates are fabricated in the factory, and then the resources are configured in the field by setting switches linking them together to obtain the desired functionality. For Teramac, these components are in the FPGAs, and they are the building blocks from which almost any digital machine can be constructed. The architecture of a computer is determined by the logical graph of wires connecting the gates. In FPGAs, software (field)–addressable switches determine the wiring relationships among the components. An FPGA can be logically thought of as consisting of two planes. In one plane are address lines that control a large configuration memory determining what switches in a crossbar are open and closed, and thus what functions are carried out by the LUTs. The other plane contains a separate set of application data lines and the actual LUTs that are connected into the desired configuration by the switches. A drawing of a crossbar is shown in Fig. 1A.

The use of FPGAs allows one to load a desired custom architecture onto Teramac through an automated software routine. To do this, Teramac uses an extreme version of what is known as the very long instruction word (VLIW) architecture. This indefinitely (or insanely) long instruction word (ILIW) is essentially the translated logical description of the custom computer that is desired. In a typical nonparallel machine, 32-bit instructions are issued sequentially. Machines that use a VLIW achieve instruction level parallelism in a processor by having the compiler issue single instructions with perhaps several hundred bits. Teramac uses a 300-megabit word, essentially as a single instruction that sets every configuration bit in every FPGA (most of which are in the crossbars to be discussed below). Such an instruction is only rarely downloaded, and the process is relatively expensive. However, this single instruction is powerful enough to reconfigure the entire machine into the desired custom computer.

Mapping a particular logical machine onto the physical resources of a CCC could easily be intractable, especially if there are a very large number of switches and LUTs but only a few viable configurations. This problem is similar to that of the traveling salesman who is forced to pick the shortest possible route among a large number of cities. The physical architecture of Teramac was designed to ensure that a very large

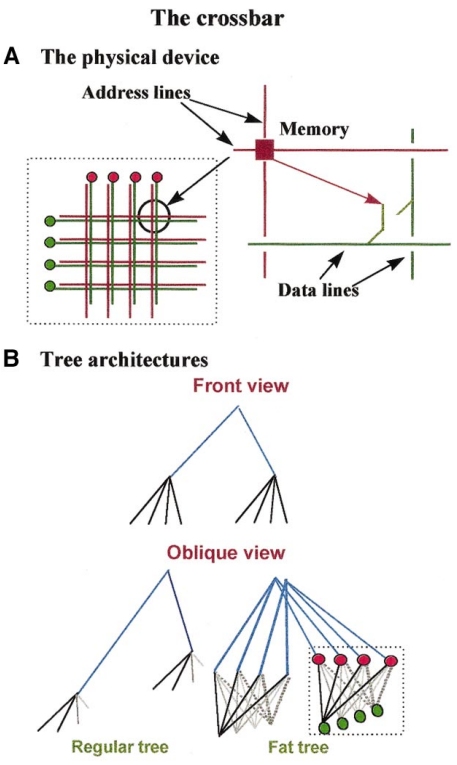

Fig. 1. Graphical presentation of concepts related to the logical architecture of Teramac. (**A**) The crossbar represents the heart of the configurable wiring network that makes up Teramac. (**Inset**) A configurable bit (a memory element) that controls a switch, which required six transistors to physically implement. The bit is located and configured by applying a voltage across the address lines, and its status is read by means of the data lines (they are either shorted or open). The crossbar provides not only a means of mapping many configuration bits together into some desired sequence, but it also represents a highly redundant wiring network. Between any two configuration bits, there are a large number of pathways, which implies a high communication bandwidth within a given crossbar. Logically, this may be represented as a "fat tree." Such a "fat tree" is shown in (**B**), where it is contrasted with a standard tree architecture. Both trees appear the same from the front view, but from an oblique view, the fat tree has a bandwidth that the standard tree does not. Color-coded dots and a dashed box are included to show the correspondence between a given level of the fat tree and the crossbar in (A). Several important issues are highlighted in this representation of the crossbar architecture. At every junction of the crossbar is a switch. Start at any point in the crossbar, and it is apparent that, by setting the appropriate switches, there are many possible pathways to any other junction. This degeneracy of pathways lends the crossbar architecture a high threshold for defect tolerance. It is also apparent from the drawing that $2n^{1/2}$ address lines are needed to address n switches, and that wires dominate all the drawings.

number of instruction words (switch settings) provide satisfactory configurations for any desired computer design. It may still be essentially impossible to find the optimum mapping, but as long as there are many possible solutions, it should be relatively easy to find a reasonable one (just as traveling salesmen do when planning their trips). Two concepts are important for understanding how Teramac was designed to provide a large number of satisfactory physical realizations for any logical configuration: the "fat tree" (*17*), which is a special case of a Banyan network (*18*), and "Rent's rule" (*19*).

The power of the fat-tree architecture can be appreciated by first considering the regular family tree (Fig. 1B). A parent produces two children, each of which in turn produces four more children, so that the width of the tree expands in the direction of younger generations. Each child is connected to one parent, so that the communication bandwidth of the tree remains constant, irrespective of generation. All the children of any parent are equally easy to communicate with. This is the advantage of a treelike architecture—at a given level, devices may be arranged in any arbitrary order because all arrangements are equally efficient. However, same-branch children cannot communicate directly with each other but must pass messages through their parent. Furthermore, if two same-branch children want to speak to a grandparent, then communication must flow through a single node (the parent), and so the children must communicate in series, rather than in parallel. Even worse, if the line of communication between a parent and grandparent is broken, then communication to a whole branch of the family tree is cut off. In a fat tree all of these problems are avoided. Each single-parent node is replaced by several nodes, and communications between levels of the tree occur through crossbars that connect multiple nodes at each level (and can communicate with other levels as well). Connectivity between the various levels is determined by the amount of bandwidth necessary for local (same level) or long distance (level-to-level) communication. The fat tree shown in Fig. 1B has been constructed with a higher bandwidth at the lowest level, and less bandwidth at the next level up. Large communication bandwidth is critical for both parallel computation and for defect tolerance. If one of the wires or nodes in the fat tree were blocked or damaged, communication among the remaining elements would only be slightly affected.

Rent's rule is an empirically derived guideline that may be used to determine the minimum communication bandwidth that should be included in a fat-tree architecture (*20*). Rent's rule states that for the realistic circuits people actually build (not random graphs), the number of wires coming out of a particular region of the circuit should scale as a power of the number of devices (n) in that region, ranging from $n^{1/2}$ to $n^{2/3}$. These are the exponents that one would intuitively expect if designers were constrained to build in two-dimensional ($n^{1/2}$) or three-dimensional ($n^{2/3}$) space and be as efficient as possible. For the crossbars of Teramac, exponents ranging between 2/3 and 1 were used, and thus significantly more bandwidth than required by Rent's rules was incorporated into the fat tree. This bandwidth is much higher than is normally used for a standard architecture computer or even a CCC, but it provides a great deal of extra communication capacity and redundancy in the system. This extra capacity is critical for the operation of a defect-tolerant machine, and will be revisited below. Given the framework of the fat-tree architecture and the communication guidelines suggested by Rent's rule, how are the available resources assembled to create a computer? At the top of the Teramac fat tree are the FPGAs that communicate globally, and at the bottom are the FPGAs that are used as logic elements. Everything in-between is determined by these two extrema. To see this, it is instructive to work up through the fat tree of Teramac, beginning at the LUT level (Fig. 2).

As mentioned above, logic operations are performed with the LUTs, which are essentially memories with six-bit addresses. Any Boolean function with six input variables can be stored in such a truth table. The function is evaluated simply by looking up the answer. In principle, any of the LUTs may be wired to any other LUT to execute arbitrarily complex instructions. For a given configuration of Teramac, LUTs may comprise less than 10% of the total silicon area used (*21*). The rest of Teramac's active resources are devoted to communication. The detailed map of Teramac's fat-tree hierarchy in Fig. 2 shows that most of the resources are devoted to communication among the various LUTs, between adjacent levels of the computational hierarchy, and between the computer and

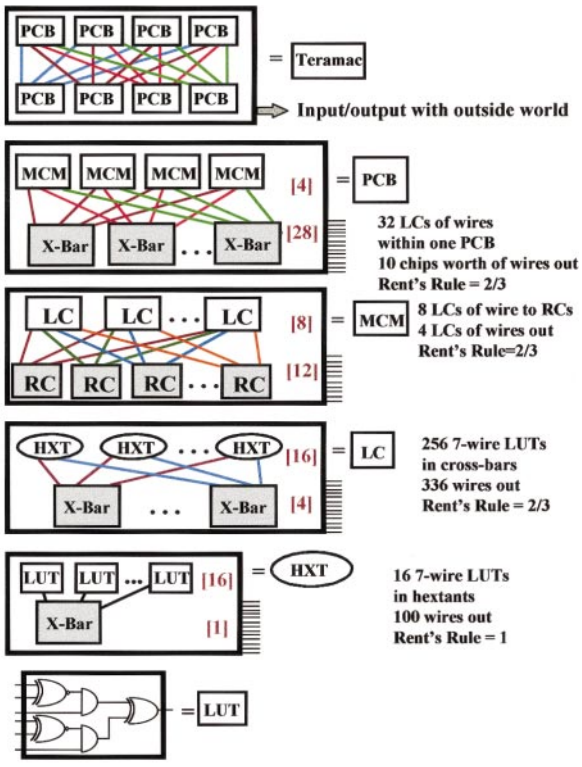

Fig. 2. The logical map of Teramac. An example of a six-input logic element is shown (bottom). In Teramac, logic is performed with memory, rather than with a central processing unit (CPU), and the results of various logic operations are stored in a look-up table (LUT). Sixteen of these LUTs are connected to each other through a crossbar (X-bar) to make up a hextant. The number of wires leaving the LUTs is equal to the number leaving the hextant, and this represents a Rent's rule exponent of 1. Sixteen hextants communicate through four crossbars to make a logic chip (LC), with a Rent's rule exponent of ~2/3. This exponent can be calculated as follows: Each LC contains 256 (=16 × 16) LUTs with 7 wires each, for a total of 7 × 256 = 1792 effective devices. The number of wires leaving the LC is 336, which is larger than $1792^{2/3}$. Each multichip module (MCM) contains eight LCs that communicate with each other and with other MCMs through 12 FPGAs used as routing chips (RCs). Because an MCM contains eight LCs worth of wires, and four RCs worth of wires leave each MCM, this also represents a Rent's rule exponent of ⅔ ($4 = 8^{2/3}$). The next level is the printed circuit board (PCB), which consists of 4 MCMs communicating through 28 crossbars (which are physically contained in the 7 FPGAs per MCM that have not yet been used). The PCB level is also characterized by a Rent's rule exponent of 2/3. Finally, all eight PCBs are wired together with ribbon cable connections to make Teramac. A large number of remaining crossbars on the MCMs are used for communication (I/O) between Teramac and the outside world. To configure Teramac, a very large (300 megabit) instruction word is downloaded through the I/O connections onto this logical graph. The instruction word sets the various configuration bits and LUTs so that Teramac becomes a custom computer.

the outside world. As an illustration of the concept, Fig. 3 shows how a simple calculation can be performed with registers and LUTs connected by a fat-tree network.

Teramac has been successfully configured into a number of parallel architectures and used for extremely demanding computations. In one configuration, Teramac was specifically designed to translate magnetic resonance imaging data into a three-dimensional map of arteries in the human brain. In another use, it was configured as a volume visualization engine referred to as the Cube 4 architecture (*22*). In one particularly efficient version of Teramac (*23*), it was actually operating at 10^{12} gate operations per second. The first configuration that was loaded onto Teramac was that of a machine that could test itself. It is this configuration that located and cataloged the defective hardware that was part of Teramac.

Defect Tolerance

Teramac was so complex and difficult to build that it was not economically feasible to construct it perfectly. A conscious decision was made to build the computer cheaply by using defective components and assembly techniques and then to compensate afterward (by programming in the field) for the mistakes. Most previous defect-tolerance work in theory (*24*) and practice (*25*) has been concerned with chip or wafer scale integration, but for Teramac, the entire machine was designed to be defect tolerant. It is thus the largest defect-tolerant computer ever built, and it strained the capabilities of available commercial technology. Each multichip module (MCM) had 33 layers of wiring to interconnect a total of 27 chips, 8 used for their LUTs and 19 for only their crossbars. Each printed circuit board (PCB) had 12 layers of interconnects for four MCMs. The interconnects for the eight PCBs that comprised Teramac were inexpensive and defect-prone ribbon cables with insulation-displacement connectors. The huge communication bandwidth incorporated to make the compiler work efficiently also forced the limits of the chip, MCM, PCB, and cable levels of interconnect, each of which contained about 10 km of wire. However, Teramac was built relatively cheaply because the fat-tree architecture is also intrinsically tolerant of manufacturing and assembly defects. There are very many reasonable ways to configure Teramac because of the multiple equally good choices for the compiler to route between any two LUTs. Adding defect tolerance to the system essentially involved avoiding those configurations that contained unreliable resources.

The use of defect tolerance in Teramac saved substantial cost for the system. Only 217 of the FPGAs used in Teramac were free of defects; the rest (75% of the total used) were free of charge, because the commercial foundry that made them would normally have discarded them. Half of the MCMs failed the manufacturer's tests, so they were also free. This represents a substantial cost saving compared to building Teramac from perfect components. The initial high cost in the redundant wiring used in the FPGAs was more than recovered by the fact that most of the FPGAs were free but still usable because of the high level of defect tolerance designed into the total system. The tests determined that 10% of the logic cells in the FPGAs used as processors were defective, and that 10% of the interchip signals were unreliable. Out of a total of 7,670,000 resources in Teramac, 3% were defective (*26*). The increase in functionality that was realized with inexpensive (or free) components was significantly greater than the cost of designing and building the defect avoidance capability. Furthermore, if Teramac is physically damaged (a chip is removed, or a set of wires cut, for example), it can be reconfigured and resume operation with only a minor loss in computational capacity (roughly proportional to the fraction of damaged parts).

For most computers, a defective chip or connection must be physically repaired or replaced for the system to be operational. For Teramac, all "repair" work was done with software. A program was written to locate the mistakes and create a defect database for the compiler. Teramac was connected to an independent workstation that performed the initial testing, but in principle a CCC can be configured into a machine that tests itself. The testing process can be separated into running configurations that measure the state of the CCC, and a set of algorithms that are run on these measurements to determine the defect. LUTs were connected in a wide variety of configurations to determine if a resource (switch, wire, or LUT) was reliable or not. If any group failed, then other configurations that used the resources in question in combination with other devices were checked. Those resources found in the intersection of the unreliable configurations were declared bad and logged in a defect database. The actual testing was performed by downloading designs onto Teramac called "signature generators." These are sets of LUTs that generate long pseudo-random number strings that are sent around Teramac by a large number of different physical paths. If the bit stream was both correctly generated and transmitted by the network, all the resources used are probably (but not always) good. The bit sequences were designed to diverge exponentially in time after an error in computation, and so this is an especially sensitive detector for bad resources. This procedure is designed to find the physical defects, such as opens, shorts, and stuck at 1 or 0, which is much easier than finding a logic design error.

There is an obvious problem in having a device test itself when one does not know whether anything is working. How do you trust the testers? In practice, only a small subset of resources have to be perfect. For the FPGAs about 7% of the chip area, for the MCMs about 4% of the wires, and for the PCBs about 7% of the wires could not

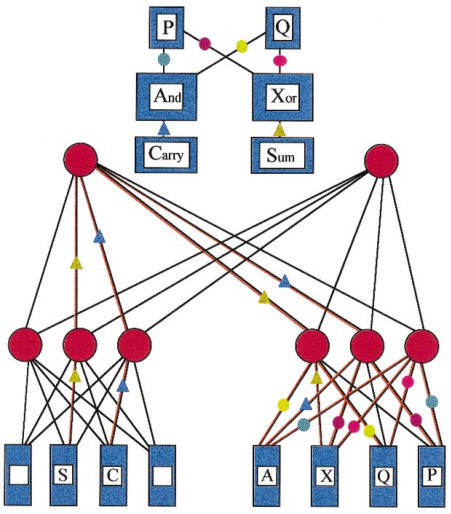

Fig. 3. This figure demonstrates how a particular implementation of a custom configurable computer is downloaded onto a given set of resources, and how the crossbar architecture, with sufficient bandwidth, allows for defect-tolerant computation. The blue boxes at the bottom are logic elements or memory (or both). The role of this system is to add two bits, P and Q, together to produce a Sum (S) and a Carry (C). When P and Q are the inputs to an And gate (A), then the output is the Carry. When they are inputs into an Xor gate, then the output is the Sum. Thus, both P and Q must be connected to both A and X, and the ouput of A and X must be connected to the memory locations for S and C, respectively. The red circles are crossbars, and there are two levels to this fat tree. This particular logical implementation illustrates how various components with widely varying numbers of defect can still be used to construct a working system. From the bottom left crossbar, and proceeding clockwise, we use 20, 70, 0, and 80% of the available resources. Similar arguments can be made for the other components. To understand this system more completely, it is advisable to reassign the look-up tables differently, define some of the crossbar switches to be defective and thus removed from the available resources, and then reconnect the system to enable the adder. Such an exercise is very similar to what the compiler does when it downloads the logical architecture onto the available resources.

have defects in Teramac. These are the wires that are used for clocks and to get data out of the system for observability. Furthermore, some small percentage of resources must be working to guarantee that the defect-finding algorithms would work if the system was to test itself. Those resources that were part of this privileged set were deliberately designed with explicit additional redundancy to ensure that they had a high probability of survival.

Once the defect data base had been established, computer architectures could be loaded onto Teramac. The presence of defects makes this task more difficult than for a perfect system, but because of all the extra bandwidth that resulted from using interconnects that exceed Rent's rule exponents in the fat tree, it turned out to be surprisingly easy to do. In any given configuration of Teramac, only 70 to 90% of the healthy resources are actually used. However, such inefficiency is a relatively inexpensive cost associated with the defect tolerance. Scaling properties are very important for any architecture that aspires to eventually have moles of components. The compiler algorithms are dominated by the partitioning time, which scales linearly with the number of gates in the design (*27*). Experiments with various-sized partitions of Teramac showed that the time required to find the defects also scaled linearly with the total number of wires and switches in the fat tree. This empirical result is extremely important, for if the scaling had been superlinear, the extension of this architecture to extraordinary numbers of components would not be so promising. The explicit effect of defects on the scaling properties are still issues of active research, but there does not appear to be any significant scaling penalty.

Lessons for Nanotechnology

The ability of Teramac to operate reliably in the presence of large numbers of defects shows that a CCC architecture is applicable to, and may be essential for, computational nanotechnology. As perfect devices become more expensive to fabricate, defect tolerance becomes a more valuable method to deal with the imperfections. Any computer with nanoscale components will contain a significant number of defects, as well as massive numbers of wires and switches for communication purposes. It therefore makes sense to consider architectural issues and defect tolerance early in the development of a new paradigm. The Teramac design and assembly philosophy differs significantly from the usual ideas of building complex computer systems, and thus there are several important lessons for nanotechnology.

The first lesson is that it is possible to build a very powerful computer that contains defective components and wiring, as long as there is sufficient communication bandwidth in the system to find and use the healthy resources. The machine is built cheaply but imperfectly, a map of the defective resources is prepared, and then the computer is configured with only the healthy resources. At present, such an approach is not economically competitive with CMOS technology, which requires perfection in all the components of a computer, because so many of the resources in a CCC are not used (for example, most of the LUTs in Teramac). However, the cost of the fabrication plants for integrated circuits (Fabs) is escalating exponentially with time as chips continue to shrink in size, an observation that is sometimes called Moore's second law (*2*). By the year 2012, a single Fab could cost $30 billion (*1*) or more, which may simply be too expensive and risky to build. At the same time, the sophistication of inexpensive chemically synthesized components is increasing dramatically. There may eventually be a crossover from one manufacturing paradigm to another, and the defect tolerance possibilities raised by Teramac could be the key enabling economic issue that ushers in the era of chemically assembled electronic computers.

A second and related lesson from Teramac is that the resources in a computer do not have to be regular, but rather they must have a sufficiently high degree of connectivity. The wiring mistakes in the MCMs introduced a significant element of randomness to the connectivity of the system, such that it was not possible to know what resources were connected together without performing a test. Thus, it is not essential to place a component at a specific position as long as the components can be located logically. A crude analogy here is the comparison between the American and the Japanese post offices. If residences are laid out in a Cartesian coordinate system, then it does not take much complexity in the mail-delivery system to find an address. In Japan, however, there are no regular street addresses. Nevertheless, the knowledge of many local postmen is sufficient to deliver a letter. A system at the nanoscale that has some random character can still be functional if there is enough local intelligence to locate resources, either through the laws of physics or through the ability to reach down through random but fixed local connections.

The third lesson addresses the issue of what are the most essential components for an electronic nanotechnology. In Teramac, wires are by far the most plentiful resource, and the most important are the address lines that control the settings of the configuration switches and the data lines that link the LUTs to perform the calculations. In a nanotechnology paradigm, these wires may be physical or logical, but they will be essential for the enormous amount of communication bandwidth that will be required. Next, in terms of the number of elements, are the crossbar switches and the configuration bits that control them. This may well be the most important active device that will be needed for computational nanotechnology. One possible physical implementation of a crossbar switch is illustrated in Fig. 4, although this example should not be viewed as restrictive. The replacement of the six transistors required by an FPGA for a single configurable bit by one quantum dot that may require only a single electron to change its state would represent an enormous energy saving for a bit operation. This would represent a tremendous advance toward the thermodynamic limit for a nonreversible machine. The LUTs make up less than 3% of the fat-free utilizable resources of Teramac. As such an architecture is scaled to significantly larger sizes, that percentage will decrease because there will be more levels added to

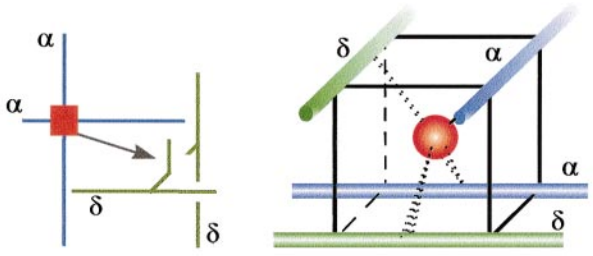

Fig. 4. An idealized version of the chemically fabricated configurable bit (right), compared with the logical description of a configurable bit redrawn from Fig. 1. The components labeled α and δ are the address lines and data lines, respectively. The orange component (β) is the switch. The address lines are used to locate and "set" the bit. Once the bit is set, the connection between the two data lines is shorted, and thus the status of the bit may be read. A chemically fabricated switch could consist of a single semiconductor quantum dot in capacitive contact with the two address wires. The dot is also in tunnelling or ohmic contact with two data wires. Ligands that connect the dot to the four wires are varied to control the nature of the contact. Operationally, this switch is a dual-gated single-electron transistor (*5*). When the two address lines are biased "on," the quantum dot is shifted out of the Coulomb blockade voltage region, and the data lines are effectively shorted.

the hierarchy. Because the logic operations are in the form of LUTs, there may be no need for traditional logic gates in a computer; it may require only wires, switches, and memory. Teramac already has half a femtomole of configuration bits, and as the availability of chemically assembled memory devices and switches approaches a mole, memory-only computers will become even more attractive.

The fourth lesson of Teramac relates to the division of labor for building a computer. The conventional paradigm for computation is to design the computer, build it perfectly, compile the program, and then run the algorithm. The Teramac paradigm is to build the computer (however imperfectly), find the defects, configure the resources with software, compile the program, and then run it. This new paradigm moves what is difficult to do in hardware into a software task, which is just the continuation of a trend that has accompanied the development of electronic computers from their first appearance. A large fraction of the expense in fabricating present-day microprocessors is in building multibillion dollar Fabs, where the cost of making perfect devices requires ever-increasing cleanliness and precision. However, another model is to fabricate cheap processors by the techniques of chemical assembly, and then train the devices up to the desired level of proficiency with computer tutors that find the defects and record their locations in databases. In this model, all chip equivalent devices are made in the same manner, but their ultimate capacity (and price) depends both on the number of random defects that accumulate during their fabrication and on the amount of training they receive (a combination of nature and nurture).

In summary, Teramac illustrates an entirely different mode, essentially a top-down approach, of research for nanotechnology. By examining an architecture that will be compatible with nanoscale fabrication, one may then be able to identify the types of components that are most crucial for implementing that architecture. If this argument is valid, then wires, switches, and memory are the key enabling ingredients for computational nanotechnology, and a configuration bit appears to be the most enticing initial target for research in active devices (*28*).

REFERENCES AND NOTES

1. The 1997 National Technology Roadmap for Semiconductors is a publication of the Semiconductor Industry Association (www.semichips.org). The Roadmap costs $250. Distribution is being coordinated by SEMATECH, based in Austin, Texas. Copies may be obtained on-line by contacting SEMATECH's home page (www.sematech.org).
2. R. R. Schaller, *IEEE Spectrum* **34**, 53 (June 1997).
3. R. Landauer, *IBM J. Res. Dev.* 3, 183 (1961).
4. An especially good introduction to information science for physical scientists is R. P. Feynman, *Lectures in Computation*, A. J. G. Hey and R. W. Allen, Eds. (Addison-Wesley, Menlo Park, CA, 1996).
5. R. H. Chen, A. N. Korotov, K. K. Likharev, *Appl. Phys. Lett.* **68**, 1954 (1996).
6. P. D. Tougaw and C. S. Lent, *J. Appl. Phys.* **75**, 181 (1994).
7. W. A. Caldwell *et al.*, *Science* **277**, 93 (1997).
8. C. Mead, *Proc. IEEE* **78**, 1629 (1990).
9. J. J. Hopfield and D. W. Tank, *Science* **233**, 625 (1986).
10. A. Aviram and M. Ratner, *Chem. Phys. Lett.* **29**, 277 (1974).
11. M. C. Petty, M. R. Bryce, D. Bloor, Eds., *Introduction to Molecular Electronics*, (Edward Arnold, London, 1995).
12. The assembly of the machine is chemical but the operation of the machine is electronic. This is to be distinguished from the DNA computers and others in which computation proceeds by way of chemical reactions, as discussed in L. M. Adelman [*Science* **266**, 1021 (1994)].
13. W. B. Culbertson, R. Amerson, R. J. Carter, G. S. Snider, P. J. Kuekes, *Proc. IEEE Symp. FPGA's for Custom Computing Machines* (1997).
14. We distinguish between "defect tolerant," the capacity of a machine to operate perfectly in the presence of errors made in the hardware during manufacture, from "fault tolerant," the ability of a machine to recover from errors made during a calculation. Fault tolerance is an active research area and will also be required for computational nanotechnology. However, we do not discuss fault tolerance in this article.
15. The Teramac approach is actually more conservative than other massively parallel defect tolerant architectures that have been proposed. Once the defective components have been found, we propose to construct a fairly conventional computer with a standard instruction set out of the good components. Others propose to use cellular automata or neural nets to achieve robustness in the presence of defects. These approaches depend on significant progress in algorithm development through use of cellular automata or neural nets (*6–9*).
16. An excellent recent discussion of configurable computing can be found in J. Villasenor and W. Mangione-Smith, *Sci. Am.* **276**, 68 (June 1997).
17. C. E. Leiserson, *IEEE Trans. Comput.* **C34**, 892 (1985).
18. L. R. Goke and G. L. Lipovski, in *Proc. First Annual Symposium On Computer Architecture* (IEEE, New York, 1973), pp. 21–28.
19. HP scientists chose a fat-tree architecture with larger than necessary Rent's rule exponents to enable fast compilation.
20. Rent's rule was discovered by Richard Rent at IBM in the 1960s. Although it was never published by Rent, subsequent work has confirmed its validity [B. S. Landman and R. L. Russo, *IEEE Trans. Comp.* **C20**, 1469 (1971)].
21. This small percentage of area devoted to logic holds true for all VLSI chips built today.
22. U. Kanus *et al.*, *11th Eurographics Workshop on Computer Graphics Hardware*, B.-O. Schneider and A. Schilling, Eds. (IBM, New York, 1996).
23. This version of Teramac was running a DNA string matching algorithm. For further detail on this algorithm, see R. J. Lipton and D. P. Lopresti, in *1985 Chapel Hill Conference on VLSI*, H. Guchs, Ed. (Computer Science Press, Maryland, 1985), pp. 363–376.
24. N. J. Howard, A. M. Tyrrell, N. M. Allinson, in *IEEE Transactions on VLSI Systems* **2**, 115 (1994).
25. J. Otterstedt, H.-J. Iden, M. Kuboshek, *Proceedings of the 1994 International Conference on Wafer Scale Integration*, R. M. Lea and S. Tewksbury, Eds. (IEEE, New York, 1994), pp. 315–323.
26. We estimate that Teramac could have contained up to 50% defective wires and still have operated successfully, although the question of how many defects can be tolerated in a Teramac-like system is a subject of investigation. One critical aspect of the crossbar architecture is that if switches fail in the "open" state, then many more defects can be tolerated. Switches that fail "closed," however, will render an entire row or column of associated gates inoperable.
27. B. Krishnamurthy, *IEEE Trans. Comput.* **C-33**, 438 (1984).

28. We thank W. Robinett for helpful comments on the manuscript. J.R.H. acknowledges support from an NSF-GOALI grant and the Hewlett Packard Corporation during the writing of this manuscript.