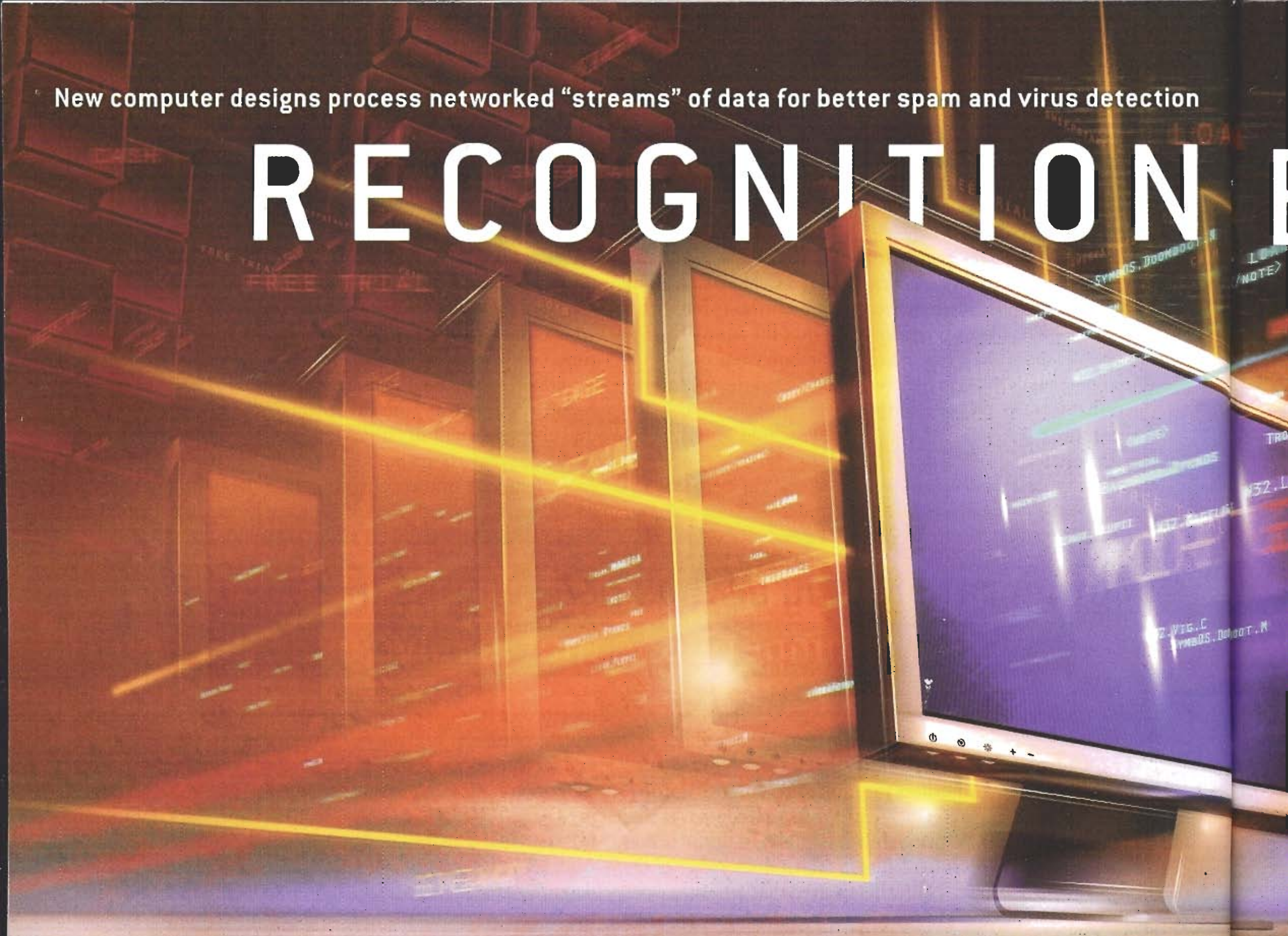
New computer designs process networked "streams" of data for better spam and virus detection

RECOGNITION



The computer industry survived for much longer than it should have on the assurance that faster processors every few years would cover a multitude of sins—the inefficiency and bloated size of application software being some of the worst transgressions. That luxury appears to be fading as power consumption has skyrocketed and the circuit boards on which the microprocessors sit threaten to transmute themselves into space heaters. Intel (where the hallowed Moore's law has reigned) and other hardware makers have responded by designing computers to run multiple processors at slower speeds.

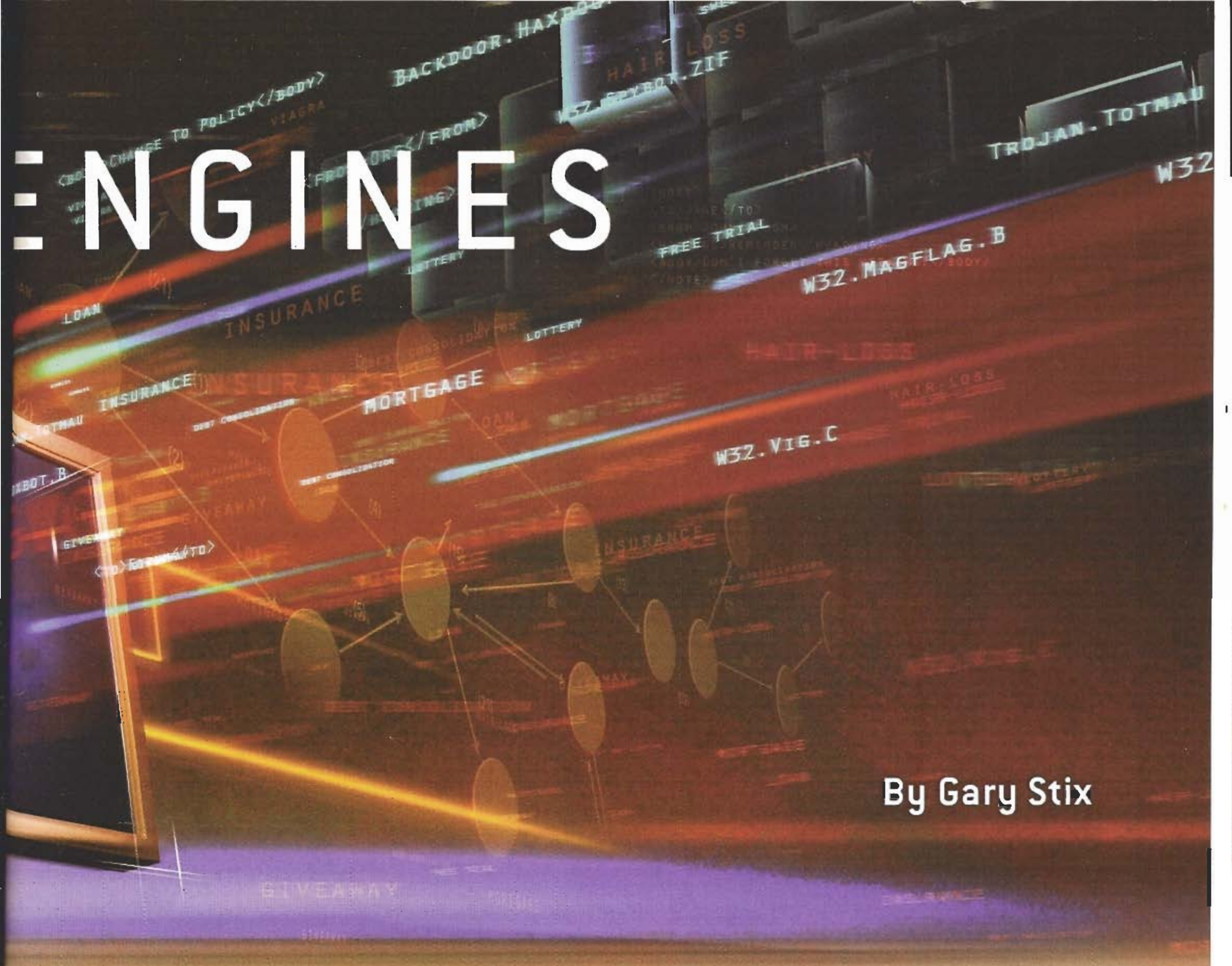
Multiprocessors come with their own baggage, however. First, writing software that apportions computational tasks among several processors remains an unwanted burden for many programmers. Moreover, a number of the fastest-growing networking applications—from virus scanning to reading Web documents encoded in extensible markup language (XML)—do not lend themselves readily to parallel processing.

Determining whether a message contains a word that denotes spam, such as "lottery" or "Viagra," requires evaluation of consecutive parameters: Is the word "lottery" followed later in the document by the word "payout"? Distributing this task among an array of processors is asking for trouble. Instead engineers have begun to embrace a more specialized role for coprocessors. The main microprocessor retains its responsibility as chief dispatcher for key operating system functions. Meanwhile designs for processors that perform spam and virus hunting or XML processing have taken a page from graphics processing, which has long had its own specialized units. In recent years, so-called intrusion-detection accelerator engines have often taken over some of the work from increasingly overburdened central processing units (CPUs). A few academic and industrial laboratories have even begun to advance this concept one step further by accommodating all types of "streamed" information that move over a network. In essence, they have created a general-purpose stream processor that can be readily reprogrammed and can handle multiple applications, whether it be guarding a firewall or compressing files.

Pattern-Matching Engine

THE IBM ZURICH Research Laboratory has netted Nobel Prizes for the creation of the scanning tunneling microscope and high-temperature superconductivity. It has also served as a nexus for developing network hardware and software. At Hot Chips, a conference put on by the Institute of Electrical

RYAN ETTER



ENGINES

By Gary Stix

and Electronics Engineers that took place in August 2005 at Stanford University, Jan van Lunteren of IBM Zurich gave a presentation on a stream processor—a "pattern-matching engine" for catching viruses, spam and other bad actors—that he developed along with colleague Ton Engbersen.

IBM's processor emerged from earlier research on how to move data through the Internet's network computers, called routers. Van Lunteren, a native of the Netherlands, worked during the late 1990s at IBM Zurich on improving techniques for efficiently looking through the data tables of routers to find the forwarding information for data packets traveling through a network. Routers have to examine tens of millions of packets a second and inspect tens of thousands of entries in their databases to procure the next link on the network before sending packets out of one of multiple output ports. Van Lunteren devised a hash function for searching routing tables. This mathematical formula produces a number, or hash index, that indicates in a table lodged in the processor hardware where the relevant output port is that connects to the link that in turn will move the packet to the next router on the network.

Van Lunteren designed an algorithm based on a hash function—the Balanced Routing Table, or BaRT, search—that compresses dramatically the number of bits needed to store the routing tables in memory. BaRT, which could find its way into a number of IBM products, can handle 25 million packets a second and might eventually take care of four times that amount of data traffic.

Routing table searches require only a look at a short string of data in the initial part of the packet, the header that tells a packet where to go. With the avalanche of spam, viruses and other so-called malware, however, network processors now also have to read much more deeply inside the contents of the packet itself for telltale signs that a sender is up to no good. Similarly, reading document-encoding languages such as XML also places high demands on network hardware. The hash function that van Lunteren devised for routing became essential for IBM's stream processor.

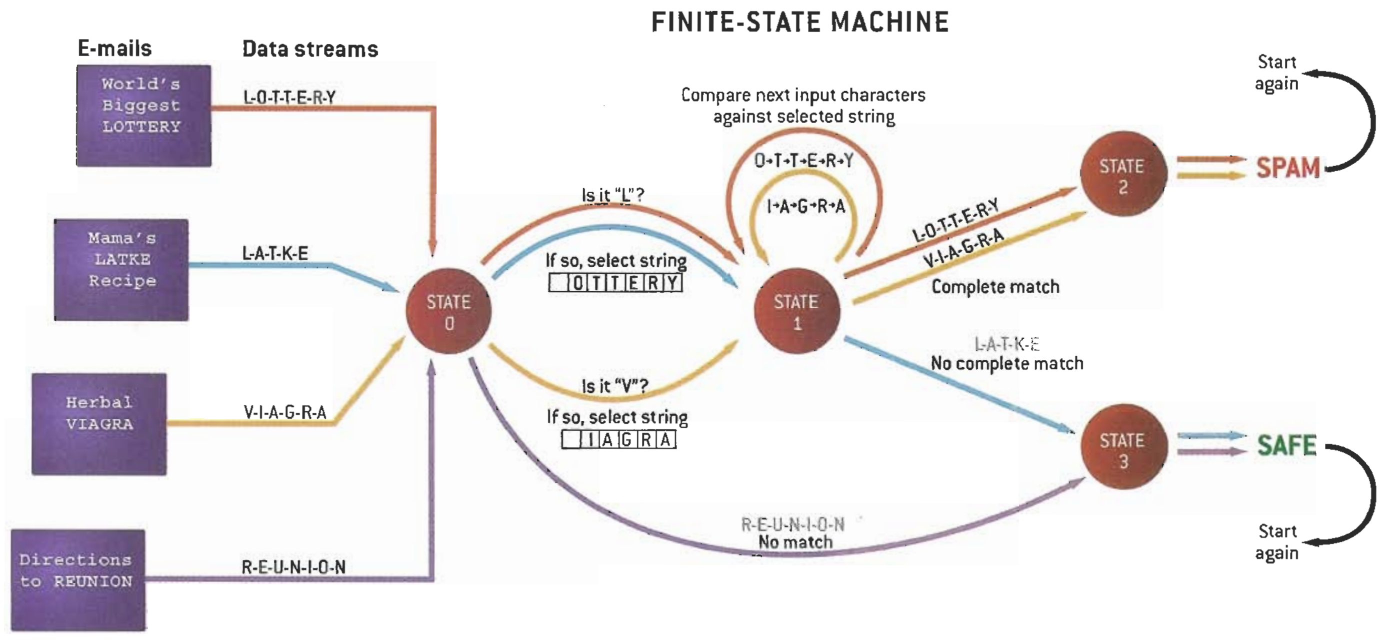
Beyond von Neumann

CONVENTIONAL PROCESSORS require multiple instructions to deal with XML codes or to look for malware, creating a bottleneck in which tens of clock cycles are needed to handle a single character. Despite many refinements, the average CPU still relies largely on an architecture initiated in the 1940s by the great mathematician John von Neumann as well as computer pioneers J. Presper Eckert and John Mauchly.

MATCHING MANY VS. COMPARING ONE BY ONE

Finite-state machines process data streams by matching each input character simultaneously against many different characters indicative of spam that are stored in memory. A conventional von Neumann machine, in contrast, must evaluate the characters stored in memory one by one.

In state 0, the finite-state machine initially compares character "L" against two others, "L" and "V," to determine whether it can be the first letter of "LOTTERY" or "VIAGRA," two stored words that denote spam. Once a match occurs, the machine switches to state 1, checking successive input characters against a stored character string, either "OTTERY" or "IAGRA." If it finds a complete match for one of the two strings, the machine moves to



The von Neumann architecture, as it is called, fetches an instruction from an address in memory and executes it, and a program counter is updated with the address of the next instruction to be performed. The cycle then repeats itself, unless explicitly told otherwise by an instruction that requires the processor to jump to another place in the program. If the processor confronts a task with any degree of complexity—for instance, evaluation of whether a particular character is legal in XML coding—it must grind through multiple instructions and clock cycles to complete the task.

Van Lunteren and Engbersen borrowed a conceptual scheme from the earliest years of computing, a finite-state machine that is rooted in the work of computing pioneer Alan M. Turing. A finite-state machine is a basic description of how any computing machine operates: how it performs operations in a discrete series of steps and assumes a finite list of internal states at any one time. On an abstract level, even the von Neumann architecture can be characterized as a finite-state machine. But the type of finite-state machine designed by van Lunteren and Engbersen distinguishes itself from a CPU that relies on the von Neumann architecture because it forgoes inclusion of a program counter.

Unlike the von Neumann namesake, van Lunteren and Engbersen's finite-state machine can evaluate multiple things simultaneously in a single cycle, instead of considering just one, as happens in the process that is controlled by the program counter. That is one of the reasons finite-state machines have been deployed for years in graphics processors and voice-recognition systems and in hardware design. Finite-state machines, however, have not lent themselves to being reprogrammed readily, thus sacrificing the flexible, general-purpose quality of the von Neumann–based CPU.

Yet the bottleneck posed by the sequential nature of conventional CPUs has begun to diminish some of the distinctions between the two types of processors. The IBM finite-state hardware, for one, can be reprogrammed with a software update if new viruses proliferate or if the XML standard changes.

The design of van Lunteren and Engbersen's processor relies on a state diagram, a type of graph that consists of circular nodes, or states, and links between nodes that represent transitions from one state to another. A subway turnstile is a form of finite-state machine. Its initial node is a state called "locked." Insertion of a coin is indicated in the graph by a line that traces a "transition" from the current state to an "unlocked" node. Passing through the turnstile is another line that marks a transition back to the locked node.

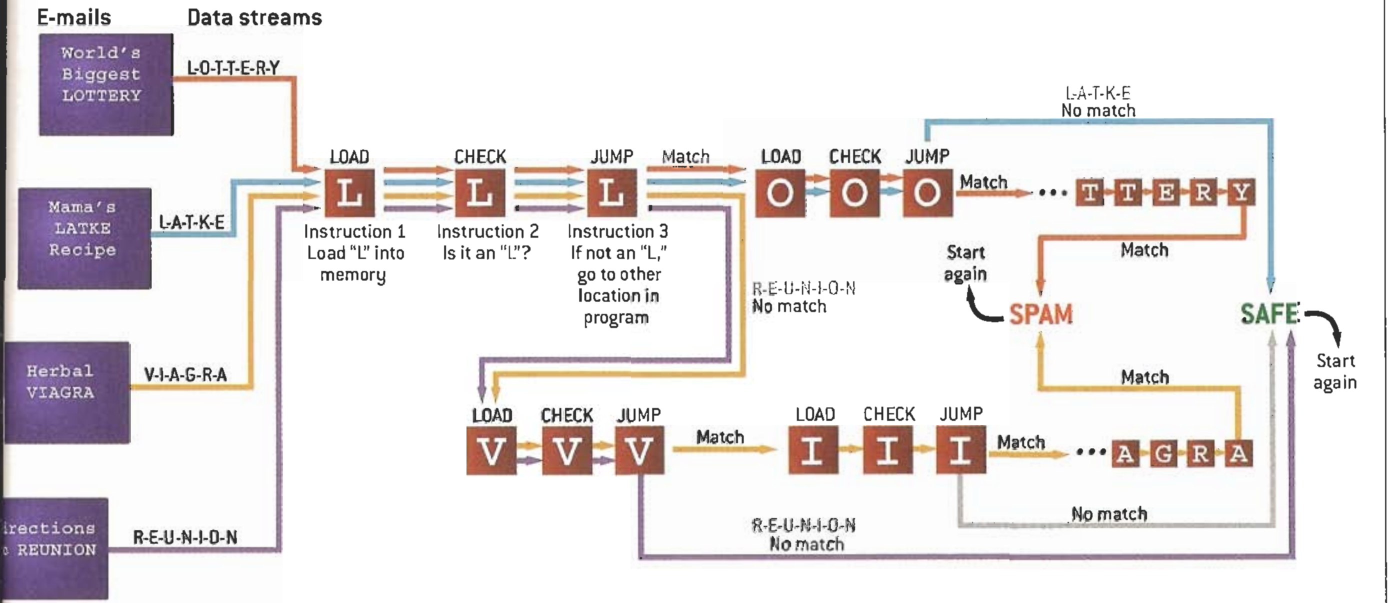
In IBM's finite-state machine, a given state may link more than two nodes. In an actual stream-processing application, a node might have links to many others, and each link would be assessed at the same time before a decision is made to move to the next state in the diagram. In looking for spam in an incoming stream of data, the processor would read from

MELISSA THOMAS

state 2, indicating detection of a word found in spam messages. If no match occurs, as with the word "LATKE," the hardware shifts to state 3, suggesting that no spam is lurking. If the initial input letter does not jibe with words in memory, such as the "R" in "REUNION," the machine proceeds directly from state 0 to state 3.

In the typical von Neumann architecture, each input character can be tested only against one stored character at a time. Moreover, three and sometimes more instructions, and thus multiple processor cycles, are required for each character—one to load the character, another to check whether it is the desired character, and the third to jump to another location in the program if it is not the character being sought.

VON NEUMANN ARCHITECTURE



memory the word "lottery." It could evaluate not only whether the character "o" followed an "l" in an arriving string of characters but also whether a spam message may have inserted an underscore character—"l_o"—to try to fool a spam blocker. As part of the same search, executed in a single processor cycle, it could look for the "l" in "lottery" as well as for the "V" in "Viagra" and many other characters in its memory. In a conventional processor, each of those steps would have to be executed sequentially [*see box on these two pages*].

At least in the laboratory, applying a finite-state machine to streaming applications improves performance substantially. Van Lunteren reported at Hot Chips that IBM's finite-state machine can process characters at up to 20 gigabits a second for viruses, spam and other applications, 10 to 100 times faster than a conventional processor. A key enabling tool was BaRT. In many finite-state machines, storing rules for carrying out the transitions in a state diagram consumes a large amount of memory. IBM can store in its finite-state machine hardware up to 25,000 characters in less than 100 kilobytes of memory, as little as 1/500 the requirements for some other finite-state machines. The efficiency of the algorithm devised originally for routing tables allows for a linear increase in memory needs: if the number of transition rules rises from one to 10, memory demands go up by a comparable factor. In many other finite-state machines, a similar increment would require 100 times more space.

IBM already offers the finite-state machine technology for custom applications—licensed through its engineering and technology services group—and it is evaluating the processor for a number of products. IBM is not alone in adopting this idea. Universities and other companies have also developed programmable finite-state machines. John Lockwood, a professor at Washington University in St. Louis, co-founded a company called Global Velocity to commercialize such a processor. Van Lunteren says that the IBM design is special because of its ability to handle a wide range of applications, becoming a general-purpose processor for any stream-processing application. The sophistication of these coprocessors may continue to evolve as critical tasks such as stream processing stray further and further from the control of the central processing unit. This work ensures that the legacy of Turing and von Neumann will coexist a few centimeters away on the same circuit board. SA

MORE TO EXPLORE

The Alphabets, Words and Languages of Finite State Machines. This explanation of how finite-state machines work can be accessed at **www.c3.lanl.gov/mega-math/workbk/machine/mabkgd.html**

Global Velocity is a company that has developed similar processing concepts to the IBM team: **www.globalvelocity.com/index.html**

XML Accelerator Engine. Jan van Lunteren, Ton Engbersen, Joe Bostian, Bill Carey and Chris Larsson. First International Workshop on High Performance XML Processing, May 18, 2004. Available online at **www.research.ibm.com/XML/IBM_Zurich_XML_Accelerator_Engine_paper_2004May04.pdf**