THE FIFTH GENERATION

IN SEARCH OF 第五代

Kyoko Okamoto

LAST OF THREE ARTICLES

By JANET RALOFF

If logic programming is to be the soul of Japan's fifth-generation computer, through what corporeal manifestation will it take form? No one knows. Not the Japanese, Americans or Europeans. What's being sought is not only a blueprint for configuring the neural network that will link myriad cells within this electronic brain, but also an analysis of what specific types of cells will be present and where.

Contributing to the difficulty of this design task are the lofty performance targets that have been set for these machines: among them, the ability to execute as many as one billion logic inferences per second (a gigalips) and the ability to efficiently shuffle through huge stores of knowledge — files expected to hold between 10 billion and 100 billion bytes of information. (A byte consists of 8 bits — adjacent binary digits — operated on as a unit by the computer.)

The sheer magnitude of this anticipated data and inferencing is challenging computer architects' ability to conceptualize what must be not only workable but also efficient hardware configurations. Their designs must provide for communications between discrete electronic components, control the allocation of processing tasks between functionally related communities of such components, and simplify the number of programming instructions that will be required to execute those logic-oriented operations — such as searching and backtracking — which are expected to typify the way these machines solve problems (SN: 6/2/84, p. 346).

About the only thing all fifth-generation architects agree on is that the machinery must be designed to exploit *parallel processing* (italicized phrases appear in glossary, p. 379), the concurrent execution of several computer operations. "It used to be that a computer had one *processor*," explains Ross Overbeek at Argonne National Laboratory, near Chicago, meaning that serial, also known as *sequential processing*, was the rule. "But the Japanese are talking about execution rates on the order of 10 million to a billion inferences a second, something that I think will only be achievable through use of fairly massive *multiprocessing*," he says, using many individual processors.

There are several ways to accomplish parallel processing. For example, one machine being specially designed to execute *PROLOG*, a *logic-programming* language, would make creative use of a single processor. According to its architects, PROLOG-*implementation* expert David H. D. Warren and Stanford University computer architect Evan Tick, simulated program runs indicate that this machine might be capable of executing 400,000 lips at peak capacity. Right now, Warren points out, their machine is just "a paper design." In fact, Tick notes, "our design hasn't specified something that you'd actually build. We're just looking at the critical paths, seeing if it's feasible to actually build such a system, and if so, what its performance would be."

Tick is focusing on its hardware requirements. Organization is rather conventional, he says, explaining that the computer would use a single processor and large *memory*. What makes the processor "interesting," he says, is that it amounts to a three-stage pipeline. "The actual hardware would be partitioned into stages, like an actual pipe," he says; computer *instructions* "would flow through these stages."

Because of this design element, as many as three instructions could be operating simultaneously in the pipe, Tick says. However, the system has been designed so that individual sets of data can only be operated on by one instruction within the pipe at a time. To get three instructions occurring at once, all must use different data. "Studies I'm doing now are to determine how frequently the bad things would occur," he says, referring to system inefficiencies caused by data tie-ups that leave one or two stages of the pipeline temporarily empty.

Among the unique architectural details of this PROLOG machine, Tick notes, is use of "choice points" in the system's stack buffer. This buffer is sort of the equivalent of a fast memory, Tick says, for temporary storage of several sets of data or information pertaining to procedures that may be recalled in the near future. Problem solving with logic programming involves orderly searches among collections of rules. The goal is to find data that logically prove a premise. If, during these searches, a promising lead suddenly turns out to be a dead end, the program must backtrack to the last point it had a choice of leads to try, and then begin again along some new computational path. By keeping a record of the last choice point, Tick says, the program can automatically back up to that point whenever necessary, then speed on in its search.

While the degree of parallelism possible with this particular *architecture* is limited, the design is noteworthy within the context of fifth-generation systems for its attempt to optimize machine-level support of inferencing. In an interview with SCIENCE NEWS, Kazuhiro Fuchi, director of the Institute for New Generation Computer Technology (ICOT) in Tokyo, explained that in his view, logic programming was probably insufficient for achieving the inferencing performance that would be asked of fifth-generation systems. So in addition to programming, he says, "the machine should have inferencing capabilities as a basic machine operation."

At least that's how ICOT, which is spearheading Japan's fifth-generation program, will be approaching design of an initial PROLOG-based machine, he says. By way of indicating how such machine-level inferencing support might be represented, he says the binary 0's and 1's, or on/off steps, used for computing in conventional machines today might be replaced by use of a "go-to" or "don't go to" as fundamental processing operations.

Like Warren and Tick's design, Fuchi says ICOT's initial machine will probably contain a single processor. However, he says a range of designs employing many processors will also be explored for use in later phases of the ICOT 10-year program. Among those he finds most promising is "data flow," one of a class of concepts described as being "non-von Neumann."

"In von Neumann architecture [named

Science Service, Inc. is collaborating with JSTOR to digitize, preserve, and extend access to *Science News*. www.jstor.org

GIGALIPS ARCHITECTURES

Fifth-generation computer architects are exploring the design of machines for lightning-quick reason

for the mathematician who wrote the first widely circulated description of a logic framework for stored-program computers and for programming concepts (SN: 3/13/82, p. 172)], instructions are held in successive locations in a computer's memory and they are executed one at a time," explains Jack Dennis, of the Massachusetts Institute of Technology in Cambridge. Controlling the executions, he explains, "is a pointer which moves through the memory one step at a time." Though one can sometimes figure out how to execute more than one instruction at once, when using a machine with von Neumann architecture, he says "the program will always look as though only one execution were going on."

In contrast, the order in which instructions will be performed within a data flow machine is determined by the availability of the data needed for their execution, explains Dennis, who more than any other individual is credited with invention of the data flow concept. For instance, he says, "if you have an 'add' instruction and the two operands to be added have been computed, then that instruction is made available for execution." Envisioned to ultimately employ thousands of processors, any or all of the data flow machine's processors may operate simultaneously, depending on the availability of data each needs to stay busy.

Using a *functional-programming* language called VAL, Dennis and colleagues at MIT are exploring the potential of data flow machines for harnessing massive parallelism to expedite processing of huge numerical scientific computations, such as are involved with weather prediction and fusion reaction modeling. "If you wanted to solve artificial-intelligence problems on this machine [as in fact, the Japanese have said they will attempt to do] you would find it quite a challenge to use it efficiently," Dennis told SCIENCE NEWS.

The problem, explains colleague Gary Lindstrom, "is that data flow architectures are primarily oriented to functional languages — that is, toward doing computational steps in preplanned orders." So, he explains, "there is the question of how one represents the search that needs to be done in executing a logic-programming language, because it's not automatically there in the architecture" as it is in the Warren and Tick design. Lindstrom understands this problem well. At MIT on a leave of absence from the University of Utah, he is tackling — as the Japanese are — development of a scheme to implement logic programming on a data flow architecture.

"The data flow world is divided into three schools of thought," Lindstrom points out. "Dennis represents what's called the 'static'

Glossary

Architecture — the way in which a computer's hardware — physical components — are configured so as to support and interact with programs.

Controller — the part of a processing unit or system that directs the step-by-step operations required to solve a problem or that part of the problem over which it has been given control.

Expert systems — programs that apply artificial-intelligence techniques — such as rules of inference — for problem solving in a narrowly focused subject area. These systems usually derive their expertise by consulting lists of facts, rules-of-thumb and observations gleaned by querying human experts.

Functional programming — the ordered lists of functions (mathematical processes, such as calculations of the square, cosine or inverse of something) that direct sequences of computer operations.

Implementation — the representation of a programming language on a specific computer system; for example, a version of FORTRAN on a particular computer.

Instruction — the message, usually characterized by a group of characters, bits (binary digits — either 0 or 1) or groups of bits, that defines an operation a computer is to perform.

Logic programming — rule-based, logic-oriented set of instructions by which computers make inferences.

Memory — the unit(s) within a computer or integrated-circuit chip in which data are stored.

Multiprocessing — simultaneous or parallel processing of several operations at once, accomplished within a single computer through use of more than one processor.

Parallel processing — concurrent or simultaneous execution of two or more processes in the same computer.

Processor — a device or system capable of performing operations, such as addition, on data.

Program — ordered instructions that direct sequences of computer operations.

PROLOG — a rule-based, logic-programming language designed for artificial-intelligence applications such as expert systems. An acronym for "programming in logic," PROLOG is Japan's current preference for fifth-generation systems.

Sequential processing — execution of processes, such as individual additions, serially within a computer such that they occur one after the other.

data flow community, within which computers are designed primarily for intensely parallel, high speed execution of programs" of a large and essentially numerical nature. "There, in order to get the intense parallelism and speed, the functional languages used are quite highly structured." Moreover, Lindstrom notes, "the kind of flexible sequencing that is required in a logic-programming search is not really even appropriate."

The second school is "dynamic" data flow. Instead of designating beforehand which machine processors will handle which problem solving tasks, dynamic data flow architectures more flexibly schedule processors as they become available, Lindstrom says, noting that this is preferable when problems are so complicated that mapping the most efficient path to their solution becomes unfeasible.

The third data flow school represents the one Lindstrom belongs to. The reduction machine its adherents are designing is not data driven as the other data flow machines are, Lindstrom says. In data-driven machines, "pieces of the program are invoked or activated by the availability of data for execution. If you take the analogy that the program is a connection of functions, in boxes, that are connected by pipes through which data flow, then the static and dynamic data flow machines operate by pressure," Lindstrom says, "pushing data through this plumbing network." Reduction machines, with a similar network, work instead "by suction," Lindstrom analogizes. It's the overall need for a solution that propagates signals through this network, causing functions to be applied in a much more unpredictable manner. "So there is conjecture," he says, "that reduction machines might be a nice way to accomplish the parallel data-dependent search that one needs for logic programming."

David Shaw believes he's building "one of the two most parallel machines" around. Named NON VON to connote its non-von Neumann design, the Columbia University project Shaw heads in New York should produce a very small prototype that operates by year's end. It is slated to contain 64 processors, each bearing a tiny memory of 64 bytes each. A commercial-scale computer would ultimately have a million or more of these small processors, each linked to slightly larger memories. But conceptually, NON VON's most important characteristic is the fact that each processor will be paired with a single data element (stored in one or more associated memories) that's to be manipulated "so that the data elements seem as if they're capable of doing work themselves," Shaw explains.

"The static and dynamic data flow machines operate by pressure, pushing data through the plumbing network."

Underpinning NON VON's architecture is a structural configuration known as a "binary tree." In it, a single, tiny processor is connected to two identical ones beneath it, each of which is in turn connected to two more below it, and so on. A million identical processors may ultimately be connected in this manner.

At the top few levels of this organizational tree — down to where one might find 256 processors — each tiny processor would also be tied into its own additional large processor. These larger processors can work alone or can broadcast an instruction to all small processors in the subtree under and including the small processor to which it is linked. Finally, "all large processing elements are connected to each other through what we call a high-bandwidth interconnection network," Shaw says, "which means they can send a lot of information around very quickly. That's something that isn't possible between all of those small processing elements."

"Just so you know the motivation for all of this," he explains, "there's no way to connect a million processing elements to each other so that they can send data to each other quickly. But what you can do is have some small number, like 256, talk to each other." Because the small processors don't have enough memory to store individual programs, all they can do is what they have been told to do. Memories associated with the larger processors can store small programs, so these processors can handle a small amount of work autonomously. "But the most important thing they do," Shaw says, "is broadcast instructions to the small processing elements."

For instance, employee records for a company could be filed in a NON VON such that each processor manipulated data on a single employee — essentially keeping current that person's address, salary and other personnel details. Explains Shaw, "Then if you'd like to do something like raise salaries of all employees in the sales department, a central-control processor [one or more of the larger processors] would broadcast a series of instructions that would tell all the small processors to look at themselves [their associated memories], figure out if they're in the sales department, and if they are to give themselves a 10 percent raise."

Alternatively, the computer might be asked for the average employee salary. "Rather than adding them all up one by one," Shaw says, "NON VON allows one to add them pair wise — with each binary pair sending up their sum to their parent processor [up the tree]." By this route, Shaw notes, "If you add a million employees it would take you only 20 steps — instead of a million."

To Shaw, the only architecture rivaling his in parallelism is the Connection Machine under design at Thinking Machines Corp. in Waltham, Mass. It's architecture is the brainchild of W. Daniel Hillis. "What makes it a little bit different," Hillis explains, "is that this was designed starting from a problem — how to make a computer that will do 'semantic network' reasoning [reasoning based on an understanding of relationships between concepts] — rather than from the goal of just making a faster computer."

It shares several similarities with NON VON: Each of its ultimately one million tiny processors would be paired with a tiny memory (of a few hundred bits), several of these processor/memory pairs would reside on each integrated circuit chip, and each pair would essentially represent a single fact — such as, the sun is a star.

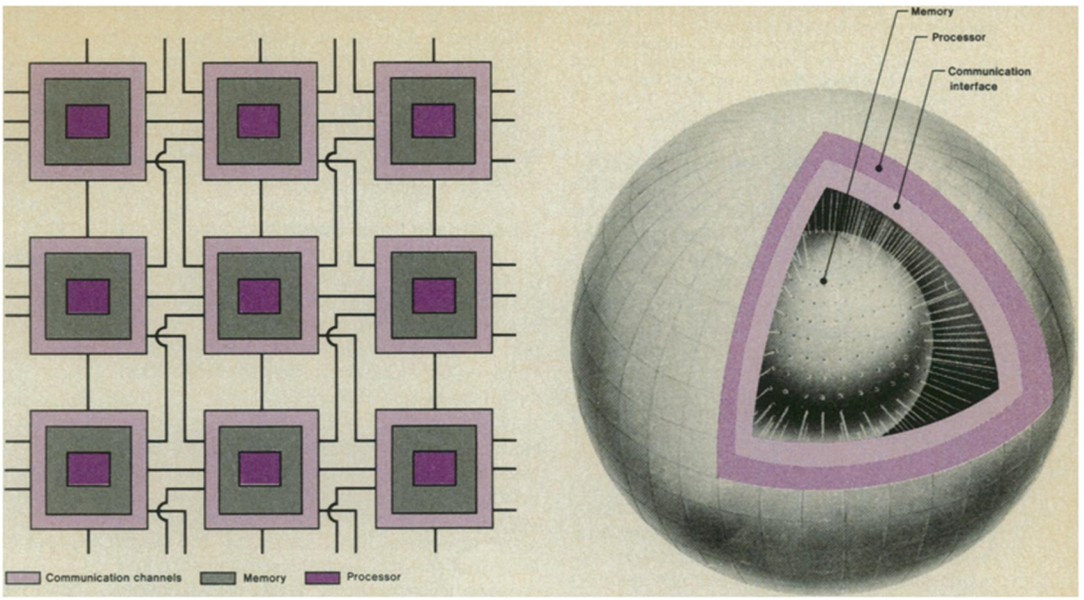
The design goal is to have the small processors listen collectively to a question — translated from a human request and put before them by a *controller* processor — and then to settle on some deduced answer after discussing among themselves the relevant facts. In contrast to NON VON, there is greater egalitarianism among the processors.

"When putting in a knowledge base, you actually form connections between processors that are associated with each other," Hillis notes. For instance, he says, "all the things having to do with physical objects are connected together via this network, which is why we call it the Connection Machine. Essentially the data is the connection."

Traditional interconnection structures for randomly accessible memories are two- or three-dimensional lattice arrays. For a two-dimensional square array — an $N \times N$ array — the maximum time it will take any processor to communicate with another is proportional to $2 \times N$. Therefore, a million-square array of processors would require time proportional to 2,000 machine cycles to communicate between opposite vertices of the array. However, in the Connection Machine, a second set of interconnections overlays the system. Hillis says it can be thought of as "neighboring clusters of 64 processors and their associated memory elements aggregated on individual chips."

The plan is to put up to 64 processors on a chip. In the one-million processor machine, 16,384 such chips would be interconnected "as if they were the vertices of a hypercube in 14-dimensional space — in addition to their square-array connections," Hillis says. The advantage is that "a message can be transmitted from any processor or any one of these chips to any other processor in time proportional to 14 cycles (that is, proportional to the base-2 logarithm of the number of chips) rather than the 2,000 cycles required by a square-array communication network,"

One of the architect's first dilemmas is how to link processors and memory units to maximize data transmission and communication/coordination between processors while minimizing data tie-ups. Each processor could contain a tiny companion memory and be linked via communication channels like a series of little islands (near right). Or processors could draw data from a common memory (far right). There are nearly endless possibilities.



Univ. of Calif./Lawrence Livermore Natl. Lab./DOE

he explains. Moreover, he says, this hypercube network "makes it possible for the Connection Machine computer to operate as if it were continually being dynamically reconfigured under software control to adapt itself to the structure of the problem it is running."

Says Hillis, "We want to make a machine that can genuinely be said to think." A prototype, now under construction, will have 100,000 processors. Directed specifically toward artificial-intelligence understanding, it would aim to become the opposite of an *"expert system"* — perhaps an "amateur system," Hillis says. "We're more interested in the sort of knowledge that a six-year old has than an expert in diagnostic medicine," he explains. "We're interested in the commonsense reasoning that lets you understand why you should put on the sock before the shoe. That's probably a much harder and more interesting problem to do than analyzing how a chess master forces a checkmate. There's certainly much less known about it."

At Argonne National Laboratory, Overbeek and colleagues are predicting a less radical architecture will make the bridge from the fourth to fifth generations, which he explains is why "we're implementing a portable, multiprocessed *PROLOG*" for an MIMD machine. Overbeek describes MIMD (which stands for multi-instruction/multi-data) in terms of a data-processing factory where, together, workers engage in the construction of a single project—a deduction. Each processor, or plant worker, slaves away at subtasks, occasionally stopping to converse with fellow workers — telling them when to expect his subtasks to be finished, asking whom he should send them to, deciding who's project he'll take on next. For much of the time, however, processors toil semiautonomously, conducting varied tasks in a flexible ordering.

"The tricky part," Overbeek explains, "is how you pass things back and forth." The goal is to keep each worker's communication to a minimum and computation to a maximum. However, he notes, there are a number of questions on whose answers the future applicability of these machines now seems to pivot. For instance, how many subparts can a large problem effectively be broken into without creating an efficiency-robbing communications bottleneck between processors? Will efficiency fall as many processors are forced to await completion of some long, indivisible subtask upon whose answer the rest are dependent? Is it better to have a few processors each tackling moderately involved and time consuming computations, or to have a multitude of processors each performing simple tasks?

Unlike many of the other architectures, this one seems certain to become commercial soon. MIMD supercomputers — generally with well under 25 processors — are already being developed for introduction within five years. Among them is the HEP-II by Denver-based Denelcor. "I personally believe it has non-numeric capabilities," Overbeek says. "It may well be a machine with this type of architecture which is capable of executions on the order of 10 million to a billion inferences per second. At least that's the basic thesis my partner and I have made, and that's what we're investigating now. We have a HEP-1 [among the first commercially built multiprocessor systems] where we're building a parallel PROLOG in hopes that it will be capable of execution at enormous rates on the HEP-II."

The designs described here only skim the surface of work being exploited by architects of the fifth generation in computing. For example, 28 separate university and industrial laboratories in this country alone are pursuing data flow concepts. Europe, particularly England, is gaining renown for its architectural pursuits. How does this compare with Japan?

Harold Stone, a computer scientist at the University of Massachusetts in Amherst, has made frequent trips to Japan's computing centers and recently coauthored an overview of that nation's fifth-generation program with University of Tokyo computer architect Tohru Moto-oka in the journal COMPUTER. According to Stone, the Japanese are far ahead in being able to construct MIMD machines and in demonstrating them. "I saw more working [MIMD] machines at Japanese universities than I have in the United States," he told SCIENCE NEWS, adding that although a few U.S. efforts are exceptional, "in Japan they have programs that are almost as interesting — or very close — and they're everywhere."

But what has impressed him most is how Japan's researchers strive to refine ideas. "I can safely state that in the area of fifth-generation architecture," Stone says, "the normal paradigm is to generate an idea, and then to leave it without trying to figure out whether it's good or bad." Not the Japanese. "Their approach," he says, "is to develop an idea, carry it through, measure its performance, find out where it's bad and then to make it better. So while we're generating more new ideas in the U.S., if there are weaknesses in them, they probably haven't been found yet," Stone says. His conclusion: Western researchers had better mend their ways — and fast — or they'll find themselves treading water in Japan's wake. □