APPLYING DATA FLOW IN THE REAL WORLD

BY WILLIAM GERHARD PASEMAN

This model for parallel processing is finding its way into commercial applications

VON NEUMANN MACHINES support a paradigm, a way of thought, that has been used successfully for 35 years. (See the text box entitled "The Von Neumann Paradigm" on page 214.) In a world in which thousands of PCs are sold in a month, the von Neumann computational model is not going to be replaced by an alternate model any time soon. However, valid reasons exist for using architectures based on alternatives to the von Neumann model of computation.

One reason is that many algorithms perform better and more inexpensively on other architectures than on von Neumann machines. It is not simply raw horsepower that produces this performance increase; it is horsepower that is tailored to the operations that the algorithm uses. Algorithms that can be expressed easily and coherently using the set of operations that the architecture provides usually perform better than those that cannot.

When algorithms and architectures mesh well together, we say that the architecture supports the algorithm. When an architecture makes implementation of the algorithm feasible, but not convenient, we say that the architecture weakly supports the algorithm. The better the mesh between the two, the better the price/performance ratio of the combination will be.

The von Neumann paradigm supports many algorithms well and weakly supports others. In this article, we will briefly review the relationship between several non–von Neumann paradigms then examine one non–von Neumann paradigm, data flow, in detail. Finally, we will look at some commercial architectures that support this model.

WHY WE SHOULD CARE ABOUT PARALLELISM

There are many ways to decrease the time an algorithm takes to complete on a given processor. If the processor is a general-purpose computer, one good way is to put the part of the algorithm that takes the most time into hardware. This is called *functional specialization*. An example of this is the Z80 IX,IY register instruction set. The instructions in this group were added to support procedure parameter passing.

Another method of speeding things up is to break the algorithm into parts and devote a separate processor to each part. This type of parallelism is called *functional decomposition*. It works well only if the processors have the work divided evenly among them. If the work is not divided evenly, one processor will become a bottleneck.

Finally, you can break the algorithm's input data into parts and have a set of identical processors handle each part. This type of parallelism will not work on all algorithms.

Of course, all these methods potentially can be used at the same time. Functional specialization usually provides the greatest speedup; however, that speedup usually is very specialized. Parallelism provides less speedup, but it is applicable to a broader range of problems.

Computer architectures that effec-

(continued)

William Gerhard Paseman is a software manager at Daisy Systems. He can be reached at 330 Sierra Vista, Apt. #3, Mountain View, CA 94043.

tively use processor parallelism possess linear price/performance curves over a wide performance range. For example, if a given algorithm takes 4 minutes to complete with $1000 worth of fifth-generation hardware, then it should take 2 minutes to complete with $2000 worth of hardware and 1 minute to complete with $4000 worth of hardware. (See the text box entitled "Linear Price/Performance and Incremental Performance," page 212).

Conventional (von Neumann) computer architectures do not have linear price/performance curves over a wide performance range. In order to make a conventional computer perform general algorithms faster, you don't simply add more components. Instead, you make its individual components faster. (There are some special cases in which you can improve performance by adding components; for example, adding more memory to a demand-paging environment.) Another way of saying this is that von Neumann architectures are not designed to be scaled over a wide range with respect to performance.

The price/performance relationship between the two approaches is illustrated in figure 1. The graph indicates that von Neumann computer architectures will experience a performance cutoff at some point. This point will occur when all the components reach the theoretical performance limit of the technology upon which they are based.

Parallel architectures will also experience a performace cutoff at some point. This point will occur when the cost of coordinating two pieces of work between two components exceeds the cost of having one component do both pieces of work. In the general case, this point must eventually occur regardless of the size or speed of the components, regardless of the speed of communication, and regardless of the complexity of the work that the components must do.

Until they reach the von Neumann cutoff, von Neumann machines probably will perform better than their parallel counterparts. This is because parallel architectures usually have a communication overhead that von Neumann architectures lack.

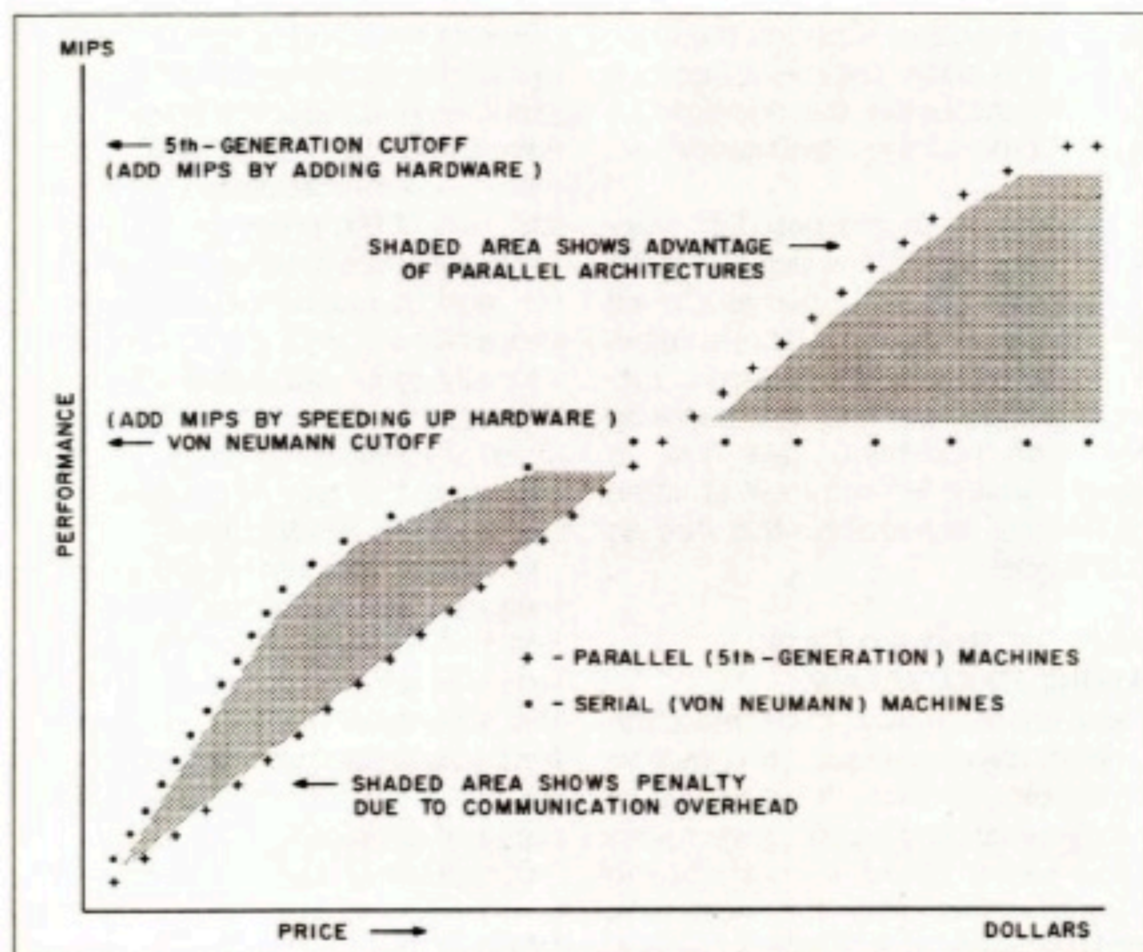


Figure 1: *A comparison of the price/performance aspects of serial and parallel computing architectures.*

Models of Computation That Support Parallelism

There are several paradigms for which it is currently popular to design parallel machines. The oldest is the control-flow paradigm.

The control-flow paradigm assumes that two or more processors share common memory. A control-flow architect usually views algorithmic parallelization and processor synchronization as being the programmer's problem. The architect supports the programmer by providing machine instructions that allow the programmer to do explicit processor synchronization in his code. Due to the wide interface between processes (i.e., the common memory), it is easy to write poor code that uses the interface in an undisciplined way. As a result, such systems have gotten bad press from many in the research community.

Most of the other paradigms are based around a weaker, more theoretically tractable concept in which, conceptually, memory sharing is not required. This concept is called message passing. Message-passing architectures allow programmers to structure their programs into islands of computation. These islands process asynchronously and communicate by passing messages to one another.

The data-flow paradigm is a message-passing model in which each island of computation is very small and usually performs the same operation repetitively on streams of values. Data-flow computation is data-driven, which means that each island starts processing whenever all data necessary to its computation is available.

The reduction paradigm is similar to the data-flow paradigm, except that a strong separation is made between the spawning of a computation and the computation itself. Here, computation is demand-driven, which

(continued)

means that the requirement for a result triggers the island that will generate it.

THE DATA-FLOW PARADIGM

The basic concepts of data flow were originally developed in the 1960s by compiler writers. Compiler writers used data-flow graphs to do performance optimization on standard serial programs. A data-flow graph is a directed graph in which the nodes represent primitive functions such as addition and subtraction, and the arcs represent data dependencies between functions. It was realized in the early 1970s that if data-flow graphs were executed directly, the architectures that executed them could be massively parallel.

A picture of a data-flow graph for the function 3 • (y + F(x)) is shown in figure 2. In this model, nodes are viewed as stations in an assembly line. The stations are connected by conveyor belts (called arcs). The conveyor belts carry containers (tokens) that hold contents (values). At each node is a person (processor) who operates the station's function. When the first token hits the F node, the processor takes its value, operates on it, and passes a new token with the result to the + node. As F was processing the first value, + could do nothing, since it required two tokens in order to operate and had only one available. Now, however, + has two values: 1 from F and 9 from y, so it adds them together and passes a token with the result to •. As + was operating on its first set of tokens, F was operating on its second token. Thus, parallel operation is achieved by pipelining values through nodes that execute fixed functions.

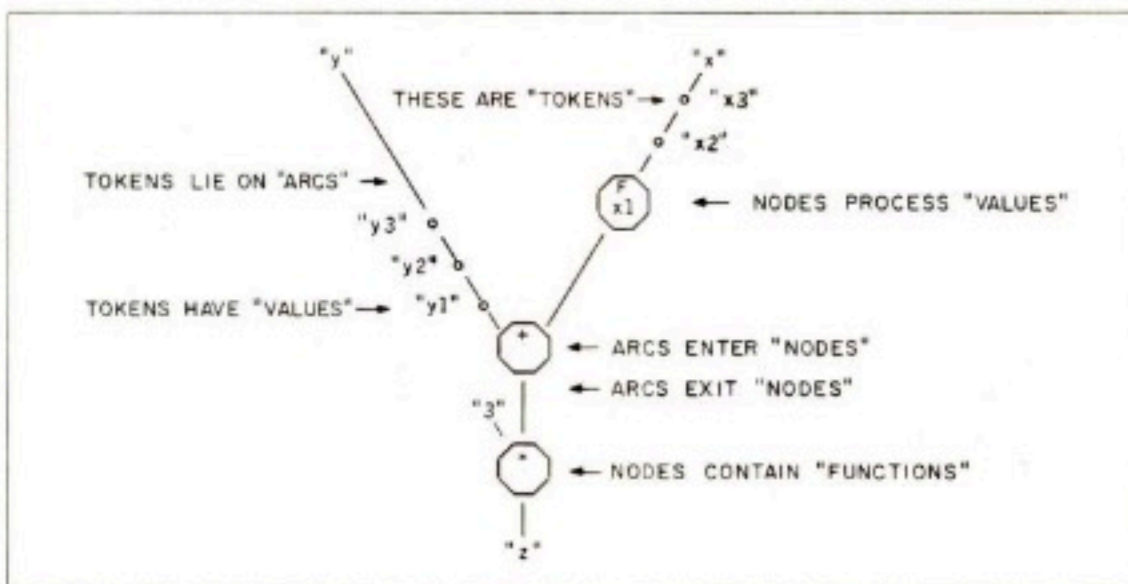


Figure 2: A *simple data-flow graph of the function z = 3 • (y + F(x)).*

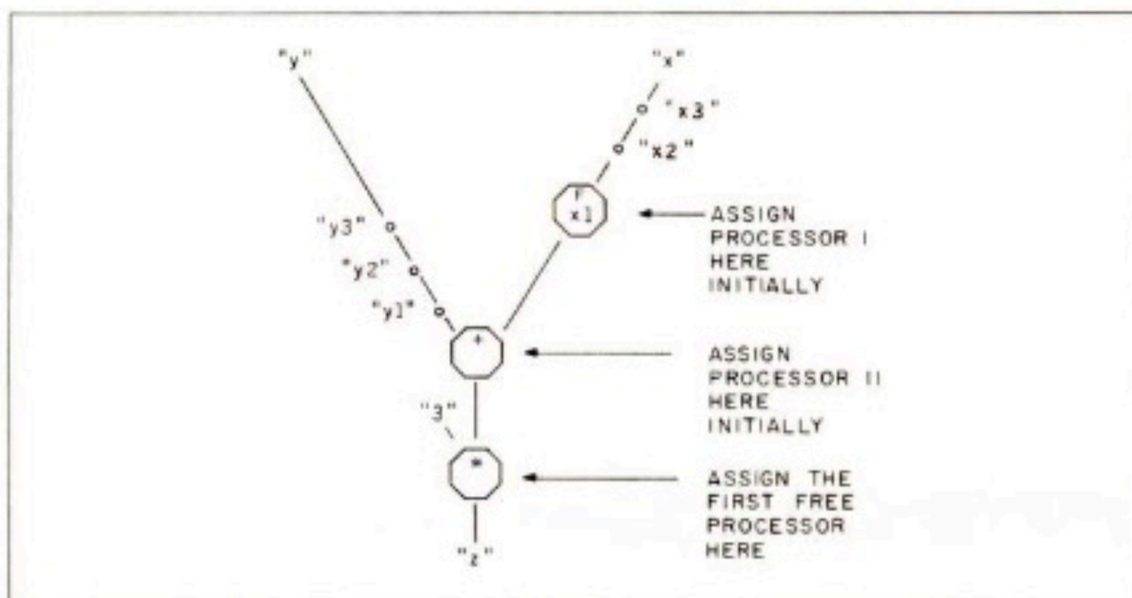


Figure 3: A *data-flow graph of the function z = 3 • (y + F(x)) illustrating static processor allocation: processors are assigned to nodes at compile time.*

DATA-FLOW EXECUTION MODELS

Normally, a data-flow graph has many more nodes than processors. Therefore, an execution model, a method of allocationg nodes to processors, is needed. We will briefly describe two models, the static and dynamic models of execution.

Figure 3 depicts the static model, in which the processors run to the nodes, where all input tokens are present and no tokens are on the output arcs.

However, this method leads to situations like that mentioned above, where the + node was bottlenecked by the F operation. In order to rectify this problem, the dynamic model was invented. In the dynamic model, instead of waiting idle, the processor at the + node would help the F processor by processing its second token for it. Figure 4 depicts the dynamic model.

DATA-FLOW ARCHITECTURE

It is still unclear exactly how to construct expandable hardware to support any of the above execution models.

One common data-flow architecture is shown in figure 5. Here, the data-flow machine consists of three stages—a matching unit, a fetch/update unit, and a processing unit (perhaps more than one). Let's see how these parts interact on the previous example. Let's refer to the nodes by symbolic name. We will call the + node PLUS and the • node MUL. At some point in the calculation, the matching unit has two tokens passed to it by the processing units. The first token indicates that the left (L) arc of the PLUS node has been set to 10 (a). Later, it receives a token indicating that the right (R) arc of the

(continued)

PLUS node has been set to 7 (b). The match unit knows that PLUS has only two inputs, so at this point it sends a token set to the fetch/update unit for processing (c). The fetch/update unit knows that PLUS performs the + function and that it fans out to MUL's arc L, so it sends this information to an arbitrary processing unit (d). The processing unit performs the addition and sends the result to the match unit (e).

If the system allows more than one instantiation of an instruction to be active at a time (this would occur if the machine were executing the same instruction for the i and $i+1$ instantiations of a loop simultaneously), then the descriptors must also be tagged with a process ID. This is done in a dynamic data-flow system.

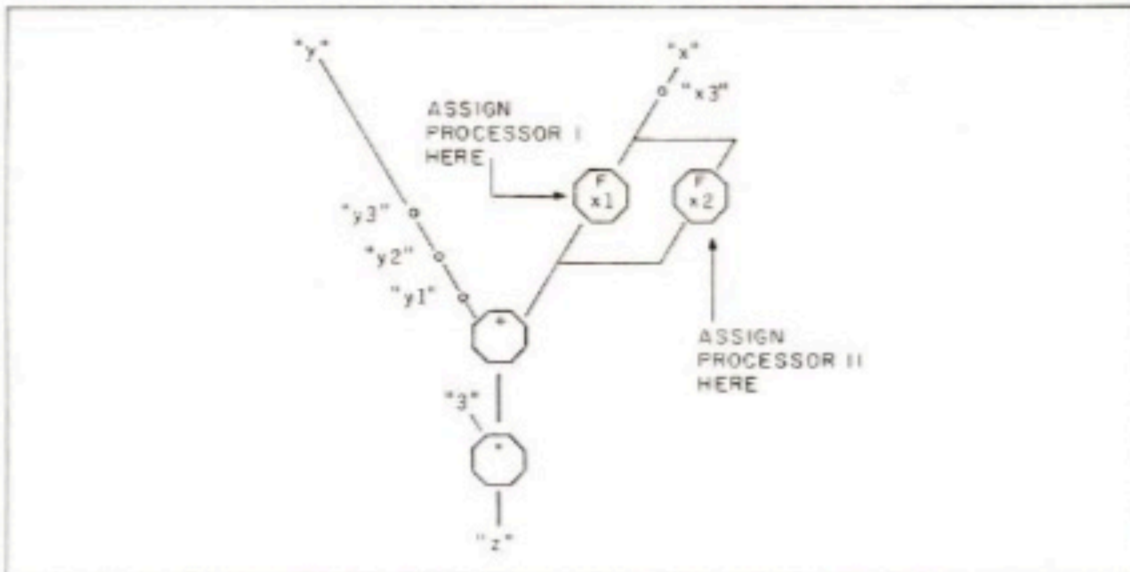


Figure 4: *A data-flow graph of the function $z = 3 \cdot (y + F(x))$ illustrating dynamic processor allocation; processors are assigned to nodes at run time.*

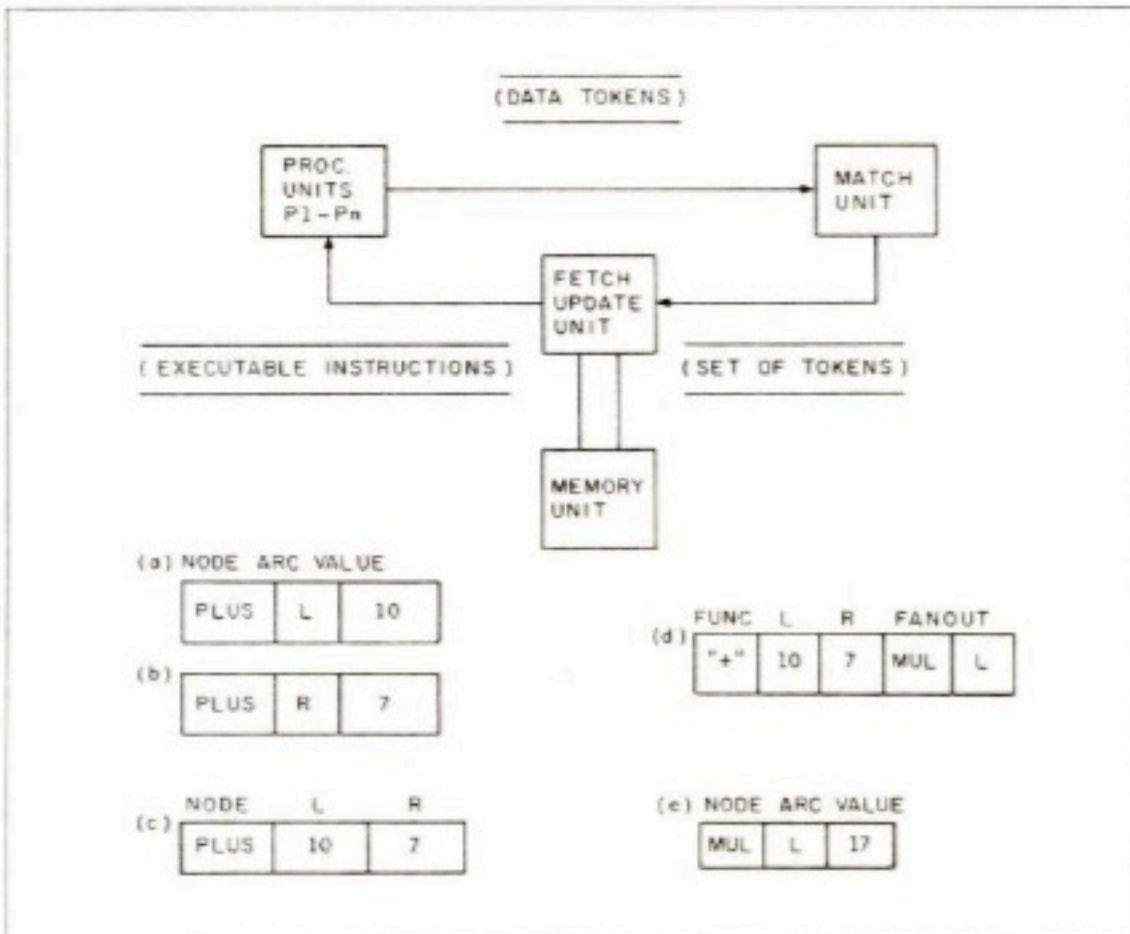


Figure 5: *An example of data-flow architecture, with packet communication and token matching.*

Properties of the Data-Flow Paradigm

The data-flow model makes many assumptions about the nature of the algorithms it runs. Some are:

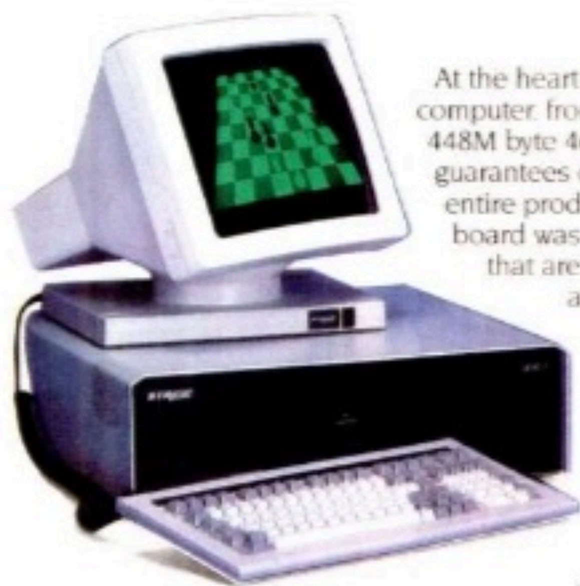
- All information needed to execute the algorithm must be contained in its data-flow graph. That is, the paradigm does not use any structures other than the data-flow graph in order to execute the algorithm that the graph represents. The graph is the data-flow machine's "machine language" for the algorithm. The machine takes advantage of the graphical nature of the program in order to produce the speedup.
- The algorithm should not have a single locus of control. That is, the data-flow graph should allow more than one node on the graph to be executed at a time. If the algorithm has a single locus of control, it will run slower on a data-flow machine than on a von Neumann machine (due to the communications overhead).
- The data-flow graph must have a high degree of granularity. In other words, the graph nodes must contain things like + primitives and not "sort" primitives. One reason this is important is that graphs with granular primitives contain the potential for more parallelism. Note that this implies that the time for a "context switch," which is the time for a processor to switch from processing one node to processing another, must be small.
- The data-flow graph must have locality of effect. This means that the nodes do not fan out to a large number of other nodes. This is important, since nonlocality would stress the communication network of the data-flow machine.

These assumptions can be used to judge whether or not an algorithm matches well with the data-flow paradigm. If the algorithm to be executed does not have the above

(continued)

STRIDE MICRO

One Board...
One Family



At the heart of every Stride 400 Series microcomputer, from the floppy-based 420 to the 448M byte 460, is an identical CPU board. This guarantees compatibility throughout the entire product family. And it means, our CPU board was designed with standard features that are either options or simply unavailable on other microcomputers:

☐ 68000 microprocessor (10 MHz with no wait states)
☐ VMEbus
☐ 256K bytes RAM
☐ 5¼" 640K byte floppy
☐ Battery-backed real time clock
☐ 4K CMOS RAM
☐ Four RS-232C serial ports (Stride multiuser BIOS)
☐ Centronics bi-directional parallel port
☐ Omninet Local Area Network (Liaison LAN software)

With this basic design, Stride is able to explore the full range of 68000 applications from an advanced multiuser, multi tasking BIOS to built-in local area networking. No other microcomputer offers the flexibility to run over a dozen different operating systems and more than 30 languages/compilers.

The basic design is backed by a rich option list:

☐ 12 MHz 68000 processor ☐ VMEbus (Eurocard) cage ☐ Low cost, high speed graphics ☐ NOD™ cursor control ☐ 12M bytes of RAM ☐ 448M bytes of hard disk storage ☐ 22 serial ports ☐ Floating point processor (NS16081) ☐ Cartridge streaming tape backup ☐ Memory Management Unit

CBASIC COBOL
Modula-2
Pascal FORTRAN
RM/COS Lisp
UNIX System V
C p-System
CP/M68K

All this, and still the best price/performance ratios in the industry: from $2900 to over $60,000. But it begins with the powerful Stride CPU board, a standard feature of every 400 series system. It's what we call "Performance By Design."

STRIDE MICRO

Formerly Sage Computer

For more information on Stride or the location of the nearest Stride Dealer call or write us today. We'll also send you a free copy of our 32 page product catalog.

Corporate Offices:
4905 Energy Way
Reno, NV 89502
(702) 322-6868

Regional Offices:
Boston: (617) 229-6868
Dallas: (214) 392-7070

Inquiry 377

properties, then the data-flow model is not the one to use to execute it.

Commercial Possibilities of Data Flow—Texas Instruments

Texas Instruments was one of the first companies to investigate the viability of data flow all the way to the hardware prototype stage. TI's research was done between 1975 and 1980. The company's architecture consisted of four "simple processors" and a host, connected in a ring architecture. TI has not yet released a commercial product based on this research.

TI's hardware/software effort was called a Data Flow Testbed. The testbed could accept a program written in a conventional programming language, compile it, link it, and automatically partition it to run on any number of processors. The people at TI did this in a relatively straightforward way. They took an existing commercial compiler/linker that generated data-flow graphs in its optimization phase. If the resulting graph completely described the algorithm, they could automatically partition the graph onto a number of processors and run it.

TI recognized that it is currently very difficult (i.e., commercially impractical) to generate data-dependence graphs for most real programs written in standard languages. The company knew this meant that "pure" data-flow processors cannot run standard software. Therefore, TI's system used a mixture of data-flow and classical control-flow techniques. That is, the computer was not a "pure" data-flow machine but rather used data-flow constructs where appropriate.

TI's primary interest was the application of data-flow concepts to large-scale machines running standard (unmodified) high-level language programs. The company investigated whether compilers could extract enough of the latent parallelism in standard programs to produce significant speedup in a data-flow architecture. One of TI's most interesting results was that the average amount

(continued)

Inquiry 45

Keep Your Computer Fit With CROSS-CHEX™

Cross-Chex is the complete menu-driven system of hardware diagnostics. It analyzes performance levels of Winchester and floppy disk drives, video display, RAM memory, video memory, ROM character generator and keyboard. Includes (1) Diagnostic Diskette (1) **Cross-Chex** Program Diskette (1) Users Manual. Let **Cross-Chex** keep your computer fit, ensure the performance of your computer, maximize your uptime and maintain the integrity of your data **all for the low price of $99.00.**

Convert any CP/M to DOS with CROSSDATA®

Crossdata is the low-cost utility software that converts data/text file formats from CP/M to MS/PC-DOS and back again on any IBM PC/XT or clone.
It is a self-contained program, ready to run, that reads/writes CP/M and MS/PC DOS Diskettes using MS/PC-DOS 2.0, 2.1 or 3.0 and comes with 28 formats—plus you can add your own!
Solve your computer incompatibility problems fast with **Crossdata,** the proven conversion package, by ordering one today **for only $99.00.**

Backup/Restore for Winchester under PC/MS-DOS, CP/M86 and CCPM with CROSSAVE™

Now you can back-up large data base files from a Winchester to a floppy for files that exceed the diskette capacity.
Crossave will save and/or restore a file or a selected group of files that have been updated. It also backs up and restores all of the files on the Winchester. It uses compression to reduce storage space requirements on the floppy and expands the file upon restoration. Requires IBM PC/XT or clone.
Reasonably priced at $99.00.

No risk 10-day money back guarantee on all products

Don't delay. Call us today: (408) 395-2773 or write:

236 North Santa Cruz Ave., Los Gatos, CA 95030

All major credit cards accepted

NEC's chip is oriented toward image processing.

of parallelism available in standard FORTRAN programs was between 5 and 20. This meant that the maximum theoretical speedup TI could achieve (using "off the shelf" hardware) in these cases was 5 to 20 times. (Data flow can take advantage of parallelism only where it exists. If the programmer writes an algorithm so that no parallelism can be extracted from it, then a data-flow version of the algorithm will run no faster than a von Neumann version of the algorithm.) Currently, using high-performance hardware in a von Neumann machine affords a much greater speedup.

NIPPON ELECTRIC CORPORATION

Of the three companies discussed here, NEC's approach comes closest to the pure data-flow paradigm. The company's approach is based on a single chip that can contain up to 64 nodes and 128 arcs. Systems can incorporate up to 14 of these chips by connecting them into a ring in a very straightforward way. (It is possible to extend the limit beyond 14 chips, but the arrangement is much more complex.) A complete standard system, then, could run up to 896 two-input nodes distributed across 14 processors.

NEC's chip is oriented toward image processing. In the company's own words, "Because the majority of application programs for image processing execute iterative operations for large volumes of data, image-processing programs are relatively small compared to general data-processing programs." Although NEC's machine has a relatively small number of arcs and nodes in its system, each node can execute a high-performance operation.

NEC's initial focus is not on running existing high-level language programs but rather on running small, easy-to-rewrite programs that require high performance. That is not to say that NEC does not address these issues; rather, that the company is first entering the market where data flow's

(continued)

LINEAR PRICE/PERFORMANCE AND INCREMENTAL PERFORMANCE

Suppose a salesman sells you a processor for $1000 and tells you that it will run your favorite program in just eight hours. He then tells you that due to the marvels of fifth-generation computing technology, you can bolt in another processor for another $1000 and your program will run twice as fast. It will now take only four hours to complete. You happily buy two processors. Still, four hours is a long time, so you call your salesman and tell him that you want to halve the time to two hours. The salesman now sells you not one but two more processors in order to do this. You realize that for each processor you buy, you incrementally increase performance by $(P+1)/P$. For one processor, this is $(1+1)/1 = 2x$, or a 100 percent speedup. For two processors, this is $(1+2)/2 = 1.5x$, or a 50 percent speedup. For three processors, this is $(1+3)/3 = 1.33x$, or a 33 percent speedup.

This is an extremely attractive situation for the salesman, of course, since he gets an order of magnitude increase in commissions every time you want to get an order of magnitude increase in performance. It is, of course, not a very good situation for you.

benefits are the strongest. In fact, NEC is now working on an integrated system in which to embed its chips. How the company approaches system-level problems (language definition, translation, and debugging) remains to be seen.

In summary, NEC was able to use the data-flow model by applying it to a domain in which

- The algorithms are easily expressed in terms of a data-flow graph.
- The algorithms contain a great deal of inherent parallelism.
- The architecture can run small, easy-to-program algorithms.
- There is a great need for fast execution. (Image processing is computer-bound.)

DAISY SYSTEMS CORPORATION

Daisy Systems started selling a commercial data-flow architecture in the first quarter of 1984. The company's approach is based on a set of board-level processors connected in a ring. The basic configuration consists of three or four processing units plus a host processor. The units are capable of processing 65,000 to 1,000,000 nodes, depending on the level of modeling. Each node can have up to 256 inputs.

Daisy Systems' data-flow architecture is the first to respond to the customer's need for high-speed discrete logic simulation. In essence, a discrete logic simulator runs an algorithmic description of a piece of hardware. By their very nature, these algorithms are expressed in terms of graphs in which each node is a simple operation.

The hardware designer of these algorithms consciously works to make his design exhibit a high degree of parallelism. Therefore, Daisy did not have to worry about the algorithm "running out of parallelism" of which to take advantage. Even better, the parallelism is very great at the machine-instruction level.

Like TI, Daisy recognized that the "pure" data-flow paradigm did not completely address all of simulation's problems satisfactorily. For example, the "pure" data-flow model has no way of handling stored state (side effects). Daisy addressed this and other similar problems by extending the paradigm.

At the programming level, Daisy recognized that the programming task in advanced architectures is difficult and error-prone. In many approaches, the user must adapt to a paradigm that is unfamiliar, unintuitive, and dif-

Daisy Systems' data-flow architecture is the first to respond to the customer's need for high-speed discrete logic simulation.

ficult to use. Daisy overcame this problem by allowing users to communicate in the languages that they have always used: graphics, Boolean expressions, and a standard behavioral language. Daisy was able to do this well because the primitives that the designer uses map easily to the primitives that Daisy's architecture supports. The mapping process (compilation, linking, and code generation) is totally automatic.

Daisy was able to use data flow by applying it to a domain in which

- The algorithms are naturally expressed in terms of a data-flow-like graph.
- The algorithms contain a great deal of inherent instruction-level parallelism.
- There is a great need for fast execution. (Logic simulators implemented on von Neumann machines may take days to run big simulations.) Daisy's machine runs approximately 100 times faster than most software simulators.

SUMMARY

NEC and Daisy have successfully used data flow to solve two different commercial problems in an appropriate manner. Both problems are easily expressed using data-flow graphs, have a great deal of instruction-level parallelism, and require scalable execution and high performance.

As more companies discover problems for which data flow is the best solution, the repertoire of practical parallel algorithms using the data-flow model will grow. ■

THE VON NEUMANN PARADIGM

Mathematicians have been proposing computational paradigms, or "models of computation," since the time of Charles Babbage (witness Turing machines, Makov productions, and Church's Lambda calculus). However, the most well known paradigm was pioneered by John von Neumann. Von Neumann's model is based on the concept of a single central processing unit that accesses a linear array of fixed-size memory cells. These cells can contain either instructions or data. Instructions are relatively low-level. They perform simple operations on elementary operands. In the von Neumann model, program control is sequential and centralized. It is upon this paradigm that most commercial computer architectures are based.

Strictly speaking, a non-von Neumann paradigm is one that departs from any of these concepts. For example, a machine that keeps its data and memory in two separate banks is not a von Neumann machine. Recently, however, "non-von Neumann" has come to mean a paradigm that differs primarily in the last of the above properties, that of sequential, centralized program control.