This low-cost auxiliary processor is based on commercially available microprocessors. Its parallel capabilities enhance the peformance of small computer systems in vector and associative operations.

A Vector Processor Based on One-Bit Microprocessors

Wayne M. Loucks, Martin SneIgrove, and Safwat G. Zaky
University of Toronto

The parallel processing capabilities of an associative processor are attractive to many applications. Operations like searching and sorting are inherently parallel, since they can be regarded as a sequence of basic operations -such as compare, shift, and mark-performed in parallel on a large number of operands. Many organizations have been proposed for associative processors.[1-3] Of these, the word-parallel, bit-serial type has received the most attention, since it requires much simpler hardware than fully parallel schemes.

Because associative processors are hardware-intensive, they tend to be economically viable only in large systems. However, we have designed an associative processor meant for relatively small applications. Based on an array of commercially available microprocessors, it is a word-parallel, bit-serial machine that stores and processes data in the form of vectors consisting of fixed numbers of elements. We have dubbed the machine Vastor, for vector associative store-Toronto.

The microprocessor we selected for Vastor-the Motorola MC14500B-consists of a one-bit-wide processing element. Although it has been marketed as an industrial control unit, its architecture is well suited to the requirements of Vastor. We chose it primarily because of its simplicity and suitability for parallel operation.

Vastor is intended to be a special-purpose processor attached to a conventional minicomputer or to a high-performance (68000- or Z8000-based) microprocessor system. We will refer to this mini- or microcomputer as the *host.* In such a system, Vastor would handle those parts of the workload that can benefit from its associative and vector capabilities. Use of associative processors in this manner has been suggested by many authors,[4] as have many applications.[5] Vastor, however, represents an associative structure and an implementation that are economically viable in a small system. We have constructed and tested a prototype.

The main design constraints were low cost and modularity. These required use of readily available components and simple internal communication and control. Furthermore, Vastor could not be allowed to overload the computer to which it is attached. Modularity required simple and easily expandable backplane interconnections between modules. We also sought to make Vastor's components and structure suited to integration.

The Vastor processor (Figure 1) consists of two main components, the *processing array* and the *controller* The

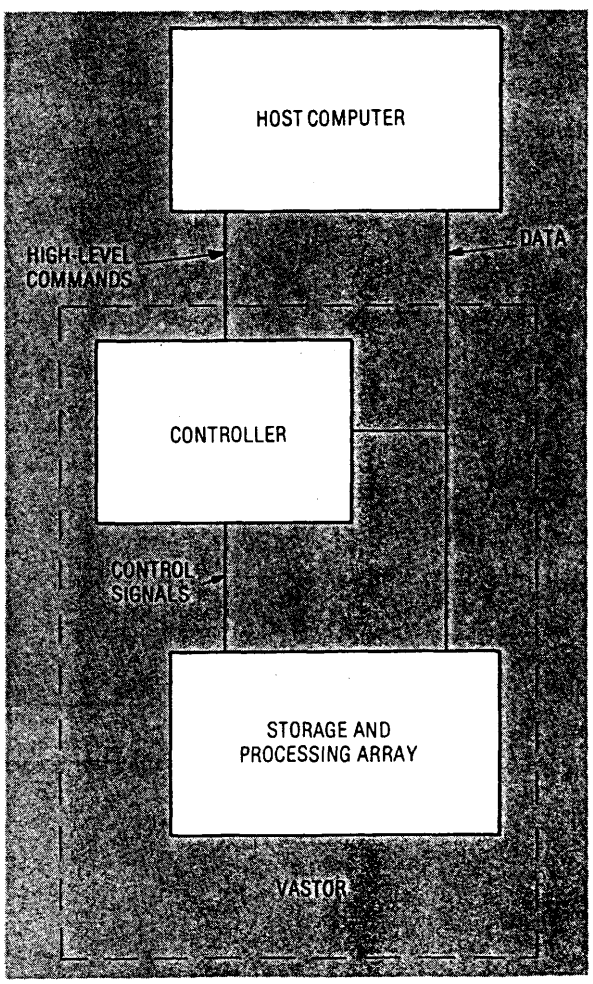


Figure 1. The Vastof processor.

0272-1732/82/0200-0053$00.75

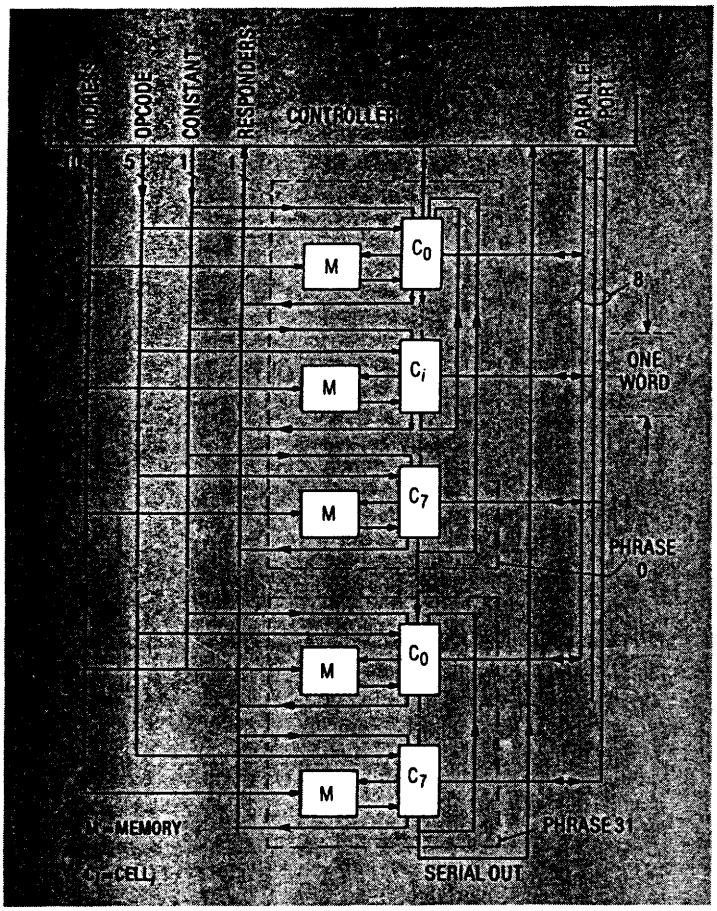


Figure 2. Control and data paths.

processing array contains all storage and processing elements, or PEs. The controller translates high-level commands received from a scalar machine-the host- into sequences of control signals for the processing array. The control signals drive all PEs of the array in parallel. However, each PE operates on its own data stream. This is a single instruction stream/multiple data stream, or SIMD, structure.[3]

Here we present a practical implementation of the array and its controller and describe input/output transfers between the array and the host computer. Algorithms that can be implememted on vector-oriented machines such as Vastor are available in the literature.[5-7]

Machine structure

The organization of the Vastor array is illustrated in Figure 2. The storage section in the array consists of a 256-word memory with a word length of several kilobits. A dedicated processing cell C is provided for each memory word M. Words are grouped into phrases of eight each for convenience during I/O transfers. Operations are performed on vectors of data elements (Figure 3); the elements of a given vector occupy the same bit position in all words. While the number of bits per element is the same for all elements of a given vector, it can vary from one vector to another. A one-bit-wide PE is a part of every word. The shift-register, or SH, provides the main mechanism for data transfer among Vastor words, as well as between the array and the outside world.

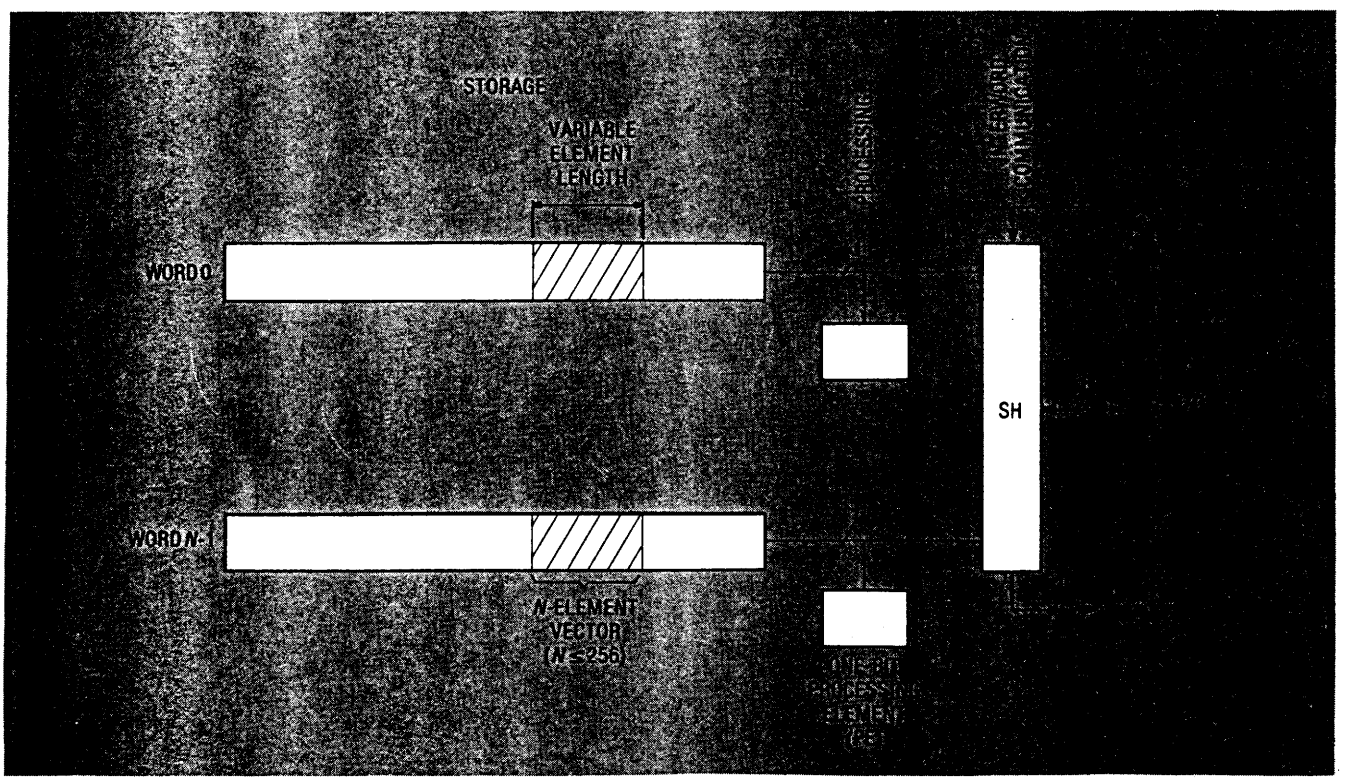


Figure 3. Data organization in the Vastor array.

Vastor is an SIMD machine because opcode lines are shared by all cells, as shown in Figure 2. Each cell contains a storage element, which can be used to mark individual words. The I/O structure enables the host to read from and write to marked words in the memory. The selectivity afforded by the mark bits allows Vastor to serve as a content-addressable memory for the host machine. Each cell can also perform logical and arithmetic operations on its memory. These operations are under the control of the mark bit. Therefore, one can operate in parallel on all data elements that satisfy some arbitrary condition. These features give Vastor the properties of an associative processor.

One could leave all words selected and use Vastor as an array of processors. Vastor's I/O structure allows large quantities of data to be transferred to and from the host machine via the parallel port shown at the right of Figure 2. The I/O data transfer rate ranges from 0.5 to eight megabits per second. Each cell C can perform data manipulation operations on its corresponding memory word M. Interprocessor communication within the array enables handling of data organized in the form of a one-dimensional array, hence the word vector in the machine's name. Associative operations can therefore be seen as a particular case of array processing in which a preliminary computation is used to select data in certain cells for either further processing or output to the host machine.

Vastor operations are essentially word-parallel and bit-serial. The major differences between Vastor and other serial machines, such as **Staran,**[3] stem from the pragmatic considerations of component cost and backplane complexity. Staran's memory is multidimensional. That is, data can be accessed either by row (horizontally) or by column (vertically) in a memory array of 256 rows by 256 columns. These two modes of access involve Staran's relatively complex flip interconnection network. A flip network is not required by Vastor.

Vastor uses 256 conventional 1024 x l-bit random-access memories, all driven by the same address lines (see Figure 2). Operations can be performed only on columns of memory. This makes it a vertical* computer, similar to that proposed by **Shooman.**[2] The design of the I/O structure compensates for the resulting difficulty in communicating with the horizontal host machine.

***Computer memory can be regarded as a twodimensional array of bits whose rows are data words. A horizontal computer (the most common type) operates on rows of that array; a vertical computer operates on columns.**

Associative memories and processors

An associative memory, or AM, consists of a number of storage cells organized in the form of *n*-bit words. Memory words can be accessed on the basis of their contents. The names *content-addressable memory, parallel search memory,* and *catalogue memory* have also been applied to this structure.

Access to an AM is initiated by specifying one or more search criteria. All memory words whose contents satisfy these criteria are said to be *responders* to the search. In general, a given search can have zero, one, or many responders. The contents of words selected in this fashion can be read out or updated. In many AM structures, a number of mark bits are associated with every word location. A successful search might result in changing the state of some of the mark bits associated with the responding words. Since mark bits can be tested in later searches, a complex search can be carried out as a sequence of primitive searches.

The search criteria that can be used and the operations that can be performed on the contents of the responding words vary considerably from one AM structure to another. Obviously, if a wide variety of search and update operations are to be supported, a considerable amount of logic circuitry is required at every word location. When the performable operations at every word location go beyond read, write, and simple numerical or logical comparisons, the device is usually referred to as an associative processor, or AP. In this case, a number of arithmetic and logic operations can be provided. Moreover, communication paths are often available between neighboring word locations. This allows the transfer of data operands to be used in the search and update operations.

An associative memory can be organized in three ways:

(1) *Fully parallel.* The specified search is carried out in parallel on all bits of all word locations. Economic considerations usually limit this structure to small AM units.

(2) *Bit serial.* The search and update circuitry associated with every word location in a bit-serial AM is one bit wide. A given operation proceeds in parallel at all word locations, one bit position at a time. This organization saves a considerable amount of the logic circuitry required for each word. Compared to the fully parallel structure, it allows economical construction of much larger AMs. It also enables more complex search and update operations to be implemented.

(3) *Word serial.* This organization takes advantage of the speed and economy of serial memories. Search and update operations are performed on one word at a time, in a bit-parallel fashion. Thus, only one *n*-bit wide search and update station is required for the whole memory, *n* is the word length.

The multiple-response problem. When a search operation is performed in an AM, more than one word might satisfy the search criteria. No problem arises when the result of the search is to be used locally to set a flag or perform some other update function. If, however, output data must be obtained from the responding words, there must be some means for selecting the responding words one at a time. In applications where multiple responses are likely, the *multiple-response resolver* might take the form of a serial control chain that threads all memory words. Responding words are selected one at a time according to their order on the chain.

When the number of elements in a data vector is greater than the number of words in memory, operations can be carried out on ***subvectors*** of 256 elements each. (Shooman also makes this compromise.)

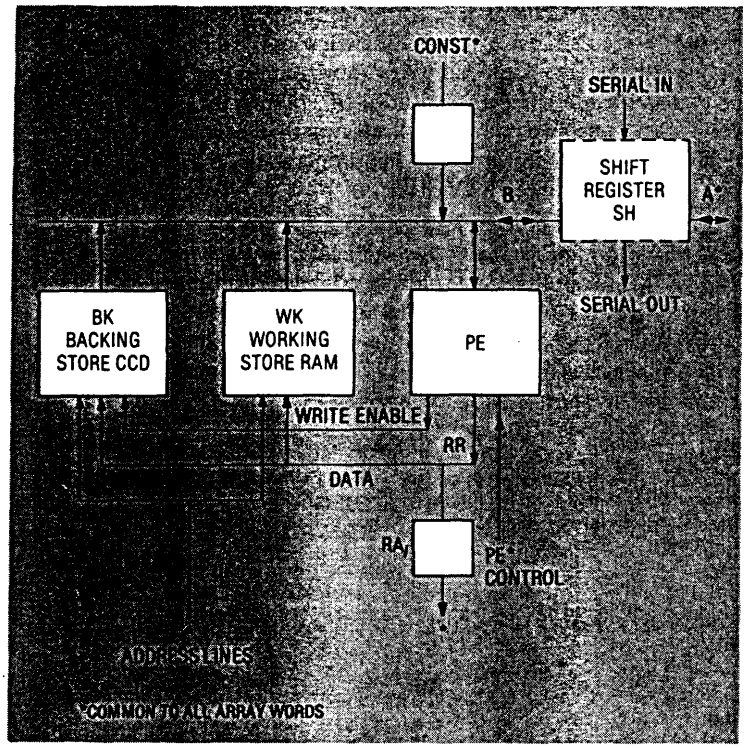


Figure 4. One word of the storage and processing array.

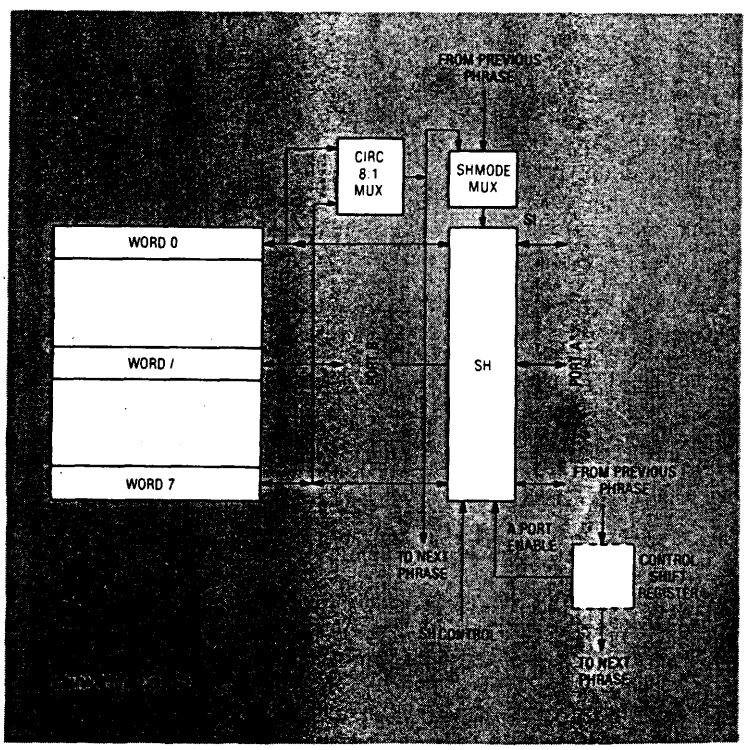


Figure 5. The phrase structure.

Interconnection considerations heavily influenced development of the Vastor structure. The array has been designed to use only daisy-chained and bused connections between circuit boards. This allows new boards to be added at any time to increase the size of the array, with only minimal modifications to the existing backplane. The structure is also well suited to large-scale integration because of the small number of interconnections required between modules.

The main effect of the restriction on backplane complexity is to limit the choice of interword and associative facilities. Hence, interword communication is via a shift register (Figure 3), which involves a daisy-chain connection between circuit boards for both data and control information. Moreover, a single bused connection common to all words of the array (the responders bus in Figure 2) combined with an analog-to-digital converter (not shown) provide limited-accuracy.associative testing.

The structure of Vastor can be discussed in terms of three separate features: intraword storage and computation, interword communication, and associative testing capabilities.

Intraword facilities. Figure 4 shows the components of a Vastor word: two kinds of storage, a one-bit processor, and one bit of a shift register.

The working store, or WK, in Figure 4 is a random-access memory. Data are taken from and returned to this memory during computation. The BK, or backing store, is a serial memory. Its contents are swapped with the contents of the working store in syllables containing 256 bits each. One more bit of storage is available for each word in its part of the shift register, or SH. This bit can be used for' temporary storage of operands. The intraword facilities can be expanded by using line B of Figure 4.

The one-bit PE with which Vastor has been implemented is Motorola's MCl45OOB industrial control unit. It performs a limited set of primitive operations on external data and on a one-bit internal accumulator called the RR, or result register. Another internal register, the output enable, or OEN, contains a mask that is used to enable selective write-back into either the working or the backing store. The collection of the OEN registers in all words constitutes the output enable vector.

Interword communication. The SH is the primary medium for interword communication. It is the only machine feature that assigns any order to the words SH is divided into eight-bit segments, as shown in Figure 5. Each segment has two parallel bidirectional ports, A and B. The B port is connected to one ***phrase*** of eight Vastor words. The A ports of all segments are connected to form an eight-bit I/O bus.

Two multiplexers, CIRC and SHMODE, connect the serial inputs of the segments of SH to any of a number of sources. This allows data transfer between the shifter and Vastor words to take place in one of the following four modes.

- ***Mode I:*** Vastor to shifter; parallel mode through the B port. In this mode, the data source can be the PE, the WK, or the BK (see Figure 4).
- ***Mode 2:*** Vastor to shifter; serial mode through the SI port. In this mode, up to eight bits of data can be

loaded from any word of a phrase into the shifter segment. This operation takes place in parallel for all phrases.

- ***Mode*** 3: Shifter to Vastor; parallel mode. Vastor words can be loaded in parallel from port B of the SH via the PE.
- ***Mode 4:*** Shifter to Vastor; serial mode. Eight bits of data can be moved serially from a shifter segment to any word in the corresponding phrase. This is accomplished via the OEN and the ability to circulate data within each of the eight-bit segments of the SH.

In the two serial modes (2 and 4 above), only one word of each phrase is involved in data transfer. This reduces the parallelism in the array by a factor of eight. However, the serial modes are necessary to simplify byte-oriented data transfer between Vastor and the host machine.

Associative tests. All Vastor operations can leave a result in register RR of the PE of each word. Contributions from all RR registers are summed, in an analog fashion, onto a single line. This simple scheme obtains a limited-accuracy estimate of the number of responders S, i.e., the number of words in which RR= 1. The most useful values for this number are zero, one, and more than one. A simple analog-to-digital converter extracts this information from the analog sum.

Examples of **vector operations**

The two example vector operations given below illustrate the capabilities of the Vastor array. The first is a vector addition; the second is an associative search for the largest element of a vector.

Let ***A*** and B be two vectors residing in the Vastor array (Figure 6). A third vector, R, represents the arithmetic sum of ***A*** and B and must be obtained. Information regarding vectors ***A*** and B is stored in a table in the controller. The table stores such relevant parameters for each vector as starting address in the array, number of elements, and number of bits.

The add operation begins when the host computer sends a high-level command specifying the operands ***A*** and B and the function to be performed. The host computer need not specify such details as the addresses of the operands, the number of elements, or the element lengths. Operands are identified by pointers into the operand table, which is stored in the controller. When the operation is complete, the controller returns to the host the value of the pointer corresponding to the result vector ***R.***

Addition is bit-serial and word-parallel The sequence of operations is given in Figure 7. As indicated in Figure 7, control of the sequence of operations and address calculations is exercised in the controller, while vector operations are performed in the array. The optional masking operation at the beginning of the sequence disables those words of the array for which the mask contains zeroes. This disabling process might be needed when the vectors involved contain a number of elements that is lower than the number of Vastor words. The mask used in such operations is set up when vectors ***A*** and B are created.

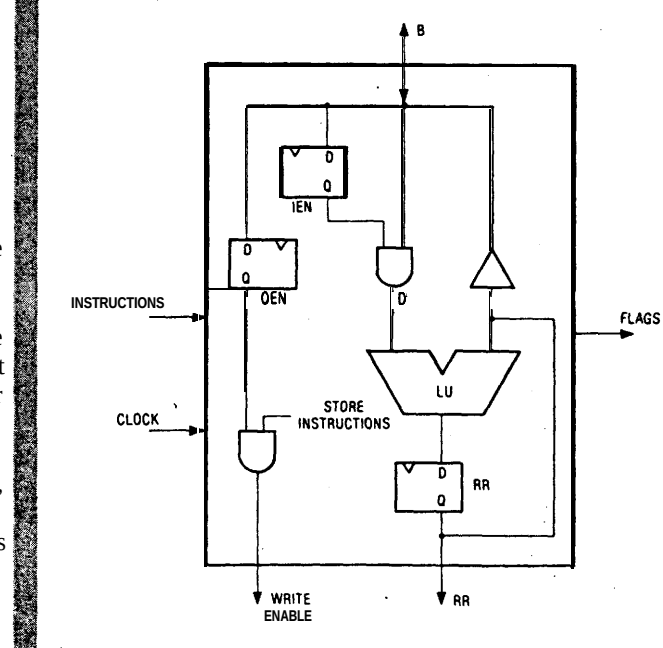
Selection of the microprocessor

We needed a simple microprocessor to serve as the processing element in Vastor. Since it had to be very cheap because 256 of them were to be used in a low-cost system. Although the prototype uses commercially available chips, one of the objectives was to produce a design well suited to eventual integration. This meant keeping the gate count to a minimum.

The SIMD approach reduced the complexity of each PE. It also made replication of functions related to instruction sequencing and operand addressing unnecessary. A central controller performs these functions for all the PEs in the Vastor array.

A bit slice is a suitable choice to provide the needed processing capability without incurring the overhead of address generation circuitry. The simplest chip on the market was Motorola's MC14500B industrial control unit. It resembles a conventional bit-slice chip but is one bit wide instead of the more common four. It also has special features related to its intended control applications. Vastor is an unusual application for the ICU, which has been marketed for use in relay replacement.

A block diagram of the ICU appears below. All data paths and storage elements are one bit wide. There is a simple accumulator, RR (the result register), and a logic unit. The LU performs seven simple boolean operations on the RR and on the input data bit D. The ICU has nine other instructions, three of which control the write enable line. In Vastor, this line controls memory write operations. The remaining six instructions are not used.



Block diagram of the ICU.

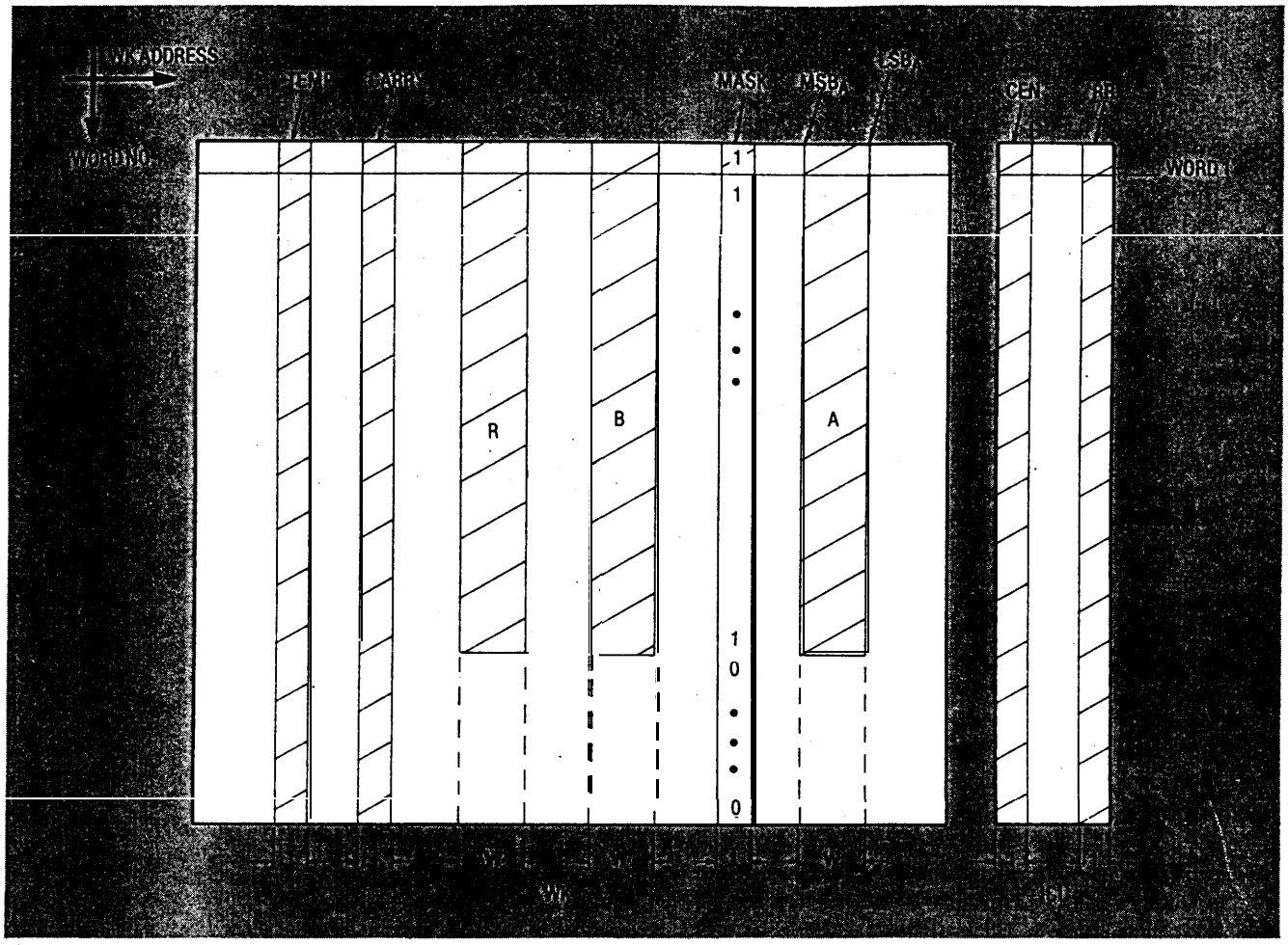


Figure 6. An example of data vectors in the working store.

An implementation of the binary search algorithm[5] for positive or unsigned integers is given in Figure 8. In this case, the elements of the vector are scanned, starting with the MSB. A one-bit-wide vector TEMP masks out the words that have been rejected at any stage of the search. The associative sum S is used to determine the first bit position in which one element of TEMP contains a one while all other elements contain zeroes. At the end of the search, TEMP contains one(s) in the word(s) containing the largest element(s).

The above examples illustrate operation on short vectors where all bits of a given element are contiguous in a field of one Vastor word. When there are more elements in a vector than words in the array, the vector can be

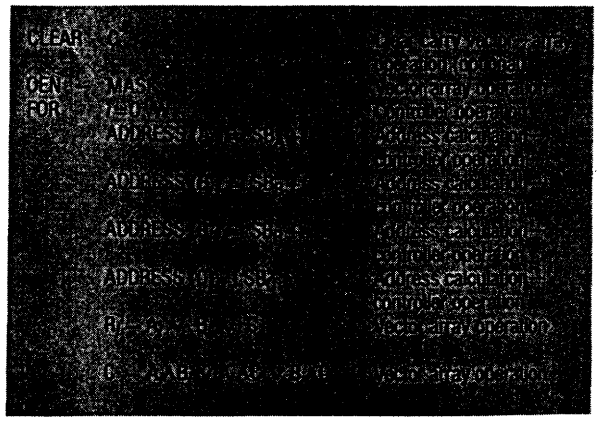


Figure 7. Implementation of vector addition.

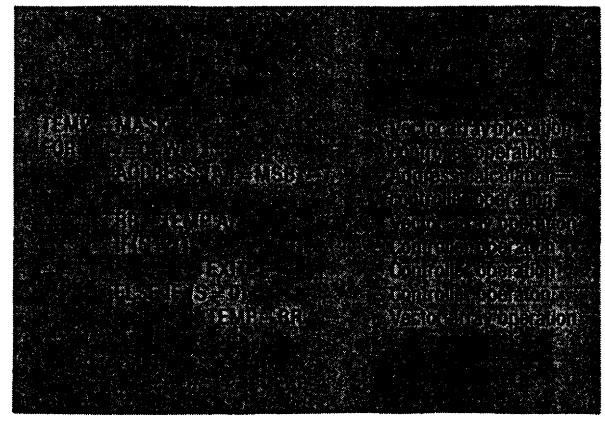


Figure 6. Search for the largest element.

broken into several subvectors, each of which is operated on independently. It is also possible for the elements of a vector to occupy two or more noncontiguous fields in a word. In this case, the controller repeats the operations on the different fields of the vector.

The controller

The controller reduces the control overhead incurred by the host in driving Vastor. To keep the Vastor array continuously active, 50 control bits are needed every microsecond. This represents a control bandwidth of 50 bits per microsecond, which is too high to be supported directly by the host. It exceeds, for example, the bandwidth of an entire PDP-11 Unibus. Hence, it must be reduced to a level that does not prevent the host from performing operations not related to Vastor.

This has been achieved with the organization shown in Figure 9. The controller receives high-level commands from the host machine, and these require a much lower control bandwidth. These commands are then translated into the sequences of control signals needed to drive the Vastor array.

The complexity of the commands interpreted by the controller is represented by the examples given in the previous section. We chose a hierarchical approach to support such operations. Each level in the hierarchy reduces the bandwidth required of the next higher level. Furthermore, interpretation of high-level commands is relatively simple, a result of the use of well defined interfaces between various levels.

The controller consists of three distinct units. A microcontroller performs low-level looping control operations, a buffer memory is a communications medium, and a microprocessor is responsible for both interpreting high-level commands received from the host and for space allocation within the Vastor array. As such, the microprocessor performs functions similar to those of the interpreter in ECAM[8]; the microcontroller corresponds to the iteration control logic. The three subsystems of Vastor's controller are discussed briefly below.

The microcontroller. The purpose of the microcontroller is to remove some of the redundancy at its output (the array control lines) in order to reduce the bandwidth required at its input. Its sophistication, and therefore its cost, can be selected to provide almost any desired bandwidth at its input. We chose to implement a device that executes sequences of microcode stored in an internal read-only memory and that has primitive branching and looping capability. Input commands to the microcontroller come from the buffer memory that, in turn, is filled by the microprocessor.

Linear microcode sequencing considerably reduces the control bandwidth. Hence, it was adopted as the main sequencing mechanism in the microcontroller. The starting address for a given microcode sequence is loaded from the buffer memory. Data can be made to appear in the Vastor array in fields of consecutive locations. Therefore, further compression of the control information is obtained with a simple loop counter/index register. This counter is decremented and tested to control microprogram loops. It also serves as an index register and modifies the addresses transmitted by the controller to the array memory.

Control bandwidth is further compressed by introducing a data-dependent branch. The associative sum of responders is compared to a reference in the microcode. One of two branch addresses is then selected from the buffer memory.

The buffer memory. The buffer memory is divided into 16 separate task control blocks. These blocks are filled by the microprocessor and interpreted by the microcontroller. Whenever the microcontroller finishes a task, it interrupts the microprocessor to request the address of the next control block. Task control blocks contain up to 26 bytes of information, including starting and loop control information for the microcode of the microcontroller. It also includes specifications for the operands in the Vastor array.

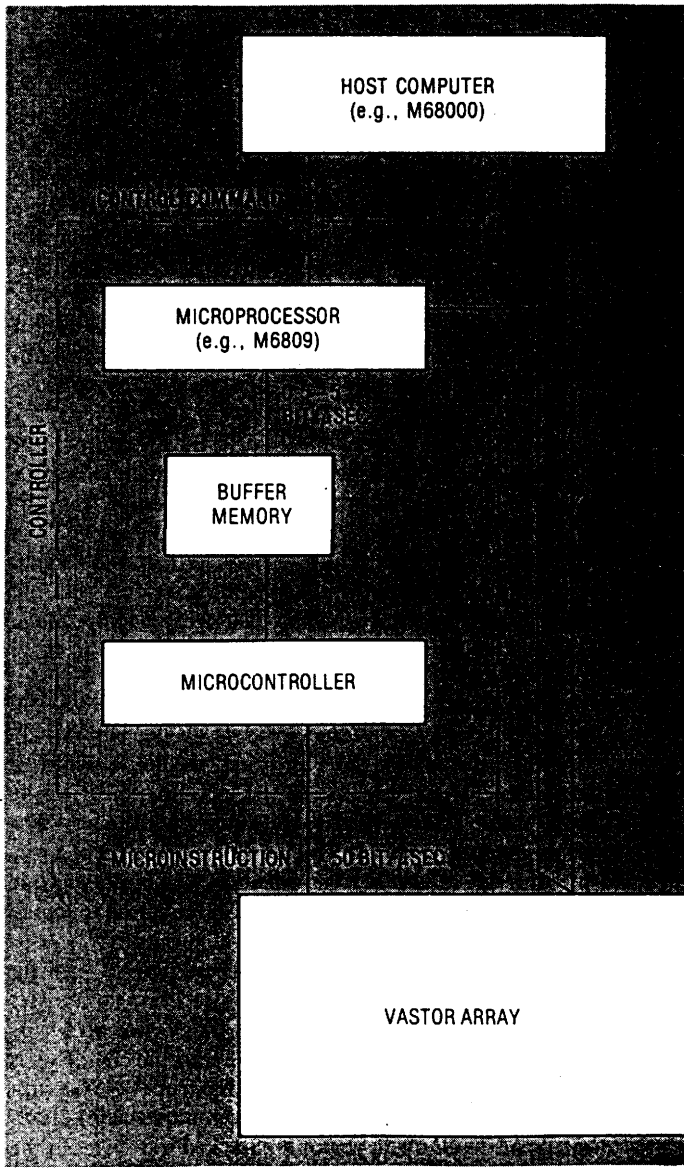


Figure 9. Controller hierarchy.

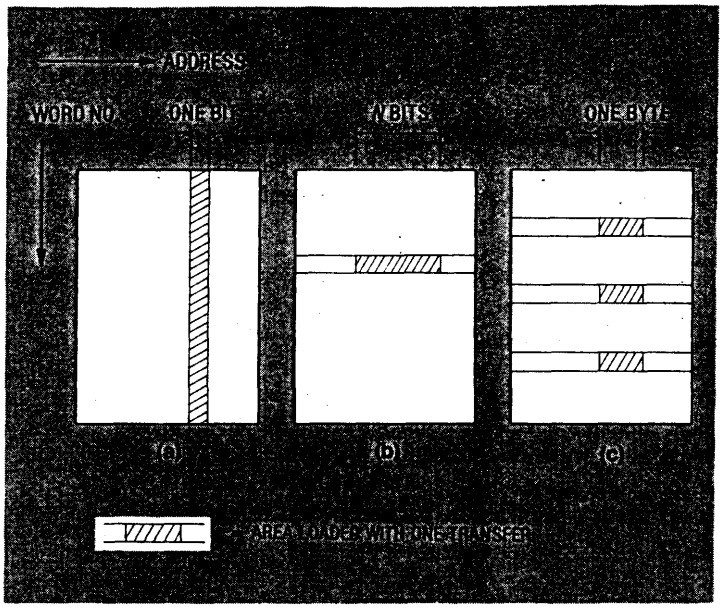


Figure 10. Alternative modes for I/O transfers: (a) one bit; (b) n bits; (c) one byte.

The microprocessor. Controller algorithms represented by one control block in the buffer memory take from one to several hundred microseconds to complete and to interrupt the microprocessor. These interrupts are usually quite simple to service, but would be uneconomically frequent for the host machine. The microprocessor is therefore included to provide further compression of the control bandwidth. It simplifies the interfacing hardware by translating high-level operations into sequences of microcontroller tasks.

In addition to sequencing control, the microprocessor handles storage management. This includes allocating and freeing fields of storage, garbage collection, transferring variables between the working store and the backing store, allowing the widths of elements (e. g., integers) to expand and contract, and segmenting vectors that are longer than the Vastor array into manageable components.

Input/output

Data transfer between Vastor and the host machine is generally difficult because of the incompatibility of the addressable units in the two machines. While a host machine generally obtains all bits of a single element of a vector with one reference to its memory, Vastor obtains one bit of each element. The transposition required to match the two machines is the source of the difficulty.

The boolean vector is the simplest type to transfer because it is only one bit wide (Figure 10a). To transfer such a vector from the host into the Vastor array, its elements are shifted serially by bit into the SH. This is followed by a parallel-mode transfer from SH to a column of WK (mode 3 of interword communication). If elements of the boolean vector are packed into bytes in the host machine-as is the case in some versions of APL-SH can be loaded serially by byte through its A port. In the current implementation, data rates for the bit-serial and byte-serial modes are one megabit per second and one megabyte per second, respectively.

Consider the case in which data is so presented to Vastor that some number of consecutive bits must be loaded into a single word (Figure 10b) This can be achieved by first loading register RR of the ICU from the CONST line (Figure 4) and then storing the content of RR in the enabled word. Due to that two-step sequence and to the fact that only one word is enabled at a time, the transfer rate is limited to 500 kilobits per second.

The phrase structure can be used to increase the transfer rate of byte-organized data, as shown in Figure 10c. This corresponds to mode 4. The data rate achievable in this case is 2.5 megabits per second. In this approach, consecutive words from the host machine are not loaded into consecutive words of Vastor. Rather, they are loaded into the same relative positions in consecutive phrases. A sentence structure consisting of two phrases per sentence also exists; it can be used for l&bit-wide I/O transfers.

Performance in application areas

Vastor's primary application is as an auxiliary processor in small computer systems. It would enhance the performance of such a system in vector and associative operations. A second and equally important potential application comes from the fact that Vastor can be regarded as a collection of one-bit-wide controllers driven in parallel by a host computer.

Vastor as an associative processor. Vector and associative operations are common in the operating system software of a computer. Symbol table manipulation and file management are two examples. Also, computer languages such as APL and Snobol are based upon the organization and manipulation of data in the form of vectors[10] or character strings.[7] A Vastor processor is ideal for such tasks and hence can relieve the host computer of a considerable load.

Table 1 gives an estimate of Vastor's performance in this area. The execution times are for a number of operations on 256-element vectors; each element is 16 bits wide (times are based on the current implementation). It uses a processing element, the ICU, which runs at a one-microsecond cycle time. The times required to perform the same operations in a PDP-11/45 minicomputer are given for comparison; Vastor is an order of magnitude faster at executing tasks that involve parallel operations on all elements of a vector. However, Vastor takes much more time on operations such as sum reduction (adding all elements of a vector). In this case, Vastor's performance is limited by its interword communication facilities.

Vastor's performance on sum reduction improves substantially on much longer vectors because many elements of the vector can be stored in the same word of the array. Hence, part of the addition can be carried out in parallel. For example, for a 2560-element vector, sum reduction requires only an additional 1440 μsec on Vastor (compare this to the 256-element case), for a total of 6816 μsec. The time required on a PDP-11/45 increases tenfold, to 3840 μsec.

Table 1.
Performance comparison: Vastor vs. POP-11/45.

OPERATION	RESULT	EXECUTION TIME (μSEC) VASTOR	PDP-11/45*
COMPARE	VECTOR†	4 μSEC/BIT • 16 BITS = 64	3.225 μSEC/WORD * 256 WORDS = 825.6
ADDITION	VECTOR	10 μSEC/BIT • 16 BITS = 160	1.9 μSEC/WORD • 256 WORDS = 486.4
MARK LARGEST ELEMENT	V E C T O R	3 μSEC/BIT * 16 BITS = 48	2.5 μSEC/WORD • 256WOROS = 640
COMPARE TO SCALAR	VECTOR	3 μSEC/BIT • 16 BITS = 48	2.5 μSEC/WORD • 256 WORDS = 640
SUM REDUCTION	SCALAR	336 μSEC/BIT • 16 BITS = 5376	1.5 μSEC/WORD • 256 WORDS = 384

*WITH BIPOLAR MEMORY
†VECTOR OPERATIONS INVOLVE 256-ELEMENT VECTOR WITH 16 BITS PER ELEMENT

At Vastor's present stage of development, it is difficult to accurately estimate the performance gain it would provide when added to a minicomputer system. While the data in Table! indicate a considerable gain, itwill be partially offset by the overhead of transferring data between Vastor and its host. This overhead is expected to be of the same order as that involved in transferring data between the main memory of a computer and a disk tile. Therefore, Vastor is most suited for use in applications in which a number of vector operations must be performed before a given vector is returned to the host machine.

Vastor as an industrial controller. A stand-alone ICU-the PE in the Vastor array-has applications in process control and monitoring. Vastor might be used in situations in which a number of ICUs performing similar tasks are interfaced to a common host. In such a case, Vastor represents an organized way of performing I/O and control functions. EachICU is capable of controlling and sampling data from an external device, at data rates on the order of a few kHz. Status information and data such as minimum values, maximum values, averages, setpoints, and enabling bits for each device can be kept in the corresponding working storage. The main limitation to this approach is that it requires synchronized data transfer between the ICUs and the various devices.

Physical characteristics

In the prototype system, 16Vastor words are housed on a single 8 x 1 l-inch circuit board. Thus, a full implementation of the array-256 words-requires 16 boards. Two more boards would be needed to house the controller.* The component cost, in small quantities, is about $2000. A minimal implementation of the machine requires three boards, one array board plus the two controller boards. The component cost is about $600. This minimal machine can be expanded in steps of 16 words, or one array board at a time. The incremental component cost is under $100. Since only bused and daisy-chained connnections are needed between circuit boards, expansion of backplane wiring is simple.

*Inthe prototype system, the controller was emulated by using a special interface to a PDP-11/34.

Conclusions

Vastor represents a trade-off between the capabilities and costs of the interword communication facilities in an associative processor. The result of this trade-off is a processor that allows a nontrivial associative processing capability to be incorporated in small-scale minicomputer systems. The communication hardware provided in the Vastor array enables data transfer among the words in the array without requiring costly and complicated hardware. It also results in simple backplane interconnections between different modules. The modular structure of Vastor allows its capabilities to be expanded easily and economically.

Some limitations of thecurrent implementation are due to the slowness of theICU. A faster and more powerful one-bit-wide PE could increase performance considerably, without requiring architectural changes. In fact, the low number of interconnections makes the structure well suited to integration. The possibilities include implementation of an array of one-bit processors and processors and memory on a single chip. Another possibility, which we are currently investigating, is a table-driven PE made of memory only.

Other limitations of Vastor, such as the difficulty of reordering a vector, are more fundamental. Performance of such operations at high speed will require a more complex, and hence more costly, interword communication scheme. ■

Acknowledgment

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada, under Research Grant A8994.

References

1. B. Parhami, "Associative Memories and Processors: An Overview and Selected Bibliography," *Proc. IEEE*, Vol. 61, No. 6, June 1973, pp. 722-730.

2. W. Shooman, "Parallel Computing with Vertical Data," *AFIPS Conf. Proc.*, Vol. 18, 1960 EJCC, pp. 111-115.
3. S. S. Yau and H. S. Fung, "Associative Processor Architecture-A Survey," *Computing Surveys*, Vol. 9, No. 1, Mar. 1977, pp. 3-27.
4. A. Kaplan, "A Search Memory Subsystem for a General-Purpose Computer," *AFIPS Conf. Proc.*, Vol. 24, 1963 FJCC, pp. 193-200.
5. C. C. Foster, *Content Addressable Parallel Processors*, Van Nostrand Reinhold Co., New York, 1976.
6. G. Baudet and D. Stevenson, "Optimal Sorting Algorithms for Parallel Computers," *IEEE Trans. Computers*, Vol. C-27, No. 1, Jan. 1978, pp. 84-87.
7. A. Mukhophadhyay, "Hardware Algorithms for Nonnumeric Computation," *Proc. 5th Ann. Symp. Computer Architecture*, Apr. 1978, pp. 8-16.
8. G. A. Anderson and R. Y. Kain, "A Content-Addressed Memory Designed for Data Base Applications," *Proc. 1976 Int'l Conf. Parallel Processing*, New York, 1976, pp. 191-195.
9. W. M. Loucks and W. M. Snelgrove, *"Vastor 1978,* "University of Toronto Computer Engineering Report 13, Toronto, Ontario, Canada, June 1978.
10. L. D. Grey, *A Course in APL 360 with Applications*, Addison-Wesley, Reading, Mass., 1973.

Wayne M. Loucks is a research associate in the Department of Electrical Engineering, University of Toronto, Ontario, Canada. He is currently involved in the development of a local-area computer network. His main interests are computer architecture, multiprocessors, microprocessor applications, and computer communications.

A member of the IEEE and the ACM, Loucks received the BASc in 1975 from the University of Waterloo, Ontario, Canada, and the MASc and PhD from the University of Toronto in 1977 and 1980, respectively.

Loucks' address is Dept. of Electrical Engineering, University of Toronto, 35 St. George St., Toronto, Ontario M5S 1A4.

Martin Snelgrove is a lecturer at the University of Toronto. His research interests include electronic circuits and filters, CAD for circuits, multiprocessor systems, and VLSI.

He obtained BASc and MASc degrees from the University of Toronto in 1975 and 1977, respectively, and is currently completing a doctorate. He is a member of the IEEE and of the Association of Professional Engineers of Ontario.

Safwat G. Zaky is an associate professor in the Department of Electrical Engineering, University of Toronto. His research interests are in computer architecture, hardware, and communications. Prior to joining the University of Toronto, he was with Bell Northern Research, Bramalea, Ontario, Canada, where he worked on applications of electro- and magneto-optics in telephone switching.

Zaky holds a BSc in electrical engineering and a BSc in mathematics, both from Cairo University. He obtained his MASc and PhD degrees in electrical engineering from the University of Toronto. He is a member of the IEEE and of the Association of Professional Engineers of Ontario.