

Second Thoughts on Parallel Processing

JOHN E. SHORE

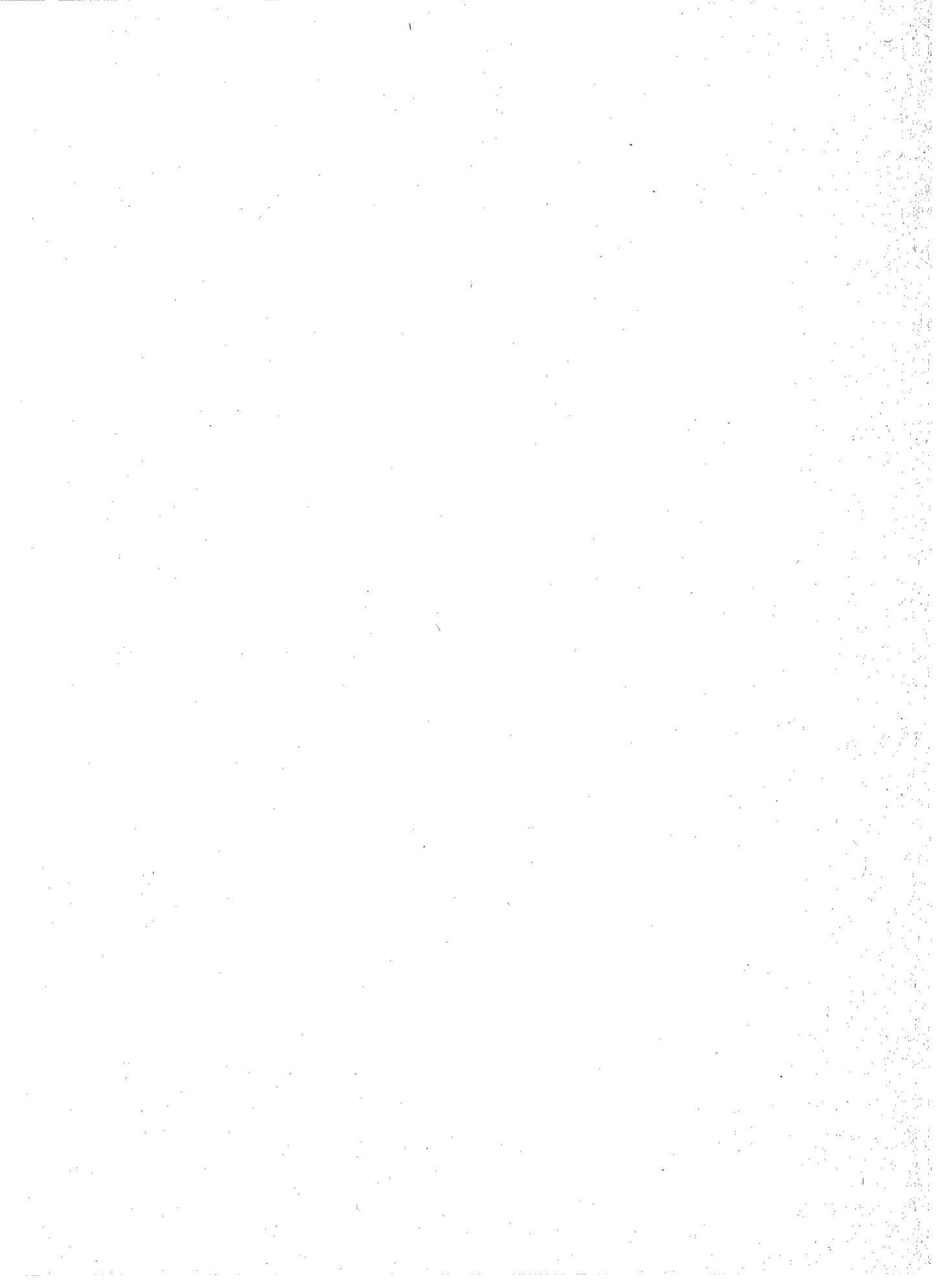
*Information Systems Group
Office of the Associate Director
of Research for Electronics*

December 30, 1971



NAVAL RESEARCH LABORATORY
Washington, D.C.

Approved for public release; distribution unlimited.



CONTENTS

Abstract	ii
Problem Status	ii
Authorization	ii
INTRODUCTION	1
COMPARISON OF ADVANCED, HIGH-THROUGHPUT ARCHITECTURES	1
HARDWARE VERSUS SOFTWARE	8
COMPLICATIONS	9
SUMMARY	10
ACKNOWLEDGMENT	11
REFERENCES	11

ABSTRACT

Consideration of advanced, high-throughput "parallel" and "associative" computer architectures has led to the following conclusions:

- Many of the advanced, high-throughput computer organizations being proposed and built today are usefully viewed as alternate ways of increasing the amount of processing hardware relative to the amount of memory hardware, i.e. increasing the processing ratio.
- In comparing alternate ways of increasing the processing ratio, a useful criterion is the duty cycle of the processing hardware, i.e. the proportion of the total amount of arithmetic and logical activity that is meaningful.
- For typical memory sizes, the higher the processing ratio, the fewer and more special purpose are the applications capable of using this ratio with a high duty cycle.
- An important consideration in the choice between hardware or software implementations of parallel instruction sequences is the amount of meaningful arithmetic and logical activity resulting from these sequences.
- An inability to find other than low-duty-cycle hardware implementations of "parallel" processes probably indicates a poor choice of algorithms and the possibility of attaining the same speed with less hardware in a conventional, sequential design using appropriate software.

PROBLEM STATUS

This is an interim report on continuing NRL problems.

AUTHORIZATION

NRL Problems B02-06, B02-09, and B02-10
Projects WF 15-241-601, XF 14-241-013, and XF 53-241-003

Manuscript submitted November 5, 1971.

SECOND THOUGHTS ON PARALLEL PROCESSING

INTRODUCTION

Subsequent to the work of Holland (1) in 1959, interest in so-called "parallel processing" has steadily increased. Among the better known designs involving multiple processors are the Orthogonal Computer (2), Solomon (3), ILLIAC IV (4), and Parallel Ensemble of Processing Elements (PEPE) (5). An alternate approach to "parallel processing" is by means of logic distributed in memory. Several logic-in-memory arrays have been described by Kautz (6,7). The use of a logic-enhanced memory in a computer system has been described by Stone (8).

Debate on alternative, parallel, high-throughput processing structures, although plentiful, has unfortunately remained inconclusive. There has been a tendency to focus debate on architectural details and to ignore fundamental issues. The mystique and special vocabulary of "parallel processing" tends to obscure such basic questions as: What do the various "parallel processing" approaches have in common? How can they be usefully compared? How can one tell if an application *requires* "parallel processing"? Can the proposed approaches be replaced by clever software? How can one tell if a particular "parallel processing" approach is well matched to the application?

In this report, an attempt is made to expose these issues in a manner not dependent on architectural details. A number of attitudes are proposed that may be useful in the analysis of existing or proposed advanced, high-throughput architectures.

COMPARISON OF ADVANCED, HIGH-THROUGHPUT ARCHITECTURES

Consider Machine I, a conventional sequential processor shown in Fig. 1. This class of computer consists of a control unit (CU), a processing unit (PU), an instruction memory (IM), and a data memory (DM). A single DM read produces all bits of any one word for processing in parallel by the PU. Let us create Machine II by rotating the DM and PU 90 degrees without rotating the organization of the data (Fig. 2). Each word is still stored horizontally, so that a single DM read now produces one bit from all words instead of all bits from one word. Let us define a bit slice as any set of bits exposed by a single vertical cut through DM. Likewise, a word slice is any set of bits exposed by a single horizontal cut through DM. Thus a memory address points to a word slice in

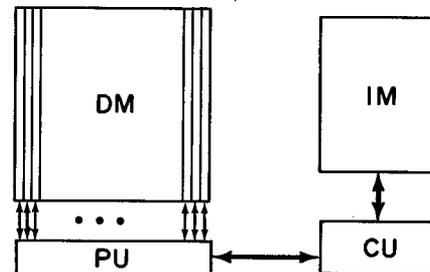


Fig. 1—Machine I, a conventional sequential processor, consisting of a data memory (DM), a processing unit (PU), a control unit (CU), and an instruction memory (IM). A memory address points to all bits of a word (word slice), the words being stored horizontally.

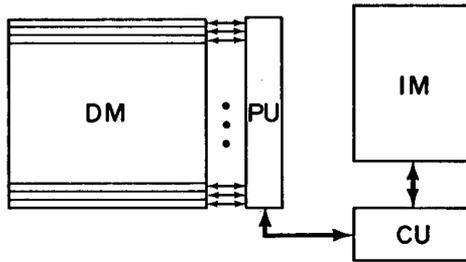


Fig. 2—Machine II. A memory address points to one bit from all words (bit slice). This machine has the organization of a bit-serial associative processor.

Machine I and a bit slice in Machine II. If both computers are to perform the same arithmetic operations on DM, the PU of Machine II must be modified to operate serially on the output of each DM word.

The organization of Machine II is precisely that of a bit-serial associative processor, a generally recognized class of “parallel processor.” The associative properties are obtained by including appropriate logic in the vertical PU. A number of processors that have been designed and built by Goodyear Aerospace Corporation are typical of this organization (9). With the exception of some modifications that optimize I/O, Goodyear’s commercially available STARAN system is basically a Machine II.

The outstanding difference between Machines I and II is that processing of DM is word serial, bit parallel in Machine I and word parallel, bit serial in Machine II. Machine II is often referred to as a “parallel processor” and Machine I as a sequential processor, but there is no intrinsic reason for this distinction. It is often stated that Machine II is a single-instruction-stream, multiple-data-stream processor, whereas Machine I is not. In fact, they both are. Machine I processes multiple-bit streams a word slice at a time, whereas Machine II processes multiple-word streams a bit slice at a time. The myopic association of multiple-data streams with multiple-word streams is a conceptual error having nothing to do with computing power.

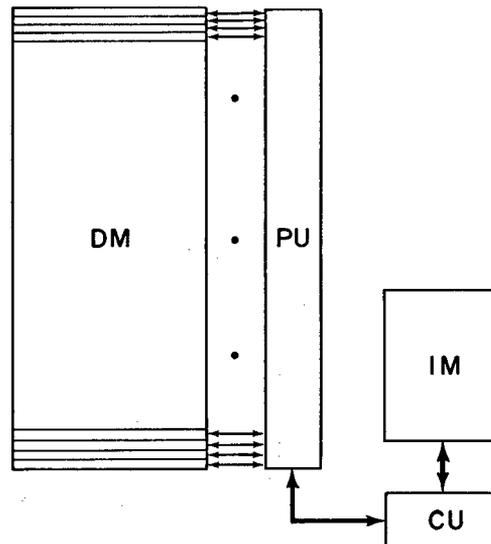
If the PUs of Machines I and II have the same complexity, then, at least to first order, Machine I processes a word slice in about the same time that Machine II processes a bit slice. Thus, if DM is square (n by n), both machines will have equivalent processing speeds, provided that each instruction is to be performed on every DM word. Whenever an instruction is to be performed on a predefined subset of DM, Machine I will be faster. This is because Machine II can operate no faster on one word than on all words, which is at the same time the best and worst feature of this architecture.

By restricting the discussion to square DMs, we have ignored the most important aspect of Machine II. Typical DMs have a vertical dimension (words) which is much larger than the horizontal dimension (Fig. 3). The PU in Machine I processes the parallel outputs of the short dimension, whereas the PU in Machine II processes the parallel outputs of the long dimension. The result is a significant speed advantage for Machine II whenever an instruction is performed on all words in DM. This may be stated in two ways:

- Data memories have fewer bit slices than word slices.
- Machine II has more arithmetic hardware than Machine I.

It is instructive to expand on the second of these. Most often, any increase in DM size is in the vertical dimension. In this context the fundamental difference between Machines I

Fig. 3—A more realistic portrayal of Machine II than shown in Fig. 2, since a square DM is not typical



and II is a difference in the ratio of processing hardware to memory hardware ("PU/DM"). Let us call this the processing ratio. For fixed word length the processing ratio of Machine I monotonically decreases towards zero as memory size increases. The processing ratio of Machine II is a constant. If an instruction is to be performed on all words in DM, the processing time in Machine I increases linearly with memory size. The processing time in Machine II is constant. Machine II can be used to best advantage in applications which involve many such instructions. In applications where operations are generally performed on only one word, the additional processing hardware in Machine II would be wasted.

The drastic change in processing ratio which results from orienting the PU in the vertical direction is bound to have a similar impact on cost. In most Machine II designs the processing ratio, and therefore the cost, is reduced by increasing the word length of DM. Word lengths of 128 to 256 bits are common.

Let us now combine Machines I and II into Machine III by providing both vertical and horizontal PUs and modifying DM so that the PU can access both bit slices and word slices (Fig. 4). Machine III has the organization of Shooman's Orthogonal Computer. Like Machine I, Machine III has a processing ratio which decreases monotonically with increasing memory. However, although the ratio for Machine I is asymptotic to zero, for Machine III it is asymptotic to a constant value equal to the processing ratio of Machine II, assuming the same word length and vertical PU. The processing ratios of Machines I, II, and III are qualitatively compared in Fig. 5.

Machine III combines the advantages of Machines I and II. The horizontal PU provides efficient execution of instructions on single DM words, whereas the vertical PU provides efficient execution of instructions on all of DM. This dual capability will result in higher throughput than either of Machines I or II. On the other hand the cost will also be higher, since the arithmetic hardware of both Machines I and II is required as well as the modification of DM to permit access in two directions.

Another organization, Machine IV (Fig. 6), is constructed by replicating the PU and DM of Machine I. A single CU broadcasts instructions to all PUs. PEPE is a well-known example (5). (The PU of PEPE is organized in two separate parts: a correlation unit and an

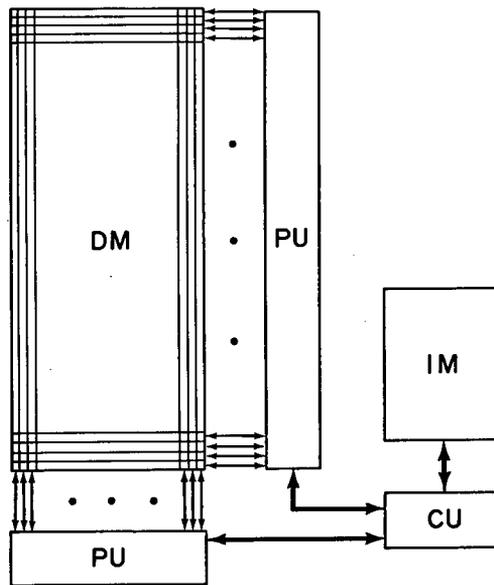


Fig. 4—Machine III

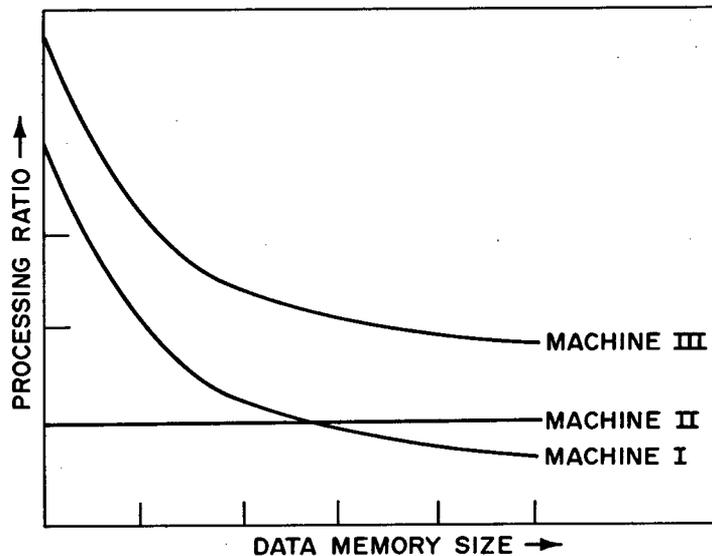


Fig. 5—Qualitative comparison of processing ratios ("PU/DM") for a fixed word length

arithmetic unit.) In general the size of DM is fixed and the system is expanded using additional sets of PUs and DMs. The processing ratio of Machine IV is therefore, like that of Machine II, constant. The actual ratio is determined by the size chosen for DM and the complexity of the PU.

There is little conceptual difference between Machines II and IV if the vertical PU of Machine II does not involve communication between DM words other than that required to resolve multiple associative response. In the case of no communication each DM word in Machine II will have one independent section of the PU devoted to processing its contents. A 256-bit-wide Machine II is, therefore, a linear unconnected array of word processing units (WPU), each with a 256-bit memory. If the 256 bit memories are organized as 16 by 16

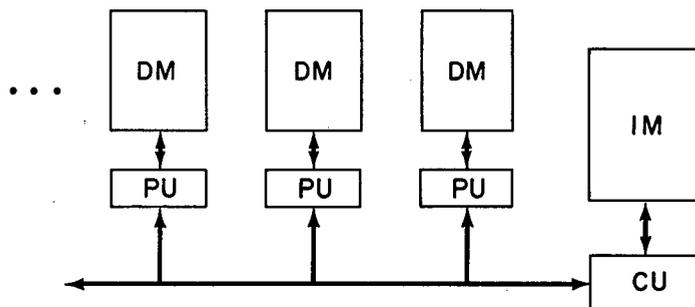


Fig. 6—Machine IV

instead of 1 by 256, then Machines II and IV look identical (Fig. 7). The only difference is that the WPU is bit serial, whereas the PU of Machine IV is generally bit parallel, operating on one word slice of its DM. However, this does not necessarily give Machine IV a performance advantage, since the increased complexity of its PUs relative to a WPU is usually compensated by a much larger DM than the several hundred bits associated with one WPU in Machine II designs.

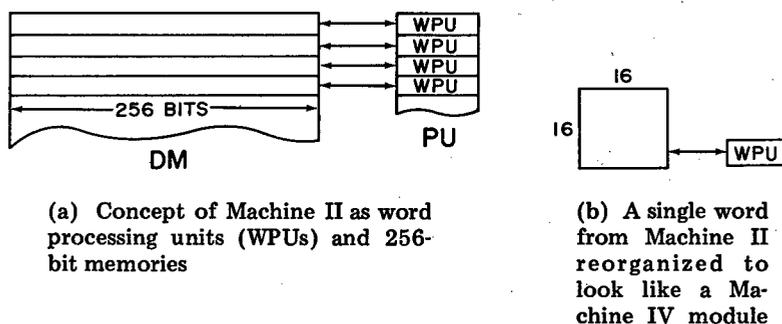


Fig. 7—Conceptual similarity of Machines II and IV

Machine IV permits communication between words within a single DM. More general communication capability is provided in Machine V, a linear connected array of PUs and DMs (Fig. 8). A single ILLIAC IV quadrant is a well-known example. Like those of Machines II and IV the processing ratio of Machine V is constant. Due to the horizontal connections the PU of Machine V is somewhat more complex than an otherwise equivalent Machine IV. Machine II designs which include interword communication in the vertical PU are conceptually the same as Machine V.

Machines II through V are examples of computer architectures in which the processing ratio has been increased by adding processing hardware external to the memory. The other approach is to increase the processing ratio by distributing the additional hardware throughout the memory. Organizations of this type, Machine VI (Fig. 9), are referred to as logic-in-memory-arrays (LIMAs), from simple associative memories to complex associative processors. Like Machines II, IV, and V, Machine VI has a constant processing ratio. Of all the organizations considered so far, the actual ratio of processing hardware to memory hardware is usually most apparent in Machine VI designs.

Machine VI designs with no communication between DM words are conceptually equivalent to Machine IV designs with one-word DMs. Similarly Machine VI designs with

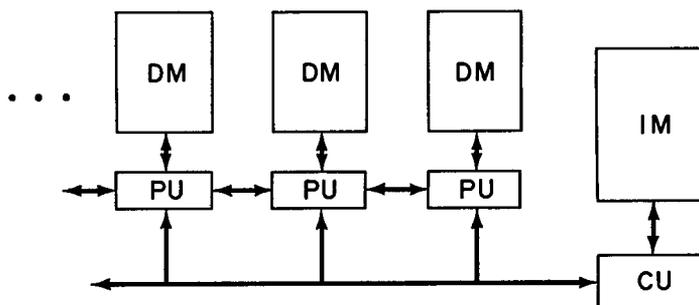


Fig. 8—Machine V

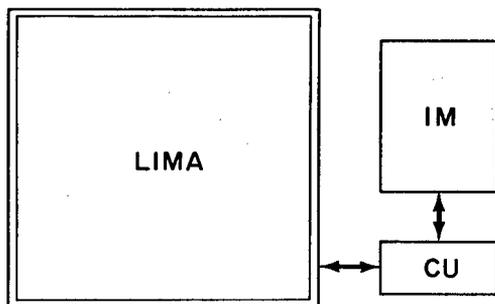


Fig. 9—Machine VI, containing a logic-in-memory array (LIMA)

communication between words are equivalent to Machine V designs with one-word DMs. Viewed this way, the difference between Machine VI and Machines IV and V may be thought of as simply a difference in packaging.

It is clear that each of Machines II through VI represents a different approach to increasing the processing ratio over that of Machine I, either by making the ratio constant or by adding a constant to it. Many of the advanced, high-throughput computers being proposed or built today that are not “conventional” (Machine I) can be recognized as an example of Machine II, III, IV, V, or VI. Unfortunately this is often obscured by new vocabulary, or new definitions of old vocabulary, that are introduced with each such design. Even the phrase “parallel processing” refers to a fuzzy concept. If a line between sequential and parallel processing does exist, it is probably between Machines III and IV rather than between Machines I and II as commonly stated, since Machine IV performs bit-parallel processing of multiple word streams. Terms such as orthogonal computer, associative memory, associative processor, and array processor, each of which means different things to different people, simply refer to particular means of increasing the performance of computers by increasing their processing ratio.

With this viewpoint it is much easier to compare the various advanced designs being discussed today. For the moment we shall put aside the complications introduced by such things as testing, reliability, architectural implications of LSI (large-scale integration) and tradeoffs among size, weight, power, speed, and cost (for example, the tradeoff between small quantities of fast, expensive state-of-the-art components and large quantities of slow, inexpensive components). Instead we shall simply consider alternate ways of assembling otherwise identical logic to satisfy requirements involving high throughput. The basic question, the answer to which may or may not depend on the application, is then really quite simple: What is the best way of increasing the amount of processing hardware relative

to the amount of memory hardware? Unfortunately, although the question is clear, the grounds on which to base the answer are not.

One criterion that is useful in comparing alternate designs is the degree to which the processing hardware is usefully busy. Many designs that include a large amount of processing hardware are inefficient in that many gates are often logically idle. A good example of this is an associative processor (AP) recently designed at NRL (10) for a specialized sorting application and then generalized somewhat (11) for potential use in the Advanced Avionic Digital Computer (AADC), an advanced computer presently under development by the Navy (12). The NRL AP is a logic-in-memory-array (Machine VI) with multifield search, logical, and arithmetic capability in each word. The logic design of the second version was implemented in a four-bit CMOS (complementary metal oxide semiconductor) integrated circuit chip containing approximately 150 MOS transistors per bit. Fourteen transistors are used for a flip-flop, giving the AP a processing ratio of about 10:1. This ratio is too high; for most applications the AP has too much processing hardware. This became apparent during development of algorithms for general avionic applications when we compared a 256-by-256 AP with a Machine IV structure of the same gate count. The logic equivalent to one 256-bit AP word has approximately 10,000 gates. This is the same number as in the arithmetic and logic section (PU) of one processing element (PE) of the AADC. One AADC PE is of the Machine I type with a PU capable of a 3-million-instructions-per-second throughput. Thus, if the processing ratio is decreased to 1:1 with the gate count remaining the same, the 256-by-256 AP could be traded for a linear array of 128 AADC PUs, each with a 128-word-by-16-bit DM. The available storage would be four times that of the AP. We preferred the linear array to the AP, simply because for most applications the processing hardware in the linear array of AADC PUs can be kept usefully busy more of the time than the processing hardware in the AP. Although the AP in principle has tremendous computing power, in practice this power can rarely be used. For example, all AP words are simultaneously capable of performing a large number of adds. In practice, however, an instruction results in a few adds performed in a few words. The add circuitry in all other words, while electronically active, is not meaningfully so and is therefore wasted. An example of more efficient processing hardware is a Machine I with a pipelined PU containing several sections each constantly busy executing one portion of an instruction. To borrow a concept from electrical engineering, the duty cycle of the AP's processing hardware is generally low, whereas that of the pipeline is usually high.

This does not mean that the AP is useless, only that it should be used in applications that result in a high duty cycle of the processing hardware. If an application results in a low duty cycle, many of the processing gates would probably be more useful elsewhere. Apparently, at least for typical DM sizes, the higher the processing ratio of a computer, the more special purpose it is in that there are fewer applications capable of running it with a high duty cycle. (This is *not* the same as saying that the faster a computer is, the fewer the applications are that require it.) In the comparison between Machines I and II we noted that Machine II is considerably faster than Machine I when an instruction is performed on all words in DM. If an instruction is performed on only one DM word, then Machine I is faster. When an instruction is performed on all words, the entire vertical PU of Machine II is usefully busy. When an instruction is performed on only one word, the PU section associated with one DM word is the only part of the PU of Machine II that is usefully busy. Clearly, Machine II is best applied to applications in which instructions are performed on all DM words, that is, to applications that run the PU with a high duty cycle.

The imprecise term "parallelism" has also been used in describing various large-scale, fast computers on the market today. Basically of the Machine I class, these computers

obtain high throughput with the use of multiple central processors, pipeline processors, multiple arithmetic units, and extended arithmetic units. Often, these techniques may be usefully compared by viewing them simply as alternate means of increasing Machine I throughput by increasing the processing ratio and using high-duty-cycle PU designs. Similarly, microprogramming is a means of increasing the duty cycle of PU hardware.

HARDWARE VERSUS SOFTWARE

Another question unfortunately makes the selection of architectures even more difficult: Is the additional hardware of Machines II through VI necessitated by the application under consideration or could all requirements be met with clever, optimum Machine I software? In answering this question, the concept of meaningful logical and arithmetic activity is again useful. Here one should consider the amount of such activity that is required by the application. If it is low, Machine I might be adequate if appropriate software is written.

As an example, it is often implied that if Machine II can be programmed for an application, then a throughput higher than Machine I will result. This is not true, especially if most Machine II instructions would result in an operation on one DM word and $N - 1$ "no-ops." In this case Machine I is preferable, since it can perform the single operation faster and with less PU hardware.

One might object that in many applications, although a large percentage of operations are performed on only one word, the particular word to be operated on is not known in advance and the additional processing hardware in Machine II therefore has its advantage in avoiding a time-consuming linear search that would be necessary if Machine I were used. This is often not true. A linear search is rarely necessary if the proper data organization and software retrieval techniques are used. One should not justify additional hardware by comparing the performance to that of Machine I performing a linear search. The comparison should be made with an optimally programmed Machine I. If it is claimed that there is no way to order a particular file so that appropriate software results in rapid access, that the only way to locate needed words is by a linear search, then one must draw the conclusion that the file is randomly filled and contains no information. In any realistic application this is not the case. Consider the example of radar tracking. The correlation of radar returns with existing tracks is often mentioned as a candidate for parallel processing, the correlation to be performed in parallel on all tracks. However, in beacon tracking, where the radar return is accomplished by IFF or transponder encoding, there is an unambiguous pointer to the correct track. This situation is not radically changed by the existence of uncooperative targets, such as encountered in the ballistic missile defense problem. If enough is known about the environment to maintain a track file, then it should be possible to identify quickly those few tracks that are candidates for updating upon the receipt of any radar return. This is particularly true in the case of modern phased-array systems in which sequential (Machine I) software techniques schedule the radar transmission used to update tracks.

To be more specific, consider the following instruction sequence:

1. SET TAG BIT 6 IN ALL WORDS THAT HAVE TAG BIT 5 SET AND THAT SATISFY THE FOLLOWING ADDITIONAL REQUIREMENTS:
 - A. CONTENTS OF FIELD 3 ARE GREATER THAN 19
 - B. CONTENTS OF FIELD 4 ARE LESS THAN 34

2. ADD THE CONTENTS OF REGISTER 7 TO FIELD 9 OF ALL WORDS IN WHICH TAG BIT 6 IS SET

Instruction sequences of this type would be extremely convenient in a wide variety of applications, regardless of whether they are implemented in hardware or in a compiler for Machine I. The choice between compiler and hardware implementations should be based on the amount of information that will actually be processed by this instruction sequence and others like it. This may or may not be sensitive to the application. Stated somewhat differently, it is a question of how much meaningful arithmetic and logical activity will result from such instruction sequences. This question has not been adequately answered for many of today's important applications. It may not have been asked.

Hardware implementations of parallel instruction sequences have the virtue of predictable execution times. This may be the basis of their current emphasis, since it is relatively straightforward to design the hardware to meet real-time requirements of the application. It is more difficult to develop techniques for compiling parallel instruction sequences into efficient Machine I code that meets the same real-time requirements.

On this basis one might argue that a low-duty-cycle PU is a price that must be paid if real-time requirements are to be met. This is not the case. An inability to design other than low-duty-cycle "parallel processors" for a particular application probably indicates a poor choice of algorithms and the possibility of attaining the same speed with less hardware in a Machine I design using appropriate software.

COMPLICATIONS

The concepts developed in the preceding discussion are by no means a panacea. Rather, they provide a useful point of view that must be modified by the variety of complications that accompany a realistic architectural comparison.

Any practical design is confronted with various tradeoffs among size, weight, power, speed, risk, and cost. However, if enough is known about the problem and proposed solutions to define these tradeoffs then they will modify the preceding results in a straightforward way. For example, if cost is the only consideration and existing components can be used, it may be cheaper to waste processing gates than to design and build components for an efficient PU.

In some applications, requirements for high I/O rates and potential interface costs may form the basis of the system design. All other things being equal, the best designs are often those that do not keep the processing hardware idle for long periods while I/O is taking place. In many cases this can be avoided by increasing DM (decreasing the processing ratio) and providing additional memory ports for direct I/O.

LSI has been controversial in its potential impact on computer architecture, and one might think that the emergence of LSI must modify the point of view developed in this paper. However, this is not the case. The fundamental impact of LSI is in releasing old constraints—not in replacing old constraints with new ones.

One of the most important aspects of LSI is that it does not strongly favor any particular way of putting gates together. Different processing structures will be considered more in terms of their own merits, perhaps in ways suggested earlier, and less in terms of

traditional technological constraints. Those who expect that LSI will obsolete conventional architectures lose sight of the fact that LSI has as many implications for these architectures as it does for others. One example is the AADC processing element mentioned earlier. It is true that certain cellular arrays (Machine VI) can result in more gates per unit area due to regular, short-run wiring requirements. However, this is a second-order effect relative to the more fundamental issues raised earlier.

Judging from the economics of existing integrated-circuit production lines, it is often concluded that the economics of LSI production will favor the use of multifunction arrays. Although this may be true as far as component manufacturers are concerned, to the cost of production they must add the cost of obtaining customer familiarity and acceptance. If a computer system is being produced, the latter costs are not highly dependent on the choice of old or new components, a choice more likely to center on the one-time costs associated with putting a component into production. These one-time costs are expected to become quite low judging from recently reported advances in the development of rapid-turnaround, fully automatic mask generation and processing systems.* If these one-time costs do become low, one can expect rejection of multifunction arrays in favor of specially designed components that increase performance while decreasing size, weight, and power requirements.

Another argument for multifunction arrays has been that their use would cut down the number of unique system parts, with resulting advantages in logistics and field support. However, significant advantages are obtained only with large reductions. Within a few years LSI parts are expected to be 3-inch packages containing about 10,000 gates, and only the most ambitious digital systems will consist of very many of them (13).

Certain processing structures are easier to test or more tolerant of failures than others. It is not clear how these factors will influence the use of LSI. However, an emphasis on fault tolerant design may be inappropriate for a technology outstanding in its ability to produce reliable components. Also, useful processing structures will be introduced irrespective of the ease of logical testing. This may lead to a change in emphasis from logical to operational testing.

The discussion of hardware versus software implementations of parallel instruction sequences is complicated by the increase of instruction memory required following a heavier reliance on software techniques. If the required size of IM is known, one can consider the total hardware required by the PU and IM when making architectural comparisons. In addition the software cost must be weighed against all costs associated with additional hardware.

SUMMARY

- Many of the advanced, high-throughput computer organizations being proposed and built today are usefully viewed as alternate ways of increasing the amount of processing hardware relative to the amount of memory hardware, i.e. increasing the processing ratio.
- In comparing alternate ways of increasing the processing ratio, a useful criterion is the duty cycle of the processing hardware, i.e. the proportion of the total amount of arithmetic and logical activity that is meaningful.

*Reference 13 is an excellent review of the present and future state of the art of LSI.

- For typical memory sizes, the higher the processing ratio, the fewer and more special purpose are the applications capable of using this ratio with a high duty cycle.
- An important consideration in the choice between hardware or software implementations of parallel instruction sequences is the amount of meaningful arithmetic and logical activity resulting from these sequences.
- An inability to find other than low-duty-cycle hardware implementations of "parallel" processes probably indicates a poor choice of algorithms and the possibility of attaining the same speed with less hardware in a Machine I design using appropriate software.

ACKNOWLEDGMENT

The author thanks T. Collins and P.H. MacGahan for helpful discussions and for critically reading the manuscript. He is especially grateful to B. Wald for stimulating and encouraging the thoughts that led to this report, participating in many helpful discussions, and critically reading the manuscript.

REFERENCES

1. Holland, J.H., "A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously," Proc. WJCC, May 1960, pp. 259-266.
2. Shoorman, W., "Parallel Computing with Vertical Data," Proc. EJCC, Dec. 1960, pp. 111-115.
3. Gregory, J., and McReynolds, R., "The SOLOMON Computer," IEEE Trans. on EC EC-12, 774-781 (Dec. 1963).
4. Barnes, G.H., et al., "The ILLIAC IV Computer," IEEE Trans. on Computers C-17, 746-757 (Aug. 1968).
5. Githens, J.A., "An Associative, Highly-parallel Computer for Radar Data Processing," pp. 71-87 in *Parallel Processor Systems, Technologies, and Applications*, Hobbs, L.C., et al., editors, Spartan, New York, 1970.
6. Kautz, W.H., et al., "Cellular Logic-in-Memory Arrays," Final Report, Pt. 1, SRI Project 5509, Contract Nonr-4833(00), Stanford Research Institute, Menlo Park, Calif., 1970.
7. Kautz, W.H., "An Augmented Content-Addressed Memory Array for Implementation With Large-Scale Integration," J. ACM 18, 19-33 (Jan. 1971).
8. Stone, H.S., "A Logic-in-Memory Computer," IEEE Trans. on Computers, C-19, 73-78 (Jan. 1970).
9. Meilander, W.C., et al., "A Mission Oriented Associative Processor Using Plated Wire," pp. 153-165 in *Parallel Processor Systems, Technologies, and Applications*, Hobbs, L.C., et al., editors, Spartan, New York, 1970.
10. Shore, J.E., and Polkinghorn, F.A., Jr., "A Fast Flexible, Highly Parallel Associative Processor," NRL Report 6961, Nov. 1969.
11. Shore, J.E., and Collins, T.L., "Another Associative Processor," NRL Report 7348, Dec. 1971.
12. Deerfield, A., et al., "AADC (Advanced Avionics Digital Computer) Arithmetic and Control Functional Block Diagram Design Analytical Study," Raytheon Company Report BR-6154, Naval Air Development Center Contract N62269-70-C 0210, Dec. 1970.
13. *Proceedings of the Advanced Digital Technology Conference*, 8-10 June 1971, vols, I and II, Naval Air Systems Command, Code 520022.

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)		2a. REPORT SECURITY CLASSIFICATION	
Naval Research Laboratory Washington, D.C. 20390		Unclassified	
3. REPORT TITLE		2b. GROUP	
SECOND THOUGHTS ON PARALLEL PROCESSING			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates)			
An interim report on continuing NRL Problems			
5. AUTHOR(S) (First name, middle initial, last name)			
John E. Shore			
6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS	
December 30, 1971	16	13	
8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)		
NRL Problems B02-06, B02-09, and B02-10	NRL Report 7364		
b. PROJECT NO.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)		
WF 15-241-601			
c. XF 14-241-013			
XF 53-241-003			
d.			
10. DISTRIBUTION STATEMENT			
Approved for public release; distribution unlimited			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY	
		Department of the Navy, Naval Air Systems Command and Naval Electronics Systems Command, Washington, D.C. 20360	
13. ABSTRACT			
<p>Consideration of advanced, high-throughput "parallel" and "associative" computer architectures has led to the following conclusions:</p> <ul style="list-style-type: none"> • Most of the advanced, high-throughput computer organizations being proposed and built today are usefully viewed as alternate ways of increasing the amount of processing hardware relative to the amount of memory hardware, i.e., increasing the processing ratio. • In comparing alternate ways of increasing the processing ratio, a useful criterion is the duty cycle of the processing hardware, i.e., the proportion of the total rate of arithmetic and logical activity that is meaningful. • The higher the processing ratio, the fewer and more special purpose are the applications capable of using this ratio with a high duty cycle. • An important consideration in the choice between hardware or software implementations of parallel instruction sequences is the amount of meaningful arithmetic and logical activity resulting from these sequences. • An inability to find other than low-duty-cycle hardware implementations of "parallel" processes probably indicates a poor choice of algorithms and the possibility of attaining the same speed with less hardware in a conventional, sequential design using appropriate software. 			

UNCLASSIFIED

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Data processing Computers General purpose computers Special purpose computers Computer components Computer systems hardware Computer storage devices Parallel processors Serial processors Computer architecture Computer design Large scale integration Associative processors						