Chapter 26

NOVA: a list-oriented computer[1]

Joseph E. Wirsching

Since the advent of the internally-stored program computer, those of us concerned with problems involving massive amounts of computation have taken a one-operation, one-operand approach. But there is a very large class of problems involving massive amounts of computation that may be thought of as one-operation, many-operand in nature. Some familiar examples are numerical integration, matrix operations, and payroll computation.

This article proposes a computer, called NOVA, designed to take advantage of the one-operation, many-operand concept. NOVA would use rotating memory instead of high-cost random access memory, reduce the number of program steps, and reduce the number of memory accesses to program steps. In addition it is shown that NOVA could execute typical problems of the one-operation, many-operand type in times comparable to that of modern high-speed random access computers.

Rotating memories were used in early computers because of low cost, reliability, and ease of fabrication. These machines have been replaced by machines with more costly random access memories primarily to increase computing speed as the result of a decrease in access time to both operands and instructions.

The NOVA approach

Let us take two simple examples and use them to compare conventional computing techniques with those proposed for NOVA.

Example 1. Consider two lists (a's and b's) of which the corresponding pairs are to be added. With a conventional computer this is done with a program that adds the first a to the first b, the second a to the second b, etc., and counts the operations. The working part of such a program might consist of the following instructions:

```
Fetch a
Add b
Store (a + b)
Count, Branch, and Index
```

[1]*Datamation*, vol. 12, no. 12, pp. 41–43, December, 1966.

In general, the four or more instructions must be brought from the memory to the instruction register once for each pair in the lists. This seems to be a great waste when only one arithmetic operation is involved. Indeed it is, when one considers that the majority of computing work consists of the performance of highly repetitive operations that are merely combinations of the simple example given. Attempts have been made to alleviate this waste by incorporating "instruction stacks" and "repeat" commands into the instruction execution units of more recent computers.

Example 2. Consider three lists (a's, b's and c's), where we wish to compute $(a + b) \times c$ for each trio. There are two distinct methods by which this can be accomplished: first, by forming $(a + b) \times c$ for each trio of numbers in the list, or second, by forming a new list consisting of $(a + b)$ for each a and b, and then multiplying each c by the corresponding member of the new list. Clearly the second method is wasteful of memory space and wasteful of programming steps.

Next, let us take a look at the memory requirements for these two examples. First, the instructions are kept in a high-speed random access memory, and while the bulk of the variables need not be kept in a random access memory, they must be brought to one before the algorithm can be performed. This extra transfer may entail more instructions to perform the logistics. Thus the simplicity of the overall program is directly related to the size of the memory. The variables (a's, b's, etc.) are usually stored in consecutive memory locations. Except for indexing this ordering of the data is not exploited.

In NOVA, lists of variables are kept on tracks of a rotating bulk memory. When called for, the lists of variables are streamed through an arithmetic unit and the results immediately replaced on another track for future use. This process takes maximum advantage of the sequential ordering of the variables. Instructions need only be brought to the instruction execution unit *once* for each pair of lists rather than once for each operand; thus the instructions need not be stored in a random access memory but may also be stored on the rotating bulk memory. This departure from the requirement for random access memory significantly reduces the cost of the computer, without sacrificing speed of problem solution.

Solution of a network problem

Before going further into the structure of NOVA, let us consider a significant example, which shows that NOVA is well suited to the solution of differential equations using difference methods over a rectangular network.

Let Fig. 1 represent an artificial network used as a model for some physical process. Generally speaking, the method of advancing the variables at a mesh point (j, k) from one time step to the next involves only information from the neighboring mesh points. A typical hydrodynamics problem will require a list of 10 to 20 variables (physical quantities) at each mesh point. The traditional computer solution involves listing these variables to each point in a contiguous fashion and in a regular sequence with respect to the rows and columns of the array. If the total array does not fit into the fast memory, three adjacent columns (or rows) are brought to the fast memory; as a new column is calculated, the next column in sequence is brought in from bulk memory and the oldest of the three is written to bulk memory. In this fashion one proceeds across the array. This process is then repeated until some significant physical occurrence happens and the problem is ended.

In NOVA, the variables are organized into separate lists rather than by mesh point. From a computational standpoint this is possible since the main memory of NOVA may be essentially unlimited in size, at least exceeding the size of the largest present network problems. One then proceeds to execute operations on lists of variables rather than single variables, performing a single operation for all mesh points in the array in sequence.

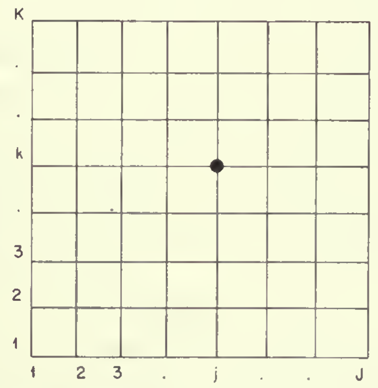

Fig. 1. Two-dimensional array.

Original Lists	*V Shifted Down by 1*	*V Shifted Down By 2*	*V Shifted Down By K*
$U_{0,0}$ $V_{0,0}$	—	—	—
$U_{0,1}$ $V_{0,1}$	$V_{0,0}$	—	—
$U_{0,2}$ $V_{0,2}$	$V_{0,1}$	$V_{0,0}$	—
· ·	$V_{0,2}$	$V_{0,1}$	—
· ·	·	·	—
$U_{1,0}$ $V_{1,0}$	$V_{0,K}$	$V_{0,K-1}$	$V_{0,0}$
$U_{1,1}$ $V_{1,1}$	$V_{1,0}$	$V_{0,K}$	$V_{0,1}$
$U_{1,2}$ $V_{k,2}$	$V_{1,1}$	$V_{1,0}$	$V_{0,2}$
· ·	·	·	·
· ·	·	·	·
· ·	·	·	·
$U_{j,k}$	$V_{j,k}$	$V_{j,k-1}$	$V_{j-1,k}$
· ·	·	·	·
· ·	·	·	·
· ·	·	·	·
$U_{J,K}$ $V_{J,K}$	$V_{J,K-1}$	$V_{J,K-2}$	$V_{J-1,K}$
	$V_{J,K}$	$V_{J,K-1}$	·
		$V_{J,K}$	·

Fig. 2. Lists of variables.

Let us look more closely at the variables and their possible combinations. Let $U_{j,k}$ *and* $V_{j,k}$ be variables associated with the array of Fig. 1. These variables are listed sequentially by column in Fig. 2, along with further lists of the V column shifted by various increments.

With some concentration, one discovers in Fig. 2 that an arithmetic operation between $U_{j,k}$ and $V_{j,k}$ is simply a matter of taking the two columns as they exist and operating on them in pairs. To combine $U_{j,k}$ with a nearby neighbor, $V_{j,k-1}$, the V column is shifted down one place, at which time the proper neighboring variables are found opposite one another for the entire network. At certain boundaries of the array some elements have no proper neighbors. In NOVA these boundary elements must be handled separately in the same way as they must be handled separately in a conventional machine. In NOVA, calculations at boundaries may be temporarily inhibited by having a third input to the arithmetic unit which allows the calculation of a result for a pair of operands to proceed or not, as appropriate. This third input is defined as "conditions," and is brought as a bit string to the arithmetic unit concurrently with the operands. This bit string may contain any number from one to several bits for each pair of operands.

Further observation shows not only that it is possible to obtain the nearest neighbors easily by shifting the columns of variables with respect to one another, but that any neighbor relationship can be obtained. In general, for an operation with a neighbor $\pm n$ rows away and $\pm m$ columns away, the lists are offset by $\pm n \pm m \cdot K$, where K is the number of rows in the array.

Many problems (for example, payroll and inventory records) are essentially list-structured but do not require offsetting of variables. Clearly the NOVA structure is well suited for the solutions of these problems also.

Structure

The most difficult problem to be solved in the proposed computer is to synchronize movement of the columns of data that require offset. Buffers of various types could be used to solve this problem; they could range all the way from rotating memory devices or delay lines to core memories. The former are simple, direct, and low in cost but are limited in their general capabilities. On the other hand, a number of small random access buffer memories could be used for offsetting lists of variables and for facilitating special functions such as boundary calculations but at a higher equipment cost.

Figure 3 shows a block diagram of the organization of NOVA. The rotating memory, which might be a disc or drum, would be composed of several hundred tracks, each storing several thousand words, with a total capacity between one and two million words. Each track would have an individual read-write head. The heads would be organized in such a way as to attain a high word-transfer rate, perhaps as high as one million words per second. With this in mind an ideal execution time for one addition would be the time required to move two operands from the disc to the arithmetic unit; i.e., 1-2 microseconds. The disc synchronizer would be capable of simultaneously reading two lists of operands, writing one list of results, and reading one list and writing one list of conditional control information. In addition, instructions would be read from another channel in small blocks.

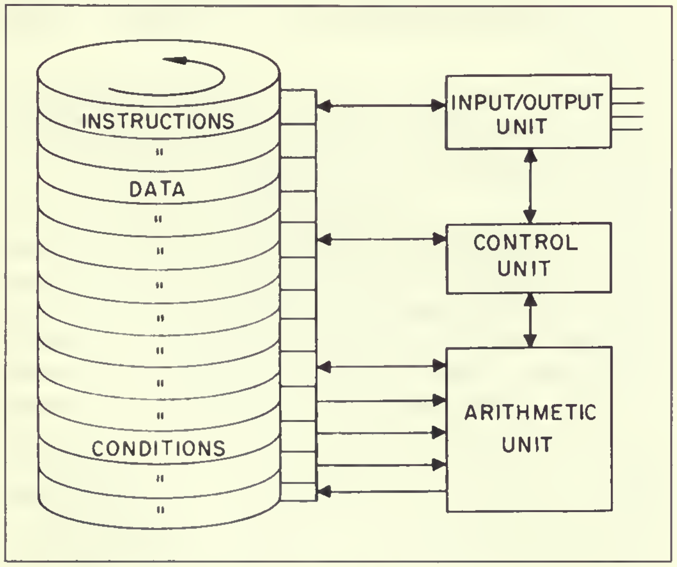

Fig. 3. Block diagram of NOVA computer.

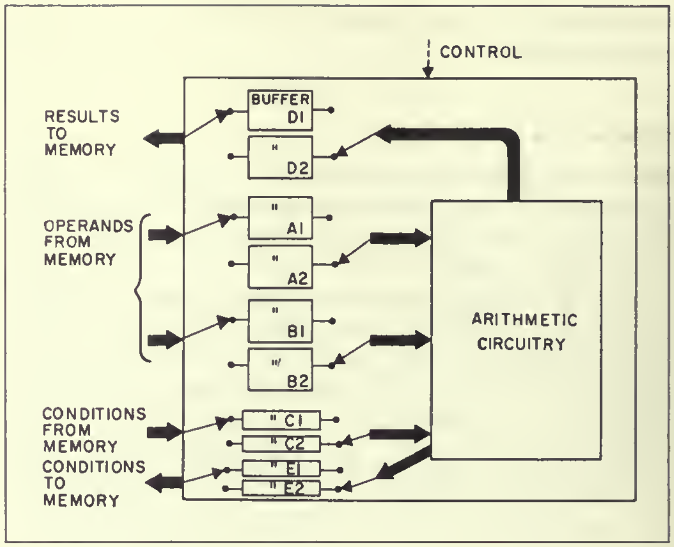

Fig. 4. Buffering in arithmetic unit.

The bit string of conditions coming from the memory is used to control individual operations on pairs of operands in the lists, and in essence each bit (or bits) is a subordinate part of the individual operations. Conditions going to the memory are the subsidiary result of the operation of one list upon another. These bit strings may be used later as control during another list operation. They want also to contain information on the occurrence of an overflow or underflow, or on the presence of an illegal operand, etc.

Figure 4 shows a suggested organization for the arithmetic unit that incorporates five sets of alternating buffers. Two sets are for lists of operands coming from the memory, one set for lists of results going to the memory, and two sets for "conditions" (conditional control information) coming from and going to the memory.

These buffers should be equivalent in length to the number of words on a track of the rotating memory.

The loading and unloading of the buffers to and from the rotating memory is dependent on the timing of the rotating memory, whereas the loading and unloading of the buffers to and from the arithmetic unit is guided solely by the rate at which the arithmetic can be performed. Here again it may also be possible to take advantage of the streaming nature of the operands by designing an "assembly-line" arithmetic unit in which more than one pair of operands could be in process at the same time. With this kind of unit it may be possible to execute additions at a rate equal to the word-transfer rate from the rotating memory; however, a multiplication or division of two lists may require several revolutions of the memory. The timing diagram of Fig. 5 shows several typical instructions being carried out. A certain amount of look-ahead is required, but there is ample time for this, since instructions are prepared for execution at an average rate of less than one per revolution of the rotating memory.

While a detailed cost estimate has not been made for a simple prototype NOVA, a quick estimate would be \$50,000 for a head-per-track disc and \$50,000 for the arithmetic and control section, making a total of \$100,000. For a buffering scheme such as the one shown in Fig. 4 the cost would be considerably higher but would be offset by increased versatility.

Conclusions

In the previous paragraphs we have demonstrated that NOVA is capable of handling network problems at a significantly lower cost than contemporary computers, and at a comparable speed. The availability of such a machine as NOVA would stimulate further interest in the one-operation, many-operand approach to computation and no doubt would uncover many other problems to which it could be applied.

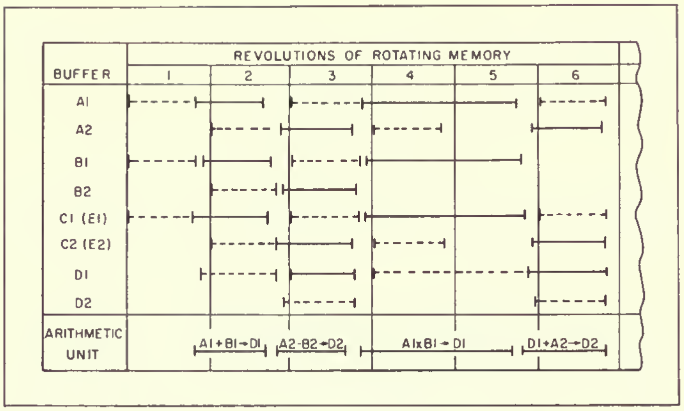

Fig. 5. Timing diagram of buffers, rotating memory, and arithmetic unit. Dotted line shows movement of data into a device; solid line shows movement out.

Because NOVA makes it possible to easily establish neighbor-relationships between mesh points that are further away than nearest neighbors, it may be possible to develop new differencing techniques for the solution of coupled sets of differential equations. This may increase the accuracy or shorten the time required for their solution.

The memory, arithmetic, and other units needed for NOVA are commercially available now. No new technology would be required to fabricate a prototype model. In view of the potential advantages of such a machine, it seems clear that construction of a model would justify the minimal development costs.