LBL-15958
c.2

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

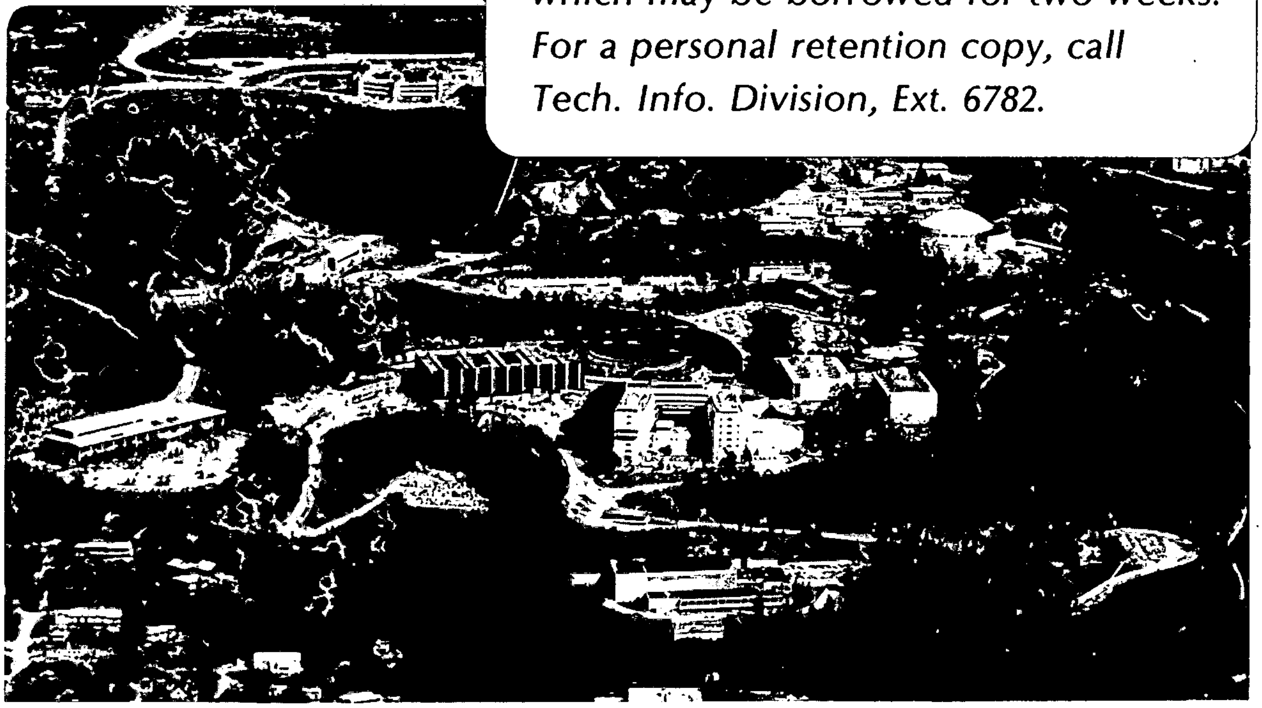
Engineering & Technical Services Division

Presented at the IEEE Nuclear Science Symposium, San Francisco, CA, October 19-22, 1983; and to be published in the IEEE Transactions on Nuclear Science

AN INTERACTIVE PARALLEL PROCESSOR FOR DATA ANALYSIS

J. Meng, D. Weaver, C. Maples, W. Rathbun, and D. Logan

October 1983



Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

LBL-15958
c.2

LBL-15958

AN INTERACTIVE PARALLEL PROCESSOR FOR DATA ANALYSIS

J. Meng, D. Weaver, C. Maples, W. Rathbun, and D. Logan

Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

Abstract

A parallel array of eight minicomputers has been assembled in an attempt to deal with kiloparameter data events. By exporting computer system functions to a separate processor, we have been able to achieve computer amplification linearly proportional to the number of executing processors.

Introduction

Empirically, it is true that much of today's nuclear science results in multi-parameter data recorded verbatim on line to be later replayed for complete and detailed analysis. Furthermore, whereas a decade ago such data would have typically been fewer than eight parameters per event, today's data tends to have more than eight parameters per event and in some cases is hundreds or even thousands of parameters per event. The results of this trend for data analysis systems are two fold. First, collection of more parameters per event implies a search for very complex correlations. This in turn implies that to be truly effective, data analysis will often require some interactive intervention on the part of the experimenter. Second, the physical volume of data with which such interaction is desirable becomes very large. These two results are conflicting. Effective interaction requires fast response times, but large raw data volumes imply longer data perusal times for the data analysis system. The response of many experimenters to this conflict is a continuous and expensive subsidy to computer manufacturers in the form of ongoing purchases of more and more machines used by fewer and fewer experimenters with poorer and poorer results. Data analysis requirements remain many paces ahead of the ultimate capacity of available sequential computing machines.

In an attempt to solve this dilemma, we have paralleled eight stripped-down minicomputers in a configuration* which demonstrably gives the resulting system greater than eight times the processing speed of a single minicomputer. Furthermore, the array is highly modular, allowing repair, replacement or upgrading of major components without major downtime periods or major system overhauls. It is planned to assemble the 8-processor array into an array which shares a common interactive processor; one programmed to interface with several users simultaneously. The system is expected to defy obsolescence by virtue of its ability to have its relatively inexpensive high-speed processing modules upgraded as new ones appear, and by virtue of its use of a separate processor dedicated to user interaction. The user-interaction processor, its main virture being its program instead of its hardware, is not expected to become rapidly obsolete.

* The processor described is a prototype of a module intended to eventually be part of an array of such modules. The project has been named MIDAS, for Modular Interactive Data Analysis System, and all its various parts are referred to as such in much of the literature.

This paper is a description of our initial eight-processor array and some results based on the data analyses presently being run through the system.

Hardware

As Fig. 1 illustrates, the heart of the configuration is the crossbar-like connection of sixteen memories to any system processor. System processors include pipelined processors for inputting and outputting data, the array of minicomputer central processors, a special sorting module (not yet implemented), and each memory is given its own zero processor, allowing it to be cleared rapidly.

The crossbar configuration allows data to be switched through the system in blocks. After being filled from the input pipeline, the memory (4096 words by 32 bits in the present system) is switched to become part of the address space of an available minicomputer central processor, after which it is switched to be unloaded by the output pipeline processor. Rather than being centralized, the gating which effects the crossbar is distributed among the sixteen memory modules, thus relieving interconnect congestion.

Control of the crossbar is centralized, however, illustrated schematically in Fig. 2. A microprocessor is used to remember the desired sequence of processors. It also remembers when data must exit from the system in the same sequence in which it entered. Using these remembrances, it empties and then loads the First-In-First-Out (FIFO) memory devices used to control the crossbar connections during a processing pass.

Before the system is started, the sequence processor loads a list of four-bit numbers into the FIFO associated with the first process to be executed. For example, codes representing memories zero through fifteen would normally be loaded into the FIFO controlling the memory-zeroing operation. Since memory-zeroing is independent of all other operations, this process would initiate for each memory in rapid sequence. As a memory finishes being zeroed, its unique tag is passed into it's exhaust FIFO where its presence flags the sequence processor. The sequence processor, using its pre-loaded list of processor sequence, removes the memory code from the zero-processor exhaust FIFO and puts it into the next-prescribed processor FIFO; usually the input pipeline processor FIFO. Upon finishing with a memory, the input pipeline processor causes the memory code to be passed to its exhaust FIFO, once again flagging the sequence processor. In this way, the memory passes circuitly throughout all the prescribed processors. If the sequence processor has been told that the sequence of data entering the system must be the same as that of data exiting the system, it remembers the sequence of memory tags going into the connect FIFO for the input pipeline processor, and uses this to quarantee the same sequence of memories passing into the connect FIFO for the output processor pipeline. Within the system, since eight minicomputer central processors are processing eight different blocks of data simultaneously and since the analysis time may be dependent on the data, there is no guarantee that the sequence of data going into this processing block will be retained on release.

The crossbar in the prototype system is 5 x 16. If each minicomputer central processor were given one axial element of the crossbar, it would have become 12 x 16 and we felt the volume of gates and interconnections would be impractically large, so we compromised by using only one crossbar axial element for all eight minicomputer central processors. This is a reasonable compromise because the central processors use the crossbar only when accessing data. Program memory is local to each processor and does not share the crossbar memory interface. Also, average central processor access time is about 600 ns, whereas data memory cycle time is 200 ns. Consequently, memory accesses are relatively far apart and take a relatively short time to complete, making a shared access route functionally practical with very little sacrifice in performance. Connection of the eight central processors to a single crossbar axial bus is achieved using a high-speed time-slicer which asynchronously samples all eight memory requests every 170 ns, pausing 50 ns for a transfer when a request is active (see Fig. 3).

The input pipeline processor is a shift register running on a 4 MHz clock. In the 250 ns between clock edges, discrete logical operations are performed. Thirty two of the forty entering bits are reserved for data and pass through the pipeline unaltered. The remaining eight bits are used as control or status lines. Each stage contains some logic peculiar to the assigned function of the stage. For example, one stage is used to detect the beginning of an event. It contains logic to select which bits of the data stream to monitor for the event-beginning tag, and a high-speed Random Access Memory (RAM) preprogrammed with the truth-table for logic required to detect the specified code. Results of the test may be put onto one of the eight status lines, to be counted when setting up memory storage addresses in the final stage of the pipeline. The final stage of the pipeline contains a 4096 x 12-bit high speed ram used to generate storage addresses for data. The twelve bits of ram address may be driven either from an event counter or from a selected part of the data word or from some combination of the two. The RAM is programmed to put each incoming event on one of a set of preselected fixed address boundries, thus greatly enhancing the speed of data analysis when the memory arrives at its central processor unit.

The output pipeline processor is not a pipeline in the prototype unit (see Fig. 5). It is a counter used to generate memory addresses and some logic used to detect codes added to the data by the central processor units and used to select either a destination for the data or an end of the data buffer. Future plans call for a genuine pipeline as the output processor to enhance the flexibility of the system. Data from the system may be sent either to a histogramming memory or to a choice of two bulk storage devices, or to any combination simultaneously.

Programs which run in the parallel CPUS are compiled on a separate computer system, and only the required run-time code is down-loaded into the parallel CPU's. Since this separate computer looks like an operator at the control panel of each of the parallel CPU's, it has absolute control. After downloading a program, the separate computer system puts the start address into the appropriate hardware register and turns on the RUN switch. Implementing the computer - CPU connection in this manner makes it unnecessary to have any resident code in any of the parallel CPU's. This prevents 'overhead functions'--those normally associated with operating systems and computer input/output--from robbing us of actual computing power. The result is a system whose actual computing power increases linearly with the number of parallel CPU's running.

To date, our user programs have generally been written in Fortran, and most of them are copies of Fortran programs running on other nonparallel processors.

Actual data analyses are being run through the system. Early results, based on comparing analysis time on an independent minicomputer with the time required to do the same analysis on our parallel processing system indicate a speed amplification greater than nine. Analysis results have been compared and verified. The independent minicomputer uses the same make and model of central processor as that being used in the parallel-processor array. This particular data analysis was fairly well matched to the capacity of the system. The system was mostly compute limited.

Endpoint tests have been run on the system. By running programs which guarantee the system will be limited by compute time, and then turning on central processors one at a time, we have plotted a linear function of data throughput rate versus number of running central processors[3] (Fig. 6). We have run the system with very short programs in the central processors, and have observed the data input rate limiting at the capability of our bulk storage device: a 300 Mbyte disk. At the other end, the system limits at the incrementing rate of our histogramming memory--about 1 μs per increment.

Conclusions

By connecting processors in a parallel array within a structure that excludes the normal overhead associated with running operating systems in independent processors, we have demonstrated our ability to multiply processing capacity by the number of processors executing in the parallel array.

Acknowledgments

Principal members of the dedicated group who helped develop, build and test this system are Fong Gin, Bill Jackson, Tom Merrick, and Tom Shimizu.

This work was supported by the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. Reference to a company or product name does not imply approval or recommendation of the product by the University of California or the U.S. Department of Energy to the exclusion of others that may be suitable.

References

1. Meng, J. D., "Controlling a Radially-Connected Array of Minicomputers," Conference Record, Sixteenth Asilomar Conference on Circuits, Systems and Computers, pp. 280-284, Nov., 1982. Also Lawrence Berkeley Laboratory Report No. LBL-14471.

2. Maples, C., Rathbun, W., Weaver, D., and Meng, J., "A Design of MIDAS - A Modular Interactive Data Analysis System," Conference Record, Topical Conference on Computerized Data Acquisition in Particle and Nuclear Physics, 1981. Also Lawrence Berkeley Laboratory Report No. LBL-12504.

3. Maples, C., Weaver, D., Meng, J., Rathbun, W., and Logan, D., "Utilizing a Multiprocessor Architecture - The Performance of MIDAS,". To be published in the IEEE Trans. Nucl. Sci. NS-31, No. 1, Feb. 1984.

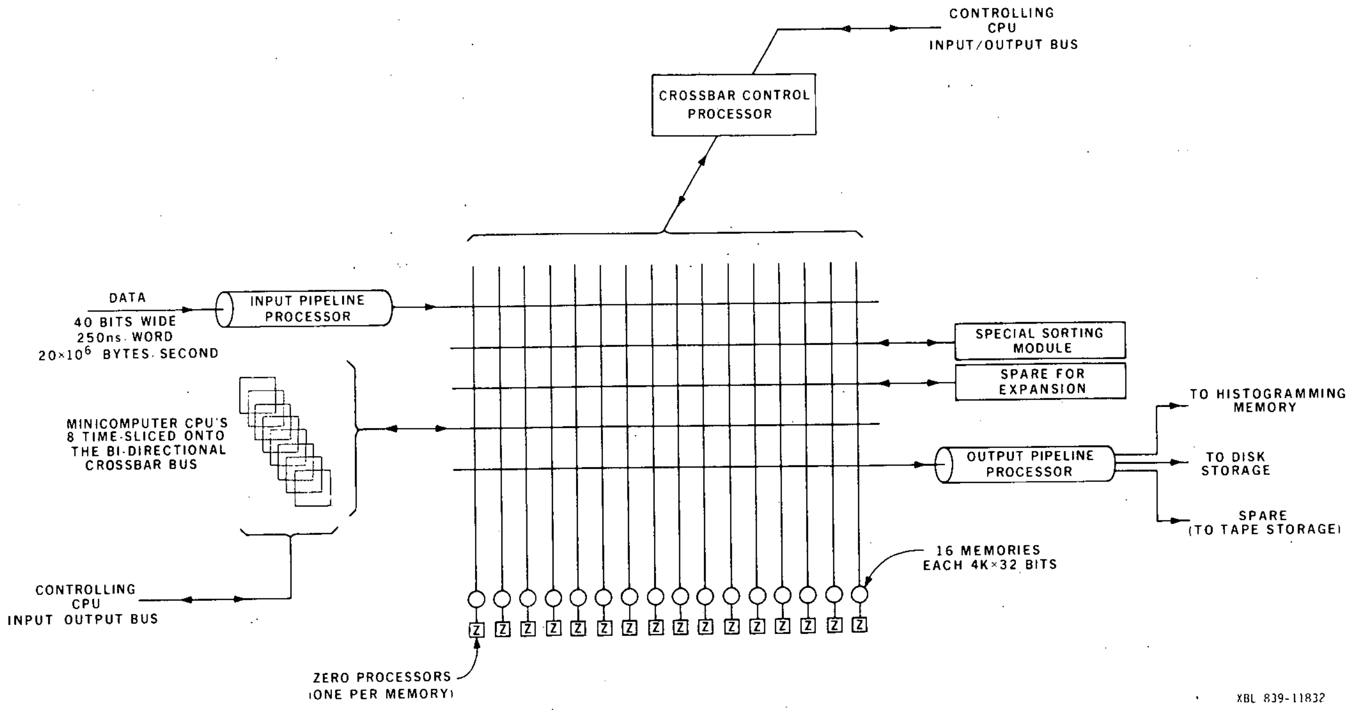

CONTROLLING CPU INPUT/OUTPUT BUS
CROSSBAR CONTROL PROCESSOR
DATA
40 BITS WIDE
250ns WORD
20×10^6 BYTES SECOND
INPUT PIPELINE PROCESSOR
MINICOMPUTER CPU'S 8 TIME-SLICED ONTO THE BI-DIRECTIONAL CROSSBAR BUS
CONTROLLING CPU INPUT OUTPUT BUS
SPECIAL SORTING MODULE
SPARE FOR EXPANSION
OUTPUT PIPELINE PROCESSOR
TO HISTOGRAMMING MEMORY
TO DISK STORAGE
SPARE (TO TAPE STORAGE)
16 MEMORIES EACH 4K×32 BITS
ZERO PROCESSORS (ONE PER MEMORY)
XBL 839-11832

Fig. 1 The processor array consists of an input pipeline, output pipeline, special sorting module (not yet built) and zero-processors used to clear each memory. A crossbar connects processors to memories. The crossbar control processor sequences the connections, guaranteeing no interference.

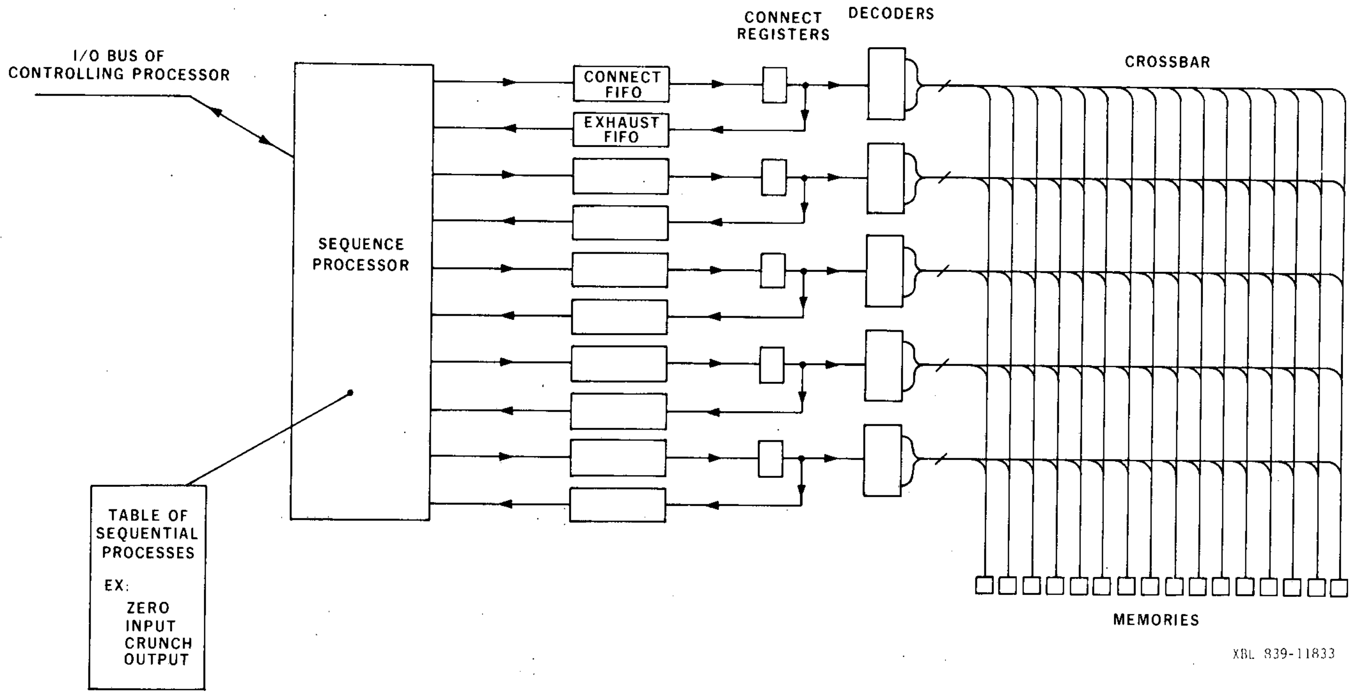


Fig. 2 The crossbar control processor is a microprocessor/discrete-logic hybrid. The microprocessor remembers desired and actual sequences and uses FIFO's to save its decisions until the hardware executes a switch. An actual switch requires less than 50 ns.

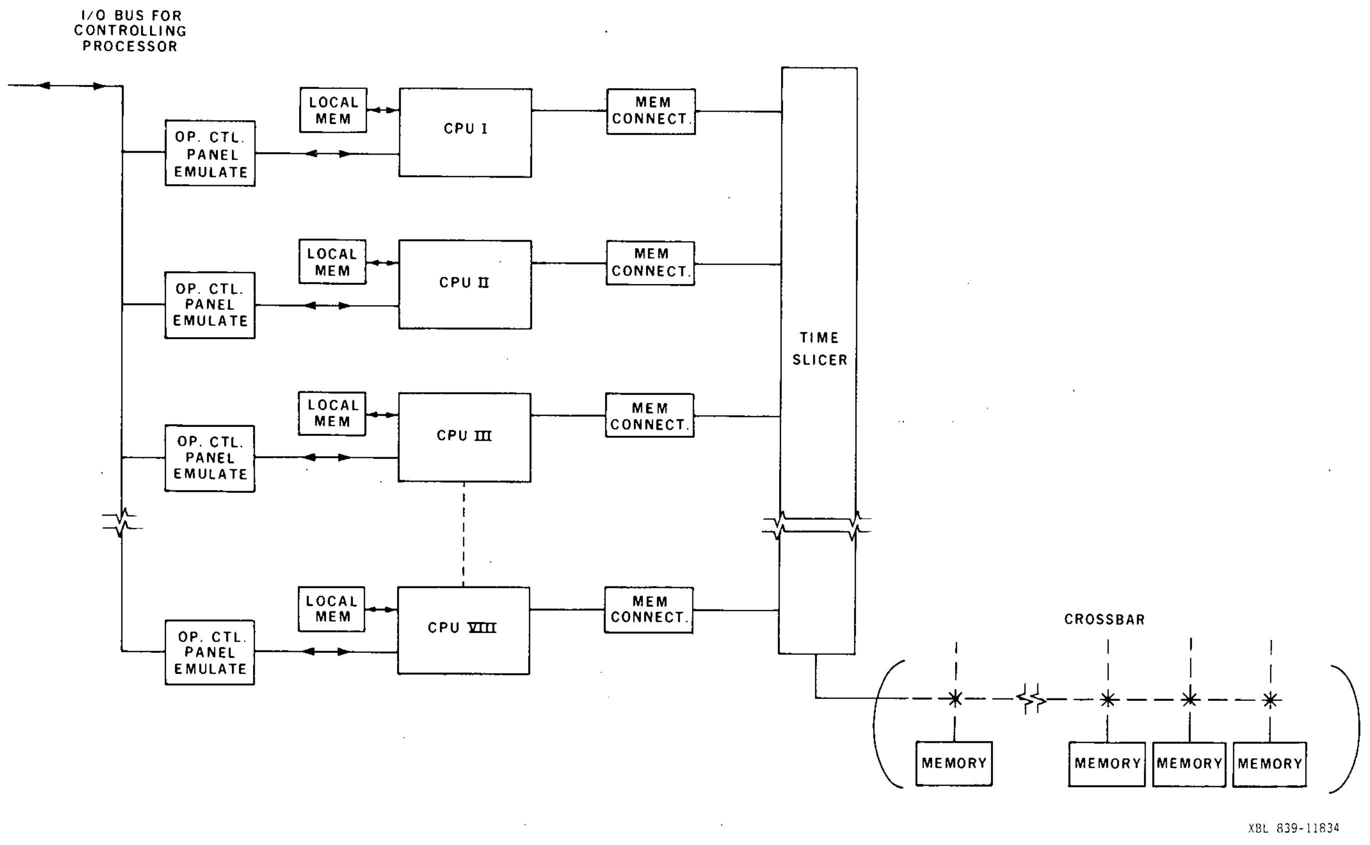


Fig. 3 Central processor units require relatively very little memory access compared with the 200 ns cycle of the memory chips. The time-slicer distributes one crossbar connection among all eight mini-computer central processors with very little processor performance degradation.

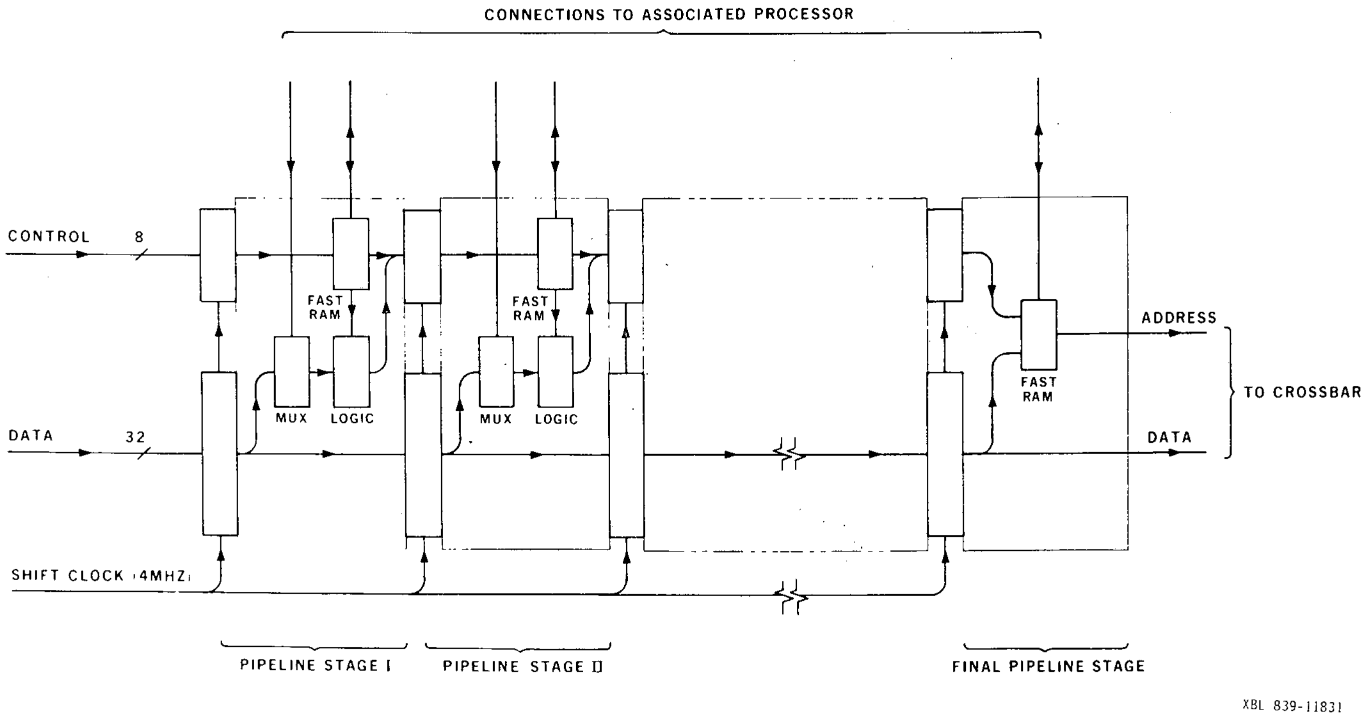


Fig. 4 The input pipeline is a shift register containing simple processing logic between stages. Logic is used to detect and divert headers and comments. It finds event boundries and places them at pre-defined locations in the memory into which it empties.

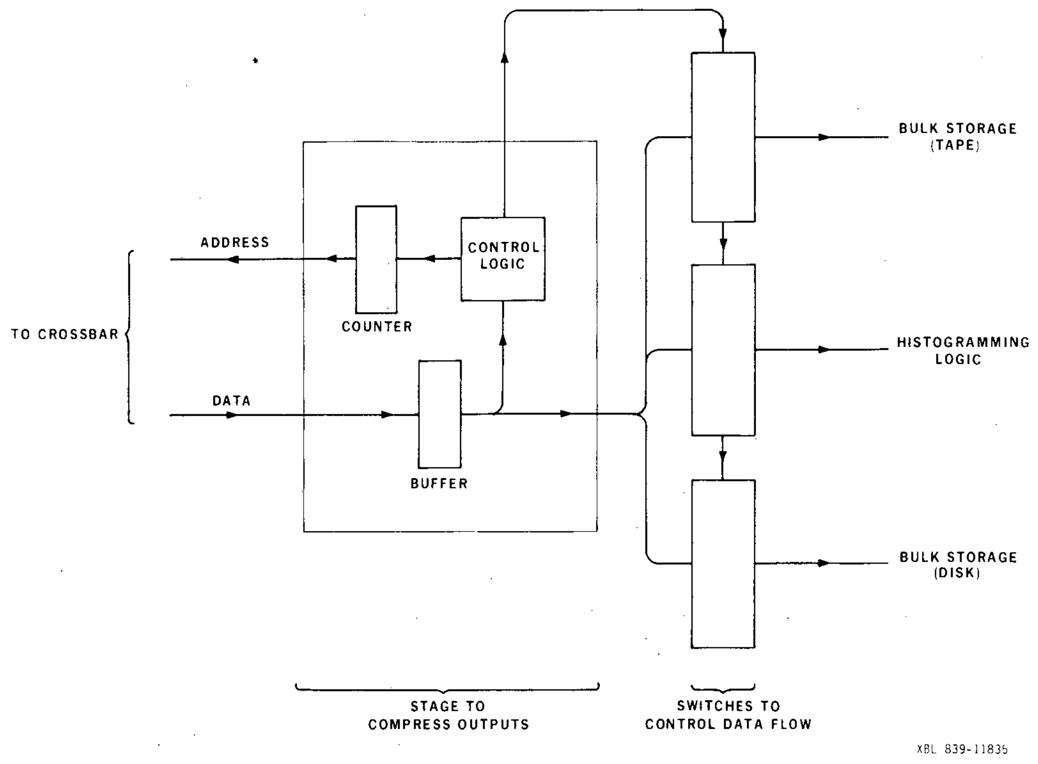


Fig. 5 At the output processor, only valid data is selected for passage either to histogramming memory or to bulk storage.

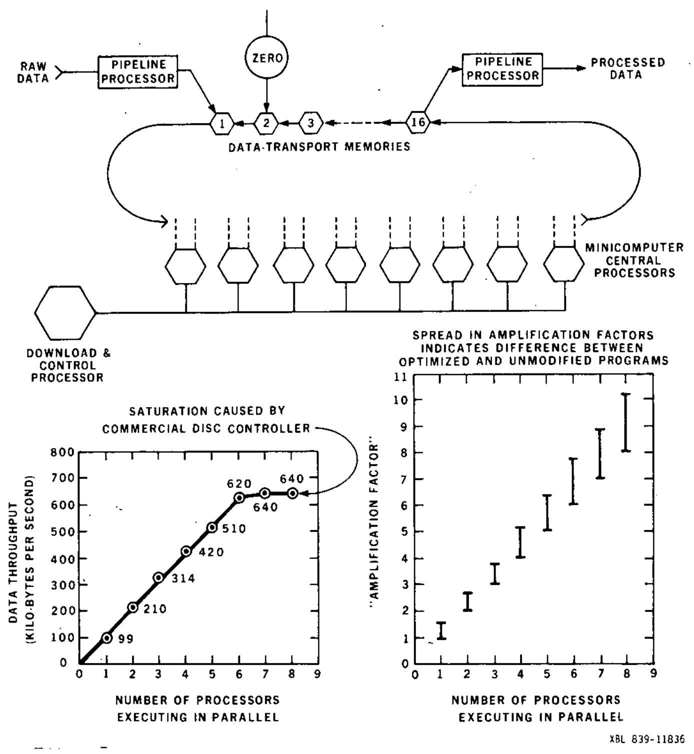


Fig. 6 Data running the loop as shown is analyzed 8-10 times faster than the same data being run through the same program in a single processor. Multiple processors find it relatively easy to out perform our commercial 300 MByte disk controller, as shown on the left graph.