CONF-790549--8

DATA ACQUISITION AND ANALYSIS SYSTEM FOR THE HOLIFIELD HEAVY ION RESEARCH FACILITY

W. T. Milner, J. A. Biggerstaff, D. C. Hensley, and R. O. Sayer*

Abstract

The Holifield Heavy Ion Research Facility is a national resource which will serve a large number of nuclear and atomic physicists who expect to perform experiments which vary widely in type and complexity. Although much consideration must be given to the problem of rapid acquisition and processing of many-parameter data, an equal emphasis will be placed on operational simplicity and the standardization of hardware and software. Two "active" experimental counting areas and two or more "setup" areas are served by three remotely located Perkin-Elmer 8/32 computers which are interfaced to the user equipment by means of three CAMAC branch highways. Other equipment includes a large disk system, alphanumeric/graphic terminals and printer-plotters located in each of the counting areas. The system operation as well as techniques for the rapid sorting of data into large (∿ 10 million channels) histograms on disk are discussed.

Introduction

The Holifield Heavy Ion Research Facility is expected to be utilized by about 50 resident and 150 non-resident researchers to perform a wide variety of experiments in nuclear and atomic physics. We anticipate that some experiments will require the accumulation of several single parameter histograms of up to 8 k channels each at total data rates of 100 kHz or greater. Others will involve the acquisition of multiparameter event data ("typically" around 16 parameters per event at data rates of several kHz) together with the nearly-real-time processing of part of this event data into a large number (∿ 100) of one- and two-parameter histograms whose total array size may exceed 20 million bytes. A few experiments may require the acquisition of data with more than 50 parameters per event. It is, of course, always desirable and sometimes essential to be able to quickly retrieve and display selected portions of the histograms being accumulated.

General Characteristics of the System

The Holifield facility contains two experimental counting areas, one associated with the booster cyclotron (ORIC) and one with the 25 MV tandem electrostatic accelerator. Since the demand for facility time is expected to be great and many experimental groups will require a significant amount of setup time using the data acquisition system, we will provide service to one "active" station and one "setup" station in each of these two counting areas. A block diagram of the system is shown in Fig. 1. The dashed block indicates that CPU C has not been acquired at this time. Interfacing to the data acquisition equipment will be by means of three CAMAC branch highways (one connected to each CPU through a dual DMA channel). The system is designed to provide the "active" user private access to a CPU (A or B). "Setup" users may be required to time share the resources of CPU C. Distributed data processing (in the general sense) is not planned but the user's data will be accessible from all CPU's through the use of shared disks. Crates located in the cyclotron area will be tied to highways A and C and those in the tandem area tied to B and C. Thus, switching from "setup" to "active" status can be simply a software operation.

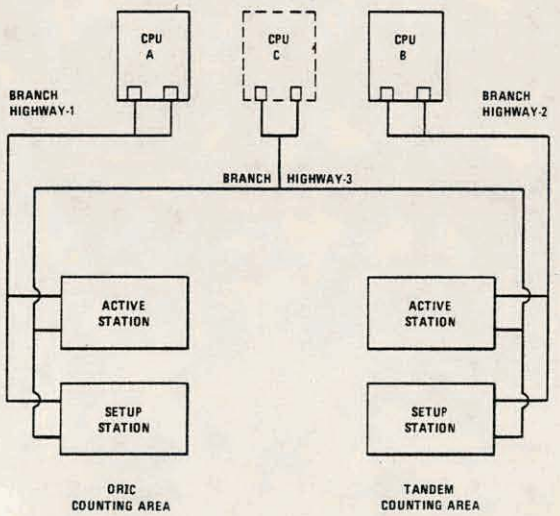

Fig. 1 Block diagram of system.

*Oak Ridge National Laboratory, Oak Ridge, TN 37830. Research sponsored by the Division of Basic Energy Sciences, U.S. Department of Energy, under contract W-7405-eng-26 with the Union Carbide Corporation.

DISCLAIMER

The Data Acquisition Station

Fig. 2 shows a block diagram of one of the data acquisition stations. Most acquisition devices (ADC's, scalers, TDC's, etc.) are available as CAMAC modules, and others may be easily connected to the CAMAC system by means of input registers. The heart of the acquisition station is the EVENT HANDLER (EH) developed by one of the authors (DCH) and described in another paper.[1] Briefly stated, the EH is composed of a programmable processor which is physically independent of CAMAC and an auxiliary crate controller with which the processor communicates. The main function of the EH is to read the parameters of an event from multiple CAMAC addresses and buffer them (FIFO) in order to permit the host computer to do fixed-address block transfers. The dashed region of the dataway (Fig. 2) is intended to indicate that the EH as well as all data input devices may reside in a crate which is separated from the branch highway.

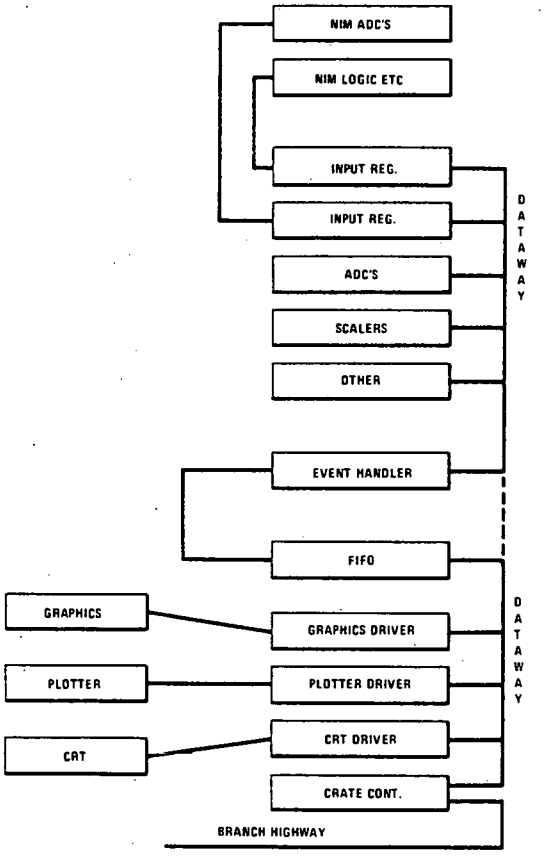

Fig. 2 Block diagram of a data acquisition station.

Other equipment to be located at each station will include a graphics terminal and a CRT terminal from which the data acquisition can be controlled. One printer/plotter will be located in each of the counting areas.

The Host Computer System

A diagram of the host system is shown in Fig. 3 approximately as it exists at this time. The two CPU's share a card reader and a line printer which are switched as required by the operating systems' spoolers. This equipment, together with the console CRT's and (1 each) auxiliary CRT's, are serviced by a multiplexed (MUX) bus in each CPU whose bandwidth is about 400 k bytes/sec. Two magnetic tape units (9 track, 800/1600 bpi, 75 ips) are also switched (manually) between the CPU's. Each CPU is equipped with one private 300 Megabyte (Mb) disk, while a third disk is accessible from both CPU's. Each CPU is connected to one CAMAC branch highway by means of two branch highway drivers; one of which is to be devoted to block DMA transfers from the acquisition station FIFO, while the other will service both DMA and programmed I/O for all other equipment on the highway. All devices serviced by the external DMA (EDMA) bus (bandwidth approximately 5 M bytes/sec) gain direct access to memory by means of a memory access multiplexor (MAM).

The system will be expanded in the near future by the addition of up to 768 k bytes of memory to each of the existing CPU's and by the procurement of a third CPU with 1024 k bytes of memory, a card reader, line printer, two tape drives and a private 300 Mb disk. The final configuration will probably involve at least three shared disks (one between each pair of CPU's). Other major items to be purchased include graphic terminals and printer/plotters.

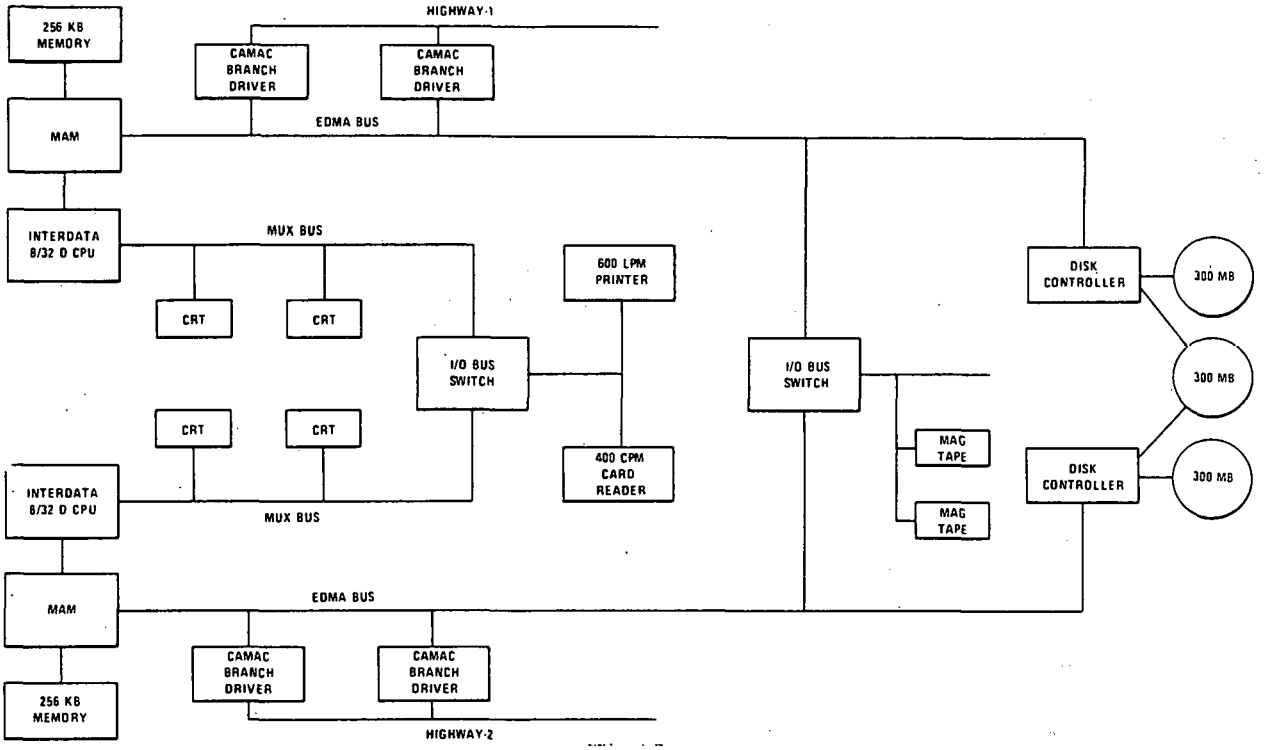

Fig. 3 Block diagram of the host computer system.

Operation of the System

Fig. 4 illustrates how the data will flow through the system for a "typical" experiment. The event handler reads the event parameters and loads the FIFO. The host CPU unloads the FIFO by block transfers over the CAMAC branch highway and records the event-by-event (list) data on disk for subsequent batch job transfer to mag tape. Any histograms (spectra) which are to be kept in memory are, of course, updated in real time. If sufficient memory is available to store all of the spectra being accumulated, the real-time operation is complete. It is often desirable to accumulate a number of one- and two-parameter spectra whose total array size exceeds the memory which is available. In such cases the histogram addresses generated by the selection (gating) routines are passed into one of the multiphase sorting routines described in the section entitled "Generation of Large Histograms Using Moving-Head Disks". These procedures involve the use of disk files into which the address list is partially sorted as well as a disk file into which the histograms are finally accumulated (labeled "RUN" HISTOGRAM FILE in Fig. 4). Spectra which are accumulated in the disk file will be subject to the same user operations (save, display, zero, etc.) as those accumulated in memory. Insofar as the user is concerned, the only apparent difference will be the time lag (of 5 minutes or more) in the updating of spectra on disk. The user may transfer spectra from either memory or the "RUN" file to a larger "SAVE" file for subsequent batch job transfer to mag tape.

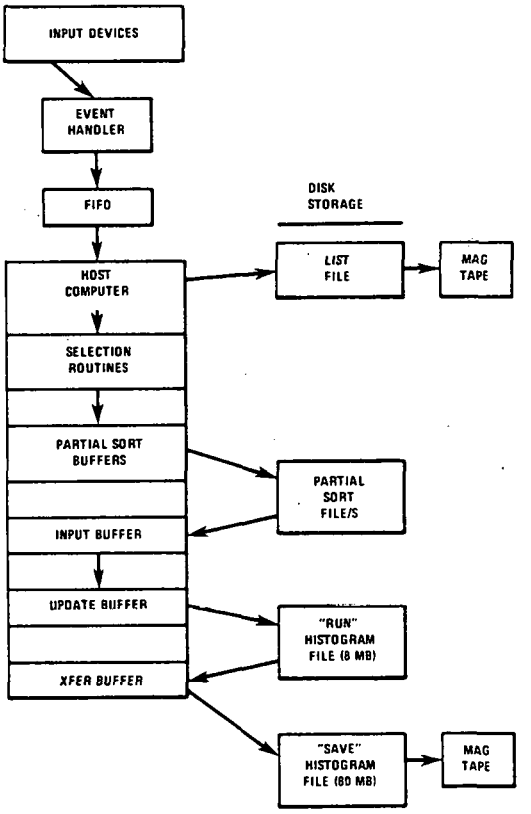

Fig. 4. Diagram illustrating data flow through the system.

Gating or Masking by Means of Mapping Arrays

The processing of multiparameter event data into histograms usually involves the setting of gates (or masks) on some of the parameters and the accumulation of histograms (counts versus parameter value) of other parameters. A typical case might involve the setting of ∿ 100 gates on each of a few (2 or 3) parameters and the setting of a few (1 to 5) gates each on several other parameters. Thus, for each event processed, one must determine which ones of the (possibly many) sets of restrictions (gates) have been satisfied. Of course, each gate set will generally involve several parameters. For those parameters on which only a few gates are set, it is practical to determine which, if any, gate is satisfied by a table search. However, if many gates are set on a particular parameter, the determination can be made much faster by setting up a mapping (or pointer) array such that each element in the array corresponds to a possible value of the parameter of interest. If the element is zero, the corresponding parameter value is to be rejected. Otherwise, the value of the mapping element gives the "gate number" into which the parameter being tested falls. The method is easily illustrated by a few FORTRAN statements.

Assume that we have NGATES limit pairs given by elements in the arrays MIN and MAX. Let the range of the parameter of interest be 0 to 4096. If we assume that NGATES is less than 256, the mapping array MAP can be a byte array. Setup of the mapping array is illustrated below.

```
      INTEGER*1 MAP(4096)
      DIMENSION MIN(256), MAX(256)
      Dϕ 10 I=1, 4096
10    MAP(I)=0
      Dϕ 20 N=1, NGATES
      ILO=MIN(N)
      IHI=MAX(N)
      Dϕ 20 I=ILO,IHI
20    MAP(I)=N
```

Let IV be the parameter value to be tested (gated), N the gate number, if any, which is satisfied, IH (range = 0 to 511) the parameter on which histograms are to be accumulated and ARRAY be a two-dimensional array which contains these histograms. Use of the mapping array is illustrated below.

```
      INTEGER*2 ARRAY(512,256)
      N=MAP(IV+1)
      IF(N·NE·0) ARRAY(IH+1,N)=ARRAY(IH+1,N)+1
```

A similar procedure can be used for two-dimensional masks, although the memory requirements may be excessive if the resolution required is very great. However, if a 128 by 128 array is sufficient to specify the required masks, one can (by the use of 4 bits per element) specify up to 15 masks with an array size of only 8192 bytes.

The methods just described do not allow overlapping gates or masks. These are not often justifiable but accommodated by the use of a single bit per mapping element to indicate that the mapping element is to be interpreted as a pointer to an auxiliary gate list. If the requirements for overlapping gates is infrequent, it is probably better to use table search for those cases rather than clutter up the mapping procedure.

The Generation of Large Histograms Using Moving-Head Disks

In many cases (e.g., in tape scanning or nearly-real-time processing) it is desirable to produce a set of histograms whose total size exceeds the memory available. This can be done "easily" if one or more large capacity moving-head disks are available. Even with a modest amount (∿ 32 k bytes of available memory, throughputs of several thousand counts/sec are possible for histogram sizes of more than one million channels. In the following paragraphs we describe three procedures (one-, two- and three-phase sorts) of increasing complexity and power. We have used the two phase sort extensively (for about 5 years) to produce histograms of around one million channels. We expect to use both the two- and three-phase procedures in the future.

One-Phase Sort

Let it be required that we produce a set of histograms (or data array) whose total size can be expressed as an array of M by N channels as shown in Fig. 5. In practice it is convenient if M and N can be given values which are expressible as powers of 2. We divide the available memory into an "update" buffer of length N and M "partial sort" buffers. Of course, one or more small arrays are required for counters, etc. but their impact on available memory is usually negligible. Input to the procedure is assumed to be an unordered list of addresses in the histogram array which are to be incremented. Each address is placed into that buffer (one of M) which corresponds to the block of

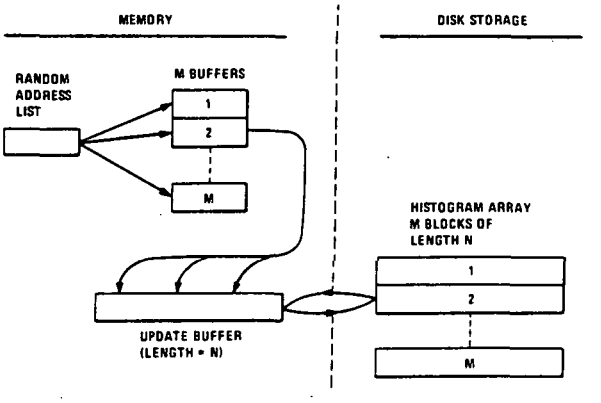

Fig. 5 Diagram illustrating a one-phase sorting procedure.

the histogram array in which incrementing is specified. When one of the M buffers becomes filled, the corresponding block of the histogram array is read into the update buffer, the specified channels are incremented and the block is written back onto the disk. After all addresses are processed we must carry out a final update for all blocks.

Two-Phase Sort

In the two-phase sort (see Fig. 6) the list of addresses to be incremented is "fanned out" to the M partial sort buffers just as in the one-phase sort. However, when one of these buffers is filled it is written into an intermediate disk file rather than used to increment the update buffer. Each of the M buffers is written onto disk as a "daisy-chain"—that is, each new record written contains a pointer that gives the location (on disk) of the previous record from that buffer. The first record from each of the M buffers contains a pointer value of zero. Of course, an array which contains the location on disk of the last record written from each of the M buffers must be kept in memory.

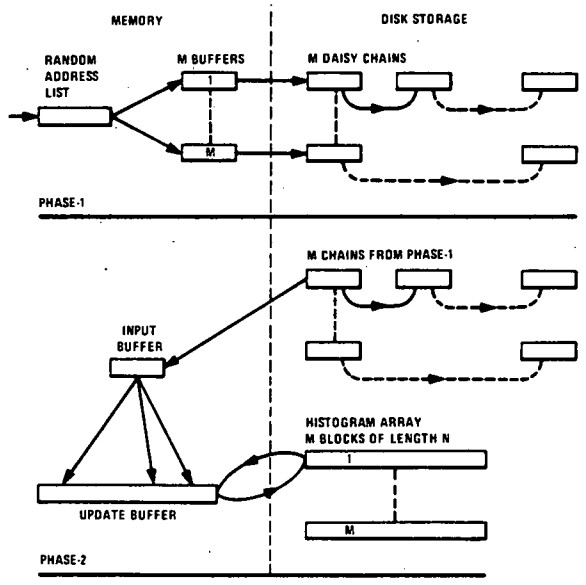

Fig. 6 Diagram illustrating a two-phase sorting procedure.

The second phase of the process proceeds as follows. When the disk file containing the M chains becomes filled (or when the program is so instructed), a block of the histogram array is read into the update buffer and all members of the corresponding daisy-chain are read (one at a time) into an input buffer and used to increment the update buffer. Subsequently, the update buffer is written back onto disk. The next block of the histogram array is read in and the process is continued until the update is completed.

Phases one and two may proceed sequentially if previously acquired data is being processed (as in tape scanning). In this case, the same memory can be used for the M partial sort and the update buffers. However, if the process is to proceed concurrently with data acquisition or tape scanning (i.e., the input address list should not be halted) it is necessary to divide the available memory between the M buffers and the update buffer.

The two-phase sort can be much faster than the one-phase for large histograms because many more counts can be added to the update buffer each time it is read and written. This fact more than overcomes the disadvantage of writing and reading the daisy-chains.

For the disks in our system the average radial seek and rotational latencies are 28 and 8.35 ms, respectively. A data density of 16384 bytes/track and a rotational speed of 3600 r/min leads to a maximum transfer rate of one megabyte/sec. The average transfer rate depends strongly on the length of the record transferred. In tests with random seeks, a 256 byte record (1 sector) was transferred at 7 k bytes/sec and a 16384 byte record at 450 k bytes/sec. In the latter case the transfer rate begins to approach that of "slow" core memory. There is clearly a speed advantage in transferring long records.

Three-Phase Sort

For a given available memory, the record length can be increased by reducing the number of partial sort buffers—i.e., we can sort on fewer bits per pass but make more passes (see Fig. 7). Addresses from the input stream are fanned out into L partial sort buffers which lead to L chains on disk (i.e., exactly as in the two-phase case). Next, each of the L chains from phase-1 are read in and fanned out into M buffers and chains. Finally, each of the M chains from phase-2 are read in and used to increment an update buffer (one block of the histogram array).

If the three phases are performed sequentially, the L, M and update buffers can use the same memory space. In a data acquisition environment the L buffers will always be present but the M and update buffers can overlap.

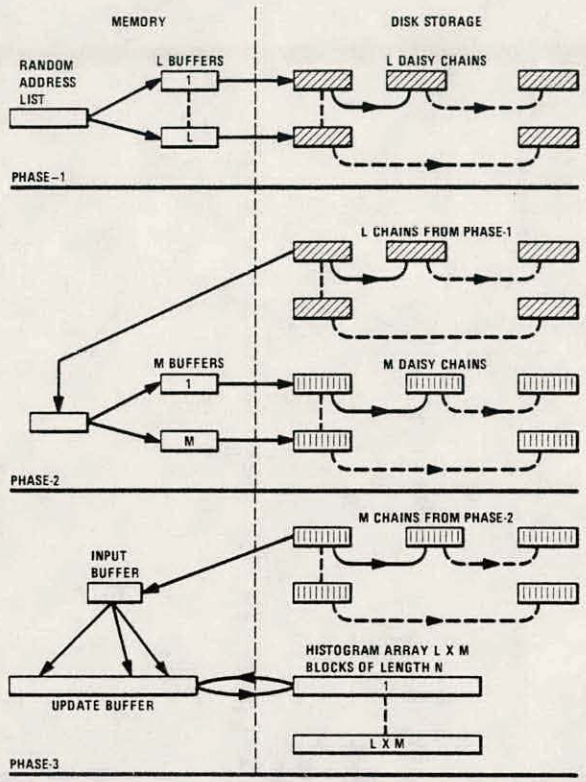

Fig. 7 Diagram illustrating a three-phase sorting procedure.

Table 1 shows a comparison between the one-, two-and three-phase procedures as a function of histogram array size and available memory. Assumptions are that the different phases are performed sequentially and that enough disk space is available for chains to permit the addition of one count per channel (average) during each update. Two bytes/channel are allowed for the histogram array and the update buffer. Two or three bytes (as required) are allowed for elements in the address list. It is further assumed that chains are written sequentially (only rotational latency) and read randomly and that the histogram file is read and written with only rotational latency. The latter assumption requires that the histogram file be on a separate disk from the chain files. If all files are on the same disk the throughput will be reduced slightly.

Entries in the table give the maximum (upper limit due to I/O) throughput in thousands of counts per sec. The CPU time required to generate the address list will not be completely additive since CPU and I/O operations can overlap to a large extent.

Table 1
Maximum Throughput for One-, Two- and Three-Phase Sorting Procedures

Histogram Size	-----Available Memory (k bytes)-----				
	32	64	128	256	512
1 MC = 2 MB	0.9	2.6	7.1	17.6	41.2
	5.9	20.6	54.5	71.3	89.5
	15.4	31.1	48.3	52.1	57.9
2 MC = 4 MB	0.4	1.3	3.5	8.8	20.6
	3.0	11.3	35.4	56.3	82.6
	11.8	25.2	43.1	49.7	56.4
4 MC = 8 MB	0.2	0.65	1.8	4.4	10.3
	1.5	5.9	20.8	39.7	71.6
	8.7	20.2	37.1	45.5	54.9
8 MC = 16 MB	0.1	0.32	0.9	2.2	5.1
	0.8	3.0	11.4	24.9	56.5
	6.5	15.5	31.3	41.9	52.2
16MC = 32 MB	0.05	0.16	0.45	1.1	2.6
	0.4	1.56	6.0	14.3	39.8
	4.6	11.8	25.4	36.2	49.8
32MC = 64 MB	0.02	0.08	0.22	0.5	1.3
	0.2	0.8	3.0	7.7	25.0
	3.4	8.7	20.3	31.9	45.6

MC and MB denote millions of channels and bytes, respectively. Entries are in units of thousands of counts/sec. First, second and third entries (top to bottom) are for one-, two- and three-phase sorts, respectively.

Concluding Remarks

Although some experiments will undoubtedly require special software, we will provide a small number (perhaps three) standard general purpose acquisition programs which will acquire new capabilities as time goes on but will very rarely lose old features. That is, new operational procedures will almost always be a superset of the old procedures. To a large extent these programs will be composed of software modules which can be used in the configuration of more specialized programs.

References

1. D. C. Hensley, proceedings of this conference.