CONF-721202--4

LIST OR SORT? - SOME EXPERIENCE WITH THE ORIC MULTIPARAMETER DATA ACQUISITION SYSTEM*

D. C. Hensley
Oak Ridge National Laboratory, P. O. Box X, Oak Ridge, Tennessee 37830

Summary

The data acquisition system at the ORIC is reviewed, and several of the data acquisition techniques used are described. The experience of two years of data acquisition with the system is discussed, and a special effort is made to assess the merits and capabilities of an associative memory technique. A current development in the programs for data acquisition is sketched, and a simple graphics technique for line printers is presented which permits plotting 3 data points per line.

Introduction

The data acquisition computer at the Oak Ridge Isochronous Cyclotron (ORIC) laboratory was in general use beginning in January 1968. Realistic use of the system as an event-by-event, data-acquisition device did not begin, however, until the autumn of 1970 with the development of a fast ADC system. The first experiment performed required 3 parameters, and the data were listed automatically onto disc and also onto the line printer in blocks of 600. The counting rate was less than one per second, and whenever a new data block was listed on the line printer, it was scanned by the experimenters for any valid events. By the end of the day, when there were some 40 valid events, our present data acquisition system was considered to be finally launched.

Since that time we have given up line printer listings, as data now are generated much too rapidly for that technique. A multichannel analyzer capability was soon introduced into the repertoire of data acquisition programs. And, as our 20,000 channel, 2-dimensional analyzer began to be inadequate and/or unreliable, ways were considered for replacing it with the ADC system and extending its capability--soon a data acquisition program was implemented based on an associative memory technique.

The hardware of the ADC system[1] and the conceptual basis for the associative memory technique[2] have already been presented. This is the first opportunity to present our general experience with our fast, computer-based, data-acquisition system. Because of the somewhat unique character of the associative memory technique which we have implemented, the merits and capabilities of this technique will be discussed in some detail. Certain new experiments for which the associative memory technique is spectacularly inadequate are now being prepared, and this has stimulated work on a program to throughput processed data into a large data matrix on a disc with a movable head. The conceptual basis for this program and the overall throughput capability to be expected of it will be discussed. Lastly, a simple graphics technique for line printers which permits plotting 3 data points per line will be described.

*Research sponsored by the U. S. Atomic Energy Commission under contract with the Union Carbide Corporation.

The ORIC Data-Acquisition System

Computer

The data acquisition and processing equipment at ORIC are sketched in Fig. 1. The features of the system which are most relevant to this paper are the Multiplexed ADC system (MUX ADC), which has direct memory access (DMA) to the mainframe through a block transfer controller (BTC), and the two mega-word discs, one entirely dedicated to data acquisition, both with DMA through the BTC. The special properties of these movable-head discs will be discussed later, but as they are movable-head discs, their random access time is long, of the order of 100 milliseconds. The SEL 840A computer has 32k words of 24 bits each, and a program-protect bit can be set for each of the 32k words.

Core Allotment

The core of the computer is generally allotted as shown in Fig. 2. FEXE is a foreground executive whose main function is to process interrupts. The ADC program is placed next to FEXE and processes any interrupts pertaining to the multiplexed ADC system. The main I/O package is placed at the top of core, and the disc I/O is the topmost part of the package. EXEC, the background executive, the program which handles all background requests, butts up against the bottom of the I/O package.

The background user has available for his use all of core from the top of the ADC program to the bottom of the I/O package--he may write on top of EXEC if he so desires. FEXE, the ADC programs, and the disc I/O are all placed under program protect and are immune to mistakes of the background user--in addition, the ADC programs have their own I/O packages, so that neither the system not the data acquisition programs need to fear the novice who is debugging his program.

Note that the background user is normally given about half of the total core space. Even though this requires that the ADC programs be lean and tidy, no significant problem with the data acquisition has arisen because of insufficient allotment of the core space to the acquisition programs.

The ADC System

The multiplexed ADC system at the ORIC has been discussed[1] previously--a schematic of its basic components is shown in Fig. 3. In particular, the system permits the easy acquiring of single or multiparameter data and has turned out to be a hardware system which is extremely flexible and which has very high resolution capability for a variety of pulse producing detectors. The front end deadtime of the system is approximately 5 μsec to store signals and about 10 μsec per parameter to convert and DMA the result. One machine cycle per parameter is stolen.

Standard Double-Buffered Front End

One very favorable aspect of the direct-memory-access (DMA) method of data acquisition is that the arrival time of the data can be derandomized. The first stage of the program schematic in Fig. 4 shows how this is accomplished. Data are brought into the buffers by DMA through the Block Transfer Controller (BTC). When the BTC senses that the allotted buffer is full, it generates an interrupt which gains the attention of the Central Processing Unit (CPU) which then immediately gives the BTC a fresh buffer to fill. The CPU processes the full buffer and then returns to the point from which it was interrupted.

Additional Boundary Conditions

I would like to point out a few boundary conditions for data acquisition at the ORIC. First, even though the computer was bought to be a data acquisition device, it is used to a great extent as a data processing center, a computational device, a tour conversation piece--that is, the background use of the computer is considered to be as valuable as the data acquisition use, though of a lower immediate priority. We have, therefore, developed the data acquiring programs so that the minimum amount of time is spent in the actual data acquiring--all remaining time is available for any background use.

In fact, most all of the data processing is done on the ORIC computer itself, even though a large central computer facility is available. This results because:

1) it would be tedious and difficult to duplicate the data processing routines on the big central computer;

2) the ORIC computer is handily located, convenient to use, and usually available;

3) both the computer and myself are friendly and informative;

4) the computer time and facilities are free.

So a primary boundary condition is that background use of the computer is not simply tolerated; it is indulged.

A second boundary condition is that the ORIC computer is a "small" computer, and certain data-acquisition techniques which may be trivial (?) elsewhere strain the capability of both computer and programmer.

The last boundary condition is perhaps the most restrictive, and that is that there is essentially less than one full time person devoted to the system. While I spend much time on the data acquisition system, I tend to think of myself as being originally a nuclear physicist, and I try to force myself to spend some time away from the computer. For those problems, both hardware and software, which are beyond my experience, my interest, or my patience, the ORIC computer system has the very capable and dedicated help of Drs. C. A. Ludemann and C. D. Goodman.

Thus, the ORIC computer system should be viewed as a smallish system which strongly supports background use, which has less than one full-time systems programmer, and which aspires to be the most efficient and versatile data-acquisition system that it can be.

List Mode--Event Recording

In the list mode, little or no attempt is made to process data on their way to the permanent storage device. One kind of gross processing that might be done would be to eliminate those events from the throughput stream which do not satisfy certain simple requirements. For example, one might eliminate zero or amplifier-saturated data, or one might check the time channel so that only "real" coincidences were stored--and the time channel itself might then not be stored.

The throughput rate on the ORIC system, that is, the rate at which data can be brought from the ADC front end through core to disc, far exceeds our capability to dump the disc data-files to mag tape. For all practical purposes, we can acquire data onto mag tape slightly faster than 4k words per second--this corresponds to writing a mag tape every 10 to 15 minutes.

This capability turns out to be largely irrelevant, however, since any experimenter who generated data tapes so rapidly would be unable to analyze the resulting mass of tapes in any reasonable time. To make one pass through a mag tape requires a little more than 15 minutes, so that many passes through many tapes would require a prohibitive amount of time. In addition, as we have only two tape stations, the scanning of more than two tapes requires constant mounting and de-mounting of tapes. Fortunately, for very complex, multiparameter experiments, it is often the case that the rate of fully acceptable events is low, few data tapes are generated, the pace (at least at the computer part of the experiment) is relaxed, and one is capable, during the experiment, of both monitoring the quality and direction of the experiment and of achieving much of the final analysis of the data. As the data acquisition rate increases, more and more time is spent in simply keeping up, and the ability to monitor all (or any) aspects of the experiment is reduced.

The practical limit on the use of the list mode is related directly to the amount of time and effort it takes to produce final processed results from the recorded data. Of course, it is usually the case that the experimenter is somewhat more tolerant of various burdens if he has conducted a careful, complex-multiparameter experiment. And some experiments are seen as being so important that there is scarcely a limit to how much data a person will try to list. But, there is a limit, and when that limit is reached, one either abandons the experiment or looks to the sort mode of data acquisition.

Singles and Multiscaled Singles

The least cumbersome form of data acquisition arises when the instantaneous data matrix fits in core. For example, the computer can be used as a standard multichannel analyzer, and a singles spectrum (or spectra) can be accumulated directly in core. Since multiparameter data can be input, a pleasing variety of singles spectra can be contemplated.

For a single multichannel spectrum, the average computer process time per event is approximately 30 μsec, but, because of the derandomizing effect of the double-buffered front end, if the counting rate is such that the average spacing is greater than 30 μsec, the actual deadtime will revert to the hardware deadtime of approximately 15 μsec.

Another in-core acquisition technique which can be readily implemented with our system is that of multiscaling, as when data are cycled through a time-succession of spectra. This technique is feasible provided:

1) The available core space will hold two spectra concurrently, so that, while data are acquired into one spectrum, the previous, still-in-core spectrum can be dumped to permanent storage and the next spectrum can be fetched into the then available core space.

2) The time to dump and fetch a spectrum is less than the time spent acquiring per spectrum.

Associative Memory

This data-acquisition technique does not assign a storage location to every member of the data matrix--only non-zero data elements are allotted storage space. A simple technique is to assign two storage locations to each non-zero data element: one for the actual value of the data element and one for the descriptor for that element.

When clustering of data elements occurs, a more compact technique is to assign one storage location for the descriptor associated with the first of a string of contiguous data elements, to assign one storage location to the string size, and to assign the next storage spaces to the data elements themselves, respectively. A more complex and compact method merges the first-descriptor and the string size into one word, so that fewer storage elements are required.

The software schematic for the data acquisition program for this associative-memory technique is shown in Fig. 4. The first section has the standard double buffer for time-derandomizing the input.

Because the associative memory, like the disc, has a poor random access capability, the process routine first orders the data and enters the associative memory routine only when one of the ordered buffers is full. (The data are now derandomized both in time of arrival and in size.) The associative memory routine runs with a "quasi-background" priority so that data may continue to be ordered into the second descriptor buffer while the associative memory routine is working on the first buffer--this helps to reduce the deadtime of the system.

When the associative memory finally fills, the memory is expanded and transferred to disc. The disc I/O is also done with a quasi-background priority, and the data acquisition may continue, even during the dumping to disc.

Though the associative memory technique of data acquisition can be extremely useful, it can be extremely limited--especially with respect to throughput capability. To get an adequate idea of the effectiveness of this technique, one must have some idea of the rate at which new channels will be required for the incoming data. If the instantaneous number of channels needed remains smaller than the number of core channels available for a time long compared to the time required to update the data matrix on disc, the associative memory should improve the possible real throughput counting rate, or it may, as compared with more straight forward and possibly faster techniques make more background time available on the computer.

The practical question with an associative memory is not: "How many channels do you want?" but rather "What is the time profile of your actual need for new channels?" Do the data cluster significantly within the data matrix, or are they smoothly distributed through the matrix? If they are smoothly distributed, you can forget about using the associative memory. If the data cluster, however, the associative memory may be effective. But this will depend on how well the data cluster and in how few channels.

I have sketched an approximation for the channel-growth profile in Fig. 5 for two experiments I have been associated with. In the upper graph, the required significant channels were determined right away--those channels accumulated many more data before enough additional and less significant channels finally filled the associative memory space. The lower graph indicates a case in which few channels were really significant, and the associative memory filled up before there was any appreciable accumulating of data.

In the 2-dimensional counter-telescope spectrum shown in Fig. 6, there is an obvious amount of clustering. Only approximately 25 percent of the reserved channels contain any data. Equally important was the fact that the data clustered well and that only a few percent of the total number of channels had many counts--this was indicated in the upper channel-growth profile. Consequently, the associative memory could handle this spectrum for a long time before it was necessary to expand the associative memory into the disc memory.

Typically, we reserved 200,000 channels for this spectrum, and we were able to sustain real throughput rates near 400 counts per second The associative memory would take from 15 to 30 minutes to fill its 7000 allotted core spaces, and it then took about 30 seconds to update the disc array.

For another experiment requiring 500,000 channels and for which there was effectively no clustering of the data--indicated by the lower channel-growth profile--the throughput rate was less than 40 per second. Since this is not a whole lot better than the random access capability of our disc, the use of the associative memory for this experiment was a disaster. In fact, by simply bypassing the associative memory, a factor of 2 in throughput rate could have been achieved.

In the case of the counter-telescope experiment, the throughput rate could have been greatly increased by decreasing the number of channels. This would have enhanced both the ordering rate and the associative memory storage rate, because a reduction in the number of channels would have emphasized the clustering and the disc array would have been updated less often.

In the case of the smooth 500,000 channel spectrum, a reduction in the number of channels would have had little effect on the ordering or on the associative memory storage time, though it would have reduced the disc updating time proportionately.

Obviously, the clustering factor is one of the most important considerations in determining whether the associative memory technique will be effective. As the number of overall channels is increased, either to enhance the experimental resolution or to expand the scope of the experiment, the throughput capability will decline. There is, therefore, a

fundamental compromise between throughput rate and data-matrix size.

A small part of the throughput limitation comes about because the computer must generate the necessary descriptors and must also reject any unsuitable data. Another part of the limitation arises from extraneous computations required simply to overcome hardware defects. However, most of the limitation arises because this kind of associative memory technique requires both an ordering of the data and a scanning and expanding of the associative memory. Just the ordering section of the program can limit the effective throughput capability to under 1000 counts per second for fully random data, and the scanning and expanding of the associative memory reduce the rate capability still further.

The amount of time to order an array of data is proportional to the square of the buffer size, and I suspect that the overall efficiency of the associative memory is proportional to the size of the ordered-data buffer, so that there probably is an optimum buffer size. There is a limit to the size of the associative memory--if it gets too large, the descriptor search time becomes too long, and it is faster to put the data onto disc and start the memory over again.

From all of this comes the conclusion that the associative memory technique is best suited to small computers and to not too greedy experimenters. If the whole associative memory program is made large, it becomes very slow, and other techniques would probably be more effective. For small computers, the associative memory technique should prove a powerful tool for data acquiring into large data arrays whenever the rate of new channel growth is kept small by clustering of the data.

The Associative Memory on Disc

There is a further option of the associative memory which might be of some interest. This is when the associative memory is kept on disc and not in core. Because a large associative memory is a most inefficient beast, I mention this only as a possible means of handling a data matrix whose size exceeded even the disc capacity.

Direct Updating onto a Moyable Head Disc

I would like to propose a technique for the efficient updating of a large data matrix on a disc with a movable head. Our disc has 10 heads, and the 10 heads move as a unit and may be positioned at any of 100 tracks. There are 1024 numbers per head per track (divided into 16 sectors of 64 words each). To update a given channel, one must first read the respective sector, update the channel, and rewrite the sector (on the next disc revolution). Because our disc is divided, essentially, into strips of 1024, it seems reasonable to base the updating technique upon this feature. In order to exploit the full capability of the disc storage, the data must be sifted in some way so that the maximum amount of data that can be added to the data in a 1024-strip on disc will be. It is far too slow to sort the data as they come in, and it is also too slow to scan the entire data buffer each time a 1024-strip is updated. Since the maximum number of strips is 1000, there is no reason why an address chaining technique should not be employed.

Suppose that the data matrix were N x 1024 in size, where N, for our disc, could be as large as 1000. Reserve N spaces for the initial address of of the descriptor address chain, and the remainder of available core can be given over to the data buffer. Then the descriptors for the incoming data can be chained in the following manner.

1) Determine which 1024-strip the descriptor belongs in.

2) Find the address of the previous descriptor for that strip in the 1024-strip address table.

3) Merge the least significant 9 bits (0 to 1024) of the descriptor with the address found in step 2.

4) Store this word in the first available space in the data buffer.

5) The address of the word in the data buffer is then stored back in the 1024-strip address table--creating an address chain linking all data associated with that 1024-strip.

Fig. 7 shows a schematic of a chain for the second 1024-strip.

When the disc updating routine has read in a particular 1024-strip, it finds the relevant data by going to the chain-address table and following the chain through to its end. (The last address might be set to zero to signal the end of the chain.) Whenever the data buffer contains sufficient data (more than half full, for example), initiate the disc updating routine.

An important feature of the program is that the spaces in the data buffer are released for more storage just as soon as the data are added to the 1024-strip on disc. Rather than being a static buffer (one that is filled and then emptied), the data buffer becomes a dynamic buffer, as places at random within the buffer are being either filled or emptied all of the time. The available spaces in the data buffer should, therefore, also be chained, since they are scattered at random. The descriptor storing routine could work on one end of the chain, and the disc updating routine could be constantly linking the freshly available spaces into the chain.

To get an idea of the possible maximum throughput rate, let us assume that N = 1000; the data matrix contains 1,024,000 channels and fills the entire disc. It takes about 52 seconds to read and write the entire disc, and, if we assume that the big buffer is allotted 10,000 words, a throughput rate of 200 per second seems feasible. And the throughput rate should increase directly as the size of the data matrix decreases.

This scheme is feasible only because there is so little overhead involved in reading and writing the disc--all disc I/O is done through the DMA channel. I suspect that the in-core processing time will be less than the disc I/O time, so that the program should be able to go as fast as the disc will allow.

There is one aspect of this dynamic data buffer which I think has great charm. The 1024-strip chain in core linking the least data is that one which has just been transferred to disc storage. The chain linking the next least amount of data is the one for the strip just previously up dated on the disc. By extension, the chain linking the most data is the one for the strip currently being transferred to disc. That particular chain has been steadily growing while

the other chains were being dumped, so that the number of data points linked in that chain is about a factor of 2 greater than would have been if all chains were added to for an equal amount of time. Thus, one may actually realize a factor of 2 increase in the throughput rate over that for a "static" buffer under the condition that the data input rate is greater than the rate for dumping to disc.

The 3-Plot

In keeping with the spirit of this paper which centers on the problem of maximizing the data acquisition capability for a small computer complex, I would like to present a simple, paper-saving, graphics technique for line printers which permits plotting 3 data points per line. This is totally an aside but is done in deference to my predecessors who in years past have presented fantastic examples of data-display graphics.

If your line printer has the capability of crowding the lines so that there is little or no space between lines (our type face is 1/8 inch, and the normal spacing of 6 lines per inch can be adjusted to 8 lines per inch), then you can very effectively plot 3 data points per line.

The symbols appostrophe ('), minus (-), and period (.) lie at the top, middle, and the bottom of the line, respectively. For the first data point, the symbol (') is printed in the appropriate column on the page, (-) for the second data point, and (.) for the third data point. If the (') and (.) symbols fall in the same column, these may be printed quickly by using the exclamation point (!). If some other combination of the 3 data symbols fall in the same column in a given line, then the (1H+) format technique of Fortran must be used. An example of this technique is compared to a standard line printer plot in Fig. 8.

The amount of paper used in long plots is reduced by more than a factor of 2, and the time required to plot a long spectrum is also reduced by more than a factor of 2. In addition, data acquired with a large dispersion often look considerably better when plotted with this technique.

Conclusions

Now that I have discussed both the associative memory technique and the technique for direct updating of the disc, I would like to say a word in behalf of the associative memory technique. It seems plausible that direct updating of the disc has a higher throughput capability than does the associative memory technique for some kinds of experiments, especially those with little clustering of the data and those which require a large data matrix. Even when the clustering is very favorable, a technique for direct updating of the disc may be capable of going faster. The use of the associative memory technique, however, whenever it is reasonably competitive, will surely save immeasurable wear and tear on the disc hardware.

But more important, when one is acquiring data at rates reasonably below the maximum possible, the associative memory technique will release far more CPU time to the background user (particularly the experimenter). And I think that this should be taken into account, especially in cases where direct updating of the disc is potentially faster, but where the expected data rates are low.

References

1. C. D. Goodman, C. A. Ludemann, D. C. Hensley, R. Kurz and E. W. Anderson, IEEE Trans. Nucl. Sci. NS-18 (1), 323 (1971).

2. D. C. Hensley and C. D. Goodman, IEEE Trans. Nucl. Sci. NS-18 (1), 312 (1971).

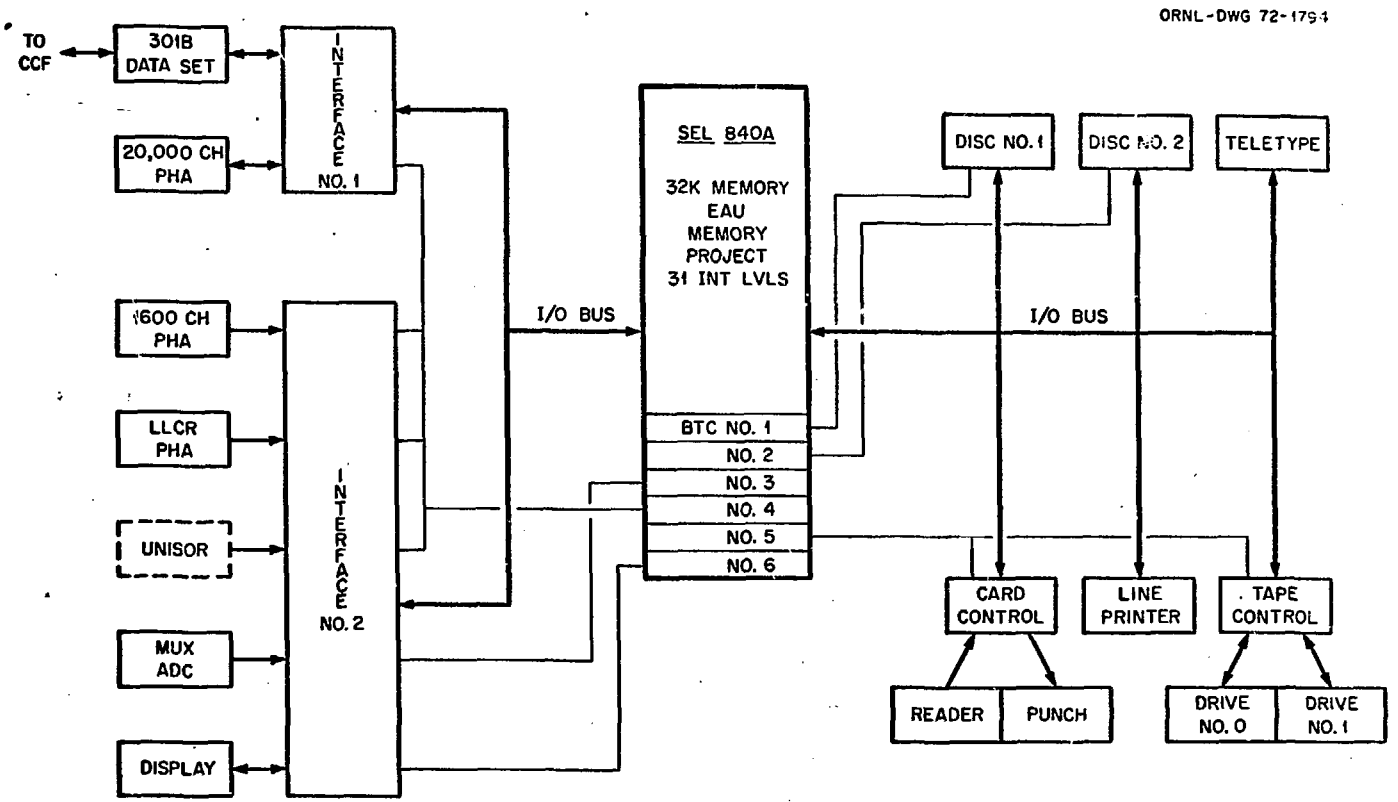

Fig. 1. Schematic of the ORIC Computer and Data Acquisition systems

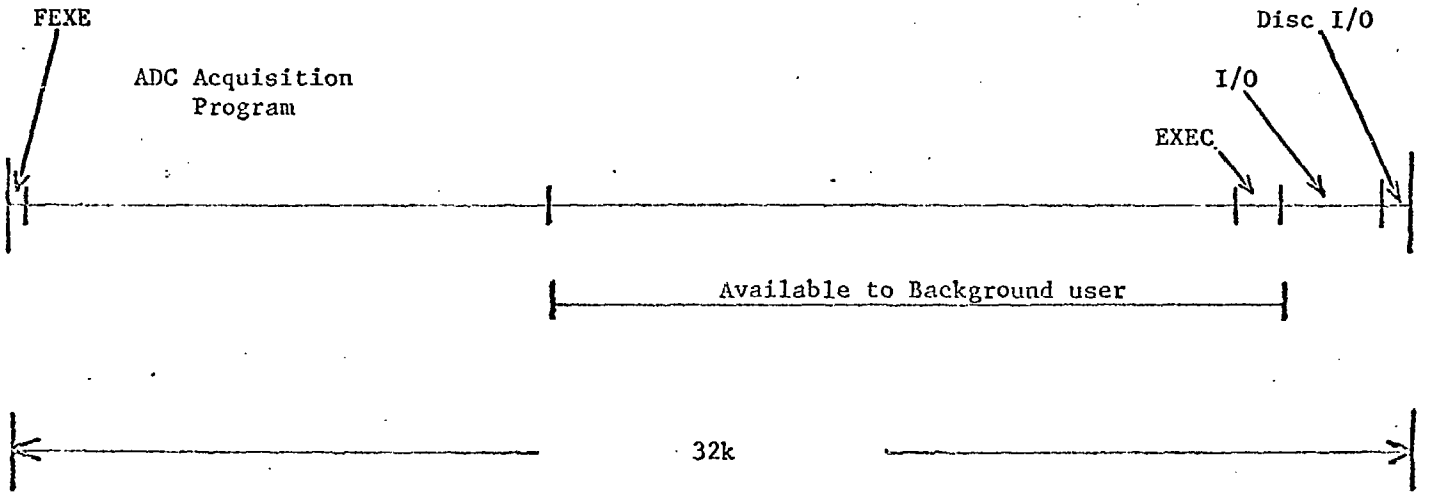

Fig. 2. Normal Core Allocation

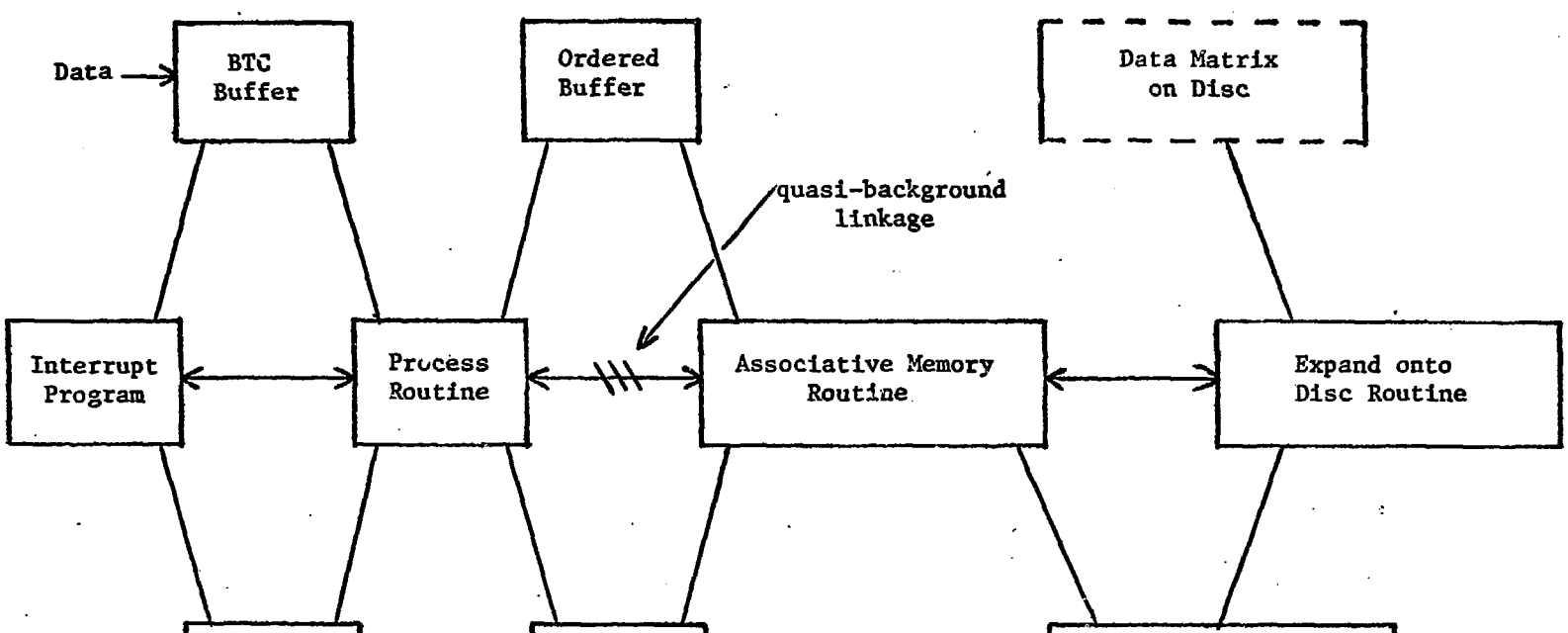

Fig. 3. Schematic of Double-buffered Front End and Associative Memory Program

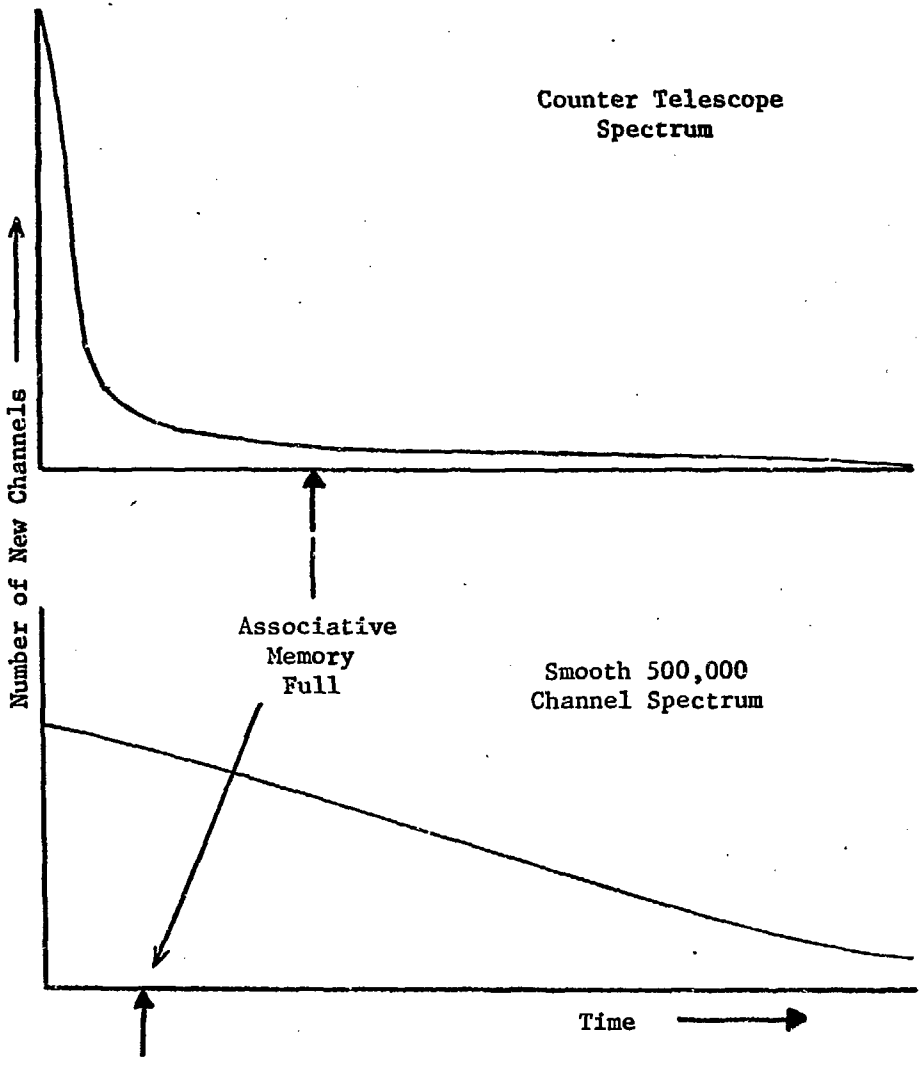

Fig. 4. Channel Growth Profiles

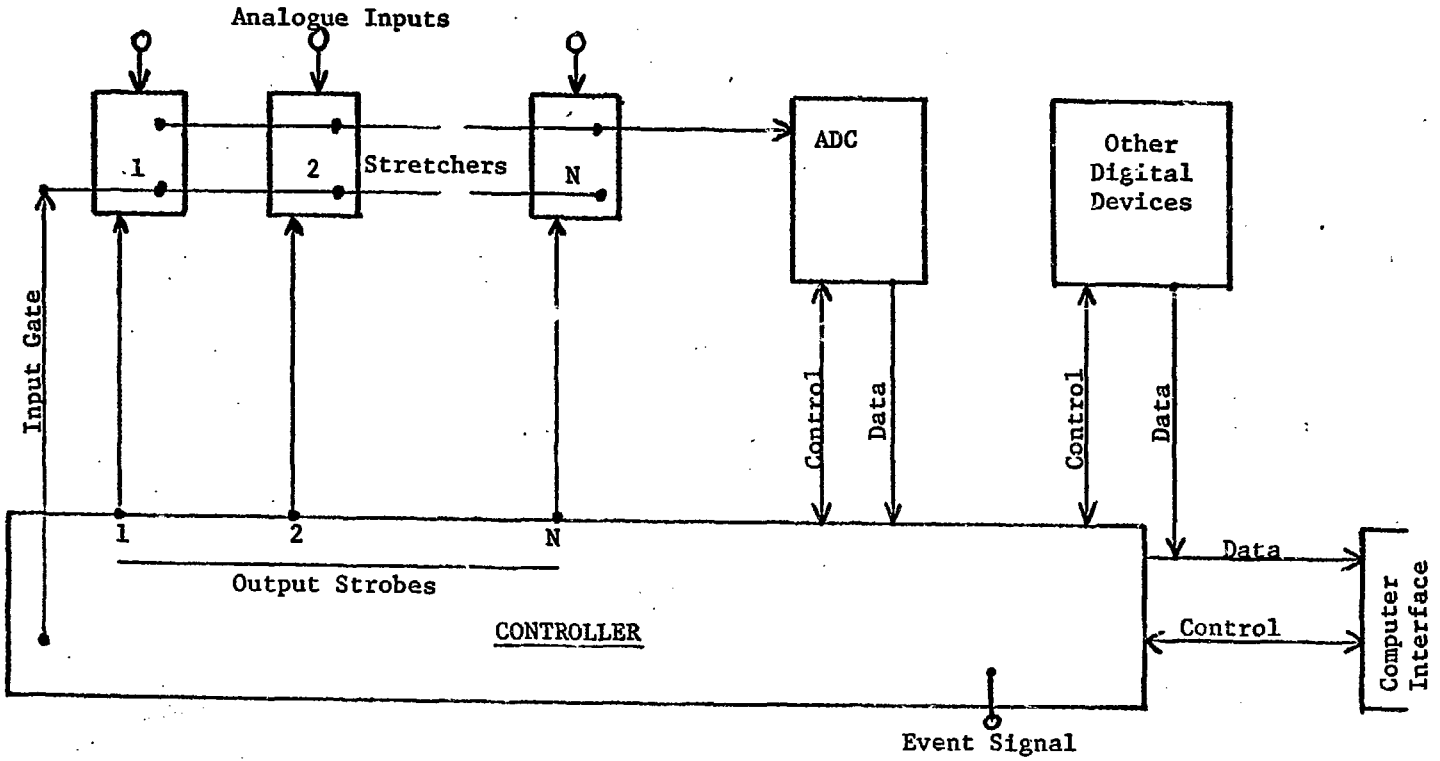

Fig. 5. ADC System Schematic

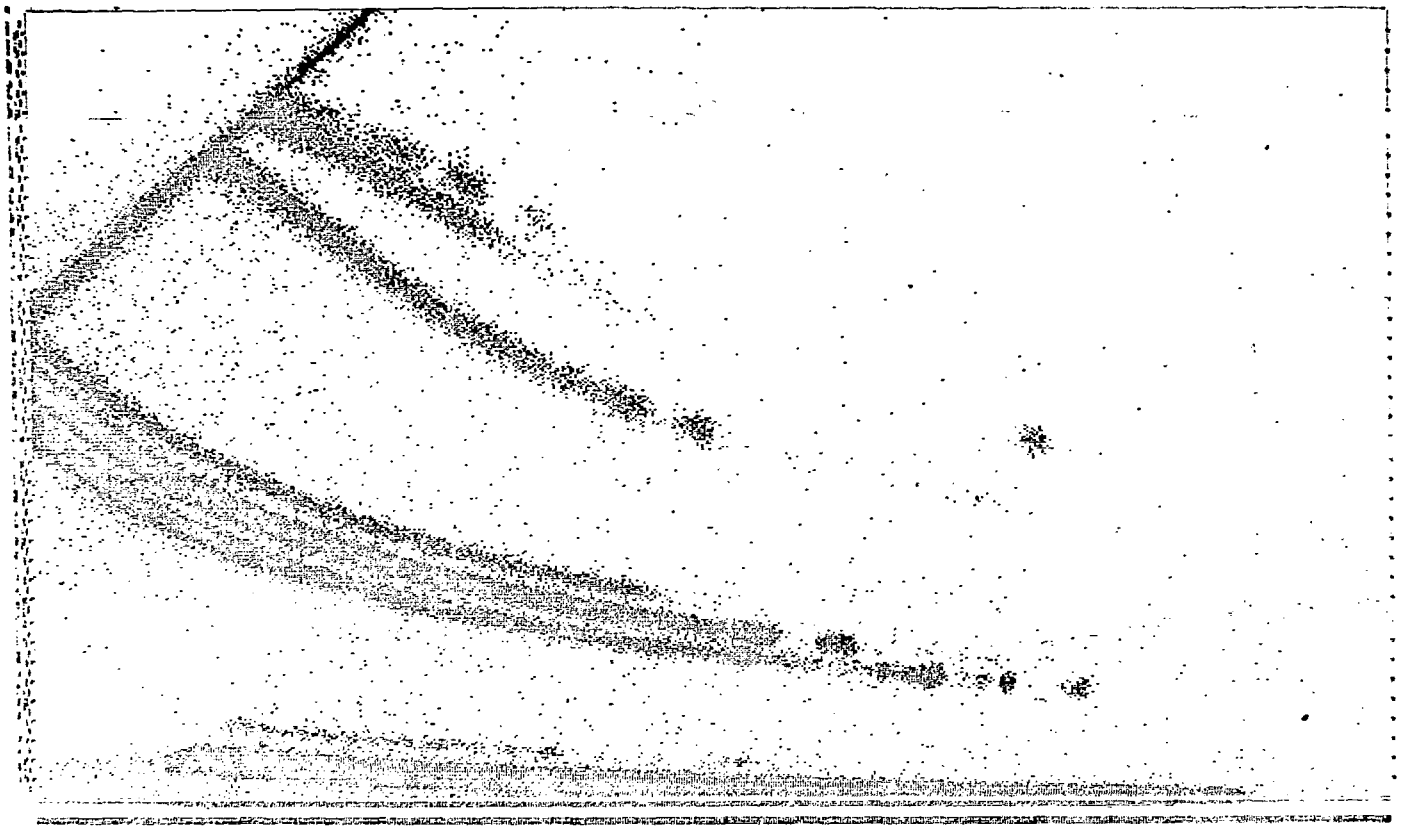

Fig. 6. Charged particle spectrum taken with a counter telescope

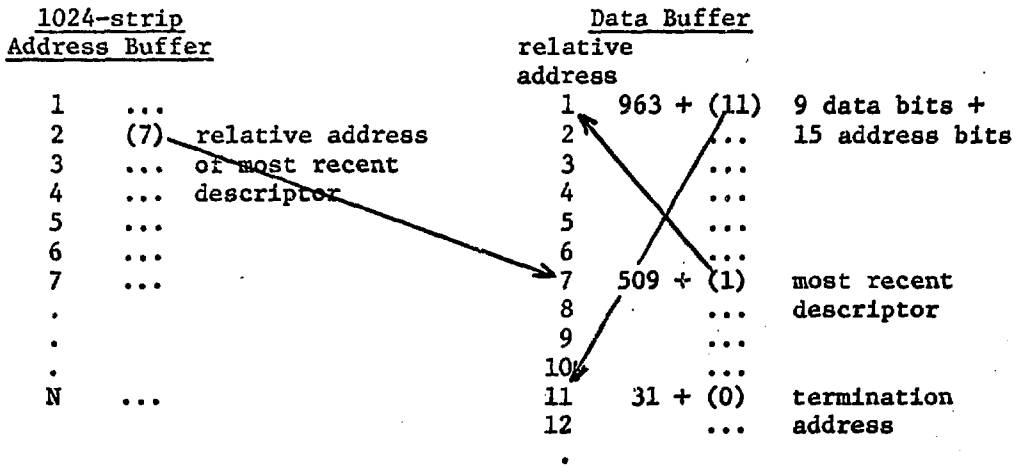

Fig. 7. Example of Data Chaining for 2nd 1024-strip

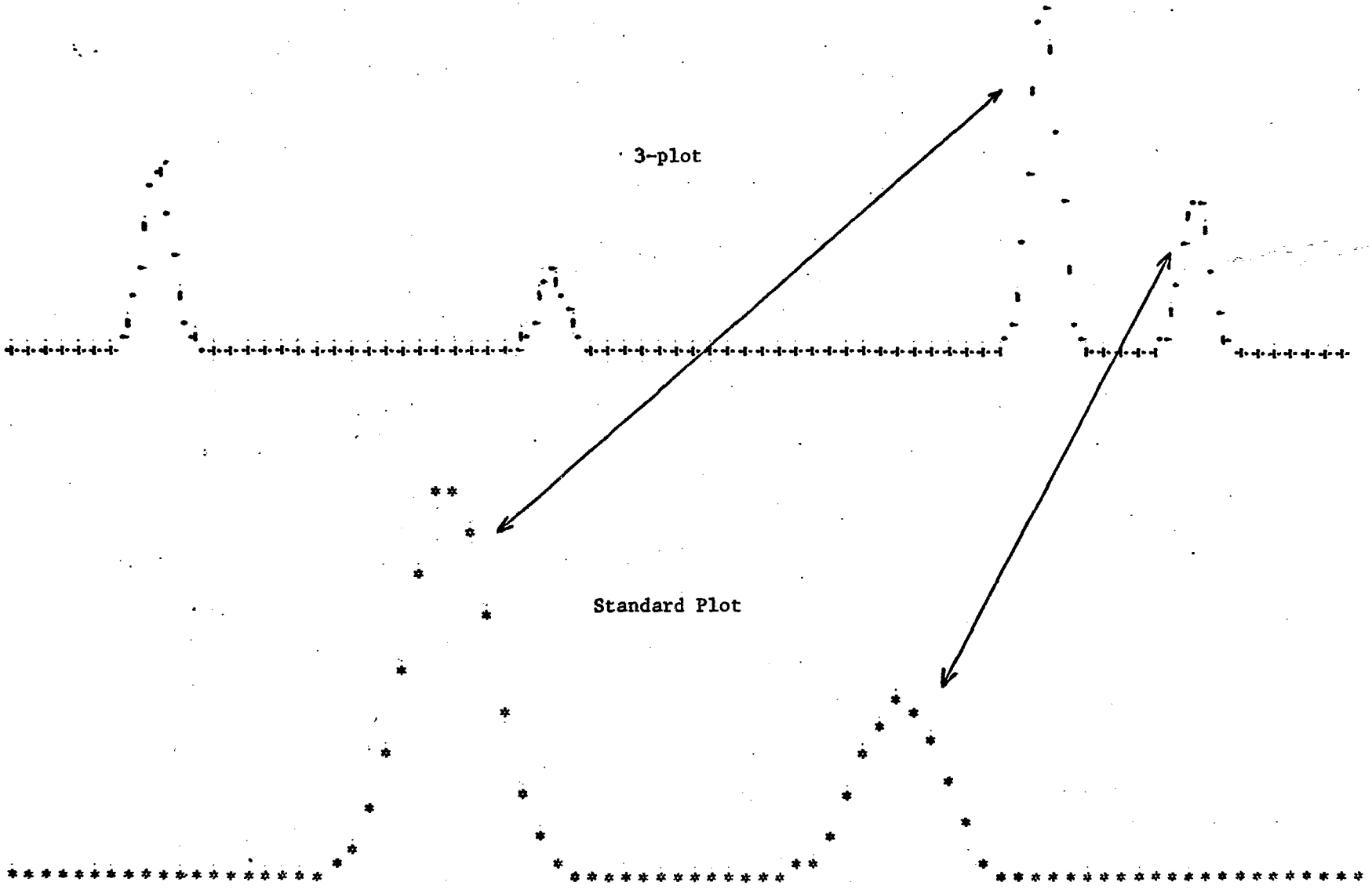

Fig. 8. Comparison of Line Printer Graphics: 3-plot versus Standard Plot