DATA DRIVEN PROCESSING

H. Cunitz, Y. Hsiung, B. Knapp, W. Sippach
Columbia University, New York, NY 10027

ABSTRACT

Herein is described a very high speed processing method. It is based on the data driven principle, and depends on constructing algorithms from hardware operators that are generally interconnectable. A set of modules and their application to Fermilab E 605 is discussed.

INTRODUCTION

In order to deal with the very high information rates provided by modern detectors, we are forced to introduce some sequential processing in the data stream. To achieve high bandwidth, we must consider the limitations of each part of the system. If we are willing to treat the system as a pipeline structure and fully buffer the data stream, then the time for encoding the data into numerical form can overlap the processing time. By the same token, we can distribute the processing in the pipeline so that only the average processing time of any section limits the rate in the data stream. In addition, raw data can be passed thru the structure for recovery to permanent storage. This view leads to the concept of a data driven structure, where the natural segmentation of the detector is imposed on the data formating and buffering arrangement, and the processing algorithm appears in a hardware pipeline.

DATA DRIVEN CONCEPT[1,2]

The traditional stored program computer treats computation as an ordered sequence of operations to be performed on a set of data. A data driven processor is based on the principle that any operation can proceed when its operands are available and the destinations of the results are able to receive them.

The step to a hardware processor is quite simple. We must build a set of useful operators that are generally interconnectable, and then put together computation structures that match the natural structure of the problem. This will result in genuine concurrency of computation, and an enormous increase in speed.

This structure requires no central control, except for maintenance. Data words and blocks are aligned by the operators, so that quite complex systems involving nested loops can be constructed that are completely determinate and free of conflicts.

DATA TRANSFER PRINCIPLE

The basis of this scheme is the data transfer principle that allows generalized interconnectability.

We define a cable with a 16 bit data value field, and an 8 bit control field. The control field contains a bit called _valid_ that defines a non-empty data word for that clock cycle, a bit called _complete_ that defines a block boundary, and 4 bits called _name_ that identify the data subset to which the valid belongs. A bit called _hold_ is produced by any destination module on the cable that is unable to accept the data transfer, and a bit called _block reset_ allows data within a block boundary to be destroyed by a downstream device. Data transfers between modules, and internal to modules, are register to register, synchronous to a central clock. The hold presents a special problem since it propagates backwards in the sequence. The hold is de-skewed with respect to the clock at the output of each module, where a normally transparent latch is provided to prevent loss of data because the output register sees the hold one cycle too late. A register that contains no valid data can be loaded regardless of a downstream hold, effectively blocking the hold for that cycle. The data flow is optimally controlled in complicated structures, where the process of data alignment generates both empty words forwards and holds backwards, that annihilate on contact.

More than one module is allowed to receive the same cable, and branches are constructed by pre-programming each module to accept a subset of the name space.

PROCESSING SYSTEM

A system includes the readout, the data buffering, the data transfer buses, the processor(s), the host, tape units, etc.

Figure 1 shows a fully pipelined system. Here each readout segment drives a ring buffer so that the detector can transfer output at a rate limited only by the rate data can be processed and removed from the buffers. The buffers are truly data driven and require no communication with the readout except with their holds, and no part of the detector readout communicates with any other part, except for the readout busy signals that must merge at the fast trigger source.

The buffers individually feed various inputs to the processor structure, and have internal counters for keeping track of the event blocks. In general, the buffers will contain new events awaiting processing, and old events still in the processing pipeline. The output of the processor is a decision to skip or read the earliest event stored in the ring buffers. All ring buffers attach to a read bus and are expected to accept this skip/read command in unison. Hold, on this read bus, allows any buffer, momentarily unable to execute this command, to prevent other modules from prematurely acting on the command.

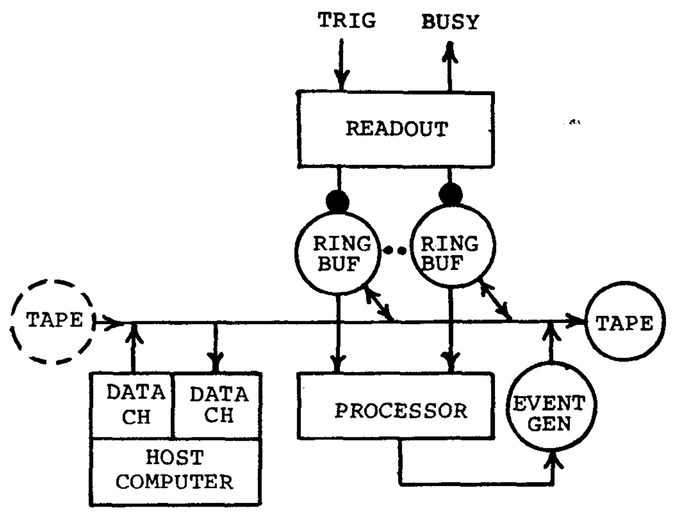


FIG. 1 PROCESSOR SYSTEM

A property of this system is that it can be data driven either from the readout, which is a parallel structure, or from the data bus, where data can be written into ring buffers from the host, or from tape. The same data tape that is written from the data bus can be rewound, written directly into the ring buffers, and processed to produce a reduced tape. The holds, and alignment of block boundaries between events guarantees determinancy.

The processor and readout have a control bus that attaches to every module, and is used to maintain the system and to load fixed data into the modules. All registers, counters, and memory locations are accessible via this bus. The internal registers are constructed from shift register IC's, so that these off line operations can be bit serial on this bus, which requires only one data and one response bit (in addition to address and control bits). For system testing, blocks of code are sent from the host data channel into the system, and output code from the system is independently received by an input data channel, which interrupts the host either when it is filled or when an interrupt word appears in the output code. This way the host requires no special knowledge of the system state for transfer of data in and out.

The readout bus is based on the data transfer principles described in the previous section, except that it has four more control bits in the name (address) space; a tag control bit that allows the name to be associated with the data source or to represent the address of a destination, and a tag control bit that defines use of the data field as a value or as command information. These tags result from merging control and data transfers onto the same physical bus. Because the bus is synchronous, and all words are fully specified by the name and tag fields, it is possible to have interleaved, autonomous communication on the bus. A small number of additional module types allow asynchronous communication to external devices, for example the host, tape units, etc.

PROCESSOR MODULES

So far we have constructed a set of 14 modules found to be useful in track following and data organization problems. The modules are based on ECL 10K logic and operate at a 40 Mhz clock rate. The modules can be organized into four groups; lists, sequences, functions, relations.

The list modules are differentiated by the way the data is organized for access.

List/Index-Data is written in sequence at the write port, and assigned an index or word count. This index defines the data storage location, so that data can be retrieved from the read port by index value. The write complete word is held at the write port until the read port complete is received, at which point the words are merged to produce the output complete word, and reset the index.

List/Counter-Data at the write port passes thru the output port, and those data elements that match the pre-assigned name are written into memory and assigned an index (i.e. word count) that defines the storage location in memory. In order to retrieve data via the input port, a read index is generated by counting data elements at this port and reading from memory those elements that match a pre-assigned name. This subset of the data that entered the write port corresponds to a relation on the original set. It is assumed here that all the elements that pass thru the list counter via the write port are tested by a processing relation and passed back to the read port with a name assigned by the test that indicates which elements pass the test and therefore belong to the relation. Any number of list counters can be connected in series, where there are multiple nested loops in the relation. The output complete is produced when all elements sent from the module have been counted at the read port. The index counters are reset when the output complete is formed.

Buffer-This is a FIFO that allows data alignment between different parts of the processing structure without causing holds in the connecting path. Data can be written and read during the same 25 nanosecond cycle. Data is available at the output port whenever the buffer is not empty and there is no hold on the output. Holds cannot propagate to the input port if the average output rate is equal to or greater than the input rate, and the dynamic space of 128 words is not filled.

Map-This is a storage device that allows associative data retrieval. A memory cell is assigned for each possible write data value, with all cells initially set to zero. Data is retrieved via the read port by value in the form of nine contiguous cells around the integer part of the input value. An optional form of the map allows 16 cell access around the integer read value, but requires two read cycles. The truncated part of the read value is passed to the output and concatenated with the cell data. If data is written as an ordered sequence, the read hold is used to prevent reads until the last write value exceeds the value at the read port. For unordered write data, a read hold must be present until the write complete is transferred. When both write and read completes have been received, the output complete is sent, and the map is erased from an internal list containing the last block of cell numbers loaded into the map.

These modules generate sequences from the data;

Binary Index Generator-This module generates the cartesian cross product of two data sets in index pair form by counting the data elements at each of two input ports, and generating all possible index pairs at the output port. Outputs are produced as soon as two or more inputs are counted, under control of two read counters and two pointer registers. No input holds are produced except by the input completes which are held until the full array has been generated, at which point the output complete is sent and the index counters are initialized.

Unary Index Generator-This module generates all the unique index pairs of the product of the set on itself. The diagonal elements of the set, that is the elements themselves, are generated with a different name to distinguish them. No holds are produced on the input port, except by the input complete which passes to the output when all index pairs have been produced.

Page Generator-This module is used to copy the data elements a pre-selected number of times. Only data with a pre-specified name will generate this copy sequence, all other data elements pass directly thru in one cycle.

The following modules are available for generating functions;

Arithmetic Operator-This operator performs any of the standard binary arithmetic and logical operations on the output data (add, subtract, and, or, exclusive or, etc.) provided by the ECL 10181 ALU. A plug-in patch allows the 20 bit name and data space to be connected in a general way to the two 16 bit inputs and to the control space of the ALU. Alignment of data in the input registers causes data transfer to the output and new data to be entered at the input registers, under control of the holds. Alignment of completes at the input ports produces a complete at the output port.

Normalizer-The normalizer is used to give a linear function value ax+b of its input value x. Two internal memories (8 address bits each) can be preloaded with 16 bit function values. A plug-in patch allows any 16 of the bits of input data and name to be connected to the two 8 bit address tables, and the outputs of the tables are added with 16 bit precision. In order to normalize 16 bit input words, the 8 high order bits are patched to the high order table, and the 8 low order bits to the low order table. For smaller size values, name bits can be connected in common to the two tables, resulting in sets or normalizations. Any function of $F_n(X_1) + G_n(X_2)$ can be produced, where X_1 and X_2 are separate data fields of the input words. The complete passes thru the normalizer without producing holds.

Binary Table-The table is used to give a general function value of $F(X_1,X_2)$ of its two input values. A plug-in patch for each of the inputs allows any part of the value, name space of the two input words to be patched into the 8 bit address space to produce a 16 bit function value at the output. Alignment of data in the input registers causes data transfer to the output under control of the holds. The ~~table~~ can also be a test if appropriate output function bits are patched into the output name space.

Unary Table-This is identical to the binary table, except that only one data input is provided.

The following modules are available for generating relations involving greater than, less than tests. A relation results in a name being assigned to the output data depending on the test result.

Ordered Merge-This binary input module merges two ordered data sets into a new ordered set. The data elements at each input port are compared over a selectable part of their data fields. The larger (or smaller) word is passed to the output, and the other input is held until it is larger (smaller) than the word at the other input port. When the inputs are equal, either the value is passed to the output with a special name, or optionally both values are passed in sequence with their respective names. The input completes must be aligned before an output complete is produced.

Associate-This is a unary input module where adjacent words in a sequence are subtracted and their difference compared to a pre-set number, for greater than or less than. The result of the compare assigns a name to the words according to this association. The complete word passes thru without producing a hold (unless there is a hold on the output port).

Cut-This unary device compares the input value with two preloaded 16 bit numbers and names the data word according to whether it is within the cut, or above or below the cut. Only holds on the output cable affect the data flow. All words, including the complete, pass thru without producing hold.

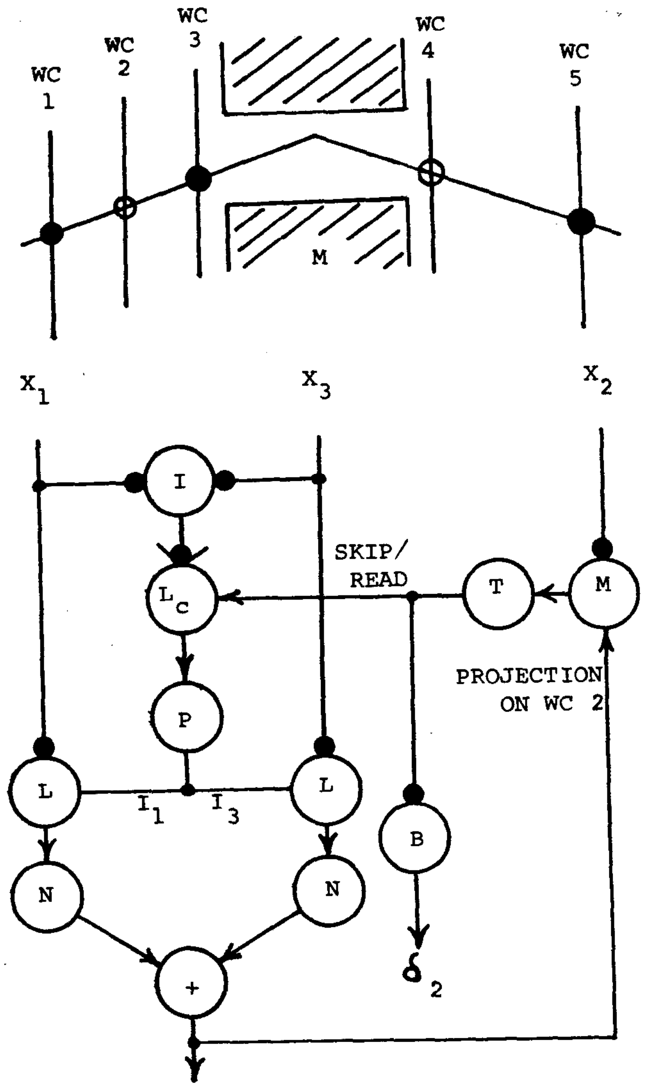
ALGORITHMS

Because the system is modular, the processing algorithm can be flexibly adapted to the experiment. Each module and the complete algorithm is emulated in Fortran so that the algorithm can be pre-tested both with Monte Carlo data, and with data tapes produced by the readout. These data tapes can be directly off loaded into the ring buffers and processed, with no modification to the system.

An example of a simple binary loop structure for track finding is shown in Figure 2. This structure is designed to predict the location of hits in wire chamber 2, all possible pairs from chambers 1 and 3, and then generate information needed for testing the trajectory past the magnet by using hits in chamber 5 to test chamber 4.

The symbols are defined as follows; a module is indicated by a circle containing an identifying symbol, and lines connecting the modules represent the interconnecting cables. Inputs are of two types, a read input, indicated by an arrow, forms part of the active processing path thru the output, while a write input, indicated by a solid blob, directly affects only internal storage within the module.

The binary index generator (I) counts valid transmissions and generates all index pairs over the accumulating counts. The lists (L) sequentially store incoming data which is later retrieved by index. The normalizers (N) generate linear mappings of their inputs which are then added to define a projection of chamber 2. Data written into the map from chamber 2 is retrieved in the form of the 9 contiguous cells around the prediction, and concatenated with the non-integral part of the prediction. A table (T) transforms this data into a weighted value and a name is assigned to the data. This causes the index pair associated with the test to be retrieved from the list counter



LINEAR COMBINATIONS OF X_1, X_3 FOR DOWNSTREAM CALCULATIONS

FIG. 2 TYPICAL BINARY LOOP STRUCTURE

and reinserted into the calculation if the test was passed. The new name is used to look up another set of normalizations for calculations needed in the downstream structure. The copy device (P) allows any number of calculations to be generated in sequence for the same index pair.

This type of structure indicates how quite complicated loop structures can often be decomposed into binary structures. In this example, instead of $N_1XN_2XN_3$ cycles (where N is the number of hits), we have only N_1XN_3 cycles because of the use of the map. Whenever we re-use the structure, as in this example, we create a loop. This is worthwhile here, since the test only causes infrequent re-use, but considerable savings in hardware. Because the data elements are distinguished by name, they are allowed to coexist in the sequence without causing conflicts. The computation proceeds at the highest possible speed since the operations take place concurrently at the 40 Mhz clock rate, moderated only by holds and data alignment in the modules. Any amount of parallelism can be added to the computation part of the structure by expanding the number of normalizers and other arithmetic devices, or alternatively the structure can be re-used by adding more nested loops where we use list counters to close the loops. The use of a buffer (B) at the table output allows data from the table to be automatically aligned with the downstream data without producing holds.

We can duplicate this structure for each of 3 or 4 wire plane views, and fold them onto each other to reduce hardware, or unfold them to gain speed. If we consider the more complete problem of finding tracks in three views, the nested loop structure decomposes into loops for lines in front of the magnet, those for tracking to the back, and those for matching tracks found in each view. This decomposition leads to a high degree of concurrency, where the delay thru the structure is irrelevant since the new events keep the pipeline optimally filled.

TRIGGER PROCESSOR FOR FERMILAB E605

We have built a small trigger processor for E605. The detector configuration is shown in Figure 3. The target is in the field of a focusing magnet which focuses high mass pairs around a beam dump into the detector. There are two UVY MWPC chambers in front of an analyzing magnet, and two drift chamber stations behind it, with staggered pair UVY planes. In addition to this tracking system, there is an imaging Cerenkov counter between the drift chambers, and there are electron, hadron and muon detectors in the back.

Three planes of x,y hodoscope counters are used to define a fast trigger for gating the readout systems. The data subset required for the trigger processor transfers thru the ring buffers to the processor inputs.

The preliminary form of the processor selects tracks that are consistent with the target and constrains P_y. The system modularity will allow development of the trigger as running experience accumulates.

Wire hits from the staggered pair drift chambers are merged with ordered merge modules. Adjacent wire pairs and singles are encoded by associator modules which assign a low order bit to the wire number corresponding to one-half wire space.

Wire hits from the MWPC chambers are written into separate maps. A binary loop structure, similar to the one described in the previous section, forms lists of the associated signals from the two drift chambers and generates all possible line projections for the MWPC plane maps. Both projections are calculated simultaneously to gain speed.

A binary table transforms the output road data from the maps into a cut. If the test is met, a new name causes the list counter module to retrieve the index pair for the track, and the linear combinations of Y_3, Y_4, for calculating the P_z and P_y momentum components are simultaneously accessed in the normalizers (the test name is used to address the appropriate pages of the normalizer memory). Log tables are used to generate logs of these two quantities which are subtracted to give the log P_y. A table forms a cut on P_y.

The projections on the calorimeter, muon detector, and counter hodoscope are also

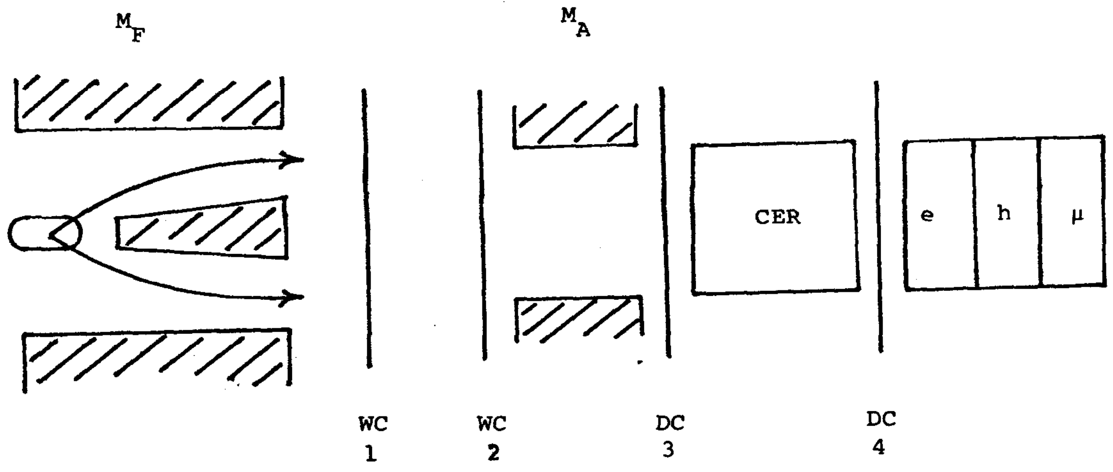


FIG. 3 E605 DETECTOR

copied out during this pass of the loop. The various cuts are buffered, to eliminate holds, and concatenated to form a parametrization for each track candidate. A unary index generator module forms all track pairs and singles, and a trigger cut is produced in a binary table. The trigger output word assigns trigger identification to 12 unique bits, and a frequency which is matched against an event counter in the event generator module. A match sets a flag that causes the ring buffers to transfer the last event in the ring to the readout. If the flag is not set when the event boundary passes out of the processor, the event generator causes the last event in the ring to be skipped. The identification bits are accumulated in flip-flops and sent with the event count whenever the ring buffers are read. Track parameters from the processor are also sent, and all data is transferred to a mega-byte memory[3], at a 10 Mhz word rate for PDP-11 processing during the beam-off time. The system deadtime due to the trigger processing is 1 to 2 μs.

NEW MODULES

For economy, we wish to vary the amount of hardware for a particular computation to match the desired speed of computation. For the modules described so far, this can be accomplished in an obvious but limited manner by varying the degree to which modules are repeated. For less frequently performed calculations we use another family of modules in which data transfer is serial rather than parallel, i.e., rather than transmitting one word each clock period with a 24 bit cable, we transmit n bit words on one bit of cable in n clock periods. Communication and computation thus require less hardware to operate more slowly.

Another feature of previously outlined computations was the presence of frequent decisions based on simple computations, which then alter the sequence of subsequent operations. Large computations with fixed sets of operations permit additional optimization of the hardware, because more operations may be performed in parallel.

An interesting arithmetic structure with serial communication, but performing normally distinct computations in parallel, is the Σ module used to provide linear combinations of several variables:

$$Y_n = \Sigma A_{ni} X_i + A_{no}$$

The variables X_i are transmitted simultaneously but bit serially on a narrow bus cable which may have several Σ modules. Each transmission provides an 8 bit address for a 256 word table of sums of all constants A_{ni} for which the corresponding bit of the address is one. The table entry is added to a shifting accumulator. If the 8 bit address is inadequate, the computation must be distributed over more than one Σ module and the results added. Bit serial addition is sufficiently simple that Σ modules can simply be cascaded with increased propagation delay, or another simple module can add up to eight pairs of numbers in parallel. For large computations, we can quite freely vary speed with module count. A single Σ module, for example, could provide 8 different linear combinations of five variables.

Examples of large computations without branches are plentiful in the processor which we are building to reconstruct charged particle trajectories measured in a magnetic spectrometer. A measurement consists of one-dimensional projections of particle trajectories in 24 drift chamber planes inside a moderately non-uniform magnetic field. Each of the 24 measurements of a single trajectory is a separate nonlinear function of a single set of 5 parameters. A moderately accurate initial estimate of these parameters is five linear combinations of 6 measurements found in the initial pattern recognition.

To accurately determine the five parameters with a least-square fit of 24 measurements, we generate an initial parameter estimate, using Σ modules to provide the linear combinations of 6 measurements, then carry out nine multiplications to form 6 higher order products of these parameters. An eleven term polynomial expansion of the predicted track coordinates is then carried out for each of the 24 planes. Generation of the initial estimate of the 5 parameters and the 24 measurements implied, can be performed for a new track every 300 nanoseconds with about 70 modules, or more slowly with corresponding reduction in the number of modules.

READOUT MODULES

High speed data driven processing may well require data to be supplied in numerical form at very high rates. Detector systems developed at Nevis consist of several small subsystems, each capable of supplying encoded measurements at tens of Mhz. By buffering these subsystems in parallel, a single measurement consisting of several hundred numbers can be completely transferred in less than a microsecond. These subsystems attach to a control bus, so that each readout module can be addressed from the host for testing time and charge. Signals can be automatically injected into modules for calibration and testing.

The following readout modules have been built;

MWPC System-This is a coincidence register system that has been in use for many years, consisting of chamber mounted discriminators, flat polyethylene signal delay cables to 32 channel coincidence register cards. These cards attach to a read bus segmented by wire plane. A newly designed wire number encoder allows sparse readout at a 20 Mhz word rate onto our standard processor cable, where the word format is the binary encoded 10 bit wire number and crate name. The encoder also generates a word count for limiting the block size to a pre-set number of words, and a truncated event number in the complete word.

Drift Chamber System-This is a single hit time recording system designed for relatively close wire spaced drift chambers. The time is directly encoded into 6 bit gray code for time bins greater or equal to 4 nanoseconds, or 5 bit code for 2.5 nanosecond time bins. 32 signal TDC cards attach to a read bus segmented by wire plane. The sparse data can be transferred to a standard processor cable with a valid word every 25 nanoseconds. The data word contains a 10 bit wire number, a 6 bit time, and the plane number. A word counter in the readout allows the block size to be limited to a pre-set number, and a truncated event count is sent with each complete word.

ADC Readout[4]-This 8 channel ADC was developed by a member of the E 605 group for readout into our system. The ADC has 8 bits of square root encoding, and a digital cut for each channel to sparsify the data. The sparsified data can be transferred at a 20 Mhz rate to the processor cable.

Unencoded Register Data-This system consists of 16 bit fast coincidence registers. The data is read out in an unencoded, fixed block size form, at a 20 Mhz rate to the processor cable.

SUMMARY

Data driven machines have no natural scale association. The computation time does not have to increase as more computation is added, and the physical size of the system is not constrained since there is no centralized communication.

The cost per operation per second seems to be much lower than any other method. This is a result of the property of concurrency, the close match of the operators to the calculation, and to the simplicity of the operators.

REFERENCES

1. "A Hardware Architecture For Processing Detector Data In Real Time", G. Benenson, B. Knapp, W. Sippach, in Proceedings of 1978 Summer Workshop, Brookhaven National Laboratory.
2. "Real Time Processing of Detector Data", W. Sippach, G. Benenson, B. Knapp, IEEE Transactions of Nuclear Science, Vol. NS-27, No. 1, Feb. 1980
3. J. Rutherfoord, University of Washington
4. D. Kaplan, Fermi National Laboratory